Cryptographic decoding of the Leech lattice

Alex van Poppelen

Supervisors:
Dr. Léo Ducas
Dr. Gerard Tel

September 21, 2016

Contents

Chapter 1

Introduction

Lattice-based cryptography is an exciting field of research that promises to be the solution to the quantum crisis facing modern cryptography. It is no secret that quantum computing threatens the cryptographic building blocks in widespread use today, through quantum algorithms that are much faster than their classical counterparts. While the field of quantum computing has existed since the 1980s, the practical realization of quantum computation has remained firmly beyond technical reach.

However, small computers utilizing quantum effects have been built in recent years. Research carries on, and advances continue to be made. It is still unclear if or when large-scale quantum computers will become a reality. For many, this isn't a major concern. For others, who have bet large amounts on the difficulty of certain problems, it is a very pressing concern.

1.1 Post-quantum cryptography

The effects of quantum computing on the various cryptographic primitives in use today varies. For symmetric encryption algorithms and message authentication codes (MACs), the effects are moderate and well understood. Grover's algorithm [Gro96] is a probabilistic search algorithm that finds the unique input to a black box function using only $O(\sqrt{n})$ evaluations of the function. This result was proven to be asymptotically optimal in [BBBV97]. It is therefore believed that doubling the size of the secret key will be sufficient to protect against quantum attacks. Many popular primitives in use today like AES already support these larger key sizes.

For asymmetric primitives, the effects are more dire. It is well known that Shor's algorithm [Sho94] solves the integer factorization and discrete logarithm problems in polynomial time. For discrete logarithms over elliptic curve groups, the algorithm has also been shown to run in polynomial time [PZ03]. The primitives in widespread use today, like RSA and Diffie-Hellman, are based on these number-theoretic problems. They are therefore essentially broken when faced with a quantum adversary.

As asymmetric primitives are crucial building blocks in any cryptosystem, the need for

mathematical problems that are hard in both the classical and quantum sense arises. Cryptographers have found several interesting candidates, including some based on lattices. The advent of a real, large-scale quantum computer may seem far away to some. However, even if this is the case, there are several reasons why rapid deployment of post-quantum cryptography is crucial, and pressing.

Maturity of cryptanalysis. The cryptanalysis of post-quantum cryptographic schemes (including lattice-based schemes) is far less mature than that of those currently in use, based on the hardness of integer factorization and the discrete logarithm. Even if the theoretical foundations are strong, only time will tell whether weaknesses exist and can be exploited. Similarly, the implementation of such a scheme needs time to mature. There are many ways to break a system, and weak cryptography is only one of them. Implementations need time and testing to reveal potentional side channels and incorrect application of the theory.

Forward secrecy. The notion of forward secrecy is an important aspect in secure communication such as TLS. It is designed to protect against an adversary who records and stores encrypted communications, intending to compromise the long-term secret key or password at a later date. By using forward secrecy (realized through the use of ephemeral keys), past communications are protected against future compromises. However, it does not and cannot protect against cryptographic protocols that are broken on a future date. Therefore, deployment of post-quantum key exchange primitives today is crucial, in order for long-term security of documents and communications.

These concerns have prompted government and industry alike to take notice [Dir15, Ces15]. Many government agencies and standards bodies have announced their plans for a transition to post-quantum cryptography [Dir15, CJL+16, ABB+15].

1.2 Lattice-based cryptography

Lattice-based cryptography is one of the most promising candidates for post-quantum cryptography. Besides the conjectured security against quantum attacks, lattice-based schemes tend to be algorithmically simple and highly parallelizable. Additionally, hard lattice problems offer a much broader diversity of cryptographic primitives. Examples of this include fully homomorphic encryption [Gen09], which allows computations to be carried out on encrypted data. When decrypted, the plaintext matches the results as if they had been performed directly on the plaintext.

Lastly, schemes constructed from lattices typically enjoy a worst-case hardness guarantee [Ajt96, Reg09, BLP+13]. This means, informally, that they can be proven to be infeasible to break—provided that at least some instances of an underlying problem are intractable. This is a very strong security guarantee. The combination of all of these properties make lattice-based cryptography very attractive for adoption.

Of the many different theoretical primitives available, this thesis focuses on encryption and key exchange schemes based on the Learning With Errors (LWE) problem, and a more efficient variant called Ring Learning With Errors (R-LWE). The previously mentioned problem of forward secrecy has prompted the recent proposal and implementation of a number of key exchange schemes based on R-LWE [Pei14, ZZD+15, BCNS15, ADPS15] and LWE [DXL12, BCD+16].

The implementation of R-LWE schemes conservatively reaching acceptable and even demanding security requirements have been shown to achieve performance comparable to classical ECDHE software [BCNS15]. Although less efficient, the LWE scheme described in [BCD+16] is not far behind. In terms of bandwidth, however, the proposed schemes perform more poorly. The handshake size for R-LWE schemes is approximately 4× (and up) that of ECDHE, while the LWE scheme is more than 18× as large (all authenticated by ECDSA) [BCD+16].

While this is acceptable in most circumstances, especially considering the increased security guarantees afforded by these schemes, it is not ideal. When considering a protocol such as TLS, a large handshake size increases the latency observed when establishing a connection. This thesis looks at the possibility of reducing the size of the exchange, without compromising the security guarantees.

1.3 Applying coding theory

In an LWE key exchange or encryption scheme, the relevant bits are encoded and hidden in the ciphertext being transferred. This hiding is done by applying error to each element. When the ciphertext is reconciled or decrypted, there is still error masking the encoded bits. The decoding algorithm must decode correctly, despite this error, which is only possible if the error is small enough. Typically, this is the case with overwhelming probability, and controlled by the parameters of the problem. Note that the manner of encoding and decoding has no influence on the security of the scheme. The security of LWE is determined exclusively by the problem parameters.

In most schemes so far (e.g. [BCD+16]), the encoding and decoding is done on a per coordinate basis. For the recommended parameters, four bits are extracted from each coordinate in $\mathbb{Z}_q$ by rounding to specific intervals. This is equivalent to decoding the trivial $k\mathbb{Z}^k/4$ lattice, where $4k$ is the number of bits required by the key exchange. The integer lattice $\mathbb{Z}^n$ decreases in density as the dimension increases. Even for very small dimensions, there exist much denser lattices that maintain the same minimal distance between points. It is clear that this encoding is far from optimal, and it should be possible to pack more bits into a smaller number of coordinates, thereby decreasing the size of the ciphertext.

This thesis examines one such lattice, the 24-dimensional Leech lattice. We review the properties of the Leech lattice, and consider the bandwidth improvements it could offer over the current solution. By implementing a cryptographically secure decoder for the Leech lattice, we also investigate the feasibility of such an implementation, and the effects on efficiency in the context of an encryption scheme.

Chapter 2 recalls the necessary coding theory and lattice background. The third chapter describes the Leech lattice, its place among lattices, and the properties that make it interesting for our application. Chapter 4 discusses the construction of the Leech lattice and the resulting types of decoders available. Chapter 5 describes the LWE problem, a simple encryption scheme and its security, and compares the standard integer lattice encoding to the Leech lattice encoding. It also compares parameter sets for both types of encoding, under similar security guarantees. Chapter 6 documents the timing safe implementation of a Leech lattice decoder, compares it to an unsafe version, and gives benchmarks. Finally, Chapter 7 concludes with a discussion on the practicality and efficacy of the use of Leech lattice encoding for LWE key exchange and encryption schemes.

Chapter 2

Preliminaries

2.1 Notation

In this document, we use bold lower-case letters such as $\mathbf{x}$ to denote column vectors, and the transpose $\mathbf{x}^t$ for row vectors. The symbol $\mathbf{0}$ will refer to the zero vector in the context-defined dimension. Bold upper-case letters denote matrices. Typically, the letter will associate the matrix with its ordered set of column vectors, e.g. $\mathbf{A}$ and $\mathbf{a}_j$.

For each non-negative integer n, we let $\mathbb{R}^n$ denote the Euclidean vector space with the *canonical inner product* defined by

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=0}^{n} x_i y_i$$

where $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. For $\mathbf{z} \in \mathbb{R}^n$, we write the Euclidean norm $\|\mathbf{z}\| = \sqrt{\langle \mathbf{z}, \mathbf{z} \rangle}$.

For a distribution χ over a set S, $x \xleftarrow{\chi} S$ denotes sampling $x \in S$ according to χ. Similarly, vectors and matrices can be generated by sampling each coordinate individually, in an independent manner. This is denoted for vectors as $\mathbf{x} \xleftarrow{\chi} S^n$ where $\mathbf{x} \in S^n$, and for matrices as $\mathbf{X} \xleftarrow{\chi} S^{n \times m}$ where $\mathbf{X} \in S^{n \times m}$. The uniform distribution is denoted by $\mathcal{U}$.

2.2 Error-correcting codes

Error-correcting codes are closely related to lattices. Codes are discrete inside a discrete space, while lattices are discrete inside a continuous space—a natural generalization. Algorithmic advances in relation to codes often lead to similar advances in their lattice counterparts. A basic understanding of coding theory is summarized here for the reader.

2.2.1 Definitions

Error-correcting codes solve the problem of data transmission over noisy or unreliable communication channels. The sender will use redundancy to encode messages, allowing the

receiver to detect and/or correct a limited number of errors. We begin with the definition of a code, as given in [CS93].

Definition 2.1 (Q-ary code)**.** A *q-ary code* is a subset of $\mathbb{F}_q^n$, where $\mathbb{F}_q$ is the Galois field of order q and q is a prime or prime-power.

Typically, we will refer to a *binary code*, for which $q = 2$. A code is only useful if its codewords are easily distinguished from each other. In other words, they must be well separated by some metric. For two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}_q^n$, we define the *Hamming distance* to be the number of coordinates for which they differ. The *minimal distance* denotes the minimal Hamming distance between any two codewords in a code.

Definition 2.2 (Linear code)**.** A code $\mathcal{C}$ is *linear* if $\mathbf{0} \in \mathcal{C}$, and $-\mathbf{c}_1, \mathbf{c}_1 + \mathbf{c}_2 \in \mathcal{C}$ for every $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}$.

In other words, the linear code $\mathcal{C}$ is a linear subspace of $\mathbb{F}_q^n$. The dimension k of the linear code is equal to the dimension of this subspace. Trivially, it can be seen that there are q^k codewords in $\mathcal{C}$.

A linear code of length n, dimension k, and minimal distance d is referred to as a $[n, k, d]$ code.

For efficiency reasons, it is desirable to find error-correcting codes with maximal k, given n and d. This is called the *error-correcting code problem* for linear codes, and is in general unsolved.

2.2.2 Examples

The following are simple examples of codes that exist for every $n \geq 2$ and prime or prime-power q.

- The trivial *zero* code $[n, 0, n]$ of length n contains only the codeword $\mathbf{0}$.
- The *universe* code $[n, n, 1]$ is equal to $\mathbb{F}_q^n$.
- The *repetition* code $[n, 1, n]$ contains all codewords $\mathbf{c} = (a, \ldots, a)$ where $a \in \mathbb{F}_q$.
- The *zero-sum* code $[n, n-1, 2]$ contains codewords $\mathbf{c} = (c_1, \ldots, c_n)$ such that $\sum_i c_i = 0$. In the context of $q = 2$, this is more commonly referred to as the *even weight* or *even parity* code, since the last bit essentially acts as a parity bit.

2.3 Lattices

Some basic knowledge of lattices, associated properties, and computational problems is required before proceeding to specific algorithms and integration into cryptosystems. This theory is refreshed here.

2.3.1 Definitions

Before we can proceed to the definitions of a lattice and its properties, it is first necessary to formalize the concept of a *discrete subset*.

Definition 2.3 (Discrete subset)**.** A subset S of $\mathbb{R}^n$ is discrete if for each $x \in S$ there exists a positive real number ϵ such that the only vector $y \in S$ with $\|x - y\| < \epsilon$ is given by $y = x$.

We can now formally define a lattice.

Definition 2.4 (Lattice)**.** A lattice is an additive subgroup $\mathcal{L}$ of a finite dimensional Euclidean vector space $\mathbb{R}^n$ such that $\mathcal{L}$ is discrete as a subset of $\mathbb{R}^n$.

It can be shown that a subset $\mathcal{L}$ of a Euclidean vector space is a lattice if and only if there exists a basis of linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^n$ such that

$$\mathcal{L} = \mathcal{L}(\mathbf{B}) = \mathbf{B} \cdot \mathbb{Z}^n = \left\{ \sum_{i=1}^{n} z_i \mathbf{b}_i : z_i \in \mathbb{Z} \right\}. \tag{2.1}$$

The integer n is called the *rank* of $\mathcal{L}$. Note that the basis $\mathbf{B}$ is not unique. This can be shown by noting that for any unimodular matrix $\mathbf{U} \in \mathbb{Z}^{n \times n}$, $\mathbf{U} \cdot \mathbb{Z}^n = \mathbb{Z}^n$. Therefore $\mathbf{B} \cdot \mathbf{U}$ is also a basis of $\mathcal{L}(\mathbf{B})$, using (2.1).

The region defined by

$$\mathcal{P}(\mathbf{B}) = \theta_1 \mathbf{b}_1 + \cdots + \theta_n \mathbf{b}_n \quad (0 \leq \theta_i < 1)$$

is called a *fundamental parallelotope* for $\mathcal{L}$, or a building block that when tiled, fills the entire space with one lattice point per copy. Since the choice of basis is free, there are many different fundamental parallelotopes. However, the volume of this region is constant for $\mathcal{L}$. This volume is the called the *determinant* of $\mathcal{L}$, and is easily calculated as

$$\det \mathcal{L} = \sqrt{|\det \mathbf{B}^t \mathbf{B}|}.$$

Since lattices can be scaled arbitrarily by multiplying them by a constant, it is helpful to define some kind of standard representative of a lattice. We begin with a measure of how close together the points of a lattice are.

Definition 2.5 (Minimal distance)**.** A shortest nonzero lattice vector defines the *minimal distance* of a lattice $\mathcal{L}$:

$$\lambda_1(\mathcal{L}) := \min_{\mathbf{v} \in \mathcal{L} \setminus \{\mathbf{0}\}} \|\mathbf{v}\|.$$

The minimal distance is also often referred to as the *minimal norm*. We can use this measure to define a density on the lattice, combined with the volume of a fundamental parallelotope (recall that the fundamental parallelotope contains exactly one lattice point). A suitably scaled n dimensional lattice $\mathcal{L}$ of minimal distance 2 (e.g. the set of largest identical non-overlapping spheres centered on each lattice point have radius 1) has density $V_n / \det \mathcal{L}$, where V_n is the volume of an n-dimensional sphere of radius 1. This value is not normalized, meaning it is not preserved under scalar multiplication of $\mathcal{L}$. This leads us to the definition of *center density*.

Definition 2.6 (Center density)**.** The *center density* δ of a lattice $\mathcal{L}$ is given by

$$\delta = \left(\frac{\lambda_1}{2}\right)^2 \frac{1}{\det \mathcal{L}}.$$

If $\lambda_1 = 2$, the center density can be interpreted as the average number of lattice points per unit volume, since it is simply the inverse of the volume of a fundamental parallelotope. The center density makes it easy to compare densities of lattices. We will now define several more properties and associated lattices that will be useful in future sections.

Definition 2.7 (Dual lattice)**.** Given an n-dimensional lattice $\mathcal{L}$, the *dual* lattice $\mathcal{L}^*$ is defined as

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

An *integral* lattice is one for which the inner product of any two lattice vectors is integral. Trivially, it can be seen that a lattice $\mathcal{L}$ is integral if and only if $\mathcal{L} \subseteq \mathcal{L}^*$. When $\mathcal{L} = \mathcal{L}^*$ (equivalently $\det \mathcal{L} = 1$), it is termed *self-dual* or *unimodular*.

If one lattice $\mathcal{L}_1$ can be obtained from another $\mathcal{L}_2$ by a rotation, reflection, and/or scaling, we say that the lattices are *equivalent*, and write $\mathcal{L}_1 \cong \mathcal{L}_2$.

Along with fundamental parallelotopes, there is another important way to partition the space spanned by an n dimensional lattice $\mathcal{L}$. Consider a point $\mathbf{p} \in \mathcal{L}$. The set of all points of the underlying space that are closer to $\mathbf{p}$ than any other point in $\mathcal{L}$ is called the *Voronoi cell* of $\mathbf{p}$.

Definition 2.8 (Voronoi cell)**.** For any lattice $\mathcal{L} \subset \mathbb{R}^n$, the Voronoi cell V associated with a point $\mathbf{p} \in \mathcal{L}$ is defined to be the closed set

$$V(\mathbf{p}) = \left\{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{p}\| \leq \|\mathbf{x} - \mathbf{q}\| \ \forall \mathbf{q} \in \mathcal{L} \setminus \{\mathbf{p}\}\right\}.$$

Only finitely many points contribute to the shape of the Voronoi cell. These points are called *Voronoi relevant vectors*.

Lattices are often studied for their packing properties. The sphere packing problem tries to find a packing in n dimensions that achieves the highest possible density Δ, defined as the proportion of space taken up by spheres. If the spheres have a radius of 1 (e.g. $\lambda_1 = 2$), then

$$\delta = (\det \mathcal{L})^{-1}.$$

Aptly named, the center density can therefore be interpreted as the number of centers per unit volume, and is therefore ideally suited to comparing different sphere packings.

The sphere packing problem remains unsolved in general, and optimal solutions have been proven in only the first three dimensions. Optimal packings are not necessarily lattice

packings. Nonetheless, many lattices are conjectured (or proven, as for $n = 1, 2$ and most likely 3, see [Hal05]) to give optimal packings in their respective dimensions.

The best lattice packings often have surprising connections to other areas in mathematics, see [CS93] for a comprehensive summary.

2.3.2 Examples

The most trivial example of a lattice is the set of integers in one dimension, $\mathbb{Z}$. This can be extended to the n-dimensional unimodular *integer lattice* $\mathbb{Z}^n$. The center density is $\delta = 2^{-n}$, so the integer lattice becomes an increasingly less efficient sphere packing in higher dimensions. Besides $\mathbb{Z}^n$, there are several other important sequences of lattices, called the *root lattices*.

The root lattice A_n is defined for $n \geq 1$ as

$$A_n = \left\{(x_0, x_1, \ldots, x_n) \in \mathbb{Z}^{n+1} : x_0 + \cdots + x_n = 0\right\}.$$

Obviously, $A_1 \cong \mathbb{Z}$. A_2 is equivalent to the well-known unimodular hexagonal lattice, and A_3 is equivalent to the face-centered cubic lattice, the best packings possible in 2 and 3 dimensions respectively.

The root lattice D_n is known as the checkerboard lattice, and is defined for $n \geq 2$ as

$$D_n = \{(x_1, \ldots, x_n) \in \mathbb{Z}^n : x_1 + \cdots + x_n \equiv 0 \mod 2\}.$$

D_2 is simply a scaled rotation of $\mathbb{Z}^2$. $D_3 \cong A_3$ is the face-centered cubic lattice.

The *deep holes* of a lattice are defined as the points in space furthest away from any lattice point, and this distance is called the *covering radius*. For D_n, the covering radius increases with n. For $n = 8$, the covering radius happens to equal the minimal distance, meaning we can actually slide another copy of D_8 into space to coincide with the deep holes. This doubles the density of the lattice without reducing the minimal distance! This concept can be generalized for $n \geq 8$, and is defined as follows:

$$D_n^+ = D_n \cup \left(\frac{1}{2}, \ldots, \frac{1}{2}\right) + D_n.$$

However, the 8 dimensional case is especially important because of the equality between the covering radius and minimal distance. D_8^+ is known as E_8, another root lattice.

E_8 is a very remarkable even unimodular lattice in 8 dimensions, with unique density and minimal norm [CS93]. Its existance has been known since the 1800s, and cross-sections of E_8 give rise to the other root lattices E_7 and E_6.

Together, the root lattices $A_1, A_2, D_3, D_4, D_5, E_6, E_7$, and E_8 give the densest packings known in the first 8 dimensions. They are also proven to be optimal among lattice packings. All root lattices have the property that the minimal vectors are the only Voronoi relevant vectors [CS82c]. This is not necessarily the case for other lattices.

2.3.3 Lattice constructions

Glue theory. An n-dimensional integral lattice $\mathcal{L}$ can sometimes be described in terms of component integral lattices $\mathcal{L}_1, \ldots, \mathcal{L}_k$ of total dimension n, that have been "glued" together. Consider such a lattice $\mathcal{L}$ containing the sublattice $\mathcal{L}'$ which is the direct sum

$$\mathcal{L}' = \mathcal{L}_1 \oplus \cdots \oplus \mathcal{L}_k.$$

where the operator $\oplus$ denotes the direct sum.

Any vector $\mathbf{y} \in \mathcal{L}$ can be written as $\bigoplus_i \mathbf{y}_i$ where each $\mathbf{y}_i$ is in the subspace spanned by $\mathcal{L}_i$. Note that $\mathbf{y}_i$ may or may not be in $\mathcal{L}_i$. We call $\mathbf{y}_i$ a *glue vector* of $\mathcal{L}_i$. Since the inner product of $\mathbf{y}$ with any vector in $\mathcal{L}_i$ is integral (because $\mathcal{L}$ is integral), it follows that the inner product of $\mathbf{y}_i$ with that vector is integral. Thus $\mathbf{y}_i$ is in the dual lattice $\mathcal{L}_i^*$. Since adding any vector in $\mathcal{L}_i$ to $\mathbf{y}_i$ results in a vector in $\mathcal{L}$, it is helpful to choose a standard representative glue vector $\mathbf{g}_i$ for each coset. Therefore $\mathcal{L}_i^*/\mathcal{L}_i$ is referred to as the *glue group* for $\mathcal{L}_i$, with order equal to $(\det \mathcal{L}_i)^2$ [CS93].

The lattice $\mathcal{L}$ can be obtained from $\mathcal{L}'$ and a subset of vectors from the group

$$G = \left\{ \mathbf{g} : \mathbf{g} = \sum_i \mathbf{g}_i, \mathbf{g}_i \in \mathcal{L}_i^*/\mathcal{L}_i \right\}$$

with the additional restrictions that each $\mathbf{g} \in G$ has an integral inner product with every other vector in G, and that G is closed under addition modulo $\mathcal{L}'$.

Glue theory is an incredibly powerful tool for lattices with a high degree of structure. Being able to formulate lattices using lower dimensional component lattices forms the basis for efficient decoding algorithms and greatly contributes to the understanding of large structured lattices.

Constructions from codes. Lattices are closely related to error-correcting codes. An efficient error-correcting code can be used to construct a dense sphere packing. The following constructions have been shown to give valid sphere packings [LS71, Slo77, CS93]. They have been specified here over linear codes, to guarantee the construction of a lattice as opposed to a general sphere packing.

Definition 2.9 (Construction A)**.** Let $\mathcal{C}$ be a $[n, k, d]$ linear binary code. A vector $\mathbf{y} = (y_1, \ldots, y_n)$ is a lattice vector if and only if $\mathbf{y}$ is congruent (modulo 2) to a codeword of $\mathcal{C}$.

Definition 2.10 (Construction B)**.** Let $\mathcal{C}$ be a $[n, k, d]$ linear binary code with the property that the weight of each codeword is even. A vector $\mathbf{y} = (y_1, \ldots, y_n)$ is a lattice vector if and only if $\mathbf{y}$ is congruent (modulo 2) to a codeword of $\mathcal{C}$, and $\sum_i y_i \equiv 0 \mod 4$.

The center density of the resulting packings can be shown to be bounded by some function of n (see [CS93]). Conway and Sloane remark that much denser packings have been found for large n, meaning these constructions are not useful for finding dense sphere packings in higher dimensions, regardless of the codes used.

2.3.4 Computational problems

Of the large number of computational problems on lattices, there are several of particular relevance to this thesis. We will define them now.

Definition 2.11 (Shortest Vector Problem (SVP))**.** Given an arbitrary basis $\mathbf{B}$ and norm $\|\cdot\|$ for an n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$, find a shortest non-zero vector $\mathbf{v} \in \mathcal{L}$, as defined by the norm. In other words, $\|\mathbf{v}\| = \lambda_1(\mathcal{L})$.

The Shortest Vector Problem has been studied intensively, and appears to be intractable in general, even including quantum algorithms. A lot of attention has been given to the approximation version of SVP, which is particularly applicable to cryptography. In the γ-approximation Shortest Vector Problem (SVP_γ, this means finding a lattice vector at distance of at most $\gamma\lambda_1(\mathcal{L})$. Variants of this approximate problem are typically used to prove the security of cryptosystems.

Next we define two problems more typically used in coding theory.

Definition 2.12 (Closest Vector Problem (CVP))**.** Given an arbitrary basis $\mathbf{B}$, norm $\|\cdot\|$ for an n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$ and some target point $\mathbf{t} \in \mathbb{R}^n$, find a lattice vector $\mathbf{v}$ such that $\mathbf{v} := \min_{\mathbf{y}\in\mathcal{L}} \|\mathbf{y} - \mathbf{t}\|$.

In the context of coding, the Closest Vector Problem is often referred to as *maximum likelihood decoding*. If we consider the points in a lattice $\mathcal{L}$ as codewords, an algorithm that solves CVP for $\mathcal{L}$ can be used as a decoder. This algorithm guarantees to decode every point in the Voronoi cell (excluding the border) of a codeword to the codeword itself. Points on the border between one or more Voronoi cells may be decoded to any one of those codewords.

Definition 2.13 (Bounded Distance Decoding Problem (BDD))**.** Given an arbitrary basis $\mathbf{B}$ for an n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$ and a target point $\mathbf{t} \in \mathbb{R}^n$ with the guarantee that $\exists \mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{v} - \mathbf{t}\| < d = \lambda_1(\mathcal{L})/2$, find $\mathbf{v}$.

Note that $\mathbf{v}$ is, by definition, unique. An algorithm that solves BDD on a lattice $\mathcal{L}$ can also be used as a decoder. However, it is only guaranteed to send points within an n-dimensional sphere of radius $\lambda_1/2$ (excluding the border) of a codeword to that codeword. Depending on the center density of $\mathcal{L}$, this may leave a lot of space for which correct decoding cannot be guaranteed.

Obviously, CVP is at least as hard as BDD. A reduction from BDD to CVP is trivial, by passing valid $\mathbf{t}$ and $\mathbf{B}$ for BDD to an algorithm for CVP. This algorithm will necessarily decode to the unique lattice vector within distance d of $\mathbf{t}$, since it is uniquely closest to $\mathbf{t}$.

Like SVP, the above two problems are hard in general. However, for some select small, well known lattices with a high degree of structure, fast algorithms exist. While for coding purposes CVP algorithms are most desirable, their BDD counterparts may be significantly more efficient (e.g. compare [Var95] to [VB93]). In some settings, a small loss in coding gain is acceptable in order to drastically increase efficiency.

2.4 Lattice decoding

Remark. We have given two different definitions of decoding, and have shown the stronger properties of CVP decoding. From now on, "decoding" will therefore refer to CVP (or maximum likelihood decoding), unless otherwise specified. The following strategies are specified on the CVP problem, but can be applied to BDD problems as well (see subsection 2.4.4).

The more trivial lattices are very easy to decode. For example, let us take the integer lattice $\mathbb{Z}^n$. Decoding a point $\mathbf{t} \in \mathbb{R}^n$ to $\mathbb{Z}^n$ is simply a matter of rounding each coordinate individually, meaning any point $\mathbf{t}$ can be decoded in linear time. Efficient decoding algorithms exist for all the root lattices and their duals [CS82a, MCSQ08].

Larger lattices are more difficult, and no general, efficient algorithm is known. However, the various constructions given in section 2.3.3 can provide the basis for a general decoding strategy. In general, one tries to find a sublattice (often a direct sum of root lattices or their duals) of smallest index for which a fast algorithm exists [CS86]. Sometimes, the specific structure of the glue group can also be exploited, as will be demonstrated in this section.

In the remainder of this section, it is assumed we are decoding a point $\mathbf{t} \in \mathbb{R}^n$. Before describing the strategies, it is helpful to first define the closest vector more generally, on discrete sets instead of only lattices.

Definition 2.14 (Closest vector)**.** For any discrete set $S \subset \mathbb{R}^n$ we denote $\mathsf{CV}_{\mathbf{t}}(S)$ a closest point $\mathbf{p} \in S$ from the target $\mathbf{t} \in \mathbb{R}^n$. In the case of equality, $\mathsf{CV}_{\mathbf{t}}(S)$ is chosen arbitrarily from the closest points.

Since every lattice is a discrete set, this definition can be used for both sublattices and their cosets. The definition of the closest vector admits the following identity on discrete sets, called the *union identity*.

Lemma 2.1 (Union identity)**.** *For any discrete set $S \in \mathbb{R}^n$, we have that*

$$\mathsf{CV}_{\mathbf{t}}(S) = \mathsf{CV}_{\mathbf{t}}(\{\mathsf{CV}_{\mathbf{t}}(S_i) : i = 1, \ldots, k\}) \quad \textit{where } S = \bigcup_{i=1}^{k} S_i.$$

The proof follows easily by noting that a closest vector in S must lie in some subset S_i, and is by definition as close or closer than the closest vectors from all the other subsets. We are now in a position to describe some general decoding strategies for lattices.

2.4.1 Decoding direct sums

Direct sums are trivially decoded when efficient algorithms exist for each of the component lattices. To decode a direct sum decomposition $\mathcal{L} = \mathcal{L}_1 \oplus \cdots \oplus \mathcal{L}_k$ one simply decodes each orthogonal projection of $\mathbf{t}$ onto the space spanned by each component lattice, and sums the results. Formally,

$$\mathsf{CV}_{\mathbf{t}}(\mathcal{L}) = \bigoplus_{i=1}^{k} \mathsf{CV}_{\pi_i(\mathbf{t})}(\mathcal{L}_i) \tag{2.2}$$

where π_i denotes the orthogonal projection onto the space spanned by $\mathcal{L}_i$. This algorithm admits the following bound on the cost of decoding.

Lemma 2.2 (Direct sum)**.** *For a direct sum decomposition $\mathcal{L} = \mathcal{L}_1 \oplus \cdots \oplus \mathcal{L}_k$, the cost $C(\mathcal{L})$ of calculating $\mathsf{CV}_{\mathbf{t}}(\mathcal{L})$ is bounded by $\sum_i C(\mathcal{L}_i) + c_i$, where c_i is the cost of the projection π_i.*

In general, we choose the basis of $\mathcal{L}$ such that the spaces spanned by the component lattices split along the coordinates. In this case, the cost of a projection is essentially free.

2.4.2 Decoding unions of cosets

An efficient decoding algorithm for a lattice $\mathcal{L}'$ can easily be applied to a coset $\mathbf{g} + \mathcal{L}'$. If $\mathsf{CV}_{\mathbf{t}}(\mathcal{L}')$ is the closest point of $\mathcal{L}'$ to $\mathbf{t}$, then

$$\mathsf{CV}_{\mathbf{t}}(\mathbf{g} + \mathcal{L}') = \mathbf{g} + \mathsf{CV}_{\mathbf{t}-\mathbf{g}}(\mathcal{L}'). \tag{2.3}$$

For a lattice $\mathcal{L}$ with glue group $G = \mathcal{L}/\mathcal{L}'$, decoding $\mathbf{t}$ is therefore a matter of computing the closest vector in each coset using a decoder for $\mathcal{L}'$, and applying the union identity. Formally,

$$\mathsf{CV}_{\mathbf{t}}(\mathcal{L}) = \mathsf{CV}_{\mathbf{t}}(\{\mathbf{g} + \mathsf{CV}_{\mathbf{t}-\mathbf{g}}(\mathcal{L}') : \mathbf{g} \in G\}). \tag{2.4}$$

Lemma 2.3 (Sublattice)**.** *For a lattice $\mathcal{L}$ with sublattice $\mathcal{L}'$ of index $|G|$, the cost $C(\mathcal{L})$ of calculating $\mathsf{CV}_{\mathbf{t}}(\mathcal{L})$ is bounded by $|G|(1 + C(\mathcal{L}'))$.*

This strategy is frequently combined with the previous approach, where the sublattice $\mathcal{L}'$ in question is a direct sum of component lattices. We get

$$\mathsf{CV}_{\mathbf{t}}(\mathcal{L}) = \mathsf{CV}_{\mathbf{t}}(\{\mathbf{g} + \bigoplus_{i=1}^{k} \mathsf{CV}_{\pi_i(\mathbf{t}-\mathbf{g})}(\mathcal{L}_i) : \mathbf{g} \in G\}). \tag{2.5}$$

There is often redundancy when decoding a component lattice $\mathcal{L}_i$ for different glue vectors $\mathbf{g}_a$ and $\mathbf{g}_b$. If $\pi_i(\mathbf{g}_a) = \pi_i(\mathbf{g}_b)$, the decoding needs only be done once. In this way, the complexity of the overall decoder can be greatly reduced.

Lemma 2.4 (Direct sum sublattice)**.** *For a lattice $\mathcal{L}$, with a sublattice $\mathcal{L}' = \mathcal{L}_1 \oplus \cdots \oplus \mathcal{L}_k$, the cost $C(\mathcal{L})$ of calculating $\mathsf{CV}_{\mathbf{t}}(\mathcal{L})$ is bounded by $\sum_i |G| + |G_i| C(\mathcal{L}_i) + c_i$, where G_i is defined as $\pi_i(G)$.*

The term $|G|$ is inside the sum due to the need to perform $i - 1$ additions to construct the distance metric for every glue vector in G, combined with $|G| - 1$ comparisons in order to minimize over the glue group. With additional structure in G, it is sometimes possible to optimize this further.

2.4.3 Decoding glue with parity

We now describe a technique to decode lattices consisting of a sublattice with glue groups of a specific parity structure. While this technique is not new, and is already used in existing lattice decoders (e.g. [VB93]), to our knowledge this is the first time it has been described in the context of glue theory.

We write the lattice $\mathcal{L}$ as

$$\mathcal{L} = G + (\mathcal{L}_1 \oplus \cdots \oplus \mathcal{L}_k)$$

with projected glue groups $G_1, \ldots, G_k$. The component lattices $\mathcal{L}_i$ may be decoded efficiently. We assume that we have a parity group P, and that each G_i admits an injection in $P : G_i \simeq P_i \subset P$. We will use the isomorphism $\mu_i : G_i \mapsto P_i$.

We assume the group G is the direct sum of the component glue groups, along with a parity condition. Formally,

$$G = \Big\{(\mathbf{g}_1, \ldots, \mathbf{g}_k) \in G_1 \oplus \cdots \oplus G_k \text{ such that } \sum_i \mu_i(\mathbf{g}_i) = 0\Big\}.$$

This parity condition can be exploited, drastically reducing the amount of decoding to be done.

First, $\pi_i(\mathbf{t})$ is decoded in $\mathbf{g} + \mathcal{L}_i$ for each i and $\mathbf{g} \in G_i$. This can be done easily using the strategy in Section 2.4.2. Let $\hat{\mathbf{g}}_i$ be the closest vector to $\pi_i(\mathbf{t})$ in $G_i + \mathcal{L}_i$. By direct summing all $\hat{\mathbf{g}}_i$, one obtains the closest vector $\hat{\mathbf{g}}$ in the superlattice $\hat{\mathcal{L}} = (G_1 \oplus \cdots \oplus G_k) + (\mathcal{L}_1 \oplus \cdots \oplus \mathcal{L}_k)$.

Lemma 2.5 (Superlattice)**.** *For the lattice $\mathcal{L}$, with a sublattice $\mathcal{L}' = \mathcal{L}_1 \oplus \cdots \oplus \mathcal{L}_k$ and glue group $G = G_1 \oplus \cdots \oplus G_k$, the cost $C(\mathcal{L})$ of calculating $\mathsf{CV}_{\mathbf{t}}(\mathcal{L})$ is bounded by $\sum_i |G_i| C(\mathcal{L}_i) + c_i$.*

We call the parity of the closest vector in $\hat{\mathcal{L}}$ the *syndrome* $s \in P$. If s is equal to zero, then the closest vector in $\hat{\mathcal{L}}$ satisfies the parity condition. It is therefore also in $\mathcal{L}$, and is obviously the closest vector to $\mathbf{t}$. If $s \neq 0$, one has to explore the variation around the solution and force it into $\mathcal{L}$. Since every $\mathbf{g} + \mathcal{L}_i$ for $\mathbf{g} \in G_i$ has already been decoded, most of the work has been done. What remains is finding the best combination of alternative cosets for some set of $\mathcal{L}_i$, effecting a change of s to the parity, that minimizes the increase in distance to $\mathbf{t}$.

The number of combinations of alternative cosets effecting a change equal to the syndrome s may be very large. However, it is generally possible to restrict oneself to a set of minimal combinations that are guaranteed to be better than any combinations outside this set. The strategies to do this vary with the nature of P, and are beyond the scope of this thesis. We will instead assume some number n of combinations of cosets leading to a change of s, which is always smaller than $|P|^k$.

Having such a combination of alternative cosets, it remains to find the optimal set of coordinates i to apply these alternative cosets. First, a penalty is calculated for each $\mathbf{g}_i \in G_i \setminus \{\hat{\mathbf{g}}_i\}$, where the penalty is the increase in distance to $\pi_i(\mathbf{t})$, compared to the distance of $\hat{\mathbf{g}}_i$ to $\pi_i(\mathbf{t})$. The set of all penalties can be partitioned according to the change $q \in P$ they effect on the parity, as compared to $\hat{\mathbf{g}}$.

Finding the optimal coordinates for these penalties is equivalent to the assignment problem, or alternatively finding a minimum weight perfect matching in a weighted complete bipartite graph (the graph is made complete by using dummy vertices). This problem is well known, and can be solved in polynomial time using the Hungarian algorithm [Mun57]. However, for small orders of P, it is generally faster to sort each list of penalties corresponding to a particular change on the parity, and enumerate the combinations of best available penalties in unique coordinates. It is this sorting and enumeration method that is used in the decoder described in Section 3.3.

Having minimized the cumulative penalty in distance to $\mathbf{t}$, while changing the parity to that of a vector in $\mathcal{L}$, we have forced the closest vector in $\hat{\mathcal{L}}$ into the lattice $\mathcal{L}$. This is necessarily the closest vector to $\mathbf{t}$ in $\mathcal{L}$.

Lemma 2.6 (Glue with parity)**.** *Consider the lattice $\mathcal{L}$, with a sublattice $\mathcal{L}' = \mathcal{L}_1 \oplus \cdots \oplus \mathcal{L}_k$ and glue group $G = \{(\mathbf{g}_1, \ldots, \mathbf{g}_k) \in G_1 \oplus \cdots \oplus G_k$ such that $\sum \mu_i(\mathbf{g}_i) = p\}$ where each G_i admits an injection in a parity group P, such that $\mu_i : G_i \mapsto P_i$, and $p \in P$. Let n be the maximal number of minimal constructions of an element p. Then the cost $C(\mathcal{L})$ of calculating $\mathsf{CV}_{\mathbf{t}}(\mathcal{L})$ is bounded by*

$$C(\mathcal{L}) \leq nk^4 C_{hun} + \sum_i |G_i| C(\mathcal{L}_i) + c_i$$

where C_{hun} is defined as the overhead of the Hungarian algorithm.

The first term comes from the number of ways to combine cosets to effect a change in parity of s, and the use of the Hungarian algorithm (running in $O(k^3)$ to solve the optimal assignment of penalties to coordinates. The extra k comes from the need to sum the penalties before being able compare different combinations. The last term corresponds to lemma 2.5.

This technique is heavily dependent on the order of P. For small $|P|$, it can be very efficient, especially if it eliminates the need for the Hungarian algorithm. However, increasing the order of P drastically increases the number of ways to construct s, thereby greatly increasing the number of penalty combinations that need to be compared.

2.4.4 Applying decoding strategies to BDD

The BDD problem is essentially the same as the CVP problem, with an extra guarantee on the solution. We will now examine whether BDD algorithms can be broken down in the same manner as CVP algorithms in the previous section.

Let $\mathsf{BDD}_{\mathbf{t}}(\mathcal{L})$ be the vector in $\mathcal{L}$ returned by a BDD algorithm given $\mathbf{t}$. For $\mathbf{t}$ sufficiently close to $\mathcal{L}$, we have that $\mathsf{BDD}_{\mathbf{t}}(\mathcal{L}) = \mathsf{CV}_{\mathbf{t}}(\mathcal{L})$. Otherwise, we assume that the algorithm returns an arbitrary vector in $\mathcal{L}$.

Lemma 2.7 (Bounded distance direct sum)**.** *Let $\mathcal{L} = \mathcal{L}_1 \oplus \cdots \oplus \mathcal{L}_k$. Given BDD decoders for all $\mathcal{L}_i$, it holds that*

$$\mathsf{BDD}_{\mathbf{t}}(\mathcal{L}) = \bigoplus_{i=1}^{k} \mathsf{BDD}_{\pi_i(\mathbf{t})}(\mathcal{L}_i).$$

Proof. The direct sum $\mathcal{L} = \mathcal{L}_1 \oplus \cdots \oplus \mathcal{L}_k$ necessarily admits the following relation:

$$\lambda_1(\mathcal{L}) = \min\{\lambda_1(\mathcal{L}_i)\}.$$

Therefore, given a target $\mathbf{t}$ for which $\exists \mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{v} - \mathbf{t}\| < \lambda_1(\mathcal{L})/2$, we have that

$$\|\pi_i(\mathbf{v}) - \pi_i(\mathbf{t})\| < \lambda_1(\mathcal{L})/2 \leq \lambda_1(\mathcal{L}_i)/2.$$

A BDD algorithm for $\mathcal{L}_i$ and target $\pi_i(\mathbf{t})$ will necessarily yield $\pi_i(\mathbf{v})$, indicating that the algorithm is valid for BDD decoders. □

Lemma 2.8 (Bounded distance sublattice)**.** *Let $\mathcal{L}$ have glue group $G = \mathcal{L}/\mathcal{L}'$. Given a BDD decoder for $\mathcal{L}'$, it holds that*

$$\mathsf{BDD}_{\mathbf{t}}(\mathcal{L}) = \mathsf{BDD}_{\mathbf{t}}(\{\mathbf{g} + \mathsf{BDD}_{\mathbf{t}-\mathbf{g}}(\mathcal{L}') : \mathbf{g} \in G\}).$$

Proof. We can show a similar bound on the minimal vector of $\mathcal{L}$. Trivially, $\lambda_1(\mathcal{L}) \leq \lambda_1(\mathcal{L}')$. We are guaranteed that there is a unique $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{v} - \mathbf{t}\| < \lambda_1(\mathcal{L})/2$. The bounded distance vector $\mathbf{v}$ is in one of the cosets of $\mathcal{L}'$. Call this coset $\mathbf{g} + \mathcal{L}'$.

A BDD algorithm for $\mathcal{L}'$ applied to this coset using 2.3 will output $\mathbf{v}$, since

$$\|\mathbf{v} - \mathbf{t}\| < \lambda_1(\mathcal{L})/2 \leq \lambda_1(\mathcal{L}'),$$

meaning $\mathsf{BDD}_{\mathbf{t}}(\mathbf{g} + \mathcal{L}') = \mathsf{CV}_{\mathbf{t}}(\mathbf{g} + \mathcal{L}')$. Since $\mathbf{v}$ is closer to $\mathbf{t}$ than any other vector in $\mathcal{L}$, the union identity will preserve this solution. □

The two algorithms aboved can be combined to produce a similar algorithm for direct sum sublattices, as in 2.5. They are also the building blocks for a similar algorithm using the decoding strategy for glue with parity, described in the previous section. The exact details will not be discussed here.

2.5 Public key cryptography

As described briefly in the introduction, cryptography is concerned with many different kinds of primitives, and the kinds of security they can offer through their properties. There are a few definitions that will be relevant to this thesis, and they are described here.

Public key encryption is also known as asymmetric encryption. It consists of the following three algorithms that depend on some overall parameters set by the scheme.

Key generator. The key generator takes a security parameter (defined later in this section), and produces a valid public key $p \in P$ and a valid private key $s \in S$, where (p, s) is called a keypair.

$$\text{KeyGen} : \mathbb{N} \to P \times S.$$

The objects p and s have a mathematical relation specific to the scheme that allows the following two algorithms to work correctly.

Encryption. The encryption algorithm uses the public key to hide a message m from some set of valid messages M in an outputted ciphertext, $c \in C$.

$$\text{Enc} : P \times M \to C.$$

Decryption. The decryption algorithm uses the private key to recover the message from the ciphertext. If the ciphertext is invalid, it may also return some kind of error, denoted $\perp$.

$$\text{Dec} : S \times C \to M \cup \{\perp\}.$$

The scheme is considered to be correct if the decryption algorithm yields the same valid message that was input to the encryption algorithm, for every private and public keypair output of the key generator. Sometimes, a very small probability of incorrect decryption is permissible.

The notion of security is more difficult to define. We regulate the running times of the algorithms above, and those of an attacker, sub-exponentially by a *security parameter* λ. The typically accepted definition of security is called semantic security in [GM84], also known as *indistinguishability under chosen plaintext attack* (IND-CPA).

Definition 2.15 (IND-CPA security)**.** The concept of IND-CPA security is defined by the following experiment, using a uniformly random secret bit $b \in \{0, 1\}$.

1. The challenger generates a keypair (p, s) based on a security parameter λ, and gives p to the attacker.
2. The attacker responds with two valid messages, m_0 and m_1, in $2^{o(\lambda)}$ time.
3. The challenger encrypts m_b using p, and gives the resulting challenge ciphertext c to the attacker.
4. The attacker outputs a guess $g \in \{0, 1\}$ for the value of b in $2^{o(\lambda)}$ time.

The encryption scheme is considered IND-CPA secure if an attacker guesses correctly with probability

$$\Pr[g = b] = \frac{1}{2} + \epsilon(\lambda)$$

where $|\epsilon(\lambda)| \leq 2^{-o(\lambda)}$.

Informally, this means that at best, the attacker has only a negligible advantage over random guessing.

It is important to note that IND-CPA security only models passive adversaries. Active adversaries are modeled using IND-CCA security, or *indistinguishability under chosen ciphertext attack*, which is beyond the scope of this thesis. There are efficient and generic techniques to upgrade IND-CPA schemes to IND-CCA security. Therefore, we are only concerned with IND-CPA security and passive attackers.

Chapter 3

The Leech lattice

The 24-dimensional Leech lattice Λ_{24} is one of the most important and remarkable lattices known. It was discovered in 1965 by Leech [Lee67], who was looking for solutions to the sphere packing problem in higher dimensions. It turned out to be an unexpectedly good and symmetrical packing. Λ_{24} has so many exceptional properties and intimate relationships to other branches of mathematics, that an entire book is essentially dedicated to this lattice (see [CS93]).

3.1 Construction

There are a huge amount of constructions for Λ_{24} available in literature, where it is constructed from specific vector formulas, the different Golay codes, cyclotomic fields, simpler lattices such as D_{24}, A_1^{24}, and other sublattices (see the Niemeier constructions), as a Lorentzian lattice, and many more (all of these constructions and more are given in [CS93]). However, one of the most remarkable properties of the Leech lattice is that it is the unique laminated lattice in 24 dimensions. Regarding the Leech lattice as a laminated lattice is particularly interesting, as it is a "no-input" kind of construction.

We will begin with the definition of a laminated lattice by induction.

Definition 3.1 (Laminated lattice)**.** Let Λ_0 be the single point lattice in 0 dimensions. Take all n-dimensional lattices for $n \geq 1$ with minimal norm 4 containing at least one sublattice Λ_{n-1}, and select those with minimal determinant. Any such lattice Λ_n is a *laminated lattice*.

This inductive construction is a very natural way to construct a lattice packing. We start with the trivially maximal lattice packing in one dimension, $\Lambda_1 = 2\mathbb{Z}$. Drawing a row of circles with a radius of 1 centered around these points, we can obtain the hexagonal lattice $\Lambda_2 = A_2$ by packing rows as close to each other as possible. Layering the hexagonal lattice in 3 dimensions gives us the face-centered cubic lattice $\Lambda_3 = D_3$. Repeating this process, we eventually arrive at the unique Λ_{24} known as the Leech lattice. That Λ_{24} is the unique laminated lattice in its dimension is remarkable, as this is not generally the case.

3.2 Properties

The laminated lattices for $n \leq 24$ can be thought of as the main sequence of cross-sections of the Leech lattice. Interestingly, Leech showed in [Lee64, Lee67, LS71] that there is another important sequence of cross-sections, denoted $K_0, \ldots, K_{24}$. This sequence is identical to the laminated sequence for $n \leq 6$ and $n \geq 18$, but differs in the middle.

3.2.1 Sphere packing

A natural question raised by this inductive construction is the density of these laminated lattices. We already know that the laminated lattice packings in the first two and possibly three dimensions are proven to give optimal packings. But how do higher dimensional Λ_n fare? It turns out that for $n \leq 10$ and $14 \leq n \leq 29$, the laminated lattices do indeed give the densest lattice packings known [CS93]. Notably, K_{11}, K_{12} (the Coxeter-Todd lattice), and K_{13} have higher densities than their laminated counterparts, and are indeed the densest known lattices for their respective dimensions [Lee67, LS71]. Thus Λ_{24} contains the densest lattices in all lower dimensions.

For completeness, it's interesting to note that while some densest lattice packings known are also the densest packings known, for some dimensions denser non-lattice packings exist. For $n = 10, 11, 13, 18, 20, 22$ there exist denser non-lattice packings than the densest laminated and K lattices in those dimensions.

While these sequences are interesting in their own right, they fail to highlight the extraordinary density of the Leech lattice. The center density of Λ_{24} is 1, the highest of any packing of dimension less than 30 (excepting in zero dimensions, trivially 1). Compare this to 0.5 for A_1 and Λ_{23}, 0.29 for A_2 and Λ_{22}, and much lower for every lattice in between (0.06 for E_8). Interestingly, $\delta_n = \delta_{n-24}$ for the laminated (and K-lattices) where $n \leq 24$.

The remarkable density of the Leech lattice makes it very interesting for coding theory. Before getting into the bulk of this document, we take the time to become more familiar with the general properties, shape and symmetries of the Leech lattice.

3.2.2 Relationship to the Golay Code

The Golay code $\mathcal{C}_{23}$ was discovered by M.J.E. Golay in 1949 [Gol49]. It is a $[23, 12, 7]$ binary code that is the quadratic residue code of length 23. In other words, it is a cyclic code over $\mathbb{F}_2$ whose generator polynomial has roots $\{\alpha^i : i \neq 0 \text{ is a square modulo } n\}$.

The much more commonly used extended Golay code $\mathcal{C}_{24}$ was obtained by appending a parity bit to $\mathcal{C}_{23}$. It is the unique $[24, 12, 8]$ binary code, and therefore has 4096 codewords with weight distribution $0^1 8^{729} 12^{2576} 16^{759} 24^1$.

The Leech lattice is intimately connected to $\mathcal{C}_{24}$, as illustrated by the following construction of Λ_{24} taken from [CS93]. It should be noted that this is only one of very many constructions using $\mathcal{C}_{24}$.

Definition 3.2 (Leech lattice)**.** The Leech lattice Λ_{24} consists of all the vectors

$$\frac{1}{\sqrt{8}}(\mathbf{0} + 2\mathbf{c} + 4\mathbf{x}), \tag{3.1}$$

$$\frac{1}{\sqrt{8}}(\mathbf{1} + 2\mathbf{c} + 4\mathbf{y}), \tag{3.2}$$

where $\mathbf{c} \in \mathcal{C}_{24}$ and the elements of $\mathbb{F}_2$ are regarded as real 0's and 1's, and $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^{24}$ such that $\sum_i x_i \equiv 0 \mod 2, \sum_i y_i \equiv 1 \mod 2$.

In essence, the Leech lattice is constructed by lifting $\mathcal{C}_{24}$ to $\mathbb{Z}^{24}$, restricting the sum of the coordinates to zero modulo 4, and sliding another copy of this lattice (called the Leech half-lattice) into the resulting deep holes. The vectors given by (3.1) are called the even vectors of Λ_{24}, while (3.2) are called the odd vectors of Λ_{24}.

3.2.3 Shape

Using the above construction, we can get a feel for the shape of the Leech lattice by examining the minimal vectors, which have length 2. For convenience, we will say a vector $\mathbf{v} = (v_1, \ldots, v_n)$ has shape $(a^j, \ldots)$ if $v_i = a$ for j coordinates, etc.

Minimal length vectors of Λ_{24} have three different shapes, and can be easily generated using the equations above. For the first, we can use (3.1), and combine the 729 $\mathcal{C}_{24}$ codewords of weight 8 with $\mathbf{x}$ to invert the signs of the coordinates containing a 2. This gives $2^7 \cdot 729 = 97152$ vectors of shape $(0^{16}, \pm 2^8)$, since there must be an even number of nonzero $\mathbf{x}$ coordinates. Second, we use (3.2), all 2^{12} codewords of $\mathcal{C}_{24}$, and an odd number of coordinates with ± 1 in $\mathbf{y}$. This gives $2^{12} \cdot 24 = 98304$ vectors of shape $(\pm 1^{23}, \pm 3)$. Lastly, using (3.1) and no codewords at all, we have $2^2 \cdot \binom{24}{2} = 1104$ minimal vectors of shape $(0^{22}, \pm 4^2)$.

In total, we have 196560 minimal vectors of length 2. This number is also by definition the *kissing number* of the Leech lattice, or the maximal number of nonoverlapping unit spheres that can simultaneously touch exactly one point of the boundary of the unit sphere centered at the origin. This result was proven optimal in 24 dimensions by E. Bannai and N. J. A. Sloane in 1981 [BS81].

The minimal vectors alone are not sufficient to define the Voronoi regions of Λ_{24}. However, the minimal vectors combined with the second layer of norm $\sqrt{6}$ are sufficient to determine this region for the Leech lattice [CS82c]. The Voronoi region of Λ_{24} centered around the origin therefore has $196560 + 16773120 = 16969680$ faces.

3.2.4 Symmetries

All lattices give rise to groups, due to their symmetry. We will begin with a definition.

Definition 3.3 (Automorphism group)**.** The automorphism group of a lattice $\mathcal{L}$ denoted $\mathrm{Aut}(\mathcal{L})$, is the set of distance-preserving linear transformations of the space that fix the origin and take the lattice to itself.

The automorphism group of Λ_{24} is the group Co_0, which has order $2^{22}3^9 5^4 7^2 11 \cdot 13 \cdot 23$. This group was discovered in 1968 by J. H. Conway [Con68]. Co_0 is particularly interesting due to its connection with the classification of finite groups. Co_0 led to the discovery of three new sporadic simple groups known as the Conway groups (Co_1, Co_2, and Co_3). Many of the 26 sporadic simple groups are found within Co_0, and it was also instrumental in the construction of another, the *Monster* simple group M.

3.2.5 The Niemeier lattices

While the Leech lattice is unique in many ways, it is not the only even unimodular lattice in 24 dimensions. Niemeier enumerated 24 such lattices, and showed that this enumeration is complete [Nie73]. Of these lattices, 23 have minimal norm $\sqrt{2}$, while just one, Λ_{24}, has minimal norm $\sqrt{4}$. Each Niemeier lattice is generated by gluing several component root lattices together. The component lattice with highest dimension uniquely identifies a Niemeier lattice [CS82b].

While this is interesting in and of itself, a much more remarkable result is the relationship between the deep holes of Λ_{24} and the Niemeier lattices. There are 23 inequivalent types of deep holes in Λ_{24}, and there is a one-to-one correspondence between these holes and the 23 Niemeier lattices [CPS82]. This led to the 23 constructions of the Leech lattice from each of the Niemeier lattices, in the form of the "holy construction" given in [CS82b]. For each of the 23 Niemeier lattices, we can define a set of fundamental vectors $\mathbf{v}_i$ and a set of glue vectors $\mathbf{g}_j$. Then the Niemeier lattice is the set of all integer combinations

$$\sum_i m_i \mathbf{v}_i + \sum_j n_j \mathbf{g}_j \quad \text{such that } \sum_j n_j = 0.$$

In turn, the Leech lattice is the set of all integer combinations

$$\sum_i m_i \mathbf{v}_i + \sum_j n_j \mathbf{g}_j \quad \text{such that } \sum_i m_i + \sum_j n_j = 0.$$

For example, one Niemeier lattice is generated by the component lattice A_1^{24}. The glue code for this lattice is the Golay code $\mathcal{C}_{24}$, of order 4096. This is equivalent to applying Construction A to $\mathcal{C}_{24}$. Another Niemeier lattice is characterized by the component lattices D_4^6. Here, the desired glue code is the hexacode H_6, which will be defined in the following section.

The different constructions of Λ_{24} from the other Niemeier lattices provides insight into the relationships between Λ_{24} and the root lattices. These relationships form the basis for many efficient decoding algorithms that have been discovered. In particular, the construction from D_4^6 will form the basis of the decoder described in the following chapter.

$$\frac{1}{\sqrt{8}}\begin{bmatrix}
8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 2 & 2 & 2 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 2 & 0 & 0 & 2 & 2 & 0 & 0 & 2 & 2 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 2 & 2 & 0 & 0 & 2 & 2 & 0 & 0 & 2 & 2 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 2 & 0 & 2 & 0 & 0 & 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 2 & 2 & 2 & 0 & 0 & 2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\
2 & 2 & 0 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\
0 & 2 & 2 & 2 & 2 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 2 & 2 & 0 & 0 & 2 & 2 & 0 & 0 & 2 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 & 2 & 0 \\
-3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}$$

Figure 3.1: Generator matrix for the Leech lattice in standard MOG coordinates, as given in [CS93]. The vectors are given as rows.

3.3 Integer Leech lattice

Cryptographic schemes based on lattices are typically defined over the integers, modulo some number n. In this context, it becomes important to find scaling factors that allow the Leech lattice to exist in discrete space.

The typical generator matrix for the Leech lattice is shown in Figure 3.1. Trivially, by multiplying the Leech lattice by a factor of $\sqrt{8}$, we obtain an integer Leech lattice scaling. Since $\det \Lambda_{24} = 1$, the determinant of this scaling is $(\sqrt{8})^{24} = 2^{36}$. Similarly, since $\lambda_1(\Lambda_{24}) = 2$, the minimal distance of this scaling is $2\sqrt{8}$. Multiplying further by arbitrary positive integers, we achieve a sequence of integer Leech lattice scalings, with volumes of 2^{24k+36}, where k is a positive integer.

There is, however, another sequence of integer scalings possible. We define a two dimensional transformation that rotates two dimensions through $\pi/4$ radians, and scales it by a factor of $\sqrt{2}$:

$$\mathbf{R} = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}. \tag{3.3}$$

By tensoring this rotation matrix with the 12 dimensional identity matrix, and multiplying the result by another factor of $\sqrt{2}$, we obtain a transformation $\mathbf{T}$ over 24 dimensions.

$$\mathbf{T} = \sqrt{2}\, \mathbf{I}_{12} \otimes \mathbf{R}.$$

When this transformation is applied to the generator matrix for the Leech lattice in Figure 3.1 (noting that the generator matrix represents vectors as rows), the resulting generator matrix is also integer. This scaling of the Leech lattice has determinant 2^{24} and minimal distance 4. This yields another sequence of integer Leech lattice scalings.

Combined, the two sequences defined above give Leech lattice scalings with volumes of 2^{12j} and minimal distances $2^{(j/2+1)}$, for integer $j \geq 2$. The associated transformations are defined as

$$\mathbf{T}(j) = \sqrt{2}\, \mathbf{I}_{12} \otimes \mathbf{R}^j. \tag{3.4}$$

Note that the first sequence is included, since when $\mathbf{I}_{n/2} \otimes \mathbf{R}^2$ is applied to a lattice of even dimension n, it is equivalent to scaling this lattice by a factor of two. Thus the transformation family $\mathbf{T}(j)$ contains both sequences described above.

In terms of notation, it would make sense to define a transformation $\mathbf{Y}$ such that $\mathbf{T}(j) = \mathbf{Y}^j$. However, the extra scale factor of $\sqrt{2}$ makes this difficult, unless we rotate the generator matrix for Λ_{24}. Rather than redefine this familiar basis, we leave $\mathbf{T}$ as is.

Chapter 4

Decoding the Leech lattice

Having demonstrated the remarkable properties of the Leech lattice, it becomes apparent that its unique density and high degree of structure make it particularly useful for coding purposes. Indeed, a lot of attention has been given to the decoding of the Leech lattice, using the general decoding strategies described in Section 2.4.

Early focus was primarily on maximum likelihood decoding, with bounded distance decoders arising from the constructions used for the maximum likelihood decoders. In 1984, Conway and Sloane proprosed a decoding algorithm for Λ_{24} based on the occurance of D_{24} as a sublattice of index 8192, resulting in a particularly slow algorithm [CS84]. Two years later, they improved on this by a factor of 14, noting that Λ_{24} could also be constructed from three glued copies of E_8, with index of 4096 [CS86]. In 1989, Be'ery et al constructed the Leech half-lattice using the Golay code and a parity check, resulting in a much more efficient decoder [BSS89]. Subsequent improvements split the Leech lattice into four cosets, allowing for an even better decoder based on the quaternary hexacode and two parity checks [VB93].

4.1 Advanced construction

The most efficient decoders (both maximum likelihood and bounded distance) use a construction of the Leech lattice based on the unique $[6, 3, 4]$ code over the Galois field $\mathbb{F}_4 = \{\mathbf{0}, \mathbf{1}, \omega, \bar{\omega}\}$, called the *hexacode*, H_6. This code has 64 codewords, a minimal distance of 4, and is described in detail in [CS93], where its relationship to the Leech lattice and relatively simple decoding algorithms are also discussed.

The construction given in [VB93] is very technical and written from an engineering point of view. We will first briefly restate this technical construction, as given in the paper. Then we will try to adopt a higher level overview, and describe the Leech lattice construction in terms of component lattices, cosets, and glue theory.

4.1.1 Technical construction

This technical construction used in the paper is difficult to understand from a lattice perspective. Nevertheless, we restate it first, and then attempt to motivate this construction in the following section.

We first define what it means to project onto the hexacode. A binary 4-tuple $\mathbf{a} = (a_1, a_2, a_3, a_4) \in \{0,1\}^4$ may be regarded as an interpretation of a character $\mathbf{x} \in \mathbb{F}_4$ by setting $(\mathbf{0}, \mathbf{1}, \omega, \bar{\omega}) \cdot \mathbf{a} = \mathbf{x}$. Conversely, the character $\mathbf{x}$ is a projection of $\mathbf{a}$. The specific construction of Λ_{24} is aided by the following notation.

The two-dimensional lattice D_2 is partitioned into 16 subsets, each labeled with a A_{ijk} or B_{ijk} as according to the diagrams in [VB93, ABV+94] (these papers also discuss the construction in greater detail). The subscripts i, j, k are binary variables. Each point of Λ_{24} is represented by a 2×6 array whose entries are points of D_2. An array has only A_{ijk} points (type-A) or only B_{ijk} points (type-B). For each two-element column $(A_{i_1 j_1 k_1}, A_{i_2 j_2 k_2})^t$ or $(B_{i_1 j_1 k_1}, B_{i_2 j_2 k_1})^t$, i_1 is called the h-parity, $k_1 \oplus k_2$ is called the k-parity, and the 4-tuple (i_1, j_1, i_2, j_2) is interpreted as a character $\mathbf{x} \in \mathbb{F}_4$. The column is denoted even or odd as according to the parity of (i_1, j_1, i_2, j_2). Then the Leech lattice is defined as follows.

Definition 4.1 (Leech lattice)**.** The Leech lattice Λ_{24} consists of all 2×6 arrays of points of D_2 such that:

1. The array is either type-A or type-B.
2. It consists of only even columns, or only odd columns.
3. If the array is type-A, the overall k-parity is even. Otherwise, it is odd.
4. If the array consists of even columns, the overall h-parity is even. Otherwise, it is odd.
5. The projection of the six columns is a codeword of H_6.

The distinction between type-A and type-B is equivalent to the distinction between the Leech half-lattice or its coset. The columns being even or odd further split the half-lattice cosets into the Leech quarter-lattice and its cosets. The hexacode and two parity checks complete the construction (note the dependency of the parity checks on the specific Leech quarter-lattice being investigated).

4.1.2 Glue theory construction

Remark. This description of the decoder assumes an integer Leech lattice, with minimal distance $4\sqrt{2}$.

We begin with the definition of several lattices. We will make extensive use of a particular transformation, an extension of the transformation defined in Equation 3.3 to four dimensions. From now on, $\mathbf{R}$ will refer to Equation 3.3 tensored with $\mathbf{I}_2$.

The root lattice D_4 is defined as $D_4 = \{(x_1, x_2, x_3, x_4) : x_1 + x_2 + x_3 + x_4 \equiv 0 \mod 2\}$. Applying the transformation $\mathbf{R}$, it is apparent that $\mathbf{R}D_4$ is equivalent to $\{(x_1, x_2, x_3, x_4) :$

$x_1, x_2, x_3, x_4 \equiv 0 \mod 2$ or $x_1, x_2, x_3, x_4 \equiv 1 \mod 2\}$. Generator matrices for these lattices can be defined as

$$D_4 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}, \qquad \mathbf{R}D_4 = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

It is easy to verify that $\mathbf{R}D_4$ is a sublattice of D_4, of index 4. Similarly, we can verify that $\mathbf{R}^2 D_4 = 2D_4$ is a sublattice of $\mathbf{R}D_4$, also of index 4. We obtain an infinite chain of sublattices $\mathbf{R}^j D_4$, where each sublattice has index 4 in the previous. We are particularly interested in the subchain $\mathbf{R}D_4, 2D_4, \mathbf{R}^3 D_4$ and $4D_4$. These lattices are the building blocks for this construction of the Leech lattice. The construction can be described in terms of three "levels", each corresponding to one partition in this subchain. Additionally, there is a small dependency between the first and third levels.

Level 1. The construction begins with the direct sum of six copies of $4D_4$, yielding the lattice

$$(4D_4)^6 = 4D_4 \oplus 4D_4 \oplus 4D_4 \oplus 4D_4 \oplus 4D_4 \oplus 4D_4.$$

This is a sublattice of Λ_{24}, of index 2^{18}. Now we take coset representatives for each coset of $4D_4$ in $\mathbf{R}^3 D_4$. We call these coset representatives $\mathbf{0}$, $\mathbf{g}_1$, $\mathbf{g}_2$ and $\mathbf{g}_1 + \mathbf{g}_2$, forming the glue group $G_{1,2}$. This glue group is isomorphic to $\mathbb{Z}_2^2$. We will use the isomorphism $\mu_{\mathbb{Z}_2^2} : G_{1,2} \mapsto \mathbb{Z}_2^2$. Define the parity set S_6 for an element $\mathbf{z} \in \mathbb{Z}_2^2$ as follows:

$$S_6(\mathbf{z}) = \left\{ (\mathbf{g}_1, \ldots, \mathbf{g}_6) \in G_{1,2} \oplus \cdots \oplus G_{1,2} \text{ such that } \sum_i \mu_{\mathbb{Z}_2^2}(\mathbf{g}_i) = \mathbf{z} \right\}.$$

Note that if $\mathbf{z} = (0,0)$, S_6 is a group. We can now define the lattice

$$\mathcal{L}' = \bigcup_{\mathbf{z} \in \mathbb{Z}_2^2} S_6(\mathbf{z}) + (4D_4)^6.$$

Note that $\mathcal{L}' = (\mathbf{R}^3 D_4)^6$. If we fix the parity $\mathbf{z} = (0,0)$, the resulting sublattice is also a sublattice of the Leech lattice, of index 2^8. The parity set S_6 is essentially a double parity check over the cosets of the six orthogonal copies of $4D_4$. They are equivalent to the h and k parities defined in the previous section.

Level 2. Similar to the first level, we let the four coset representatives of $\mathbf{R}^3 D_4$ in $2D_4$ be $\mathbf{0}, \mathbf{g}_3, \mathbf{g}_4$, and $\mathbf{g}_3 + \mathbf{g}_4$. This forms a glue group we will call $G_{3,4}$, which is also isomorphic to $\mathbb{Z}_2^2$. This time, we choose a slightly different isomorphism. We define the group isomorphism $\mu_{\mathbb{F}_4} : G_{3,4} \mapsto \mathbb{F}_4$ (ignoring the multiplication operation of $\mathbb{F}_4$). This places us in a position to define the hexacode group G_{H_6}.

$$G_{H_6} = \left\{ (\mathbf{g}_1, \ldots, \mathbf{g}_6) \in G_{3,4} \oplus \cdots \oplus G_{3,4} \text{ such that } (\mu_{\mathbb{F}_4}(\mathbf{g}_1), \ldots, \mu_{\mathbb{F}_4}(\mathbf{g}_6)) \in H_6 \right\}.$$

We can define the lattice

$$\mathcal{L}'' = \bigcup_{\mathbf{z}\in\mathbb{Z}_2^2} \Big(G_{H_6} \times S_6(\mathbf{z}) \Big) + (4D_4)^6.$$

As before, we obtain a sublattice of Λ_{24} if we fix the parity $\mathbf{z} = (0,0)$. Since the hexacode contains 2^6 codewords, the index of this sublattice is 2^2. This sublattice has a name, as it is known as the Leech quarter-lattice Q_{24}. Obviously, Q_{24} is a sublattice of $(2D_4)^6$. It also happens to be equal to the intersection $(2D_4)^6 \cap \Lambda_{24}$.

Level 3. Lastly, we follow the same routine and let the four coset representatives of $2D_4$ in $\mathbf{R}D_4$ be $\mathbf{0}, \mathbf{g}_5, \mathbf{g}_6$, and $\mathbf{g}_5 + \mathbf{g}_6$. We call the resulting glue group $G_{5,6}$. Again, we define an isomorphism $\mu'_{\mathbb{Z}_2^2} : G_{5,6} \mapsto \mathbb{Z}_2^2$. Next we define the repetition set $S_6^\perp$ as

$$S_6^\perp(\mathbf{z}) = \Big\{ (\mathbf{g}, \ldots, \mathbf{g}) \in G_{5,6} \oplus \cdots \oplus G_{5,6} \text{ such that } \mu'_{\mathbb{Z}_2^2}(\mathbf{g}) = \mathbf{z} \Big\}.$$

For a given $\mathbf{z} \in \mathbb{Z}_2^2$, the size of this set is one. We can now define the Leech lattice.

$$\Lambda_{24} = \bigcup_{\mathbf{z}\in\mathbb{Z}_2^2} \Big(S_6^\perp(\mathbf{z}) \times G_{H_6} \times S_6(\mathbf{z}) \Big) + (4D_4)^6.$$

The Leech lattice is therefore a sublattice of $(\mathbf{R}D_4)^6$, of index 2^{18}. To verify this, we note that the third level glue $S_6^\perp$ "loses" density in the order of $(2^2)^5$ due to its repetition. The complement of the second level hexacode glue is size 2^6, while the first level glue loses density in the order of 2^2, due to the parity conditions set by the third level. Combined, we obtain a loss of 2^{18} cosets. This leaves 2^{18} cosets of $(4D_4)^6$ in Λ_{24}.

The relationship to the previous technical construction now becomes apparent. One column of the array given in Definition 4.1 can therefore be interpreted as the particular coset of $4D_4$ in $\mathbf{R}D_4$, out of the 64 possibilities. The requirements one through five restrict the ways that these columns, or cosets, are glued together.

The first two requirements of Definition 4.1 correspond to the coset of Q_{24}, which is level 3 of our construction using glue theory. For example, type-A and even columns define Q_{24}, while other combinations describe its cosets. The fifth requirement is fulfilled by the coset representatives and glue group defined in level 2. Finally, the third and fourth requirements correspond to level 1 of the construction, and show the dependency between the first and third levels. In other words, the coset of Q_{24} being examined determines the value of the parity of the first level.

4.2 Maximum likelihood decoding

The following maximum likelihood decoding algorithm was given by Vardy and Be'ery in [VB93]. It appears to be the fastest maximum likelihood decoder known for the Leech lattice.

As the construction showed, a decoder for Λ_{24} may be regarded as four decoders for Q_{24} and its cosets. By choosing the most likely of the four outputs, the final decision is reached. The following therefore describes only a decoder for Q_{24}.

As in the previous section, the decoder given in the paper is very technical. We will restate this algorithm first, and then attempt to explain it in terms of the strategies discussed in Section 2.4.

4.2.1 Technical description

The decoder assumes that the channel output consists of 12 two-dimensional symbols $\{r(n)\}_{n=1}^{12}$, where $r(n) \in \mathbb{R}^2$ for $n = 1, \ldots, 12$. For each $r(n)$, the decoder finds in each A_{ijk} subset a point $\hat{A}_{ijk}$ closest to $r(n)$. During the precomputation step, we calculate the following two values for each $r(n)$.

$$d_{ij}(n) = \min\left\{\|\hat{A}_{ij0} - r(n)\|, \|\hat{A}_{ij1} - r(n)\|\right\} \quad \text{for } n = 1, \ldots, 12$$

$$\delta_{ij}(n) = \|\hat{A}_{ij1} - r(n)\| - \|\hat{A}_{ij0} - r(n)\| \quad \text{for } n = 1, \ldots, 12$$

These values need only be calculated once for each A_{ijk}, and are the input to the Q_{24} decoder and its coset in the Leech half-lattice (the B_{ijk} values are input to the other two cosets). The decoding algorithm consists of five steps, described briefly here. See [VB93] for a more detailed explanation.

1. **Computing the confidence values, preferable representations, and penalties.** This step involves calculating the metrics for the representatives of the 16 cosets of $4D_4$ in $2D_4$. These metrics are categorized as a preferred metric (the closest representative) and three penalties for each character in $\mathbb{F}_4$.

2. **Sorting the penalties.** This step sorts all the penalties with the same alternate (h, k)-parities relative to the preferable representation. This results in three ordered sets of 24 penalties. These will be used to later to resolve the total (h, k)-parity as according to the definition of Λ_{24}.

3. **Computing the confidence values of the blocks.** The vectors $\mathbf{x} = (x_1, \ldots, x_6) \in \mathbb{F}_4^6$ are split up into three blocks of two coordinates. For each $\{x_i, x_{i+1}\}$, the confidence values of the preferable representations are summed.

4. **Finding the images of the hexacodewords.** Each hexacodeword $\mathbf{x} \in H_6$ may be regarded as the projection of exactly 2^{18} points in Λ_{24}, of which 2^{16} are points of Q_{24}. The decoder determines for each $\mathbf{x} \in H_6$ which of these points is closest to the received signal. The decoder can use the previously sorted penalties to resolve any disparities in the (h, k)-parity.

5. **Computing the metrics of the hexacodewords and final minimization.** The metric for each hexacodeword is calculated using the confidence values of the blocks in step 3, with appropriate penalties added to resolve (h, k)-parities. The point of Q_{24} that projects on the hexacodeword with minimum metric, and is closest to the received signal among these points is the output of the Q_{24} decoder.

The decoder described above decodes the component lattices $4D_4$ as according to Section 2.4. This decoding is trivial, and very inexpensive computationally. Earlier work [CS86] advised looking for sublattices of small index. The index for $(4D_4)^6$ is large, but the decoder exploits the relationships between the glue vector components to a great degree. In essence, the algorithm comprises of repetitive maximum likelihood decoding of the codes representing the glue vectors, as opposed to delegating the bulk of the computation to the component lattice decoders.

In total, Vardy *et al.* calculate the complexity to be 2955 operations on average (not including the precomputation, memory addressing, negation, and absolute value), and 3595 in the worst case.

4.2.2 Glue theory description

We follow the previous technical description of the decoder with an explanation in terms of glue theory. Recall the three different levels of the Leech lattice construction, as given in Section 4.1.2.

The decoder given by Vardy and Be'ery only decodes a total of 2^{24} Leech lattice points. This includes the glue group described in the construction, whose order is 2^{18}. This leaves a very small subset of points in the cosets of $4D_4$. Extending the decoder to a larger number of points is not discussed explicitly in the paper; instead it is assumed that the distance of the target point to the closest vector of each of the 64 cosets of $4D_4$ is given. In practice, this is not an issue, since D_n can be decoded linearly in the number of coordinates [CS93].

Third level decoding. The third level of the construction is decoded using Algorithm 2.4. This splits the Leech lattice into four cosets of Q_{24}, and chooses the closest of the four as output. Each is decoded independently, meaning the repetition glue $S_6^\perp$ is known. This leaves only 16 cosets of $4D_4$ in $2D_4$ to be considered. Additionally, the parities required in the first level are known.

Second level decoding. Similarly, the second level of the construction is decoded using a variant of Algorithm 2.5. The hexacode glue group G_{H_6} is enumerated entirely, and the metrics of the glue corresponding to each hexacodeword are compared. There is significant redundancy here due to the six component lattices, since the cosets of $\mathbf{R}^3 D_4$ in $2D_4$ for each block are reused for many hexacodewords. These need only be calculated once, resulting in cost savings according to Lemma 2.4. The hexacodeword with the best metric is returned as output of the Leech quarter-lattice decoder.

First level decoding. The first level is where it gets interesting, and also where the decoder's efficiency comes from. The parity glue S_6 is decoded in this level, using the strategy described in Section 2.4.3. As stated in the strategy, if the parities are already correct, then the decoder is done. Otherwise, due to the small parity group (isomorphic to $\mathbb{Z}_2^2$), the use of the Hungarian algorithm is unecessary, as it requires too much overhead. Instead, the penalties corresponding to a given change in parity across each block are sorted, and the ways to fix the parities are enumerated, and the best is chosen.

The enumeration of the second and third levels make a lot of the above calculations redundant. The decoder makes efficient use of this redundency, greatly reducing the number of coset calculations that need to be done. In addition to the cost savings previously mentioned for the second level, some of the calculations associated with each coset of Q_{24} in each Leech half-lattice can be reused. Additionally, the sorting of the penalties can be amortized over all of the hexacodewords, resulting in a single sort of 24 elements for each parity syndrome (4 cosets of $\mathbf{R}^3 D_4$ in $2D_4$, in 6 blocks), as opposed to 64 sorts of 6 elements per parity syndrome. The preferable representations and associated penalties obviously also need only be calculated once per coset of $\mathbf{R}^3 D_4$ in $2D_4$, as opposed to being recalculated for each hexacodeword.

In short, the multi-level construction of the glue group for the Leech lattice, along with a small and very simple sublattice allows for very efficient decoding.

4.3 Bounded distance decoding

The maximum likelihood decoding algorithm described in the previous two sections spurred the development of similar bounded distance decoders, using an identical or nearly identical construction of the Leech lattice. In 1994 Amrani *et al.* [ABV$^+$94] developed a bounded distance decoder of Q_{24} using the same maximum likelihood decoder of the hexacode H_6 as in [VB93]. It should be noted that their multilevel construction of Λ_{24} is slightly different to eliminate the dependency between the first and last levels. They note that this difference does not have an effect on the complexity of the algorithm. This result was further improved in [AB96] to include a bounded distance hexadecoder. Finally in 1995 Vardy improved the bounded distance hexadecoder to achieve a bounded distance Leech lattice decoder using 331 real operations in the worst case, an order of magnitude faster than the corresponding maximum likelihood decoder.

An important result for all these decoders is that the error-correction radius achieved is equal to that of a maximum likelihood decoder.

Overview. The main speedup exploited in the bounded distance decoders is the removal of the hexacode enumeration. Due to the guarantee that the target point lies within the error-correction radius of a lattice point, we are given the guarantee that the confidence values for each hexacodeword character will decode to the correct hexacodeword, even if the resulting parities are not valid. This allows the application of a decoder to the hexacode, using only the metrics of the preferable representation in each direct sum block. Not only

do we not have to enumerate the hexacode, instead letting a decoder do that work for us, we only need to resolve penalties for one hexacodeword instead of 64. This results in a large reduction in complexity.

While the bounded distance decoder is much more time efficient, we have no guarantee on the output given a point outside the error-correction radius. Depending on the application, this may or may not be acceptable.

In our case, we wish to use the efficient encoding of the Leech lattice to improve the bandwidth of an encryption scheme. We are less concerned with time efficiency, as long as it remains acceptable. Therefore, we focus on maximum likelihood (CVP) decoders, as opposed to bounded distance decoders.

Chapter 5

Learning with errors

In this section we formalize the learning with errors problem (LWE). We then describe a typical encryption scheme based on this problem, and provide a proof of security. We examine the theoretical implications of the use of the Leech lattice for encoding and decoding. By comparing the resulting parameter sets to those of the current approach, which is integer lattice encoding and decoding, we observe that the Leech lattice can be used to significantly decrease the bandwidth of an LWE encryption or key exchange scheme.

5.1 Foundations

We will begin with the definition of the original LWE problem, and then some equivalent versions more closely related to the encryption scheme.

Definition 5.1 (Decision LWE problem)**.** Let n and q be positive integers, and χ an error distribution over $\mathbb{Z}$. Let $\mathbf{s}$ be a uniformly random vector in $\mathbb{Z}_q^n$. Define two oracles:

$$A_{\mathbf{s},\chi} : \mathbf{a} \xleftarrow{\mathcal{U}} \mathbb{Z}_q^n, e \xleftarrow{\chi} \mathbb{Z}_q;\ \text{return } (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e).$$
$$U : \mathbf{a} \xleftarrow{\mathcal{U}} \mathbb{Z}_q^n, u \xleftarrow{\mathcal{U}} \mathbb{Z}_q;\ \text{return } (\mathbf{a}, u).$$

The decision LWE problem for n, q, and χ is to distinguish $A_{\mathbf{s},\chi}$ from U.

Regev introduced the LWE problem in [Reg09], along with a quantum reduction from the decision version of LWE to the GapSVP and SIVP problems on arbitrary n-dimensional lattices. The GapSVP problem (decisional approximate SVP) is a decision variant of SVP, while SIVP (approximate shortest independent vectors problem) is a variant of SVP that attempts to find an entire short basis instead of a single short vector. Subsequent work [Pei09, BLP+13] showed that for $q = \text{poly}(n)$, LWE classically reduces to the same standard worst-case lattice problems. These worst-case hardness guarantees form a strong foundation on which to construct cryptographic primitives.

In a variant of the original LWE problem, the secret $\mathbf{s}$ is sampled from χ instead of $\mathcal{U}$, called a *short secret*. This alteration enjoys a polynomial reduction to the original decision LWE problem [ACPS09].

Definition 5.2 (Matrix decision LWE problem with short secrets)**.** Let m, n and q be positive integers, and χ an error distribution over $\mathbb{Z}$. Let k be a positive integer, and let $\mathbf{S} \xleftarrow{\chi} \mathbb{Z}_q^{n\times k}$. Define two oracles:

$$A_{\mathbf{S},\chi} : \mathbf{A} \xleftarrow{\mathcal{U}} \mathbb{Z}_q^{m\times n}, \mathbf{E} \xleftarrow{\chi} \mathbb{Z}_q^{m\times k};\ \text{return } (\mathbf{A}, \mathbf{AS}+\mathbf{E}).$$
$$U : \mathbf{A} \xleftarrow{\mathcal{U}} \mathbb{Z}_q^{m\times n}, \mathbf{U} \xleftarrow{\mathcal{U}} \mathbb{Z}_q^{m\times k};\ \text{return } (\mathbf{A}, \mathbf{U}).$$

The matrix decision LWE problem with short secrets is to distinguish $A_{\mathbf{S},\chi}$ from U.

A standard hybrid argument shows that using any adversary that can distinguish between the two distributions with advantage ϵ, one can construct an efficient adversary that distinguishes the original decision LWE problem with an advantage of minimally $\epsilon/(mk)$.

5.2 Cryptosystem

The LWE problem is parameterized by the modulus q, the dimension n, and an error distribution χ. We choose χ to be a continuous Gaussian distribution on $\mathbb{Z}$, with a mean of 0 and standard deviation of σ. These parameters determine the security of the scheme.

Remark. Note that the following assumes the values drawn from χ are integer. Indeed, in a real application of an LWE based cryptosystem, χ would be a discrete distribution that approximates Gaussian as closely as possible. However, to facilitate our analysis, it is easier to model χ as a continuous Gaussian. Since our purpose is to compare the efficiency of two different types of encoding, rather than examine specific parameter sets, we deem this acceptable.

Key generation. A uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$ is set, and shared by all users. This matrix is generated by a trusted source, and used by all parties in the system. If such a source is not available, then $\mathbf{A}$ may be generated as part of the key generation algorithm, and distributed along with the public key.

To generate a secret and public keypair, two matrices are sampled from the error distribution as $\mathbf{S}, \mathbf{E} \xleftarrow{\chi} \mathbb{Z}^{n\times k}$. $\mathbf{S}$ is the secret key, while the public key is defined as $\mathbf{B} := \mathbf{AS}+\mathbf{E} \in \mathbb{Z}_q^{n\times k}$.

Encryption. The matrix $\mathbf{A}$ and public key $\mathbf{B}$ are concatenated into a matrix

$$\mathbf{A}' = \begin{bmatrix} \mathbf{A}^t \\ \mathbf{B}^t \end{bmatrix} \in \mathbb{Z}_q^{(n+k)\times n}.$$

Two vectors are sampled from the distribution: $\mathbf{s}' \xleftarrow{\chi} \mathbb{Z}^n, \mathbf{e}' \xleftarrow{\chi} \mathbb{Z}^{n+k}$. To encrypt a binary message $\mathbf{m} \in \{0,1\}^l$ using the public key $\mathbf{B}$, one generates the ciphertext

$$\mathbf{c} = \mathbf{A}'\mathbf{s}' + \mathbf{e}' + (\mathbf{0}, \text{enc}(\mathbf{m})) \in \mathbb{Z}_q^{n+k},$$

where the function enc$(\cdot)$ outputs a vector of length k. Typically, the encoding used is the lattice code $\frac{q}{2}\mathbb{Z}^k \mod q$, which is isomorphic to $\{0,1\}^k$. This makes encoding very easy; enc$(\cdot)$ is defined as $\text{enc}(\mathbf{m}) := \left\lfloor \frac{q}{2} \right\rceil \cdot \mathbf{m}$, where $l = k$. Similarly, decoding is a simple rounding function. However, a more complicated but also more efficient lattice code (such as the Leech lattice) may allow for smaller failure rates and/or parameters.

Decryption. To decrypt $\mathbf{c}$ given the secret key $\mathbf{S}$, one computes

$$\begin{aligned}
\begin{bmatrix} -\mathbf{S}^t & \mathbf{I}_k \end{bmatrix} \cdot \mathbf{c} &= \begin{bmatrix} -\mathbf{S}^t & \mathbf{I}_k \end{bmatrix} \cdot (\mathbf{A}'\mathbf{s}' + \mathbf{e}') + \text{enc}(\mathbf{m}) \\
&= -\mathbf{S}^t\mathbf{A}^t\mathbf{s}' + \mathbf{B}^t\mathbf{s}' + \begin{bmatrix} -\mathbf{S}^t & \mathbf{I}_k \end{bmatrix} \cdot \mathbf{e}' + \text{enc}(\mathbf{m}) \\
&= -\mathbf{S}^t\mathbf{A}^t\mathbf{s}' + (\mathbf{AS})^t\mathbf{s}' + \mathbf{E}^t\mathbf{s}' + \begin{bmatrix} -\mathbf{S}^t & \mathbf{I}_k \end{bmatrix} \cdot \mathbf{e}' + \text{enc}(\mathbf{m}) \\
&= \underbrace{\mathbf{E}^t\mathbf{s}' + \begin{bmatrix} -\mathbf{S}^t & \mathbf{I}_k \end{bmatrix} \cdot \mathbf{e}'}_{\text{error term}} + \text{enc}(\mathbf{m}) \\
&\approx \text{enc}(\mathbf{m}) \mod q
\end{aligned} \tag{5.1}$$

where the approximation relies on the small size of $\mathbf{S}$, $\mathbf{E}$, $\mathbf{s}'$, and $\mathbf{e}'$ which were all drawn independently, coordinate-wise from χ. For the typical definition of enc$(\cdot)$, this amounts to testing whether each of the k coordinates is closer to 0 or $\left\lfloor \frac{q}{2} \right\rceil$ modulo q.

5.3 Security

The cryptosystem above is very similar to two separate instances of the LWE problem. One is the public key, while the other is the ciphertext.

Theorem 5.1. *The cryptosystem described in Section 5.2 is IND-CPA secure assuming the hardness of matrix decision-LWE with short secrets with modulus q, dimension n, and error distribution χ.*

Proof. To prove theorem 5.1, we only need to show that the entire view of an efficient, passive (eavesdropping) adversary is indistinguishable from uniformly random, for any plaintext $\mathbf{m} \in \{0,1\}^l$. The adversary's view consists of $(\mathbf{A}, \mathbf{B}, \mathbf{c})$. The tuple $(\mathbf{A}, \mathbf{B})$ is computationally indistinguishable from $(\mathbf{A}, \mathbf{U})$, where $\mathbf{U} \in \mathbb{Z}_q^{n \times k}$ is uniformly random, under the assumption in the theorem. This can be verified by noting that $\mathbf{A}$ is uniformly random and that $\mathbf{B}$ is constructed by sampling χ as in definition 5.2. Similarly, $(\mathbf{A}, \mathbf{U}, \mathbf{c})$ is computationally indistinguishable from $(\mathbf{A}, \mathbf{U}, \mathbf{u})$ where $\mathbf{u} \in \mathbb{Z}_q^{n+k}$ is uniformly random, under the same assumption (but for slightly different dimensions; there are $n + k$ LWE samples). Therefore, the passive adversary's view $(\mathbf{A}, \mathbf{B}, \mathbf{c})$ is indistinguishable from $(\mathbf{A}, \mathbf{U}, \mathbf{u})$. □

It is important to note that the security of the scheme relies only on the dimensions n, k, the modulus q, and the error distribution χ. The type of encoding function used has no effect, meaning we are free to modify this function as we see fit (keeping in mind that the decoding function must still work correctly, in the presence of errors).

5.4 Encoding

The encoding used determines how many bits are extracted from each of the k coordinates. However, it also has implications for the size of the error tolerated. For decryption to be correct, the accumulated errors must be small enough that the $\mathrm{dec}(\cdot)$ function can correctly decode the computed ciphertexts to the message $\mathbf{m}$. The failure rate is therefore dependent on the error distribution and the encoding used.

5.4.1 Current approach

We assume any distribution χ, and an encoding function defined as

$$\mathrm{enc}(\mathbf{m}) := \left\lfloor \frac{q}{2} \right\rceil \cdot \mathbf{m} \quad \text{where } \mathbf{m} \in \{0,1\}^k. \tag{5.2}$$

After the addition of errors, recovering $\mathbf{m}$ is done bitwise, each coordinate being independent of the others. This is akin to decoding the lattice $\frac{q}{2}\mathbb{Z}_q^k$. Thus for each coordinate, the accumulated error (calculated in equation 5.1) must be bounded by some value. For the correct decoding of coordinate i, where $0 < i \leq k$, we require that

$$\left|(\mathbf{E}^t\mathbf{s}' + \begin{bmatrix}-\mathbf{S}^t & \mathbf{I}_k\end{bmatrix}\mathbf{e}')_i\right| < L$$

where L is the upper bound on the error that the decoder can tolerate. For the above definition of $\mathrm{enc}(\cdot)$, it follows that $L = \frac{q}{4}$.

Since the coordinates of each vector and each matrix involved are all independently drawn from the same distribution, the distribution of the accumulated error is identical for each coordinate. We can therefore drop i from our calculations. For clarity, we now redefine some vectors. Let $\mathbf{e}$ refer to the i-th row of $\mathbf{E}^t$, and $\mathbf{s}$ refer to the i-th row of $\mathbf{S}^t$. Let $\mathbf{e}''$ refer to the first n coordinates of $\mathbf{e}'$, and let e''' be the $(n+i)$-th coordinate of $\mathbf{e}'$. Then the accumulated error is equal to

$$\begin{aligned} &\langle \mathbf{e}, \mathbf{s}' \rangle + \langle \mathbf{s}, \mathbf{e}'' \rangle + e''' \\ &= \langle (\mathbf{e}, \mathbf{s}, e'''), (\mathbf{s}', \mathbf{e}'', 1) \rangle. \end{aligned} \tag{5.3}$$

Each coordinate in the vectors $(\mathbf{e}, \mathbf{s}, e''')$ and $(\mathbf{s}', \mathbf{e}'')$ were independently drawn from the same distribution, χ.

The distribution of this error depends on χ. In typical schemes, χ is taken to be a discrete distribution that very closely approximates the continuous Gaussian of a certain variance (see [BCD+16] for such a scheme). The Rényi divergence between this distribution and the continuous Gaussian is used in the security reduction. The failure rates are then calculated by means of a computationally intensive analysis.

We are interested in comparing the current method of integer lattice encoding to our proposed Leech lattice encoding, and not the specific nature of the error distribution, which can differ between schemes and implementations. Therefore, we will calculate the failure rates by modeling χ as a continuous Gaussian. Since both encoding methods deal with the

same accumulated errors in each coordinate, we expect that this approximation in the exact nature of χ will have little impact on our comparisons.

By taking χ to be Gaussian, we obtain the following lemma.

Lemma 5.1 (Failure rate for integer encoding)**.** *Assuming χ to be a Gaussian of mean 0 and variance σ^2, and an encoding function defined as in 5.2, the probability* $\Pr[F]$ *that a ciphertext generated by the cryptosystem in Section 5.2 is decoded incorrectly is bounded by*

$$\Pr\left[F\right] \leq k\left[2 - \operatorname{erf}\left(\frac{L}{\sqrt{2B}\,\sigma}\right) - \frac{\gamma\left(n, \frac{B-1}{2\sigma^2}\right)}{\Gamma(n)}\right],$$

where Γ is the ordinary gamma function, γ is the lower incomplete gamma function, and B is any real number greater than 1.

To obtain the tightest possible bound on the failure rate, B should be tuned numerically for each set of parameters.

Proof. Each coordinate in the vector $(\mathbf{e}, \mathbf{s}, e''')$ is independently drawn from χ, so we may conclude it has a spherically symmetric distribution (since χ is Gaussian with mean 0). As such, the distribution of the inner product in 5.3 depends only on $\|(\mathbf{s}', \mathbf{e}'', 1)\|$, and not the direction of this vector. It is therefore normally distributed, with a mean of 0 and a variance of $\sigma^2\|(\mathbf{s}', \mathbf{e}'', 1)\|^2$. The quantity $\|(\mathbf{s}', \mathbf{e}'', 1)\|^2$ is distributed according to

$$\begin{aligned}\|(\mathbf{s}', \mathbf{e}'', 1)\|^2 &= \|(\mathbf{s}', \mathbf{e}'')\|^2 + 1 \\ &\sim \sigma^2\chi^2_{2n} + 1 \\ &\sim \Gamma_n(2\sigma^2) + 1\end{aligned}$$

where χ^2_{2n} is the chi-squared distribution with $2n$ degrees of freedom, and $\Gamma_n(2\sigma^2)$ is the gamma distribution with shape parameter n and scale parameter $2\sigma^2$. Let F_1 be the event that $\|(\mathbf{s}', \mathbf{e}'', 1)\|^2$ is larger or equal to some bound B. The probability of this happening is equal to the complementary cumulative distribution function of Γ_n evaluated at $B-1$. This is equal to

$$\Pr\left[F_1\right] = 1 - \frac{\gamma\left(n, \frac{B-1}{2\sigma^2}\right)}{\Gamma(n)}$$

where $\Gamma(n)$ is the ordinary gamma function, and $\gamma(\cdot, \cdot)$ is the lower incomplete gamma function, defined as

$$\gamma(s, x) = \int_0^x t^{s-1}\mathrm{e}^{-t}\mathrm{d}t.$$

Let F_2 be the event that $\|(\mathbf{s}', \mathbf{e}'', 1)\|^2 < B$, and that $|N(0, \sigma^2 B)|$ is larger or equal to the upper bound L tolerated by the decoder. Then

$$\begin{aligned}\Pr\left[F_2\right] &= 2\left[1 - \Phi\left(\frac{L}{\sigma\sqrt{B}}\right)\right] \\ &= 2\left[1 - \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{L}{\sqrt{2}\sigma\sqrt{B}}\right)\right]\right] \\ &= 1 - \operatorname{erf}\left(\frac{L}{\sqrt{2B}\,\sigma}\right)\end{aligned}$$

where Φ is defined as the cumulative density function of the standard normal distribution, and erf is the Gauss error function

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \mathrm{e}^{-t^2} \mathrm{d}t.$$

Then the probability of the event F_0 that $|\langle(\mathbf{e}, \mathbf{s}, e'''), (\mathbf{s}', \mathbf{e}'', 1)\rangle| \geq L$, or equivalently, that one bit is incorrectly decoded, is upper bounded by

$$\Pr\left[F_0\right] \leq \Pr\left[F_1\right] + \Pr\left[F_2\right].$$

This holds in any single coordinate. By applying the union bound, we can calculate an upper bound on the failure rate of the entire decryption algorithm.

$$\Pr\left[F\right] \leq k\left[2 - \operatorname{erf}\left(\frac{L}{\sqrt{2B}\,\sigma}\right) - \frac{\gamma\left(n, \frac{B-1}{2\sigma^2}\right)}{\Gamma(n)}\right].$$

□

Remark. The above describes the failure rate for the encoding function $\mathrm{enc}(\mathbf{m}) := \left\lfloor \frac{q}{2} \right\rceil \cdot \mathbf{m}$. In practice, the encoding function is easily modified to extract multiple bits from each coordinate. This allows k to be made smaller, or alternatively allows for more bits to be encrypted in a single ciphertext. However, it also decreases the allowable L exponentially, and therefore drastically increases the failure rate, all else being equal. In any case, the calculations for the failure rate remain the same, and only k and L need to be adjusted.

5.4.2 Our Leech lattice approach

To improve the efficiency of the encoding, we can apply a suitably scaled and possibly rotated version of the Leech lattice, such that

$$q\mathbb{Z}^{24} \subset \mathbf{T}\Lambda_{24} \subset \mathbb{Z}^{24}.$$

In order for the encoding to fit into the cryptosystem, $\mathbf{T}$ must be chosen such that the Leech lattice remains integer. See Equation 3.4 for a family of transformations that fulfill this requirement. We can encode and decode binary data by defining a bijective function

$$f : \{0,1\}^l \mapsto \mathbf{T}\Lambda_{24}/q\mathbb{Z}^{24},$$

that maps bit strings to points in the Leech lattice modulo $q\mathbb{Z}^{24}$. One way to do this is given by the decoder described in Chapter 4, where the binary subscripts of the arrays defining Leech lattice points can be interpreted as a bit string. The choice of q and $\mathbf{T}$ will determine how many bits l are encoded in every 24 coordinates. However, it will also affect the failure probability for the decoder.

The correctness of the decoding is now dependent on the accumulated error vector in blocks of 24 coordinates. More specifically, assuming we have a CVP algorithm for Λ_{24},

correct decoding requires the error vector in each block to lie within the Voronoi cell of the encoded point (excluding the border). The Voronoi cell of the Leech lattice is bounded by hyperplanes corresponding to the first two layers of the lattice [CS93]. This includes $N_1 = 196560$ minimal vectors, and $N_2 = 16773120$ vectors in the second layer, $\sqrt{3/2}$ times the size of the first. This gives a total of 16969680 faces.

As in the integer lattice encoding approach, we assume χ is Gaussian. We obtain the following lemma.

Lemma 5.2 (Failure rate for Leech lattice encoding)**.** *Assuming χ to be a Gaussian of mean 0 and variance σ^2, and $\mathbf{R}\Lambda_{24}$ encoding, the probability* $\Pr[F]$ *that a ciphertext generated by the cryptosystem in Section 5.2 is decoded incorrectly is bounded by*

$$\Pr\left[F\right] \leq \frac{mN_1}{2}\left[1 - \operatorname{erf}\left(\frac{\lambda_1(\mathbf{R}\Lambda_{24})}{2\sqrt{2B}\,\sigma}\right)\right] + \frac{mN_2}{2}\left[1 - \operatorname{erf}\left(\frac{\sqrt{3}\,\lambda_1(\mathbf{R}\Lambda_{24})}{4\sqrt{B}\,\sigma}\right)\right] + 1 - \frac{\gamma\left(n, \frac{B-1}{2\sigma^2}\right)}{\Gamma(n)},$$

where B is a real number greater than 1.

Proof. Recall that the accumulated error is equal to

$$\begin{bmatrix}\mathbf{E}^t & -\mathbf{S}^t & \mathbf{I}_k\end{bmatrix} \cdot (\mathbf{s}', \mathbf{e}')^t.$$

This value follows directly from equation 5.1, and is not influenced by the type of encoding used. If we let $\mathbf{e}$ be the i-th row of $\mathbf{E}^t$, $\mathbf{s}$ the i-th row of $\mathbf{S}^t$, $\mathbf{e}''$ the first n coordinates of $\mathbf{e}'$, and e''' the $(n+i)$-th coordinate of $\mathbf{e}'$, we can write the accumulated error in the i-th coordinate as

$$\langle(\mathbf{e}, \mathbf{s}, e'''), (\mathbf{s}', \mathbf{e}'', 1)\rangle. \tag{5.4}$$

Note that the first vector $(\mathbf{e}, \mathbf{s}, e''')$ of the inner product is unique for each coordinate, while the second vector $(\mathbf{s}', \mathbf{e}'', 1)$ is the same for each i. The accumulated errors in each coordinate are therefore dependent, but only on the second vector in the inner product.

We follow the proof for 5.1, and "fix" the quantity $\|(\mathbf{s}', \mathbf{e}'', 1)\|$. This squared norm is distributed according to $\Gamma_n(2\sigma^2) + 1$. Again, we let F_1 be the event that $\|(\mathbf{s}', \mathbf{e}'', 1)\|^2$ is larger or equal to some bound B, which has probability

$$\Pr\left[F_1\right] = \frac{\gamma\left(n, \frac{B-1}{2\sigma^2}\right)}{\Gamma(n)}.$$

Due to the spherically symmetric nature of the vector $(\mathbf{e}, \mathbf{s}, e''')$, the distribution of the inner product in equation 5.4 is Gaussian with mean 0 and a variance of $\sigma^2\|(\mathbf{s}', \mathbf{e}'', 1)\|^2$.

We define a random variable $\mathbf{X} = (X_1, \ldots, X_{24})$ as the accumulated error in a block of 24 coordinates, subject to the restriction that $\|(\mathbf{s}', \mathbf{e}'', 1)\|^2 = B'$ for some $B' \leq B$. Then the X_i are normally distributed, with mean 0 and variance $\sigma^2 B'$. Since $(\mathbf{e}, \mathbf{s}, e''')$ is unique for each i, and B' is fixed, the X_i variables are independent. The variable $\mathbf{X}$ therefore follows a multivariate normal distribution. More formally, $\mathbf{X} \sim N_{24}(\mathbf{0}, \mathbf{\Sigma})$, where Σ is the covariance matrix. Due to the independence of the coordinates, $\mathbf{\Sigma}$ is a diagonal matrix with entries equal to $\sigma^2 B'$.

For $\mathbf{X}$ to lie within the Voronoi cell associated with the encoded point (excluding the border), we need that

$$\frac{\langle \mathbf{X}, \mathbf{v} \rangle}{\|\mathbf{v}\|^2} < \frac{1}{2} \quad \forall \mathbf{v} \in V$$

where V refers to the set of all Voronoi relevant vectors for the Leech lattice. For a given $\mathbf{v} \in V$,

$$\begin{aligned}\frac{\langle \mathbf{X}, \mathbf{v} \rangle}{\|\mathbf{v}\|^2} &= \frac{1}{\|\mathbf{v}\|^2} \sum_{i=1}^{24} v_i X_i \\ &\sim \frac{1}{\|\mathbf{v}\|^2} N(0, \|\mathbf{v}\|^2 \sigma^2 B') \\ &\sim N\left(0, \frac{\sigma^2 B'}{\|\mathbf{v}\|^2}\right).\end{aligned}$$

The probability of $\mathbf{X}$ lying on the border of, or on the wrong side of the Voronoi cell face corresponding to the lattice vector $\mathbf{v}$ is

$$\Pr\left[N\left(0, \frac{\sigma^2 B'}{\|\mathbf{v}\|^2}\right) \geq \frac{1}{2}\right] = \frac{1}{2}\left[1 - \operatorname{erf}\left(\frac{\|\mathbf{v}\|}{2\sqrt{2B'}\,\sigma}\right)\right] \leq \frac{1}{2}\left[1 - \operatorname{erf}\left(\frac{\|\mathbf{v}\|}{2\sqrt{2B}\,\sigma}\right)\right].$$

Recall the shape of the Voronoi cell for Λ_{24}. The quantities $\|\mathbf{v}\|$ only have two possible values, corresponding to the norms of first two layers of the Leech lattice. These values are $\lambda_1(\mathbf{R}\Lambda_{24})$ and $\sqrt{3/2}\,\lambda_1(\mathbf{R}\Lambda_{24})$.

We apply the union bound to all of the Voronoi relevant vectors, giving a bound on the probability that the accumulated error in a block of 24 coordinates lies outside the Voronoi cell. By applying the union bound yet again to the number m of Leech lattices needed to encode the desired number of points, we obtain an upper bound for the probability of F_2, or incorrect decryption, given $\|(\mathbf{s}', \mathbf{e}'', 1)\|^2 < B$:

$$\Pr\left[F_2\right] \leq \frac{mN_1}{2}\left[1 - \operatorname{erf}\left(\frac{\lambda_1(\mathbf{R}\Lambda_{24})}{2\sqrt{2B}\,\sigma}\right)\right] + \frac{mN_2}{2}\left[1 - \operatorname{erf}\left(\frac{\sqrt{3}\,\lambda_1(\mathbf{R}\Lambda_{24})}{4\sqrt{B}\,\sigma}\right)\right].$$

For the decryption to fail, one of F_1 (the event that $\|(\mathbf{s}', \mathbf{e}'', 1)\|^2 \geq B$) and F_2 (given $\neg F_1$, the event of incorrect decryption) must occur. Therefore, using the probabilities for these events, we can obtain an upper bound on the failure rate.

The upper bound on the failure rate for decryption becomes

$$\Pr\left[F\right] \leq \frac{mN_1}{2}\left[1 - \operatorname{erf}\left(\frac{\lambda_1(\mathbf{R}\Lambda_{24})}{2\sqrt{2B}\,\sigma}\right)\right] + \frac{mN_2}{2}\left[1 - \operatorname{erf}\left(\frac{\sqrt{3}\,\lambda_1(\mathbf{R}\Lambda_{24})}{4\sqrt{B}\,\sigma}\right)\right] + 1 - \frac{\gamma\left(n, \frac{B-1}{2\sigma^2}\right)}{\Gamma(n)}.$$

□

5.5 Attacks

In this section, we very briefly outline the main attacks against LWE schemes similar to the one above, and explain the different security estimates used in our comparisons, and the reasoning behind their values.

The number of LWE samples revealed to an attacker is small (n for the public key, and $n+k$ for the ciphertext). This rules out several possible strategies to break the scheme, based on large amounts of LWE samples. Instead, we are concerned with two practical attacks, typically referred to as *primal* and *dual* attacks. Both attacks use the BKZ algorithm, and rely on finding short vectors in a certain lattice.

SVP hardness. The Blockwise Korkine-Zolotarev (BKZ) algorithm [SE94] is used to solve for the desired short vectors. BKZ takes a blocksize parameter b as input, and involves approximately linear many calls to an SVP oracle in dimension b [CN11]. To account for possible future advancements and/or amortization in certain attacks, we lower bound this cost by the running time of a single SVP call in dimension b.

The exact cost of such a call is difficult to determine, but for large enough b ($b \geq 200$) we can bound the cost by $b2^{cb}$ operations (see [ADPS15, BCD$^+$16] for details). The constant c is provided by the best sieving algorithms available. In the classical case, it is given by $c_C = \log_2 \sqrt{3/2} \approx 0.292$. Including quantum algorithms, this constant is lowered to $c_Q = \log_2 \sqrt{13/9} \approx 0.265$. Lastly, it is plausible to lower bound this constant by noting that all sieving algorithms require a list of a certain number of vectors to be built. This lower bound is given by $c_P = \log_2 \sqrt{4/3} \approx 0.2075$.

These constants therefore imply different security estimates under classical and quantum assumptions. The constants c_C and c_Q can be used to calculate security estimates corresponding to state of the art classical and quantum adversaries. The value c_P can be used to calculate estimates for long term security allowing that significant advancements in SVP algorithms could be made.

Primal attack. The primal attack takes the LWE samples $(\mathbf{A}, \mathbf{b} = \mathbf{As}+\mathbf{e}) \in \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^{m\times 1}$ and builds the lattice

$$\Lambda = \{\mathbf{x} \in \mathbb{Z}^{m+n+1} : (\mathbf{A} \mid \mathbf{I}_m \mid -\mathbf{b})\mathbf{x} = \mathbf{0} \mod q\}.$$

This lattice can be regarded as a unique-SVP instance, where the solution yields the vector $\mathbf{v} = (\mathbf{s}, \mathbf{e}, 1)$. By sufficiently reducing the basis using BKZ, this vector can be found with a high degree of probability.

Dual attack. The dual attack constructs the lattice

$$\hat{\Lambda} = \{(\mathbf{x}, \mathbf{y}) \in \mathbb{Z}^n \times \mathbb{Z}^n : \mathbf{A}^t\mathbf{x} = \mathbf{y} \mod q\}$$

and uses BKZ to find many short vectors of the form $(\mathbf{v}, \mathbf{w})$. By computing $z = \langle \mathbf{v}^t, \mathbf{b} \rangle$ and analyzing its distribution, the attacker can distinguish between LWE samples and uniform samples.

The runtimes of these attacks determine the security estimate given to a parameter set. The number of LWE samples m that are used in BKZ can be chosen to be anywhere from 0 to $n + k$ (for the ciphertext, maximally n for the public key). For the dual attack, the optimal m is given in [MR09] as

$$m = \sqrt{n \log q / \log \delta}$$

where δ refers to the root-Hermite factor (a measure of how short the smallest vector in a lattice basis is). It is estimated to be the same for the primal attack [LP11]. This indicates it is easier to attack the ciphertext than the public key, due to the increased number of samples.

In [LP11], the running time of BKZ for large blocksize b (required by the two attacks above) is analyzed. We lower bound our estimates by assuming the attacker is attacking the ciphertext, and has access to an optimal number of samples. The analysis by Lindner and Peikert found the running time of BKZ to be primarily governed by δ, the root-Hermite factor. By computing a bound β on the length of a nonzero vector in the lattice Λ or $\hat{\Lambda}$ that would yield the desired results, the root-Hermite factor was computed to be

$$\delta = 2^{(\log^2 \beta)/(4n \log q)}$$

where β is proportional to q/σ. This gives us a rough idea of how the parameters n, q, and σ influence the security of the scheme. Additionally, it is apparent that k only influences the security insofar as to provide enough LWE samples for an optimal value of m.

Briefly, increasing q will decrease the security, while increasing σ or n increases the security (n much faster than σ). Increasing q and σ while keeping the ratio between them constant will increase the security.

5.6 Parameter selection

As shown in the previous section, the choices of n, q, k and σ have differing impacts on the security of the scheme. However, as long as the desired security estimate is met, their specific values can be shuffled around almost indefinitely to make the scheme more practical. We do, however, impose a lower bound on the size of σ. As a general rule, σ should not be smaller than 1. Under this bound, combinatorial attacks on the cryptosystem may become feasible.

In this section we compare parameter sets using our Leech lattice encoding approach to those using the encoding in use today, the more simple integer lattice encoding. We are particularly interested in parameter sets that minimize the bandwidth, while maintaining a certain level of security. We will consider two use cases. The first is as an encryption scheme, where it is assumed that public keys have already been distributed. The second is as a key exchange algorithm, where the size of the public key is as important as the size of the ciphertext. In practice, application in the latter would probably be more important.

encoding	q	n	σ^2	k	bits	failure	bandwidth CT
$\mathbb{Z}_q^k$	2^{12}	568	1.75	128	**256**	2^{-38}	**1044** B
	2^{13}	568	3.50	128	256	2^{-38}	1131 B
	2^{14}	572	7.00	128	256	2^{-37}	1225 B
Λ_{24}	2^{10}	500	1.15	192	**288**	2^{-37}	**865** B
	2^{11}	552	1.10	120	300	2^{-38}	924 B
	2^{12}	574	1.55	96	288	2^{-38}	1005 B
	2^{13}	576	3.10	96	288	2^{-37}	1092 B
	2^{14}	556	8.90	120	300	2^{-37}	1183 B

Table 5.1: **Encryption schemes.** Parameter sets achieving 128 bit post-quantum security estimates. The column bandwidth CT refers to the size in bytes of one ciphertext. The failure column denotes the probability of unsuccessful decryption.

5.6.1 Encryption schemes

For an encryption scheme, it is desirable to minimize the size of the ciphertext. It is assumed that the public keys have already been distributed, and do not need to be renewed per session. The size of the public key, therefore, is of secondary concern. Typically, the ciphertext will encrypt 256 uniformly random bits (to account for Grover's algorithm, discussed in Section 1.1), which will later be used to derive a session key for symmetric encryption. Table 5.1 shows some parameter sets for both integer and Leech lattice encoding. All sets achieve 128 bits of post-quantum security (as per the c_Q constant defined in the previous section), and admit approximately equal failure rates.

The best parameter set for integer encoding requires a ciphertext of 1044 bytes to encrypt 256 bits. By contrast, the best parameter set for Leech lattice encoding requires a ciphertext of size only 865. This corresponds to a reduction of 17.1% in the size of the ciphertext. The reason for this gain is primarily due to the lower bound we set on σ. One of the most efficient ways to reduce the size is by reducing q. However, this increases the effect of σ. For large relative error (e.g. q as small as possible, and σ close to our bound), the use of the Leech lattice gives greater control over k. We can extract an average of 1, 1.5, or 2 bits from each coordinate. Contrast this to integer encoding, for which we can choose to extract 1 or 2 bits from each coordinate, meaning $k = 256$ or $k = 128$! Combined with the fact that the Leech lattice can simply better tolerate large errors, we can decrease q further than normal.

If we compare parameter sets for similar q, the gains are much less. Taking $q = 2^{12}$, the reduction in ciphertext size is only 3.7%. This is similar for other values of q that are available for both encodings, given the bound on σ.

Note also the differences in bits of plaintext being encrypted. For integer encoding, it is straightforward to encrypt exactly 256 bits, by finetuning k. For Leech lattice encoding, this is not so simple. The value k must be a multiple of 24, since the Leech lattice exists in 24 dimensions. Similarly, the amount of bits that can be encoded in one copy of the Leech lattice is restricted to multiples of 12 (this is an artifact of the requirement that the Leech lattice

encoding	q	n	σ	k	$\bar{n}$	$\bar{m}$	bits	failure	bandwidth
$\mathbb{Z}_q^k$	2^{12}	568	1.75	128	12	11	256	2^{-38}	19.1 KB
	2^{13}	618	1.70	86	10	9	258	2^{-38}	18.6 KB
	2^{14}	664	1.65	64	8	8	**256**	2^{-38}	**18.2** KB
Λ_{24}	2^{10}	500	1.15	192	16	12	288	2^{-37}	17.1 KB
	2^{11}	552	1.10	120	12	10	**300**	2^{-38}	**16.3** KB
	2^{12}	574	1.55	96	12	8	288	2^{-37}	16.8 KB
	2^{13}	626	1.50	72	9	8	288	2^{-37}	16.9 KB
	2^{14}	700	1.00	48	8	6	264	2^{-38}	16.7 KB

Table 5.2: Parameter sets achieving 128 bit post-quantum security estimates. The bandwidth column refers to the size of the matrices sent by both parties, as in [BCD+16]. The failure column denotes the probability of unsuccessful decryption.

must be integer, see Section 3.3). This makes it difficult to find efficient combinations of σ and k that encrypt 256 bits with little overhead, resulting in the desired security parameter and failure rate bound.

If we fix q, and compare the ratios of ciphertext bits per plaintext bit for both types of encoding, we get a much more significant improvement. For example, for $q = 2^{14}$, the reduction in ciphertext size is only 3.4%. However, Leech lattice encoding is able to encrypt 300 bits per ciphertext, while integer encoding is limited to 256. In terms of ciphertext bits per plaintext bit, the gain is 17.6%. In practice, this gain is useless as an asymmetric scheme will only ever be used to agree on symmetric keys. However, it indicates the dimension of the Leech lattice is causing a significant loss in coding gain, when applied to keys of 256 bits.

5.6.2 Key exchange

In the context of key exchange algorithms, minimizing the size of the entire handshake is most important. In this case, the ciphertext becomes a matrix instead of a vector. In calculating the size of a handshake, we follow [BCD+16]. We split k into two factors, $\bar{n}$ and $\bar{m}$. The matrix sent by the first party is of dimension $n \times \bar{n}$, while the second party sends a matrix of size $\bar{m} \times n$ (we ignore the seed for $\mathbf{A}$, and the reconciliation matrix, as their sizes are negligible). This construction allows both parties to obtain a matrix of size $\bar{n} \times \bar{m}$, which can be reconciled to reveal 256 bits of shared secret data. Table 5.2 shows some parameter sets optimized for minimal handshake size. As before, all sets achieve 128 bit post-quantum security, and admit approximately equal failure rates.

Interestingly, this table is much different. Minimizing the total handshake size relies less heavily on the values of q and n. Instead, it is most beneficial to minimize $\bar{n}$ and $\bar{m}$, which have a much greater impact on the size of the handshake. This is done by decreasing the value of σ, allowing more bits to be encoded in every matrix coordinate.

The best parameter set using integer lattice encoding achieves a handshake size of 18.2

kilobytes. This set encodes four bits in every coordinate, meaning we can set $\bar{n} = \bar{m} = 8$. This results in room for $4 \times 8^2 = 256$ encoded bits. Encoding more bits would necessitate an increase in q, due to the lower bound we require on σ. This increase offsets any gains made by the smaller $\bar{n}$ or $\bar{m}$.

For Leech lattice encoding, the best parameter sets for each value of q are quite similar. The best has a handshake size of 16.3 kilobytes, which is a reduction of 10.2% compared to integer lattice encoding. This set makes use of very small q, and relatively large $\bar{n}$ and $\bar{m}$, resulting in 8 copies of the Leech lattice, encoding 36 bits each. However, it is also possible to use only 2 copies of the Leech lattice, encoding 144 bits each. This handshake is 16.7 KB, a reduction of 7.8% relative to integer lattice encoding.

For higher failure rates, in the order of 2^{-256}, the difference between integer and Leech lattice encodings is approximately the same.

Again, the obligation to agree on exactly 256 bits, combined with the unwieldy dimensions of the Leech lattice, causes a significant loss in coding gain. The best Leech lattice parameter set has room for 300 encoded bits, compared to exactly 256 for integer lattice encoding. While it is clear the Leech lattice is a superior method of encoding than the integer lattice and that this is applicable to LWE cryptosystems, the specifics of the Leech lattice do not mesh well with established conventions in security, which define security measures in powers of two.

5.6.3 Key exchange for 240 bits

As an experiment, we consider a scenario for which we only require the exchange of 240 bits of key material. This number fits the dimensions of the Leech lattice better, while it should have only a small impact on the efficiency of integer lattice encoding. This may result in a much smaller bandwidth for Leech lattice encoding, relative to current practices.

The requirement for 256 bits comes from Grover's algorithm [Gro96], which searches for input to a black box function in $O(\sqrt{n})$ time. Grover's algorithm is probabilistic, and sequential—each iteration uses the output of the previous iteration as input. The number of iterations determines the probability of success, and decreasing this number more than linearly decreases the success probability. In other words, there is no easy way to parallelize this algorithm. In practice, this may make the requirement for 256 bits unecessarily large.

We take the best parameter set for integer lattice encoding, for which

$$(q, n, \sigma) = (2^{14}, 664, 1.65)$$

and four bits are extracted from each coordinate. For 240 bits, we only need $k = 60$ coordinates. Unfortunately, the best matrix sizes $n \times \bar{n}$ and $\bar{m} \times n$ are still $\bar{n}, \bar{m} = 8$, meaning there are some unused coordinates (four of them). This means the overal size of the key exchange is still 18.2 kilobytes.

For Leech lattice encoding, we examine the parameter set $(2^{11}, 552, 1.10)$. For the 256 bit case, this set utilized five copies of the Leech lattice, each encoding 60 bits. Since we now only require 240 bits, we can reduce the number of Leech lattice copies down to 4. This gives $k = 96$, and therefore $\bar{n} = 10$ and $\bar{m} = 10$. Note that there are some redundant

coordinates here, too (also four). The resulting overall size of the key exchange is reduced to 14.8 kilobytes.

The improvement to the bandwidth by using Leech lattice encoding, compared to integer lattice encoding, is now 18.4%, rather than 10.2%. We have almost doubled this improvement, by adopting a security parameter that is more friendly to the Leech lattice, and a little less friendly to the integer lattice encoding four bits in each coordinate.

These results may vary somewhat given different parameters and error distributions. However, it seems reasonable to suggest that the 256 bit requirement significantly limits the encoding efficiency of the Leech lattice.

Chapter 6

Implementation

This chapter describes the cryptographic implementation of Vardy and Be'ery's maximum likelihood decoding algorithm [VB93], described in Chapter 4. The decoder was implemented in the language `C`, and this chapter is written from this point of view. Higher level languages may introduce a whole new set of considerations. The primary concern of the implementation is the adherence to cryptographic requirements. The speed of the implementation is of secondary concern, but remains important to ensure the practicality of the application of the Leech lattice.

6.1 Requirements

In any cryptographic system, care must be taken not to leak information which can be exploited in a *side-channel attack*. A side-channel attack uses information gained from the physical implementation of a system in order to break it, rather than a weakness in the actual cryptography used. Some examples of side-channels include the timing of computations, error messages thrown, and the power consumption of particular instructions. Most of these side-channels are outside the scope of this thesis, and need to be dealt with on a per-system basis. However, preventing timing and cache attacks is crucial to the implementation of the decoding algorithm. This chapter therefore deals exclusively with these considerations.

It is essential that the Leech lattice decoder decodes in constant time with respect to the input, where constant time refers to the actual execution time of the program (not asymptotic time complexity). Non-constant time decoding could leak information on the values being transferred. This could have devastating effects on the security of the system.

Similarly, it is important that the decoder does not leak information through its use of caches. A second process on the same CPU should not be able to extract information from the decoder's use of the cache.

Making software timing and cache attack resistant is a non-trivial problem, and not solved in general. The multitude of chip architectures and unpredictability of modern compilers makes it very difficult to reliably write portable, cryptographically safe software. With this in mind, there are several issues we need to address in any implementation.

Constant time algorithm. The algorithm itself must terminate in a constant number of steps, regardless of the input. For example, a sorting algorithm like insertion sort applies a different number of operations depending on how well the input was sorted. This leaks information on the array being sorted.

Data-dependant branching. Modern CPUs implement a pipeline architecture. This means that there are multiple sequential instructions being executed simultaneously. In order to realize this, a branch predictor is used. This is a digital circuit that tries to guess which way a branch will go before the actual branch instruction is reached. If this circuit guesses incorrectly, the processor needs to start over with the correct branch, incurring a delay called a *branch misprediction*. This means that even if the exact same number of instructions are executed, branch mispredictions could be detected. If these are dependent on the input to the decoder, the branch predictor can leak information.

Data-dependent memory access. Accessing memory is orders of magnitudes slower than accessing a register, which is why modern CPUs store frequently accessed memory values in a cache, closer to the CPU. If the processor requests to read data that is not in the cache (known as a *cache miss*), it has to wait for the data to be transferred from memory, which is much slower than reading directly from cache. Besides the resulting inconsistent timing of the decoder, this can potentially be exploited in several ways, as described in [CLS06].

Briefly, a potential attack could work as follows. A second process running on the same CPU as the decoder would necessarily share the cache. By measuring the timing of its own memory access calls, the second process can determine when it has experienced a cache miss for certain addresses, and thus determine whether the decoder has tried to access memory previously stored at that cache address. Data dependent memory access can therefore leak information to a second process, in a manner that is more specific and useful than the overall execution time of the algorithm. By implementing memory access patterns that are the same for all inputs, this type of side channel attack can be avoided.

These are the most important considerations. However, it is not an exhaustive list of oversights that could break a crypto implementation on a specific compiler and architecture combination. We make several assumptions that may not be valid for all scenarios.

6.1.1 Assumptions

It is assumed that the addition, subtraction, and multiplication instructions for up to 64 bit integer values are constant time operations. Additionally, we assume that a bitshift operation on a 64 bit integer is constant time, regardless of the size of the shift. These assumptions may not be valid for all computer architectures.

In any case, discrepencies between the timing of individual arithmetic instructions are orders of magnitude smaller than algorithmic considerations, branch mispredictions, and

cache misses. As any cryptographic algorithm will need to deal with these issues, these assumptions are assumed reasonable and will be ignored during the rest of this thesis. The implementation designed in this thesis is intended as a proof of concept, and not for use in any production code.

6.2 Making the algorithm constant-time

The Leech lattice decoder consists of a precomputation step, followed by five different stages.

Precomputation. This step calculates the squared Euclidean distance (SED) between twelve two-dimensional inputs, and the closest $\hat{A}_{ijk}$ in each A_{ijk} subsets. The number of Leech lattice points being decoded (alternatively, the size of the constellation used) determine how large the subset A_{ijk} is, minimally 1. This results in $|A_{ijk}|$ SED calculations and $|A_{ijk}| - 1$ comparisons in order to choose the smallest (for large constellations, this step could be optimized to only consider points within a certain distance of the input). From these SEDs, the d_{ij} and δ_{ij} values are computed. The precomputation is therefore a constant time step.

Computing the Confidence Values, Preferable Representations, and Penalties. This step computes the preferable representation and its confidence value for each $x \in \mathbb{F}_4$ and each coordinate of H_6, along with the three penalties for each differing (h, k)-parity. This is also a constant number of operations.

Sorting the Penalties. This step sorts the 24 penalties for each differing (h, k)-parity. While the sorting could be made constant time by using a special type of sort, the subsequent steps need to choose the best penalty for varying combinations of block coordinate and hexacode character x. Queries into the sorted list of penalties may need to visit a data-dependent number of penalties before finding the best one matching the required coordinate and character. Therefore, the sorting should be done individually for each hexacodeword. This increases the complexity of the algorithm, from 3 sorts of size 24, to 192 sorts of size 6. However, the sorts of 6 do not need to be entirely sorted, as only the lowest 2 values are required. Queries into the sorted lists now need only one operation, instead of potentially many. As we are sorting the same 24 values many different times, it may be possible to optimize this in a constant-time fashion. This is an open question; in the interest of keeping the algorithm simple, partial sorts of size 6 were used in later steps.

Computing the Confidence Values of the Blocks. This step computes the confidence values for each of the three blocks for each hexacodeword. This is a constant number of operations.

Finding the Images of the Hexacodewords. For each of the 64 hexacodewords, this step finds the closest Leech lattice point by resolving the (h, k)-parity of the preferable

representation. The sorting is therefore moved to this step, to find the relevant penalties. After the sorting, there are three possible cases. One is that the parity already matches the coset of Q_{24}, in which no penalties need be applied. The second is that one penalty must be applied, and the third is that two penalties in non-conflicting coordinates must be applied to resolve the parities. To make this step constant time, the penalties are sorted for each parity regardless of the correctness of the parities. The three different cases are calculated according to the parity discrepency, minimized, and the best case is applied. For example, in the case the parities were already correct, all calculated cases are dummies, and there is no effective change.

Computing the metrics of the Hexacodewords and Final Minimization. In this step, the metric of the preferable representation along with any necessary penalties is calculated for each hexacodeword. The minimum of these metrics is chosen as our output, along with the corresponding Leech lattice point. The metrics are calculated in constant time. The Leech lattice point is stored internally using a single 64-bit integer. This 36 bit array is combined with the information gained from the A_{ijk} or B_{ijk} subsets, to output a bit string whose size is dependent on the number of Leech lattice points being used in the encoding. This point can easily be generated and returned in constant time, meaning this last step is constant time.

6.3 Constant-time sorting and minimization

While most of the algorithm is straightforward computation, there are numerous steps which involve sorting and/or minimization of values. These require the comparison of values, and a choice based upon the result of the comparison. The natural implementation is a branch, but this is not allowed due to the requirements stated above. However, there are some tricks we can apply.

Consider the need to find the minimum and maximum of two values, say a and b. Let s be the sign of $a - b$, where $s = 1$ if $a - b$ is negative, and 0 otherwise. For integers, the s can be extracted using a clever bitshift (this is also possible for floating point values, but not required in our implementation). Then the minimum and maximum can be reconstructed using the following formulas.

$$\min = as + b(s \ \texttt{XOR}\ 1),$$
$$\max = bs + a(s \ \texttt{XOR}\ 1).$$

By setting a equal to min, and b equal to max, we obtain a swap function that uses the same amount of instructions regardless of the comparison result. To swap objects, or structures associated with the values a and b, the same formulas can be applied. It is not sufficient to simply swap associated pointers since this will result in inconsistent memory access patterns. By rebuilding values and associated data in this manner, we ensure that the same cache lines are touched for any input.

Armed with a timing-safe swap operation, we can now easily design minimization and sorting algorithms. It is important to apply swaps in a manner that is independent of the values themselves.

Minimization. Minimization is typically done in $n - 1$ comparisons, which is constant time. Our swap operation allows a similar $O(n)$ implementation. We apply `swap` with element $n - 1$ as a and element n as b. Next we apply `swap` to elements $n - 2$ and $n - 1$, and continue until we've done elements 1 and 2. At this point, we are guaranteed to find the minimum value in the first position of the array.

If we are not interested in preserving the integrity of the array, and are only concerned with the minimal value, it is possible to optimize this algorithm. In this case, we only construct min as above, and set a to this value. The value b is left untouched, meaning values in the array may be lost. However, we are guaranteed to find the minimum value at the first index in the array, which is all we required.

Sorting. Typical sorts such as quicksort or mergesort are largely concerned with minimizing the amount of comparisons, and have little to no regard for the movement of the underlying data. Since we must maintain consistent memory access, these sorts are not an option. Instead, a sort with a consistent access pattern must be used. The sort must have a fixed "plan", which does not change depending on comparisons made during the sort. In [Knu98] this is called a *homogeneous* decision structure, and gives sorting networks as the solution. Sorting networks can be constructed using general algorithms giving $O(n \log^2 n)$ complexity.

However, with the previous algorithmic changes to the algorithm, it turns out that sorting networks are not necessary. We only require the smallest two values in an array of size 6, meaning we can simply use an extension of the minimization technique described above. Nevertheless, sorting networks are a powerful technique for constant-time software.

6.4 Implementations

All of the following software was implemented in `C`. The implementation was compiled on a 4.4.9-300.f23.x86_64 linux kernel with GCC 5.3.1, and run on a 2.4GHz Intel Core i3 M370 processor. The maximum likelihood decoder by Vardy and Be'ery [VB93] was implemented twice.

The first implementation is a cryptographically safe version, conforming to the requirements laid out in Section 6.1. It also makes use of the algorithmic adjustments described in Section 6.2. Using the patterns defined in the previous section, the algorithm was implemented without any input-dependent branches or memory access. Unless the compiler optimizes these in, the implementation should therefore resist timing and cache attacks.

The second implementation of Vardy and Be'ery's decoder is an unsafe version. This implementation follows the algorithm as described in the paper, and makes no timing considerations. The sorting of the penalties is done using an insertion sort. While there are

asymptotically faster sorts available, insertion sort is typically used as a subroutine for more complex sorts. It has a very simple implementation with minimal overhead. Since the sorts are of small size ($n = 24$), it is assumed the use of insertion sort is close to optimal.

6.5 Correctness

In this section we review several tests that give us confidence that the implementation is correct and conforms to the requirements laid out in Section 6.1.

The correctness of a CVP algorithm is difficult to verify, without another algorithm that is known to be correct. The timing safety of an implementation is also difficult to establish. However, there are several tests we can run.

Error-correction radius. This test generates random Leech lattice points, by generating random bits and constructing a valid 2×6 array of A_{ijk} or B_{ijk} points (see Chapter 4). This points are converted to vectors using a lookup table. For each Leech lattice point, a random error within the error-correction radius of the Leech lattice ($\lambda_1(\Lambda_{24})/2$) is added. The resulting point is decoded, and verified against the starting point. The outputted distance is also verified, between the starting point plus error and the outputted closest vector. Then a different random error is added to the starting point, and the process starts anew.

Both the safe and unsafe versions of the decoder decoded hundreds of thousands of these lattice points plus error, with a 100% success rate. This gives us confidence that both decoder implementations are correct, at least within the error-correction radius of Λ_{24}.

Consistency. This test generates uniformly random points in the space spanned by the Leech lattice. For each generated point, both the safe and unsafe decoders are applied, and the points are compared. This test was run on hundreds of thousands of random points. For a very small percentage, the two decoders returned different closest vectors. However, the distance between the random point and each of the two returned closest vectors was equal in every such case, indicating the random point happened to lie exactly in between two (or possibly more) Leech lattice points.

This means the two decoders are consistent when decoding points in all space, excepting the boundaries of the Voronoi cells. This is compatible with the definition of CVP.

Timing safety. Like the previous test, we generate a set of uniformly random points in the space spanned by the Leech lattice. For each of these points, the decoder is applied 10000 times. These decoder executions are interleaved, meaning the decoder cycles through the set of random points 10000 times. The start and end times for each decoder execution are measured, and the times for each random point are summed and averaged.

For the timing safe implementation of the decoder, the average time of execution for each uniformly random point was 55.4 ± 0.1 microseconds (note that the resolution of the timing mechanism used was limited to microseconds).

For the unsafe implementation, the average time of execution for each random point was 41.2 ± 2.3 microseconds.

Exactly measuring execution time is difficult, as the program has no control over its scheduling on the CPU. The system scheduler may choose to interrupt the program at any time to run another task. By decoding each random point a very large number of times, and interleaving these executions, we can average out the effects of the scheduler on our timing. If there are data dependent timing issues with the implementation, these should become visible.

In the unsafe implementation, this is indeed very visible. The average times varied over a range of well over 10% of the average execution time. This could be due to coincidentally presorted penalties, or a lack of penalties to apply for a large number of hexacodewords. In any case, there is clearly a distinction to be made between certain points. By contrast, the safe implementation shows no such variance.

As a result, we have some confidence that this implementation will not leak information to an attacker.

6.6 Performance

As mentioned previously, the cryptographically safe implementation runs in an average of 55.4 microseconds, while the unsafe implementation runs in an average of 41.2 microseconds. The timing considerations therefore add 34.5% of overhead to the algorithm. Significant overhead is to be expected for any cryptographic software, so this is not out of the ordinary.

At the very least, we expect a timing safe implementation of an algorithm to have a worst-case running time. In order to avoid data dependent cache misses and branch mispredictions, a lot more data needs to be examined and moved in memory than normal. With these considerations, 34.5% seems very reasonable, especially considering that the algorithm, as described by Vardy and Be'ery, has a worst case complexity 21.7% more than its average case complexity.

The algorithm was described as requiring 3595 "real" operations in the worst case, and 2955 on average. They describe real operations as including addition, subtraction, or comparisons, and specifically ignore memory addressing, negation, and absolute value calls.

The unsafe implementation, averaging 41.2 microseconds, takes almost 100,000 clock cycles to complete. While this implementation can probably be optimized to some extent, it does not account for this massive difference in running times. Clearly, the complexity measures used in [VB93] and others are far from realistic for CPUs. We note nevertheless that the cost model of [VB93] could very well be realistic for dedicated circuits.

Further optimizations. The cryptographically secure implementation has been optimized to a large extent. There may be small gains to be made in certain steps of the algorithm. Larger gains might be possible by implementing hardware specific optimizations, such as fast integer types and the use of subword parallel instruction set architectures.

The greatest gains, however, can be made by parallelizing the implementation. The decoder is highly parallelizable, and also vectorizable. Trivially, the four decoders for the cosets of Q_{24} can run concurrently. Using SIMD (single instruction, multiple data), these decoders could also easily be vectorized. In most cases, this will be sufficient, and will cleanly decrease the running time by a factor of four. It is also easily possible to parallelize the algorithm to an even greater degree.

The bulk of the algorithm is spent minimizing the penalties, calculating metrics and applying the relevant penalties for each hexacodeword. There are 64 hexacodewords, and these could all be implemented in a vector, if supported by the hardware. More probably, the hexacodewords would be split into several groups, each run on a separate vector. The information associated with a penalty (or any other arithmetic in the decoder) could be reduced to 64 bits, or even 32 bits, if q is sufficiently small. Assuming 256 bit vectorization, which has been widely available for some time, it could be possible to decrease the most computationally intensive steps of the decoding algorithm by a factor of 4, or possibly even 8. With 512 bit vectorization not far away, this could be doubled again.

The alterations made to the algorithm to make it timing safe (particularly, the penalty sorting step) are very useful for vectorization. Since the computations done for each hexacodeword are not dependent on data used for other codewords, vectorization is exceedingly simple.

Suitability for LWE. In [BCD+16], the authors state the running times of the key exchange to be in the order of 1.3 milliseconds for both parties. For [ADPS15], the running times are nearly 10 times as fast, at approximately 150 microseconds per party.

While the best parameter set in Chapter 5 requires 8 copies of the Leech lattice, and thus 8 decoders, a slightly suboptimal set needs only 2. Assuming the implementation has been vectorized using 256 bit SIMD, a single decoder would run in under 14 microseconds, or 7 for small enough q. While this is significant compared to the running time of R-LWE (NewHope), it is negligible compared to LWE (Frodo). The running times for 3072 bit RSA and elliptic curve Diffie-Hellman using the same hardware are approximately 4.6 and 1.4 milliseconds respectively, indicating that it is much faster than conventional crypto. Additionally, these comparisons does not even take into account the difference in hardware, as the CPU yielding a running time of 55 microseconds for the decoder is vastly inferior compared to the hardware used in [BCD+16].

In summary, the decoder of Vardy and Be'ery appears to be suitable for cryptographic applications, provided the implemention takes advantage of vectorization, and the number of decoders required is small.

Chapter 7

Conclusion

Lattice based asymmetric primitives have many properties that make them attractive choices for post-quantum security. However, while the key exchanges based on LWE and R-LWE are computationally efficient, they still lag behind in terms of the amount of data exchanged. This thesis examined the possible application of the Leech lattice in order to reduce the bandwidth of such a scheme.

7.1 Results

We first examined the properties of the Leech lattice that make it suitable for coding applications. We investigated the maximum likelihood decoding algorithm for the Leech lattice by Vardy and Be'ery, and generalized several concepts used by this algorithm to lattice decoding in general. We took these complex techniques from an engineer's perspective, and placed them in the framework of lattice and glue theory.

We applied Leech lattice encoding to an encryption scheme based on the LWE problem, and compared it to the integer encoding used in existing schemes. While we found marginal improvements, in the order of 17% for the encryption scheme, and 10% for the key exchange, the true encoding efficiency of the Leech lattice was stunted by the cumbersome nature of its dimension, and the resulting densities. When we are required to exchange 256 bits of key material, the Leech lattice simply doesn't fit nicely, and a significant amount of coding gain is lost. As noted in Section 5.6.3, the efficiency of the key exchange could be improved by more than 18% if we only require 240 bits of key material.

In any case, we confirmed that the Leech lattice is better at tolerating larger errors than the more simple integer lattice encoding. In the context of LWE, this can allow for a larger reduction in q in some scenarios, when compared to the use of integer lattice encoding, all the while maintaining the same security estimates.

On the implementation side of things, we found that the decoding algorithm can indeed be made timing and cache attack safe, with several algorithmic adjustments. We implementated a proof of concept in `C`, and demonstrated the desired effects of the changes made in order to make the implementation cryptographically secure. Lastly, we also showed that the algorithm

can be made fast enough to be practical for use in the real world. With some parallelization and/or vectorization, the algorithm would have only a very small effect on the latency of a handshake, as compared to the computations involved in LWE. For R-LWE, the impact is more noticeable, but still manageable.

7.2 Further research

While the Leech lattice failed to make very significant improvements to the bandwidth of existing LWE schemes, we did demonstrate that improved coding is possible. We may be able to apply other efficient lattice codes with more agreeable dimensions, in order to realize a practically useful reduction in bandwidth.

It is also worth looking at whether the efficient coding properties of the Leech lattice can be used in an R-LWE scheme. The Leech lattice is an ideal in the cyclotomic ring $\mathbb{Z}[\zeta_{35}]$, which could be used in R-LWE. In this context, the dimensions of the Leech lattice may be a more natural fit. If the maximum coding gain of the Leech lattice can be realized, the reduction in bandwidth could be significant.

Bibliography

[AB96] Ofer Amrani and Yair Be'ery. Efficient bounded-distance decoding of the hexacode and associated decoders for the leech lattice and the golay code. *Communications, IEEE Transactions on*, 44(5):534–537, 1996.

[ABB+15] Daniel Augot, Lejla Batina, Daniel J Bernstein, Joppe Bos, Johannes Buchmann, Wouter Castryck, O Dunkelmann, Tim Güneysu, Shay Gueron, Andreas Hülsing, T Lange, MSE Mohamed, C Rechberger, P Schwabe, N Sendrier, F Vercauteren, and BY Yang. Initial recommendations of long-term secure post-quantum systems (2015), 2015. `https://pqcrypto.eu.org/docs/initial-recommendations.pdf`.

[ABV+94] Ofer Amrani, Yair Be'ery, Alexander Vardy, Feng Wen Sun, and Henk CA Van Tilborg. The leech lattice and the golay code: bounded-distance decoding and multilevel constructions. *Information Theory, IEEE Transactions on*, 40(4):1030–1043, 1994.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *Advances in Cryptology-CRYPTO 2009*, pages 595–618. Springer, 2009.

[ADPS15] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - a new hope. Technical report, Cryptology ePrint Archive, 2015. `http://eprint.iacr.org/2015/1092`.

[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. ACM, 1996.

[BBBV97] Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997.

[BCD+16] Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. Technical report, Cryptology ePinrt Archive, 2016. `http://eprint.iacr.org/2016/659`.

[BCNS15] Joppe W Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the tls protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy*, pages 553–570. IEEE, 2015.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 575–584. ACM, 2013.

[BS81] Eiichi Bannai and Neil JA Sloane. Uniqueness of certain spherical codes. *Canad. J. Math*, 33(2):437–449, 1981.

[BSS89] Yair Be'ery, Boaz Shahar, and Jakov Snyders. Fast decoding of the leech lattice. *IEEE Journal on Selected Areas in Communications*, 7(6):959–967, 1989.

[Ces15] Chris Cesare. Online security braces for quantum revolution. *Nature*, 525:167–168, 2015. `http://www.nature.com/news/online-security-braces-for-quantum-revolution-1.18332`.

[CJL+16] Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. Report on post-quantum cryptography. *National Institute of Standards and Technology Internal Report*, 8105, 2016.

[CLS06] Anne Canteaut, Cedric Lauradoux, and Andre Seznec. *Understanding cache attacks*. PhD thesis, INRIA, 2006.

[CN11] Yuanmi Chen and Phong Q Nguyen. Bkz 2.0: Better lattice security estimates. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 1–20. Springer, 2011.

[Con68] John H Conway. A perfect group of order 8,315,553,613,086,720,000 and the sporadic simple groups. *Proceedings of the National Academy of Sciences*, 61(2):398–400, 1968.

[CPS82] JH Conway, RA Parker, and NJA Sloane. The covering radius of the leech lattice. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 380, pages 261–290. The Royal Society, 1982.

[CS82a] John H Conway and NJA Sloane. Fast quantizing and decoding and algorithms for lattice quantizers and codes. *Information Theory, IEEE Transactions on*, 28(2):227–232, 1982. `http://neilsloane.com/doc/Me83.pdf`.

[CS82b] John H Conway and NJA Sloane. Twenty-three constructions for the leech lattice. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 381, pages 275–283. The Royal Society, 1982.

[CS82c] John H Conway and NJA Sloane. Voronoi regions of lattices, second moments of polytopes, and quantization. *Information Theory, IEEE Transactions on*, 28(2):211–226, 1982.

[CS84] John H Conway and NJA Sloane. On the voronoi regions of certain lattices. *SIAM Journal on Algebraic Discrete Methods*, 5(3):294–305, 1984.

[CS86] John H Conway and NJA Sloane. Soft decoding techniques for codes and lattices, including the golay code and the leech lattice. *Information Theory, IEEE Transactions on*, 32(1):41 – 50, 1986.

[CS93] John H Conway and NJA Sloane. *Sphere packings, lattices, and groups*. Springer-Verlag, New York, 1993.

[Dir15] Information Assurance Directorate. Commercial national security algorithm suite, 2015. `https://www.iad.gov/iad/programs/iad-initiatives/cnsa-suite.cfm`.

[DXL12] Jintai Ding, Xiang Xie, and Xiaodong Lin. A simple provably secure key exchange scheme based on the learning with errors problem. *IACR Cryptology ePrint Archive*, 2012:688, 2012.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[Gol49] Marcel JE Golay. Notes on digital coding, 1949.

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[Hal05] Thomas C Hales. A proof of the kepler conjecture. *Annals of mathematics*, 162(3):1065–1185, 2005.

[Knu98] Donald Ervin Knuth. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1998.

[Lee64] John Leech. Some sphere packings in higher space. *Canad. J. Math*, 16:657–682, 1964.

[Lee67] John Leech. Notes on sphere packings. *Canad. J. Math*, 19(251):267, 1967.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for lwe-based encryption. In *Cryptographers' Track at the RSA Conference*, pages 319–339. Springer, 2011.

[LS71] John Leech and NJA Sloane. Sphere packings and error-correcting codes. *Canad. J. Math*, 23(4):718–745, 1971.

[MCSQ08] Robby G McKilliam, I Vaughan L Clarkson, Warren D Smith, and Barry G Quinn. A linear-time nearest point algorithm for the lattice A_n^*. In *Information Theory and Its Applications, 2008. ISITA 2008. International Symposium on*, pages 1–5. IEEE, 2008. `https://www.itr.unisa.edu.au/itrusers/mckillrg/public_html/papers/ISITA_Anstar.pdf`.

[MR09] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In *Post-quantum cryptography*, pages 147–191. Springer, 2009.

[Mun57] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[Nie73] Hans-Volker Niemeier. Definite quadratische formen der dimension 24 und diskriminante 1. *Journal of Number Theory*, 5(2):142–178, 1973.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 333–342. ACM, 2009.

[Pei14] Chris Peikert. Lattice cryptography for the internet. In *International Workshop on Post-Quantum Cryptography*, pages 197–219. Springer, 2014.

[PZ03] John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *arXiv preprint quant-ph/0301141*, 2003.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[SE94] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, 1994.

[Sho94] Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE, 1994.

[Slo77] NJA Sloane. Binary codes, lattices and sphere packings. *Combinatorial surveys*, pages 117–164, 1977.

[Var95] Alexander Vardy. Even more efficient bounded-distance decoding of the hexacode, the golay code, and the leech lattice. *Information Theory, IEEE Transactions on*, 41(5):1495–1499, 1995.

[VB93] Alexander Vardy and Yair Be'ery. Maximum likelihood decoding of the leech lattice. *Information Theory, IEEE Transactions on*, 39(4):1435–1444, 1993.

[ZZD+15] Jiang Zhang, Zhenfeng Zhang, Jintai Ding, Michael Snook, and Özgür Dagdelen. Authenticated key exchange from ideal lattices. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 719–751. Springer, 2015.