A Gentle Introduction to Lattice-Based Cryptography

Alfred Menezes
University of Waterloo
Email: ajmeneze@uwaterloo.ca
URL: https://cryptography101.ca

Version 1.1: June 9, 2026

Abstract

We present the quantum-safe Kyber key encapsulation mechanism (ML-KEM) and the Dilithium signature scheme (ML-DSA). We also develop the mathematical background on lattices needed to understand why Kyber and Dilithium are regarded as lattice-based cryptosystems, and we provide insight into the computational hardness of the underlying lattice problems. The exposition is intended to be accessible to senior undergraduate students and beginning graduate students, and closely follows the accompanying video lecture series *Kyber and Dilithium*, *The Mathematics of Lattice-Based Cryptography*, and *The LLL Algorithm*, which are available at the website cryptography101.ca.

Contents

Revision history

- Version 1.0: May 26, 2026.
- Version 1.1: June 9, 2026.
  - Added Example 2.13 (Leech lattice).
  - Added Example 3.5 (ISIS instance).
  - Added Example 4.7 (ss-LWE instance).
  - Added Example 4.13 (Lindner-Peikert encryption).
  - Added §9.5 on sieving.
  - Added answers to computational exercises (§13).

1 Introduction

In August 2024, the U.S. government's National Institute of Standards and Technology (NIST) published a suite of standards for a key encapsulation mechanism and for digital signature schemes designed to be resistant to attacks by quantum computers. These quantum-safe schemes are intended to replace widely deployed RSA and elliptic curve cryptography (ECC), both of which are vulnerable to quantum computing attacks. Although the timeline for the development of cryptographically relevant quantum computers remains uncertain, efforts to transition from RSA and ECC to quantum-safe alternatives are rapidly gaining momentum.

Among the quantum-safe cryptographic schemes standardized by NIST, the key encapsulation mechanism Kyber and the digital signature scheme Dilithium are expected to see the broadest deployment in the near future. The security of Kyber relies on the presumed hardness of the Module Learning With Errors (MLWE) problem, whereas the security of Dilithium is based on both the MLWE problem and the Module Short Integer Solution (MSIS) problem. These problems are the module variants of the well-studied Learning With Errors (LWE) and Short Integer Solution (SIS) problems, respectively.

These notes provide detailed descriptions and analyses of Kyber and Dilithium. We explain why the SIS and LWE problems—and consequently their module variants MSIS and MLWE—can be viewed as computational problems on lattices. This will justify classifying Kyber and Dilithium as *lattice-based cryptosystems*.

The remainder of this document is organized as follows. In §2, we give an introduction to the theory of lattices and define fundamental computational lattice problems: the Shortest Vector Problem (SVP), the Shortest Independent Vectors Problem (SIVP), the Closest Vector Problem (CVP), and their approximate versions, approx-SVP, approx-SIVP, and approx-CVP. In §3 and §4, we show how SIS and LWE can be formulated as lattice problems, and explain their connections to SVP and SIVP. The module variants of SIS and LWE are introduced in §5, while detailed descriptions of Kyber and Dilithium are provided in §6 and §7. The lattice problems SIS, LWE, MSIS, and MLWE can be tackled using the LLL and BKZ lattice basis reduction algorithms described in §8 and §9. The performance of LLL and BKZ plays a central role in assessing the hardness of the lattice problems and in motivating the parameter choices used in Kyber and Dilithium. Six applications of the LLL algorithm are presented in §10: solving short-secret LWE, solving SIS, finding integer relations, performing simultaneous Diophantine approximation, mounting a side-channel attack on the ECDSA signature scheme, and finding small roots of univariate polynomials modulo a composite integer with unknown factorization. In §11, we describe the Number-Theoretic Transform (NTT), a fast algorithm for polynomial multiplication that is essential for efficient implementation of Kyber and Dilithium. Additional remarks and pointers to the literature are given in §12.

2 Lattices

Contents

2.1 Basic definitions

The sets of natural numbers, integers, rational numbers, and real numbers are denoted by $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$, respectively. We write $\mathbb{R}^n$ for the n-dimensional real vector space consisting of all length-n vectors $x = (x_1, x_2, \ldots, x_n)$ with $x_i \in \mathbb{R}$. The *length* of such a vector x is $\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$, which is also denoted $\|x\|_2$ and called the *Euclidean length*, *Euclidean norm*, or ℓ_2*-norm*. The *infinity norm* of x is $\|x\|_\infty = \max_i |x_i|$. The set $\mathbb{Z}^n$ denotes the subset of $\mathbb{R}^n$ consisting of all length-n vectors whose components are integers.

Informally, a lattice can be viewed as a regular, repeating arrangement of points in real space. Formally, lattices are defined as follows.

Definition 2.1 A *lattice* L in $\mathbb{R}^n$ is the set of all integer linear combinations of m linearly independent vectors $b_1, b_2, \ldots, b_m \in \mathbb{R}^n$, where $m \le n$. The (ordered) set $B = [b_1, b_2, \ldots, b_m]$ is called a *basis* of L and we write $L = L(B)$. The *dimension* of L is n, and the *rank* of L is m.

The lattice L consists of all vectors of the form

$$v = c_1 b_1 + c_2 b_2 + \cdots + c_m b_m,$$

where the scalars c_i are integers. The basis B can be represented by an $n \times m$ *basis matrix*, also denoted by B, whose columns are the basis vectors $b_1, b_2, \ldots, b_m$:

$$B = \begin{bmatrix} | & | & \cdots & | \\ b_1 & b_2 & \cdots & b_m \\ | & | & \cdots & | \end{bmatrix}.$$

With this notation, the lattice can be described compactly as $L = \{Bx : x \in \mathbb{Z}^m\}$.

An equivalent characterization of lattices is given by the following definition.

Definition 2.2 A *lattice* L is a discrete additive subgroup of $\mathbb{R}^n$.

Here, L being an *additive subgroup* of $\mathbb{R}^n$ means that L is a non-empty subset of $\mathbb{R}^n$ that is closed under vector addition and subtraction; that is, $x + y, -x \in L$ for all $x, y \in L$.

The term *discrete* means that there exists an $\varepsilon > 0$ such that, for every $x \in L$, there are no other lattice points within distance ε of x. We omit the proof that Definitions 2.1 and 2.2 are equivalent.

In cryptographic applications, one typically works with lattices whose basis vectors have integer coordinates, that is, $b_1, b_2, \ldots, b_m \in \mathbb{Z}^n$. In this case, $L \subseteq \mathbb{Z}^n$ and L is called an *integer lattice*. The lattices considered in the remainder of this document will primarily be integer lattices.

Definition 2.3 A *full-rank lattice* in $\mathbb{R}^n$ is a lattice of rank n.

If L is a full-rank lattice in $\mathbb{R}^n$ with basis $B = [b_1, b_2, \ldots, b_n]$, then B is also a basis of the vector space $\mathbb{R}^n$. The lattice L consists of all *integer* linear combinations of $b_1, b_2, \ldots, b_n$, whereas the vector space $\mathbb{R}^n$ consists of all *real* linear combinations of these same vectors.

Definition 2.4 Let L and L' be lattices in $\mathbb{R}^n$. Then L' is a *sublattice* of L if $L' \subseteq L$.

To show that L' is a sublattice of L, it suffices to verify that each vector b' in a basis B' of L' is an element of L, that is, can be expressed as an integer linear combination of the vectors in a basis B of L.

Henceforth, unless explicitly stated otherwise, all lattices and sublattices will be assumed to be full-rank integer lattices.

2.2 Examples

The most basic example of a full-rank integer lattice is $\mathbb{Z}^n$, whose basis matrix is the identity matrix $B = I_n$. We next give several examples of full-rank integer lattices in two dimensions.

Example 2.5 (*two-dimensional lattice*) Consider the basis $B_1 = [(1, 0), (0, 1)]$. The lattice generated by B_1 is

$$L_1 = L(B_1) = \{B_1 x \ : \ x \in \mathbb{Z}^2\},$$

where

$$B_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Thus, $L_1 = \mathbb{Z}^2$ is simply the integer grid in the plane, as illustrated in Figure 2.1(a).

Definition 2.6 Let $L = L(B)$ be a full-rank lattice in $\mathbb{R}^n$ with basis $B = [b_1, b_2, \ldots, b_n]$. The *fundamental parallelepiped* of L (with respect to the basis B) is defined as

$$P(B) = \{a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \ : \ a_i \in [0, 1)\}.$$

Equivalently, $P(B) = \{Bx \ : \ x \in [0, 1)^n\}$.

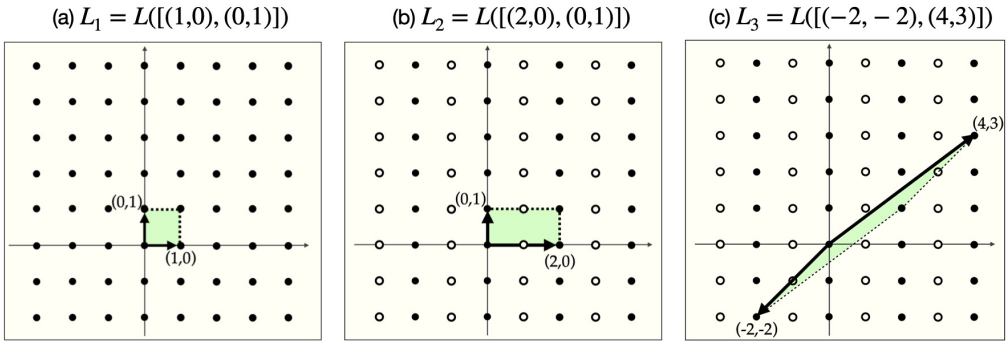


Figure 2.1: Three full-rank integer lattices in $\mathbb{R}^2$ and their fundamental parallelepipeds.

The fundamental parallelepiped $P(B)$ partitions $\mathbb{R}^n$ into non-overlapping regions known as *parallelepipeds* (Exercise 2.6). The "corners" of these parallelepipeds are precisely the lattice points of $L(B)$. Three examples of fundamental parallelepipeds are shown in Figure 2.1.

Example 2.7 (*two-dimensional lattice*) Consider the basis $B_2 = [(2,0),(0,1)]$. The lattice generated by B_2 is

$$L_2 = L(B_2) = \{B_2 x \ : \ x \in \mathbb{Z}^2\},$$

where

$$B_2 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}.$$

The lattice L_2 and its fundamental parallelepiped $P(B_2)$ are shown in Figure 2.1(b).

Observe that L_2 is a sublattice of L_1. To see that $L_2 \neq L_1$, it suffices to exhibit one vector in L_1 that does not belong to L_2. For instance, $(1,0) \in L_1$ but

$$(1,0) = \frac{1}{2} \cdot (2,0) + 0 \cdot (0,1),$$

which is not an integer linear combination of the basis vectors of B_2. Hence, $(1,0) \notin L_2$.

Example 2.8 (*two-dimensional lattice*) Consider the basis $B_3 = [(-2,-2),(4,3)]$. The lattice generated by B_3 is

$$L_3 = L(B_3) = \{B_3 x \ : \ x \in \mathbb{Z}^2\},$$

where

$$B_3 = \begin{bmatrix} -2 & 4 \\ -2 & 3 \end{bmatrix}.$$

The lattice L_3 and its fundamental parallelepiped $P(B_3)$ are depicted in Figure 2.1(c).

Figures 2.1(b) and 2.1(c) suggest that L_2 and L_3 are in fact the same lattice. This can be confirmed by verifying that each lattice is a sublattice of the other. Indeed, $L_2 \subseteq L_3$ since

$$(2,0) = 3 \cdot (-2,-2) + 2 \cdot (4,3) \quad \text{and} \quad (0,1) = -2 \cdot (-2,-2) - 1 \cdot (4,3),$$

and $L_3 \subseteq L_2$ since

$$(-2,-2) = -1 \cdot (2,0) - 2 \cdot (0,1) \quad \text{and} \quad (4,3) = 2 \cdot (2,0) + 3 \cdot (0,1).$$

The basis $B_2 = [(2,0),(0,1)]$ for lattice L_2 is intuitively "nicer" than the basis $B_3 = [(-2,-2),(4,3)]$, since the vectors in B_2 are shorter and mutually orthogonal. In general, the difficulty of computational problems on lattices depends heavily on the quality of the basis used to represent the lattice. For example, consider the Shortest Vector Problem (SVP), which asks for a shortest nonzero vector in a lattice, given a basis for the lattice. Examining the basis B_2 immediately reveals that $(0,1)$ is a shortest nonzero vector of L_2. In contrast, if L_2 were presented using the basis B_3, then identifying such a vector would be significantly less straightforward.

A common approach to solving lattice problems is therefore to first find a "good" basis for the lattice in question, meaning one whose vectors are relatively short and nearly orthogonal. The most prominent algorithm for this task is the LLL algorithm, which is described in §8.

2.3 Bases of a lattice

Let $L = L(B)$ be an n-dimensional lattice in $\mathbb{R}^n$ with ordered basis $B = [b_1, b_2, \ldots, b_n]$. The following operations transform B into another basis for the same lattice:

(i) Exchange two basis vectors: $b_i \leftrightarrow b_j$.

(ii) Replace a basis vector by its negation: $b_i \leftarrow -b_i$.

(iii) Add an integer multiple of one basis vector to another: $b_i \leftarrow b_i + cb_j$ where $c \in \mathbb{Z}$.

Each of these corresponds to an elementary column operation on the basis matrix B. The first two operations change the sign of the determinant of B, while the third operation leaves the determinant unchanged.

By repeatedly applying these operations, one obtains infinitely many distinct bases for the same lattice. The following theorem characterizes all bases of a lattice.

Theorem 2.9 (*characterization of lattice bases*) Let $L = L(B_1)$ be an n-dimensional lattice. An $n \times n$ matrix B_2 is also a basis for L if and only if $B_2 = B_1 U$, where U is an $n \times n$ matrix with integer entries and determinant ± 1. (Such a matrix U is called *unimodular*.)

Proof: ($\Rightarrow$) Suppose that B_1 and B_2 are both bases of the lattice L. Since each is a basis of the vector space $\mathbb{R}^n$, there exists an invertible matrix $U \in \mathbb{R}^{n\times n}$ such that $B_2 = B_1U$. Because the columns of B_2 belong to L, the entries in U must be integers, and hence $U \in \mathbb{Z}^{n\times n}$. Similarly, there exists an invertible matrix $V \in \mathbb{Z}^{n\times n}$ such that $B_1 = B_2V$.

Substituting yields $B_1 = B_2V = (B_1U)V = B_1(UV)$. Since B_1 is invertible, it follows that $UV = I_n$. Taking determinants, we obtain $\det(UV) = \det(U)\det(V) = 1$, which implies that $\det(U) = \det(V) = 1$ or $\det(U) = \det(V) = -1$. Hence, U is unimodular and $B_2 = B_1U$.

($\Leftarrow$) Conversely, suppose that $B_2 = B_1U$ for some unimodular matrix U. Then $L(B_2) \subseteq L(B_1)$. Recall the inverse formula $U^{-1} = \operatorname{adj}(U)/\det(U)$. Since the adjugate $\operatorname{adj}(U)$ has integer entries and $\det(U) = \pm 1$, it follows that U^{-1} is also unimodular. Thus $B_1 = B_2U^{-1}$ implies that $L(B_1) \subseteq L(B_2)$. Therefore, $L(B_1) = L(B_2)$. □

Example 2.10 (*two bases of a lattice*) As we saw in Examples 2.7 and 2.8, the matrices

$$B_2 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad B_3 = \begin{bmatrix} -2 & 4 \\ -2 & 3 \end{bmatrix}$$

are both bases for the same lattice in $\mathbb{R}^2$. Indeed, one verifies that $B_2 = B_3U$, where

$$U = \begin{bmatrix} 3 & -2 \\ 2 & -1 \end{bmatrix}$$

is unimodular.

The volume[1] of a lattice provides a quantitative measure of how densely the lattice points are distributed in space: larger volumes correspond to sparser lattices. If L is two-dimensional, then its volume coincides with the area of any of its parallelepipeds.

Definition 2.11 The *volume* of a lattice $L = L(B)$ is defined as $\operatorname{vol}(L) = |\det(B)|$.

If B' is any other basis of $L = L(B)$, then $B' = BU$ for some unimodular matrix U. Consequently, $\det(B') = \det(B)\det(U)$, and hence $\det(B') = \pm\det(B)$. It follows that $|\det(B')| = |\det(B)|$. This shows that the volume is a *lattice invariant*, meaning that it is independent of the particular basis chosen to represent the lattice.

2.4 Successive minima

The successive minima are important invariants of a lattice.

Definition 2.12 Let $L \subseteq \mathbb{R}^n$ be a lattice. For each $i \in [1, n]$, the *ith successive minimum* $\lambda_i(L)$ is defined as the smallest real number r such that L contains i linearly independent vectors, each of length at most r.

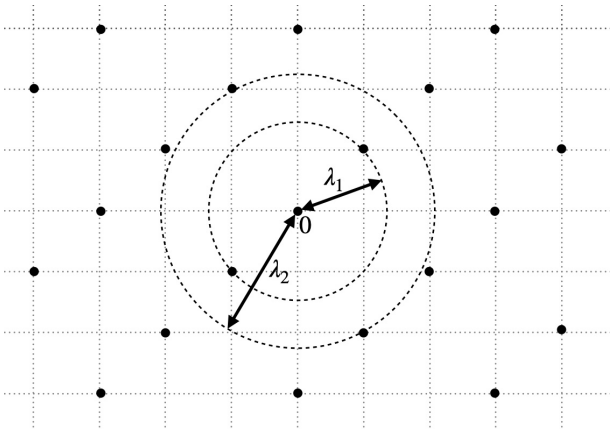


Figure 2.2: Successive minima.

Figure 2.2 shows the first two successive minima for a two-dimensional lattice. By definition, the successive minima form a non-decreasing sequence:

$$\lambda_1(L) \le \lambda_2(L) \le \cdots \le \lambda_n(L). \tag{1}$$

The first successive minimum, $\lambda_1(L)$, is simply the length of a shortest nonzero vector in L. Such a vector is not unique; for example, if $x \in L$ has length $\lambda_1(L)$, then so does $-x$.

Example 2.13 The *Leech lattice* Λ_{24} is a full-rank lattice in $\mathbb{R}^{24}$ with basis matrix shown in Figure 2.3. The length of a shortest nonzero vector in Λ_{24} is $\lambda_1(\Lambda_{24}) = 2$. In fact, there are 196,560 vectors of length 2 in Λ_{24}. This shows that a lattice can have a large number of shortest nonzero vectors.

Finding a shortest nonzero vector—or even determining its length—is a central problem in algorithmic number theory and lattice-based cryptography. In 1850, Hermite proved an exponential upper bound on the first successive minimum.

Definition 2.14 For each integer $n \ge 1$, the *Hermite constant*[2] γ_n is defined by

$$\gamma_n = \max_L \left(\frac{\lambda_1(L)}{\mathrm{vol}(L)^{1/n}} \right)^2, \tag{2}$$

where the maximum is taken over all full-rank lattices $L \subseteq \mathbb{R}^n$.

It follows immediately from (2) that every full-rank lattice $L \subseteq \mathbb{R}^n$ satisfies

$$\lambda_1(L) \le \sqrt{\gamma_n}\ \mathrm{vol}(L)^{1/n}. \tag{3}$$

[1]Formally, the volume of a measurable set $A \subseteq \mathbb{R}^n$ is defined as its Lebesgue measure, given by the integral $\int_A 1\,dx$. The volume of a lattice is defined to be the volume of its fundamental parallelepiped.

[2]γ_n is a constant in the sense that its value depends only on the dimension n, and not on any particular n-dimensional lattice.

$$B = \frac{1}{\sqrt{8}} \left[\begin{array}{cccc|cccc|cccc|cccc|cccc|cccc} 8&4&4&4&4&4&4&2&4&4&4&2&4&2&2&2&4&2&2&2&0&0&0&-3 \\ 0&4&0&0&0&0&0&2&0&0&0&2&0&2&0&0&0&0&0&2&2&0&0&1 \\ 0&0&4&0&0&0&0&2&0&0&0&2&0&0&2&0&0&2&0&0&2&0&0&1 \\ 0&0&0&4&0&0&0&2&0&0&0&2&0&0&0&2&0&0&2&0&2&0&0&1 \\ \hline 0&0&0&0&4&0&0&2&0&0&0&0&0&2&2&2&0&2&2&2&2&0&0&1 \\ 0&0&0&0&0&4&0&2&0&0&0&0&0&2&0&0&0&0&2&0&0&0&0&1 \\ 0&0&0&0&0&0&4&2&0&0&0&0&0&0&2&0&0&0&0&2&0&0&0&1 \\ 0&0&0&0&0&0&0&2&0&0&0&0&0&0&0&2&0&2&0&0&0&0&0&1 \\ \hline 0&0&0&0&0&0&0&0&4&0&0&2&0&2&2&2&0&2&2&2&2&2&2&1 \\ 0&0&0&0&0&0&0&0&0&4&0&2&0&2&0&0&0&2&0&0&0&2&0&1 \\ 0&0&0&0&0&0&0&0&0&0&4&2&0&0&2&0&0&0&2&0&0&0&2&1 \\ 0&0&0&0&0&0&0&0&0&0&0&2&0&0&0&2&0&0&0&2&0&0&0&1 \\ \hline 0&0&0&0&0&0&0&0&0&0&0&0&4&2&2&2&0&0&0&0&2&2&2&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&2&0&0&0&0&0&0&0&2&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&0&0&0&0&0&0&0&2&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&0&0&0&0&0&0&0&1 \\ \hline 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&4&2&2&2&2&2&2&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&0&0&0&2&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&0&0&0&2&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&0&0&0&1 \\ \hline 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&2&2&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \end{array}\right].$$

Figure 2.3: A basis matrix for the Leech lattice $\Lambda_{24} \subseteq \mathbb{R}^{24}$ (Example 2.13). The basis vectors are the columns of the matrix.

Hermite proved that

$$\gamma_n \leq \left(\frac{4}{3}\right)^{(n-1)/2}, \tag{4}$$

which yields an upper bound on $\lambda_1(L)$:

$$\lambda_1(L) \leq \left(\frac{4}{3}\right)^{(n-1)/4} \mathrm{vol}(L)^{1/n}. \tag{5}$$

In 1896, Minkowski gave a much improved upper bound on the first successive minimum.

Theorem 2.15 (*Minkowski's Theorem*) Let $L \subseteq \mathbb{R}^n$ be a full-rank lattice. Then

$$\lambda_1(L) \leq \sqrt{n}\ \mathrm{vol}(L)^{1/n}. \tag{6}$$

Minkowski's Theorem immediately gives a linear upper bound on the Hermite constants:

$$\gamma_n \leq n.$$

The exact values of the Hermite constants have been explicitly determined only for a few small dimensions, as shown in Table 2.1.

n	2	3	4	5	6	7	8	24
γ_n	$2/\sqrt{3}$	$2^{1/3}$	$\sqrt{2}$	$8^{1/5}$	$(64/3)^{1/6}$	$64^{1/7}$	2	4
γ_n^n	4/3	2	4	8	64/3	64	256	2^{48}

Table 2.1: The known Hermite constants.

More generally, tight asymptotic bounds show that γ_n grows linearly in n:

$$\frac{n}{2\pi e} + \frac{\log(\pi n)}{2\pi e} + o(1) \leq \gamma_n \leq \frac{1.744n}{2\pi e}(1 + o(1)). \tag{7}$$


Herman Minkowski

The bound in Minkowski's Theorem is not tight in general (Exercise 2.12). Nevertheless, the *Gaussian heuristic* suggests that, for a suitably defined notion of a random full-rank lattice, this bound is close to optimal. More specifically, the expected length of a shortest nonzero vector in such a lattice is approximately

$$\lambda_1(L) \approx \sqrt{\frac{n}{2\pi e}}\ \mathrm{vol}(L)^{1/n}. \tag{8}$$

The quantity on the right-hand side of (8) is approximately equal to the radius of the n-dimensional sphere whose volume is $\mathrm{vol}(L)$.

Upper bounds for the second and higher successive minima that are similar to (6) and (3) are not possible; see Exercise 2.14. Nonetheless, the following theorem provides an upper bound on the geometric mean of the successive minima. In view of (1), this result is stronger than Theorem 2.15.

Theorem 2.16 (*Minkowski's Second Theorem*) Let $L \subseteq \mathbb{R}^n$ be a full-rank lattice. Then

$$\left(\prod_{i=1}^{n} \lambda_i(L)\right)^{1/n} \leq \sqrt{n}\ \mathrm{vol}(L)^{1/n}. \tag{9}$$

2.5 Fundamental lattice problems

We now introduce several fundamental computational problems on lattices whose presumed hardness underpins the security of lattice-based cryptosystems. Throughout, a lattice L is specified by a basis B, and all lattices are assumed to be full-rank integer lattices of dimension n.

Definition 2.17 The *Shortest Vector Problem* (SVP) is defined as follows: given a lattice $L = L(B)$, find a nonzero lattice vector of smallest length $\lambda_1(L)$.

SVP has been shown to be NP-hard, and it is therefore reasonable to assume that it is hard in the worst case. The fastest classical algorithm known for SVP runs in (heuristic) time $2^{0.292n+o(n)}$, whereas the best quantum algorithm known is slightly faster, with (heuristic) running time $2^{0.256n+o(n)}$. In both cases, the running time is fully exponential in the lattice dimension.

A natural relaxation of SVP is the approximate version, in which one seeks a nonzero lattice vector whose length is within a prescribed approximation factor of $\lambda_1(L)$.

Definition 2.18 The *Approximate Shortest Vector Problem* (approx-SVP or SVP_γ) is defined as follows: given a lattice $L = L(B)$, find a nonzero lattice vector of length at most $\gamma \cdot \lambda_1(L)$, where γ is the approximation factor.

Notice that SVP_1 coincides with SVP. As γ increases, SVP_γ becomes easier; more precisely, if $\gamma_1 \geq \gamma_2$ then $\mathrm{SVP}_{\gamma_1} \leq \mathrm{SVP}_{\gamma_2}$[3]. For constant approximation factors γ, SVP_γ remains NP-hard. However, SVP_γ is unlikely to be NP-hard when $\gamma > \sqrt{n}$. For approximation factor $\gamma = 2^k$, the fastest classical algorithm known for SVP_γ runs in time $2^{\tilde{\Theta}(n/k)}$, where the $\tilde{\Theta}$ notation suppresses a power of $\log n$. When the approximation factor is very large, more precisely when $\gamma > 2^{(n \log\log n)/\log n}$, SVP_γ can be solved in polynomial time by applying the BKZ lattice basis reduction algorithm (§9) to compute a short lattice basis and then selecting the shortest basis vector.

[3]Let A and B be two computational problems. We say that A *polynomial-time* reduces to B, denoted $A \leq B$, if there exists a polynomial-time algorithm for solving A that uses a polynomial-time algorithm for solving B as an oracle (a hypothetical subroutine). We say that A and B are *computationally equivalent* if $A \leq B$ and $B \leq A$.

Another important lattice problem is to find many short linearly independent vectors.

Definition 2.19 The *Shortest Independent Vectors Problem* (SIVP) is defined as follows: given a lattice $L = L(B)$, find n linearly independent lattice vectors, each of length at most $\lambda_n(L)$.

Definition 2.20 The *Approximate Shortest Independent Vectors Problem* (approx-SIVP or SIVP_γ) is defined as follows: given a lattice $L = L(B)$, find n linearly independent lattice vectors, each of length at most $\gamma \cdot \lambda_n(L)$.

It is worth emphasizing that a solution to SIVP need not form a lattice basis; the vectors obtained might be linearly independent without spanning the full lattice (see Exercises 2.17 and 2.18).

The complexity status of SIVP and approx-SIVP closely parallels that of SVP and approx-SVP. In particular, SIVP is NP-hard, as is SIVP_γ for constant approximation factors γ, and the fastest algorithms for these problems have fully exponential running time. Moreover, there is a known reduction $\mathrm{SIVP}_{\gamma\sqrt{n}} \leq \mathrm{SVP}_\gamma$, further highlighting the close relationship between these fundamental lattice problems.

The fundamental lattice problem we consider is finding a lattice vector closest to a given target vector.

Definition 2.21 Let $L \subseteq \mathbb{R}^n$ and let $t \in \mathbb{R}^n$. The *distance* from t to L is defined by

$$\mathrm{dist}(t, L) = \min_{x \in L} \|x - t\|.$$

Definition 2.22 The *Closest Vector Problem* (CVP) is defined as follows: given a lattice $L = L(B)$ and a target vector $t \in \mathbb{R}^n$, find a lattice vector $x \in L$ that is closest to t, that is, satisfying $\|x - t\| = \mathrm{dist}(t, L)$.

Definition 2.23 The *Approximate Closest Vector Problem* (approx-CVP or CVP_γ) is defined as follows: given a lattice $L = L(B)$ and a target vector $t \in \mathbb{R}^n$, find a lattice vector $x \in L$ such that $\|x - t\| \leq \gamma \cdot \mathrm{dist}(t, L)$.

The CVP and SIVP problems are closely related. It is known that $\mathrm{SIVP}_\gamma \leq \mathrm{CVP}_\gamma$ for every $\gamma \geq 1$, and $\mathrm{CVP} \leq \mathrm{SIVP}$. The CVP and SVP problems are also closely related: known relationships are $\mathrm{SVP}_\gamma \leq \mathrm{CVP}_\gamma$ for every $\gamma \geq 1$ (Exercise 8.24) and $\mathrm{CVP}_{\gamma^2\sqrt{n}} \leq \mathrm{SVP}_\gamma$. Figure 2.4 summarizes the known relationships between SVP, SIVP, and CVP, and their approximate versions.

2.6 Exercises

Exercise 2.1 Consider the lattice $L \subseteq \mathbb{Z}^3$ with basis matrix

$$B = \begin{bmatrix} 2 & 0 & -5 \\ 4 & 3 & 1 \\ -2 & -3 & 2 \end{bmatrix}.$$

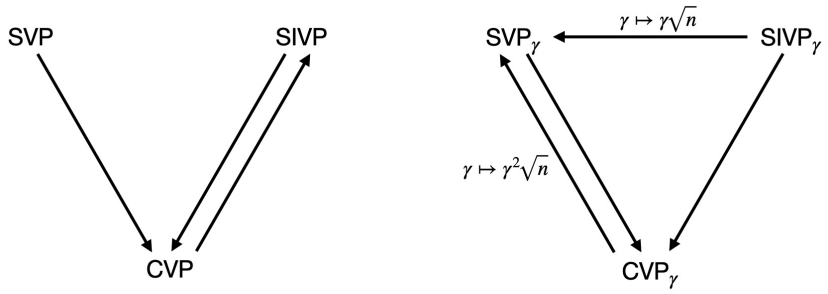


Figure 2.4: The known relationships between SVP, SIVP, CVP, and their approximate versions. An arrow $A \to B$ indicates that A polynomial-time reduces to B. Any loss in the approximation factor incurred by the reduction is shown next to the arrow.

(a) Show that $(7, -3, 2)$ is in L.
(b) Show that $(-4, 4, 0)$ is not in L.

Exercise 2.2 Show that the following two matrices are basis matrices for the same lattice.

$$B_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}, \qquad B_2 = \begin{bmatrix} 1 & 3 & 5 \\ 1 & 4 & 6 \\ 0 & 2 & 4 \end{bmatrix}.$$

Exercise 2.3 Consider the lattices $L_1 = L(B_1)$ and $L_2 = L(B_2)$ where

$$B_1 = \begin{bmatrix} 2 & 1 & 1 \\ 0 & 3 & 1 \\ 1 & 2 & 4 \end{bmatrix} \quad \text{and} \quad B_2 = \begin{bmatrix} 3 & 4 & 2 \\ 2 & 6 & 5 \\ 5 & 5 & 6 \end{bmatrix}.$$

(a) Show that L_2 is a sublattice of L_1.
(b) Show that $L_1 \neq L_2$.

Exercise 2.4 Let L_1 and L_2 be full-rank integer lattices with $L_1 \subseteq L_2$. Prove that $\mathrm{vol}(L_2)$ divides $\mathrm{vol}(L_1)$.

Exercise 2.5 Let $n \geq 1$. Prove that there is a unique full-rank integer lattice of volume 1 in $\mathbb{R}^n$.

Exercise 2.6 Let $L = L(B) \subseteq \mathbb{R}^n$ be a full-rank lattice. Recall that $P(B)$ is the fundamental parallelepiped of L with respect to B. Prove that every vector $x \in \mathbb{R}^n$ can be written uniquely as $x = u + v$ where $u \in L$ and $v \in P(B)$.

Exercise 2.7 Prove that $\lambda_1(L) = \min_{x,y \in L,\ x \neq y} \|x - y\|_2$.

Exercise 2.8 Prove that $\lambda_i(\mathbb{Z}^n) = 1$ for all $1 \leq i \leq n$.

Exercise 2.9 Consider the two-dimensional integer lattice L with basis $b_1 = (2, 0)$, $b_2 = (0, 3)$. Determine $\lambda_1(L)$.

Exercise 2.10 Consider the two-dimensional integer lattice L with basis $b_1 = (2, 1)$, $b_2 = (1, 2)$. Determine $\lambda_1(L)$.

Exercise 2.11 Consider the *hexagonal lattice* $L \subseteq \mathbb{R}^2$ with basis $b_1 = (1, 0)$, $b_2 = (\frac{1}{2}, \frac{\sqrt{3}}{2})$.
(a) Find all lattice vectors of length 1.
(b) Prove that $\lambda_1(L) = 1$.

Exercise 2.12 Let d_i be positive real numbers with $d_1 \leq d_2 \leq \cdots \leq d_n$. Prove that there exists a lattice $L \subseteq \mathbb{R}^n$ with $\lambda_i(L) = d_i$ for all $1 \leq i \leq n$.

Exercise 2.13 Let $L \subseteq \mathbb{R}^n$ be a full-rank lattice. Prove that there exists $x \in L$ with $0 < \|x\|_\infty \leq \mathrm{vol}(L)^{1/n}$.

Exercise 2.14 Let $0 < \varepsilon < 1$, and consider the lattice $L_\varepsilon \subseteq \mathbb{R}^2$ with basis matrix

$$B_\varepsilon = \begin{bmatrix} \varepsilon & 0 \\ 0 & 1/\varepsilon \end{bmatrix}.$$

Prove that $\mathrm{vol}(L_\varepsilon) = 1$, $\lambda_1(L_\varepsilon) = \varepsilon$, and $\lambda_2(L_\varepsilon) = 1/\varepsilon$. Conclude that, relative to the lattice volume, the first and second successive minima can be made arbitrarily small and arbitrarily large, respectively.

Exercise 2.15 Let $L = L(B)$ be a full-rank lattice in $\mathbb{R}^n$. Define $L^* = \{x \in \mathbb{R}^n : \langle x, v \rangle \in \mathbb{Z} \text{ for all } v \in L\}$.
(a) Prove that L^* is a lattice with basis matrix $(B^{-1})^T$. (L^* is called the *dual* of L.)
(b) Prove that $(L^*)^* = L$.
(c) Prove that $\mathrm{vol}(L^*) = 1/\mathrm{vol}(L)$.
(d) Determine L^* where $L = \mathbb{Z}^n$.

Exercise 2.16 An integer lattice L is said to be *q-ary* if $q\mathbb{Z}^n \subseteq L$. Prove that every integer lattice L is q-ary for $q = \mathrm{vol}(L)$.

Exercise 2.17 Let L be the set of all vectors $(x_1, x_2, x_3, x_4) \in \mathbb{Z}^4$ such that $\sum_{i=1}^4 x_i$ is even.
(a) Prove that L is a full-rank lattice.
(b) Prove that $\lambda_1(L) = \lambda_2(L) = \lambda_3(L) = \lambda_4(L) = \sqrt{2}$.
(c) Prove that the following is a basis matrix for L:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

(d) Find a set of linearly independent lattice vectors $[b_1, b_2, b_3, b_4]$ that is not a lattice basis, but which is shortest possible, i.e., $\|b_i\| = \sqrt{2}$.

Exercise 2.18 Let $n \geq 5$ and let e_i denote the ith unit vector in $\mathbb{R}^n$. Consider the lattice $L = L(B) \subseteq \mathbb{Z}^n$ with basis $B = [2e_1, 2e_2, \ldots, 2e_{n-1}, e_1 + \cdots + e_n]$.
(a) Prove that $\lambda_1(L) = 2$.
(b) Prove that $I = [2e_1, 2e_2, \ldots, 2e_n]$ is a linearly independent set of lattice vectors.
(c) Prove that $\lambda_2(L) = \cdots = \lambda_n(L) = 2$.
(d) Prove that L does not have a basis $B = [b_1, \ldots, b_n]$ with $\max_i \|b_i\| = \lambda_n(L)$.

Exercise 2.19 Let $i \geq 1$ and let $L \subseteq \mathbb{R}^i$ be a lattice having a vector v of length $\gamma_i \operatorname{vol}(L)^{1/i}$ (cf. Definition 2.14). Let B be a basis for L and define

$$B' = \begin{bmatrix} B & 0 \\ 0 & \gamma_i \operatorname{vol}(L)^{1/i} \end{bmatrix}_{(i+1)\times(i+1)}.$$

(a) Prove that $\lambda_1(L') = \gamma_i \operatorname{vol}(L)^{1/i}$ where $L' = L(B')$.
(b) Prove that $\gamma_i^i \leq \gamma_{i+1}^{i+1}$ for all $i \geq 1$.
(c) Prove that $\gamma_i^i \leq \gamma_j^j$ for all $1 \leq i \leq j$.

3 Short Integer Solution (SIS) problem

Contents

The security of the Dilithium signature scheme relies on the assumed hardness of the Short Integer Solution (SIS) problem. This section introduces SIS, presents a lattice formulation of SIS, and explores its relationship with SVP and SIVP.

3.1 Problem statement

Notation. q is a prime; $\mathbb{Z}_q = \{0, 1, 2, \ldots, q-1\}$ is the set of integers modulo q; $\mathbb{Z}_q^m$ is the set of length-m (column) vectors whose components are in $\mathbb{Z}_q$; $\mathbb{Z}_q^{n\times m}$ is the set of $n \times m$ matrices whose entries are in $\mathbb{Z}_q$; and $[-B, B]^m$ denotes the set of all length-m (column) vectors whose components are integers in the interval $[-B, B]$.

The SIS problem was introduced by Miklós Ajtai in 1996. It is parameterized by four values: positive integers n and m with $n \ll m$, a prime modulus q, and a bound B satisfying $B \ll q/2$ and $B^m > q^n$.

Definition 3.1 The *Short Integer Solution* problem, denoted $\mathrm{SIS}(n, m, q, B)$, is defined as follows. Given a uniformly random matrix $A \in_R \mathbb{Z}_q^{n\times m}$, find a nonzero vector $z \in \mathbb{Z}^m$ such that $Az = 0 \pmod q$ and $z \in [-B, B]^m$; see Figure 3.1.

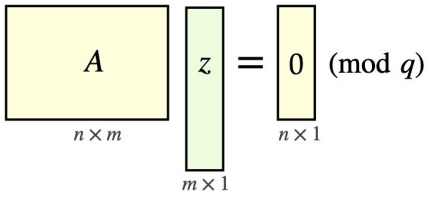


Figure 3.1: SIS problem.

SIS seeks to find a small nonzero solution in the null space of $A \pmod q$. While Gaussian elimination can efficiently determine a basis for this null space, the challenge is to find a nonzero element all of whose coordinates lie within $[-B, B]$.

When $n \geq m$, one expects that A has rank m, meaning that $z = 0$ is the only solution to $Az = 0 \pmod q$, and thus no SIS solution exists. This explains the requirement that

$n \ll m$, which also ensures that A has full rank over $\mathbb{Z}_q$ with overwhelming probability. Throughout the remainder of this section, we therefore assume that A has rank n.

Because $B^m > q^n$, the pigeonhole principle guarantees the existence of two distinct vectors $z_1, z_2 \in [-B/2, B/2]^m$ such that $Az_1 = Az_2 \pmod q$. Consequently, their difference $z = z_1 - z_2$ yields a solution to the SIS problem, establishing that a solution must exist. It is important to note, however, that an SIS solution is not unique. Indeed, if z is one SIS solution, then so is $-z$.

Example 3.2 (*SIS instance*) Let $n = 3$, $m = 5$, $q = 13$, and $B = 3$. Consider the SIS instance[4]

$$A = \begin{bmatrix} 10 & 6 & 3 & 6 & 0 \\ 9 & 11 & 0 & 10 & 3 \\ 7 & 11 & 5 & 7 & 8 \end{bmatrix} \in \mathbb{Z}_{13}^{3\times 5}.$$

We need to find nonzero $z = (z_1, z_2, z_3, z_4, z_5) \in [-3, 3]^5$ such that $Az = 0 \pmod{13}$.

Gaussian elimination (mod 13) on A yields the reduced matrix

$$A' = \begin{bmatrix} 1 & 0 & 0 & 8 & 12 \\ 0 & 1 & 0 & 5 & 7 \\ 0 & 0 & 1 & 0 & 11 \end{bmatrix}.$$

The rows of A' each provide one equation, and thus the complete solution to $Az = 0 \pmod{13}$ is $z_1 = 5z_4 + z_5$, $z_2 = 8z_4 + 6z_5$, $z_3 = 2z_5$, $z_4 \in \mathbb{Z}_{13}$, $z_5 \in \mathbb{Z}_{13}$. Among the $13^2 = 169$ solutions $z \in \mathbb{Z}_{13}^5$ are four SIS solutions (easily found by trial and error): $z = \pm(2, -2, 0, 3, 0)$ and $z = \pm(3, -3, 0, -2, 0)$.

We now give an example of a cryptographic primitive whose security relies on the hardness of SIS.

Example 3.3 (*cryptographic hash function*) Let $A \in_R \mathbb{Z}_q^{n\times m}$ with $m > n \log q$. Define the hash function $H_A : \{0,1\}^m \longrightarrow \mathbb{Z}_q^n$ by

$$H_A(z) = Az \pmod q.$$

The function H_A is collision resistant under the assumption that $\mathrm{SIS}(n, m, q, 1)$ is hard. To justify this, suppose there exists an efficient algorithm that finds a collision for H_A, meaning two distinct $z_1, z_2 \in \{0,1\}^m$ such that $H_A(z_1) = H_A(z_2)$. Then $Az_1 = Az_2 \pmod q$, which implies that $Az = 0 \pmod q$ where $z = z_1 - z_2$. Since $z \neq 0$ and $z \in [-1, 1]^m$, the vector z is an $\mathrm{SIS}(n, m, q, 1)$ solution that has been found efficiently.

We next present two alternative formulations of the SIS problem.

Definition 3.4 The *Inhomogeneous Short Integer Solution* problem $\mathrm{ISIS}(n, m, q, B)$ is the following. Given $A \in_R \mathbb{Z}_q^{n\times m}$ and $b \in_R \mathbb{Z}_q^n$, find $z \in \mathbb{Z}^m$ such that $Az = b \pmod q$ and $z \in [-B, B]^m$. Here, $B \ll q/2$.

[4] In this example, the condition $B^m > q^n$ is not satisfied.

Our assumption that $B^m \gg q^n$ implies that $(2B+1)^m \gg q^n$, and thus an ISIS solution is likely to exist.

Example 3.5 (*ISIS instance*) Let $n = 3$, $m = 5$, $q = 29$, and $B = 5$. Consider the ISIS instance

$$A = \begin{bmatrix} 4 & 19 & 5 & 4 & 22 \\ 8 & 10 & 19 & 1 & 7 \\ 13 & 10 & 20 & 7 & 12 \end{bmatrix} \in \mathbb{Z}_{29}^{3\times 5}, \quad b = \begin{bmatrix} 7 \\ 26 \\ 7 \end{bmatrix} \in \mathbb{Z}_{29}^{3}.$$

We need to find nonzero $z = (z_1, z_2, z_3, z_4, z_5) \in [-5, 5]^5$ such that $Az = b \pmod{29}$.

By trial and error, we find four ISIS solutions: $z = (-5, 3, -4, 3, -1)$, $(-4, -3, 3, -4, 5)$, $(0, 2, 2, -3, 0)$, $(4, -2, 3, 0, -2)$.

Theorem 3.6 SIS and ISIS are computationally equivalent.

Proof: (ISIS $\leq$ SIS) Let (A, b) be an ISIS(n, m, q, B) instance. Select $j \in_R [1, m+1]$ and $c \in_R [-B, B]$ with $c \neq 0$. Let A' be the $n\times(m+1)$ matrix obtained by inserting $-c^{-1}b \bmod q$ as a new jth column in A. Now, use an SIS solver to find an SIS solution $z' \in [-B, B]^{m+1}$ to $A'z' = 0 \pmod{q}$. If indeed the jth entry in z' is c, then $Az = b \pmod{q}$, where $z \in [-B, B]^m$ is obtained from z' by deleting its jth entry. Thus, z is an ISIS solution that we have efficiently found with non-negligible probability approximately $1/(2Bm)$.

(SIS $\leq$ ISIS) Let A be an SIS(n, m, q, B) instance. Write $A = [A' | -b']$ where $A' \in \mathbb{Z}_q^{n\times(m-1)}$ and $b' \in \mathbb{Z}_q^n$. Use an ISIS solver to find an ISIS solution z' to the ISIS instance (A', b'). We then have $A'z' = b' \pmod{q}$ and $z' \in [-B, B]^{m-1}$. Then $z = (z', 1) \in \mathbb{Z}^m$ satisfies $Az = 0 \pmod{q}$, $z \neq 0$, and $z \in [-B, B]^m$. Thus, z is an SIS solution that we have efficiently found. □

Definition 3.7 The *normal-form ISIS* problem nf-ISIS(n, m, q, B) is the following. Given $A \in_R \mathbb{Z}_q^{n\times m}$ and $b \in_R \mathbb{Z}_q^n$, find $z \in \mathbb{Z}^{m+n}$ such that $[A \,|\, I_n]z = b \pmod{q}$ and $z \in [-B, B]^{m+n}$. Here, I_n is the $n \times n$ identity matrix and $B \ll q/2$.

Exercise 3.3 shows that nf-ISIS(n, m, q, B) and ISIS$(n, m+n, q, B)$ are computationally equivalent.

3.2 The SIS lattice

We next introduce the lattice associated with an SIS instance.

Definition 3.8 Given an instance A of SIS(n, m, q, B), the associated *SIS lattice* is defined by

$$L_A^{\perp} = \{z \in \mathbb{Z}^m : Az = 0 \pmod{q}\}.$$

The proof of Theorem 3.10 relies on the following standard result on free (additive) abelian groups.

Theorem 3.9 Let G be a free abelian group of rank m with $\mathbb{Z}$-basis $x_1, \ldots, x_m$, and let H be a rank-m subgroup of G with $\mathbb{Z}$-basis $y_1, \ldots, y_m$. For each $1 \le i \le m$, let $y_i = \sum_j a_{ij} x_j$ where $a_{ij} \in \mathbb{Z}$. Then the quotient group G/H is finite and $|G/H| = |\det(a_{ij})|$.

Theorem 3.10 The SIS lattice $L_A^\perp \subseteq \mathbb{R}^m$ is a full-rank integer lattice with volume q^n.

Proof: It is easy to check that $L_A^\perp$ is closed under addition and negation, and hence forms an additive subgroup of $\mathbb{Z}^m$. Since all of its elements have integral coordinates, the set is discrete, and thus $L_A^\perp$ is an integer lattice.

To see that $L_A^\perp$ has full rank, observe that the lattice $q\mathbb{Z}^m$, which consists of all integer vectors whose coordinates are multiples of q, is generated by the m linearly independent vectors $(q, 0, \ldots, 0), (0, q, \ldots, 0), \ldots, (0, 0, \ldots, q)$. Hence, $q\mathbb{Z}^m$ is a full-rank lattice. Moreover, $q\mathbb{Z}^m$ is a sublattice of $L_A^\perp$, which implies that $L_A^\perp$ itself is full rank.

Lastly, note that both $\mathbb{Z}^m$ and $L_A^\perp$ are free abelian groups of rank m. Since $L_A^\perp$ is a subgroup of $\mathbb{Z}^m$, Theorem 3.9 asserts that $\mathrm{vol}(L_A^\perp) = |\mathbb{Z}^m / L_A^\perp|$. To compute this quantity, we need to determine the number of cosets of $L_A^\perp$ in $\mathbb{Z}^m$. Now, let $x, y \in \mathbb{Z}^m$. Then x and y are in the same coset of $L_A^\perp$, meaning that $L_A^\perp + x = L_A^\perp + y$, if and only if $x - y \in L_A^\perp$, i.e., $A(x-y) = 0 \pmod q$. The latter condition is equivalent to $Ax = Ay \pmod q$. Thus, the number of cosets of $L_A^\perp$ in $\mathbb{Z}^m$ equals the size of the image of $A \pmod q$. Since A has rank n over $\mathbb{Z}_q$, the image has size q^n, and hence $\mathrm{vol}(L_A^\perp) = |\mathbb{Z}^m / L_A^\perp| = q^n$. □

The lattice $L_A^\perp$ is an example of a *q-ary lattice*: membership is determined entirely by reducing the vector modulo q. We now describe an explicit basis for this lattice.

Theorem 3.11 Suppose that the first n columns of A are linearly independent over $\mathbb{Z}_q$. Then A can be row-reduced over $\mathbb{Z}_q$ to the form $\widetilde{A} = [I_n | \overline{A}]$, where $\overline{A} \in \mathbb{Z}_q^{n \times (m-n)}$. In this case, the matrix

$$C = \begin{bmatrix} qI_n & -\overline{A} \\ 0 & I_{m-n} \end{bmatrix} \in \mathbb{Z}^{m \times m} \tag{10}$$

is a basis matrix for the SIS lattice $L_A^\perp$.

Proof: Since A and $\widetilde{A}$ are row equivalent over $\mathbb{Z}_q$, they have the same null space modulo q and hence define the same SIS lattice: $L_{\widetilde{A}}^\perp = L_A^\perp$. It therefore suffices to find a basis for $L_{\widetilde{A}}^\perp$.

Each column v of C satisfies $\widetilde{A}v = 0 \pmod q$, and thus lies in $L_{\widetilde{A}}^\perp$. The columns of C are linearly independent over $\mathbb{R}$, since $\det(C) = q^n \neq 0$. Consequently, C is a basis matrix for a full-rank sublattice L of $L_{\widetilde{A}}^\perp$. Finally, the determinant of C shows that $\mathrm{vol}(L) = q^n = \mathrm{vol}(L_{\widetilde{A}}^\perp)$. This implies that $L_{\widetilde{A}}^\perp = L$, and therefore C is a basis matrix for $L_A^\perp$. □

3.3 Hardness of SIS

We are now ready to reformulate the SIS problem purely as a lattice problem: namely, the task of finding a short nonzero vector in a specific lattice.

Definition 3.12 The *Short Integer Solution* problem $\mathrm{SIS}(n, m, q, B)$ is defined as follows. Given a uniformly random matrix $A \in_R \mathbb{Z}_q^{n\times m}$, find a vector $z \in \mathbb{Z}^m$ with $0 < \|z\|_\infty \le B$ that lies in the SIS lattice $L_A^\perp = L(C)$, where C is the basis matrix defined in (10).

SIS hardness is usually studied using the Euclidean norm rather than the infinity norm. The two formulations are closely related since for every $z \in \mathbb{R}^m$, we have $\|z\|_\infty \le \|z\|_2 \le \sqrt{m}\,\|z\|_\infty$. This motivates the following closely related variant of SIS.

Definition 3.13 The $\mathrm{SIS}_2(n, m, q, B)$ problem is defined as follows. Given $A \in_R \mathbb{Z}_q^{n\times m}$, find a vector $z \in \mathbb{Z}^m$ in $L_A^\perp$ such that $0 < \|z\|_2 \le B$.

3.3.1 Dual attack on SIS_2

Recall that the SIS lattice $L_A^\perp$ has rank m and volume q^n. By Minkowski's Theorem (Theorem 2.15), the length of a shortest nonzero vector in $L_A^\perp$ satisfies

$$\lambda_1(L_A^\perp) \le \sqrt{m}\, q^{n/m}.$$

We henceforth assume that $B \ge \sqrt{m}\, q^{n/m}$, which guarantees the existence of a solution to SIS_2. According to the Gaussian heuristic (equation (8)), one expects that

$$\lambda_1(L_A^\perp) \approx \sqrt{m/(2\pi e)}\, q^{n/m}.$$

Let γ denote the ratio between the SIS_2 solution bound B and the shortest length predicted by the Gaussian heuristic, namely

$$\gamma = B\sqrt{2\pi e}/(\sqrt{m}\, q^{n/m}). \tag{11}$$

With this notation, the SIS_2 problem can be viewed as an instance of the approximate shortest vector problem SVP_γ in the lattice $L_A^\perp$ with approximation factor γ. This strategy of reducing SIS_2 to an instance of SVP_γ is called a *dual attack*. An example is presented in §10.2.

3.3.2 Average-case hardness of SIS

It is natural to conjecture that SIS is hard in the *worst case*. However, in cryptographic applications an SIS instance is generated by choosing a matrix A uniformly at random. Therefore, what is required is not merely worst-case hardness, but rather the guarantee that SIS is *always* hard, except with negligible probability. In other words, we need SIS to be hard *on average*.

Another concern with SIS is that the corresponding approx-SVP instance has considerable structure. In particular, the SIS lattice admits a basis matrix of the special form given by (10), which is upper-triangular with q's in the upper-left diagonal block and 1's in the lower-right diagonal block. One might therefore be concerned that, even if approx-SVP is hard in general, it could be significantly easier on the restricted family of lattices arising from SIS instances.

In his seminal 1996 work, Ajtai alleviated these concerns by proving that SIS is hard *on average* under standard complexity-theoretic assumptions. Specifically, he showed that if approx-SIVP is hard in the worst case, then SIS is hard on average. Reductions of this type are known as *worst-case to average-case reductions*, and they provide (conditional) assurances for the average-case hardness of SIS since approx-SIVP is a natural and well-studied lattice problem, at least for relatively small approximation factors.

Ajtai's reduction is technically involved and beyond the scope of this introductory treatment. At a very high level (see Figure 3.2), the reduction starts from an *arbitrary*

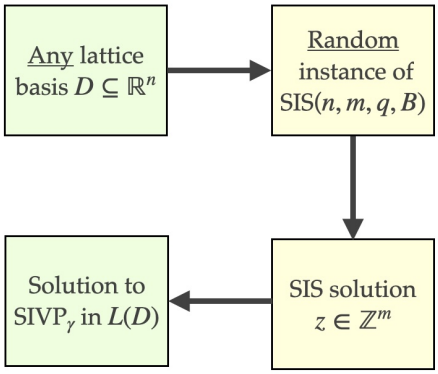


Figure 3.2: Proving the average-case hardness of SIS.

n-dimensional lattice with basis D and efficiently generates a *random* SIS instance. A solution to this SIS instance can then be efficiently transformed into a solution to approx-SIVP in the lattice generated by D. Consequently, an efficient algorithm for solving SIS on average would imply an efficient algorithm for solving approx-SIVP in the worst case.

Ajtai's worst-case to average-case reduction for SIS is one of the most remarkable results in lattice-based cryptography. It should be noted, however, that the guarantee it provides for the average-case hardness of SIS is asymptotic in nature. Furthermore, the reduction is highly non-tight, which means that the algorithm that solves approx-SIVP using an SIS solver is very inefficient. The asymptotic nature and non-tightness of the reduction prevents it from being used directly to derive concrete parameters for practical cryptographic schemes based on SIS.

3.4 Exercises

Exercise 3.1 Write a program to find all solutions z to the following SIS$(4, 6, 23, 5)$ instance:

$$A = \begin{bmatrix} 16 & 12 & 19 & 0 & 7 & 3 \\ 12 & 6 & 5 & 1 & 20 & 11 \\ 14 & 9 & 14 & 15 & 1 & 10 \\ 1 & 0 & 16 & 19 & 17 & 2 \end{bmatrix}.$$

Exercise 3.2 Write a program to find all solutions z to the following ISIS$(3, 5, 29, 5)$ instance:

$$A = \begin{bmatrix} 7 & 9 & 6 & 27 & 15 \\ 22 & 9 & 22 & 19 & 24 \\ 0 & 2 & 16 & 19 & 11 \end{bmatrix}, \quad b = \begin{bmatrix} 13 \\ 25 \\ 16 \end{bmatrix}.$$

Exercise 3.3 Prove that nf-ISIS(n, m, q, B) and ISIS$(n, m + n, q, B)$ are computationally equivalent.

4 Learning With Errors (LWE) problem

Contents

The security of a wide range of cryptographic protocols, including the Kyber key encapsulation mechanism and the Dilithium signature scheme, relies on the hardness of the Learning With Errors (LWE) problem. This section introduces LWE, presents a lattice formulation of LWE, and explores its relationships with SVP and SIVP.

4.1 Problem statement

The LWE problem was introduced by Oded Regev in 2005. It is parameterized by four values: positive integers m and n with $m \gg n$, a prime modulus q, and a noise bound B satisfying $B \ll q/2$.

Definition 4.1 The *Learning With Errors* problem, denoted LWE(m, n, q, B), is defined as follows. Choose a matrix $A \in_R \mathbb{Z}_q^{m \times n}$, a secret vector $s \in_R \mathbb{Z}_q^n$, and an error vector $e \in_R [-B, B]^m$, and let $b = As + e \pmod q \in \mathbb{Z}_q^m$. Given the pair (A, b), the goal is to recover s; see Figure 4.1. The parameter m is the number of *LWE samples*.

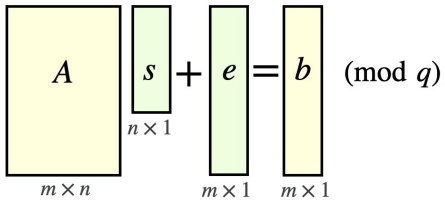


Figure 4.1: LWE problem.

Observe that if $B = 0$, so that $e = 0$, then the problem is to solve the system of linear equations $As = b \pmod q$ for s. In this case, the set of all solutions can be efficiently determined using Gaussian elimination over $\mathbb{Z}_q$. Hardness of LWE arises because of the introduction of the noise e. We henceforth assume that $B \neq 0$.

Remark 4.2 (*distribution choices in LWE*) In Definition 4.1, the secret vector s is sampled uniformly over $\mathbb{Z}_q^n$ and the error vector e uniformly from the range $[-B, B]^m$. More generally, s and e can be sampled from any distribution. For instance, discrete Gaussian distributions are used in numerous cryptographic protocols including the Falcon signature scheme, and are central to establishing worst-case to average-case security proofs. Alternatively, the centered binomial distribution is employed in Kyber; see §6.2. Furthermore, each coordinate of s and each coordinate of e can be drawn from the same distribution, as is done in the short-secret variant of LWE (Definition 4.6).

The statement of the LWE problem implicitly assumes that the LWE solution s is unique. In fact, the solution is expected to be unique with overwhelming probability provided that the number m of rows of A is sufficiently greater than the number n of columns of A.

To support this claim, observe that multiplying A by a vector $s \in \mathbb{Z}_q^n$ results in a vector $t = As \pmod q$ in $\mathbb{Z}_q^m$, a vector space with q^m elements. For each $s \in \mathbb{Z}_q^n$, the sphere centered at $t = As \pmod q$ is defined as $T_s = \{t + e \pmod q \mid e \in [-B, B]^m\}$; note that the number of vectors in each sphere is $|T_s| = (2B + 1)^m$. Uniqueness of the LWE solution is guaranteed only if no two of the q^n spheres overlap (see Figure 4.2).

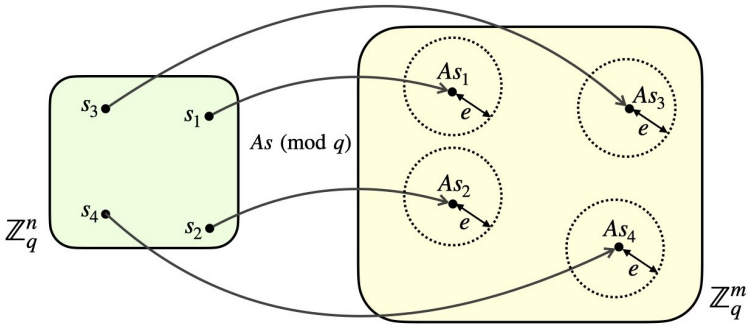


Figure 4.2: Uniqueness of the LWE solution.

Now[5], the two spheres centered at As_1 and As_2 overlap if and only if there exist $e_1, e_2 \in [-B, B]^m$ such that $As_1 + e_1 = As_2 + e_2 \pmod q$. Letting $\Delta s = s_1 - s_2$ and $\Delta e = e_2 - e_1$, we see that the LWE solution is not unique if and only if there exist $\Delta s \in \mathbb{Z}_q^n \setminus \{0\}$ and $\Delta e \in [-2B, 2B]^m$ such that[6] $A\,\Delta s = \Delta e$. We make the heuristic assumption that $A\,\Delta s$ is a uniformly random vector in $\mathbb{Z}_q^m$ for each nonzero Δs. Then, for each nonzero $\Delta s \in \mathbb{Z}_q^n$, the probability that $A\,\Delta s$ is in $[-2B, 2B]^m$ is

$$\left(\frac{4B + 1}{q}\right)^m.$$

[5]Thang Cao Trinh showed me this argument.

[6]Henceforth, the $\pmod q$ operator will often be omitted when clear from context.

By the union bound, the probability that $A\Delta s$ is in $[-2B, 2B]^m$ for at least one nonzero $\Delta s \in \mathbb{Z}_q^n$ is at most

$$(q^n - 1)\left(\frac{4B+1}{q}\right)^m.$$

For $B \ll q/2$ and $n \ll m$, this probability is negligible, so the LWE solution is unique with overwhelming probability. In the remainder of these notes, we will assume that the LWE solution is unique.

Example 4.3 (*LWE instance*) Let $m = 5$, $n = 3$, $q = 47$, and $B = 2$. Consider the LWE(m, n, q, B) instance

$$A = \begin{bmatrix} 27 & 13 & 13 \\ 1 & 46 & 23 \\ 16 & 13 & 30 \\ 18 & 16 & 19 \\ 22 & 0 & 3 \end{bmatrix} \in \mathbb{Z}_{47}^{5\times 3}, \qquad b = \begin{bmatrix} 30 \\ 28 \\ 25 \\ 34 \\ 32 \end{bmatrix} \in \mathbb{Z}_{47}^5.$$

The LWE challenge is to find $s \in \mathbb{Z}_{47}^3$ and $e \in [-2, 2]^5$ such that $As + e = b \pmod{47}$. It turns out that this LWE instance has three solutions: $s = (2, 15, 12)$, $e = (1, 0, 2, 0, -1)$; $s = (19, 34, 12)$, $e = (0, 2, 0, 1, 1)$; and $s = (41, 33, 7)$, $e = (1, 0, -1, -2, 2)$.

Decisional LWE problem. In the decisional variant of LWE, the challenge is to distinguish an LWE instance (A, b) from a random instance (A, r) where $r \in \mathbb{Z}_q^m$ is randomly selected. In the latter case, one expects that $As + e = r \pmod{q}$ does not have an LWE solution. Essentially, the decisional problem is to decide whether a solution exists.

Definition 4.4 The *Decisional Learning With Errors* problem DLWE(m, n, q, B) is the following. Let $A \in_R \mathbb{Z}_q^{m\times n}$, $s \in_R \mathbb{Z}_q^n$, $e \in_R [-B, B]^m$, $r \in_R \mathbb{Z}_q^m$, and let $b = As + e \pmod{q}$. With equal probability $\frac{1}{2}$, set $c = b$ or $c = r$. Given a problem instance (A, c), decide (with success probability significantly greater than $\frac{1}{2}$) whether $c = b$ or $c = r$.

Theorem 4.5 LWE and DLWE are computationally equivalent.

Proof: We will first prove that LWE reduces to DLWE (denoted LWE $\leq$ DLWE) and then prove that DLWE reduces to LWE (denoted DLWE $\leq$ LWE).

(LWE $\leq$ DLWE) Let (A, b) be an LWE instance (where $b = As + e$). Assume we have a DLWE-solver[7] to test guesses for the coordinates of s, one at a time, beginning with the first coordinate s_1. We know there is a solution s, and proceed to find it. This procedure may need to be repeated up to qn times before s is determined.

Let $d \in \mathbb{Z}_q$. To test whether $s_1 = d$, select $\Delta \in_R \mathbb{Z}_q^m$. Let A' be the matrix obtained by adding Δ to the first column of A, and let $b' = b + d\Delta$.

Now, if $s_1 = d$, then $b' = A's + e$ so (A', b') is a valid LWE instance. On the other hand, if $s_1 \neq d$, then $b' = A's + e + (d - s_1)\Delta$. Since $d - s_1$ is nonzero, and Δ is uniformly random

[7] We assume that the DLWE solver returns correct answers with overwhelming probability.

and independent of A', s and e, it follows that b' is uniformly random and independent of A'. Thus, (A', b') is a valid DLWE instance, whence the DLWE solver with input (A', b') will inform us whether or not $s_1 = d$. We eventually find s_1 and the other s_i's similarly.

(DLWE $\leq$ LWE) Let (A, c) be a DLWE instance. Assume that we have an LWE-solver to efficiently solve the DLWE instance.

Now, if $c = b$, then (A, c) is an LWE instance and so one expects that $As + e = c \pmod q$ has a unique LWE solution (s, e). If $c = r$, as noted earlier, one expects that $As + e = c \pmod q$ has no LWE solution. So, the LWE solver is run with input (A, c). If a valid LWE solution is returned, then we conclude that $c = b$. If the LWE solver terminates without a valid LWE solution, or fails to terminate, then we conclude that $c = r$. □

Short-secret LWE problem. In the short-secret variant of the LWE problem, the coordinates of the secret vector s are sampled from the same distribution as those of the error vector e.

Definition 4.6 The *Short-secret Learning With Errors* problem, denoted ss-LWE(m, n, q, B), is defined as follows. Choose a matrix $A \in_R \mathbb{Z}_q^{m\times n}$, a secret vector $s \in_R [-B, B]^n$, and an error vector $e \in_R [-B, B]^m$, and define $b = As + e \pmod q \in \mathbb{Z}_q^m$. Given the pair (A, b), the goal is to recover the secret vector s.

Unlike the situation for LWE, the condition $n \ll m$ is not needed to ensure that ss-LWE has a unique solution with high probability (Exercise 4.3).

Example 4.7 (*ss-LWE instance*) Let $m = 5$, $n = 5$, $q = 97$, and $B = 4$. Consider the ss-LWE(m, n, q, B) instance

$$A = \begin{bmatrix} 15 & 30 & 51 & 23 & 77 \\ 6 & 46 & 54 & 84 & 23 \\ 32 & 12 & 80 & 78 & 34 \\ 32 & 4 & 88 & 18 & 4 \\ 52 & 49 & 67 & 31 & 26 \end{bmatrix} \in \mathbb{Z}_{97}^{5\times 5}, \qquad b = \begin{bmatrix} 34 \\ 40 \\ 90 \\ 85 \\ 29 \end{bmatrix} \in \mathbb{Z}_{97}^5.$$

The ss-LWE challenge is to find s and e in $[-4, 4]^5$ such that $As + e = b \pmod{97}$. It turns out that this ss-LWE instance has a unique solution $s = (1, -1, 0, 3, 1)$, $e = (0, -1, -4, -1, 4)$.

Theorem 4.8 The LWE and ss-LWE problems are computationally equivalent. More precisely, ss-LWE$(m, n, q, B) \leq$ LWE(m, n, q, B) and LWE$(m, n, q, B) \leq$ ss-LWE$(m - n, n, q, B)$.

Proof: First consider an instance (A, b) of ss-LWE(m, n, q, B), where $b = As+e \pmod q$ with $s \in_R [-B, B]^n$ and $e \in_R [-B, B]^m$. Select a random vector $d \in_R \mathbb{Z}_q^n$ and define

$$b' = b + Ad = (As + e) + Ad = A(s + d) + e \pmod q.$$

Then (A, b') forms an LWE(m, n, q, B) instance. If (s', e) is a solution to this LWE instance, then $(s' - d, e)$ is immediately a solution to the original ss-LWE instance. This establishes that ss-LWE$(m, n, q, B) \leq$ LWE(m, n, q, B).

Now suppose (A, b) is an LWE(m, n, q, B) instance, so that $b = As + e \pmod q$. Partition the matrix A and vectors b and e as

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \quad e = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix},$$

where $A_1 \in \mathbb{Z}_q^{n \times n}$, $A_2 \in \mathbb{Z}_q^{(m-n) \times n}$, $b_1, e_1 \in \mathbb{Z}_q^n$, and $b_2, e_2 \in \mathbb{Z}_q^{m-n}$. Assuming A_1 is invertible modulo q, define

$$A' = -A_2 A_1^{-1} \in \mathbb{Z}_q^{(m-n) \times n} \text{ and } b' = A' b_1 + b_2 \in \mathbb{Z}_q^{m-n}.$$

Then

$$\begin{aligned} b' &= A' b_1 + b_2 = (-A_2 A_1^{-1})(A_1 s + e_1) + (A_2 s + e_2) \\ &= -A_2 s - A_2 A_1^{-1} e_1 + A_2 s + e_2 = A' e_1 + e_2. \end{aligned}$$

Therefore, (A', b') forms an ss-LWE$(m-n, n, q, B)$ instance. Solving this ss-LWE instance yields e_1, from which the original secret s can be immediately recovered, thereby solving the original LWE instance. Hence, LWE$(m, n, q, B) \leq$ ss-LWE$(m - n, n, q, B)$. □

4.2 Lindner-Peikert public-key encryption

We now give an example of a public-key encryption scheme whose security relies on the hardness of the decisional version of ss-LWE, which is computationally equivalent to the ss-LWE problem (Exercise 4.4).

Definition 4.9 The *Decisional short-secret Learning With Errors* problem, denoted ss-DLWE(m, n, q, B), is defined as follows. Choose a matrix $A \in_R \mathbb{Z}_q^{m \times n}$, a secret vector $s \in_R [-B, B]^n$, an error vector $e \in_R [-B, B]^m$, and a random vector $r \in_R \mathbb{Z}_q^m$. Define $b = As + e \pmod q \in \mathbb{Z}_q^m$. Let $d = b$ with probability $\frac{1}{2}$ and $d = r$ with probability $\frac{1}{2}$. Given the pair (A, d), the goal is to determine whether $d = b$ or $d = r$ with success probability significantly greater than $\frac{1}{2}$.

Decryption relies on a rounding procedure, which is introduced next.

Definition 4.10 Let $q \geq 3$ be an odd integer, and let $r \in \mathbb{Z}_q$. Define the *symmetric mod q* of r to be

$$r \bmod^s q = \begin{cases} r, & \text{if } r \leq (q-1)/2 \\ r - q, & \text{if } r > (q-1)/2. \end{cases}$$

The mods operation extends naturally to all integers: if $z \in \mathbb{Z}$ then

$$z \bmod^s q = (z \bmod q) \bmod^s q.$$

Note that $-(q-1)/2 \le r \bmod q \le (q-1)/2$. For example, if $q = 17$ and $r \in \mathbb{Z}_{17}$ then $-8 \le r \bmod 17 \le 8$. We have $6 \bmod 17 = 6$ and $13 \bmod 17 = -4$.

Definition 4.11 Define the rounding function $\mathrm{Round}_q : \mathbb{Z} \to \{0, 1\}$ as follows:

$$\mathrm{Round}_q(x) = \begin{cases} 0, & \text{if } -q/4 < x \bmod q < q/4, \\ 1, & \text{otherwise.} \end{cases}$$

The rounding function is depicted in Figure 4.3. For example, if $q = 1223$ then $\mathrm{Round}_q(x) = 0$ if $0 \le x \le 305$ or $918 \le x \le 1222$, and $\mathrm{Round}_q(x) = 1$ if $306 \le x \le 917$.

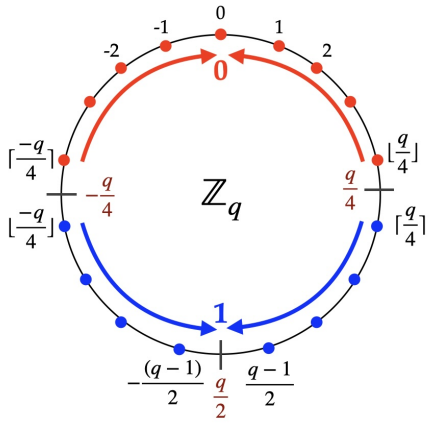


Figure 4.3: The rounding function.

In the Lindner-Peikert encryption scheme (Algorithm 4.12), we assume that the parameters q, n, B satisfy

$$B \le \sqrt{\frac{q}{4(2n+1)}}. \tag{12}$$

This condition guarantees the correctness of the decryption procedure. Also, $\lceil x \rfloor$ denotes the closest integer to x, with ties broken upwards.

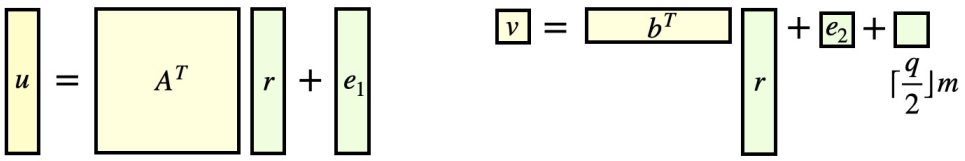


Figure 4.4: Lindner-Peikert encryption.

Algorithm 4.12: Lindner-Peikert public-key encryption scheme

Domain parameters. q, n, B with $B \leq \sqrt{q/(4(2n+1))}$.

Key generation. Bob does the following:

1 Select $A \in_R \mathbb{Z}_q^{n \times n}$, $s \in_R [-B, B]^n$, and $e \in_R [-B, B]^n$.
2 Compute $b = As + e \pmod{q}$.
3 Bob's encryption (public) key is (A, b); his decryption (private) key is s.

Encryption. To encrypt a message $m \in \{0, 1\}$ for Bob, Alice does the following:

4 Obtain an authentic copy of Bob's encryption key (A, b).
5 Select $r \in_R [-B, B]^n$, $e_1 \in_R [-B, B]^n$, and $e_2 \in_R [-B, B]$.
6 Compute $u = A^T r + e_1 \pmod{q}$ and $v = b^T r + e_2 + \lceil \frac{q}{2} \rfloor m \pmod{q}$ (see Figure 4.4).
7 Output $c = (u, v)$.

Decryption. To decrypt $c = (u, v)$, Bob does:

8 Compute $m = \text{Round}_q(v - s^T u)$.

Correctness of decryption. Observe that

$$\begin{aligned} v - s^T u &= (b^T r + e_2 + m\lceil q/2 \rfloor) - s^T (A^T r + e_1) \\ &= (s^T A^T + e^T) r + e_2 + m\lceil q/2 \rfloor - s^T (A^T r + e_1) \\ &= e^T r - s^T e_1 + e_2 + m\lceil q/2 \rfloor. \end{aligned}$$

Hence decryption succeeds, meaning $m = \text{Round}_q(v - s^T u)$, if

$$|e^T r - s^T e_1 + e_2 \bmod q| < q/4.$$

Using the error bound (12), we have

$$\begin{aligned} |e^T r - s^T e_1 + e_2| &\leq nB^2 + nB^2 + B \\ &\leq \frac{2nq}{4(2n+1)} + \sqrt{\frac{q}{4(2n+1)}} \\ &= \frac{nq}{2(2n+1)} + \sqrt{\frac{q}{4(2n+1)}} \\ &< \frac{q}{4}. \end{aligned}$$

Thus the rounding step correctly recovers m.

Example 4.13 (*Lindner-Peikert encryption*)
Domain parameters. $q = 997$, $n = 4$, and $B = 5$.
Key generation. Bob selects

$$A = \begin{bmatrix} 301 & 352 & 400 & 269 \\ 585 & 916 & 438 & 560 \\ 510 & 108 & 14 & 157 \\ 265 & 620 & 886 & 348 \end{bmatrix}, \quad s = \begin{bmatrix} 2 \\ 3 \\ -5 \\ 1 \end{bmatrix}, \quad e = \begin{bmatrix} 4 \\ 5 \\ 0 \\ 1 \end{bmatrix},$$

and computes

$$b = As + e \pmod q = \begin{bmatrix} 928 \\ 299 \\ 434 \\ 303 \end{bmatrix}.$$

Bob's encryption key is (A, b), and his decryption key is s.
Encryption. To encrypt the plaintext message $m = 0$ for Bob, Alice selects

$$r = \begin{bmatrix} -3 \\ -3 \\ 4 \\ -5 \end{bmatrix}, \quad e_1 = \begin{bmatrix} -1 \\ 2 \\ -3 \\ -5 \end{bmatrix}, \quad e_2 = -3,$$

and computes

$$u = A^T r + e_1 = \begin{bmatrix} 50 \\ 509 \\ 88 \\ 384 \end{bmatrix} \text{ and } v = b^T r + e_2 + \lceil 997/2 \rfloor m = 525.$$

The ciphertext is $c = (u, v)$.
Decryption. To decrypt c, Bob uses his decryption key s to compute

$$v - s^T u = 948$$

and rounds it to recover the plaintext $m = 0$.

Security. Lindner-Peikert encryption is indistinguishable against chosen-plaintext attack assuming that ss-DLWE is hard. To justify this claim, observe that the encryption operation can be written as

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} A^T \\ b^T \end{bmatrix} r + \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \lceil \frac{q}{2} \rfloor m \end{bmatrix}.$$

Under the ss-DLWE(n, n, q, B) hardness assumption, the matrix

$$\begin{bmatrix} A^T \\ b^T \end{bmatrix}$$

is indistinguishable from a random matrix. Consequently, by the ss-DLWE$(n+1, n, q, B)$ hardness assumption,

$$\begin{bmatrix} A^T \\ b^T \end{bmatrix} r + \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} A^T r + e_1 \\ b^T r + e_2 \end{bmatrix}$$

is also indistinguishable from a random vector. Thus, from the adversary's perspective, the ciphertext component v looks like the sum of a random element $b^T r + e_2 \in \mathbb{Z}_q$ and the plaintext $\lceil \frac{q}{2} \rfloor m$. Therefore, the adversary can gain no information about the plaintext m.

The Lindner-Peikert encryption scheme has two main shortcomings. First, it supports the encryption of only a single bit. Second, it achieves security against chosen-plaintext attacks, but not against chosen-ciphertext attacks (Exercise 4.5). The first issue can be addressed by using the module-LWE variant (introduced in §5), where elements in $\mathbb{Z}_q$ are replaced by polynomials in the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. This modification, which closely resembles the Kyber public-key encryption scheme (§6.1), enables the encryption of n bits simultaneously. The second issue can be resolved by applying the Fujisaki-Okamoto transform, a generic technique that upgrades a public-key encryption scheme from chosen-plaintext security to chosen-ciphertext security. This transformation effectively yields the Kyber key encapsulation mechanism (§6.4).

4.3 The LWE lattice

As with SIS, LWE admits a natural interpretation in terms of lattices.

Definition 4.14 Given an LWE(m, n, q, B) instance (A, b), the associated *LWE lattice* is defined by

$$L_A = \{y \in \mathbb{Z}^m : y = Az \pmod q \text{ for some } z \in \mathbb{Z}^n\}.$$

Let A_1 denote the $n \times n$ submatrix consisting of the first n rows of A. With overwhelming probability over the choice of A, the matrix A_1 is invertible modulo q. Throughout the remainder of this section, we assume that this condition holds.

Theorem 4.15 The LWE lattice $L_A \subseteq \mathbb{R}^m$ is a full-rank integer lattice of volume q^{m-n}.

Proof: As in the proof of Theorem 3.10, it is straightforward to show that L_A is a discrete additive group of $\mathbb{Z}^m$, and hence an integer lattice. Moreover, $q\mathbb{Z}^m$ is a sublattice of L_A, which immediately implies that L_A has full rank.

To determine the volume of L_A, write

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix},$$

where $A_1 \in \mathbb{Z}_q^{n \times n}$ and $A_2 \in \mathbb{Z}_q^{(m-n) \times n}$. Define the matrix $D_2 = A_2 A_1^{-1} \pmod q$, and consider the matrix

$$D = \begin{bmatrix} I_n & 0 \\ D_2 & qI_{m-n} \end{bmatrix} \in \mathbb{Z}^{m \times m}.$$

We claim that the columns of D form a basis for L_A, from which it follows that $\mathrm{vol}(L_A) = |\det(D)| = q^{m-n}$.

Since $\det(D) = q^{m-n} \neq 0$, the columns of D are linearly independent over $\mathbb{R}$. Let $y \in \mathbb{Z}^m$, and decompose it as

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

where $y_1 \in \mathbb{Z}^n$ and $y_2 \in \mathbb{Z}^{m-n}$. By definition, $y \in L_A$ if and only if there exists a vector $z \in \mathbb{Z}^n$ such that $y = Az \pmod q$, which in turn is true if and only if $y_1 = A_1 z$

$(\text{mod } q)$ and $y_2 = A_2 z \pmod q$. Thus, $y \in L_A$ if and only if $y_2 = A_2 A_1^{-1} y_1 \pmod q$, or equivalently, $y_2 = D_2 y_1 + qc$ for some $c \in \mathbb{Z}^{m-n}$. In this case, we can write

$$y = D \begin{bmatrix} y_1 \\ c \end{bmatrix},$$

showing that every vector in L_A is an integer linear combination of the columns of D. It follows that D is indeed a basis matrix for L_A. □

4.4 Hardness of LWE

This section describes three attacks on LWE and briefly discusses the average-case hardness of LWE.

4.4.1 Primal attack

Consider an LWE instance (A, b) with $b = As + e \pmod q$, and define $y = As \bmod q \in L_A$. If we are able to determine y from (A, b), then the secret vector s could be efficiently recovered by solving the overdetermined system of linear equations $As = y \pmod q$.

Since the error vector satisfies $e \in [-B, B]^m$, we have

$$\|b - y\|_2 = \|e\|_2 \le \sqrt{m}\, B.$$

In other words, the lattice vector y is close to the target vector b. This observation naturally leads to a connection between LWE and the Bounded Distance Decoding (BDD) problem, illustrated in Figure 4.5, which seeks a lattice vector that is close to a given target vector.

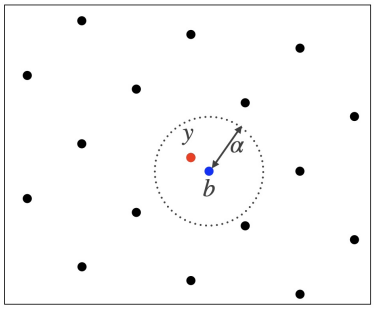


Figure 4.5: The BDD_α problem.

Definition 4.16 The *Bounded Distance Decoding Problem*, denoted BDD_α, is defined as follows: Given a lattice $L = L(D) \subseteq \mathbb{R}^m$ and a vector $b \in \mathbb{R}^m$ with the guarantee that there is a unique $y \in L$ within distance α of b, find y.

BDD_α can be seen as a promise version of CVP_γ (Definition 2.23). We next use *Kannan's embedding technique* (see also Exercise 10.6) to reduce BDD_α to unique SVP (uSVP) under the assumption that

$$\alpha < \lambda_1(L)/\sqrt{2}. \tag{13}$$

This strategy of reducing LWE to an instance of uSVP is known as the *primal attack* on LWE. An example of the primal attack on the short-secret variant of LWE is presented in §10.1.

Define

$$D' = \begin{bmatrix} D & -b \\ 0 & \alpha \end{bmatrix} \in \mathbb{R}^{(m+1)\times(m+1)} \text{ and } L' = L(D') = \left\{ \begin{bmatrix} v - cb \\ c\alpha \end{bmatrix} : v \in L(D) \text{ and } c \in \mathbb{Z} \right\}.$$

Notice that for $(v, c) = (y, 1)$ we obtain the lattice vector

$$\tilde{v} = \begin{bmatrix} y - b \\ \alpha \end{bmatrix} \in L' \text{ with } \|\tilde{v}\|_2 = \sqrt{\|y - b\|_2^2 + \alpha^2} \leq \sqrt{2}\alpha.$$

Thus, $\lambda_1(L') \leq \sqrt{2}\alpha < \lambda_1(L)$ by (13).

Now, let

$$v' = \begin{bmatrix} v - cb \\ c\alpha \end{bmatrix} \in L'$$

be a shortest nonzero vector, i.e., $\|v'\|_2 = \lambda_1(L')$. We can rule out several possibilities for c. If $c = 0$, then

$$\|v'\|_2 = \|v\|_2 \geq \lambda_1(L) > \lambda_1(L'),$$

a contradiction. If $|c| \geq 2$, then

$$\|v'\|_2 \geq 2\alpha > \sqrt{2}\alpha \geq \lambda_1(L'),$$

also a contradiction. Hence, the only possibilities are $c = \pm 1$. Suppose that $c = 1$, so

$$v' = \begin{bmatrix} v - b \\ \alpha \end{bmatrix}$$

for some $v \in L$. (The case $c = -1$ is similar, except for signs.) If $v \neq y$, then $\|v - b\|_2 > \|y - b\|_2$ implying that

$$\|v'\|_2 = \sqrt{\|v - b\|_2^2 + \alpha^2} > \sqrt{\|y - b\|_2^2 + \alpha^2} = \|\tilde{v}\|_2;$$

this contradicts the assumption that v' is a nonzero vector of shortest length in L'. We conclude that $\pm\tilde{v}$ are the only vectors of length $\lambda_1(L')$ in L', and solving SVP in $L'(D)$ immediately yields the BDD solution y.

4.4.2 Dual attack

This attack reduces DLWE to SIS_2 (Definition 3.13), which is then solved using the dual attack described in §3.3.1. Let (A, c) be a DLWE(m, n, q, B) instance, where either $c = b$ or $c = r$ (see Definition 4.4). Select an integer β such that

$$\beta B\sqrt{m} \ll q/4.$$

Apply the SIS_2 dual attack to obtain a nonzero vector $x \in \mathbb{Z}^m$ such that $A^T x = 0 \pmod q$ and $\|x\|_2 \leq \beta$. If (A, c) is an LWE instance, so that $c = As + e \pmod q$ for some error vector $e \in [-B, B]^m$, then

$$c^T x = s^T A^T x + e^T x = e^T x \pmod q.$$

Since both $\|e\|_2 \leq B\sqrt{m}$ and $\|x\|_2 \leq \beta$, the Cauchy-Schwarz inequality[8] implies that

$$|e^T x| \leq \beta B\sqrt{m} \ll q/4.$$

Hence,

$$|c^T x \bmod q| = |e^T x \bmod q| \ll q/4.$$

In contrast, if (A, c) is a random instance, so that c is uniformly random, then one expects that

$$|c^T x \bmod q| \approx q/4.$$

Consequently, the value $v = |c^T x \bmod q|$ can be used to distinguish LWE instances from random ones.

4.4.3 Arora-Ge attack

The Arora-Ge attack on LWE runs in polynomial time when the error bound B is fixed. However, it is inefficient because its running time function is a polynomial of very high degree in n. Furthermore, the attack is ineffective against LWE instances arising in Kyber and Dilithium since it requires a large number of LWE samples ($m \gg n^{2B+1}$), whereas in Kyber and Dilithium one has $m \approx n$.

Suppose that $m \gg n^3$, and let (s, e) be the solution to an LWE(m, n, q, B) instance (A, b). We first treat the case $B = 1$. For each $1 \leq \ell \leq m$, we have

$$a_\ell^T s - b_\ell = e_\ell,$$

where a_ℓ denotes the ℓ-th row of A, and e_ℓ and b_ℓ denote the ℓ-th components of e and b, respectively. Since $e_\ell \in \{-1, 0, 1\}$, it follows that

$$(a_\ell^T s - b_\ell - 1) \cdot (a_\ell^T s - b_\ell) \cdot (a_\ell^T s - b_\ell + 1) = 0. \tag{14}$$

[8]The Cauchy-Schwarz inequality states that $|u^T v| \leq \|u\| \cdot \|v\|$ for all $u, v \in \mathbb{R}^n$.

Viewing the components $s_1, s_2, \ldots, s_n$ of s as variables over $\mathbb{Z}_q$, equation (14) can be expanded into the form

$$\sum_{1 \le i \le j \le k \le n} \alpha_{ijk} s_i s_j s_k + \sum_{1 \le i \le j \le n} \beta_{ij} s_i s_j + \sum_{1 \le i \le n} \gamma_i s_i = c_\ell, \tag{15}$$

where the coefficients α_{ijk}, β_{ij}, γ_i and c_ℓ are known elements of $\mathbb{Z}_q$. This nonlinear system of equations can be solved using the method of *linearization*.

For each $1 \le i \le j \le k \le n$, introduce new variables

$$x_{ijk} = s_i s_j s_k, \quad y_{ij} = s_i s_j, \quad z_i = s_i.$$

Equation (15) then becomes

$$\sum_{1 \le i \le j \le k \le n} \alpha_{ijk} x_{ijk} + \sum_{1 \le i \le j \le n} \beta_{ij} y_{ij} + \sum_{1 \le i \le n} \gamma_i z_i = c_\ell. \tag{16}$$

Notice that (16) is a linear equation in $N = \binom{n+2}{3} + \binom{n+1}{2} + n$ variables, and there is one such equation for each $1 \le \ell \le m$. Since $m \gg n^3 > N$ for all $n \ge 3$, we have an overdetermined system of m linear equations over $\mathbb{Z}_q$ which can be solved efficiently. Since $z_i = s_i$, the solution immediately yields the secret s.

Because the linear algebra step takes cubic time, the running time of the Arora-Ge attack is

$$O(N^3) = O(n^9).$$

More generally, for arbitrary fixed B, the same linearization approach applies with monomials of degree up to $2B + 1$, yielding $O(n^{2B+1})$ variables. The running time is then

$$O((n^{2B+1})^3) = O(n^{6B+3}),$$

provided that $m \gg n^{2B+1}$.

4.4.4 Average-case hardness of LWE

As with SIS, it is reasonable to conjecture that LWE is hard in the *worst case*. However, LWE instances are generated by selecting (A, s, e) uniformly at random. For cryptographic applications, it is therefore crucial to establish *average-case hardness*, namely that LWE is hard for random instances except with negligible probability.

In his seminal 2005 work, Regev proved that LWE is hard *on average*, assuming that approx-SIVP is quantum hard (i.e., intractable on quantum as well as classical computers) in the worst case. This worst-case to average-case reduction provides (conditional) evidence for the average-case hardness of LWE, since it is reasonable to assume that approx-SIVP is quantum hard in the worst case, at least for relatively small approximation factors. As with Ajtai's reduction for SIS (§3.3.2), Regev's result is asymptotic and non-tight, and thus does not directly yield concrete parameter choices for practical cryptosystems.

4.5 Exercises

Exercise 4.1 Write a program to find all solutions (s, e) to the following LWE$(5, 3, 53, 2)$ instance:

$$A = \begin{bmatrix} 24 & 4 & 37 \\ 51 & 41 & 17 \\ 2 & 11 & 13 \\ 44 & 52 & 50 \\ 1 & 0 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 5 \\ 3 \\ 12 \\ 10 \\ 6 \end{bmatrix}.$$

Exercise 4.2 Find all solutions s to the following ss-LWE$(4, 4, 53, 3)$ instance:

$$A = \begin{bmatrix} 5 & 22 & 14 & 47 \\ 45 & 25 & 17 & 49 \\ 39 & 14 & 26 & 51 \\ 31 & 9 & 46 & 24 \end{bmatrix}, \quad b = \begin{bmatrix} 36 & 14 & 46 & 30 \end{bmatrix}.$$

Exercise 4.3 Suppose that $B \ll \frac{q}{2}$. Give a heuristic justification for why the solution to ss-LWE(n, n, q, B) is expected to be unique.

Exercise 4.4 Prove that ss-LWE and ss-DLWE are computationally equivalent.

Exercise 4.5 Show that the Lindner-Peikert public-key encryption scheme is not indistinguishable against chosen-ciphertext attacks.

Exercise 4.6 Consider the Lindner-Peikert public-key encryption scheme with domain parameters $q = 997$, $n = 4$, $B = 5$ and keys

$$A = \begin{bmatrix} 956 & 579 & 385 & 946 \\ 325 & 648 & 268 & 820 \\ 329 & 837 & 955 & 923 \\ 224 & 287 & 608 & 837 \end{bmatrix}, \quad s = \begin{bmatrix} -2 \\ 5 \\ 5 \\ -2 \end{bmatrix}, \quad e = \begin{bmatrix} 4 \\ -2 \\ 1 \\ 0 \end{bmatrix}, \quad b = \begin{bmatrix} 23 \\ 294 \\ 475 \\ 359 \end{bmatrix}.$$

(a) Verify that the encryption of plaintext $m = 1$ using random quantities

$$r = \begin{bmatrix} 2 \\ -1 \\ 4 \\ -4 \end{bmatrix}, \quad e_1 = \begin{bmatrix} 5 \\ -4 \\ 4 \\ 0 \end{bmatrix}, \quad e_2 = -4,$$

yields the ciphertext $c = (u, v)$ where

$$u = \begin{bmatrix} 18 \\ 712 \\ 897 \\ 419 \end{bmatrix} \quad \text{and} \quad v = 711.$$

(b) Decrypt c.

5 Module-SIS and Module-LWE

Contents

The security of Kyber and Dilithium relies on the assumed hardness of the module versions of the SIS and LWE problems. These variants are derived from SIS and LWE by replacing integers modulo q with elements of the polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$.

5.1 The polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$

Let q be a prime and n a positive integer. The *polynomial ring* $R_q = \mathbb{Z}_q[x]/(x^n+1)$ is composed of the polynomials in $\mathbb{Z}_q[x]$ of degree less than n, with multiplication performed using the *reduction polynomial* $x^n + 1$.

To multiply two polynomials $f(x), g(x) \in R_q$, one (i) multiplies them using ordinary polynomial multiplication in $\mathbb{Z}_q[x]$ to obtain a polynomial $h(x) \in \mathbb{Z}_q[x]$ of degree at most $2n - 2$; and then (ii) divides $h(x)$ by $x^n + 1$ to get a remainder polynomial $r(x) \in \mathbb{Z}_q[x]$ of degree at most $n - 1$. The product of $f(x)$ and $g(x)$ in R_q is the remainder polynomial $r(x)$.

Example 5.1 (*polynomial ring*) Let $q = 43$ and $n = 4$. Then $R_q = \mathbb{Z}_{43}[x]/(x^4 + 1)$ is composed of the polynomials in $\mathbb{Z}_{43}[x]$ of degree at most 3.

Let $f(x) = 33 + 41x + 12x^3 \in R_q$ and $g(x) = 2 + 17x + 39x^2 + 14x^3 \in R_q$. Then their sum and difference in R_q are

$$f(x) + g(x) = 35 + 15x + 39x^2 + 26x^3$$

and

$$f(x) - g(x) = 31 + 24x + 4x^2 + 41x^3.$$

The product of f and g in $\mathbb{Z}_{43}[x]$ is the degree-6 polynomial

$$h(x) = 23 + 41x + 6x^2 + 21x^3 + 4x^4 + 38x^5 + 39x^6.$$

The division of $h(x)$ by $x^4 + 1$ can be accomplished by replacing x^4 by -1, x^5 by $-x$, and x^6 by $-x^2$, and then simplifying. We obtain

$$r(x) = 23 + 41x + 6x^2 + 21x^3 - 4 - 38x - 39x^2 = 19 + 3x + 10x^2 + 21x^3.$$

Thus, the product of f and g in R_q is

$$f(x) \times g(x) = r(x) = 19 + 3x + 10x^2 + 21x^3.$$

In §11, we will describe the Number-Theoretic Transform (NTT), a very fast method for performing polynomial multiplication in R_q.

Polynomials as vectors. Let $R_q = \mathbb{Z}_q[x]/(x^n+1)$. A polynomial $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} \in R_q$ can be represented by its length-n vector of coefficients $f = (a_0, a_1, a_2, \ldots, a_{n-1})$.

Example 5.2 (*representing polynomials as vectors*) Let $q = 31$ and $n = 5$, so $R_q = \mathbb{Z}_{31}[x]/(x^5+1)$. Let $f(x) = 12 + 3x + 4x^3 + 30x^4 \in R_q$ and $g(x) = 7 + 13x + 22x^2 + 27x^3 + 28x^4 \in R_q$. The vector representations of $f(x)$ and $g(x)$ are

$$f = (12, 3, 0, 4, 30) \text{ and } g = (7, 13, 22, 27, 28).$$

One can verify that the vector representations of the sum, difference, and product of $f(x)$ and $g(x)$ in R_q are

$$f + g = (19, 16, 22, 0, 27), \quad f - g = (5, 21, 9, 8, 2), \text{ and } f \times g = (18, 29, 1, 12, 28).$$

Polynomial multiplication via matrices. Polynomial multiplication in R_q can be expressed as a matrix-vector product. Let $a(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} \in R_q$ and $b(x) = b_0 + b_1x + b_2x^2 + \cdots + b_{n-1}x^{n-1} \in R_q$, and let $c(x) = a(x) \times b(x)$ in R_q. Then

$$\begin{aligned} c(x) &= a(x)(b_0 + b_1x + b_2x^2 + \cdots + b_{n-1}x^{n-1}) \\ &= b_0a(x) + b_1x \cdot a(x) + b_2x^2 \cdot a(x) + \cdots + b_{n-1}x^{n-1} \cdot a(x). \end{aligned} \tag{17}$$

Now,

$$\begin{aligned} x \cdot a(x) &= a_0x + a_1x^2 + \cdots + a_{n-2}x^{n-1} + a_{n-1}x^n \\ &= -a_{n-1} + a_0x + a_1x^2 + \cdots + a_{n-2}x^{n-1} \pmod{x^n+1} \\ &\leftrightarrow (-a_{n-1}, a_0, a_1, \ldots, a_{n-2}) \end{aligned}$$

since $x^n = -1 \pmod{x^n+1}$. Thus, the vector representation of $x \cdot a(x)$ is the right cyclic shift of the vector representation of $a(x)$, with the cycled element negated. It follows from (17) that if $c(x) = a(x) \times b(x)$ in R_q, then

$$\underbrace{\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix}}_{c} = \underbrace{\begin{bmatrix} a_0 & -a_{n-1} & -a_{n-2} & \cdots & -a_1 \\ a_1 & a_0 & -a_{n-1} & \cdots & -a_2 \\ a_2 & a_1 & a_0 & \cdots & -a_3 \\ \vdots & \vdots & \vdots & & \vdots \\ a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_0 \end{bmatrix}}_{\overline{\text{circ}}(a)} \underbrace{\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{bmatrix}}_{b}.$$

The $n \times n$ matrix above is an *anti-circulant matrix*, denoted $A = \overline{\text{circ}}(a)$.

Example 5.3 (*matrix representation of R_q multiplication*) Let $q = 17$ and $n = 5$, so $R_q = \mathbb{Z}_{17}[x]/(x^5 + 1)$. Let $a(x) = 11 + 2x + 9x^3 + 13x^4$, $b(x) = 5 + 3x + 14x^2 + 7x^3 + 15x^4 \in R_q$. Then the vector representation of $c(x) = a(x)b(x)$ is

$$c = \underbrace{\begin{bmatrix} 11 & 4 & 8 & 0 & 15 \\ 2 & 11 & 4 & 8 & 0 \\ 0 & 2 & 11 & 4 & 8 \\ 9 & 0 & 2 & 11 & 4 \\ 13 & 9 & 0 & 2 & 11 \end{bmatrix}}_{\overline{\text{circ}}(a)} \underbrace{\begin{bmatrix} 5 \\ 3 \\ 14 \\ 7 \\ 15 \end{bmatrix}}_{b} = \begin{bmatrix} 13 \\ 2 \\ 2 \\ 6 \\ 16 \end{bmatrix}$$

Hence, $a(x)b(x) = 13 + 2x + 2x^2 + 6x^3 + 16x^4$ in R_q.

We will next define a notion of "size" for integers in $\mathbb{Z}_q$ and polynomials in R_q.

Definition 5.4 Let $q \geq 3$ be an odd integer. The *size* of an integer $r \in \mathbb{Z}_q$, also called the *infinity norm* of r, is $\|r\|_\infty = |r \bmod q|$ where $|\cdot|$ denotes absolute value.

Note that $0 \leq \|r\|_\infty \leq (q-1)/2$. For example, if $q = 17$ then $0 \leq \|r\|_\infty \leq 8$. We have $\|6\|_\infty = 6$ and $\|13\|_\infty = 4$.

Definition 5.5 The *size* of a polynomial $f(x) = f_0 + f_1 x + f_2 x^2 + \cdots + f_{n-1}x^{n-1} \in R_q$ is $\|f\|_\infty = \max_i \|f_i\|_\infty$, also called the *infinity norm* of $f(x)$.

Definition 5.6 Let η be a positive integer that is small compared to $q/2$. The set of *small polynomials* in R_q is $S_\eta = \{f \in R_q \mid \|f\|_\infty \leq \eta\}$.

The product of small polynomials is also (relatively) small, as shown next.

Theorem 5.7 Fix $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. Let η_1 and η_2 be positive integers such that $n\eta_1\eta_2 < q/2$. Let $f \in S_{\eta_1}$ and $g \in S_{\eta_2}$. Then $fg \in S_{n\eta_1\eta_2}$.

Proof: Consider $f(x) = f_0 + f_1 x + \cdots + f_{n-1}x^{n-1} \in S_{\eta_1}$ and $g(x) = g_0 + g_1 x + \cdots + g_{n-1}x^{n-1} \in S_{\eta_2}$. Let $h(x) = h_0 + h_1 x + \cdots + h_{n-1}x^{n-1}$ be the product of $f(x)$ and $g(x)$ in R_q. By multiplying $f(x)$ and $g(x)$, then replacing x^{n+j} by $-x^j$ for $0 \leq j \leq n-2$, and then simplifying, one can observe that the coefficient h_i of x^i in $h(x)$ for each $i \in [0, n-1]$ is:

$$h_i = f_0 g_i + f_1 g_{i-1} + \cdots + f_i g_0 - f_{i+1}g_{n-1} - f_{i+2}g_{n-2} - \cdots - f_{n-2}g_{i+2} - f_{n-1}g_{i+1}. \quad (18)$$

Hence, $\|h_i\|_\infty \leq n\eta_1\eta_2$, whence $\|h\|_\infty \leq n\eta_1\eta_2$. □

Example 5.8 (*small polynomials*) Let $q = 379$ and $n = 5$ so $R_q = \mathbb{Z}_{379}[x]/(x^5 + 1)$. Let $f(x) = 378 + x^2 + 2x^3 + 377x^4 \in R_q$. The mods q representation of f is $f(x) = -1 + x^2 + 2x^3 - 2x^4$ so $f \in S_2$ is a small polynomial (with respect to $\eta_1 = 2$). Similarly, let $g(x) = 3 + 376x + 2x^2 + 378x^3 + 377x^4 \in S_3$ be a small polynomial (with respect to $\eta_2 = 3$). Then the product of f and g in R_q is $h(x) = 367 + 11x + 3x^2 + 371x^4$ which is in S_{12} since its mods q representation is $h(x) = -12 + 11x + 3x^2 - 8x^4$. Note that $12 \leq n\eta_1\eta_2 = 30 < q/2$.

5.2 Module-SIS (MSIS)

A *module* extends the notion of a vector space by replacing the vector space's field of scalars with a ring. The module of interest here is R_q^k, which consists of length-k vectors of polynomials in R_q. The *size* of module element $a = (a_1, a_2, \ldots, a_k) \in R_q^k$ is defined by

$$\|a\|_\infty = \max\{\|a_i\|_\infty \ : \ 1 \leq i \leq k\},$$

where the size $\|a_i\|_\infty$ of polynomial $a_i \in R_q$ is given in Definition 5.5. Addition and subtraction of elements in R_q^k are performed componentwise, ensuring that the result remains within R_q^k. The *inner product* of two vectors $a = (a_1, a_2, \ldots, a_k)$ and $b = (b_1, b_2, \ldots, b_k)$ in R_q^k is defined to be

$$a^T \cdot b = a_1b_1 + a_2b_2 + \cdots + a_kb_k,$$

which is a polynomial in R_q.

Example 5.9 (*module*) Let $q = 31$, $n = 5$ (so $R_q = \mathbb{Z}_{31}[x]/(x^5 + 1)$) and $k = 3$. Let $a = (a_1, a_2, a_3)$ and $b = (b_1, b_2, b_3)$ be:

$$a = \begin{bmatrix} 12 + 7x + x^3 + 29x^4 \\ 3 + 17x + 22x^2 + 9x^3 + 16x^4 \\ 14 + 8x + 29x^2 + 7x^3 + 4x^4 \end{bmatrix} \quad \text{and} \quad b = \begin{bmatrix} 25 + 12x + 5x^2 + 23x^3 + 18x^4 \\ 3 + 29x + x^2 + 4x^3 + 7x^4 \\ 3 + 24x + 14x^2 + 4x^3 + 7x^4 \end{bmatrix} \in R_q^3.$$

Then

$$a + b = \begin{bmatrix} 6 + 19x + 5x^2 + 24x^3 + 16x^4 \\ 6 + 15x + 23x^2 + 13x^3 + 23x^4 \\ 17 + x + 12x^2 + 11x^3 + 11x^4 \end{bmatrix}, \quad a - b = \begin{bmatrix} 18 + 26x + 26x^2 + 9x^3 + 11x^4 \\ 19x + 21x^2 + 5x^3 + 9x^4 \\ 11 + 15x + 15x^2 + 3x^3 + 28x^4 \end{bmatrix},$$

and

$$a^T \cdot b = a_1b_1 + a_2b_2 + a_3b_3 = 4 + x + 25x^2 + 13x^3 + 18x^4.$$

MSIS is a variant of SIS in which elements of $\mathbb{Z}_q$ are replaced by polynomials in the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. Given a set of randomly selected elements in the module R_q^k, MSIS involves finding a linear combination of the polynomial vectors that equals 0, with the scalar multipliers of the linear combination being small polynomials.

Definition 5.10 The *Module Short Integer Solution* problem MSIS(k, ℓ, n, q, B) is the following. Given $A \in_R R_q^{k\times\ell}$, find $z \in R_q^\ell$ such that $Az = 0$ in R_q^k, where $z \neq 0$ and $\|z\|_\infty \leq B$. Here, $k < \ell$ and $B \ll q/2$.

Note that an MSIS solution is not unique. Indeed, if $z = (z_1, z_2, \ldots, z_\ell)$ is one MSIS solution, then so is $(xz_1, xz_2, \ldots, xz_\ell)$. Thus, an MSIS solution yields as many as $2n - 1$ other distinct solutions.

Example 5.11 (*MSIS instance*) Let $k = 2$, $\ell = 3$, $n = 4$, $q = 71$ (so $R_q = \mathbb{Z}_{71}[x]/(x^4 + 1)$), and $B = 10$. Consider the MSIS instance

$$A = \begin{bmatrix} 16 + 58x^2 + 30x^3 & 22 + 40x + 4x^2 + 57x^3 & 53 + 8x + 21x^2 + 38x^3 \\ 61 + 32x + 3x^2 + 45x^3 & 23 + 5x + 58x^2 + 40x^3 & 49 + 40x + 29x^2 + 54x^3 \end{bmatrix} \in R_q^{2\times 3}.$$

Given A, the task is to find nonzero $z = (z_1, z_2, z_3) \in R_q^3$ such that $Az = 0$ and $\|z\|_\infty \leq 10$.

Recall that polynomial multiplication in R_q can be expressed as a matrix-vector product. Consequently, an equivalent formulation of this MSIS instance is to solve $\overline{A}\overline{z} = 0 \pmod{71}$ for nonzero $\overline{z} \in [-10, 10]^{12}$, where

$$\overline{A} = \left[\begin{array}{cccc|cccc|cccc} 16 & 41 & 13 & 0 & 22 & 14 & 67 & 31 & 53 & 33 & 50 & 63 \\ 0 & 16 & 41 & 13 & 40 & 22 & 14 & 67 & 8 & 53 & 33 & 50 \\ 58 & 0 & 16 & 41 & 4 & 40 & 22 & 14 & 21 & 8 & 53 & 33 \\ 30 & 58 & 0 & 16 & 57 & 4 & 40 & 22 & 38 & 21 & 8 & 53 \\ \hline 61 & 26 & 68 & 39 & 23 & 31 & 13 & 66 & 49 & 17 & 42 & 31 \\ 32 & 61 & 26 & 68 & 5 & 23 & 31 & 13 & 40 & 49 & 17 & 42 \\ 3 & 32 & 61 & 26 & 58 & 5 & 23 & 31 & 29 & 40 & 49 & 17 \\ 45 & 3 & 32 & 61 & 40 & 58 & 5 & 23 & 54 & 29 & 40 & 49 \end{array}\right]_{8\times 12}.$$

The 4×4 blocks in $\overline{A}$ are the anti-circulant matrices corresponding to the polynomials in A.

Gaussian elimination $\pmod{71}$ on $\overline{A}$ yields the following rank-8 matrix in reduced form:

$$\tilde{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 37 & 4 & 48 & 8 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 63 & 37 & 4 & 48 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 23 & 63 & 37 & 4 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 67 & 23 & 63 & 37 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 52 & 19 & 47 & 54 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 17 & 52 & 19 & 47 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 24 & 17 & 52 & 19 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 52 & 24 & 17 & 52 \end{bmatrix}.$$

Since the null space of $\overline{A}$ has dimension 4, the total number of solutions $\overline{z}$ to $\overline{A}\overline{z} = 0 \pmod{71}$ is $71^4 = 25,411,681$. Checking all these solutions $\overline{z}$ yields 16 nonzero solutions all of whose coordinates are in $[-10, 10]$. The two MSIS solutions, up to multiplication by $\pm 1, \pm x, \pm x^2, \pm x^3$, are $\overline{z} = (0, -5, 9, 1, -10, 4, -10, 10, -9, 5, 9, -6)$ and $\overline{z} =$

$(3, -10, -7, -4, 5, 0, 1, 5, 4, -9, 6, -6)$. The polynomial form of the first solution is $z = (-5x + 9x^2 + x^3,\ -10 + 4x - 10x^2 + 10x^3,\ -9 + 5x + 9x^2 - 6x^3)$.

We next define the inhomogeneous MSIS problem and its normal-form variant. Both of these problems are equivalent to MSIS (Exercises 5.3 and 5.4).

Definition 5.12 The *Module Inhomogeneous Short Integer Solution* problem MISIS(k, ℓ, n, q, B) is the following. Given $A \in_R R_q^{k\times\ell}$ and $b \in_R R_q^k$, find $z \in R_q^\ell$ such that $Az = b$, where $z \neq 0$ and $\|z\|_\infty \leq B$. Here, $k < \ell$ and $B \ll q/2$.

Definition 5.13 The *normal-form MISIS* problem nf-MISIS(k, ℓ, n, q, B) is the following. Given $A \in_R R_q^{k\times\ell}$ and $b \in_R R_q^k$, find $z \in R_q^{k+\ell}$ such that $[A \mid I_k]z = b$ and $\|z\|_\infty \leq B$.

5.3 Module-LWE (MLWE)

MLWE is a variant of LWE in which integers modulo q are replaced by polynomials in the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$.

Definition 5.14 The *Module Learning With Errors* problem MLWE(k, ℓ, n, q, B) is the following. Let $A \in_R R_q^{k\times\ell}$, $s \in_R R_q^\ell$ and $e \in_R S_B^k$. Here, $k > \ell$, $B \ll q/2$, and $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. Given A and $t = As + e \in R_q^k$, find s.

The columns $A_1, A_2, \ldots, A_\ell$ of A belong to the module R_q^k. MLWE asks to express the module element t, up to a small error e, as a linear combination of the A_i, where the coefficients of the linear combination are polynomials.

Example 5.15 (*MLWE instance*) Let $k = 3$, $\ell = 2$, $n = 4$, $q = 37$ (so $R_q = \mathbb{Z}_{37}[x]/(x^4+1)$), and $B = 1$. Randomly select

$$A = \begin{bmatrix} 33 + 34x^2 + 8x^3 & 21 + 10x + 19x^2 + 18x^3 \\ 3 + 13x + 19x^2 + 18x^3 & 34 + 19x + 15x^2 + 4x^3 \\ 16 + 6x + 5x^2 + 7x^3 & 24 + 6x + 35x^2 + 31x^3 \end{bmatrix} \in R_q^{3\times 2},$$

$$s = \begin{bmatrix} 32 + 15x + 16x^2 + 3x^3 \\ 10 + 28x + 6x^2 + 2x^3 \end{bmatrix} \in R_q^2, \quad \text{and } e = \begin{bmatrix} 1 - x \\ 1 - x^2 - x^3 \\ -1 + x - x^3 \end{bmatrix} \in S_1^3,$$

and define t to be

$$t = As + e = \begin{bmatrix} 2 + 29x + 6x^2 + 14x^3 \\ 28 + 20x + 9x^2 + 17x^3 \\ 13 + 5x + 12x^2 + 3x^3 \end{bmatrix} \in R_q^3.$$

Given (A, t), the MLWE challenge is to find $s \in R_q^2$ and $e \in S_1^3$ such that $As + e = t$.

Recall that polynomial multiplication in R_q can be expressed as a matrix-vector product. Consequently, an equivalent formulation of this MLWE instance is to solve $\overline{A}\overline{s}+\overline{e} = \overline{t} \pmod{37}$ for $\overline{s} \in \mathbb{Z}_{37}^8$ and $\overline{e} \in [-1,1]^{12}$, where

$$\overline{A} = \left[\begin{array}{cccc|cccc} 33 & 29 & 3 & 0 & 21 & 19 & 18 & 27 \\ 0 & 33 & 29 & 3 & 10 & 21 & 19 & 18 \\ 34 & 0 & 33 & 29 & 19 & 10 & 21 & 19 \\ 8 & 34 & 0 & 33 & 18 & 19 & 10 & 21 \\ \hline 3 & 19 & 18 & 24 & 34 & 33 & 22 & 18 \\ 13 & 3 & 19 & 18 & 19 & 34 & 33 & 22 \\ 19 & 13 & 3 & 19 & 15 & 19 & 34 & 33 \\ 18 & 19 & 13 & 3 & 4 & 15 & 19 & 34 \\ \hline 16 & 30 & 32 & 31 & 24 & 6 & 2 & 31 \\ 6 & 16 & 30 & 32 & 6 & 24 & 6 & 2 \\ 5 & 6 & 16 & 30 & 35 & 6 & 24 & 6 \\ 7 & 5 & 6 & 16 & 31 & 35 & 6 & 24 \end{array}\right]_{12\times 8} \quad \text{and} \quad \overline{t} = \left[\begin{array}{c} 2 \\ 29 \\ 6 \\ 14 \\ \hline 28 \\ 20 \\ 9 \\ 17 \\ \hline 13 \\ 5 \\ 12 \\ 3 \end{array}\right]_{12\times 1}.$$

The 4×4 blocks in $\overline{A}$ are the anti-circulant matrices corresponding to the polynomials in A, whereas the 4×1 blocks in $\overline{t}$ are the vector representations of the polynomials in t.

As it turns out, there are two MLWE solutions:

$$\overline{s} = (17, 4, 36, 30, 35, 31, 13, 12), \quad \overline{e} = (-1, 0, 1, 0, -1, -1, 1, 0, 0, -1, -1, -1);$$

and

$$\overline{s} = (32, 15, 16, 3, 10, 28, 6, 2), \quad \overline{e} = (1, -1, 0, 0, 1, 0, -1, -1, -1, 1, 0, -1).$$

The polynomial form of the first solution is $s = (17 + 4x + 36x^2 + 30x^3,\ 35 + 31x + 13x^2 + 12x^3)$, $e = (-1 + x^2,\ -1 - x + x^2,\ -x - x^2 - x^3)$. The second solution is the (s, e) pair that used to construct this MLWE instance.

The decisional variant D-MLWE of MLWE is defined next. Analogously to Theorem 4.5, one can show that MLWE and D-MLWE are computationally equivalent (Exercise 5.7).

Definition 5.16 The *Decisional Module Learning With Errors* problem D-MLWE(k, ℓ, n, q, B) is the following. Let $A \in_R R_q^{k\times\ell}$, $s \in_R R_q^{\ell}$, $e \in_R S_B^k$, $r \in_R R_q^k$, and let $t = As + e \in R_q^k$. Let $z = t$ with probability $\frac{1}{2}$ and $z = r$ with probability $\frac{1}{2}$. Given A and z, decide (with success probability significantly greater than $\frac{1}{2}$) whether $z = t$ or $z = r$.

In the short-secret variants of MLWE and D-MLWE, the secret polynomial vector s is required to have small coefficients. Each of these variants is equivalent to MLWE (Exercises 5.8 and 5.9).

Definition 5.17 The *short-secret Module Learning With Errors* problem ss-MLWE$(k, \ell, n, q, \eta_1, \eta_2)$ is the following. Let $A \in_R R_q^{k\times\ell}$, $s \in_R S_{\eta_1}^{\ell}$ and $e \in_R S_{\eta_2}^k$. Here, $k \geq \ell$ and $\eta_1, \eta_2 \ll q/2$. Given A and $t = As + e \in R_q^k$, find s.

Definition 5.18 The *short-secret Decisional Module Learning With Errors* problem ss-D-MLWE$(k, \ell, n, q, \eta_1, \eta_2)$ is the following. Let $A \in_R R_q^{k \times \ell}$, $s \in_R S_{\eta_1}^{\ell}$, $e \in_R S_{\eta_2}^{k}$, $r \in_R R_q^k$, and let $t = As + e \in R_q^k$. Let $z = t$ with probability $\frac{1}{2}$ and $z = r$ with probability $\frac{1}{2}$. Given A and z, decide (with success probability significantly greater than $\frac{1}{2}$) whether $z = t$ or $z = r$.

5.4 Exercises

Exercise 5.1 Let $R_q = \mathbb{Z}_{677}[x]/(x^4 + 1)$, and let $a(x) = 625 + 436x + 84x^3 \in R_q$ and $b(x) = 166 + 285x + 376x^2 + 282x^3 \in R_q$. Verify the following computations in R_q:
(a) $a(x) + b(x) = 114 + 44x + 376x^2 + 366x^3$.
(b) $a(x) - b(x) = 459 + 151x + 301x^2 + 479x^3$.
(c) $a(x) \cdot b(x) = 186 + 246x + 457x^2 + 59x^3$.

Exercise 5.2 Write a program to find all solutions z to the following MSIS$(2, 3, 4, 83, 10)$ instance:

$$A = \begin{bmatrix} 68 + 68x^2 + 54x^3 & 13 + 30x + 75x^2 + 18x^3 & 24 + 16x + 68x^2 + 79x^3 \\ 36 + 19x + 70x^2 + 36x^3 & 3 + 63x + 42x^2 + 53x^3 & 81x + 13x^2 + 45x^3 \end{bmatrix}.$$

Exercise 5.3 Prove that MSIS and MISIS are computationally equivalent.

Exercise 5.4 Prove that nf-MISIS$(k, \ell, n, q, B) \leq$ MISIS$(k, k + \ell, n, q, B)$ and MISIS$(k, k + \ell, n, q, B) \leq$ nf-MISIS(k, ℓ, n, q, B).

Exercise 5.5 Write a program to find all solutions (s, e) to the following MLWE$(3, 2, 4, 37, 1)$ instance:

$$A = \begin{bmatrix} 32x^2 + 11x^3 & 14 + 13x + 22x^2 + 4x^3 \\ 5 + 11x + 14x^2 + 17x^3 & 7 + 6x + 5x^2 + 20x^3 \\ 21 + 10x + 5x^2 + 7x^3 & 6x + 5x^2 + 14x^3 \end{bmatrix}, \quad t = \begin{bmatrix} 2 + 31x + 18x^2 + 27x^3 \\ 3 + 18x + 14x^2 + 8x^3 \\ 5 + 27x^2 + 31x^3 \end{bmatrix}.$$

Exercise 5.6 Give a heuristic justification for why the solution to MLWE(k, ℓ, n, q, B) where $k > \ell$ and $B \ll q/2$ is expected to be unique.

Exercise 5.7 Prove that MLWE and D-MLWE are computationally equivalent.

Exercise 5.8 Prove that MLWE and ss-MLWE are computationally equivalent.

Exercise 5.9 Prove that ss-MLWE and ss-D-MLWE are computationally equivalent.

6 Kyber (ML-KEM)

Contents

Kyber-KEM is a quantum-safe key encapsulation mechanism (KEM) standardized by NIST in FIPS 203, where its official name is Module-Lattice-based KEM (ML-KEM). It is designed to offer strong security guarantees against attacks from both classical and quantum computers. Kyber-KEM is based on the Kyber public-key encryption scheme (Kyber-PKE), described next.

6.1 Kyber public-key encryption

The Kyber public-key encryption scheme can be viewed as a natural extension of the Lindner-Peikert scheme described in §4.2, with elements in $\mathbb{Z}_q$ replaced by polynomials in the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$.

Notation. Recall that $S_\eta = \{f \in R_q \mid \|f\|_\infty \leq \eta\}$, and that $\lceil x \rfloor$ denotes the nearest integer to x with ties broken upward. The plaintext space is $\{0,1\}^n$. A plaintext $m = (m_0, m_1, \ldots, m_{n-1}) \in \{0,1\}^n$ is associated with the polynomial $m(x) = m_0 + m_1 x + \cdots + m_{n-1}x^{n-1} \in R_q$. The rounding function Round_q (Definition 4.11) extends naturally to polynomials in R_q by applying it coefficientwise. For instance, if $q = 3329$ then $\mathrm{Round}_q(100 + 1000x + 2000x^2 + 3000x^3) = x + x^2$.

The Kyber public-key encryption scheme without ciphertext compression is described in Algorithm 6.1. The standardized version, which incorporates ciphertext compression, is presented in §6.3. The three Kyber parameter sets specified in FIPS 203 are listed in Table 6.1. Their intended security levels against classical attacks are approximately 128, 192 and 256 bits.

Example 6.2 (*Kyber-PKE*)
Domain parameters. $q = 991$, $n = 3$, $k = 3$, $\eta_1 = 3$, and $\eta_2 = 3$.
Key generation. Bob selects

$$A = \begin{bmatrix} 329 + 26x + 34x^2 & 295 + 745x + 292x^2 & 732 + 248x + 641x^2 \\ 833 + 849x + 647x^2 & 396 + 363x + 924x^2 & 312 + 803x + 31x^2 \\ 176 + 501x + 802x^2 & 738 + 941x + 521x^2 & 19 + 582x + 224x^2 \end{bmatrix},$$

Parameter	Description	ML-KEM -512	ML-KEM -768	ML-KEM -1024
q	prime modulus	3329	3329	3329
n	$R_q = \mathbb{Z}_q[x]/(x^n + 1)$	256	256	256
k	dimensions of A	2	3	4
η_1	coefficient bound for s, e, r	3	2	2
η_2	coefficient bound for e_1, e_2	2	2	2
(d_u, d_v)	compression parameters	(10,4)	(10,4)	(11,5)
λ	security level	≈ 128	≈ 192	≈ 256

Table 6.1: FIPS 203 parameter sets for Kyber-PKE and Kyber-KEM.

Algorithm 6.1: Kyber public-key enc. scheme (without ciphertext compression)

Domain parameters. q, n, k, η_1, η_2.

Key generation. Bob does the following:

1 Select $A \in_R R_q^{k \times k}$, $s \in_R S_{\eta_1}^k$, and $e \in_R S_{\eta_1}^k$.

2 Compute $t = As + e$.

3 Bob's encryption (public) key is (A, t); his decryption (private) key is s.

Encryption. To encrypt a message $m \in \{0,1\}^n$ for Bob, Alice does the following:

4 Obtain an authentic copy of Bob's encryption key (A, t).

5 Select $r \in_R S_{\eta_1}^k$, $e_1 \in_R S_{\eta_2}^k$, and $e_2 \in_R S_{\eta_2}$.

6 Compute $u = A^T r + e_1$ and $v = t^T r + e_2 + \lceil \frac{q}{2} \rfloor m$ (cf. Figure 4.4).

7 Output $c = (u, v)$.

Decryption. To decrypt $c = (u, v)$, Bob does:

8 Compute $m = \text{Round}_q(v - s^T u)$.

$$s = \begin{bmatrix} 1+3x^2 \\ 1+x \\ -2+2x^2 \end{bmatrix}, \text{ and } e = \begin{bmatrix} -2+3x-x^2 \\ 1-x+2x^2 \\ 3+x-x^2 \end{bmatrix},$$

and computes

$$t = As + e = \begin{bmatrix} 274+180x+257x^2 \\ 484+971x+42x^2 \\ 664+145x+399x^2 \end{bmatrix}.$$

Bob's encryption key is (A, t), and his decryption key is s.
Encryption. To encrypt the plaintext message $m = 110 \leftrightarrow 1 + x$ for Bob, Alice selects

$$r = \begin{bmatrix} -2-2x+x^2 \\ 3x^2 \\ -1-x+3x^2 \end{bmatrix}, \quad e_1 = \begin{bmatrix} 3-2x+2x^2 \\ 2-3x-x^2 \\ -3+x-2x^2 \end{bmatrix}, \quad \text{and } e_2 = 3-3x+2x^2$$

and computes

$$u = A^T r + e_1 = \begin{bmatrix} 918+176x+944x^2 \\ 77+530x+160x^2 \\ 572+989x+130x^2 \end{bmatrix}$$

and

$$v = t^T r + e_2 + \lceil 991/2 \rfloor m = 636 + 169x + 320x^2.$$

The ciphertext is $c = (u, v)$.
Decryption. To decrypt c, Bob uses his decryption key s to compute

$$v - s^T u = 478 + 492x + 3x^2$$

and rounds its coefficients to obtain $1 + x$, thereby recovering the plaintext $m = 110$.

Correctness of decryption. Decryption succeeds precisely when $\text{Round}_q(v - s^T u) = m$. Now, substituting $u = A^T r + e_1$ and $v = t^T r + e_2 + \lceil \frac{q}{2} \rfloor m$ into $v - s^T u$ yields

$$\begin{aligned} v - s^T u &= (t^T r + e_2 + \lceil \frac{q}{2} \rfloor m) - s^T (A^T r + e_1) \\ &= (s^T A^T + e^T) r + e_2 + \lceil \frac{q}{2} \rfloor m - s^T (A^T r + e_1) \qquad \text{(after replacing } t^T \text{ by } s^T A^T + e^T) \\ &= e^T r - s^T e_1 + e_2 + \lceil \frac{q}{2} \rfloor m. \end{aligned}$$

Thus, $\text{Round}_q(v - s^T u) = m$ if and only if each coefficient E_i of the *decryption error polynomial*

$$E(x) = e^T r - s^T e_1 + e_2 \tag{19}$$

satisfies $-q/4 < E_i \bmod q < q/4$, i.e., $\|E\|_\infty < q/4$. Notice that e, e_1, e_2, r, and s all have small coefficients, so one can hope that $E(x)$ also has relatively small coefficients

(cf. Theorem 5.7). Indeed, since $\|e\|_\infty \leq \eta_1$ and $\|r\|_\infty \leq \eta_1$, we have $\|e^T r\|_\infty \leq kn\eta_1^2$ if $kn\eta_1^2 < q/2$. Similarly, $\|s^T e_1\|_\infty \leq kn\eta_1\eta_2$ if $kn\eta_1\eta_2 < q/2$. Thus, if

$$kn\eta_1^2 + kn\eta_1\eta_2 + \eta_2 < q/4, \tag{20}$$

then $\|E_i\|_\infty \leq kn\eta_1^2 + kn\eta_1\eta_2 + \eta_2 < q/4$ and decryption succeeds.

Example 6.3 (*decryption success probability*) The ML-KEM-512 parameters (Table 6.1) are $q = 3329$, $n = 256$, $k = 2$, $\eta_1 = 3$, $\eta_2 = 2$. For these parameters, $kn\eta_1^2 + kn\eta_1\eta_2 + \eta_2 = 7682 \not< \frac{q}{4}$, and hence decryption is not guaranteed to succeed. However, significant cancellation is expected when computing $E(x) = e^T r - s^T e_1 + e_2$ so its coefficients are expected to be substantially smaller than the pessimistic bound above. Indeed, we will show in §6.2 that the probability that $\|E\|_\infty > \frac{q}{4}$ is less than 2^{-233}. Hence, the probability of decryption failure is negligibly small.

Security. The security of Kyber-PKE is captured in the following theorem.

Theorem 6.4 The Kyber public-key encryption scheme is semantically secure against chosen-plaintext attack assuming that ss-D-MLWE is intractable.

Proof: The encryption operation can be written as

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} A^T \\ t^T \end{bmatrix} r + \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \lceil \frac{q}{2} \rfloor m \end{bmatrix}.$$

By the ss-D-MLWE$(k, k, n, q, \eta_1, \eta_1)$ intractability assumption,

$$\begin{bmatrix} A^T \\ t^T \end{bmatrix}$$

is indistinguishable from random. Next, by the ss-D-MLWE$(k+1, k, n, q, \eta_1, \eta_2)$ intractability assumption,

$$\begin{bmatrix} A^T \\ t^T \end{bmatrix} r + \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} A^T r + e_1 \\ t^T r + e_2 \end{bmatrix} \tag{21}$$

is indistinguishable from random. Thus, from the adversary's perspective, v is the sum of the random polynomial $t^T r + e_2$ and the scaled message polynomial $\lceil \frac{q}{2} \rfloor m$. It is also clear that u is independent of m, and thus leaks nothing about m. Hence, the adversary learns nothing about m. □

6.2 Decryption error probability

In this section, we analyze the probability of decryption failure in Kyber (without ciphertext compression).

6.2.1 Centered binomial distributions

One detail not mentioned in the description of Kyber in §6.1 is that all coefficients of polynomials in the vectors s, e, r, e_1, and in the polynomial e_2 are sampled from centered binomial distributions instead of uniform distributions.

Let $S \subseteq \mathbb{Z}$ be a finite set, and let X be a discrete random variable taking values in S. For each $j \in S$, let p_j denote the probability that $X = j$. The *generating series* of X is

$$\Phi_X(z) = \sum_{j \in S} p_j z^j.$$

The coefficient of z^j in $\Phi_X(z)$ is precisely p_j, which we denote using the coefficient operator $[\cdot]$ by

$$p_j = [z^j]\, \Phi_X(z).$$

Now, let Y be another random variable taking values in S, and suppose that X and Y are independent. Then the generating series of the sum $X + Y$ is

$$\Phi_{X+Y}(z) = \Phi_X(z) \times \Phi_Y(z). \tag{22}$$

Definition 6.5 For $\eta \geq 1$, define the random variable

$$C_\eta = \sum_{i=1}^{\eta} (a_i - b_i), \tag{23}$$

where the a_i and b_i are independent random variables, each uniformly distributed over $\{0, 1\}$. Then C_η is said to follow the *centered binomial distribution with parameter* η, written

$$C_\eta \sim \mathrm{CBD}_\eta.$$

The support of C_η is $\{-\eta, \ldots, \eta\}$, and the distribution is symmetric, i.e.,

$$\Pr(C_\eta = j) = \Pr(C_\eta = -j) \;\text{ for }\; -\eta \leq j \leq \eta.$$

Theorem 6.6 Let $C_\eta \sim \mathrm{CBD}_\eta$. Then $E(C_\eta) = 0$ and $E(C_\eta^2) = \eta/2$. Moreover,

$$\Pr(C_\eta = j) = \binom{2\eta}{\eta + j} / 2^{2\eta} \;\text{ for all }\; -\eta \leq j \leq \eta. \tag{24}$$

Proof: Since the distribution of C_η is symmetric about 0, we have $E(C_\eta) = 0$.

Let A_η and B_η be independent binomial random variables, each corresponding to the sum of η independent Bernoulli trials with success probability $p = \frac{1}{2}$. Recall that

$$E(A_\eta) = E(B_\eta) = \eta p = \frac{\eta}{2} \;\text{ and }\; \mathrm{Var}(A_\eta) = \mathrm{Var}(B_\eta) = \eta p(1-p) = \frac{\eta}{4}.$$

Since $C_\eta = A_\eta - B_\eta$, we obtain

$$\begin{aligned} E(C_\eta^2) &= E((A_\eta - B_\eta)^2) = E(A_\eta^2) - 2E(A_\eta)E(B_\eta) + E(B_\eta^2) \\ &= 2E(A_\eta^2) - 2E(A_\eta)^2 = 2\mathrm{Var}(A_\eta) = \frac{\eta}{2}. \end{aligned}$$

Now, for each $1 \le i \le \eta$, the generating series of a_i and $-b_i$ are

$$\Phi_{a_i}(z) = \frac{1}{2} + \frac{z}{2} \quad \text{and} \quad \Phi_{-b_i}(z) = \frac{1}{2} + \frac{1}{2z},$$

respectively. Hence by (22), the generating series of $C_\eta = A_\eta - B_\eta$ is

$$\Phi_{C_\eta}(z) = \left(\left(\frac{1}{2} + \frac{z}{2}\right)\left(\frac{1}{2} + \frac{1}{2z}\right)\right)^\eta = \frac{z^{-\eta}(z+1)^{2\eta}}{2^{2\eta}}.$$

Thus, for $0 \le j \le \eta$, we have

$$\Pr(C_\eta = j) = [z^j]\,\Phi_{C_\eta}(z) = [z^j]\frac{z^{-\eta}(z+1)^{2\eta}}{2^{2\eta}} = [z^{\eta+j}]\frac{(z+1)^{2\eta}}{2^{2\eta}} = \binom{2\eta}{\eta+j}/2^{2\eta}.$$

The result for negative j follows by symmetry. □

In the standardized version of Kyber, the coefficients of the polynomials in the vectors s, e and r are sampled from the centered binomial distribution with parameter η_1, rather than uniformly from $[-\eta_1, \eta_1]$; we denote this by

$$s, e, r \in_{CBD} S_{\eta_1}^k. \tag{25}$$

Similarly, the coefficients of the polynomials in e_1 and the polynomial e_2 are sampled from the centered binomial distribution with parameter η_2, so that

$$e_1 \in_{CBD} S_{\eta_2}^k \text{ and } e_2 \in_{CBD} S_{\eta_2}. \tag{26}$$

Kyber uses the centered binomial distribution as its noise distribution for three main reasons. First, it is efficient to sample in practice: each sample can be generated using only a small number of uniform random bits (the a_i and b_i) and simple integer operations (additions and subtractions), avoiding costly floating-point arithmetic (required for Gaussian sampling) and rejection sampling (needed for uniform sampling in intervals $[-\eta, \eta]$). This makes it well-suited for constant-time implementations. Second, the distribution places higher probability on values close to 0, which helps reduce the probability of decryption errors. Third, it provides a good approximation to a discrete Gaussian, the standard choice in theoretical security analyses in lattice-based cryptography.

6.2.2 Computing the decryption error probability

Recall that the decryption error polynomial is $E(x) = e^T r - s^T e_1 + e_2$, and that decryption is correct precisely when $\|E\|_\infty < q/4$. For each $0 \le i \le n-1$, let δ_i be the probability

that the coefficient E_i of x^i in $E(x)$ satisfies $\|E_i\|_\infty > q/4$. Note that the δ_i's are all equal, so we will denote them by δ. We will derive an upper bound δ' on δ.

Let $A \sim \mathrm{CBD}_{\eta_1}$, $A' \sim \mathrm{CBD}_{\eta_1}$, and $B \sim \mathrm{CBD}_{\eta_2}$ be random variables. Define the random variables $C = AA'$ and $D = AB$ with supports contained in $[-\eta_1^2, \eta_1^2]$ and $[-\eta_1\eta_2, \eta_1\eta_2]$, respectively. Let $f \in_{CBD} S_{\eta_1}$ and $g \in_{CBD} S_{\eta_1}$, and consider their product in $\mathbb{Z}[x]$, i.e., without applying the mods q operation to the coefficients. Then each coefficient of the product is a random variable $\widehat{C} = \sum_{j=1}^{n} C_j$, where each C_j is an independent and identically distributed (i.i.d.) copy of C. Similarly, each coefficient of the product of polynomials $f \in_{CBD} S_{\eta_1}$ and $g \in_{CBD} S_{\eta_2}$ in $\mathbb{Z}[x]$ is a random variable $\widehat{D} = \sum_{j=1}^{n} D_j$, where each D_j is an i.i.d. copy of D. It follows that when $E(x)$ is considered a polynomial in $\mathbb{Z}[x]$, each coefficient is a random variable

$$E_i = \sum_{\ell=1}^{k} \widehat{C}_\ell + \sum_{\ell=1}^{k} \widehat{D}_\ell + B, \tag{27}$$

where the $\widehat{C}_\ell$ and $\widehat{D}_\ell$ are i.i.d. copies of $\widehat{C}$ and $\widehat{D}$. The three terms on the right-hand side of (27) correspond to the coefficients of x^i in $e^T r$, $-s^T e_1$ and e_2, respectively. Notice that the support of E_i is $[-kn\eta_1^2 - kn\eta_1\eta_2 - \eta_2,\ kn\eta_1^2 + kn\eta_1\eta_2 + \eta_2]$. Thus, the generating series of E_i is

$$\Phi_{E_i}(z) = \Phi_{\widehat{C}}(z)^k \cdot \Phi_{\widehat{D}}(z)^k \cdot \Phi_B(z) = \Phi_C(z)^{kn} \cdot \Phi_D(z)^{kn} \cdot \Phi_B(z). \tag{28}$$

Therefore, an upper bound on δ is

$$\delta \le \delta' = 1 - \sum_{j=\lceil -q/4 \rceil}^{\lfloor q/4 \rfloor} [z^j]\, \Phi_{E_i}(z). \tag{29}$$

By the union bound, an upper bound on the decryption error probability p is

$$p \le n\delta \le n\delta'.$$

6.2.3 ML-KEM-512 decryption error probability

The ML-KEM-512 parameters (see Table 6.1) are

$$q = 3329, \quad n = 256, \quad k = 2, \quad \eta_1 = 3, \quad \eta_2 = 2.$$

Let $A \sim \mathrm{CBD}_3$, $A' \sim \mathrm{CBD}_3$, and $B \sim \mathrm{CBD}_2$ be random variables, and let $C = AA'$ and $D = AB$. The probability distributions of A, B, C and D are shown in Table 6.2. Hence, the generating series of B, C and D are:

$$\Phi_B(z) = \frac{1}{16z^2} + \frac{1}{4z} + \frac{3}{8} + \frac{z}{4} + \frac{z^2}{16},$$

$$\Phi_C(z) = \frac{1}{2048z^9} + \frac{3}{512z^6} + \frac{9}{512z^4} + \frac{15}{1024z^3} + \frac{45}{512z^2} + \frac{225}{2048z} + \frac{135}{256} + \frac{225z}{2048} + \frac{45z^2}{512} + \frac{15z^3}{1024} + \frac{9z^4}{512} + \frac{3z^6}{512} + \frac{z^9}{2048},$$

$$\Phi_D(z) = \frac{1}{512z^6} + \frac{3}{256z^4} + \frac{1}{128z^3} + \frac{39}{512z^2} + \frac{15}{128z} + \frac{73}{128} + \frac{15z}{128} + \frac{39z^2}{512} + \frac{z^3}{128} + \frac{3z^4}{256} + \frac{z^6}{512}.$$

A	-3	-2	-1	0	1	2	3
$64 \cdot \Pr(A)$	1	6	15	20	15	6	1

B	-2	-1	0	1	2
$16 \cdot \Pr(B)$	1	4	6	4	1

C	-9	-6	-4	-3	-2	-1	0	1	2	3	4	6	9
$2048 \cdot \Pr(C)$	1	12	36	30	180	225	1080	225	180	30	36	12	1

D	-6	-4	-3	-2	-1	0	1	2	3	4	6
$1024 \cdot \Pr(D)$	2	12	8	78	120	584	120	78	8	12	2

Table 6.2: Probability distributions of A, B, C and D for the ML-KEM-512 parameters.

Therefore, the generating series of each E_i is

$$\Phi_{E_i}(z) = \Phi_C(z)^{512} \cdot \Phi_D(z)^{512} \cdot \Phi_B(z)$$

and an upper bound on δ is

$$\delta \leq \delta' = 1 - \sum_{j=-832}^{832} [z^j] \, \Phi_{E_i}(z).$$

Finally, an upper bound on p is

$$p \leq 256\delta'.$$

The Maple computer algebra system was used to extract coefficients from the generating series $\Phi_{E_i}(z)$, yielding the upper bound

$$p \leq 2^{-233}$$

on the decryption error probability for Kyber (without ciphertext compression) when used with the ML-KEM-512 parameters.

Similarly, upper bounds on the decryption error probability for Kyber (without ciphertext compression) when used with the ML-KEM-768 and ML-KEM-1024 parameter sets are $p \leq 2^{-301}$ and $p \leq 2^{-231}$, respectively.

6.3 Ciphertext compression

The FIPS 203 standard specifies a method for compressing and decompressing Kyber-PKE ciphertexts. The approach involves discarding the "low-order" bits of the coefficients of all polynomials in the ciphertext $c = (u, v)$. This compression is lossy, but the compression parameters were carefully selected to ensure that the errors introduced during decompression do not significantly affect the decryption error probability.

For a *compression parameter* $d \in [1, \lfloor \log_2 q \rfloor]$ and integer $X \in [0, q-1]$, define

$$\text{Compress}_q(X, d) = \lceil (2^d/q) \cdot X \rfloor \bmod 2^d. \tag{30}$$

For $Y \in [0, 2^d - 1]$, define

$$\text{Decompress}_q(Y, d) = \lceil (q/2^d) \cdot Y \rfloor \bmod q. \tag{31}$$

The decompression error is quantified next.

Theorem 6.7 Consider compression parameter $d \in [1, \lfloor \log_2 q \rfloor]$. Let $X \in [0, q-1]$, $Y = \text{Compress}_q(X, d)$, and $X' = \text{Decompress}_q(Y, d)$. Then $\|X' - X\|_\infty \leq \lceil q/2^{d+1} \rfloor$.

Example 6.8 Let $q = 19$ and $d = 2$. For each $X \in [0, 18]$, Figure 6.1 shows the $Y = \text{Compress}_{19}(X, 2)$ values (in blue) and the $X' = \text{Decompress}_{19}(Y, 2)$ values (in red).

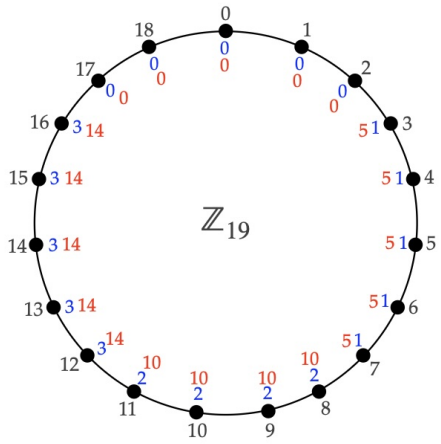


Figure 6.1: Compression with $q = 19$ and $d = 2$.

The functions Compress and Decompress extend to polynomials in R_q and polynomial vectors in R_q^k by applying them to each coefficient of a polynomial.

Example 6.9 Let $q = 3329$. If $X \in [0, q-1]$, $Y = \text{Compress}_q(X, 10)$, and $X' = \text{Decompress}_q(Y, 10)$, then $\|X' - X\|_\infty \leq 2$. Also, if $Y = \text{Compress}_q(X, 4)$, and $X' = \text{Decompress}_q(Y, 4)$, then $\|X' - X\|_\infty \leq 104$.

For example, if

$$U = 2426 + 1198x + 2055x^2 + 2409x^3 \in \mathbb{Z}_q[x]/(x^4 + 1),$$

then

$$V = \text{Compress}_q(U, 10) = 746 + 369x + 632x^2 + 741x^3$$

and

$$U' = \text{Decompress}_q(V, 10) = 2425 + 1200x + 2055x^2 + 2409x^3.$$

We have $U - U' = 1 - 2x$.

Also,

$$W = \text{Compress}_q(U, 4) = 12 + 6x + 10x^2 + 12x^3,$$

$$U'' = \text{Decompress}_q(W, 4) = 2497 + 1248x + 2081x^2 + 2497x^3,$$

and $U - U'' = -71 - 50x - 26x^2 - 88x^3$.

The Kyber public-key encryption scheme with ciphertext compression is described in Algorithm 6.10. The *compression parameters* d_u and d_v specified in FIPS 203 are listed in Table 6.1. Ciphertext compression reduces the ciphertext size from 1152, 1536, and 1920 bytes for the ML-KEM-512, ML-KEM-768, and ML-KEM-1024 parameter sets, respectively, to 768, 1088, and 1568 bytes.

Algorithm 6.10: Kyber public-key enc. scheme (with ciphertext compression)

Domain parameters. q, n, k, η_1, η_2, d_u, d_v.

Key generation. Bob does the following:

1 Select $A \in_R R_q^{k \times k}$, $s \in_{CBD} S_{\eta_1}^k$, and $e \in_{CBD} S_{\eta_1}^k$.
2 Compute $t = As + e$.
3 Bob's encryption (public) key is (A, t); his decryption (private) key is s.

Encryption. To encrypt a message $m \in \{0,1\}^n$ for Bob, Alice does the following:

4 Obtain an authentic copy of Bob's encryption key (A, t).
5 Select $r \in_{CBD} S_{\eta_1}^k$, $e_1 \in_{CBD} S_{\eta_2}^k$, and $e_2 \in_{CBD} S_{\eta_2}$.
6 Compute $u = A^T r + e_1$ and $v = t^T r + e_2 + \lceil \frac{q}{2} \rfloor m$ (cf. Figure 4.4).
7 Compute $c_1 = \text{Compress}_q(u, d_u)$ and $c_2 = \text{Compress}_q(v, d_v)$.
8 Output $c = (c_1, c_2)$.

Decryption. To decrypt $c = (c_1, c_2)$, Bob does:

9 Compute $u' = \text{Decompress}_q(c_1, d_u)$ and $v' = \text{Decompress}_q(c_2, d_v)$.
10 Compute $m = \text{Round}_q(v' - s^T u')$.

Correctness of decryption. Ciphertext compression introduces additional noise into the decryption procedure. Decryption succeeds precisely when $m = \text{Round}_q(v' - s^T u')$. Let $e_u = u' - u \in R_q^k$ and $e_v = v' - v \in R_q$ denote the compression errors. Then

$$v' - s^T u' = (v + e_v) - s^T(u + e_u) = e^T r - s^T e_1 + e_2 - s^T e_u + e_v + \lceil \frac{q}{2} \rfloor m.$$

Hence, decryption is correct if and only if every coefficient E_i of the *decryption error polynomial*

$$E(x) = e^T r - s^T e_1 + e_2 - s^T e_u + e_v \tag{32}$$

satisfies $-q/4 < E_i \bmod q < q/4$, i.e., $\|E\|_\infty < q/4$. The probability that this condition holds can be estimated in the same manner as in §6.2.

Assume that the quantities u and v are uniformly distributed over R_q^k and R_q, respectively. Let C_u and C_v denote the random variables capturing the compression error of a uniformly random element of R_q under parameters d_u and d_v. Let $F = AC_u$, where $A \sim \mathrm{CBD}_{\eta_1}$. Then, analogously to (28), the generating series of the random variable E_i is

$$\Phi_{E_i}(z) = \Phi_C(z)^{kn} \cdot \Phi_D(z)^{kn} \cdot \Phi_B(z) \cdot \Phi_F(z)^{kn} \cdot \Phi_{C_v}(z), \tag{33}$$

where the factors $\Phi_F(z)^{kn}$ and $\Phi_{C_v}(z)$ correspond to the contributions of $-s^T e_u$ and e_v in (32). Consequently, an upper bound on the decryption error probability is given by $p \le n\delta'$ where

$$\delta' = 1 - \sum_{j=\lceil -q/4 \rceil}^{\lfloor q/4 \rfloor} [z^j]\, \Phi_{E_i}(z).$$

ML-KEM-512 decryption error probability. The ML-KEM-512 parameters (see Table 6.1) are

$$q = 3329, \quad n = 256, \quad k = 2, \quad \eta_1 = 3, \quad \eta_2 = 2, \quad d_u = 10, \quad d_v = 4.$$

The probability distributions of C_u, C_v, and F are given in Table 6.3. This yields the

C_u	-2	-1	0	1	2
$3329 \cdot \Pr(C_u)$	129	1024	1024	1024	128

C_v	-104	-103	-102	$\cdots$	102	103	104
$3329 \cdot \Pr(C_v)$	9	16	16	$\cdots$	16	16	8

F	-6	-4	-3	-2	-1	0
$64 \cdot 3329 \cdot \Pr(F)$	257	1542	2048	16143	30720	11636

F	1	2	3	4	6
$64 \cdot 3329 \cdot \Pr(F)$	30720	16143	2048	1542	257

Table 6.3: Probability distributions of C_u, C_v and F for the ML-KEM-512 parameters.

following upper bound on the decryption error probability for Kyber with ciphertext compression:

$$p \le 2^{-141}.$$

Using the same approach, upper bounds on the decryption error probability for Kyber with ciphertext compression when used with the ML-KEM-768 and ML-KEM-1024 parameter sets are $p \le 2^{-168}$ and $p \le 2^{-177}$, respectively.

Remark 6.11 (*selecting Kyber parameters*) The ML-KEM-512, ML-KEM-768 and ML-KEM-1024 parameter sets for Kyber specified in FIPS 203 (see Table 6.1) were carefully chosen to balance security, public key size, ciphertext size, decryption error probability, and implementation efficiency. The following notes illustrate the delicate balance required when selecting the parameters n, q, k, η_1, η_2, d_u and d_v.

1. To facilitate fast polynomial multiplication in R_q via the Number-Theoretic Transform (see §11.4), the degree n was chosen to be a power of two, and the prime q was chosen to satisfy the condition $q \equiv 1 \pmod{n}$.

2. A larger modulus q reduces the decryption error probability, but at the expense of larger public keys and ciphertexts.

3. The parameter k can be increased to achieve higher levels of security against MLWE attacks without altering the underlying polynomial ring R_q. However, increasing k also increases the sizes of both public keys and ciphertexts.

4. As demonstrated by the decryption error polynomial in (32), smaller secret and error bounds η_1 and η_2 reduce the decryption error probability. Conversely, decreasing η_1 and η_2 also makes the underlying MLWE problem easier.

5. Smaller compression parameters d_u and d_v yield smaller ciphertexts, but their use increases the decryption error probability.

The impact of the parameters n, q, k, η_1 and η_2 on the concrete security of Kyber is analyzed further in §9.6.

6.4 Kyber-KEM

The official name of Kyber-KEM in FIPS 203 is Module-Lattice-based KEM (ML-KEM). As shown in Algorithm 6.12, Kyber-KEM is derived by applying a variant of the Fujisaki-Okamoto (FO) transform to the Kyber public-key encryption scheme. The FO transform is a generic method for converting a public-key encryption scheme that is secure against chosen-plaintext attacks to one that is secure against chosen-ciphertext attacks.

The transform utilizes three hash functions, G, H, and J, which are derived from SHA3 and SHAKE256. Kyber-PKE is employed to encrypt a randomly chosen bit string m of length 256. During encapsulation, *derandomization* is applied, whereby m and the encapsulation key ek are hashed to produce a random seed R and the secret key K. The random polynomial vectors r, e_1 and the random polynomial e_2 required for Kyber-PKE are derived from R. This process transforms the randomized Kyber public-key encryption scheme into a deterministic one, meaning the ciphertext c depends only on the message m and encapsulation key ek, without relying on any additional random bits generated during encryption.

In decapsulation, the recipient decrypts the Kyber-PKE ciphertext c to recover m', then hashes m' and ek to obtain R' and K'. He then re-encrypts m' using R' and compares the resulting ciphertext c' with the received c. If the encryption was performed correctly, then m' will equal m with overwhelming probability and consequently $R' = R$ and $K' = K$. This ensures that the re-encryption of m results in the ciphertext c, which is why a deterministic encryption scheme is necessary. However, if decryption fails and $m' \neq m$, the re-encrypted ciphertext c' will differ from c with overwhelming probability.

Therefore, if c' equals c, Bob accepts K as the shared key. Otherwise, he generates a random key $\overline{K}$ obtained by hashing c and the secret z; in this case the decapsulation is deemed to have failed.

Kyber-KEM exhibits *plaintext awareness*, meaning that decapsulation will produce K (and not $\overline{K}$) only if the entity who performed the encapsulation knew K. This property is designed to ensure resistance to chosen-ciphertext attacks because the adversary cannot gain any useful information by submitting a ciphertext to Bob for decapsulation. Either the adversary already knew K (in which case nothing new is learned when Bob returns the same K), or the adversary does not know K, in which case Bob will almost certainly return a key $\overline{K}$ that is randomly selected and independent of K, preventing the adversary from learning anything useful.

Decapsulation fails when c does not equal c', in which case the key $\overline{K}$ outputted is different from the encapsulated key K. This failure can occur even if Alice and Bob behave honestly, as there is a very small chance that a failure in the underlying Kyber-PKE will cause m' to differ from m. However, for the domain parameters specified in FIPS 203, the probability of Kyber-PKE decryption failure is negligible, and consequently the probability of a Kyber-KEM failure is also extremely low.

Decapsulation error probability. Kyber-KEM decapsulation fails if $\overline{K}$ is returned. The probability p of this event coincides with the decryption error probability for Kyber encryption, namely $p \leq 2^{-141}$, 2^{-168}, and 2^{-177} for the ML-KEM-512, ML-KEM-768, and ML-KEM-1024 parameter sets, respectively. In comparison, the targeted security levels for these parameter sets are approximately 128, 192 and 256 bits. Ideally, the decryption error probabilities for ML-KEM-768 and ML-KEM-1024 would be closer to 2^{-192} and 2^{-256} in view of the following potential chosen-ciphertext attack.

Decapsulation fails when at least one coefficient E_i of the error polynomial $E(x) = e^T r - s^T e_1 + e_2 - s^T e_u + e_v$ satisfies $\|E_i\|_\infty > q/4$. An adversary selects a random m, computes the corresponding ciphertext c, and submits c to a decapsulation oracle. This procedure is repeated until a decapsulation error occurs. Upon such a failure, the adversary learns *some* information about the secret (s, e); specifically, the adversary has learned values r, e_1, e_2, e_u and e_v such that $\|E(x)\|_\infty > q/4$.

For the ML-KEM-768 parameter set, an adversary can make $t = 2^{64}$ queries to the decapsulation oracle and expects to observe at least one failure with probability $\varepsilon = 2^{-104}$. In this setting, we have $t/\varepsilon = 2^{168}$ which is below the target security level of 2^{192}. However, it is not known how to effectively exploit the information leaked by a decapsulation failure to break the KEM, i.e., to distinguish between (K, C) and $(\overline{K}, C)$ where (K, C) is a valid key-ciphertext pair and $\overline{K}$ is an independently generated random key.

Some strategies have been proposed to increase the probability of decapsulation failure. For example, an adversary might repeatedly sample random m until the corresponding values r and e_1 have unusually large norms, thereby increasing the likelihood of failure. However, no such strategies are currently known to be effective in breaking Kyber-

Algorithm 6.12: Kyber key encapsulation mechanism.

Domain parameters. Kyber-PKE parameters q, $n = 256$, k, η_1, η_2, and hash functions $G : \{0,1\}^* \to \{0,1\}^{2n}$, $H : \{0,1\}^* \to \{0,1\}^n$, $J : \{0,1\}^* \to \{0,1\}^n$.

Key generation. Bob does the following:

1 Use the Kyber-PKE key generation algorithm to select a Kyber-PKE encryption key (A, t) and decryption key s.
2 Select $z \in_R \{0,1\}^{256}$.
3 Bob's encapsulation (public) key is $ek = (A, t)$; his decapsulation (private) key is $dk = (s, ek, H(ek), z)$.

Encapsulation. To establish an n-bit shared secret key with Bob, Alice does:

4 Obtain an authentic copy of Bob's encapsulation key ek.
5 Select $m \in_R \{0,1\}^{256}$.
6 Compute $h = H(ek)$ and $(K, R) = G(m, h)$ where $K, R \in \{0,1\}^{256}$.
7 Use the Kyber-PKE encryption algorithm to encrypt m with ek, and using R to derive the random quantities needed for encryption; the resulting ciphertext is c.
8 Output the secret key K and ciphertext c.

Decapsulation. To recover the secret key K from c using $dk = (s, ek, H(ek), z)$, Bob does the following:

9 Use the Kyber-PKE decryption algorithm to decrypt c using decryption key s; the resulting plaintext is m'.
10 Compute $(K', R') = G(m', H(ek))$.
11 Compute $\overline{K} = J(z, c)$.
12 Use the Kyber-PKE enc. algorithm to encrypt m' with ek, and using R' to derive the random quantities needed for encryption; the resulting ciphertext is c'.
13 If $c \neq c'$ then return($\overline{K}$).
14 Return(K').

KEM, especially when the number of decapsulation queries is limited to 2^{64}, which is a realistic upper bound in practice.

Security. Kyber-KEM has been proven to be indistinguishable against chosen-ciphertext attacks assuming that D-MLWE is intractable, and that G, H, and J are modeled as random oracles. Kyber-KEM has also been proven indistinguishable against chosen-ciphertext attacks by a quantum adversary, who is also able to make quantum queries (in superposition) to G, H and J.

6.5 Exercises

Exercise 6.1 Consider an instance of Kyber-PKE with domain parameters $q = 937$, $n = 3$, $k = 3$, $\eta_1 = 3$ and $\eta_2 = 3$. Bob's encryption key is (A, t) where

$$A = \begin{bmatrix} 849 + 691x + 562x^2 & 523 + 589x + 133x^2 & 326 + 726x + 311x^2 \\ 553 + 157x + 10x^2 & 301 + 135x + 449x^2 & 777 + 905x + 49x^2 \\ 806 + 516x + 661x^2 & 540 + 37x + 438x^2 & 533 + 675x + 676x^2 \end{bmatrix}$$

and

$$t = \begin{bmatrix} 889 + 194x + 301x^2 \\ 597 + 204x + 448x^2 \\ 707 + 348x + 50x^2 \end{bmatrix},$$

and his decryption key is

$$s = \begin{bmatrix} -2 - 3x - 2x^2 \\ 2 + x \\ -3 + 3x - 2x^2 \end{bmatrix}.$$

To encrypt the plaintext $m = 011$ for Bob, Alice selects

$$r = \begin{bmatrix} 1 + 2x - x^2 \\ 1 + 3x - 2x^2 \\ -3 - 2x \end{bmatrix}, \quad e_1 = \begin{bmatrix} 2 - x + 2x^2 \\ 1 \\ -2 + 3x^2 \end{bmatrix} \quad \text{and} \quad e_2 = 1 - 3x + 3x^2.$$

(a) Verify that the ciphertext that Alice computes is $c = (u, v)$ where

$$u = \begin{bmatrix} 159 + 689x + 268x^2 \\ 264 + 639x + 589x^2 \\ 747 + 58x + 209x^2 \end{bmatrix} \quad \text{and} \quad v = 933 + 361x + 229x^2.$$

(b) Verify that decryption of c by Bob will recover m.

Exercise 6.2 Show that the Kyber public-key encryption scheme is not indistinguishable against chosen-ciphertext attacks.

Exercise 6.3 Prove Theorem 6.7.

Exercise 6.4 Let c be a Kyber-PKE ciphertext generated using public key (A, t) and randomness (r, e_1, e_2). Show that the plaintext m can be efficiently determined by an adversary who knows (A, t), c, r, e_1 and e_2.

Exercise 6.5 Show that a Kyber-KEM ciphertext (with compression) has size 768, 1088, and 1568 bytes, when used with the ML-KEM-512, ML-KEM-768, and ML-KEM-1024 parameter sets, respectively.

7 Dilithium (ML-DSA)

Contents

Dilithium is a quantum-safe signature scheme standardized by NIST in FIPS 204, where it is officially called the Module-Lattice-based Digital Signature Algorithm (ML-DSA). It is designed to provide strong security guarantees against attacks from both classical and quantum computers. In §7.1, we introduce a "toy" version of Dilithium to illustrate the main ideas underlying the scheme. We then examine several weaknesses of this simplified scheme and address them in §7.2. Finally, we incorporate public key compression in §7.3 to obtain the full version of Dilithium.

7.1 Dilithium (toy version)

The design of Dilithium is inspired by the Schnorr signature scheme, which we briefly recall next.

Schnorr signature scheme. Let g be a generator of a cyclic group G of prime order n, and let H be a hash function. Alice's secret signing key is an integer $a \in_R [1, n-1]$, and her corresponding public verification key is the group element $A = g^a$. Observe that recovering a from A is precisely the discrete logarithm problem in G.

To sign a message $M \in \{0,1\}^*$, Alice first selects an integer $y \in_R [1, n-1]$ and then computes $w = g^y$, $c = H(M\|w)$, and $z = y + ca \bmod n$. The value w is called the *commitment*, c is the *challenge*, and z is the *response*. Alice's signature on M is the pair $\sigma = (c, z)$. To verify a purported signature, Bob uses Alice's public key A to compute $w' = g^z A^{-c}$, and accepts the signature if and only if $c = H(M\|w')$.

Correctness follows by observing that

$$z \equiv y + ca \pmod{n} \;\Leftrightarrow\; g^z = g^{y+ca} \;\Leftrightarrow\; g^y = g^z (g^a)^{-c} \;\Leftrightarrow\; w = g^z A^{-c}.$$

Therefore $w' = w$, so Bob and Alice compute the same challenge value.

In our initial description of Dilithium, we work with parameters q, n, k, ℓ, η, γ_1 and τ. Alice randomly selects a matrix $A \in_R R_q^{k\times\ell}$ whose entries are polynomials in $R_q = \mathbb{Z}_q[x]/(x^n+1)$, together with two small polynomial vectors $s_1 \in_R S_\eta^\ell$ and $s_2 \in_R S_\eta^k$. (Recall that S_η denotes the set of polynomials $f \in R_q$ satisfying $\|f\|_\infty \le \eta$.) She then computes $t = As_1 + s_2 \in R_q^k$. Alice's public verification key is (A, t), and her secret signing key is (s_1, s_2). Observe that recovering the secret key from the public key corresponds to an instance of ss-MLWE, the short-secret variant of MLWE (Definition 5.17).

To sign a message $M \in \{0,1\}^*$, Alice first selects a secret vector $y \in_R S_{\gamma_1}^\ell$ and then computes the *commitment* $w = Ay \in R_q^k$. Here, γ_1 is chosen to be significantly smaller

than $q/2$, so that recovering y from w corresponds to an instance of the inhomogeneous variant of MSIS (Definition 5.12). Alice then computes the *challenge* $c = H(M\|w)$, where $H : \{0,1\}^* \longrightarrow B_\tau$ is a hash function whose output is interpreted as an element of B_τ. The set B_τ comprises all polynomials in S_1 having exactly τ coefficients (mods q) equal to ± 1; note that $|B_\tau| = 2^\tau \binom{256}{\tau}$. Finally, Alice computes the *response* $z = y + cs_1 \in R_q^\ell$. Alice's signature on M is $\sigma = (c, z)$.

The polynomial vector cs_1 is designed to mask the secret y, even though both c and z are revealed. Let $\beta = \tau\eta$. The parameters γ_1, η and τ are chosen so that $\gamma_1 + \beta < q/2$. Since $c \in B_\tau$ and $s_1 \in S_\eta^\ell$, it follows that $\|cs_1\|_\infty \leq \beta$. Furthermore, $\|z\|_\infty = \|y + cs_1\|_\infty \leq \gamma_1 + \beta < q/2$.

To verify Alice's signature $\sigma = (c, z)$ on a message M, Bob, who possesses Alice's public key (A, t), must reconstruct the commitment w and then check whether $c = H(M\|w)$. Multiplying both sides of $z = y + cs_1$ by A, we obtain

$$Az = Ay + c(As_1) = w + c(t - s_2),$$

which can be rearranged as

$$Az - ct = w - cs_2. \tag{34}$$

Thus, Bob can compute $w - cs_2$, although not w itself. However, note that the coefficients of cs_2 are small, specifically $\|cs_2\|_\infty \leq \beta$. To overcome the difficulty of Bob being unable to compute w, we will introduce the notions of "HighBits" and "LowBits" applied to the mods q representation of an integer. These operations, which will be explained in §7.2, are analogous to the "most significant bits" and "least significant bits" in the standard binary representation of an integer. Using this approach, one then defines HighBits(g) and LowBits(g) for a polynomial vector g by applying the HighBits and LowBits operations to all coefficients of all polynomials in g.

The signing procedure is then adjusted as follows to yield our toy version of Dilithium. Alice repeatedly selects $y \in_R S_{\gamma_1}^\ell$ until the LowBits of all coefficients of the polynomials in $w - cs_2$ are "sufficiently small" (this notion will be made precise later). This ensures that adding cs_2 to $w - cs_2$ does not affect any HighBits, so that HighBits$(w - cs_2)$ = HighBits(w). She then computes the commitment $w_1 =$ HighBits(w), the challenge $c = H(M\|w_1)$, and the response $z = y + cs_1$. As before, her signature on M is $\sigma = (c, z)$. The verifier, Bob, is now able to recover the commitment w_1 since

$$w_1 = \text{HighBits}(w) = \text{HighBits}(w - cs_2) = \text{HighBits}(Az - ct),$$

and can therefore check that the challenge was correctly derived from M and w_1. The resulting signature scheme is presented in Algorithm 7.1. Table 7.1 lists the standardized FIPS 204 parameter sets. Their intended security levels against classical attacks are approximately 128, 192 and 256 bits.

In Algorithm 7.1, the vector y is drawn from $\tilde{S}_{\gamma_1}^\ell$, the set of polynomial vectors in R_q^ℓ all of whose coefficients lie in the interval $(-\gamma_1, \gamma_1]$. Notice that the endpoint $-\gamma_1$ is excluded from this interval. Therefore, each coefficient admits exactly $2 \cdot \gamma_1$ possible

Parameter	Description	ML-DSA-44	ML-DSA-65	ML-DSA-87
q	prime modulus	$2^{23} - 2^{13} + 1$	$2^{23} - 2^{13} + 1$	$2^{23} - 2^{13} + 1$
n	$R_q = \mathbb{Z}_q[x]/(x^n + 1)$	256	256	256
(k, ℓ)	dimensions of A	(4,4)	(6,5)	(8,7)
η	coefficient bound for s_1, s_2	2	4	2
γ_1	coefficient bound for y	2^{17}	2^{19}	2^{19}
τ	number of 1's in c	39	49	60
β	$\beta = \tau\eta$	78	196	120
γ_2	HighBits/LowBits param.	$(q-1)/88$	$(q-1)/32$	$(q-1)/32$
α	$\alpha = 2\gamma_2$	$(q-1)/44$	$(q-1)/16$	$(q-1)/16$
λ	collision strength of $\tilde{c}$	128	192	256
d	# of dropped bits from t	13	13	13
ω	max # of 1's in hint vector	80	55	75
Λ	security level	≈ 128	≈ 192	≈ 256

Table 7.1: FIPS 204 parameter sets for Dilithium (ML-DSA).

values, which is a power of two when $\gamma_1 = 2^{17}$ or $\gamma_1 = 2^{19}$, as specified in Table 7.1. As a result, the coefficients of candidate polynomial vectors y can be generated efficiently by sampling uniformly random binary strings of length 17 or 19, respectively.

Next, we examine five issues with the toy version of Dilithium and modifications that address them.

Issue #1: Large public keys. For the ML-DSA-87 parameters, the matrix $A \in R_q^{k \times \ell}$ has size 41,216 bytes, and the vector $t \in R_q^k$ has size 5,888 bytes, so the public key (A, t) has size 47,104 bytes. To reduce this, A is generated from a randomly selected 256-bit seed ρ, and its entries are derived from ρ using a publicly known pseudorandom bit generator called ExpandA. The updated public key, (ρ, t), has size 5,920 bytes.

FIPS 204 also specifies a method for compressing t, reducing its size from 5,888 bytes to 2,560 bytes, further reducing the public key size to 2,592 bytes.

Issue #2: Large private keys. For the ML-DSA-87 parameters, the secret key $(s_1, s_2) \in S_\eta^\ell \times S_\eta^k$ has a size of 1,440 bytes. To reduce this, s_1 and s_2 are generated from a randomly selected 512-bit secret seed ρ' using a pseudorandom bit generator called ExpandS.

Issue #3: Repeated hashing of M. If the message M is very large, then the hash operation $H(M\|w_1)$ in step 8 will be the most time-consuming operation in signature generation. Moreover, step 8 might be executed several times. To reduce the time cost, a message representative μ is first computed before the while loop by hashing M, and then μ is used in place of M in step 8.

Algorithm 7.1: Dilithium signature scheme (toy version)

Domain parameters. q, n, k, ℓ, η, γ_1, τ (see Table 7.1).

Key generation. Alice does the following:

Select $A \in_R R_q^{k \times \ell}$, $s_1 \in_R S_\eta^\ell$, and $s_2 \in_R S_\eta^k$.
Compute $t = As_1 + s_2$.
Alice's verification (public) key is (A, t); her signing (private) key is (s_1, s_2).

Signature generation. To sign a message $M \in \{0,1\}^*$, Alice does the following:

Found $\leftarrow$ false.
while Found = false **do**
    Select $y \in_R \tilde{S}_{\gamma_1}^\ell$.
    Compute $w = Ay$ and $w_1 = \text{HighBits}(w)$.
    Compute $c = H(M \| w_1)$.
    if LowBits$(w - cs_2)$ are "sufficiently small" **then**
        Compute $z = y + cs_1$.
        Found $\leftarrow$ true.
Return$(\sigma \leftarrow (c, z))$.

Signature verification. To verify Alice's signature $\sigma = (c, z)$ on M, Bob does:

Obtain an authentic copy of Alice's verification key (A, t).
Compute $w_1' = \text{HighBits}(Az - ct)$.
Accept if and only if $c = H(M \| w_1')$.

Issue #4: Random bits required for signing. A large number of random bits are needed to generate $y \in \tilde{S}^{\ell}_{\gamma_1}$ in each iteration of the signature generation process. To eliminate the need for random bits, y is generated using a secret seed ρ'' and a counter κ via a pseudorandom bit generator called ExpandMask. The seed ρ'' is derived by hashing a secret key K and the message hash μ. This approach makes the signature process *deterministic*, meaning a signature is solely dependent on the message and the signing key. Additionally, FIPS 204 allows the signer to supplement the input to ExpandMask with a 256-bit randomly generated string. If this option is chosen, then the signing process becomes *randomized*.

Issue #5: A signature (c, z) may reveal information about s_1. Since $y \in \tilde{S}^{\ell}_{\gamma_1}$, we have $-\gamma_1 < \text{coeff}(y) \leq \gamma_1$ where $\text{coeff}(y)$ denotes the mods q representation of an arbitrary coefficient of an arbitrary polynomial in y. Also, since $c \in B_\tau$ and $s_1 \in S^{\ell}_{\eta}$, we have $-\beta \leq \text{coeff}(cs_1) \leq \beta$ where $\beta = \tau\eta$. Thus, since $z = y + cs_1$ and $\gamma_1 + \beta < q/2$, we have $-\gamma_1 - \beta < \text{coeff}(z) \leq \gamma_1 + \beta$.

Now, if a coefficient of a polynomial in z happens to equal $\gamma_1 + \beta$, then the corresponding coefficient of cs_1 must be β. This leaks *some* information about the coefficients of the private key s_1. Similarly, if a coefficient of a polynomial in z is $\gamma_1 + \beta - 1$, the corresponding coefficient of cs_1 must be either $\beta - 1$ or β, again leaking *some* information about s_1.

More generally (as shown in Figure 7.1), if a coefficient of a polynomial in z is $\gamma_1 + a$ for some $-\beta + 1 \leq a \leq \beta$ (or the coefficient is $-\gamma_1 + a$ for some $-\beta < a \leq \beta$), the corresponding coefficient of cs_1 must fall within the interval $[a, \beta]$ (resp. in $[-\beta, a - 1]$).

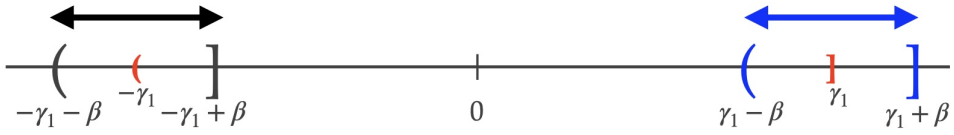


Figure 7.1: Rejection sampling in Dilithium.

To prevent this leakage of information about s_1, *rejection sampling* is employed. The signer repeatedly selects $y \in_R \tilde{S}^{\ell}_{\gamma_1}$ and computes $z = y + cs_1$ until $-\gamma_1 + \beta < \text{coeff}(z) \leq \gamma_1 - \beta$; FIPS 204 imposes the slightly stricter constraint that $\|z\|_\infty < \gamma_1 - \beta$. Rejection sampling aims to ensure that z does not reveal any information about cs_1, and thus does not leak information about s_1.

7.2 Dilithium (without t compression)

This section presents a slightly simplified variant of the standardized Dilithium signature scheme, drawing on the discussion from the previous section. We introduce the concepts of HighBits_0 and LowBits_0 for integers modulo q, which will later be refined in §7.3 to support compression of the public key component t.

We begin by generalizing the symmetric mod q operation (Definition 4.10) to even moduli α.

Definition 7.2 Let $\alpha \geq 2$ be an even integer, and let $x \in \mathbb{Z}_\alpha$. Define the *symmetric mod* α of x to be

$$x \bmod^s \alpha = \begin{cases} x, & \text{if } x \leq \alpha/2 \\ x - \alpha, & \text{if } x > \alpha/2. \end{cases}$$

The mods operation extends naturally to all integers: if $z \in \mathbb{Z}$ then

$$z \bmod^s \alpha = (z \bmod \alpha) \bmod^s \alpha.$$

Note that $-\alpha/2 \leq z \bmod^s \alpha \leq \alpha/2$.

HighBits and LowBits (preliminary). Let α be an even divisor of $q-1$, and set $m = (q-1)/\alpha$. For integers $r \in [0, q-1]$, define

$$r_0 = r \bmod^s \alpha \quad \text{and} \quad r_1 = (r - r_0)/\alpha.$$

Then r can be written uniquely as

$$r = r_1\alpha + r_0, \text{ where } 0 \leq r_1 \leq m \text{ and } -\alpha/2 < r_0 \leq \alpha/2.$$

We define

$$\text{HighBits}_0(r, \alpha) = r_1 \quad \text{and} \quad \text{LowBits}_0(r, \alpha) = r_0. \tag{35}$$

Figure 7.2 illustrates this decomposition for all $r \in [0, q-1]$ in the case $q = 21$ and $\alpha = 4$. For a polynomial vector $w \in R_q^\ell$, the functions $\text{HighBits}_0(w, \alpha)$ and $\text{LowBits}_0(w, \alpha)$ are obtained by applying the corresponding operations coefficientwise to every polynomial in w.

Algorithm 7.3 presents a streamlined version of Dilithium that incorporates the resolutions to the five issues identified in the toy construction in §7.1. Additional implementation details, omitted from Algorithm 7.3, can be found in FIPS 204. Specifically, the pseudorandom bit generator ExpandA is instantiated using SHAKE128, while the pseudorandom bit generators ExpandS and ExpandMask are implemented using SHAKE256. The variable-output length hash function H, which maps input $m \in \{0,1\}^*$ and a length parameter $d \geq 1$ to an output $H(m, d) \in \{0,1\}^d$, is also instantiated using either SHAKE128 or SHAKE256. Finally, the function SampleInBall maps bitstrings in $\{0,1\}^{2\lambda}$ to polynomials in B_τ.

Signature verification works. Recall (equation 34) that $Az - ct = w - cs_2$. Since $\|\text{LowBits}_0(w - cs_2,\, 2\gamma_2)\|_\infty < \gamma_2 - \beta$ and $\|cs_2\|_\infty \leq \beta$, we have

$$w_1' = \text{HighBits}_0(Az - ct,\, 2\gamma_2) = \text{HighBits}_0(w - cs_2,\, 2\gamma_2) = \text{HighBits}_0(w,\, 2\gamma_2) = w_1.$$

Thus, $H(\mu \| w_1',\, 2\lambda) = H(\mu \| w_1,\, 2\lambda)$ and the signature will be accepted.

Algorithm 7.3: Dilithium signature scheme (without t compression)

Domain parameters. $q, n, k, \ell, \eta, \gamma_1, \gamma_2, \tau, \beta, \lambda$ (see Table 7.1).

Key generation. Alice does the following:

Select $\xi \in_R \{0,1\}^{256}$.
Compute $(\rho, \rho', K) = H(\xi, 1024)$, where $\rho \in \{0,1\}^{256}$, $\rho' \in \{0,1\}^{512}$, $K \in \{0,1\}^{256}$.
Compute $A = \text{ExpandA}(\rho) \in R_q^{k \times \ell}$.
Compute $(s_1, s_2) = \text{ExpandS}(\rho') \in S_\eta^\ell \times S_\eta^k$.
Compute $t = As_1 + s_2$.
Compute $tr = H(\rho \| t,\ 2\lambda)$.
Alice's verification (public) key is (ρ, t); her signing (private) key is (ρ, K, tr, s_1, s_2).

Signature generation. To sign a message $M \in \{0,1\}^*$, Alice does the following:

Compute $A = \text{ExpandA}(\rho)$.
Compute $\mu = H(tr \| M,\ 512)$.
Compute $\rho'' = H(K \| \text{rnd} \| \mu,\ 512)$ where either $\text{rnd} = 0^{256}$ or $\text{rnd} \in_R \{0,1\}^{256}$.
Set $\kappa = 0$ and Found = false.
while Found = false **do**
    Compute $y = \text{ExpandMask}(\rho'', \kappa) \in \tilde{S}_{\gamma_1}^\ell$.
    Compute $w = Ay$ and $w_1 = \text{HighBits}_0(w, 2\gamma_2)$.
    Compute $\tilde{c} = H(\mu \| w_1, 2\lambda)$ and $c = \text{SampleInBall}(\tilde{c})$.
    Compute $r_0 = \text{LowBits}_0(w - cs_2, 2\gamma_2)$.
    if $\|r_0\|_\infty < \gamma_2 - \beta$ **then**
        Compute $z = y + cs_1$.
        if $\|z\|_\infty < \gamma_1 - \beta$ **then**
            Found $\leftarrow$ true.
    $\kappa \leftarrow \kappa + \ell$ (increment the counter κ).
Return$(\sigma \leftarrow (\tilde{c}, z))$.

Signature verification. To verify Alice's signature $\sigma = (\tilde{c}, z)$ on M, Bob does:

Obtain an authentic copy of Alice's verification key $PK = (\rho, t)$.
if $\|z\|_\infty \geq \gamma_1 - \beta$ **then**
    Reject the signature.
Compute $A = \text{ExpandA}(\rho)$.
Compute $tr = H(\rho \| t, 512)$ and $\mu = H(tr \| M, 512)$.
Compute $c = \text{SampleInBall}(\tilde{c})$.
Compute $w_1' = \text{HighBits}_0(Az - ct, 2\gamma_2)$.
Check that $\tilde{c} = H(\mu \| w_1', 2\lambda)$; if not then reject the signature.
Accept the signature.

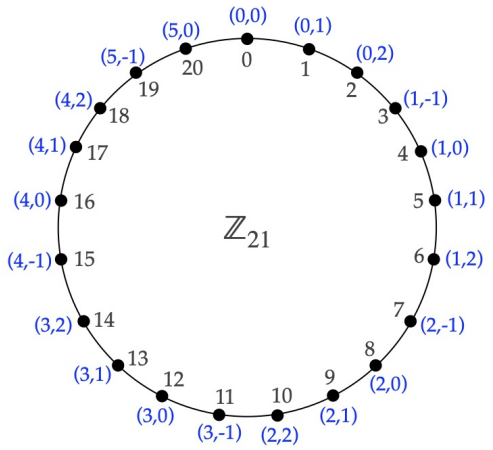


Figure 7.2: $(\text{HighBits}_0(r,4), \text{LowBits}_0(r,4))$ for $q = 21$ and $0 \le r \le 20$.

Example 7.4 (*toy example of Dilithium signature scheme per Algorithm 7.3*)
Domain parameters. $q = 98993$ and $n = 4$ (so $R_q = \mathbb{Z}_{98993}[x]/(x^4+1)$), $k = 5$, $\ell = 4$, $\eta = 15$, $\gamma_1 = 2^{12} = 4096$, $\gamma_2 = (q-1)/16 = 6187$, $\tau = 3$, $\beta = 45$.
Key generation. Alice generates the following matrix[9] $A \in R_q^{5\times 4}$:

$$\left[\begin{array}{cc}
83831 + 86182x + 34460x^2 + 53827x^3 & 52753 + 4611x + 33768x^2 + 68049x^3 \\
26611 + 31871x + 51791x^2 + 53135x^3 & 77983 + 73743x + 73630x^2 + 76856x^3 \\
\hline
84109 + 26539x + 57487x^2 + 49250x^3 & 78952 + 68437x + 21583x^2 + 35337x^3 \\
7915 + 41093x + 54045x^2 + 86724x^3 & 95963 + 98131x + 25363x^2 + 57139x^3 \\
\hline
95705 + 84707x + 74757x^2 + 85481x^3 & 8110 + 6018x + 86989x^2 + 84426x^3 \\
38849 + 66776x + 12868x^2 + 40821x^3 & 42661 + 14351x + 34674x^2 + 64475x^3 \\
\hline
52599 + 20391x + 59361x^2 + 18240x^3 & 30833 + 51209x + 68814x^2 + 96318x^3 \\
51956 + 43227x + 14013x^2 + 88049x^3 & 19027 + 33104x + 17796x^2 + 44217x^3 \\
\hline
34258 + 93745x + 15427x^2 + 3945x^3 & 567 + 74669x + 28186x^2 + 94530x^3 \\
42638 + 83911x + 12415x^2 + 44545x^3 & 40249 + 60948x + 57689x^2 + 70091x^3
\end{array}\right].$$

Alice then selects the signing key components

$$s_1 = \begin{bmatrix} 3 + 13x - 6x^2 - 2x^3 \\ -10 + 7x \\ -2 + 5x - 13x^2 + 9x^3 \\ -7 + x^2 \end{bmatrix} \in S_{15}^4, \quad s_2 = \begin{bmatrix} 14 - 13x + 12x^2 + 3x^3 \\ 5 - 2x - 14x^2 - 14x^3 \\ 10 + 13x - 3x^2 - 11x^3 \\ -14 - 13x + 10x^2 + 11x^3 \\ 14 + 14x + 9x^2 + 10x^3 \end{bmatrix} \in S_{15}^5,$$

[9] Due to space constraints, each row of A spans two lines.

and computes the verification key component

$$t = As_1 + s_2 = \begin{bmatrix} 57924 + 81390x + 10162x^2 + 83131x^3 \\ 40264 + 19231x + 1880x^2 + 93259x^3 \\ 52802 + 92079x + 23039x^2 + 13512x^3 \\ 16927 + 8353x + 32968x^2 + 33005x^3 \\ 6826 + 54397x + 87113x^2 + 45439x^3 \end{bmatrix} \in R_q^5.$$

Here, the coefficients of polynomials in s_1 and s_2 are expressed in their mods q representation. Alice's verification key is (A, t); her signing key is (s_1, s_2).

Signature generation. To sign a message M, Alice generates

$$y \bmod q = \begin{bmatrix} 915 + 3143x - 1424x^2 + 893x^3 \\ 2 - 819x + 2904x^2 - 805x^3 \\ -797 - 3770x + 1184x^2 - 1217x^3 \\ 3051 + 1878x + 1447x^2 + 1773x^3 \end{bmatrix} \in \tilde{S}^4_{\gamma_1}$$

and computes

$$w = Ay = \begin{bmatrix} 17575 + 82813x + 17567x^2 + 46483x^3 \\ 75506 + 5076x + 88490x^2 + 21861x^3 \\ 23112 + 54839x + 57476x^2 + 21701x^3 \\ 84380 + 57159x + 73769x^2 + 8929x^3 \\ 27202 + 41260x + 96026x^2 + 74081x^3 \end{bmatrix}$$

and the commitment

$$w_1 = \mathrm{HighBits}_0(w, 2\gamma_2) = \begin{bmatrix} 1 + 7x + x^2 + 4x^3 \\ 6 + 7x^2 + 2x^3 \\ 2 + 4x + 5x^2 + 2x^3 \\ 7 + 5x + 6x^2 + x^3 \\ 2 + 3x + 8x^2 + 6x^3 \end{bmatrix}.$$

Alice computes $\tilde{c} = H(\mu \| w_1)$ and $c = \mathrm{SampleInBall}(\tilde{c})$ (details omitted), obtaining

$$c \bmod q = -x + x^2 - x^3 \in B_3,$$

and then computes the response

$$z = y + cs_1 = \begin{bmatrix} 932 + 3136x + 97557x^2 + 909x^3 \\ 9 + 98184x + 2887x^2 + 98205x^3 \\ 98223 + 95203x + 1186x^2 + 97796x^3 \\ 3050 + 1886x + 1440x^2 + 1779x^3 \end{bmatrix}.$$

Using mods q representation for coefficients, we have

$$z \bmod q = \begin{bmatrix} 932 + 3136x - 1436x^2 + 909x^3 \\ 9 - 809x + 2887x^2 - 788x^3 \\ -770 - 3790x + 1186x^2 - 1197x^3 \\ 3050 + 1886x + 1440x^2 + 1779x^3 \end{bmatrix}.$$

Note that $\|z\|_\infty < \gamma_1 - \beta = 4051$. Alice also computes

$$w - cs_2 = \begin{bmatrix} 17597 + 82818x + 17537x^2 + 46522x^3 \\ 75508 + 5081x + 88497x^2 + 21854x^3 \\ 23107 + 54841x + 57490x^2 + 21695x^3 \\ 84392 + 57146x + 73759x^2 + 8938x^3 \\ 27187 + 41275x + 96016x^2 + 74090x^3 \end{bmatrix}$$

and

$$r_0 = \text{LowBits}_0(w - cs_2, 2\gamma_2) = \begin{bmatrix} 5223 - 3800x + 5163x^2 - 2974x^3 \\ 1264 + 5081x + 1879x^2 - 2894x^3 \\ -1641 + 5345x - 4380x^2 - 3053x^3 \\ -2226 - 4724x - 485x^2 - 3436x^3 \\ 2439 + 4153x - 2976x^2 - 154x^3 \end{bmatrix}.$$

Note that $\|r_0\|_\infty < \gamma_2 - \beta = 6142$. Alice's signature on M is $\sigma = (\tilde{c}, z)$.

Signature verification. To verify Alice's signature $\sigma = (\tilde{c}, z)$ on M, Bob computes $c = \text{SampleInBall}(\tilde{c})$ obtaining c mods $q = -x + x^2 - x^3 \in B_3$,

$$Az - ct = \begin{bmatrix} 17597 + 82818x + 17537x^2 + 46522x^3 \\ 75508 + 5081x + 88497x^2 + 21854x^3 \\ 23107 + 54841x + 57490x^2 + 21695x^3 \\ 84392 + 57146x + 73759x^2 + 8938x^3 \\ 27187 + 41275x + 96016x^2 + 74090x^3 \end{bmatrix},$$

and

$$w_1' = \text{HighBits}_0(Az - ct, 2\gamma_2) = \begin{bmatrix} 1 + 7x + x^2 + 4x^3 \\ 6 + 7x^2 + 2x^3 \\ 2 + 4x + 5x^2 + 2x^3 \\ 7 + 5x + 6x^2 + x^3 \\ 2 + 3x + 8x^2 + 6x^3 \end{bmatrix}.$$

Since $w_1' = w_1$, we have $H(\mu \| w_1', 2\lambda) = H(\mu \| w_1, 2\lambda)$ so Bob accepts the signature.

Security of Dilithium without t compression (Algorithm 7.3). If the short-secret D-MLWE problem (Definition 5.18) is intractable, an adversary cannot derive any information about the secret key component s_1 (or s_2) from the public key (ρ, t). Evaluating the difficulty of signature forgery is more complex. To simplify the analysis, we will make a few simplifications to the signature scheme and provide an informal argument that a forger with access to the public key (ρ, t) cannot produce a valid signed message.

The forger's task is as follows: given the verification key (ρ, t), find a message M and signature $(\tilde{c}, z)$ such that $\|z\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = H(\mu \| w_1)$, where $\mu = H(M)$, $w_1 = \text{HighBits}_0(Az - ct,\ 2\gamma_2)$, and $c = \text{SampleInBall}(\tilde{c})$. It is important to note that c depends on $\tilde{c}$, $\tilde{c}$ depends on w_1, and w_1 depends on c, creating a circular dependency. Assuming that H has no weaknesses, it would appear that the forger's best strategy is to randomly select M and $w_1 \in R_q^k$, where each coefficient in w_1 is in the range $[0, (q-1)/2\gamma_2]$, and

then compute $\mu = H(M)$, $\tilde{c} = H(\mu \| w_1)$, and $c = \text{SampleInBall}(\tilde{c})$. The remaining task is to find a z that satisfies the signature conditions.

Now, since $w_1 = \text{HighBits}_0(Az - ct,\ 2\gamma_2)$, we can write

$$Az - ct = 2\gamma_2 w_1 + w_0$$

where $w_0 \in R_q^k$ with $-\gamma_2 < \text{coeff}(w_0) \leq \gamma_2$. Rearranging terms gives

$$Az - w_0 = ct + 2\gamma_2 w_1.$$

Thus, the forger's task is to find a solution (z, w_0) to the linear system of equations

$$\begin{bmatrix} A \mid I_k \end{bmatrix} \begin{bmatrix} z \\ -w_0 \end{bmatrix} = ct + 2\gamma_2 w_1, \tag{36}$$

where $z \in R_q^\ell$ and $w_0 \in R_q^k$ are relatively small (more precisely, $\|z\|_\infty < \gamma_1 - \beta$ and $-\gamma_2 < \text{coeff}(w_0) \leq \gamma_2$). This is an instance of the normal-form version of Inhomogeneous MSIS (Definition 5.13).

7.3 Dilithium (with t compression)

The FIPS 204 standardized Dilithium signature scheme incorporates a compression technique for the public key component t. This requires adjusting the definitions of HighBits_0 and LowBits_0 and introducing the use of "hint bits", which in turn results in a modest increase in signature size. For the ML-DSA-44, ML-DSA-65 and ML-DSA-87 parameter sets, Dilithium public keys have sizes of 1312, 1952 and 2592 bytes, respectively, while the corresponding signature sizes are 2420, 3309 and 4627 bytes.

Power2Round. Let d be an integer satisfying $1 \leq d \leq \log_2 q$. For an integer $r \in [0, q-1]$, define

$$r_0 = r \bmod 2^d \quad \text{and} \quad r_1 = (r - r_0)/2^d.$$

Then r can be expressed as

$$r = r_1 2^d + r_0, \text{ where } -2^{d-1} < r_0 \leq 2^{d-1} \text{ and } 0 \leq r_1 \leq \lceil (q-1)/2^d \rceil.$$

The function Power2Round[10] decomposes r into a high-order component r_1 and a low-order component r_0. Figure 7.3 illustrates this decomposition for the case $q = 23$ and $d = 3$. The Power2Round function extends naturally to polynomial vectors $t \in R_q^k$ by applying the decomposition coefficientwise to each polynomial in t. For example, if $t = 5 + 2x + 19x^2 + 8x^3 \in R_q = \mathbb{Z}_{23}[x]/(x^4 + 1)$, then $\text{Power2Round}(t, 3) = (t_1, t_0)$ where $t_1 = 1 + 2x^2 + x^3$ and $t_0 = -3 + 2x + 3x^2$.

[10] Power2Round is similar to $\text{HighBits}_0/\text{LowBits}_0$, except that the modulus is required to be a power of two and does not necessarily divide $q - 1$.

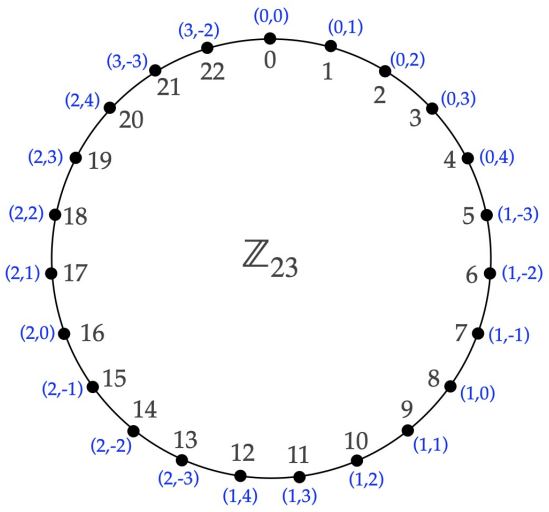


Figure 7.3: Power2Round$(r, d) = (r_1, r_0)$ for $q = 23$, $d = 3$, and $0 \le r \le 22$.

t compression. Recall that the Dilithium verification key includes a polynomial vector $t \in R_q^k$. The size of this key can be reduced by discarding the d low-order bits of each coefficient of t. Concretely, Power2Round is used to write $t = t_1 2^d + t_0$. The compressed component t_1 is included in the verification key, while t_0 is retained as part of the signing key.

For the ML-DSA-87 parameter set, we have $k = 8$ and $d = 13$. Since each coefficient in R_q requires 23 bits, the full vector t has size $8 \times 256 \times 23$ bits, or 5,888 bytes. By contrast, the coefficients of t_1 have size 10 bits, giving a total size of $8 \times 256 \times 10$ bits, or 2,560 bytes. Consequently, the public key (ρ, t_1) has total size 2,592 bytes.

However, omitting t_0 from the verification key introduces an additional complication in verification. Recall that the verifier needs to compute $Az - ct$ in order to recover $\text{HighBits}_0(w - cs_2, 2\gamma_2)$. Since only t_1 is available, the verifier instead computes

$$Az - ct_1 2^d = Az - c(t - t_0) = Az - ct + ct_0 = w - cs_2 + ct_0. \tag{37}$$

The coefficients of ct_0 are expected to be relatively small. In particular, we expect that $\|ct_0\|_\infty \le \gamma_2$ with very high probability. When this condition holds, adding $-ct_0$ to $w - cs_2 + ct_0$ changes the corresponding HighBits_0 by at most ± 1. Consequently, although the verifier cannot directly compute $\text{HighBits}_0(w - cs_2, 2\gamma_2)$, they can still recover it from $w - cs_2 + ct_0$ provided that suitable auxiliary information is supplied.

Hint bits. To enable recovery of $\text{HighBits}_0(w - cs_2, 2\gamma_2)$, the signer appends a small amount of auxiliary information to the signature in the form of "hint bits". Intuitively, these bits encode the "carry information" induced when the correction term $-ct_0$ is added to $w - cs_2 + ct_0$.

Let $r \in [0, q-1]$ with $r_1 = \text{HighBits}_0(r, \alpha)$ and $r_0 = \text{LowBits}_0(r, \alpha)$, and let $z \in [-\gamma_2, \gamma_2]$. A hint bit $h \in \{0, 1\}$ is intended to allow the computation of $\text{HighBits}_0(r + z, \alpha)$ from r and h, without knowledge of z. In this setting, z models a coefficient of $-ct_0$, while r corresponds to a coefficient of $w - cs_2 + ct_0$.

A difficulty in defining such a hint bit arises from the fact that only two values, 0 and 1, are available, whereas there are three possible carry digits, namely -1, 0 and 1. To illustrate this issue, consider the case where $q = 21$, $\alpha = 4$, $\gamma_2 = 2$ and $m = 5$. For $r \in [0, 18]$, one observes that $\text{HighBits}_0(r, 4)$ changes when either $z = 2$ or $z = -2$ is added to r, but not both; see Figure 7.4. For instance, when $z = 2$ is added to $r = 6$

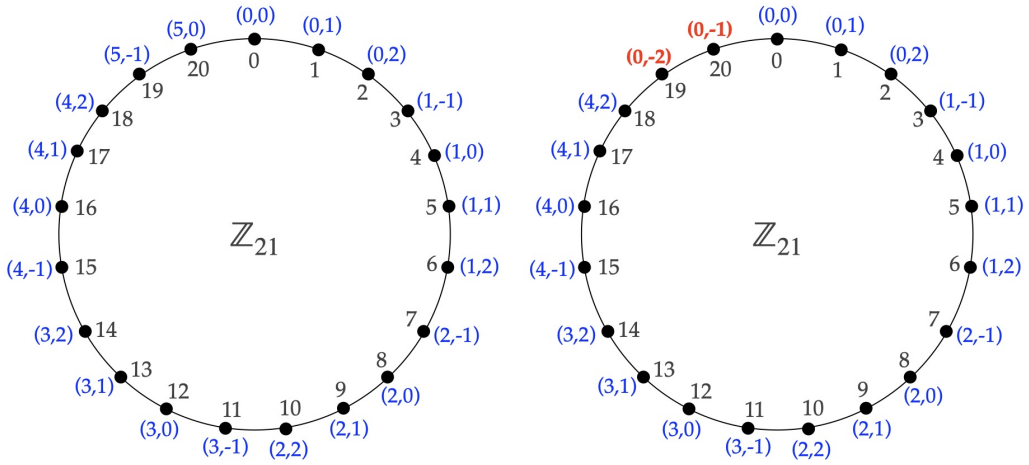


Figure 7.4: The original (left) and adjusted (right) definitions of $(\text{HighBits}(r, \alpha), \text{LowBits}(r, \alpha))$ for $(q, \alpha) = (21, 4)$ and $0 \le r \le 20$.

then $\text{HighBits}_0(r, 4)$ is increased from 1 to 2, whereas if $z = -2$ is added to $r = 6$ then $\text{HighBits}_0(r, 4)$ is unchanged. However, this behaviour does not hold for $r = 19$ and $r = 20$. For example, if $z = 2$ is added to $r = 20$, then $\text{HighBits}_0(r, 4)$ changes from 5 to 0, and adding $z = -2$ changes it from 5 to 4.

The underlying issue is that the multiples 20 and 0 of $\alpha = 4$ are side-by-side. More generally, the multiples $q - 1 = m \cdot \alpha$ and $0 = 0 \cdot \alpha$ of α lie next to each other. To resolve this, the definitions of HighBits_0 and LowBits_0 are slightly adjusted.

HighBits and LowBits (adjusted). Returning to the example with $q = 21$ and $\alpha = 4$, we redefine $(\text{HighBits}(20, 4), \text{LowBits}(20, 4))$ from $(5, 0)$ to $(0, -1)$, noting that $20 \equiv 0 \cdot \alpha - 1 \pmod{21}$. Similarly, we redefine $(\text{HighBits}(19, 4), \text{LowBits}(19, 4))$ from $(5, -1)$ to $(0, -2)$, since $19 \equiv 0 \cdot \alpha - 2 \pmod{21}$. See Figure 7.4. With this adjustment, $r = 19$ and $r = 20$ satisfy the desired property: $\text{HighBits}(r, 4)$ changes when either $z = 2$ or $z = -2$ is added to r, but not both.

We now present the modified definitions of HighBits and LowBits.

Definition 7.5 Let α be an even divisor of $q-1$, and let $r \in [0, q-1]$. Let $s = r \bmod \alpha$. Define

$$(r_1, r_0) = \begin{cases} (0, s-1), & \text{if } r - s = q - 1, \\ ((r-s)/\alpha, s), & \text{otherwise.} \end{cases}$$

Then $\text{HighBits}(r, \alpha) = r_1$ and $\text{LowBits}(r, \alpha) = r_0$.

MakeHint and UseHint. With this convention in place, a single hint bit h suffices to indicate unambiguously whether HighBits remained unchanged ($h = 0$) or whether it increases or decreases by one modulo m ($h = 1$).

Definition 7.6 Let α be an even divisor of $q-1$. For $r \in [0, q-1]$ and $z \in [-\alpha/2, \alpha/2]$, define the *hint bit*

$$\text{MakeHint}(z, r, \alpha) = \begin{cases} 1, & \text{if } \text{HighBits}(r+z, \alpha) \neq \text{HighBits}(r, \alpha), \\ 0, & \text{otherwise.} \end{cases}$$

The function UseHint is used to recover $\text{HighBits}(r+z, \alpha)$ from r and h.

Definition 7.7 Let α be an even divisor of $q-1$, and let $m = (q-1)/\alpha$. Let $r \in [0, q-1]$ and $h \in \{0, 1\}$, and let $r_1 = \text{HighBits}(r, \alpha)$ and $r_0 = \text{LowBits}(r, \alpha)$. Define

$$\text{UseHint}(h, r, \alpha) = \begin{cases} r_1, & \text{if } h = 0, \\ r_1 + 1 \bmod m, & \text{if } h = 1 \text{ and } r_0 > 0, \\ r_1 - 1 \bmod m, & \text{if } h = 1 \text{ and } r_0 \leq 0. \end{cases}$$

The following result shows that the functions MakeHint and UseHint operate as intended.

Lemma 7.8 Let α be an even divisor of $q-1$, and let $r \in [0, q-1]$ and $z \in [-\alpha/2, \alpha/2]$. Then

$$\text{UseHint}(\text{MakeHint}(z, r, \alpha), r, \alpha) = \text{HighBits}(r+z, \alpha).$$

The operations MakeHint and UseHint extend naturally to polynomial vectors in R_q^k.

Algorithm 7.9 presents the full version of the Dilithium signature scheme with t compression. The signer first checks[11] that $\|ct_0\|_\infty < \gamma_2$ and then constructs a *hint vector*

$$h = \text{MakeHint}(-ct_0,\ w - cs_2 + ct_0,\ 2\gamma_2),$$

which is appended to the signature. Since $-ct_0 \in R_q^k$, we have $h = (h_1, h_2, \ldots, h_k)$ where each $h_i \in \{0,1\}^{256}$. To further control the signature size, an additional constraint is imposed requiring that the total number of one bits in h does not exceed ω. The verifier applies the hint vector h to compute

$$w_1' = \text{UseHint}(h,\ Az - ct_1 2^d,\ 2\gamma_2).$$

[11] FIPS 204 specifies the condition $\|ct_0\|_\infty < \gamma_2$; alternatively, the check could have been defined as $\|ct_0\|_\infty \leq \gamma_2$.

Algorithm 7.9: Dilithium signature scheme (with t compression)

Domain parameters. $q, n, k, \ell, \eta, \gamma_1, \gamma_2, \tau, \beta, \lambda, d, \omega$ (see Table 7.1).

Key generation. Alice does the following:

Select $\xi \in_R \{0,1\}^{256}$.
Compute $(\rho, \rho', K) = H(\xi, 1024)$, where $\rho \in \{0,1\}^{256}$, $\rho' \in \{0,1\}^{512}$, $K \in \{0,1\}^{256}$.
Compute $A = \text{ExpandA}(\rho) \in R_q^{k \times \ell}$.
Compute $(s_1, s_2) = \text{ExpandS}(\rho') \in S_\eta^\ell \times S_\eta^k$.
Compute $t = As_1 + s_2$ and $(t_1, t_0) = \text{Power2Round}(t, d)$.
Compute $tr = H(\rho \| t_1,\ 2\lambda)$.
Alice's verification (public) key is (ρ, t_1); her signing (private) key is $(\rho, K, tr, s_1, s_2, t_0)$.

Signature generation. To sign a message $M \in \{0,1\}^*$, Alice does the following:

Compute $A = \text{ExpandA}(\rho)$.
Compute $\mu = H(tr \| M,\ 512)$.
Compute $\rho'' = H(K \| \text{rnd} \| \mu,\ 512)$ where either $\text{rnd} = 0^{256}$ or $\text{rnd} \in_R \{0,1\}^{256}$.
Set $\kappa = 0$ and Found = false.
while Found = false **do**
    Compute $y = \text{ExpandMask}(\rho'', \kappa) \in \tilde{S}_{\gamma_1}^\ell$.
    Compute $w = Ay$ and $w_1 = \text{HighBits}(w, 2\gamma_2)$.
    Compute $\tilde{c} = H(\mu \| w_1, 2\lambda)$ and $c = \text{SampleInBall}(\tilde{c})$.
    Compute $r_0 = \text{LowBits}(w - cs_2, 2\gamma_2)$ and $s_0 = ct_0$.
    if $\|r_0\|_\infty < \gamma_2 - \beta$ and $\|s_0\|_\infty < \gamma_2$ **then**
        Compute $h = \text{MakeHint}(-s_0,\ w - cs_2 + s_0,\ 2\gamma_2)$.
        if the number of 1's in h is $\leq \omega$ **then**
            Compute $z = y + cs_1$.
            if $\|z\|_\infty < \gamma_1 - \beta$ **then**
                Found $\leftarrow$ true.
    $\kappa \leftarrow \kappa + \ell$ (increment the counter κ).
Return$(\sigma \leftarrow (\tilde{c}, z, h))$.

Signature verification. To verify Alice's signature $\sigma = (\tilde{c}, z, h)$ on M, Bob does:

Obtain an authentic copy of Alice's verification key $PK = (\rho, t_1)$.
if $\|z\|_\infty \geq \gamma_1 - \beta$ or the number of 1's in h is $> \omega$ **then**
    Reject the signature.
Compute $A = \text{ExpandA}(\rho)$.
Compute $tr = H(\rho \| t_1, 512)$ and $\mu = H(tr \| M, 512)$.
Compute $c = \text{SampleInBall}(\tilde{c})$.
Compute $w_1' = \text{UseHint}(h, Az - ct_1 2^d, 2\gamma_2)$.
Check that $\tilde{c} = H(\mu \| w_1', 2\lambda)$; if not then reject the signature.
Accept the signature.

Signature verification works. Recall (equation 37) that $Az - ct_1 2^d = w - cs_2 + ct_0$. Since $\|-ct_0\|_\infty < \gamma_2$, we have

$$\begin{aligned} w_1' &= \text{UseHint}(h, Az - ct_1 2^d,\ 2\gamma_2) = \text{HighBits}(Az - ct_1 2^d - ct_0,\ 2\gamma_2) \\ &= \text{HighBits}(Az - ct,\ 2\gamma_2) = \text{HighBits}(w - cs_2,\ 2\gamma_2). \end{aligned}$$

And, since $\|\text{LowBits}(w - cs_2,\ 2\gamma_2)\|_\infty < \gamma_2 - \beta$ and $\|cs_2\|_\infty \le \beta$, we have

$$w_1' = \text{HighBits}(w - cs_2,\ 2\gamma_2) = \text{HighBits}(w,\ 2\gamma_2) = w_1.$$

Thus, $H(\mu \| w_1',\ 2\lambda) = H(\mu \| w_1,\ 2\lambda)$ and the signature will be accepted.

Number of iterations in signature generation. The main loop in signature generation terminates when the following four conditions are simultaneously satisfied:

1. $\|z\|_\infty < \gamma_1 - \beta$.
2. $\|r_0\|_\infty < \gamma_2 - \beta$.
3. $\|ct_0\|_\infty < \gamma_2$.
4. The total number of 1's in the hint vector h is at most ω.

We next estimate the probability of each event.

1. Let
$$p_1 = \Pr(\|z\|_\infty < \gamma_1 - \beta). \tag{38}$$
Recall that $z = y + cs_1 \in R_q^\ell$ where $y \in_R \tilde{S}_{\gamma_1}^\ell$, $c \in B_\tau$, and $s_1 \in_R S_\eta^\ell$. Suppose that a coefficient of cs_1 equals $u \in [-\beta, \beta]$. Then the corresponding coefficient of z lies in the interval $(-\gamma_1 + \beta,\ \gamma_1 - \beta)$ provided that the corresponding coefficient of y lies in $(-\gamma_1 + \beta - u,\ \gamma_1 - \beta - u)$. This interval contains $2(\gamma_1 - \beta) - 1$ valid integer values, independent of the choice of u. Hence,
$$p_1 = \left(\frac{2(\gamma_1 - \beta) - 1}{2\gamma_1}\right)^{256\ell} = \left(1 - \frac{\beta + 1/2}{\gamma_1}\right)^{256\ell} \approx e^{-256\ell\beta/\gamma_1}. \tag{39}$$
2. Let
$$p_2 = \Pr(\|r_0\|_\infty < \gamma_2 - \beta). \tag{40}$$
Assuming that $\text{LowBits}(w - cs_2,\ 2\gamma_2)$ is uniformly distributed in $[-\gamma_2, \gamma_2]^{256k}$, we obtain
$$p_2 = \left(\frac{2(\gamma_2 - \beta) - 1}{2\gamma_2 + 1}\right)^{256k} \approx \left(1 - \frac{\beta + 1/2}{\gamma_2}\right)^{256k} \approx e^{-256k\beta/\gamma_2}. \tag{41}$$

3. Let

$$p_3 = \Pr(\|ct_0\|_\infty < \gamma_2 \text{ and the total number of 1's in } h \text{ is at most } \omega). \quad (42)$$

   Experimental evidence indicates that $p_3 > 0.98$ for the standardized parameter sets.

Assuming that these events are independent, the probability that all four requirements are satisfied in a single iteration of the main loop is at least $p_1 p_2 p_3$. Thus, the expected number of iterations required for signature generation is at most

$$\frac{1}{p_1 p_2 p_3} \approx \frac{1}{p_1 p_2} \approx e^{256\beta(\ell/\gamma_1 + k/\gamma_2)}.$$

For the standardized parameter sets ML-DSA-44, ML-DSA-65, and ML-DSA-87, the expected number of iterations is approximately 4.25, 5.1 and 3.85, respectively.

Security. Compressing the public key component t has no impact on the security of Dilithium. In particular, the full scheme has been proven existentially unforgeable under chosen-message attacks, assuming the hardness of the D-MLWE and MSIS problems, and modelling H as a random oracle.

Remark 7.10 (*selecting Dilithium parameters*) The ML-DSA-44, ML-DSA-65 and ML-DSA-87 parameter sets specified in FIPS 204 (see Table 7.1) were carefully engineered to optimize the trade-offs between security, public key size, signature size, implementation efficiency, and the expected number of iterations during signature generation. The following notes illustrate the delicate balance required when selecting the parameters q, n, k, ℓ, η, γ_1, γ_2, τ, β, d and ω.

1. To facilitate fast polynomial multiplication in R_q via the Number-Theoretic Transform (see §11.3), the degree n was chosen to be a power of two, and the prime q was chosen to satisfy the condition $q \equiv 1 \pmod{2n}$.

2. The dimensions k and ℓ can be increased to achieve higher security levels against MLWE and MSIS attacks without altering the underlying polynomial ring R_q. However, increasing k makes the public key larger (since $t_1 \in R_q^k$), while increasing ℓ makes the signature larger (since $z \in R_q^\ell$).

3. Minimizing γ_1 and γ_2 is desirable to maximize resistance to MSIS attacks; cf. equation (36). Conversely, larger values for γ_1 and γ_2 are preferred because they increase the acceptance probabilities p_1 and p_2 (see equations (39) and (41)), thereby reducing the expected number of iterations during signature generation.

4. A larger secret bound η enhances resistance against MLWE attacks, but it consequently increases β (since $\beta = \tau\eta$), which lowers the acceptance probabilities p_1 and p_2.

5. While a smaller value of τ is desirable because it reduces β, τ must remain sufficiently large to ensure that the size $2^\tau \binom{256}{\tau}$ of the challenge space B_τ provides adequate security.

6. Increasing the compression parameter d reduces the size of the public key component t_1, and decreasing ω reduces the size of the hint vector h. However, choosing a large d or a small ω reduces the acceptance probability p_3 (see equation (42)), which increases the expected number of iterations.

7.4 Exercises

Exercise 7.1 Let $q = 9769$, $n = 5$, $\alpha = 296$, and

$$a(x) = \begin{bmatrix} 4024 + 2133x + 9066x^2 + 1163x^3 + 8416x^4 \\ 5179 + 5598x + 7747x^2 + 5777x^3 + 6635x^4 \\ 7720 + 8139x + 519x^2 + 7008x^3 + 9176x^4 \end{bmatrix} \in R_q^3.$$

Verify that

$$\text{HighBits}(a(x), 296) = \begin{bmatrix} 14 + 7x + 31x^2 + 4x^3 + 28x^4 \\ 17 + 19x + 26x^2 + 20x^3 + 22x^4 \\ 26 + 27x + 2x^2 + 24x^3 + 31x^4 \end{bmatrix}$$

and

$$\text{LowBits}(a(x), 296) = \begin{bmatrix} -120 + 61x - 110x^2 - 21x^3 + 128x^4 \\ 147 - 26x + 51x^2 - 143x^3 + 123x^4 \\ 24 + 147x - 73x^2 - 96x^3 \end{bmatrix}.$$

Exercise 7.2 Prove Lemma 7.8.

Exercise 7.3 Describe and analyze the FIPS 204 SampleInBall procedure for generating pseudorandom polynomials in B_τ. This procedure is based on the "Fisher-Yates shuffle".

Exercise 7.4 FIPS 204 specifies the following representation for the hint vector h. Let $h = (h_1, \ldots, h_k)$ where $h_i \in \{0,1\}^{256}$, and where the total number u of one bits in h is at most ω.
Let y be a byte array of length $\omega + k$, all of whose entries are initialized to 0. For each $1 \le i \le k$, let $y[\omega + i]$ store the total number of one bits in $h_1, \ldots, h_i$. Lastly, let $y[1], \ldots, y[u]$ store the positions of the u one bits in $h_1, \ldots, h_k$.
Describe a method for reconstructing $h_1, \ldots, h_k$ from the array y.

Exercise 7.5 Verify that Dilithium with the ML-DSA-44, ML-DSA-65 and ML-DSA-87 parameter sets (Table 7.1) has public keys of sizes 1312, 1952 and 2592 bytes, respectively, and signatures of sizes 2420, 3309 and 4627 bytes.

8 The LLL lattice basis reduction algorithm

Contents

The LLL algorithm, also known as L^3, is a polynomial-time algorithm that finds a relatively short and nearly orthogonal basis of a lattice. It was discovered in 1982 by Arjen Lenstra, Hendrik Lenstra, and László Lovász. Its first major application was a polynomial-time algorithm for factoring polynomials with rational coefficients. Since then, LLL has become a fundamental tool in computational number theory and cryptography. Most notably, it is used to solve the Shortest Vector Problem (SVP) and its variant approx-SVP which, as we saw in §3 and §4, plays a central role in assessing the security of lattice-based cryptosystems.

The remainder of this section is organized as follows. In §8.1, we review the Gram-Schmidt orthogonalization process, a key ingredient of the LLL algorithm. Gauss's algorithm for computing a shortest basis of a two-dimensional integer lattice is presented in §8.2. The LLL algorithm itself is described and analyzed in §8.3, followed by an overview of several improvements in §8.4.

8.1 Gram-Schmidt orthogonalization

We begin by recalling some basic notions from Euclidean geometry and linear algebra. The *Euclidean length* of a vector $u = (u_1, \ldots, u_n) \in \mathbb{R}^n$ is

$$\|u\|_2 = \sqrt{u_1^2 + \cdots + u_n^2}.$$

To simplify the notation, we write $\|u\|$ instead of $\|u\|_2$. The *inner product* or *dot product* of vectors $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n) \in \mathbb{R}^n$ is

$$\langle u, v \rangle = u_1 v_1 + \cdots + u_n v_n.$$

Notice that $\langle u, u\rangle = \|u\|^2$. The vectors u and v are said to be *orthogonal* if $\langle u, v\rangle = 0$.

Orthogonal complements and projections. Let V be a k-dimensional vector subspace of $\mathbb{R}^n$ with ordered basis $B = [b_1, \ldots, b_k]$. The *orthogonal complement* of V is the subspace

$$V^\perp = \{x \in \mathbb{R}^n \ : \ \langle x, v\rangle = 0 \text{ for all } v \in V\}.$$

The subspace $V^\perp$ has dimension $n - k$. The basis B can be extended by adding vectors $b_{k+1}, \ldots, b_n$ to form a basis $[b_1, \ldots, b_k, b_{k+1}, \ldots, b_n]$ of $\mathbb{R}^n$ such that $[b_{k+1}, \ldots, b_n]$ is a basis of $V^\perp$.

Any vector $w \in \mathbb{R}^n$ can be written uniquely as a linear combination of the basis elements: $w = c_1 b_1 + \cdots + c_n b_n$ with real coefficients c_i. Define $w_1 = c_1 b_1 + \cdots + c_k b_k \in V$ and $w_2 = c_{k+1} b_{k+1} + \cdots + c_n b_n \in V^\perp$. Then $w = w_1 + w_2$, expressing w as a sum of vectors in V and $V^\perp$.

Definition 8.1 Let $w \in \mathbb{R}^n$. The *projection of w onto a subspace V* is $\Pi_V(w) = w_1$, and the *projection of w onto $V^\perp$* is $\Pi_{V^\perp}(w) = w_2$.

Every vector w decomposes as $w = \Pi_V(w) + \Pi_{V^\perp}(w)$. Figure 8.1 illustrates this decomposition in the two-dimensional case, where $V = \langle v\rangle$ is a one-dimensional subspace. Theorem 8.4 shows that computing projections $\Pi_V(w)$ is particularly simple when the

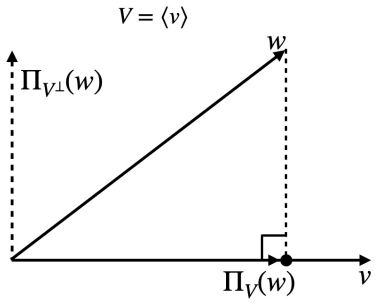


Figure 8.1: The projections of w onto V and $V^\perp$.

basis of V is orthogonal.

Definition 8.2 A basis $B = [b_1, \ldots, b_k]$ of a k-dimensional subspace $V \subseteq \mathbb{R}^n$ is called an *orthogonal basis* if $\langle b_i, b_j\rangle = 0$ for all $1 \le j < i \le k$.

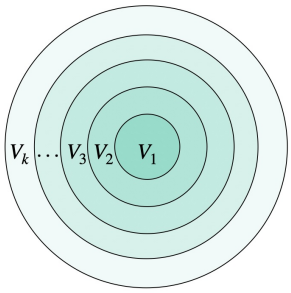
A fundamental fact from linear algebra is that orthogonal bases always exist.

Theorem 8.3 Every subspace $V \subseteq \mathbb{R}^n$ admits an orthogonal basis.

Proof: Let $B = [b_1, \ldots, b_k]$ be any basis of V, and define $V_i = \text{Span}(b_1, \ldots, b_i)$ for $1 \le i \le k$. We construct vectors $b_1^*, \ldots, b_k^*$ inductively as follows. Set $b_1^* = b_1$, and for $2 \le i \le k$, let $V_{i-1}^* = \text{Span}(b_1^*, \ldots, b_{i-1}^*)$ and $b_i^* = \Pi_{(V_{i-1}^*)^\perp}(b_i)$; that is, b_i^* is the projection of b_i

onto the orthogonal complement of the subspace spanned by the previously constructed vectors $b_1^*, \ldots, b_{i-1}^*$. Also, define $V_k^* = \text{Span}(b_1^*, \ldots, b_k^*)$. To ease the notation, we will write Π_i instead of $\Pi_{(V_i^*)^\perp}$.

We claim that for each $1 \leq i \leq k$, $V_i^* = V_i$ and $[b_1^*, \ldots, b_i^*]$ is an orthogonal basis of V_i^*. The claim is proved by induction on i.



Clearly, $V_1^* = V_1$ since $b_1^* = b_1$ and $[b_1^*]$ is an orthogonal basis of V_1^*. Let $i \geq 2$, and suppose that $V_{i-1}^* = V_{i-1}$ and that $[b_1^*, \ldots, b_{i-1}^*]$ is an orthogonal basis of V_{i-1}^*. Now,

$$b_i^* = \Pi_{i-1}(b_i) = b_i - \Pi_{V_{i-1}^*}(b_i) = b_i - \Pi_{V_{i-1}}(b_i) \in V_i,$$

and so $V_i^* \subseteq V_i$. Moreover, $b_i = b_i^* + \Pi_{V_{i-1}^*}(b_i) \in V_i^*$, and so $V_i \subseteq V_i^*$. Hence $V_i^* = V_i$ and so $[b_1^*, \ldots, b_i^*]$ is a basis of V_i^*. Finally, since $b_i^* \in (V_{i-1}^*)^\perp$, we have $\langle b_i^*, b_j^* \rangle = 0$ for all $1 \leq j \leq i-1$. Thus, $[b_1^*, \ldots, b_i^*]$ is an orthogonal basis of V_i^*, which completes the proof of the claim.

In conclusion, $B^* = [b_1^*, \ldots, b_k^*]$ is an orthogonal basis of $V_k^* = V_k = V$, which completes the proof of the theorem. □

The utility of orthogonal bases for computing projections is demonstrated in the next result.

Theorem 8.4 Let $B = [b_1, \ldots, b_k]$ be an orthogonal basis of a subspace $V \subseteq \mathbb{R}^n$, and let $w \in \mathbb{R}^n$. Then

$$\Pi_V(w) = \sum_{j=1}^{k} \frac{\langle w, b_j \rangle}{\|b_j\|^2} b_j. \tag{43}$$

Proof: Extend B to a basis $[b_1, \ldots, b_n]$ of $\mathbb{R}^n$ where $[b_{k+1}, \ldots, b_n]$ is a basis of $V^\perp$, and write $w = c_1 b_1 + \cdots + c_n b_n$ where $c_i \in \mathbb{R}$. For each $1 \leq j \leq k$, orthogonality implies that

$$\langle w, b_j \rangle = \langle c_1 b_1 + \cdots + c_n b_n, b_j \rangle = c_1 \langle b_1, b_j \rangle + \cdots + c_n \langle b_n, b_j \rangle = c_j \langle b_j, b_j \rangle = c_j \|b_j\|^2.$$

Hence, $c_j = \langle w, b_j \rangle / \|b_j\|^2$. Finally,

$$\Pi_V(w) = \Pi_V\left(\sum_{j=1}^{n} c_j b_j\right) = \sum_{j=1}^{n} c_j \Pi_V(b_j) = \sum_{j=1}^{k} c_j \Pi_V(b_j) = \sum_{j=1}^{k} c_j b_j,$$

and the result follows. □

Gram-Schmidt orthogonalization. Let $B = [b_1, \ldots, b_k]$ be a basis of a k-dimensional subspace $V \subseteq \mathbb{R}^n$. Applying the projection formula (43), we have

$$b_i^* = \Pi_{i-1}(b_i) = b_i - \Pi_{V_{i-1}^*}(b_i) = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^*,$$

where the *Gram-Schmidt coefficients* μ_{ij}, for $1 \leq j < i \leq k$, are defined as follows:

$$\mu_{ij} = \frac{\langle b_i, b_j^* \rangle}{\|b_j^*\|^2}. \tag{44}$$

This yields the Gram-Schmidt process for computing an orthogonal basis for a vector space, which is summarized in Algorithm 8.5. Theorem 8.28 asserts that Algorithm 8.5 is a polynomial-time algorithm when B is a basis for a full-rank integer lattice.

Algorithm 8.5: Gram-Schmidt orthogonalization process

Input: A basis $B = [b_1, \ldots, b_k]$ of a k-dimensional subspace $V \subseteq \mathbb{R}^n$.
Output: An orthogonal basis $B^* = [b_1^*, \ldots, b_k^*]$ of V.

```
b_1^* ← b_1.
for i = 2 to k do
    Compute μ_ij = ⟨b_i, b_j^*⟩/‖b_j^*‖^2 for 1 ≤ j < i.
    b_i^* ← b_i − Σ_{j=1}^{i−1} μ_ij b_j^*.
return (B^* = [b_1^*, ..., b_k^*])
```

Observe that each ordered basis determines a unique Gram-Schmidt basis. Two useful properties of Gram-Schmidt bases are that $\|b_i^*\| \leq \|b_i\|$ and $\langle b_i, b_i^* \rangle = \langle b_i^*, b_i^* \rangle$ for all $1 \leq i \leq n$ (Exercise 8.3).

Example 8.6 (*Gram-Schmidt orthogonalization*) Consider the basis

$$B = [(-5, -2, 5, 1),\ (3, 5, 2, 3),\ (0, 1, 2, -3),\ (-3, -4, -4, -5)]$$

for a full-rank integer lattice $L \subseteq \mathbb{Z}^4$. The squared lengths of the basis vectors are:

$$\|b_1\|^2 = 55, \quad \|b_2\|^2 = 47, \quad \|b_3\|^2 = 14, \quad \|b_4\|^2 = 66.$$

The Gram-Schmidt orthogonalization of B is

$$B^* = [(-5, -2, 5, 1),\ \left(\frac{21}{11}, \frac{251}{55}, \frac{34}{11}, \frac{177}{55}\right),\ \left(\frac{995}{2441}, \frac{2611}{2441}, \frac{3587}{2441}, -\frac{7738}{2441}\right),$$
$$\left(-\frac{35014}{32999}, \frac{31857}{32999}, -\frac{21525}{32999}, -\frac{3731}{32999}\right)].$$

The squared lengths of the Gram-Schmidt basis vectors are:

$$\|b_1^*\|^2 = 55, \quad \|b_2^*\|^2 \approx 44.4, \quad \|b_3^*\|^2 \approx 13.5, \quad \|b_4^*\|^2 \approx 2.5.$$

Notice that $\|b_i^*\|^2 \leq \|b_i\|^2$ for $i = 1, 2, 3, 4$. The lengths of the Gram-Schmidt basis vectors are:

$$\|b_1^*\| = \sqrt{55}, \quad \|b_2^*\| = \frac{\sqrt{134255}}{55}, \quad \|b_3^*\| = \frac{\sqrt{80550559}}{2441}, \quad \|b_4^*\| = \frac{287\sqrt{32999}}{32999}.$$

Even though these lengths are irrational numbers, their product is an integer:

$$\|b_1^*\| \cdot \|b_2^*\| \cdot \|b_3^*\| \cdot \|b_4^*\| = 287.$$

This is not unexpected since $\mathrm{vol}(L) = \prod_{i=1}^4 \|b_i^*\|$, as we will prove in Theorem 8.8.

The matrix of Gram-Schmidt coefficients μ_{ij} is the following lower-triangular matrix, where $\mu_{ij} = 1$ if $i = j$ and $\mu_{ij} = 0$ if $i < j$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{12}{55} & 1 & 0 & 0 \\ \frac{1}{11} & \frac{60}{2441} & 1 & 0 \\ -\frac{2}{55} & -\frac{2994}{2441} & \frac{10913}{32999} & 1 \end{bmatrix}.$$

Applications to lattices. Let $B = [b_1, \ldots, b_n]$ be a basis of a full-rank integer lattice L, and let $B^* = [b_1^*, \ldots, b_n^*]$ be its *Gram-Schmidt orthogonalization*, so

$$b_1^* = b_1 \text{ and } b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^* \text{ for } 2 \le i \le n \text{ where } \mu_{ij} = \frac{\langle b_i, b_j^* \rangle}{\|b_j^*\|^2}. \tag{45}$$

While B^* is a basis of $\mathbb{R}^n$, it is typically *not* a basis of the lattice L since the Gram-Schmidt coefficients μ_{ij} need not be integers. Nevertheless, as the next two theorems illustrate, B^* reveals useful information about L, namely a lower bound for the first successive minimum and a formula for the lattice volume.

Theorem 8.7 (*lower bound for the first successive minimum*) Let $B = [b_1, \ldots, b_n]$ be a basis of a full-rank integer lattice $L \subseteq \mathbb{R}^n$, and let $B^* = [b_1^*, \ldots, b_n^*]$ be its Gram-Schmidt orthogonalization. Then $\lambda_1(L) \ge \min_i \|b_i^*\|$.

Proof: Let $v = \sum_{i=1}^n v_i b_i$ be a nonzero lattice vector, and let k be the largest index with $v_k \ne 0$ so $v = \sum_{i=1}^k v_i b_i$. Now,

$$\langle v, b_k^* \rangle = \langle \sum_{i=1}^k v_i b_i, b_k^* \rangle = \langle \sum_{i=1}^{k-1} v_i b_i, b_k^* \rangle + \langle v_k b_k, b_k^* \rangle = v_k \cdot \langle b_k^*, b_k^* \rangle = v_k \cdot \|b_k^*\|^2.$$

Hence the projection of v onto the one-dimensional subspace $\langle b_k^* \rangle$ is

$$\Pi_{\langle b_k^* \rangle}(v) = \frac{\langle v, b_k^* \rangle}{\|b_k^*\|^2} b_k^* = \frac{v_k \|b_k^*\|^2}{\|b_k^*\|^2} b_k^* = v_k b_k^*,$$

from which we conclude that

$$\|v\| \ge \|v_k b_k^*\| = |v_k| \cdot \|b_k^*\| \ge \min_i \|b_i^*\|$$

since $|v_k| \ge 1$. This shows that $\lambda_1(L) \ge \min_i \|b_i^*\|$. □

The proof of Theorem 8.7 can be modified to yield lower bounds on all the successive minima; see Exercise 8.5.

Theorem 8.8 (*volume of a lattice*) Let $B = [b_1, \ldots, b_n]$ be a basis of a full-rank lattice L, and let $B^* = [b_1^*, \ldots, b_n^*]$ be its Gram-Schmidt orthogonalization. Then

$$\mathrm{vol}(L) = \prod_{i=1}^{n} \|b_i^*\|. \tag{46}$$

Proof: Let B, B^* and Q be the $n \times n$ matrices whose ith columns are b_i, b_i^* and $b_i^*/\|b_i^*\|$, respectively. Since B^* is an orthogonal basis, Q is an orthogonal matrix, meaning $Q^T Q = I_n$; hence $\det(Q) = \pm 1$. Let U be the following upper-triangular matrix:

$$U = \begin{bmatrix} 1 & \mu_{21} & \mu_{31} & \cdots & \mu_{n1} \\ & 1 & \mu_{32} & \cdots & \mu_{n2} \\ & & 1 & \cdots & \mu_{n3} \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}_{n \times n}$$

and notice that $\det(U) = 1$. Let D be the $n \times n$ diagonal matrix with diagonal entries $\|b_1^*\|, \ldots, \|b_n^*\|$. By definition of Gram-Schmidt orthogonalization, we have $B = B^* U$. Furthermore, $B^* = QD$, so $B = QDU$. Taking the absolute values of the determinants of both sides gives

$$\mathrm{vol}(L) = |\det(B)| = |\det(QDU)| = |\det(Q)\ \det(D)\ \det(U)| = |\det(D)| = \prod_{i=1}^{n} \|b_i^*\|,$$

as required. □

Let $L = L(B)$ be an integer lattice, implying that $\mathrm{vol}(L) = |\det(B)|$ is a positive integer. By Theorem 8.8, the product $\prod_{i=1}^{n} \|b_i^*\|$ of the lengths of the Gram-Schmidt basis vectors is also a positive integer even though the $\|b_i^*\|$'s are typically not integers.

Remark 8.9 Gram-Schmidt orthogonalization can be interpreted geometrically as transforming the fundamental parallelepiped determined by n linearly independent vectors $[b_1, \ldots, b_n]$ in $\mathbb{R}^n$ into an orthogonal rectangular box (also called an *orthotope*) of equal volume. Each new edge b_i^* of the rectangular box is the height of the original parallelepiped in a new independent direction. Figures 8.2 and 8.3 show examples for $n = 2$ and $n = 3$.

8.2 Gauss's algorithm

Let $[u, v]$ be a basis for a two-dimensional integer lattice L, ordered so that $\|u\| \le \|v\|$. Gauss's algorithm is a polynomial-time algorithm for finding a shortest possible basis of L, namely a basis $[u', v']$ satisfying $\|u'\| = \lambda_1(L)$ and $\|v'\| = \lambda_2(L)$. The geometric idea

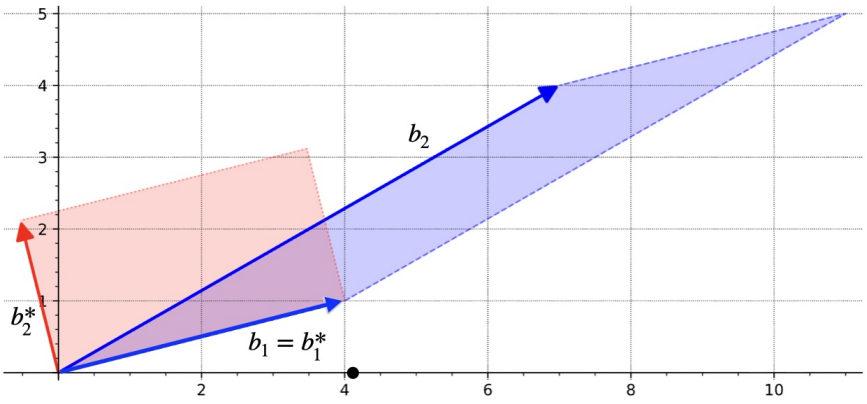


Figure 8.2: The fundamental parallelepipeds spanned by the lattice vectors $b_1 = (4, 1)$ and $b_2 = (7, 4)$ (shaded blue), and by the Gram-Schmidt vectors $b_1^* = (4, 1)$ and $b_2^* = (-9/17, 36/17)$ (shaded red); both have area 9.

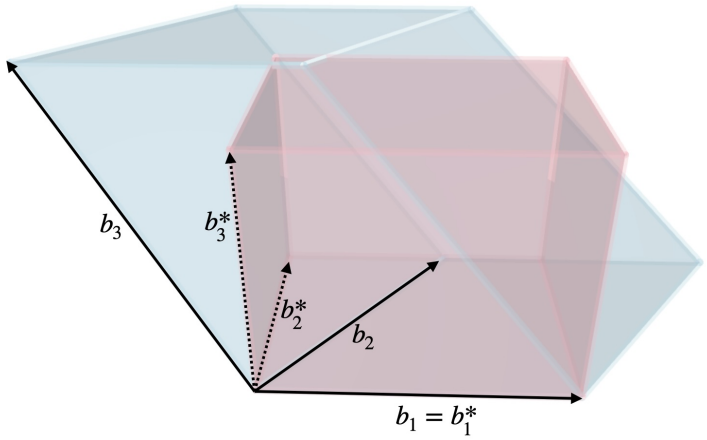


Figure 8.3: The fundamental parallelepipeds spanned by the lattice vectors $b_1 = (6, 1, 0)$, $b_2 = (3, 5, 0)$ and $b_3 = (-6, 3, 5)$ (shaded blue), and by the Gram-Schmidt vectors $b_1^* = (6, 1, 0)$, $b_2^* = (-27/37, 162/37, 0)$ and $b_3^* = (0, 0, 5)$ (shaded red); both have volume 135.

behind Gauss's algorithm is the following; see Figure 8.4. Consider the projection of v onto the one-dimensional subspace spanned by u. Using formula (43), this projection is

$$\Pi_{\langle u\rangle}(v) = \mu u, \quad \text{where } \mu = \frac{\langle u, v\rangle}{\|u\|^2}.$$

Subtracting this component from v leaves the orthogonal component

$$\Pi_{\langle u\rangle^\perp}(v) = v - \mu u.$$

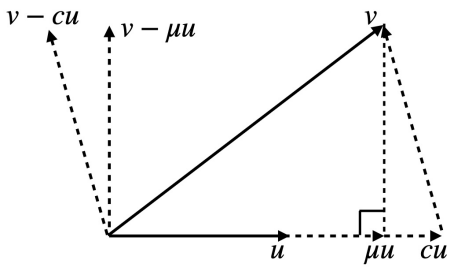


Figure 8.4: An iteration of Gauss's algorithm.

If μ were an integer, then $v - \mu u$ would itself be a lattice vector, and replacing v by this vector would immediately yield a shorter basis. In general, however, μ is rational but not integral, so $v - \mu u$ does not lie in the lattice. So, even though the vectors u and $v - \mu u$ are orthogonal, they do not form a basis of L. To address this, let[12] $c = \lfloor \mu \rceil$ be the closest integer to μ with ties broken by rounding towards 0 (so $\lfloor 3.5 \rceil = 3$ and $\lfloor -3.5 \rceil = -3$). Then $v - cu$ *is* a lattice vector, and it closely approximates the orthogonal projection $v - \mu u$. As a result, $v - cu$ is typically shorter than v and nearly orthogonal to u. Consequently, the pair $[u, v - cu]$ is a new basis of L that is usually shorter than the original one.

Gauss's algorithm (Algorithm 8.10) repeatedly applies this operation: replace v by $v - cu$, and then swap u and the new v to maintain the ordering $\|u\| \le \|v\|$. This procedure terminates once $c = 0$, which occurs exactly when $|\mu| \le \frac{1}{2}$.

Example 8.11 (*Gauss's algorithm*) Consider the lattice $L \subseteq \mathbb{R}^2$ with basis vectors $u = (3, 5)$, $v = (4, 7)$. Here are the steps of Gauss's algorithm for finding a reduced basis of L.

iteration	u	$\|u\|^2$	v	$\|v\|^2$	μ	c	$v - cu$
1	$(3, 5)$	34	$(4, 7)$	65	$47/34$	1	$(1, 2)$
2	$(1, 2)$	5	$(3, 5)$	34	$13/5$	3	$(0, -1)$
3	$(0, -1)$	1	$(1, 2)$	5	-2	-2	$(1, 0)$
4	$(0, -1)$	1	$(1, 0)$	1	0	0	$(1, 0)$

[12] In §8, §9, and §10, the notation $\lfloor \cdot \rceil$ denotes rounding to the nearest integer with ties broken *towards* 0. This contrasts with the notation $\lceil \cdot \rfloor$ used in §4, §6, and §7, where ties are broken *away from* 0.

Algorithm 8.10: Gauss's algorithm

Input: A basis $[u, v]$ of an integer lattice $L \subseteq \mathbb{R}^2$ with $\|u\| \leq \|v\|$.
Output: A basis $[u', v']$ of L with $\|u'\| = \lambda_1(L)$ and $\|v'\| = \lambda_2(L)$.

```
c ← 1.
while c ≠ 0 do
    Compute μ = ⟨u, v⟩/‖u‖² and c = ⌊μ⌉.
    Set v ← v − cu.
    if ‖u‖ > ‖v‖ then
        Swap u and v.
return ([u, v])
```

A reduced basis of L is $[u', v']$ where $u' = (0, -1)$ and $v' = (1, 0)$.

Example 8.12 (*Gauss's algorithm*) Consider the lattice $L \subseteq \mathbb{R}^2$ with basis vectors $u = (47928, 63649)$, $v = (68827, 91412)$. Here are the steps of Gauss's algorithm for finding a reduced basis of L.

#	u	$\|u\|^2$	v	$\|v\|^2$	μ	c	$v - cu$
1	(47928, 63649)	6348288385	(68827, 91412)	13093309673	$\frac{9117022844}{6348288385}$	1	(20899, 27763)
2	(20899, 27763)	1207552370	(47928, 63649)	6348288385	$\frac{2768734459}{1207552370}$	2	(6130, 8123)
3	(6130, 8123)	103560029	(20899, 27763)	1207552370	$\frac{353629719}{103560029}$	3	(2509, 3394)
4	(2509, 3394)	17814317	(6130, 8123)	103560029	$\frac{42949632}{17814317}$	2	(1112, 1335)
5	(1112, 1335)	3018769	(2509, 3394)	17814317	$\frac{7320998}{3018769}$	2	(285, 724)
6	(285, 724)	605401	(1112, 1335)	3018769	$\frac{1283460}{605401}$	2	(542, -113)
7	(542, -113)	306533	(285, 724)	605401	$\frac{72658}{306533}$	0	(285, 724)

A reduced basis of L is $[u', v']$ where $u' = (542, -113)$ and $v' = (285, 724)$.

It is immediate that the output of Gauss's algorithm is a basis of L. One can also show that the algorithm always terminates after at most $O(\log X)$ iterations, where $X = \|v\|$, and that all arithmetic operations involve integers that are $\leq X$. Consequently, the total running time of Gauss's algorithm is $O(\log^3 X)$ bit operations. We omit the proofs of these statements and instead present a proof that the algorithm produces a shortest possible basis.

Theorem 8.13 Let $[u, v]$ be the output of Gauss's algorithm. Then $\|u\| = \lambda_1(L)$ and $\|v\| = \lambda_2(L)$.

Proof: We will first prove that $\|u\| = \lambda_1(L)$. Let w be any nonzero lattice vector and write $w = au + bv$ where $a, b \in \mathbb{Z}$; note that a and b cannot both be 0 since $w \neq 0$. We have

$$\begin{aligned}
\|w\|^2 &= \|au + bv\|^2 \\
&= \langle au + bv, au + bv \rangle \\
&= a^2\|u\|^2 + 2ab\,\langle u, v\rangle + b^2\|v\|^2 \\
&\geq a^2\|u\|^2 + 2ab\langle u, v\rangle + b^2\|u\|^2 \quad \text{(since } \|v\| \geq \|u\|\text{)} \\
&\geq a^2\|u\|^2 - 2|ab| \cdot |\langle u, v\rangle| + b^2\|u\|^2 \\
&\geq a^2\|u\|^2 - |ab| \cdot \|u\|^2 + b^2\|u\|^2 \quad \text{(since } |\mu| = |\langle u, v\rangle|/\|u\|^2 \leq \frac{1}{2}\text{)} \\
&= (a^2 - |ab| + b^2)\|u\|^2.
\end{aligned}$$

Now, for $a, b \in \mathbb{R}$, define $f(a, b) = a^2 - |ab| + b^2$. We have $f(a, b) = (|a| - \frac{|b|}{2})^2 + \frac{3}{4}b^2 \geq 0$, with $f(a, b) = 0$ if and only if $a = b = 0$. Thus, if a and b are integers, not both 0, then $a^2 - |ab| + b^2 \geq 1$. It follows that $\|w\|^2 \geq \|u\|^2$, and thus $\|w\| \geq \|u\|$. This establishes that $\|u\| = \lambda_1(L)$.

We will next prove that $\|v\| = \lambda_2(L)$. Let w be any lattice vector that is not a multiple of u. Then we can write $w = au + bv$ where $a, b \in \mathbb{Z}$ with $b \neq 0$. We have

$$\begin{aligned}
\|w\|^2 &= \|au + bv\|^2 \\
&= \langle au + bv, au + bv \rangle \\
&= a^2\|u\|^2 + 2ab\langle u, v\rangle + b^2\|v\|^2 \\
&= b^2(\|v\|^2 - \|u\|^2) + (a^2 + b^2)\|u\|^2 + 2ab\langle u, v\rangle \\
&\geq b^2(\|v\|^2 - \|u\|^2) + (a^2 + b^2)\|u\|^2 - 2|ab| \cdot |\langle u, v\rangle| \\
&\geq b^2(\|v\|^2 - \|u\|^2) + (a^2 - |ab| + b^2)\|u\|^2 \quad \text{(since } |\mu| = |\langle u, v\rangle|/\|u\|^2 \leq \frac{1}{2}\text{)} \\
&\geq b^2(\|v\|^2 - \|u\|^2) + \|u\|^2 \quad \text{(since } a^2 - |ab| + b^2 \geq 1\text{)} \\
&= (b^2 - 1)(\|v\|^2 - \|u\|^2) + \|v\|^2 \\
&\geq \|v\|^2 \quad \text{(since } b \neq 0 \text{ and } \|v\| \geq \|u\|\text{)}.
\end{aligned}$$

Thus, $\|w\| \geq \|v\|$. Finally, suppose that $[u', v']$ are any two linearly independent lattice vectors with $\|u'\| \leq \|v'\|$. Then u' and v' cannot both be multiples of u. If v' is not a multiple of u, then $\|v'\| \geq \|v\|$ by what we showed above. On the other hand, if v' is a multiple of u, then u' is not a multiple of u, whence $\|u'\| \geq \|v\|$. We conclude that $\|v\| = \lambda_2(L)$. □

Definition 8.14 A basis $[u, v]$ of a lattice $L \subseteq \mathbb{Z}^2$ is called *Gauss-reduced* if $\|u\| \leq \|v\|$ and $|\mu| \leq \frac{1}{2}$ where $\mu = \langle u, v\rangle/\|u\|^2$.

Gauss's algorithm always produces a Gauss-reduced basis. As we have seen, this basis is a shortest possible basis of L. The notion of a Gauss-reduced basis extends naturally to integer lattices in higher dimensions. However, it turns out that such a reduced

lattice basis is not necessarily short. Instead, we will consider a stronger notion of reduced lattice basis due to Lovász. We will show that the vectors in such a reduced lattice basis are relatively short, and will present the LLL algorithm that efficiently finds such a basis.

8.3 The LLL algorithm

Let L be an n-dimensional integer lattice. As with Gauss's reduction in two dimensions, it is not difficult to find a lattice basis $B = [b_1, \ldots, b_n]$ so that the Gram-Schmidt coefficients μ_{ij} are small, specifically $|\mu_{ij}| \leq \frac{1}{2}$ for $1 \leq j < i \leq n$. This condition, known as *size reduction*, ensures that the basis vectors are reasonably close to being orthogonal. However, it does not guarantee that the basis vectors themselves are short.

Consider a randomly chosen lattice basis B. One expects that the Gram-Schmidt vectors $b_1^*, \ldots, b_n^*$ rapidly decrease in length. This is because each vector b_i^* is obtained by projecting b_i onto the orthogonal complement of the subspace spanned by the earlier basis vectors $b_1, \ldots, b_{i-1}$. As i increases, this orthogonal complement has smaller dimension, and consequently the projection of b_i typically tends to be shorter.

A key invariant of lattices comes into play here. Although the individual Gram-Schmidt vectors depend on the chosen lattice basis, the product of their lengths does not—it is equal to the volume of the lattice (see Theorem 8.8). The central idea behind the LLL algorithm is to exploit this fact by reshaping the lattice basis so that the lengths of the Gram-Schmidt vectors are distributed more evenly, rather than dropping off too quickly.

The mechanism that makes this possible is a sequence of local operations: *size reduction* and *swaps* of adjacent basis vectors. The swaps are designed to slow down the rate at which the Gram-Schmidt lengths decrease, thereby forcing earlier vectors to become relatively short. In particular, the first vector b_1^* is expected to be relatively short, and thus so will the first lattice basis vector b_1 (since $b_1 = b_1^*$). More precisely, we will prove that the first vector in the lattice basis produced by the LLL algorithm is guaranteed to have length no more than $2^{(n-1)/2}$ times the length of a shortest nonzero lattice vector.

8.3.1 LLL-reduced bases and the Lovász condition

We begin by formalizing what it means for a lattice basis to be reduced.

Definition 8.15 A lattice basis $[b_1, \ldots, b_n]$ is called *size-reduced* if $|\mu_{ij}| \leq \frac{1}{2}$ for all $1 \leq j < i \leq n$. A size-reduced basis is said to be *LLL-reduced* if it also satisfies the *Lovász condition*[13]:

$$\|b_k^*\|^2 \geq \left(\frac{3}{4} - \mu_{k,k-1}^2\right) \|b_{k-1}^*\|^2 \quad \text{for all } 2 \leq k \leq n. \tag{47}$$

[13] As further discussed in §8.4.2, the constant $\frac{3}{4}$ in the Lovász condition can be replaced by any constant δ where $\frac{1}{4} < \delta \leq 1$.

Since $0 \leq \mu_{k,k-1}^2 \leq \frac{1}{4}$, the Lovász condition implies that $\|b_k^*\|^2 \geq \frac{1}{2}\|b_{k-1}^*\|^2$. In other words, the Gram-Schmidt basis vectors are not allowed to shrink too quickly as we move along the basis.

Here are two equivalent ways to express the Lovász condition:

$$\|b_k^* + \mu_{k,k-1} b_{k-1}^*\|^2 \geq \frac{3}{4}\|b_{k-1}^*\|^2 \tag{48}$$

and

$$\|\Pi_{k-2}(b_k)\|^2 \geq \frac{3}{4}\|\Pi_{k-2}(b_{k-1})\|^2. \tag{49}$$

Recall from the proof of Theorem 8.3 that for $1 \leq i \leq n$, Π_i denotes the projection map $\Pi_{(V_i^*)^\perp} = \Pi_{V_i^\perp}$, where $V_i = \mathrm{Span}(b_1, \ldots, b_i)$, $V_i^* = \mathrm{Span}(b_1^*, \ldots, b_i^*)$, and $V_i = V_i^*$. The projection Π_0 is the identity map.

One of the most important guarantees offered by the LLL algorithm concerns the length of the first basis vector.

Theorem 8.16 Let $[b_1, \ldots, b_n]$ be an LLL-reduced basis of a lattice L. Then

$$\|b_1\| \leq 2^{(n-1)/2}\lambda_1(L). \tag{50}$$

Proof: For each $2 \leq i \leq n$, we have

$$\|b_i^*\|^2 \geq \left(\frac{3}{4} - \mu_{i,i-1}^2\right)\|b_{i-1}^*\|^2 \geq \left(\frac{3}{4} - \frac{1}{4}\right)\|b_{i-1}^*\|^2 = \frac{1}{2}\|b_{i-1}^*\|^2,$$

and so $\|b_{i-1}^*\|^2 \leq 2\|b_i^*\|^2$. Hence

$$\|b_1\|^2 = \|b_1^*\|^2 \leq 2^{i-1}\|b_i^*\|^2, \tag{51}$$

so

$$\|b_1\|^2 \leq 2^{n-1} \min_i \|b_i^*\|^2 \leq 2^{n-1}\lambda_1(L)^2$$

by Theorem 8.7. Taking square roots of both sides yields $\|b_1\| \leq 2^{(n-1)/2}\lambda_1(L)$. □

The following result gives an upper bound on the length of the first LLL-reduced basis vector in terms of the lattice volume. The proof is left as an exercise (Exercise 8.17).

Theorem 8.17 Let $[b_1, \ldots, b_n]$ be an LLL-reduced basis of a lattice L. Then

$$\|b_1\| \leq 2^{(n-1)/4}\,\mathrm{vol}(L)^{1/n}. \tag{52}$$

8.3.2 Size reduction

Algorithm 8.18 is a simple procedure for size reducing a lattice basis. The size reduction is performed in steps 2-5. In step 5, b_i is size-reduced with respect to b_j.

Theorem 8.19 Size reduction works, i.e., the basis $\tilde{B} = [\tilde{b}_1, \ldots, \tilde{b}_n]$ produced by Algorithm 8.18 is a size-reduced basis of $L(B)$. Moreover, size reduction leaves the Gram-Schmidt basis unchanged.

Algorithm 8.18: Size reduction

Input: Lattice basis $B = [b_1, \ldots, b_n]$
Output: Size-reduced basis $\tilde{B} = [\tilde{b}_1, \ldots, \tilde{b}_n]$

1 Compute the Gram-Schmidt orthogonalization $B^* = [b_1^*, \ldots, b_n^*]$.
2 **for** $i = 2$ **to** n **do**
3 **for** $j = i - 1$ **to** 1 **do**
4 Compute $\mu_{ij} = \langle b_i, b_j^* \rangle / \|b_j^*\|^2$.
5 Set $b_i \leftarrow b_i - \lfloor \mu_{ij} \rceil b_j$. // size-reduce b_i with respect to b_j
6 **return** $(\tilde{B} = [b_1, \ldots, b_n])$

Proof: Since each operation $b_i \leftarrow b_i - \lfloor \mu_{ij} \rceil b_j$ in the inner loop preserves the basis, $\tilde{B}$ is a basis for $L(B)$.

We claim that the Gram-Schmidt orthogonalization of $\tilde{B}$ is identical to that of B, i.e., $\tilde{b}_i^* = b_i^*$ for all $1 \le i \le n$. Let $V_i = \mathrm{Span}(b_1, \ldots, b_i)$ and $\tilde{V}_i = \mathrm{Span}(\tilde{b}_1, \ldots, \tilde{b}_i)$ for $1 \le i \le n$. Since each $\tilde{b}_i$ is obtained by subtracting integer multiples of earlier basis vectors $b_1, \ldots, b_{i-1}$ from b_i, we can write $\tilde{b}_i = b_i + c_i$ where $c_i \in V_{i-1}$. It follows that $V_i = \tilde{V}_i$ for all $1 \le i \le n$. Thus,

$$\tilde{b}_i^* = \Pi_{(\tilde{V}_{i-1})^\perp}(\tilde{b}_i) = \Pi_{(V_{i-1})^\perp}(\tilde{b}_i) = \Pi_{(V_{i-1})^\perp}(b_i + c_i) = \Pi_{(V_{i-1})^\perp}(b_i) = b_i^*,$$

which verifies the claim.

Lastly, we will verify that the Gram-Schmidt coefficients for the output basis are bounded in absolute value by $\frac{1}{2}$. Consider the operation

$$\hat{b}_i \leftarrow b_i - \lfloor \mu_{ij} \rceil b_j$$

in step 5 of Algorithm 8.18. This operation does not change any $\mu_{i'j'}$ with $i' < i$ and $j' < i'$ that was previously computed in step 4 since $\mu_{i'j'} = \langle b_{i'}, b_{j'}^* \rangle / \|b_{j'}^*\|^2$ and neither $b_{i'}$ nor $b_{j'}^*$ depend on b_i. Also, the operation does not change any previously computed $\mu_{ij'}$ with $j' > j$ since it subtracts an integer multiple of b_j from b_i, and $\langle b_j, b_{j'}^* \rangle = 0$. Finally, the new value of μ_{ij} is

$$\hat{\mu}_{ij} = \frac{\langle \hat{b}_i, b_j^* \rangle}{\|b_j^*\|^2} = \frac{\langle b_i, b_j^* \rangle}{\|b_j^*\|^2} - \lfloor \mu_{ij} \rceil \frac{\langle b_j, b_j^* \rangle}{\|b_j^*\|^2} = \mu_{ij} - \lfloor \mu_{ij} \rceil$$

since $\langle b_j, b_j^* \rangle = \langle b_j^*, b_j^* \rangle$. Thus, $|\hat{\mu}_{ij}| \le \frac{1}{2}$. □

8.3.3 Algorithm description

The LLL lattice basis reduction algorithm (Algorithm 8.20) begins by size reducing the input basis. It then swaps two adjacent basis vectors b_{j-1} and b_j for which the Lovász

condition is violated. The purpose of the swap is to reduce the rate at which the Gram-Schmidt vectors decrease in length. If multiple violations occur, the algorithm is free to choose any one of them[14]; different choices may lead to different LLL-reduced bases, but all offer the same guarantees. After the swap, the new basis is size reduced. The process of swapping a pair of adjacent basis vectors and size reducing the new basis is repeated until all adjacent pairs satisfy the Lovász condition, at which point the basis is LLL-reduced.

Algorithm 8.20: LLL lattice basis reduction algorithm

Input: Lattice basis $B = [b_1, \ldots, b_n]$
Output: LLL-reduced basis $[b_1, \ldots, b_n]$

1 Size-reduce $[b_1, \ldots, b_n]$ using Algorithm 8.18.
2 **if** there exists an index $j \in [2, n]$ such that $\|b_j^*\|^2 < (\frac{3}{4} - \mu_{j,j-1}^2)\|b_{j-1}^*\|^2$ **then**
3 Swap b_{j-1} and b_j.
4 Go to step 1.
5 **return** $([b_1, \ldots, b_n])$

Example 8.21 (*LLL algorithm*) In Example 8.6, we considered the basis

$$B = [(-5, -2, 5, 1),\ (3, 5, 2, 3),\ (0, 1, 2, -3),\ (-3, -4, -4, -5)]$$

for a full-rank integer lattice $L \subseteq \mathbb{Z}^4$ with $\mathrm{vol}(L) = 287$. The squared lengths of the basis vectors are:

$$\|b_1\|^2 = 55, \quad \|b_2\|^2 = 47, \quad \|b_3\|^2 = 14, \quad \|b_4\|^2 = 66.$$

An LLL-reduced basis for B is

$$\tilde{B} = [(0, 1, -2, -2),\ (0, 1, 2, -3),\ (-5, -1, -1, 0),\ (-2, 4, 1, 3)].$$

The squared lengths of the vectors in $\tilde{B}$ are:

$$\|\tilde{b}_1\|^2 = 9, \quad \|\tilde{b}_2\|^2 = 14, \quad \|\tilde{b}_3\|^2 = 27, \quad \|\tilde{b}_4\|^2 = 30.$$

Minkowski's Theorem (Theorem 2.15) guarantees that a shortest nonzero vector v in L satisfies

$$\|v\| = \lambda_1(L) \leq \sqrt{n}\ \mathrm{vol}(L)^{1/n} = \sqrt{4} \times 287^{1/4} \approx 8.23. \tag{53}$$

The lattice vector $\tilde{b}_1 = (0, 1, -2, -2)$ has length 3, showing that the bound in (53) is not tight in this case. An exact SVP solver confirms that $\lambda_1(L) = 3$, and hence the vector $\tilde{b}_1$ appearing in the LLL-reduced basis is indeed a shortest nonzero vector in the lattice.

[14] In the examples of the LLL algorithm, j was always chosen to be the *smallest* index for which the Lovász condition is violated.

Remark 8.22 To better appreciate the intricacies of the LLL algorithm, the reader is strongly encouraged to implement the algorithm independently using a mathematics software system. SageMath (`sagemath.org`) is particularly well suited for this purpose, as it is open source and based on a Python-like language. The reader can validate their own LLL implementation by comparing it with SageMath's built-in routine, and may also work through the computational exercises in §8.5 and §10.7.

Figure 8.5 shows SageMath script for computing the Gram-Schmidt orthogonalization and the associated Gram-Schmidt coefficients of a lattice basis (see Example 8.6), an LLL-reduced basis of the lattice (see Example 8.21), and a shortest nonzero vector in L (using the "enumeration" method that solves SVP exactly).

```
# The rows of matrix B are the basis vectors for an integer lattice L
B = Matrix(ZZ,[[-5,-2,5,1],[3,5,2,3],[0,1,2,-3],[-3,-4,-4,-5]])

print(det(B))            # Determine the volume of L

C,mu = B.gram_schmidt()  # Determine the Gram-Schmidt basis and coefficients
print(C)
print(mu)

D = B.LLL(0.75)          # Determine a (3/4)-LLL-reduced basis for L
print(D)

# Determine a shortest nonzero vector in L
from sage.modules.free_module_integer import IntegerLattice
L = IntegerLattice(B)
v = L.shortest_vector()
print(v)
```

Figure 8.5: SageMath scripts (see Remark 8.22).

Example 8.23 (*LLL algorithm*) Table 8.1 shows the intermediate basis matrices in one application of the LLL algorithm to a 5-dimensional lattice L. The components of the initial basis vectors were randomly selected from the interval $[0, 99]$. Note that the basis vectors are the *rows* of the matrix. The lengths of the vectors in the initial basis are 138.3, 66.4, 87.7, 164.8, and 110.7, respectively. The lengths of the vectors in the LLL-reduced basis are 34.8, 33.4, 42.0, 43.2, and 37.3, respectively. SageMath was used to confirm that the second vector $(16, -4, 21, -20, 2)$ is indeed a shortest nonzero lattice vector.

8.3.4 Termination

By the termination condition of Algorithm 8.20, the output of the LLL algorithm is an LLL-reduced lattice basis. Natural questions are whether the LLL algorithm must eventually terminate, and if so, how quickly. To show that the algorithm terminates, we introduce the important notion of a *potential function*.

Initial basis		Size reduction
$\begin{bmatrix} 47 & 83 & 38 & 53 & 76 \\ 24 & 15 & 49 & 23 & 26 \\ 30 & 43 & 30 & 26 & 58 \\ 92 & 69 & 80 & 73 & 47 \\ 50 & 76 & 37 & 34 & 38 \end{bmatrix}$	$\longrightarrow$	$\begin{bmatrix} 47 & 83 & 38 & 53 & 76 \\ 24 & 15 & 49 & 23 & 26 \\ -17 & -40 & -8 & -27 & -18 \\ 10 & -41 & -34 & -27 & -41 \\ 33 & 36 & 29 & 7 & 20 \end{bmatrix}$
Swap(1,2) + size reduction		Swap(2,3) + size reduction
$\longrightarrow \begin{bmatrix} 24 & 15 & 49 & 23 & 26 \\ -1 & 53 & -60 & 7 & 24 \\ 7 & -25 & 41 & -4 & 8 \\ 34 & -26 & 15 & -4 & -15 \\ 9 & 21 & -20 & -16 & -6 \end{bmatrix}$	$\longrightarrow$	$\begin{bmatrix} 24 & 15 & 49 & 23 & 26 \\ 7 & -25 & 41 & -4 & 8 \\ -11 & -12 & -27 & -24 & 14 \\ 27 & -1 & -26 & 0 & -23 \\ 16 & -4 & 21 & -20 & 2 \end{bmatrix}$
Swap(1,2) + size reduction		Swap(2,3) + size reduction
$\longrightarrow \begin{bmatrix} 7 & -25 & 41 & -4 & 8 \\ 17 & 40 & 8 & 27 & 18 \\ -11 & -12 & -27 & -24 & 14 \\ 27 & -1 & -26 & 0 & -23 \\ 16 & -4 & 21 & -20 & 2 \end{bmatrix}$	$\longrightarrow$	$\begin{bmatrix} 7 & -25 & 41 & -4 & 8 \\ -11 & -12 & -27 & -24 & 14 \\ 6 & 28 & -19 & 3 & 32 \\ 27 & -1 & -26 & 0 & -23 \\ 16 & -4 & 21 & -20 & 2 \end{bmatrix}$
Swap(1,2) + size reduction		Swap(4,5) + size reduction
$\longrightarrow \begin{bmatrix} -11 & -12 & -27 & -24 & 14 \\ 7 & -25 & 41 & -4 & 8 \\ 6 & 28 & -19 & 3 & 32 \\ 27 & -1 & -26 & 0 & -23 \\ 9 & 21 & -20 & -16 & -6 \end{bmatrix}$	$\longrightarrow$	$\begin{bmatrix} -11 & -12 & -27 & -24 & 14 \\ 7 & -25 & 41 & -4 & 8 \\ 6 & 28 & -19 & 3 & 32 \\ 9 & 21 & -20 & -16 & -6 \\ 18 & -22 & -6 & 16 & -17 \end{bmatrix}$
Swap(3,4) + size reduction		Swap(2,3) + size reduction
$\longrightarrow \begin{bmatrix} -11 & -12 & -27 & -24 & 14 \\ 7 & -25 & 41 & -4 & 8 \\ 9 & 21 & -20 & -16 & -6 \\ 6 & 28 & -19 & 3 & 32 \\ 18 & -22 & -6 & 16 & -17 \end{bmatrix}$	$\longrightarrow$	$\begin{bmatrix} -11 & -12 & -27 & -24 & 14 \\ 9 & 21 & -20 & -16 & -6 \\ 16 & -4 & 21 & -20 & 2 \\ -3 & 7 & 1 & 19 & 38 \\ 18 & -22 & -6 & 16 & -17 \end{bmatrix}$
Swap(1,2) + size reduction		Swap(2,3) + size reduction
$\longrightarrow \begin{bmatrix} 9 & 21 & -20 & -16 & -6 \\ -11 & -12 & -27 & -24 & 14 \\ 16 & -4 & 21 & -20 & 2 \\ -3 & 7 & 1 & 19 & 38 \\ 18 & -22 & -6 & 16 & -17 \end{bmatrix}$	$\longrightarrow$	$\begin{bmatrix} 9 & 21 & -20 & -16 & -6 \\ 16 & -4 & 21 & -20 & 2 \\ -11 & -12 & -27 & -24 & 14 \\ -3 & 7 & 1 & 19 & 38 \\ 18 & -22 & -6 & 16 & -17 \end{bmatrix}$

Table 8.1: Intermediate bases in an execution of the LLL algorithm (see Example 8.23). The order of the steps is left to right, top to bottom.

Consider a lattice L with basis $B = [b_1, \ldots, b_n]$ and associated Gram-Schmidt orthogonalization $B^* = [b_1^*, \ldots, b_n^*]$. For each $1 \leq \ell \leq n$, define

$$B_\ell = [b_1, \ldots, b_\ell] \text{ and } L_\ell = L(B_\ell). \tag{54}$$

Observe that L_ℓ is a rank-ℓ sublattice of L.

Remark 8.24 (*lattices that do not have full rank*) The notion of volume of a full-rank lattice (Definition 2.11) extends naturally to low-rank lattices. Let $L_\ell = L(B_\ell) \subseteq \mathbb{R}^n$ be a lattice of rank ℓ, where $1 \leq \ell \leq n$. The volume of L_ℓ is defined to be

$$\text{vol}(L_\ell) = \sqrt{\det(B_\ell^T B_\ell)}. \tag{55}$$

Note that B_ℓ is an $n \times \ell$ matrix, so $B_\ell^T B_\ell$ is an $\ell \times \ell$ matrix. Let the Gram-Schmidt orthogonalization of B_ℓ be $B_\ell^* = [b_1^*, \ldots, b_\ell^*]$. Theorem 8.8 extends naturally to low-rank lattices, so

$$\text{vol}(L_\ell) = \prod_{i=1}^{\ell} \|b_i^*\|. \tag{56}$$

Definition 8.25 Let $B = [b_1, \ldots, b_n]$ be a basis of a full-rank lattice. For each $1 \leq \ell \leq n$, define

$$d_\ell = \text{vol}(L_\ell)^2 = \prod_{i=1}^{\ell} \|b_i^*\|^2. \tag{57}$$

The *potential function* is

$$D(b_1, \ldots, b_n) = \prod_{\ell=1}^{n} d_\ell = \prod_{i=1}^{n} \|b_i^*\|^{2(n+1-i)}. \tag{58}$$

The potential function assigns greater weight to the earlier Gram-Schmidt vectors. Since we are only considering integer lattices, $d_\ell = \text{vol}(L_\ell)^2 = \det(B_\ell^T B_\ell)$ is a positive integer, and hence $D = D(b_1, \ldots, b_n)$ is also a positive integer.

Theorem 8.26 The LLL algorithm terminates after $O(n^2 \log X)$ swaps, where X is the maximum length of the original basis vectors.

Proof: The only operation in the LLL algorithm that changes any sublattice L_ℓ is the Swap operation. Consider Swap(b_{j-1}, b_j) for some $2 \leq j \leq n$, which only changes L_{j-1}. After this swap, the new lattice basis is $C = [c_1, \ldots, c_n]$, where $c_{j-1} = b_j$, $c_j = b_{j-1}$, and $c_i = b_i$ for all $i \neq j-1, j$. The new Gram-Schmidt orthogonalization is $C^* = [c_1^*, \ldots, c_n^*]$, where $c_i^* = b_i^*$ for all $i \neq j-1, j$. Now,

$$\|c_{j-1}^*\|^2 = \|\Pi_{j-2}(c_{j-1})\|^2 = \|\Pi_{j-2}(b_j)\|^2 = \|b_j^* + \mu_{j,j-1} b_{j-1}^*\|^2 < \frac{3}{4}\|b_{j-1}^*\|^2.$$

Thus,

$$\|c_{j-1}^*\|^2 < \frac{3}{4}\|b_{j-1}^*\|^2, \tag{59}$$

say $\|c_{j-1}^*\|^2 = \varepsilon\|b_{j-1}^*\|^2$ for some $\varepsilon < \frac{3}{4}$. Let $\tilde{d}_i = \prod_{j=1}^{i} \|c_j^*\|^2$ for $1 \le i \le n$. Then we have $\tilde{d}_{j-1} = \varepsilon d_{j-1}$, whereas $\tilde{d}_i = d_i$ for all $i \ne j-1$. It follows that the potential of the new lattice basis C is

$$\tilde{D} = D(c_1, \ldots, c_n) = \prod_{\ell=1}^{n} \tilde{d}_\ell = \varepsilon \prod_{\ell=1}^{n} d_\ell = \varepsilon D.$$

Consequently, the potential D decreases by a multiplicative factor greater than $\frac{4}{3}$ after each swap.

Now, the potential of the initial basis is

$$D = \prod_{i=1}^{n} \|b_i^*\|^{2(n+1-i)} \le \prod_{i=1}^{n} \|b_i\|^{2(n+1-i)} \le \left(\max_i \|b_i\|\right)^{2(1+2+\cdots+n)} = X^{n^2+n}. \tag{60}$$

Since the potential is a positive integer, the maximum number of swaps before the LLL algorithm terminates is

$$\log_{4/3} X^{n^2+n} = O(n^2 \log X),$$

which concludes the proof. □

8.3.5 Bit complexity

By Theorem 8.26, the total number of iterations executed by the LLL algorithm is polynomial in n and $\log X$. Each iteration consists of a swap followed by a size reduction, the number of arithmetic operations needed being bounded by a polynomial in n. It remains to show that the sizes of the integers arising during the execution of the algorithm do not grow exponentially.

We begin by proving that the Gram-Schmidt orthogonalization process can be carried out in polynomial time, and then show the same for the full LLL algorithm. We will use the following elementary observation to bound the bitlengths of integers appearing in the Gram-Schmidt orthogonalization process and in the LLL algorithm.

Lemma 8.27 Let $\alpha \in \mathbb{Q}$, and let $t \le X^{2n}$ be a positive integer such that $t\alpha \in \mathbb{Z}$ and $|t\alpha| \le \sqrt{n}X^{2n}$. Then $\alpha = s/t$, where both $|s|$ and t have bitlength $O(n \log X)$.

Proof: Since $t\alpha \in \mathbb{Z}$, we can write $\alpha = s/t$ for some $s \in \mathbb{Z}$. Moreover, $|s| = |t\alpha| \le \sqrt{n}X^{2n}$. Consequently, both $|s|$ and t have bitlength $O(n \log X)$. □

We now establish that the Gram-Schmidt orthogonalization process (Algorithm 8.5) runs in polynomial time.

Theorem 8.28 Let $B = [b_1, \ldots, b_n]$ be a basis of an integer lattice $L \subseteq \mathbb{Z}^n$, and let $X = \max_i \|b_i\|$. The Gram-Schmidt orthogonalization process for computing the Gram-Schmidt basis $B^* = [b_1^*, \ldots, b_n^*]$ is a polynomial-time algorithm[15].

Proof: Recall that $b_1^* = b_1$ and

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^* \text{ for } 2 \le i \le n, \text{ where } \mu_{ij} = \frac{\langle b_i, b_j^* \rangle}{\|b_j^*\|^2}.$$

Each μ_{ij} lies in $\mathbb{Q}$, and each vector b_i^* belongs to $\mathbb{Q}^n$. Our goal is to show that the bitlengths of the numerators and denominators appearing in the μ_{ij} and in the coordinates of the b_i^* do not grow exponentially.

For each $1 \le \ell \le n$, let B_ℓ, L_ℓ and d_ℓ be as defined in (54) and (57), and let $d_0 = 1$. Recall that each d_ℓ is a positive integer. Since $\|b_i^*\| \le \|b_i\| \le X$, it follows from (56) that $1 \le d_\ell \le X^{2\ell} \le X^{2n}$ for $1 \le \ell \le n$.

Fix $i \in [2, n]$. Because $b_i^* - b_i \in \mathrm{Span}(b_1, \ldots, b_{i-1})$, there exist unique coefficients $x_{ij} \in \mathbb{Q}$ such that

$$b_i^* = b_i + \sum_{j=1}^{i-1} x_{ij} b_j. \tag{61}$$

We next derive a formula for the x_{ij}.

For each $1 \le k \le i-1$, the orthogonality condition $\langle b_i^*, b_k \rangle = 0$ implies

$$\langle b_i^*, b_k \rangle = \langle b_i, b_k \rangle + \sum_{j=1}^{i-1} x_{ij} \langle b_j, b_k \rangle = 0,$$

and hence

$$\sum_{j=1}^{i-1} x_{ij} \langle b_j, b_k \rangle = -\langle b_i, b_k \rangle \text{ for } 1 \le k \le i-1. \tag{62}$$

This system of $i-1$ linear equations in $i-1$ unknowns can be written compactly as $Ax = c$ where $A = B_{i-1}^T B_{i-1}$, and where the j-th entries of vectors x and c are x_{ij} and $-\langle b_i, b_j \rangle$, respectively. Using Cramer's rule yields

$$x_{ij} = \frac{\det(A_j)}{\det(A)} = \frac{\det(A_j)}{\det(B_{i-1}^T B_{i-1})} = \frac{\det(A_j)}{d_{i-1}} \text{ for } 1 \le j \le i-1,$$

[15]Since $X = \max_i \|b_i\|$, every coordinate of each b_i has absolute value at most X. Therefore, each vector b_i can be represented using $O(n \log X)$ bits. Consequently, the total input size, i.e., the number of bits needed to represent the basis B, is $O(n^2 \log X)$. It follows that an algorithm taking B as input is considered polynomial-time if its running time is bounded by a polynomial in n and $\log X$.

where A_j is obtained from A by replacing the j-th column with c. Both $\det(A_j)$ and d_{i-1} are integers. Writing $e_j = \det(A_j)$, we have $x_{ij} = e_j/d_{i-1}$ for $1 \le j \le i-1$. Substituting this expression into (61) and multiplying both sides by d_{i-1} gives

$$d_{i-1}b_i^* = d_{i-1}b_i + \sum_{j=1}^{i-1} e_j b_j,$$

and therefore

$$d_{i-1}b_i^* \in \mathbb{Z}^n. \tag{63}$$

Furthermore,

$$\|d_{i-1}b_i^*\| = d_{i-1}\|b_i^*\| \le d_{i-1}\|b_i\| \le X^{2i-1} \le X^{2n}. \tag{64}$$

Thus, each coordinate of $d_{i-1}b_i^*$ is an integer with absolute value at most X^{2n}. It follows by Lemma 8.27 that the numerator and denominator of each coordinate of b_i^* have bitlength $O(n \log X)$.

We now bound the sizes of the Gram-Schmidt coefficients μ_{ij}. Observe that for each $1 \le j \le i-1$, we have

$$d_j\mu_{ij} = d_j \frac{\langle b_i, b_j^* \rangle}{\|b_j^*\|^2} = d_{j-1}\langle b_i, b_j^* \rangle = \langle b_i, d_{j-1}b_j^* \rangle \in \mathbb{Z}. \tag{65}$$

Moreover,

$$|d_j\mu_{ij}| = |\langle b_i, d_{j-1}b_j^* \rangle| \le \|b_i\| \cdot \|d_{j-1}b_j^*\| \le X \cdot X^{2j-1} = X^{2j} \le X^{2n},$$

where the first inequality follows from the Cauchy-Schwarz inequality. Hence,

$$|d_j\mu_{ij}| \le X^{2n} \text{ for } 1 \le j \le i-1. \tag{66}$$

Applying Lemma 8.27 once again, we conclude that each μ_{ij} has numerator and denominator of bitlength $O(n \log X)$.

We have therefore shown that all the integers arising during the Gram-Schmidt orthogonalization process have bitlength $O(n \log X)$, which completes the proof. □

We now prove that the LLL algorithm runs in polynomial time. Throughout, we use $\text{poly}(n, \log X)$ to denote a quantity of the form $O((n \log X)^c)$ for some constant c.

Theorem 8.29 Let $B = [b_1, \ldots, b_n]$ be a basis of a lattice $L \subseteq \mathbb{Z}^n$, and define $X = \max_i \|b_i\|$. The LLL algorithm outputs an LLL-reduced basis of L in time $\text{poly}(n, \log X)$.

Proof: The LLL algorithm (Algorithm 8.20) starts with an initial size reduction (Algorithm 8.18), followed by a sequence of $O(n^2 \log X)$ additional size reductions each preceded by a swap. To prove that LLL runs in polynomial time, it is therefore sufficient to show that each invocation of the size reduction procedure executes in $\text{poly}(n, \log X)$ time.

Claim 1. Throughout the execution of the LLL algorithm, $\|b_i^*\| \le X$ for all $1 \le i \le n$.
Proof of Claim 1: As observed in the proof of Theorem 8.26, the only operation in the

LLL algorithm that modifies the Gram-Schmidt vectors is a swap. Consider the operation $\text{Swap}(b_{j-1}, b_j)$ for some $2 \le j \le n$, and let $C = [c_1, \ldots, c_n]$ denote the new lattice basis. Then $c_{j-1} = b_j$, $c_j = b_{j-1}$, and $c_i = b_i$ for all $i \ne j-1, j$. Let $C^* = [c_1^*, \ldots, c_n^*]$ be the Gram-Schmidt orthogonalization of C. We have $c_i^* = b_i^*$ for all $i \ne j-1, j$, and $\|c_{j-1}^*\|^2 < \frac{3}{4}\|b_{j-1}^*\|^2$ (see equation (59)).

Now,

$$\begin{aligned} c_j^* &= c_j - \sum_{k=1}^{j-1} \frac{\langle c_j, c_k^* \rangle}{\|c_k^*\|^2} c_k^* \\ &= b_{j-1} - \sum_{k=1}^{j-2} \frac{\langle b_{j-1}, b_k^* \rangle}{\|b_k^*\|^2} b_k^* - \frac{\langle b_{j-1}, c_{j-1}^* \rangle}{\|c_{j-1}^*\|^2} c_{j-1}^* \\ &= b_{j-1}^* - \frac{\langle b_{j-1}, c_{j-1}^* \rangle}{\|c_{j-1}^*\|^2} c_{j-1}^*. \end{aligned} \tag{67}$$

Since

$$b_{j-1} = b_{j-1}^* + \sum_{k=1}^{j-2} \frac{\langle b_{j-1}, b_k^* \rangle}{\|b_k^*\|^2} b_k^*,$$

we have

$$\langle b_{j-1}, c_{j-1}^* \rangle = \langle b_{j-1}^*, c_{j-1}^* \rangle + \sum_{k=1}^{j-2} \frac{\langle b_{j-1}, b_k^* \rangle}{\|b_k^*\|^2} \langle b_k^*, c_{j-1}^* \rangle = \langle b_{j-1}^*, c_{j-1}^* \rangle$$

because $\langle b_k^*, c_{j-1}^* \rangle = \langle c_k^*, c_{j-1}^* \rangle = 0$ for $1 \le k \le j-2$. Thus, equation (67) can be written as

$$c_j^* = b_{j-1}^* - \frac{\langle b_{j-1}^*, c_{j-1}^* \rangle}{\|c_{j-1}^*\|^2} c_{j-1}^*.$$

This means that c_j^* is the projection of b_{j-1}^* onto the orthogonal complement of $\langle c_{j-1}^* \rangle$, implying that $\|c_j^*\| \le \|b_{j-1}^*\|$. Altogether, this shows that $\max_i \|c_i^*\| \le \max_i \|b_i^*\|$. Since initially $\|b_i^*\| \le X$, it follows that every Gram-Schmidt vector computed during the algorithm satisfies $\|b_i^*\| \le X$. □

Claim 2. At the conclusion of each invocation to Algorithm 8.18, the basis vectors satisfy $\|b_i\| \le \sqrt{n}\, X$ for all $1 \le i \le n$.

Proof of Claim 2: Since Algorithm 8.18 produces a size-reduced basis, we have

$$b_i = b_i^* + \sum_{j=1}^{i-1} \mu_{ij} b_j^* \quad \text{with } |\mu_{ij}| \le \frac{1}{2}.$$

Therefore,

$$\|b_i\|^2 = \|b_i^*\|^2 + \sum_{j=1}^{i-1} \mu_{ij}^2 \|b_j^*\|^2 \le X^2 + \frac{n-1}{4} X^2 \le nX^2,$$

which implies that $\|b_i\| \le \sqrt{n}\,X$. □

Claim 3. The Gram-Schmidt orthogonalization performed in each invocation of Algorithm 8.18 runs in time $\mathrm{poly}(n, \log X)$.

Proof of Claim 3: Let $B = [b_1, \ldots, b_n]$ be the input basis to an invocation of Algorithm 8.18, and let $B^* = [b_1^*, \ldots, b_n^*]$ denote its Gram-Schmidt orthogonalization. For $1 \le i \le n$, let d_i be as defined in (57), and let $d_0 = 1$. Note that $d_i \le X^{2i} \le X^{2n}$.

As shown in the proof of Theorem 8.28, $d_{i-1}b_i^* \in \mathbb{Z}^n$ for each $1 \le i \le n$. Consequently,

$$\|d_{i-1}b_i^*\| = d_{i-1}\|b_i^*\| \le X^{2i-2}X \le X^{2n}$$

for $2 \le i \le n$. Hence, Lemma 8.27 implies that the numerators and denominators of the components of the b_i^* have bitlength $O(n \log X)$.

The proof of Theorem 8.28 also established that $d_j\mu_{ij} = \langle b_i, d_{j-1}b_j^*\rangle \in \mathbb{Z}$ for all $1 \le j < i \le n$. Thus

$$|d_j\mu_{ij}| = |\langle b_i, d_{j-1}b_j^*\rangle| \le \|b_i\| \cdot d_{j-1} \cdot \|b_j^*\| \le \sqrt{n}X \cdot X^{2j-2} \cdot X \le \sqrt{n}X^{2n},$$

where the first inequality follows from the Cauchy-Schwarz inequality. Hence, we can deduce by Lemma 8.27 that the numerators and denominators of each μ_{ij} also have bitlength $O(n \log X)$. Finally, since the input basis vectors satisfy $\|b_i\| \le \sqrt{n}\,X$, it follows that the entire Gram-Schmidt orthogonalization can be computed in $\mathrm{poly}(n, \log X)$ time. □

Claim 4. The computations $\mu_{ij} = \langle b_i, b_j^*\rangle / \|b_j^*\|^2$ and $b_i \leftarrow b_i - \lfloor \mu_{ij} \rceil b_j$ in the inner loop of Algorithm 8.18 can be performed in time $\mathrm{poly}(n, \log X)$.

Proof of Claim 4: It suffices to show that the updated vector $v_{ij} = b_i - \lfloor \mu_{ij} \rceil b_j$ has bitlength $\mathrm{poly}(n, \log X)$. Indeed,

$$|\lfloor \mu_{ij} \rceil| = \left| \left\lfloor \frac{\langle b_i, b_j^*\rangle}{\|b_j^*\|^2} \right\rceil \right| \le 1 + \frac{\|b_i\| \cdot \|b_j^*\|}{\|b_j^*\|^2} = 1 + \frac{\|b_i\|}{\|b_j^*\|} \le 1 + X^{2n-2}\|b_i\| \le 2X^{2n-2}\|b_i\|, \quad (68)$$

where the first inequality follows from Cauchy-Schwarz, and the second uses the fact that $d_{j-1}b_j^* \in \mathbb{Z}^n$, implying $\|d_{j-1}b_j^*\| \ge 1$, whence $\|b_j^*\| \ge 1/d_{j-1} \ge 1/X^{2j-2} \ge 1/X^{2n-2}$. During the size reduction of b_i, all vectors b_j with $j < i$ have already been size-reduced. Therefore, applying (68) gives

$$\begin{aligned} \|v_{ij}\| &= \|b_i - \lfloor \mu_{ij} \rceil b_j\| \\ &\le \|b_i\| + |\lfloor \mu_{ij} \rceil| \cdot \|b_j\| \\ &\le \|b_i\| + 2X^{2n-2}\|b_i\| \cdot \|b_j\| \\ &\le (1 + 2X^{2n-2}\sqrt{n}X)\ \|b_i\| \\ &\le 4\sqrt{n}\,X^{2n}\|b_i\|. \end{aligned}$$

This shows that each time b_i is reduced in step 5, its length is at most $(4\sqrt{n}\,X^{2n})^n$ times the length of the b_i at the start of Algorithm 8.18. Thus,

$$\|v_{ij}\| \le (4\sqrt{n}\,X^{2n})^n\ \sqrt{n}\,X,$$

which implies that the bitlength of v_{ij} is $\text{poly}(n, \log X)$. This completes the proof. □

After incorporating some minor optimizations into the LLL algorithm, a more refined analysis (which we omit) yields the following explicit running-time bound.

Theorem 8.30 *(Lenstra, Lenstra, and Lovász)* The running time of the LLL algorithm is $O(n^6 \log^3 X)$ bit operations.

Remark 8.31 The notion of an LLL-reduced basis extends naturally to lattices that do not have full rank. The LLL algorithm can be applied to find LLL-reduced bases in such lattices while preserving its polynomial-time complexity bound.

Remark 8.32 Up to this point, we have assumed that the input to the LLL algorithm is an integer lattice. However, the algorithm extends naturally to rational lattices $L \subseteq \mathbb{Q}^n$, while preserving its polynomial-time complexity bound.

8.3.6 A dimension-15 example

We conclude this section with a more substantial illustration of the LLL algorithm.

Example 8.33 (*LLL algorithm*) We start with a randomly-chosen basis for a full-rank integer lattice L in fifteen dimensions. Each coordinate of the basis vectors was sampled uniformly at random from the interval $[-2, 2]$. This initial basis is displayed in Figure 8.6. By construction, the basis vectors are quite short, with lengths ranging from 4.47 to 6.7.

Next, we deliberately distorted the basis to obtain a much more skewed basis with significantly longer vectors. This is done by repeating the following operation 100 times: choose distinct indices $i, j \in_R [1, 15]$ and a multiplier $t \in_R [-25, 25]$, and replace basis vector b_i by $b_i - t \cdot b_j$. The resulting basis, which still spans the same lattice L, is shown in Figure 8.7.

Applying the LLL algorithm to this skewed basis produces the reduced basis shown in Figure 8.8. The resulting vectors are very short—indeed, even shorter than those of the original randomly generated basis. The shortest vector

$$v = (-2, -1, 1, -1, -1, 0, 1, 1, -1, 0, 0, 0, 0, 0, 1)$$

has length $\sqrt{12} \approx 3.46$, while the longest has length 5.0 (see also Table 8.3). Moreover, each coordinate of every basis vector lies in the interval $[-3, 3]$. SageMath was used to confirm that v is indeed a shortest nonzero vector in L.

In total, the algorithm performed 1157 swaps. Table 8.2 lists the first 66 swaps and the last 66 swaps. The table also records $\log_2 D$, the base-2 logarithm of the potential D, as well as $\log_2(\Delta D)$, the base-2 logarithm of the ratio of the new potential to the old one. Notice that the potential is strictly decreasing, and that $\log_2(\Delta D)$ is always less than $\log_2(3/4)$, i.e., $\log_2(\Delta D) < -0.415037$.

Table 8.3 reports the lengths of the original (distorted) lattice basis vectors b_i and their corresponding Gram-Schmidt basis vectors b_i^*. As expected, the Gram-Schmidt

Swap		$\log_2 D$	$\log_2(\Delta D)$	Swap		$\log_2 D$	$\log_2(\Delta D)$	Swap		$\log_2 D$	$\log_2(\Delta D)$
1	2	2550.7	-2.447960	2	3	2415.6	-1.859969	7	8	458.9	-0.964809
1	2	2547.6	-3.147762	1	2	2413.8	-1.837203	6	7	457.9	-1.036352
1	2	2536.8	-10.758942	2	3	2413.1	-0.683711	5	6	456.9	-0.969374
2	3	2533.0	-3.751924	3	4	2411.8	-1.301468	4	5	456.0	-0.946371
2	3	2529.8	-3.202972	2	3	2411.3	-0.540166	7	8	455.5	-0.527695
2	3	2520.1	-9.709751	4	5	2409.4	-1.895998	8	9	454.5	-0.931297
1	2	2517.8	-2.294698	3	4	2408.7	-0.657851	9	10	453.8	-0.773329
1	2	2514.1	-3.688553	5	6	2406.2	-2.456322	10	11	453.3	-0.451904
2	3	2512.0	-2.125418	4	5	2404.8	-1.441955	11	12	452.6	-0.701915
2	3	2505.5	-6.547939	3	4	2404.2	-0.599990	10	11	452.0	-0.607623
1	2	2502.2	-3.251262	2	3	2403.4	-0.753708	13	14	451.2	-0.842960
3	4	2499.4	-2.841144	1	2	2403.0	-0.448939	12	13	450.1	-1.084172
2	3	2498.6	-0.733080	3	4	2402.2	-0.830693	11	12	449.4	-0.708663
3	4	2494.7	-3.987387	2	3	2401.7	-0.494969	14	15	446.4	-2.963863
2	3	2491.0	-3.627501	5	6	2396.2	-5.509475	13	14	444.0	-2.389888
1	2	2487.9	-3.161642	4	5	2390.9	-5.266025	12	13	442.4	-1.615325
3	4	2484.5	-3.353147	3	4	2386.5	-4.412512	11	12	441.2	-1.191619
2	3	2482.2	-2.339682	2	3	2383.4	-3.086960	10	11	440.3	-0.911681
3	4	2481.3	-0.917240	1	2	2381.3	-2.071812	9	10	439.2	-1.075492
4	5	2477.4	-3.824909	5	6	2379.9	-1.460864	8	9	438.7	-0.528563
3	4	2475.1	-2.296750	4	5	2379.0	-0.858862	7	8	437.9	-0.781122
2	3	2473.5	-1.666312			.....		6	7	436.6	-1.307679
1	2	2472.4	-1.085987			.....		5	6	434.9	-1.681067
4	5	2470.0	-2.381498			.....		4	5	434.4	-0.531945
3	4	2467.1	-2.943220	9	10	484.6	-1.130753	3	4	433.9	-0.521572
2	3	2464.7	-2.340614	8	9	482.9	-1.669808	6	7	433.4	-0.432818
1	2	2462.9	-1.846870	7	8	482.2	-0.691385	7	8	432.4	-1.042172
4	5	2456.1	-6.781332	6	7	481.5	-0.728582	6	7	431.9	-0.519962
3	4	2450.8	-5.270088	5	6	480.4	-1.077897	8	9	430.2	-1.689900
2	3	2447.0	-3.855391	4	5	478.8	-1.630391	7	8	429.3	-0.891293
1	2	2445.7	-1.223059	3	4	477.2	-1.611077	6	7	428.6	-0.653534
4	5	2441.1	-4.619216	2	3	476.1	-1.087169	5	6	428.0	-0.677866
3	4	2439.1	-1.997948	1	2	475.2	-0.943416	4	5	427.3	-0.636691
5	6	2435.6	-3.564133	9	10	473.8	-1.344804	9	10	426.5	-0.818747
4	5	2434.1	-1.448619	10	11	473.0	-0.771700	10	11	425.2	-1.287720
3	4	2433.2	-0.957347	11	12	472.6	-0.474191	9	10	424.4	-0.857999
2	3	2432.1	-1.055315	12	13	471.5	-1.054905	8	9	423.4	-1.004275
3	4	2431.6	-0.534558	11	12	470.7	-0.836913	7	8	422.4	-1.004637
5	6	2428.7	-2.848068	10	11	470.2	-0.463727	6	7	421.8	-0.587346
4	5	2427.2	-1.510119	13	14	467.0	-3.184673	11	12	420.8	-0.950476
3	4	2426.3	-0.899297	12	13	464.6	-2.411948	10	11	420.2	-0.579370
4	5	2425.4	-0.918421	11	12	463.2	-1.433110	9	10	419.7	-0.575864
5	6	2421.4	-3.949843	10	11	462.1	-1.084950	12	13	418.8	-0.867230
4	5	2419.4	-2.057158	9	10	460.9	-1.161409	13	14	417.8	-0.941581
3	4	2417.5	-1.909732	8	9	459.9	-1.018782	14	15	417.2	-0.655858

Table 8.2: The first 66 and the last 66 swaps executed by the LLL algorithm in Example 8.33. The order of the swaps is top to bottom, left to right.

$$\begin{bmatrix}
2 & -1 & 2 & -1 & 0 & 1 & 1 & 2 & 0 & -2 & 2 & -1 & 0 & -1 & -1 \\
1 & 0 & 0 & -2 & -1 & 2 & 1 & 2 & -1 & 0 & -1 & 1 & -2 & 2 & 0 \\
-2 & 2 & 1 & 0 & 2 & 0 & 2 & 1 & 0 & -1 & 0 & -2 & -2 & 0 & 2 \\
2 & -1 & -1 & -1 & 0 & -1 & 1 & -1 & -2 & 2 & -2 & 1 & 1 & 2 & 0 \\
1 & -2 & -1 & 1 & 2 & 0 & 1 & -2 & -2 & -2 & -1 & -1 & 0 & 1 & 2 \\
-2 & 1 & -2 & -1 & -2 & 2 & -2 & -2 & 1 & 0 & 2 & 1 & 1 & 1 & -1 \\
-1 & -1 & 0 & 2 & 2 & 1 & 0 & 0 & 2 & 0 & 0 & 2 & 0 & 1 & 0 \\
-1 & 2 & -1 & 0 & 0 & 1 & 0 & -2 & 1 & -1 & 0 & -1 & -2 & -2 & -2 \\
2 & -1 & 1 & 2 & 1 & -2 & 0 & -2 & 0 & -1 & 1 & -2 & 2 & -1 & 0 \\
-1 & 1 & -2 & -2 & -1 & -2 & 2 & 0 & -1 & 2 & 2 & 0 & -2 & 2 & 1 \\
1 & 2 & 0 & 0 & -2 & 2 & 0 & -2 & -1 & -2 & 1 & 2 & -2 & -2 & -2 \\
0 & -2 & -2 & 0 & 0 & -2 & 1 & 2 & 1 & -2 & -1 & 2 & 2 & -2 & -1 \\
-2 & 2 & 1 & 1 & 2 & 1 & -1 & -2 & 1 & -2 & 2 & -2 & -2 & 2 & -2 \\
1 & 2 & 1 & 2 & -2 & 2 & -1 & 2 & 2 & 2 & -1 & 0 & -2 & 1 & 1 \\
-1 & -1 & -1 & 1 & -2 & 2 & -1 & 2 & -1 & 2 & 2 & -2 & 2 & -1 & -1
\end{bmatrix}_{15\times 15}$$

Figure 8.6: A randomly-selected short basis for a 15-dimensional lattice (Example 8.33). The basis vectors are the rows of the matrix.

lengths $\|b_i^*\|$ decrease rapidly. The table also includes the lengths of the LLL-reduced basis vectors $\tilde{b}_i$ and their Gram-Schmidt counterparts $\tilde{b}_i^*$. Notice that the lengths of the $\tilde{b}_i^*$ are approximately uniformly distributed. One can verify that $\prod_{i=1}^{15} \|b_i^*\|$ and $\prod_{i=1}^{15} \|\tilde{b}_i^*\|$ both equal the integer 23677332, which is the lattice volume.

Finally, Table 8.4 displays the Gram-Schmidt coefficients μ_{ij} of the LLL-reduced basis. As expected, these coefficients satisfy $|\mu_{ij}| \leq \frac{1}{2}$ for $1 \leq j < i \leq 15$.

8.4 LLL improvements

Since its introduction in 1982, the LLL algorithm has undergone numerous enhancements. We briefly describe some of these enhancements. The most impactful is the block Korkine-Zolotarev (BKZ) algorithm, a blockwise generalization of LLL that will be described in §9.

8.4.1 Scheduling the Gram-Schmidt computations

The efficiency of LLL can be improved by carefully scheduling the Gram-Schmidt computations. Rather than recomputing the entire Gram-Schmidt orthogonalization after each swap, one can organize the updates so that only those Gram-Schmidt vectors and coefficients μ_{ij} that are affected by the swap are recomputed.

For example, suppose that the 9th and 10th basis vectors are swapped. In this case,

$$
\begin{bmatrix}
-115876334270 & 95334076546 & 44582453025 & 57364088199 & 86794272932 \\
48376818744 & -62515108200 & -91698909830 & 60554412628 & -99007760874 \\
102291706978 & -91931888448 & -92286074031 & 67369918728 & -85178843944 \\
\hline
-413892649229 & 340531262846 & 159244076714 & 204897045599 & 310035529595 \\
172790730091 & -223283060344 & -327547371938 & 216281025848 & -353634883086 \\
365359051013 & -328376636917 & -329645272684 & 240667714177 & -304246036603 \\
\hline
-70725072756 & 58187121125 & 27210878264 & 35012148934 & 52974857343 \\
29526771769 & -38156066064 & -55968401462 & 36959352394 & -60429337352 \\
62433695384 & -56110596809 & -56326771578 & 41119227151 & -51988878529 \\
\hline
29344864262 & -24142613972 & -11290390935 & -14527068224 & -21980057784 \\
-12251045046 & 15831330296 & 23221766669 & -15334932342 & 25073054801 \\
-25904628042 & 23281234464 & 23370848279 & -17060537533 & 21570553617 \\
\hline
-65530319211 & 53712416149 & 25219705032 & 32425631689 & 48794843177 \\
27407490127 & -35485962608 & -51594189366 & 34386678123 & -56085446912 \\
58012967496 & -51869083935 & -51998985316 & 37521291477 & -48085362525 \\
\hline
1791158 & -1685872 & -655465 & -899941 & -1619153 \\
-707577 & 852136 & 1721096 & -804495 & 1436360 \\
-1417528 & 1529799 & 1618272 & -1627284 & 1445896 \\
\hline
-624001707 & 511464496 & 240151794 & 308767430 & 464636954 \\
260983533 & -337910637 & -491292191 & 327444016 & -534066865 \\
552421320 & -493913013 & -495149493 & 357279871 & -457881957 \\
\hline
-6428054113 & 5288506100 & 2473139854 & 3182181089 & 4814774407 \\
2683626559 & -3467925004 & -5086851386 & 3359158186 & -5492296131 \\
5674467972 & -5099775047 & -5119422800 & 3737241249 & -4725160676 \\
\hline
37833052667 & -31126018421 & -14556208209 & -18729114989 & -28337929097 \\
-15794737663 & 20410643195 & 29938810218 & -19770658960 & 32325595186 \\
-33397706287 & 30015479398 & 30131014582 & -21995406398 & 27809973328 \\
\hline
9360025257 & -7671966556 & -3602277252 & -4631511281 & -6969553956 \\
-3914752705 & 5068659349 & 7369381469 & -4911659962 & 8011002647 \\
-8286319777 & 7408694854 & 7427241597 & -5359197761 & 6868228288 \\
\hline
-794491835566 & 653641371951 & 305679836659 & 393310077243 & 595090601893 \\
331689187293 & -428624161845 & -628708919380 & 415184677459 & -678836947464 \\
701352321466 & -630321510956 & -632746558308 & 461893725042 & -584006153880 \\
\hline
1771941120788 & -1457815890550 & -681739794315 & -877191962578 & -1327228248029 \\
-739761653728 & 955959364476 & 1402227862677 & -925977235323 & 1513992730651 \\
-1564209634396 & 1405791028402 & 1411207019720 & -1030197188514 & 1302525192609 \\
\hline
-94600 & 208951 & 22318 & 55729 & 246209 \\
12652 & 27277 & -253968 & -34969 & -22976 \\
-20392 & -148178 & -197825 & 408721 & -136922 \\
\hline
-7747249494485 & 6373816330920 & 2980742346251 & 3835247462429 & 5802885174723 \\
3234362272426 & -4179583696992 & -6130705304876 & 4048531348870 & -6619463205782 \\
6839005255361 & -6146407224125 & -6170064964319 & 4504093360564 & -5694775864394 \\
\hline
586897205399 & -482852088933 & -225807785821 & -290541314978 & -439600904682 \\
-245020902701 & 316626651405 & 464435111645 & -306698701299 & 501461091480 \\
-518092596048 & 465624546756 & 467416782913 & -341210320554 & 431410969263
\end{bmatrix}_{15\times 15}
$$

Figure 8.7: The randomized, initial long basis for the 15-dimensional lattice in Example 8.33. The basis vectors are the rows of the matrix. Due to space constraints, each basis vector spans three lines.

$$\begin{bmatrix}
-1 & 1 & 0 & 2 & 0 & 0 & -1 & -1 & 1 & -1 & 0 & -1 & -1 & 0 & 1 \\
2 & 0 & 1 & 0 & -2 & 1 & 0 & 0 & -2 & -1 & 1 & 3 & 0 & 0 & 0 \\
0 & 0 & -1 & -2 & -1 & -2 & 1 & 1 & 0 & 0 & -1 & -1 & -1 & -1 & -2 \\
2 & -1 & 0 & 1 & -2 & -1 & 1 & 0 & -2 & 0 & -1 & 1 & 0 & -1 & 0 \\
0 & 0 & 1 & -1 & -1 & -1 & -1 & 2 & 0 & -1 & 1 & 1 & 0 & 2 & 1 \\
-1 & -1 & 1 & -1 & 1 & 0 & 0 & 0 & 1 & 0 & -1 & -1 & -1 & -2 & -2 \\
0 & 0 & -1 & 1 & -1 & -1 & 1 & 2 & 0 & 0 & 1 & -1 & 1 & -1 & 1 \\
-1 & -1 & -1 & 0 & 1 & -1 & 1 & 1 & 2 & 0 & -1 & 1 & -1 & 0 & -2 \\
1 & 2 & 1 & 0 & -1 & 2 & -1 & -1 & 1 & -1 & 1 & 0 & 2 & 0 & 1 \\
1 & 1 & 1 & 1 & -1 & 1 & -1 & 2 & -2 & 0 & 1 & -1 & -1 & 1 & 0 \\
3 & -1 & 1 & 0 & 0 & 0 & 1 & 0 & -1 & -1 & 0 & -1 & 0 & 1 & 0 \\
0 & 0 & -1 & -1 & 1 & 1 & 0 & 0 & 1 & -1 & 0 & 0 & -3 & -1 & 0 \\
-2 & -1 & 1 & -1 & -1 & 0 & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & -1 & 2 & -1 & -2 & -2 & 1 & 0 & 0 & 1 & 0 & -1 & -1 & -1 & 1 \\
0 & -2 & 1 & 0 & -1 & 1 & -2 & -1 & 1 & -1 & 1 & -1 & 1 & 2 & 0
\end{bmatrix}_{15\times 15}$$

Figure 8.8: The LLL-reduced basis of Example 8.33. The basis vectors are the rows of the matrix.

i	$\|b_i\|$	$\|b_i^*\|$	$\|\tilde{b}_i\|$	$\|\tilde{b}_i^*\|$
1	320436945197.4	320436945197.359360255541512	3.61	3.61
2	1144568732312.6	46732111.664529318780066	5.00	4.71
3	195578567373.0	12819.739043033123886	4.47	3.81
4	81148130911.1	397137.610643187052484	4.36	3.15
5	180908523242.6	14000.367117458200531	4.12	3.57
6	5340229.8	1676.733789170875920	4.12	2.88
7	1722666532.5	0.080881385402629	3.74	3.02
8	17775727743.2	0.276952900412341	4.24	3.35
9	104620743288.4	0.000045519492794	4.58	2.74
10	25839996203.4	2.172605704403767	4.36	3.09
11	2197024540111.8	0.574351076442320	4.00	2.97
12	4900011449631.2	0.019296310688108	4.00	2.44
13	656730.3	0.000000000176541	3.46	2.28
14	21423670206853.1	0.002453963889001	4.47	2.52
15	1622962205744.5	0.000000001243810	4.58	3.15

Table 8.3: The lengths of the original (randomized) basis vectors b_i, the LLL-reduced basis vectors $\tilde{b}_i$, and the corresponding Gram-Schmidt basis vectors b_i^* and $\tilde{b}_i^*$ (Example 8.33).

1.00														
-0.46	1.00													
-0.46	-0.35	1.00												
-0.38	0.48	0.37	1.00											
-0.15	0.27	0.01	-0.48	1.00										
-0.08	-0.43	0.36	-0.18	-0.43	1.00									
0.00	-0.09	0.16	0.41	0.35	-0.36	1.00								
-0.15	-0.40	0.34	-0.13	-0.05	0.33	-0.09	1.00							
0.31	0.40	-0.42	-0.45	-0.15	-0.28	0.26	-0.49	1.00						
0.08	0.38	0.03	-0.08	0.35	-0.09	0.24	-0.45	-0.46	1.00					
-0.31	0.23	-0.00	0.40	0.15	0.05	-0.19	-0.29	0.17	0.38	1.00				
0.23	-0.07	0.33	-0.49	-0.35	0.24	-0.06	0.23	-0.42	-0.06	0.36	1.00			
-0.23	-0.02	0.03	-0.02	0.36	0.29	0.24	-0.19	-0.41	-0.31	-0.24	-0.09	1.00		
-0.15	-0.04	0.47	0.42	0.38	0.46	0.09	-0.48	-0.05	-0.37	0.00	-0.01	0.38	1.00	
0.23	0.11	-0.40	-0.29	0.36	0.33	-0.25	-0.26	0.41	0.01	0.32	-0.38	0.05	-0.11	1.00

Table 8.4: Matrix of Gram-Schmidt coefficients μ_{ij} for the LLL-reduced basis in Example 8.33 (and where $\mu_{ii} = 1$).

the first eight Gram-Schmidt vectors b_i^* remain unchanged, as do all the Gram-Schmidt coefficients μ_{ij} for $1 \le j < i \le 8$. By updating only the quantities that actually change, the algorithm avoids unnecessary recomputation and achieves a noticeable improvement in running time.

8.4.2 The parameter δ

The constant $\frac{3}{4}$ appearing in the Lovász condition (47) can be replaced by an arbitrary constant δ satisfying $\frac{1}{4} < \delta \le 1$.

Definition 8.34 Let $\frac{1}{4} < \delta \le 1$. A lattice basis $B = [b_1, \dots, b_n]$ is called *δ-LLL-reduced* if it is size-reduced and satisfies

$$\|b_k^*\|^2 \ge \left(\delta - \mu_{k,k-1}^2\right) \|b_{k-1}^*\|^2 \quad \text{for all } 2 \le k \le n. \tag{69}$$

Choosing δ closer to 1 improves the approximation factor achieved by LLL. More precisely, we have the following generalization of Theorem 8.16 (Exercise 8.21(a)).

Theorem 8.35 Let $\frac{1}{4} < \delta \le 1$. If $B = [b_1, \dots, b_n]$ is δ-LLL-reduced, then

$$\|b_1\| \le \left(\frac{4}{4\delta - 1}\right)^{(n-1)/2} \lambda_1(L). \tag{70}$$

For $\frac{1}{4} < \delta < 1$ the LLL algorithm runs in polynomial time, although the number of iterations increases with δ. Setting $\delta = 1$ gives the approximation factor $(2/\sqrt{3})^{n-1}$. However, it remains an open problem to determine whether or not the LLL algorithm runs in polynomial time for this choice of δ. In practice, $\delta = 0.99$ is often used.

The first vector b_1 of a δ-LLL-reduced basis satisfies (Exercise 8.21(b))

$$\|b_1\| \le \left(\frac{4}{4\delta - 1}\right)^{(n-1)/4} \mathrm{vol}(L)^{1/n}. \tag{71}$$

Taking $\delta = 1 - \varepsilon'$ for small $\varepsilon' > 0$, (71) can be written as

$$\|b_1\| \leq \left(\frac{4}{3} + \varepsilon\right)^{(n-1)/4} \mathrm{vol}(L)^{1/n} \tag{72}$$

for some small $\varepsilon > 0$ depending on ε'. Thus, the LLL algorithm can be viewed as an algorithmic version of Hermite's upper bound (5) on $\lambda_1(L)$.

8.4.3 Using floating point arithmetic

Recall that the running time of LLL is $O(n^6 \log^3 X)$ bit operations, where n is the lattice dimension and X denotes the maximum length of the input basis vectors. During its execution, the algorithm performs arithmetic operations on integers whose bitlengths are on the order of $O(n \log X)$. In practical settings, the value of X can be quite large. For instance, in applications to the analysis of RSA implementations, X may be 2048 bits or more. For such large values of X, the $O(\log^3 X)$ dependence in the running time becomes a major computational bottleneck. As a result, the original LLL algorithm is typically practical only for lattices of relatively small dimensions.

When working with higher-dimensional lattices, it is therefore common to replace the exact integer arithmetic with floating point arithmetic in the computation of the Gram-Schmidt orthogonalization B^* and coefficients μ_{ij}. While this modification substantially improves efficiency, it also introduces the possibility of rounding errors which, in some cases, may prevent the algorithm from terminating or may result in a basis that is not LLL-reduced.

To address these difficulties, Nguyen and Stehlé proposed a floating-point version of LLL, which they called L^2. In their algorithm, the Gram-Schmidt computations are performed using floating-point arithmetic with a precision of approximately $1.58n$ bits. Notably, this precision is independent of the length X of the input basis vectors. The running time of L^2 has the important feature that it depends only on $\log^2 X$, rather than $\log^3 X$ as in the original LLL algorithm.

8.4.4 Deep insertion

The Lovász condition $\|b_k^*\|^2 \geq (\frac{3}{4} - \mu_{k,k-1}^2)\, \|b_{k-1}^*\|^2$, which can equivalently be written as $\|\Pi_{k-2}(b_k)\|^2 \geq \frac{3}{4}\|b_{k-1}^*\|^2$, guarantees that the Gram-Schmidt basis vectors do not shrink too rapidly. When this condition fails at some index k, that is, when $\|\Pi_{k-2}(b_k)\|^2 < \frac{3}{4}\|b_{k-1}^*\|^2$, the LLL algorithm swaps the adjacent basis vectors b_{k-1} and b_k. The reordered basis then becomes $[\ldots, b_{k-2}, b_k, b_{k-1}, b_{k+1}, \ldots]$.

In 1994, Schnorr and Euchner introduced a refinement of LLL, called *deep insertion*, which permits the swapping of non-adjacent basis vectors. Specifically, when the Lovász condition is violated at index k, then instead of simply swapping b_{k-1} and b_k, the vector b_k is inserted deeper into the basis: it is inserted between b_{i-1} and b_i, where i is the smallest index satisfying

$$\|\Pi_{i-1}(b_k)\|^2 < \frac{3}{4}\|b_i^*\|^2.$$

Observe that choosing $i = k - 1$ recovers the standard LLL swap.

Experiments have demonstrated that incorporating deep insertions often yields significantly shorter lattice bases. This improvement, however, comes at a computational cost: the running time typically increases, and unlike the standard LLL algorithm, LLL with deep insertions is not known to have a polynomial running-time bound.

8.5 Exercises

Exercise 8.1 Let $w, w_1, w_2 \in \mathbb{R}^n$, and suppose that $w = w_1 + w_2$ and $\langle w_1, w_2 \rangle = 0$. Prove that $\|w\|^2 = \|w_1\|^2 + \|w_2\|^2$.

Exercise 8.2 Let V be a subspace of $\mathbb{R}^n$.
(a) Prove that $\Pi_V(w) = w$ for all $w \in V$, and $\Pi_V(w) = 0$ for all $w \in V^\perp$.
(b) Prove that $\|w\| \geq \|\Pi_V(w)\|$ for all $w \in \mathbb{R}^n$.
(c) Prove that $\Pi_{V^\perp}(w + u) = \Pi_{V^\perp}(w)$ for all $w \in \mathbb{R}^n$ and $u \in V$.

Exercise 8.3 Let $B = [b_1, \ldots, b_n]$ be a basis of a vector space in $\mathbb{R}^n$, and let $B^* = [b_1^*, \ldots, b_n^*]$ be its Gram-Schmidt orthogonalization.
(a) Prove that $\|b_i^*\| \leq \|b_i\|$ for all $1 \leq i \leq n$.
(b) Prove that $\langle b_i, b_i^* \rangle = \langle b_i^*, b_i^* \rangle$ for all $1 \leq i \leq n$.

Exercise 8.4 For each of the following ordered bases B, verify that B^* is indeed the Gram-Schmidt orthogonalization.
(a) $B = [(0, 0, 2),\ (5, 2, 4),\ (-5, 1, -3)]$, $\quad B^* = [(0, 0, 2),\ (5, 2, 0),\ (-30/29, 75/29, 0)]$.
(b) $B = [(-3, 1, 5, 0),\ (2, -4, -1, -2),\ (5, 0, 4, 5),\ (-3, 1, -2, 5)]$,
$B^* = [(-3, 1, 5, 0),\ (5/7, -25/7, 8/7, -2),\ (11/2, -1/2, 17/5, 24/5),$
$\quad (-19673/8463, -20819/8463, -7640/8463, 8595/2821)]$.
(c) $B = [(2, -2, 0, -1, 1),\ (0, 0, 0, 0, -1),\ (2, -2, -2, -2, 1),\ (-1, 1, 2, 0, -2),$
$\quad (-1, 2, -1, 0, 2)]$,
$B^* = [(2, -2, 0, -1, 1),\ (1/5, -1/5, 0, -1/10, -9/10),\ (-2/9, 2/9, -2, -8/9, 0),$
$\quad (-3/11, 3/11, 6/11, -12/11, 0),\ (1/2, 1/2, 0, 0, 0)]$.

Exercise 8.5 Let $B = [b_1, \ldots, b_n]$ be a basis of a lattice $L \subseteq \mathbb{R}^n$, and let $B^* = [b_1^*, \ldots, b_n^*]$ be its Gram-Schmidt orthogonalization. For each i, $1 \leq i \leq n$, prove that

$$\lambda_i(L) \geq \min_{i \leq j \leq n} \|b_j^*\|.$$

Exercise 8.6 Consider the three-dimensional integer lattice L with basis $b_1 = (1, 2, 1)$, $b_2 = (2, 1, 1)$, $b_3 = (1, 1, 2)$. Compute the Gram-Schmidt orthogonalization $[b_1^*, b_2^*, b_3^*]$ and verify that $\mathrm{vol}(L) = \|b_1^*\| \cdot \|b_2^*\| \cdot \|b_3^*\|$.

Exercise 8.7 For each of the following two-dimensional lattices L with basis vectors u and v, determine $\lambda_1(L)$ and $\lambda_2(L)$.
(a) $u = (4, 7)$, $v = (6, 11)$.

(b) $u = (4, 7)$, $v = (5, 11)$.
(c) $u = (4321, 8765)$, $v = (1234, 5678)$.

Exercise 8.8 Let p be a prime that is congruent to 1 modulo 4, and let $i \in \mathbb{Z}_p$ with $i^2 \equiv -1 \pmod{p}$; such an i exists and can be efficiently found. We are interested in efficiently writing p as the sum of two squares.
Consider the lattice $L \subseteq \mathbb{Z}^2$ with basis matrix

$$B = \begin{bmatrix} 1 & 0 \\ i & p \end{bmatrix}.$$

(a) Prove that every $v \in L$ satisfies $\|v\|^2 \equiv 0 \pmod{p}$.
(b) Using Exercise 8.17(d), prove that $\lambda_1(L) = p$.
(c) Describe an efficient algorithm which, on input a prime $p \equiv 1 \pmod{4}$ and $i \in \mathbb{Z}_p$ with $i^2 \equiv -1 \pmod{p}$, finds integers a and b such that $p = a^2 + b^2$.
(d) Write each of the following primes p as the sum of two squares.
(i) $p = 12373$ ($i = 3243$); (ii) $p = 555557$ ($i = 202235$); (iii) $p = 44444453$ ($i = 11833757$).

Exercise 8.9 Explain why the index j in the inner loop in size reduction (Algorithm 8.18) is decremented from $i - 1$ to 1, instead of being incremented from 1 to $i - 1$.

Exercise 8.10 Write a program to find LLL-reduced bases of the lattices $L(B)$ where B is the following basis:
(a) $[(7, 8, -6, -6, -4),\ (2, 2, -1, -1, -1),\ (-2, 1, 0, -2, -4),\ (-5, -10, 7, 7, 6),$
$(0, -3, 0, -3, -3)]$.
(b) $[(-11, 11, -13, -6, 2),\ (36, -18, 54, 44, -11),\ (-1, -13, -6, -9, -1),$
$(-11, 5, -16, -13, 3),\ (-179, 91, -267, -223, 62)]$.
(c) $[(-2403, -1860, 2556, -698, -1275),\ (146, 109, -155, 37, 72),\ (-10, 23, 13, 40, 35),$
$(-482, -411, 510, -193, -305),\ (-143, -106, 152, -35, -70)]$.

Exercise 8.11 The E_8 lattice is a full-rank lattice in $\mathbb{Q}^8$ with basis matrix

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 & 1/2 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 \end{bmatrix}.$$

Construct a basis for E_8 in which every basis vector has length $\sqrt{2}$.

Exercise 8.12 Construct a basis for the Leech lattice Λ_{24} (Example 2.13) in which every basis vector has length 2. Conclude from this that $\lambda_i(\Lambda_{24}) = 2$ for all $1 \le i \le 24$.

Exercise 8.13 Prove that (48) and (49) are equivalent formulations of the Lovász condition.

Exercise 8.14 Each iteration of the LLL algorithm produces a basis of the input lattice L. Find examples of lattices for which two consecutive bases B and B' produced by the LLL algorithm satisfy $\max_{b \in B} \|b\| < \max_{b' \in B'} \|b'\|$. This shows that the basis vectors can increase in length during the LLL algorithm.

Exercise 8.15 Let $B = [b_1, \ldots, b_n]$ and $C = [c_1, \ldots, c_n]$ be two lattice bases produced in successive iterations of the LLL algorithm. Prove that $\min_i \|c_i^*\| \geq \min_i \|b_i^*\|$. (This shows that the minimum length of a Gram-Schmidt basis vector never decreases.)

Exercise 8.16 Let $B = [b_1, \ldots, b_n]$ be an LLL-reduced basis of a lattice $L \subseteq \mathbb{R}^n$, and let $B^* = [b_1^*, \ldots, b_n^*]$ be its Gram-Schmidt orthogonalization. Suppose that $\|b_1\| \leq \|b_i^*\|$ for all $1 \leq i \leq n$. Prove that $\lambda_1(L) = \|b_1\|$.

Exercise 8.17 Let $B = [b_1, \ldots, b_n]$ be an LLL-reduced basis of a lattice $L \subseteq \mathbb{R}^n$, and let $B^* = [b_1^*, \ldots, b_n^*]$ be its Gram-Schmidt orthogonalization.
(a) Prove that $\|b_j^*\|^2 \leq 2^{i-j} \|b_i^*\|^2$ for all $1 \leq j \leq i \leq n$.
(b) Prove that $\|b_i\| \leq 2^{(i-1)/2} \|b_i^*\|$ for all $1 \leq i \leq n$.
(c) Prove that $\|b_j\| \leq 2^{(i-1)/2} \|b_i^*\|$ for all $1 \leq j \leq i \leq n$.
(d) Prove that $\|b_1\| \leq 2^{(n-1)/4} \operatorname{vol}(L)^{1/n}$.

Exercise 8.18 Let $B = [b_1, \ldots, b_n]$ be an LLL-reduced basis of a lattice $L \subseteq \mathbb{R}^n$. Prove that $\|b_i\| \leq 2^{(n-1)/2} \lambda_i(L)$ for $1 \leq i \leq n$.

Exercise 8.19 Let $L \subseteq \mathbb{Z}^n$ be a full-rank integer lattice. Prove that the number of LLL-reduced bases of L is finite.

Exercise 8.20 Let $B = [b_1, \ldots, b_n]$ be a basis of a lattice $L \subseteq \mathbb{R}^n$, and let $B^* = [b_1^*, \ldots, b_n^*]$ be its Gram-Schmidt orthogonalization. For each $\ell \in [1, n]$, let L_ℓ be the sublattice of L with basis $B_\ell = [b_1, \ldots, b_\ell]$. The volume of L_ℓ is defined to be $\operatorname{vol}(L_\ell) = \sqrt{\det(B_\ell^T B_\ell)}$. Prove that $\operatorname{vol}(L_\ell) = \prod_{i=1}^{\ell} \|b_i^*\|$.

Exercise 8.21 Let $\frac{1}{4} < \delta \leq 1$, and let $B = [b_1, \ldots, b_n]$ be a δ-LLL reduced basis of a lattice $L \subseteq \mathbb{R}^n$ (see Definition 8.34).
(a) Prove that

$$\|b_1\| \leq \left(\frac{4}{4\delta - 1}\right)^{(n-1)/2} \lambda_1(L).$$

(b) Prove that

$$\|b_1\| \leq \left(\frac{4}{4\delta - 1}\right)^{(n-1)/4} \operatorname{vol}(L)^{1/n}.$$

Exercise 8.22 Explain why the proof that LLL terminates (Theorem 8.26) works for any $\frac{1}{4} < \delta < 1$, but not for $\delta = 1$.

Exercise 8.23 Design a polynomial-time algorithm that, on input a basis $B = [b_1, \ldots, b_n]$ of a lattice $L \subseteq \mathbb{Z}^n$, computes $\mathrm{vol}(L)$.

Exercise 8.24 Let $B = [b_1, \ldots, b_n]$ be a basis of a lattice $L \subseteq \mathbb{R}^n$. For each $j \in [1, n]$, define $B_j = [b_1, \ldots, b_{j-1}, 2b_j, b_{j+1}, \ldots, b_n]$ and $L_j = L(B_j)$.
(a) Let v be a shortest nonzero vector in L, and let $v = \sum_{i=1}^{n} c_i b_i$ where $c_i \in \mathbb{Z}$. Prove that at least one of the c_i's must be odd.
(b) Prove that L_j is a sublattice of L for each $j \in [1, n]$.
(c) Let $v = \sum_{i=1}^{n} c_i b_i \in L$ with c_j odd, and let $u = v + b_j$. Prove that $u \in L_j$ and $\|u - b_j\| = \|v\|$.
(d) Let $j \in [1, n]$, and let $u \in L_j$ and $v = u - b_j$. Prove that $v \in L \setminus \{0\}$ and $\|v\| = \|u - b_j\|$.
(e) Let $\gamma \geq 1$. Prove that $\mathrm{SVP}_\gamma \leq \mathrm{CVP}_\gamma$.

9 The BKZ lattice basis reduction algorithm

Contents

BKZ is a *blockwise* generalization of the LLL algorithm, introduced by Schnorr and Euchner in 1994. Rather than operating on the entire lattice at once, BKZ repeatedly solves SVP instances in certain low-dimensional lattices called *blocks*. These blocks have dimension β, called the *blocksize*, where β is typically much smaller than the full lattice dimension n. Each SVP instance is solved by an *SVP oracle*, usually implemented via either *enumeration* or *sieving*, both of which have exponential running times; enumeration is described in §9.4, and sieving in §9.5. There is a fundamental tradeoff in the choice of the blocksize: larger values of β typically produce higher-quality reduced bases, but at the cost of a substantial increase in running time. When $\beta = 2$, the BKZ algorithm coincides with the original LLL algorithm (i.e., with parameter $\delta = 1$).

Although precise cost estimates for BKZ remain difficult to obtain, extensive experimental evidence shows that BKZ significantly outperforms LLL in practice. In particular, BKZ has been used to solve the hardest SVP challenges, and BKZ running-time estimates play a central role in assessing the hardness of SVP and approx-SVP instances that arise in the security analysis of Kyber and Dilithium. These estimates, in turn, underpin the selection and justification of the standardized parameter sets for these schemes.

9.1 BKZ-reduced bases

Let $B = [b_1, \ldots, b_n]$ be a basis of a lattice $L \subseteq \mathbb{R}^n$, and let $B^* = [b_1^*, \ldots, b_n^*]$ denote its Gram-Schmidt orthogonalization. Recall that for $1 \le i \le n$, Π_i denotes the projection map onto $\langle b_1, \ldots, b_i\rangle^\perp = \langle b_1^*, \ldots, b_i^*\rangle^\perp$. Also, Π_0 is the identity map on $\mathbb{R}^n$.

For integers $1 \le i < j \le n$, define the *local projected block*

$$B[i,j] = [\Pi_{i-1}(b_i),\ \Pi_{i-1}(b_{i+1}),\ \ldots,\ \Pi_{i-1}(b_j)],$$

and let $L[i,j]$ be the lattice spanned by $B[i,j]$. Since the vectors in $B[i,j]$ are linearly independent, $L[i,j]$ is a lattice of dimension $j - i + 1$. Observe that the first vector of the local projected block $B[i,j]$ is precisely the Gram-Schmidt vector b_i^*.

Definition 9.1 Let $2 \le \beta \le n$. A basis $B = [b_1, \ldots, b_n]$ of a lattice L is said to be *β-BKZ-reduced* if B is size-reduced and

$$\|b_j^*\| = \lambda_1(L[j,k]) \text{ for each } 1 \le j \le n-1, \tag{73}$$

where $k = \min(j + \beta - 1, n)$. The number β is called the *blocksize*.

Remark 9.2 The notion of a β-BKZ-reduced basis extends naturally to lattices that do not have full rank.

A β-BKZ-reduced basis $[b_1, \ldots, b_n]$ is a size-reduced basis whose Gram-Schmidt vectors are shortest nonzero vectors in the lattices generated by the corresponding projected blocks:

$$\|b_i^*\| = \begin{cases} \lambda_1(L[i, i+\beta-1]), & \text{if } 1 \le i \le n-\beta+1, \\ \lambda_1(L[i, n]), & \text{if } n-\beta+2 \le i \le n-1. \end{cases}$$

Two extreme cases are worth noting. If $\beta = n$, then $\|b_1^*\| = \lambda_1(L)$, so the first basis vector b_1 solves SVP in L. At the other extreme, when $\beta = 2$, a β-BKZ-reduced basis coincides with a 1-LLL-reduced basis, as shown in the next result.

Theorem 9.3 Let $B = [b_1, \ldots, b_n]$ be a 2-BKZ-reduced basis of a lattice $L \subseteq \mathbb{R}^n$. Then B is 1-LLL-reduced (Definition 8.34).

Proof: For each $1 \le i \le n-1$, the projected block corresponding to $L[i, i+1]$ is $B[i, i+1] = [\Pi_{i-1}(b_i), \Pi_{i-1}(b_{i+1})]$. We have $\Pi_{i-1}(b_i) = b_i^*$ and $\Pi_{i-1}(b_{i+1}) = b_{i+1}^* + \mu_{i+1,i} b_i^*$. Since B is 2-BKZ-reduced, b_i^* is a shortest nonzero vector in $L[i, i+1]$. Therefore,

$$\|b_i^*\|^2 \le \|\Pi_{i-1}(b_{i+1})\|^2 = \|b_{i+1}^*\|^2 + \mu_{i+1,i}^2 \|b_i^*\|^2,$$

which implies

$$\|b_{i+1}^*\|^2 \ge (1 - \mu_{i+1,i}^2)\|b_i^*\|^2 \quad \text{for all } 1 \le i \le n-1.$$

This is precisely the Lovász condition with parameter $\delta = 1$. □

The next theorem provides an upper bound on the length of the first vector in a β-BKZ-reduced basis. Its proof relies on the following inequality for Hermite constants (Definition 2.14), whose proof is left as an exercise (Exercise 2.19).

Lemma 9.4 For all $1 \le i \le j$, we have $\gamma_i^i \le \gamma_j^j$.

Theorem 9.5 Let $2 \le \beta \le n$, and let $B = [b_1, \ldots, b_n]$ be a β-BKZ-reduced basis of a lattice L. Then

$$\|b_1\| \le \gamma_\beta^{(n-1)/(\beta-1)} \lambda_1(L). \tag{74}$$

Proof: For $1 \le i \le n-\beta+1$, since $\|b_i^*\| = \lambda_1(L[i, i+\beta-1])$, the Hermite bound[16] gives

$$\|b_i^*\| \le \sqrt{\gamma_\beta} \cdot \operatorname{vol}\left(L[i, i+\beta-1]\right)^{1/\beta}.$$

Using the volume formula (Remark 8.24), we obtain

$$\|b_i^*\| \le \sqrt{\gamma_\beta} \left(\prod_{j=i}^{i+\beta-1} \|\Pi_{i-1}(b_j)^*\| \right)^{1/\beta}.$$

[16] The Hermite bound (3) is also valid for lattices that do not have full rank: if $L \subseteq \mathbb{R}^n$ is a rank-β lattice, then $\lambda_1(L) \le \sqrt{\gamma_\beta}\, \operatorname{vol}(L)^{1/\beta}$.

Since $\Pi_{i-1}(b_j)^* = b_j^*$ for all $i \le j \le i+\beta-1$, we have

$$\|b_i^*\| \le \sqrt{\gamma_\beta} \left(\prod_{j=i}^{i+\beta-1} \|b_j^*\| \right)^{1/\beta}. \tag{75}$$

Raising both sides of (75) to the power β gives

$$\|b_i^*\|^\beta \le \gamma_\beta^{\beta/2} \prod_{j=i}^{i+\beta-1} \|b_j^*\|. \tag{76}$$

Multiplying (76) over $i = 1, \ldots, n-\beta+1$ yields the inequality

$$\prod_{i=1}^{n-\beta+1} \|b_i^*\|^\beta \le \gamma_\beta^{(n-\beta+1)\beta/2} \left(\prod_{i=1}^{\beta-1} \|b_i^*\|^i \right) \left(\prod_{i=\beta}^{n-\beta+1} \|b_i^*\|^\beta \right) \left(\prod_{i=n-\beta+2}^{n} \|b_i^*\|^{n-i+1} \right),$$

which after cancelling terms simplifies to

$$\prod_{i=1}^{\beta-1} \|b_i^*\|^{\beta-i} \le \gamma_\beta^{(n-\beta+1)\beta/2} \prod_{i=n-\beta+2}^{n} \|b_i^*\|^{n-i+1}. \tag{77}$$

Next, since $L[1,i] \subseteq L[1,\beta]$ for $2 \le i \le \beta$, the vector b_1^* being a shortest nonzero vector in $L[1,\beta]$, is also a shortest nonzero vector in $L[1,i]$. Applying the Hermite bound (3) and the volume formula yields

$$\|b_1^*\|^i \le \gamma_i^{i/2} \operatorname{vol}(L[1,i]) = \gamma_i^{i/2} \prod_{j=1}^{i} \|b_j^*\| \quad \text{for all } 2 \le i \le \beta-1. \tag{78}$$

Multiplying the $\beta-2$ inequalities (78) and $\|b_1^*\| \le \|b_1^*\|$, we obtain

$$\prod_{i=1}^{\beta-1} \|b_1^*\|^i \le \left(\prod_{i=2}^{\beta-1} \gamma_i^{i/2} \right) \left(\prod_{i=1}^{\beta-1} \|b_i^*\|^{\beta-i} \right).$$

Applying Lemma 9.4 and substituting (77) gives

$$\|b_1^*\|^{\beta(\beta-1)/2} \le \gamma_\beta^{\beta(\beta-2)/2} \gamma_\beta^{(n-\beta+1)\beta/2} \prod_{i=n-\beta+2}^{n} \|b_i^*\|^{n-i+1} = \gamma_\beta^{\beta(n-1)/2} \prod_{i=n-\beta+2}^{n} \|b_i^*\|^{n-i+1}. \tag{79}$$

Suppose that there is a shortest nonzero vector $v \in L$ such that $\Pi_{n-1}(v) \ne 0$. Then $\Pi_j(v) \ne 0$ for all $j \le n-1$. Since B is β-BKZ-reduced, we have $\|b_i^*\| = \lambda_1(L[i,n])$ for each $i \in [n-\beta+2, n-1]$. Hence

$$\|b_i^*\| \le \|\Pi_{i-1}(v)\| \le \|v\| = \lambda_1(L) \quad \text{for } n-\beta+2 \le i \le n-1. \tag{80}$$

Let $v = c_1 b_1 + \cdots + c_n b_n$ where $c_j \in \mathbb{Z}$, whence $\Pi_{n-1}(v) = c_n b_n^*$. Since $\Pi_{n-1}(v) \neq 0$, we have $c_n \neq 0$. Thus

$$\|b_n^*\| \leq \|\Pi_{n-1}(v)\| \leq \|v\| = \lambda_1(L). \tag{81}$$

Substituting (80) and (81) into (79) gives

$$\|b_1^*\|^{\beta(\beta-1)/2} \leq \gamma_\beta^{\beta(n-1)/2} \prod_{i=n-\beta+2}^{n} \lambda_1(L)^{n-i+1} = \gamma_\beta^{\beta(n-1)/2} \lambda_1(L)^{\beta(\beta-1)/2}.$$

Raising both sides to the power $2/\beta(\beta-1)$ and substituting $b_1 = b_1^*$ gives

$$\|b_1\| \leq \gamma_\beta^{(n-1)/(\beta-1)} \lambda_1(L).$$

If $\Pi_{n-1}(v) = 0$ for all shortest nonzero vectors $v \in L$, then let $k < n-1$ be the largest index for which one of the shortest nonzero vectors $v \in L$ satisfies $\Pi_k(v) \neq 0$. Let $B' = [b_1, \ldots, b_{k+1}]$ and $L' = L(B')$, and note that $\lambda_1(L') = \lambda_1(L)$. Then B' is a β-BKZ-reduced basis for L'. The result established above also holds for lattices which do not have full rank. Thus,

$$\|b_1\| \leq \gamma_\beta^{k/(\beta-1)} \lambda_1(L') \leq \gamma_\beta^{(n-1)/(\beta-1)} \lambda_1(L),$$

which completes the proof of the theorem. □

9.2 Algorithm description

No efficient algorithm for computing a β-BKZ-reduced basis of a lattice $L \subseteq \mathbb{R}^n$ is known. This is not surprising since, as noted above, the first vector of an n-BKZ-reduced basis solves SVP in L, which is NP-hard. Moreover, as shown in Theorem 9.3, a 2-BKZ-reduced basis is a 1-LLL-reduced basis, and no polynomial-time algorithm is known for finding the latter.

The BKZ algorithm (Algorithm 9.6) begins with an LLL-reduced basis $B = [b_1, \ldots, b_n]$ and repeatedly performs *tours* until a β-BKZ-reduced basis is obtained. A tour consists of a sequence of $n-1$ *block operations*. The first $n-\beta+1$ block operations act on the length-β blocks $B[i, i+\beta-1]$ for $1 \leq i \leq n-\beta+1$. The remaining $\beta-2$ block operations act on the smaller blocks $B[i, n]$ for $n-\beta+2 \leq i \leq n-1$, whose lengths decrease from $\beta-1$ down to 2.

Like the LLL swap operation, the BKZ block operation (Algorithm 9.7) is a local modification of a lattice basis, designed to prevent the Gram-Schmidt vectors from shrinking too rapidly. However, while the LLL swap acts on two adjacent basis vectors, the BKZ block operation works on a block of up to β consecutive basis vectors.

The BKZ block operation on $B[i, j]$ is the following. First, an SVP oracle (either enumeration or sieving) is used to find a shortest nonzero vector v' in the projected lattice $L[i, j]$. Writing $v' = \sum_{k=i}^{j} a_k \Pi_{i-1}(b_k)$ where $a_k \in \mathbb{Z}$, we define the corresponding

Algorithm 9.6: BKZ lattice basis reduction algorithm

Input: Basis $B = [b_1, \ldots, b_n]$ for a lattice L, blocksize $\beta \in [2, n]$
Output: β-BKZ-reduced basis B

$B \leftarrow \mathrm{LLL}(B)$ // compute an LLL-reduced basis of L
repeat
  Perform block operations on $B[1, \beta]$, $B[2, \beta + 1], \ldots, B[n - \beta + 1, n]$
  Perform block operations on $B[n - \beta + 2, n]$, $B[n - \beta + 3, n], \ldots, B[n - 1, n]$
until the basis B doesn't change
return (B)

lattice vector $v = \sum_{k=i}^{j} a_k b_k \in L$ so that $\Pi_{i-1}(v) = v'$. Next, the vector v is inserted between b_{i-1} and b_i producing the ordered set

$$B' = [b_1, \ldots, b_{i-1}, v, b_i, \ldots, b_n].$$

Of course, B' is not a lattice basis since it has one vector too many. Nonetheless, notice that the Gram-Schmidt orthogonalization of the first i vectors in B' is $[b_1^*, \ldots, b_{i-1}^*, v^*]$ where

$$\|v^*\| = \|\Pi_{i-1}(v)\| = \|v'\| \leq \|\Pi_{i-1}(b_i)\| = \|b_i^*\|,$$

and where b_i^* is the i-th Gram-Schmidt vector of basis B. Thus the i-th vector v^* in the Gram-Schmidt orthogonalization of B' is at most as long as the i-th vector b_i^* in the Gram-Schmidt orthogonalization of B (and the first $i - 1$ Gram-Schmidt vectors remain unchanged). Let $h = \min(j + 1, n)$, so the next BKZ block operation is on $B[i + 1, h]$. If indeed $\|v^*\| < \|b_i^*\|$, then linear independence of B' can be restored by applying a modified version of LLL (not described here) to the linearly dependent vectors $[b_1, \ldots, b_{i-1}, v, b_i, \ldots, b_h]$ yielding the linearly independent (and LLL-reduced) vectors $[\hat{b}_1, \ldots, \hat{b}_h]$. On the other hand, if $\|v^*\| = \|b_i^*\|$, then the basis vectors $[b_1, \ldots, b_h]$ are LLL-reduced.

9.3 Complexity analysis

The BKZ algorithm is guaranteed to terminate after at most $O(\beta^n)$ tours. This fact implies that, for any lattice, a β-BKZ-reduced basis always exists. However, it remains an open question whether the number of tours can be bounded by a polynomial in the lattice dimension n. In practical implementations, BKZ is terminated after successive tours no longer produce a significant improvement in the quality of the basis. The complexity analysis presented here assumes such early-abort strategies are employed.

The blocksize β is the central parameter controlling both the running time and quality of the output basis. As β increases, BKZ tends to produce shorter basis vectors. This improvement, however, comes at a computational cost since the SVP solvers employed (enumeration or sieving) require exponential time in β. Roughly speaking, the running

Algorithm 9.7: BKZ block operation

Input: Basis $B = [b_1, \ldots, b_n]$ for a lattice L, indices $1 \le i < j \le n$
Output: Basis B after performing the BKZ block operation on $B[i, j]$

Find a shortest nonzero vector v' in $L[i, j]$ using enumeration or sieving
Compute $v = \sum_{k=i}^{j} a_k b_k$ where $v' = \sum_{k=i}^{j} a_k \Pi_{i-1}(b_k)$
Set $h \leftarrow \min(j + 1, n)$
if $\|v^*\| < \|b_i^*\|$ **then**
    $\hat{B} \leftarrow$ modified-LLL$([b_1, \ldots, b_{i-1}, v, b_i, \ldots, b_h])$
else
    $\hat{B} \leftarrow$ LLL$([b_1, \ldots, b_h])$
$B \leftarrow \hat{B} \cup [b_{h+1}, \ldots, b_n]$
return (B)

time of BKZ can be written as $O(2^{c\beta}\text{poly}(n))$ for some constant $c > 0$, where the $2^{c\beta}$ term reflects the exponential cost of solving SVP in each block, and $\text{poly}(n)$ reflects the total number of SVP oracle calls and overhead per call.

In terms of quality of the output basis B, BKZ achieves a Hermite factor of roughly $\beta^{O(n/\beta)}$, where the *Hermite factor* of B is defined as

$$\delta(B) = \|b_1\| / \text{vol}(L)^{1/n}, \tag{82}$$

b_1 being the first basis vector. At one extreme, when the blocksize is $\beta = 2$, BKZ essentially coincides with the LLL algorithm which runs in polynomial time and achieves a Hermite factor approximately $(\frac{4}{3})^{n/4}$ (equation 71 with $\delta \approx 1$). At the opposite extreme, when the blocksize is $\beta = n$, BKZ reduces to an SVP solver, achieving a Hermite factor of roughly $\sqrt{n}$.

Between these two extremes lies the general case of BKZ. When the blocksize is $\beta = n^\alpha$ for some $\alpha \in (0, 1)$, then BKZ exhibits subexponential running time $2^{\tilde{O}(n^\alpha)}$ and achieves a Hermite factor $2^{\tilde{O}(n^{1-\alpha})}$. (Here, the tilde indicates that polylogarithm factors in n are suppressed.) For example, if $\alpha = \frac{1}{2}$ so $\beta = \sqrt{n}$, both the running time and the Hermite factor of BKZ are approximately $2^{\sqrt{n}}$. Table 9.1 compares the performance of the LLL, BKZ, enumeration, and sieving algorithms according to the length of the shortest vector returned by each method.

9.4 Enumeration

Enumeration is a method for solving SVP, i.e., finding a shortest nonzero vector in a lattice $L(B) \subseteq \mathbb{Z}^n$. Although its running time is fully exponential in the lattice dimension n, enumeration has the advantage of requiring very little memory.

The algorithm begins with an upper bound D on the first successive minimum $\lambda_1(L)$. It then systematically enumerates integer vectors $x \in \mathbb{Z}^n$ satisfying $0 < \|x\| \le D$, and

Algorithm	Approximate running time	Performance	Notes
LLL	$\text{poly}(n)$	Hermite factor $\approx 2^{n/4} \approx 1.19^n$ with LLL parameter $\delta = \frac{3}{4}$	proven
	$\text{poly}(n)$	Hermite factor $\approx (4/3)^{n/4} \approx 1.075^n$ with LLL parameter $\delta \approx 1$	proven
	$\text{poly}(n)$	Hermite factor $\approx 1.02^n$ with LLL parameter $\delta \approx 1$	experimental
BKZ-β	$\text{poly}(n) \cdot 2^{0.292\beta}$	Hermite factor $\approx \beta^{n/2\beta}$ for $\beta \geq 50$	see §9.6
Enumeration	$2^{n \log n}$	Exact: finds a vector of length $\lambda_1(L)$	see §9.4
Sieving	$2^{0.292n}$	Exact: finds a vector of length $\lambda_1(L)$	see §9.5

Table 9.1: Comparison of the lengths of the shortest lattice vectors produced by LLL, BKZ, enumeration, and sieving.

outputs a lattice vector of shortest length found during the search. By first applying the LLL algorithm, we may assume without loss of generality that the lattice basis $B = [b_1, \ldots, b_n]$ is LLL-reduced. Minkowski's Theorem provides a natural upper bound for $\lambda_1(L)$:

$$D = \min(\|b_1\|, \sqrt{n}\ \text{vol}(L)^{1/n}). \tag{83}$$

Naive enumeration. A naive approach to finding a shortest nonzero lattice vector involves enumerating all candidate vectors $v = (v_1, \ldots, v_n) \in [-D, D]^n$ one coordinate at a time and testing each for membership in L. This can be done systematically using a *depth-first search tree* of height n. The root, at level 0, is denoted by R. For each $1 \leq i \leq n$, nodes at level i are labelled by the admissible integer values of v_i. Consequently, each internal node has $2D + 1$ children, and the leaves correspond precisely to the vectors in $[-D, D]^n$. A node z at level i represents the partial vector $(v_1, \ldots, v_i)$, whose coordinates are the labels of nodes along the unique path from R to z. During the depth-first traversal, the search tree can be *pruned* by maintaining the length of the current partial vector. If this partial length exceeds either the bound D or the length of the shortest nonzero lattice vector found so far, the corresponding branch is discarded and the algorithm backtracks, since no extension of the current partial vector can yield a shorter lattice vector. In the worst case, the running time of this naive approach is $(2D + 1)^n$.

Basic enumeration. For integers $x_1, \ldots, x_n$, let $x = x_1 b_1 + \cdots + x_n b_n$ denote the lattice vector determined by using the x_i's as the coefficients of a linear combination of basis vectors. The basic enumeration strategy consists of enumerating all integer tuples $(x_1, \ldots, x_n)$ such that $\|x\| \leq D$, and selecting a nonzero x of shortest length. Bounds on the coefficients x_i are derived by expressing x in terms of the Gram-Schmidt orthogonalization $B^* = [b_1^*, \ldots, b_n^*]$ of the basis B.

Since $b_1 = b_1^*$ and $b_i = b_i^* + \sum_{j=1}^{i-1} \mu_{ij} b_j^*$ for $2 \le i \le n$, we can write

$$x = \sum_{i=1}^{n} x_i b_i = \sum_{i=1}^{n} x_i \left(b_i^* + \sum_{j=1}^{i-1} \mu_{ij} b_j^* \right) = \sum_{j=1}^{n} \left(x_j + \sum_{i=j+1}^{n} \mu_{ij} x_i \right) b_j^*.$$

Recall that for $0 \le i \le n-1$, Π_i denotes the projection onto the orthogonal complement of $\langle b_1^*, \ldots, b_i^* \rangle$. Since the vectors $b_1^*, \ldots, b_n^*$ form an orthogonal basis, we have $\Pi_k(b_j^*) = 0$ for all $0 \le j \le k \le n-1$, and $\Pi_k(b_j^*) = b_j^*$ for all $0 \le k < j \le n$. Therefore, for all $0 \le k \le n-1$ we have

$$\Pi_k(x) = \sum_{j=k+1}^{n} \left(x_j + \sum_{i=j+1}^{n} \mu_{ij} x_i \right) b_j^* \tag{84}$$

and thus

$$\|\Pi_k(x)\|^2 = \sum_{j=k+1}^{n} \left(x_j + \sum_{i=j+1}^{n} \mu_{ij} x_i \right)^2 \|b_j^*\|^2. \tag{85}$$

For any lattice vector x of length at most D, we have the chain of inequalities

$$\|\Pi_{n-1}(x)\| \le \|\Pi_{n-2}(x)\| \le \cdots \le \|\Pi_2(x)\| \le \|\Pi_1(x)\| \le \|\Pi_0(x)\| = \|x\| \le D.$$

Applying (85) to the inequality $\|\Pi_{n-1}(x)\| \le D$, we obtain $x_n^2 \|b_n^*\|^2 \le D^2$, which yields the bound

$$|x_n| \le D / \|b_n^*\|. \tag{86}$$

From (84), we note that $\Pi_{k-1}(x)$ depends only on the coefficients $x_k, x_{k+1}, x_{k+2}, \ldots, x_n$. Proceeding recursively, if integers $x_n, x_{n-1}, \ldots, x_{k+1}$ are fixed such that

$$\|\Pi_{n-1}(x)\| \le \|\Pi_{n-2}(x)\| \le \cdots \le \|\Pi_k(x)\| \le D,$$

the admissible values for x_k are determined by the condition $\|\Pi_{k-1}(x)\| \le D$. Using (85), this condition can be written as

$$\left(x_k + \sum_{i=k+1}^{n} \mu_{ik} x_i \right)^2 \|b_k^*\|^2 \le D^2 - \sum_{j=k+1}^{n} \left(x_j + \sum_{i=j+1}^{n} \mu_{ij} x_i \right)^2 \|b_j^*\|^2.$$

This gives the bound

$$|x_k + \Delta| \le \frac{\sqrt{D^2 - \Omega}}{\|b_k^*\|}, \tag{87}$$

where

$$\Delta = \Delta(x_{k+1}, \ldots, x_n) = \sum_{i=k+1}^{n} \mu_{ik} x_i$$

and

$$\Omega = \Omega(x_{k+1}, \ldots, x_n) = \sum_{j=k+1}^{n} \left(x_j + \sum_{i=j+1}^{n} \mu_{ij} x_i \right)^2 \|b_j^*\|^2.$$

One proceeds to find a shortest nonzero lattice vector by systematically enumerating all admissible integer tuples $(x_1, \ldots, x_n)$ where the x_i are restricted by (86) and (87). The search is structured as a depth-first search on a tree of height n. The root of the tree is denoted by R. For each $1 \leq i \leq n$, nodes at level i are labelled by the admissible integer values of x_{n-i+1}.

Analysis. The complexity of the basic enumeration procedure is bounded by the number of nodes in the search tree. From (86) and (87), we see that the number of admissible values for x_k is at most

$$2 \left\lfloor \frac{D}{\|b_k^*\|} \right\rfloor + 1 \leq \frac{4D}{\|b_k^*\|},$$

and hence the total number of admissible values for $(x_1, \ldots, x_n)$ is at most

$$\prod_{k=1}^{n} \frac{4D}{\|b_k^*\|} = \frac{4^n D^n}{\prod_{k=1}^{n} \|b_k^*\|} \leq \frac{4^n (\sqrt{n}\ \mathrm{vol}(L)^{1/n})^n}{\mathrm{vol}(L)} = 4^n n^{n/2} = 2^{\frac{1}{2} n \log n + o(n \log n)}.$$

Thus, the running time of the basic enumeration procedure is

$$O(2^{\frac{1}{2} n \log n + o(n \log n)}).$$

9.5 Sieving

Sieving is a probabilistic method for solving SVP in an integer lattice $L \subseteq \mathbb{Z}^n$. Whereas sieving algorithms are conceptually quite simple, their analysis can be very complicated. In this section, we will outline the Nguyen-Vidick sieving algorithm and give a simplified heuristic analysis.

Overview. The algorithm begins with a set S_0 of $N_0 \approx n^3 (4/3)^{n/2}$ randomly sampled lattice vectors each of length approximately R_0, for some $R_0 \leq 2^{O(n)} \lambda_1(L)$. Let $\gamma < 1$ be a *shrinking factor*; for the sake of concreteness, we will take $\gamma = 1 - \frac{1}{n}$. For $k = 1, 2, 3, \ldots$, the algorithm iteratively applies a sieving step to S_{k-1} to produce a set S_k of lattice vectors each of which has length approximately $R_k = \gamma^k R_0$; see Figure 9.1. Notably, the sets do not significantly decrease in size, so S_k has size $N_k \approx N_0$. However, once $R_k \approx \lambda_1(L)$, the sets S_k rapidly decrease in size and inevitably collapse, namely $S_\ell = \emptyset$. In that event, the shortest vector in $S_{\ell-1}$ is expected to have length close to $\lambda_1(L)$.

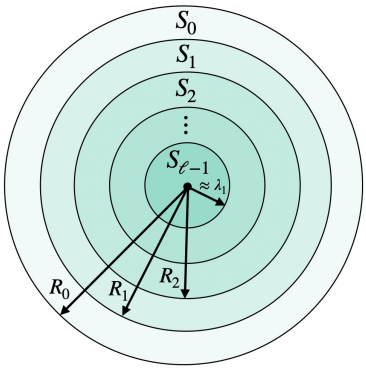


Figure 9.1: The sieving method.

Preliminaries. We collect here some notation and geometric facts about balls and spheres that will be used in the algorithm and its analysis.

Definition 9.8 The *ball* $B_n(v, r)$ of radius r centered at $v \in \mathbb{R}^n$ is defined by

$$B_n(v, r) = \{x \in \mathbb{R}^n \ : \ \|x - v\| \leq r\}.$$

If $v = 0$, we will denote $B_n(v, r)$ by $B_n(r)$. The *unit sphere* is

$$S = \{x \in \mathbb{R}^n \ : \ \|x\| = 1\}.$$

Theorem 9.9 The volume of the ball $B_n(R)$ is

$$\mathrm{vol}(B_n(R)) = \frac{\pi^{n/2}}{\Gamma(\frac{n}{2} + 1)} R^n,$$

where Γ is the gamma function.

We next prove an upper bound on the number of lattice points that can be contained in a ball centered at the origin.

Theorem 9.10 Let $L \subseteq \mathbb{Z}^n$ be a lattice with $\lambda_1 = \lambda_1(L)$, and let $R > 0$. The number of lattice points v in $B_n(R)$ satisfies

$$|L \cap B_n(R)| \leq \left(1 + \frac{2R}{\lambda_1}\right)^n.$$

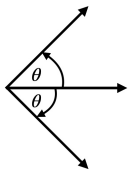
Proof: By definition of $\lambda_1(L)$, the balls $B_n(v, \frac{\lambda_1}{2})$ centered at lattice points $v \in B_n(R)$ do not overlap, and are all contained in $B_n(R + \frac{\lambda_1}{2})$. Thus,

$$|L \cap B_n(R)| \leq \frac{\mathrm{vol}(B_n(R + \frac{\lambda_1}{2}))}{\mathrm{vol}(B_n(\frac{\lambda_1}{2}))} = \frac{\left(R + \frac{\lambda_1}{2}\right)^n}{\left(\frac{\lambda_1}{2}\right)^n} = \left(1 + \frac{2R}{\lambda_1}\right)^n.$$

□

The *angle* $\theta \in [0, \pi]$ between nonzero vectors $u, v \in \mathbb{R}^n$ is defined to be

$$\cos^{-1}\left(\frac{\langle u, v\rangle}{\|u\|\,\|v\|}\right).$$



The *law of cosines* states that

$$\|u - v\|^2 = \|u\|^2 + \|v\|^2 - 2\|u\|\,\|v\|\,\cos\theta. \tag{88}$$

If u and v are unit vectors, then

$$\|u - v\| = \sqrt{2(1 - \cos\theta)}.$$

The following gives the probability density function of the angle between two random unit vectors.

Theorem 9.11 Fix $c \in S$. Let $\theta \in [0, \pi]$ be the angle between c and v, where v is selected uniformly at random from S. Then the density function of θ is

$$f_n(\theta) = \frac{\Gamma(\frac{n}{2})}{\sqrt{\pi}\,\Gamma(\frac{n-1}{2})}\,\sin^{n-2}\theta.$$

Remark 9.12 Observe that if $n = 2$, then

$$f_n(\theta) = \frac{\Gamma(1)}{\sqrt{\pi}\,\Gamma(\frac{1}{2})} = \frac{1}{\sqrt{\pi}\,\sqrt{\pi}} = \frac{1}{\pi},$$

so θ is uniformly distributed over $[0, \pi]$, as expected. On the other hand, for large n, $\Gamma(\frac{n}{2})/\Gamma(\frac{n-1}{2}) \approx \sqrt{\frac{n}{2}}$, so

$$f_n(\theta) \approx \sqrt{\frac{n}{2\pi}}\,\sin^{n-2}\theta.$$

Since $\sin^{n-2}\theta$ is maximized at $\theta = \pi/2$, and the peak sharpens exponentially with n, $f_n(\theta)$ is sharply concentrated around $\theta = \pi/2$. Consequently, two random unit vectors are nearly orthogonal with high probability, illustrating the counterintuitive geometry of high-dimensional spaces.

Lemma 9.14 estimates the probability that two random unit vectors are close to each other. Its proof will use the following technical result.

Lemma 9.13 For large n,

$$\int_0^{\pi/3} \sin^n\theta\,d\theta \approx \frac{\sqrt{3}}{n}\left(\frac{\sqrt{3}}{2}\right)^n.$$

Proof: Substituting $\theta = \pi/3 - t$, we have

$$\int_0^{\pi/3} \sin^n \theta \, d\theta = \int_0^{\pi/3} \sin^n(\pi/3 - t) \, dt.$$

Now,

$$\begin{aligned}\sin(\pi/3 - t) &= \sin(\pi/3)\cos(t) - \cos(\pi/3)\sin(t) \\ &= \frac{\sqrt{3}}{2}\left(1 - \frac{t^2}{2!} + O(t^4)\right) - \frac{1}{2}\left(t - \frac{t^3}{3!} + O(t^5)\right) \\ &= \frac{\sqrt{3}}{2}\left(1 - \frac{t}{\sqrt{3}}\right) + O(t^2).\end{aligned}$$

Therefore

$$\sin^n(\pi/3 - t) \approx \left(\frac{\sqrt{3}}{2}\right)^n e^{-nt/\sqrt{3}}.$$

Finally, by Laplace's method we get

$$\int_0^{\pi/3} \sin^n(\pi/3 - t) \, dt \approx \left(\frac{\sqrt{3}}{2}\right)^n \int_0^{\infty} e^{-nt/\sqrt{3}} \, dt \sim \frac{\sqrt{3}}{n}\left(\frac{\sqrt{3}}{2}\right)^n.$$

□

Lemma 9.14 Suppose that n is large. Fix $c \in S$. The probability that a randomly selected unit vector $v \in S$ lies in $B_n(c)$ is

$$\Pr(\|v - c\| \le 1) \approx \frac{1}{\sqrt{n}}\left(\frac{3}{4}\right)^{n/2}.$$

Proof: By the law of cosines (88), $\|v - c\|^2 = 2(1 - \cos\theta)$, where θ is the angle between v and c. Thus, $\|v - c\| \le 1$ if and only if $\cos\theta \ge 1/2$, i.e. $\theta \in [0, \pi/3]$. By Theorem 9.11,

$$\Pr(\theta \in [0, \pi/3]) = \frac{\int_0^{\pi/3} f_n(\theta)\, d\theta}{\int_0^{\pi} f_n(\theta)\, d\theta} \approx \frac{\int_0^{\pi/3} \sqrt{\frac{n}{2\pi}} \sin^{n-2}\theta \, d\theta}{1} = \sqrt{\frac{n}{2\pi}} \int_0^{\pi/3} \sin^{n-2}\theta \, d\theta.$$

Using Lemma 9.13, we have

$$\Pr(\theta \in [0, \pi/3]) \approx \sqrt{\frac{n}{2\pi}}\left(\frac{\sqrt{3}}{n-2}\right)\left(\frac{\sqrt{3}}{2}\right)^{n-2} \approx \frac{1}{\sqrt{n}}\left(\frac{3}{4}\right)^{n/2}.$$

□

The Nguyen-Vidick sieving algorithm. A simplified version of the sieving method is given in Algorithm 9.15. The algorithm begins by applying LLL to the input basis B to obtain an LLL-reduced basis. This ensures that the basis vectors are relatively short, which in turn allows us to sample[17] lattice vectors of length less than $2^n\lambda_1(L)$. The initial radius R_0 is then computed as the maximum length of the vectors in S_0.

The main loop iteratively applies a sieving step to the current set S_k and produces a new set S_{k+1} of lattice vectors of length approximately γR_k. Each sieving step maintains a set C of "centers", which are vectors in S_k possessing the property that the distance between any two centers is greater than γR_k. The sieving step processes every vector $v \in S_k$. If v has length at most γR_k, it is inserted in S_{k+1}. Otherwise, either $v - c$ is inserted in S_{k+1} if $v \in B_n(c, \gamma R_k)$ for some $c \in C$, or else v is identified as a center. After processing all $v \in S_k$, the radius R_{k+1} is computed as the maximum length of the vectors in S_{k+1}.

The sieving continues until the set S_k is empty, meaning no further reductions are possible. As we will discuss in the analysis below, this occurs once $R_k \approx \lambda_1(L)$. The algorithm then returns the shortest vector in the final nonempty set, which is expected to have length close to $\lambda_1(L)$.

Analysis. We now give a heuristic analysis of the sieving algorithm. The analysis rests on the following key heuristic assumption.

Heuristic assumption 1 For each $k \geq 0$, the unit vectors $\{v/\|v\| : v \in S_k\}$ are independently and uniformly distributed on the unit sphere S.

While this assumption is not provably true, it is supported by experiment and is standard in the lattice sieving literature. It allows us to treat the normalized vectors in each set S_k as random points on the unit sphere.

We first justify that the size N_0 of the initial set S_0 is sufficient to sustain the sieve, meaning that $N_k \approx N_0$, where $N_k = |S_k|$. We'll assume that every vector in S_k has size approximately $R_k \approx \gamma^k R_0$. Since the sieve is scale-invariant, we may normalize all vectors in S_k to lie on the unit sphere S without loss of generality. Fix a vector $c \in S_k$. Since $\gamma \approx 1$, we can use Lemma 9.14 to conclude that the probability a vector $v \in S$ satisfies $\|v - c\| \leq \gamma$ is approximately $\frac{1}{\sqrt{n}}(3/4)^{n/2}$. In other words, each vector $c \in S_k$ "covers" a $\frac{1}{\sqrt{n}}(3/4)^{n/2}$ fraction of the sphere, in the sense that any unit vector v falling within this fraction will yield a shorter vector $v - c$. For the sieve to be sustainable, every vector on the sphere must be covered by at least one vector in S_k. Assuming the covered regions have negligible overlap, a set of size N covers a $\frac{N}{\sqrt{n}}(3/4)^{n/2}$ fraction of the sphere. Full coverage therefore requires

$$N \cdot \frac{1}{\sqrt{n}} \left(\frac{3}{4}\right)^{n/2} \geq 1,$$

[17] The Nguyen-Vidick algorithm uses Klein's algorithm to select the initial lattice vectors according to a discrete Gaussian distribution. The vectors selected have length $2^{O(n)}\lambda_1(L)$, which we have simplified to $2^n\lambda_1(L)$.

Algorithm 9.15: Nguyen-Vidick sieving method for solving SVP

Input: A basis for a lattice $L \subseteq \mathbb{Z}^n$.
Output: A vector in $L = L(B)$ of length $\approx \lambda_1(L)$.

$[b_1, \ldots, b_n] \leftarrow \text{LLL}(B)$. // compute an LLL-reduced basis of L
Randomly select small integer linear combinations of $b_1, \ldots, b_n$ to obtain a set S_0 of size $N_0 \approx n^3(4/3)^{n/2}$. // Klein's algorithm is used in practice
Compute $R_0 = \max\{\|v\| : v \in S_0\}$.
$\gamma \leftarrow 1 - \frac{1}{n}$. // γ is the shrinking factor
$k \leftarrow 0$.
while $S_k \neq \emptyset$ **do**
    $C \leftarrow \emptyset$. // C is the set of centers
    $S_{k+1} \leftarrow \emptyset$. // S_{k+1} is the new set of lattice vectors
    for each vector $v \in S_k$ **do**
        if $\|v\| \leq \gamma R_k$ **then**
            $S_{k+1} \leftarrow S_{k+1} \cup \{v\}$.
        else
            if there is a $c \in C$ such that $\|v - c\| \leq \gamma R_k$ **then**
                $S_{k+1} \leftarrow S_{k+1} \cup \{v - c\}$.
            else
                $C \leftarrow C \cup \{v\}$.
    Compute $R_{k+1} = \max\{\|v\| : v \in S_{k+1}\}$.
    $k \leftarrow k + 1$.
Select $v \in S_{k-1}$ of minimum length.
return (v).

which gives

$$N \geq \sqrt{n} \left(\frac{4}{3}\right)^{n/2}.$$

The factor n^3 in the initial set size $N_0 = n^3(4/3)^{n/2}$ is a polynomial correction that ensures the coverage condition holds with overwhelming probability.

The set S_{k+1} will be smaller than S_k for two reasons: (i) vectors v in the center $C \subseteq S_k$ are discarded; and (ii) collisions may arise wherein an eligible vector $v - c$ is already in S_{k+1} and thus doesn't contribute to expanding S_{k+1}. One expects that the number of centers selected during a sieving step to be on the order of $\sqrt{n}(4/3)^{n/2}$, corresponding to the size of a random covering. Thus, if the number of collisions is negligible, then one expects that

$$N_k = |S_k| \approx |S_0| - k\sqrt{n}\left(\frac{4}{3}\right)^{n/2} \approx (n^3 - k\sqrt{n})\left(\frac{4}{3}\right)^{n/2} \approx n^3\left(\frac{4}{3}\right)^{n/2} \approx |S_0| \text{ for } k \leq n^2.$$

By the birthday paradox, collisions are only expected when $|S_k|$ exceeds the square root of the number of available lattice vectors, namely the lattice vectors in $B_n(R_{k+1})$. By Theorem 9.10, this occurs when

$$N_k \geq \left(1 + \frac{2R_{k+1}}{\lambda_1(L)}\right)^{n/2} \approx \left(\frac{2R_{k+1}}{\lambda_1(L)}\right)^{n/2},$$

which (roughly) yields the condition $R_{k+1} \approx \lambda_1(L)$.

Suppose that a (scaled) set S_k contains a covering set of the unit sphere S. Then every other vector v in S_k will be within distance γ of a vector c in the cover. This vector $v - c$ will be added to S_{k+1} except if it collides with a vector already in S_{k+1}. The algorithm must terminate by the time $R_k = \lambda_1(L)$. Since $R_k \approx \gamma^k R_0 \approx \gamma^k 2^n \lambda_1(L)$, the maximum number of sieving steps is determined by solving

$$\gamma^k 2^n \lambda_1(L) \approx \lambda_1(L),$$

yielding

$$k \approx -\frac{n}{\log \gamma} = -\frac{n}{\log(1 - \frac{1}{n})} \approx \frac{-n}{-1/n} = n^2.$$

Thus, the algorithm is expected to terminate after at most n^2 sieving steps.

Each sieving step processes all N_k vectors in the current set S_k. Using a naive search through the centers C, each vector may need to be compared against all the centers, resulting in a cost of $O(N_k^2)$ per sieving step. Since there are at most n^2 sieving steps, the total running time is at most

$$n^2 \cdot N_0^2 = n^8\left(\frac{4}{3}\right)^n \approx 2^{0.415n + o(n)},$$

while the space complexity is

$$N_0 = n^3\left(\frac{4}{3}\right)^{n/2} \approx 2^{0.2075n + o(n)}.$$

In summary, the sieving algorithm solves SVP in heuristic time $2^{0.415n+o(n)}$, which is asymptotically faster than the best enumeration algorithms. A major drawback is the exponential space requirements.

9.6 Estimating the security level of Kyber

Recall the parameters in Kyber key generation (Algorithm 6.10): $A \in_R R_q^{k\times k}$, $s \in_{CBD} S_{\eta_1}^k$, $e \in_{CBD} S_{\eta_1}^k$, and $t = As + e \pmod q$, where $\in_{CBD}$ denotes sampling from a centered binomial distribution. We are interested in assessing the difficulty of recovering the Kyber decryption key (s, e) from an encryption key (A, t); this is an instance of the short-secret MLWE problem (Definition 5.17).

From MLWE to LWE. Since no attacks on MLWE are known to be substantially faster than corresponding attacks on LWE, the complexity is analyzed by solving the standard short-secret LWE problem (Definition 4.6) with matching dimensions. Accordingly, let $A \in_R \mathbb{Z}_q^{m\times n}$, $s \in_{CBD} [-\eta_1, \eta_1]^n$, $e \in_{CBD} [-\eta_1, \eta_1]^m$, and $t = As + e \pmod q \in \mathbb{Z}_q^m$. In Kyber key generation, these parameters are $m = 256k$ and $n = 256k$.

Primal attack. The primal attack constructs a lattice where the ss-LWE solution (s, e) corresponds to a nonzero vector v of unusually small length. This transforms the problem to an instance of the Unique Shortest Vector Problem (uSVP). Applying the BKZ lattice reduction algorithm produces a basis containing sufficiently short vectors, from which the target vector v, and hence the secret key (s, e), can be extracted.

Definition 9.16 The lattice associated with the ss-LWE instance (A, t) is defined as:

$$L = \{x \in \mathbb{Z}^{m+n+1} : [I_m|A| - t]x = 0 \pmod q\}. \tag{89}$$

Theorem 9.17 *L is a full-rank lattice of dimension $d = m + n + 1$ and volume q^m.*

Proof: L is a discrete additive subgroup, and therefore a lattice. We will show that

$$B = \begin{bmatrix} qI_m & -A & t \\ 0 & I_n & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{90}$$

is a basis matrix for L, from which it follows that L has full rank and volume q^m.

Let $x \in L$ and write $x = (x_1, x_2, x_3)$ where $x_1 \in \mathbb{Z}^m$, $x_2 \in \mathbb{Z}^n$, and $x_3 \in \mathbb{Z}$. By the definition of L, we have

$$x_1 + Ax_2 - tx_3 = 0 \pmod q,$$

whence

$$x_1 = -Ax_2 + tx_3 + qu$$

for some $u \in \mathbb{Z}^m$. Thus,

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} qI_m & -A & t \\ 0 & I_n & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ x_2 \\ x_3 \end{bmatrix}$$

which shows that B is a basis matrix for L. □

The vector $x = (e, s, 1)$ belongs to this lattice L. By Theorem 6.6, each component e_i of e and each component s_j of s satisfies $E(e_i^2) = E(s_j^2) = \frac{\eta_1}{2}$. Hence, the expected length of x is

$$E(\|x\|) \approx \sqrt{E(\|x\|^2)} = \sqrt{(m+n)\frac{\eta_1}{2} + 1} \approx \sqrt{(m+n)\frac{\eta_1}{2}}.$$

Here the first approximation holds since $\|x\|$ concentrates tightly around $\sqrt{E(\|x\|^2)}$ for large $m+n$. On the other hand, the Gaussian Heuristic (8) predicts the length of a shortest nonzero vector v in L to be:

$$\|v\| \approx \sqrt{\frac{m+n+1}{2\pi e}} q^{\frac{m}{m+n+1}} \approx \sqrt{\frac{m+n}{2\pi e}} q^{\frac{1}{2}},$$

where the latter approximation uses $m \approx n$. Consequently, if

$$\eta_1 \ll q/(\pi e), \tag{91}$$

then x is expected to be an unusually short vector, and thus may reasonably be expected to be a shortest nonzero vector of L. For the ML-KEM-768 parameter set where $q = 3329$ and $\eta_1 = 2$, this condition is satisfied (since $2 \ll 389.8$).

Geometric Series Assumption (GSA). The GSA is used to estimate the performance of the BKZ lattice basis reduction algorithm. Let $L \subseteq \mathbb{Z}^n$ be a random full-rank lattice. The GSA asserts that the Gram-Schmidt norms $\|b_i^*\|$ of a β-BKZ-reduced basis $B = [b_1, \ldots, b_n]$ of L approximate a decreasing geometric series, i.e.,

$$\frac{\|b_{i+1}^*\|}{\|b_i^*\|} \approx r \quad \text{for } 1 \le i \le n-1 \tag{92}$$

for some $0 < r < 1$. Equivalently,

$$\|b_i^*\| \approx r^{i-1}\|b_1\| \quad \text{for } 1 \le i \le n. \tag{93}$$

Now by Theorem 8.8,

$$\mathrm{vol}(L) = \prod_{i=1}^{n} \|b_i^*\| \approx \prod_{i=1}^{n} r^{i-1}\|b_1\| = \|b_1\|^n r^{n(n-1)/2}.$$

Thus

$$\mathrm{vol}(L)^{1/n} \approx \|b_1\| r^{(n-1)/2}.$$

Recalling the definition (82) of the Hermite factor δ of lattice basis B, we obtain

$$r \approx \delta^{-2/(n-1)}. \tag{94}$$

Define the *root Hermite factor* of B by

$$\delta_0 = \delta_0(B) = \delta^{1/n}. \tag{95}$$

Then (94) becomes

$$r \approx \delta_0^{-2n/(n-1)} \approx \delta_0^{-2},$$

where the latter approximation holds for large n. Substituting into (93) yields the following estimate for $\|b_i^*\|$:

$$\|b_i^*\| \approx \delta_0^{-2(i-1)}\|b_1\| = \delta_0^{-2(i-1)}\delta_0^n \operatorname{vol}(L)^{1/n} = \delta_0^{n-2i+2}\operatorname{vol}(L)^{1/n} \text{ for } 1 \le i \le n.$$

Assuming both the Gaussian heuristic (8) and the GSA, it has been shown that for β-BKZ-reduced bases B of random full-rank lattices,

$$\lim_{n\to\infty} \delta_0(B) = \left(\frac{\beta}{2\pi e}(\pi\beta)^{\frac{1}{\beta}}\right)^{\frac{1}{2(\beta-1)}}. \tag{96}$$

Experimental evidence suggests that for fixed $\beta \ge 50$, this asymptotic estimate quickly converges as n gets larger. Consequently, we have the following approximation for the lengths of the Gram-Schmidt basis vectors of B:

$$\|b_i^*\| \approx \delta_0^{n-2i+2}\operatorname{vol}(L)^{1/n} \text{ for } 1 \le i \le n, \tag{97}$$

where

$$\delta_0 = \left(\frac{\beta}{2\pi e}(\pi\beta)^{\frac{1}{\beta}}\right)^{\frac{1}{2(\beta-1)}} \approx \beta^{\frac{1}{2\beta}}. \tag{98}$$

As one would expect, for fixed dimension n, larger values of β produce a smaller root Hermite factor, and hence a more reduced lattice basis, but at the cost of increased running time. Table 9.2 shows the root Hermite factors for some values of β.

β	50	100	200	300	400	500	600	700
δ_0	1.0121	1.0093	1.0063	1.0048	1.0040	1.0034	1.0030	1.0027

Table 9.2: Root Hermite factors for β-BKZ-reduced bases.

The attack. Let $d = m + n + 1$ denote the dimension of the lattice L in (89). Recall that the lattice vector we seek is $x = (e, s, 1)$, whose expected length is

$$t = \sqrt{(m+n)\frac{\eta_1}{2} + 1}.$$

The attack begins by selecting the smallest blocksize $\beta \geq 50$ for which

$$\sqrt{\frac{\beta}{d}}\, t \leq \delta_0^{2\beta - d} q^{m/d}. \tag{99}$$

The BKZ algorithm with blocksize β is then applied to the lattice basis (90). After sufficiently many tours, one expects to obtain a basis $[b_1, \ldots, b_d]$ satisfying the GSA, so that the Gram-Schmidt lengths $\|b_i^*\|$ are approximated by (97).

Now consider the local projected block

$$B[d - \beta + 1,\ d] = [\Pi_{d-\beta}(b_{d-\beta+1}),\ \Pi_{d-\beta}(b_{d-\beta+2}),\ \ldots,\ \Pi_{d-\beta}(b_d)],$$

whose vectors span the β-dimensional projected lattice $L' = L[d - \beta + 1, d]$. Recall that

$$b_{d-\beta+1}^* = \Pi_{d-\beta}(b_{d-\beta+1}).$$

Hence, by (97), we have

$$\|\Pi_{d-\beta}(b_{d-\beta+1})\| = \|b_{d-\beta+1}^*\| \approx \delta_0^{2\beta-d} q^{m/d}.$$

Let $x' = \Pi_{d-\beta}(x)$ denote the projection of the target vector. Assuming that x behaves independently of the reduced basis $[b_1, \ldots, b_d]$, one expects

$$\|x'\| \approx \sqrt{\beta\frac{\eta_1}{2}} \approx \sqrt{\frac{\beta}{d}}\, t.$$

Since condition (99) holds, it is expected that x' is a shortest nonzero vector in L'. Consequently, an SVP oracle applied to L' should recover x' (or $-x'$). Once x' is known, x can then be reconstructed (details omitted).

Analysis. As discussed in §9.3, no satisfactory upper bound is known for the number of tours performed by the BKZ algorithm. To sidestep this issue, the complexity of the primal attack is typically estimated by the running time of the best algorithm known for solving SVP in β-dimensional lattices, where β is selected as described above. Currently, the fastest known classical approach is *sieving*, with a running time of approximately $2^{0.292\beta}$. This estimate is called the *core-SVP hardness* of Kyber. Similarly, the *core-SVP quantum hardness* is estimated as $2^{0.256\beta}$, corresponding to the fastest known quantum algorithm for solving SVP in β-dimensional lattices. These security estimates should be regarded as conservative lower bounds on the actual security of Kyber.

	ML-KEM-512	ML-KEM-768	ML-KEM-1024
(k, η_1)	(2,3)	(3,2)	(4,2)
lattice dimension d'	1025	1483	1950
BKZ blocksize β	406	623	873
core-SVP hardness	119	182	255
core-SVP quantum hardness	104	159	223

Table 9.3: Core-SVP hardness for the standardized Kyber parameter sets.

The smallest feasible BKZ blocksize β was obtained by determining, for each $m' \leq m$, the smallest $\beta \geq 50$ satisfying (99), where m' denotes the number of LWE samples. The minimum such β is then used to estimate the core-SVP hardness of Kyber. The resulting estimates for the standardized Kyber parameter sets are presented in Table 9.3. In the table, $d' = m' + n + 1$.

Remark 9.18 (*state-of-the-art security analysis*) We presented a simplified security analysis of Kyber key generation. A comprehensive security analysis of Kyber-KEM would also include: (i) the use of a simulator to better predict the lengths of the Gram-Schmidt vectors after each BKZ tour; (ii) an analysis of the MLWE instance arising from the encryption operation (21), in which e_1 and e_2 may be drawn from distributions different from those of r; (iii) an analysis of dual attacks on LWE (see §4.4); and (iv) an analysis of recent improvements to hybrid dual attacks on MLWE.

9.7 Exercises

Exercise 9.1 Prove that a $(\beta + 1)$-BKZ-reduced lattice basis is also β-BKZ-reduced.

Exercise 9.2 Prove that a β-BKZ-reduced lattice basis is also LLL-reduced.

Exercise 9.3 Implement the naive and basic enumeration algorithms and use them to find all shortest nonzero vectors in the lattices defined in Exercise 8.10.

Exercise 9.4 Prove that $|\mathbb{Z}^n \cap B_n(1)| = 2n + 1$ and $|\mathbb{Z}^n \cap B_n(\sqrt{2})| = 2n^2 + 1$.

Exercise 9.5 The core-SVP hardness estimates for Dilithium are obtained by analyzing the performance of BKZ on the MLWE instance arising in key generation and the MSIS instance arising in signature generation (see equation 36). Study the methods used to assess core-SVP hardness in Sections C.2 and C.3 of the Dilithium specification [59] and then verify the core-SVP hardness values reported in Table 1 of [59].

10 LLL applications

Contents

The general methodology for applying the LLL algorithm proceeds as follows. One first constructs a lattice L in which the (unknown) solution to the problem at hand is encoded by a nonzero lattice vector v that is significantly shorter than the Gaussian heuristic (8) predicts for a shortest nonzero lattice vector. The LLL algorithm is then applied to compute a reduced basis B of L. Although LLL does not guarantee recovery of a shortest nonzero lattice vector, in practice its performance is often substantially better than suggested by the worst-case bounds, and in many practical instances the target vector v appears among the vectors of the reduced basis. One then inspects the basis B to determine whether v has been recovered; if so, the original problem can be solved efficiently.

This section illustrates this methodology through six applications of the LLL algorithm. The first application is the primal attack on short-secret LWE that was presented in §9.6. The second application is the dual attack on SIS_2 discussed previously in §3.3. The third application, discussed in §10.3, concerns the discovery of integer relations among real numbers. This problem is mainly of theoretical interest and may be skipped by readers whose primary focus is cryptographic applications. The fourth application, simultaneous Diophantine approximation, presented in §10.4, arises naturally in numerous cryptanalytic settings. The fifth application, described in §10.5, is a concrete side-channel attack on the ECDSA signature scheme. Lastly, §10.6 introduces a technique based on LLL for finding small roots of a univariate polynomial modulo a composite integer with unknown factorization; this technique has found numerous applications to problems related to RSA and integer factorization. Some other LLL applications are explored in the Exercises including solving approximate CVP (Exercises 10.5 and 10.6), solving the subset sum problem (Exercise 10.13), and writing a prime as a sum of four squares (Exercise 10.14).

10.1 Primal attack on short-secret LWE

Consider an ss-LWE(m, n, q, B) instance (A, t) (Definition 4.6), where $A \in_R \mathbb{Z}_q^{m\times n}$, $s \in_R [-B, B]^n$, $e \in_R [-B, B]^m$, and $t = As + e \pmod q$. The goal is to recover s and e.

Recall the primal attack on ss-LWE from §9.6. The vector $x = (e, s, 1) \in \mathbb{Z}^{m+n+1}$ belongs to the $(m + n + 1)$-dimensional lattice L_A of volume q^m with basis matrix

$$\begin{bmatrix} qI_m & -A & t \\ 0 & I_n & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The expected value of the square of a component of either e or s is

$$\frac{1}{2B+1}\sum_{i=-B}^{B} i^2 = \frac{2B(B+1)(2B+1)}{6(2B+1)} = \frac{B(B+1)}{3} \approx \frac{B^2}{3}.$$

Hence, the expected length of the target vector $x = (e, s, 1)$ is

$$\|x\| \approx \sqrt{(m+n)B^2/3 + 1} \approx B\sqrt{(m+n)/3}.$$

On the other hand, by the Gaussian heuristic (8), the expected length of a shortest nonzero vector in L_A is approximately

$$\sqrt{\frac{m+n+1}{2\pi e}}\, q^{\frac{m}{m+n+1}}.$$

Therefore, if

$$B\sqrt{\frac{m+n}{3}} \ll \sqrt{\frac{m+n+1}{2\pi e}}\, q^{\frac{m}{m+n+1}}$$

then $\pm x$ are expected to be the shortest nonzero vectors in L_A, and so can likely be determined by finding a sufficiently reduced basis of L_A.

Example 10.1 (*primal attack on short-secret LWE*) Let $m = 5$, $n = 3$, $q = 7919$, and $B = 100$, and consider the ss-LWE(m, n, q, B) instance (A, t) where

$$A = \begin{bmatrix} 306 & 7354 & 1517 \\ 5470 & 4491 & 670 \\ 6522 & 3601 & 5578 \\ 7427 & 917 & 607 \\ 3855 & 7914 & 896 \end{bmatrix} \quad \text{and} \quad t = \begin{bmatrix} 5891 \\ 4669 \\ 6299 \\ 4043 \\ 5973 \end{bmatrix}.$$

A basis matrix for the associated 9-dimensional lattice L_A is

$$\begin{bmatrix} 7919 & 0 & 0 & 0 & 0 & 7613 & 565 & 6402 & 5891 \\ 0 & 7919 & 0 & 0 & 0 & 2449 & 3428 & 7249 & 4669 \\ 0 & 0 & 7919 & 0 & 0 & 1397 & 4318 & 2341 & 6299 \\ 0 & 0 & 0 & 7919 & 0 & 492 & 7002 & 7312 & 4043 \\ 0 & 0 & 0 & 0 & 7919 & 4064 & 5 & 7023 & 5973 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The LLL algorithm finds the following LLL-reduced basis for L:

$$\begin{aligned} b_1 &= (-19, 33, -80, -20, -8, 53, 48, -62, 1) \\ b_2 &= (-14, 30, 6, 44, -11, -11, 89, 25, 39) \\ b_3 &= (-52, -94, 87, 15, 12, -29, 30, -17, 45) \\ b_4 &= (43, 6, -19, -114, -30, -28, 29, -65, 57) \\ b_5 &= (101, -34, 18, -31, -17, 21, 91, 107, -54) \\ b_6 &= (-46, 31, 91, -69, -126, -14, 31, 61, 12) \\ b_7 &= (-20, -6, -119, 16, 3, -111, -76, 47, 19) \\ b_8 &= (-83, 46, 13, -28, 80, -61, -20, 34, -135) \\ b_9 &= (-11, -57, 89, 4, -72, -76, 58, -35, -118). \end{aligned}$$

The ss-LWE solution (s, e) is extracted from b_1:

$$s = \begin{bmatrix} 53 \\ 48 \\ -62 \end{bmatrix}, \quad e = \begin{bmatrix} -19 \\ 33 \\ -80 \\ -20 \\ -8 \end{bmatrix}.$$

10.2 Dual attack on SIS_2

Consider an $SIS_2(n, m, q, B)$ instance (Definition 3.13) defined by a uniformly random matrix $A \in_R \mathbb{Z}_q^{n \times m}$. The goal is to find a vector $z \in \mathbb{Z}^m$ satisfying $0 < \|z\|_2 \leq B$ and $Az = 0 \pmod q$.

Recall the dual attack on SIS_2 from §3.3.1. The associated SIS lattice is

$$L_A^\perp = \{z \in \mathbb{Z}^m : Az = 0 \pmod q\}.$$

Let $[I_n | \overline{A}]$ denote the reduced row echelon form of A over $\mathbb{Z}_q$. A basis matrix for $L_A^\perp$ is

$$C = \begin{bmatrix} qI_n & -\overline{A} \\ 0 & I_{m-n} \end{bmatrix} \in \mathbb{Z}^{m \times m}.$$

The dual attack proceeds by computing a reduced basis for $L_A^\perp$, and then identifying any of its vectors whose length does not exceed B.

Example 10.2 (*dual attack on SIS_2*) Let $n = 5$, $m = 9$, $q = 97$, and $B = 12$, and consider the $\mathrm{SIS}_2(n, m, q, B)$ instance

$$A = \begin{bmatrix} 77 & 96 & 10 & 86 & 58 & 36 & 80 & 22 & 44 \\ 39 & 60 & 39 & 43 & 12 & 55 & 2 & 24 & 71 \\ 45 & 29 & 21 & 48 & 7 & 33 & 57 & 65 & 16 \\ 93 & 96 & 71 & 44 & 70 & 58 & 25 & 29 & 73 \\ 57 & 40 & 78 & 88 & 25 & 44 & 67 & 88 & 5 \end{bmatrix}.$$

The reduced row echelon form of A over $\mathbb{Z}_q$ is

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 24 & 63 & 3 & 15 \\ 0 & 1 & 0 & 0 & 0 & 50 & 42 & 20 & 13 \\ 0 & 0 & 1 & 0 & 0 & 14 & 75 & 9 & 40 \\ 0 & 0 & 0 & 1 & 0 & 17 & 43 & 7 & 25 \\ 0 & 0 & 0 & 0 & 1 & 39 & 4 & 40 & 4 \end{bmatrix},$$

and so a basis matrix for the associated lattice $L_A^\perp$ is

$$C = \begin{bmatrix} 97 & 0 & 0 & 0 & 0 & 73 & 34 & 94 & 82 \\ 0 & 97 & 0 & 0 & 0 & 47 & 55 & 77 & 84 \\ 0 & 0 & 97 & 0 & 0 & 83 & 22 & 88 & 57 \\ 0 & 0 & 0 & 97 & 0 & 80 & 54 & 90 & 72 \\ 0 & 0 & 0 & 0 & 97 & 58 & 93 & 57 & 93 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The LLL algorithm finds the following LLL-reduced basis for $L_A^\perp$:

$$\begin{aligned} b_1 &= (-4, 1, -2, 9, -2, 1, 2, 1, 3) & b_2 &= (1, 1, 0, -5, -4, 4, 9, 5, 0) \\ b_3 &= (3, 0, -8, 2, -5, -1, -9, 2, 0) & b_4 &= (1, -9, 5, 5, -2, 4, 9, 0, 1) \\ b_5 &= (-7, -4, -3, 1, 3, 5, 2, 0, -3) & b_6 &= (-8, -8, -1, 2, -5, -8, 1, -4, -3) \\ b_7 &= (7, -1, -4, 0, 6, -1, 4, -11, -7) & b_8 &= (0, 3, 11, 6, 0, 0, -3, 1, -7) \\ b_9 &= (-3, 4, 10, -3, -7, 8, 0, -6, 8). \end{aligned}$$

The basis vectors b_1 and b_5 have lengths 11 and $\sqrt{122}$, respectively, and so are SIS_2 solutions.

10.3 Finding integer relations

Let $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$ be a vector of real numbers. An *integer relation for* β is a linear combination

$$c_1\beta_1 + c_2\beta_2 + \cdots + c_n\beta_n = 0, \tag{100}$$

where $c_1, c_2, \ldots, c_n$ are integers. An integer relation algorithm attempts to find such a nonzero vector $c = (c_1, c_2, \ldots, c_n)$, preferably with small coordinates, or determines that no nontrivial relation exists.

In practice, computations must be performed using finite-precision arithmetic. Consequently, integer relation algorithms operate on rational approximations to the real numbers β_i. Specifically, one is given a vector $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{Q}^n$ satisfying $\alpha_i \approx \beta_i$ for each i. If the algorithm produces a nonzero vector $c \in \mathbb{Z}^n$ such that

$$c_1\alpha_1 + c_2\alpha_2 + \cdots + c_n\alpha_n = 0,$$

this suggests that the corresponding integer relation (100) holds for the original real numbers. The validity of the relation can then be verified using high-precision computations, after which one may seek a mathematical proof.

In this section, we explain how the LLL algorithm can be applied to the problem of finding integer relations. As an illustrative example, we also discuss its use in discovering *Machin-like formulas*, which are integer relations involving π and values of the arctangent function.

Given a rational approximation $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{Q}^n$ to $\beta = (\beta_1, \ldots, \beta_n) \in \mathbb{R}^n$, select a *scaling factor* $Q \in \mathbb{Z}$ and define a lattice $L \subseteq \mathbb{Q}^{n+1}$ with basis matrix

$$B = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ Q\alpha_1 & Q\alpha_2 & Q\alpha_3 & \cdots & Q\alpha_n \end{bmatrix}_{(n+1)\times n}. \tag{101}$$

The lattice L has rank n and is therefore *not* a full-rank lattice. Its vectors are of the form

$$v = (x_1, x_2, \ldots, x_n, Q(\alpha_1 x_1 + \cdots + \alpha_n x_n)),$$

where $x_1, \ldots, x_n \in \mathbb{Z}$. If such a vector v has small length, then the coordinates x_i must themselves be small. Furthermore, for a sufficiently large scaling factor Q, the rational number $|\sum_{i=1}^n \alpha_i x_i|$ is expected to be very small, so that the vector $x = (x_1, \ldots, x_n)$ is a promising candidate for an integer relation $x_1\beta_1 + \cdots + x_n\beta_n = 0$.

The LLL algorithm is applied to compute an LLL-reduced basis of L. One then examines the reduced basis vectors to determine whether any of them provide plausible candidates for an integer relation. The effectiveness of this approach depends on the choice of scaling factor Q and on the accuracy of the approximations $\alpha_i \approx \beta_i$, and typically requires substantial experimentation.

Example 10.3 (*discovering a Machin-like integer relation*) Consider the real numbers

$$\beta_1 = \arctan\left(\frac{1}{18}\right), \quad \beta_2 = \arctan\left(\frac{1}{57}\right), \quad \beta_3 = \arctan\left(\frac{1}{239}\right), \quad \beta_4 = \pi,$$

and their rational approximations

$$\alpha_1 = \frac{2432}{43821}, \quad \alpha_2 = \frac{11971}{682417}, \quad \alpha_3 = \frac{717}{171364}, \quad \alpha_4 = \frac{104348}{33215}.$$

Choose the scaling factor $Q = 100000000$. The associated lattice $L \subseteq \mathbb{Q}^5$ is defined by the following basis matrix

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ Q\alpha_1 & Q\alpha_2 & Q\alpha_3 & Q\alpha_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \frac{243200000000}{43821} \\ 0 & 1 & 0 & 0 & \frac{1197100000000}{682417} \\ 0 & 0 & 1 & 0 & \frac{17925000000}{42841} \\ 0 & 0 & 0 & 1 & \frac{2086960000000}{6643} \end{bmatrix}^T.$$

Applying the LLL algorithm yields the following reduced basis:

$$\tilde{B} = \begin{bmatrix} -48 & -32 & 20 & 1 & -\frac{436321916060000000}{2836839193980133797} \\ -40 & 112 & 61 & 0 & -\frac{108699521075000000}{1281125633289237} \\ -73 & 75 & -97 & 1 & \frac{766445843743945000000}{8510517581940401391} \\ -34 & 83 & 103 & 0 & \frac{209280728875000000}{1281125633289237} \end{bmatrix}^T.$$

The first basis vector is

$$\begin{aligned} v &= (x_1, x_2, x_3, x_4, Q(x_1\alpha_1 + x_2\alpha_2 + x_3\alpha_3 + x_4\alpha_4)) \\ &= \left(-48, -32, 20, 1, -\frac{436321916060000000}{2836839193980133797}\right) \end{aligned}$$

with

$$x_1\alpha_1 + x_2\alpha_2 + x_3\alpha_3 + x_4\alpha_4 \approx -0.1538 \times 10^{-8}.$$

Hence, $c = (-48, -32, 20, 1)$ is a plausible candidate solution vector. A high-precision floating point computation confirms that

$$x_1\beta_1 + x_2\beta_2 + x_3\beta_3 + x_4\beta_4 \approx 0.$$

Consequently, a candidate integer relation is

$$\frac{\pi}{4} = 12\arctan\left(\frac{1}{18}\right) + 8\arctan\left(\frac{1}{57}\right) - 5\arctan\left(\frac{1}{239}\right). \tag{102}$$

This relation was originally discovered and proven by Gauss using properties of the Gaussian integers.

10.4 Simultaneous Diophantine approximation

Simultaneous Diophantine approximation concerns the problem of finding good rational approximations to several real numbers, using a small common denominator. More precisely, a simultaneous Diophantine approximation of the vector of real numbers $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$ is a vector of rational numbers $(p_1/q, p_2/q, \ldots, p_n/q)$ that share the same (small) denominator q, and such that $p_i/q \approx \beta_i$ for $1 \le i \le n$.

The LLL algorithm can be used to efficiently find simultaneous Diophantine approximations that are good in the following sense. Let α_i be a close rational number approximation to β_i for $1 \le i \le n$, and let $\varepsilon \in \mathbb{Q}$ with $0 < \varepsilon < 1$. We will show how LLL can be used to efficiently find integers $p_1, \ldots, p_n, q$ such that

$$1 \le q \le 2^{n(n+1)/4}/\varepsilon^n \tag{103}$$

(this is the sense in which q is small) and

$$|p_i - q\alpha_i| \le \varepsilon \text{ for } 1 \le i \le n \tag{104}$$

(and so either p_i/q or $-p_i/q$ is a close approximation to α_i).

Let $L \subseteq \mathbb{Q}^{n+1}$ be the lattice with basis matrix

$$B = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & -\alpha_1 \\ 0 & 1 & 0 & \cdots & 0 & -\alpha_2 \\ 0 & 0 & 1 & \cdots & 0 & -\alpha_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -\alpha_n \\ 0 & 0 & 0 & \cdots & 0 & \varepsilon^{n+1}/2^{n(n+1)/4} \end{bmatrix}_{(n+1)\times(n+1)}, \tag{105}$$

and let $\tilde{B} = [b_1, \ldots, b_n, b_{n+1}]$ be an LLL-reduced basis for L. Note that

$$\mathrm{vol}(L) = \varepsilon^{n+1}/2^{n(n+1)/4}.$$

By Theorem 8.17 we have

$$\|b_1\| \le 2^{n/4}\,\mathrm{vol}(L)^{1/(n+1)} = \varepsilon.$$

Since $b_1 \in L$, we can write

$$b_1 = (p_1 - q\alpha_1,\ p_2 - q\alpha_2,\ \ldots,\ p_n - q\alpha_n,\ q \cdot \varepsilon^{n+1}/2^{n(n+1)/4}), \tag{106}$$

where $p_1, p_2, \ldots, p_n, q \in \mathbb{Z}$. Hence

$$\|b_1\|^2 = (p_1 - q\alpha_1)^2 + (p_2 - q\alpha_2)^2 + \cdots + (p_n - q\alpha_n)^2 + q^2(\varepsilon^{n+1}/2^{n(n+1)/4})^2 \le \varepsilon^2. \quad (107)$$

It follows that

$$|p_i - q\alpha_i| \le \varepsilon \ \text{ for } \ 1 \le i \le n,$$

and

$$|q| \le 2^{n(n+1)/4}/\varepsilon^n.$$

Since $\|b_1\| \neq 0$ and $\varepsilon < 1$, the inequality (107) implies that $q \neq 0$.

Example 10.4 (*simultaneous Diophantine approximation*) Let us find small rational approximations with a common denominator to the irrational numbers $\beta_1 = \sqrt{2}$, $\beta_2 = \sqrt{3}$, $\beta_3 = e$ and $\beta_4 = \pi$ with $n = 4$ and $\varepsilon = \frac{1}{3}$. We begin with the following rational approximations obtained from the decimal number representations of the β_i's:

$$\begin{aligned}
\sqrt{2} \approx \alpha_1 &= 1414213562/1000000000 = 707106781/500000000,\\
\sqrt{3} \approx \alpha_2 &= 1732050808/1000000000 = 216506351/125000000,\\
e \approx \alpha_3 &= 2718281828/1000000000 = 679570457/250000000,\\
\pi \approx \alpha_4 &= 3141592654/1000000000 = 1570796327/500000000.
\end{aligned}$$

The associated lattice L has basis matrix

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 & -\alpha_1 \\ 0 & 1 & 0 & 0 & -\alpha_2 \\ 0 & 0 & 1 & 0 & -\alpha_3 \\ 0 & 0 & 0 & 1 & -\alpha_4 \\ 0 & 0 & 0 & 0 & \varepsilon^{n+1}/2^{n(n+1)/4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & -707106781/500000000 \\ 0 & 1 & 0 & 0 & -216506351/125000000 \\ 0 & 0 & 1 & 0 & -679570457/250000000 \\ 0 & 0 & 0 & 1 & -1570796327/500000000 \\ 0 & 0 & 0 & 0 & 1/7776 \end{bmatrix}.$$

The LLL algorithm produces the following basis:

$$\tilde{B} = \begin{bmatrix} -\frac{2362823}{62500000} & \frac{50252533}{500000000} & -\frac{11147309}{125000000} & -\frac{18723299}{250000000} & -\frac{21681981}{500000000} \\ -\frac{1277133}{15625000} & -\frac{15544457}{125000000} & \frac{954961}{31250000} & -\frac{1461329}{62500000} & -\frac{11425551}{125000000} \\ \frac{1902069}{31250000} & -\frac{6993199}{250000000} & -\frac{5179273}{62500000} & \frac{17116697}{125000000} & -\frac{23204857}{250000000} \\ -\frac{1095141}{62500000} & \frac{4425711}{500000000} & -\frac{16607703}{125000000} & -\frac{24379033}{250000000} & \frac{66265273}{500000000} \\ \frac{83}{972} & \frac{7}{7776} & -\frac{37}{648} & -\frac{107}{1296} & -\frac{799}{7776} \end{bmatrix}.$$

The first basis vector is

$$b_1 = \left(-\frac{2362823}{62500000}, -\frac{1277133}{15625000}, \frac{1902069}{31250000}, -\frac{1095141}{62500000}, \frac{83}{972}\right).$$

Using (106), we obtain (after adjusting signs) $q = 664$, $p_1 = 939$, $p_2 = 1150$, $p_3 = 1805$, $p_4 = 2086$. The resulting simultaneous Diophantine approximations to the real numbers $\sqrt{2}$, $\sqrt{3}$, e and π are compared below.

real number	rational approximation	decimal rep.	smaller rational approximation	decimal rep.
$\sqrt{2}$	707106781/500000000	1.414213562	939/664	1.414156627
$\sqrt{3}$	216506351/125000000	1.732050808	1150/664	1.731927711
e	679570457/250000000	2.718281828	1805/664	2.718373494
π	1570796327/500000000	3.141592654	2086/664	3.141566265

Example 10.5 With $n = 5$ and $\varepsilon = \frac{1}{2}$, the LLL algorithm was used to find the following small rational approximations with a common denominator:

$$\sqrt{2} \approx \frac{481}{340}, \quad \sqrt{13} \approx \frac{1226}{340}, \quad \sqrt{29} \approx \frac{1831}{340}, \quad \sqrt{47} \approx \frac{2331}{340}, \quad \sqrt{71} \approx \frac{2865}{340}.$$

10.5 Side-channel attack on ECDSA

The LLL lattice basis reduction algorithm, together with refinements such as BKZ, has played a central role in the cryptanalysis of many public-key cryptosystems. One of the earliest and most striking applications of LLL was the successful cryptanalysis of knapsack-based public-key encryption schemes. These schemes, based on the subset sum problem, were among the very first public-key proposals of the 1970s. Since then, LLL-based techniques have also been applied to RSA, DSA, and ECDSA, especially in settings where partial information about secret key material is exposed. Such leakage can result from flawed implementations, poor parameter selection, or side-channel attacks. This section presents a side-channel attack on ECDSA.

ECDSA. The *domain parameters* for the Elliptic Curve Digital Signature Algorithm (ECDSA) consist of an elliptic curve E defined over a prime field $\mathbb{Z}_p$, chosen so that the number n of points in $E(\mathbb{Z}_p)$ is prime, together with a generator point $P \in E(\mathbb{Z}_p)$. A user, Alice, selects a secret integer $a \in_R [1, n-1]$ and computes the point $A = aP$. The point A serves as Alice's *long-term public key*, while a is her corresponding *long-term private key*.

To sign a message M, Alice first computes its hash $m = H(M)$. She then chooses a *per-message secret* $k \in_R [1, n-1]$, and computes the point $R = kP$. Let $r \in [1, n-1]$ denote the x-coordinate of R reduced modulo n. Alice also computes the integer $s \in [1, n-1]$ as

$$s = k^{-1}(m + ar) \bmod n. \tag{108}$$

Alice's signature on M is the pair of integers (r, s). Since the side-channel attack of interest targets the signature generation process, the verification algorithm is omitted.

For ECDSA to be secure, both the long-term private key a and each per-message secret key k must be chosen uniformly at random from $[1, n-1]$, and a fresh value of k must be used for every signature. In practice, however, partial information about some per-message secrets k may leak during the signing process. Such leakage can arise from weaknesses including poor random number generation, timing variations, electromagnetic emanations, and so on. In some cases, the leaked information is sufficient to recover the long-term private key a. Once a is compromised, the entire ECDSA implementation is effectively broken.

Leakage with two signatures. We will first consider a setting in which an attacker learns substantial partial information about the per-message secrets used in two ECDSA signatures. Specifically, suppose the attacker discovers that roughly the most significant half of the bits of each per-message secret k are zero. Although this scenario is rather contrived, it provides a clean illustration of how lattice techniques can be applied in side-channel attacks. We therefore assume that the secrets satisfy $k_1, k_2 \leq K$, where $K \ll n$ is a parameter to be specified later.

Assume the adversary obtains two signed messages (M_1, r_1, s_1) and (M_2, r_2, s_2). From the two ECDSA signature equations (108), the adversary can derive the linear congruences:

$$k_1 s_1 \equiv m_1 + a r_1 \pmod{n} \quad \text{and} \quad k_2 s_2 \equiv m_2 + a r_2 \pmod{n}.$$

Solving each congruence for a and equating the results yields the congruence

$$k_1 s_1 r_2 - m_1 r_2 \equiv k_2 s_2 r_1 - m_2 r_1 \pmod{n}.$$

Rearranging terms gives the linear congruence

$$k_1 - (s_1 r_2)^{-1} s_2 r_1 k_2 + (s_1 r_2)^{-1}(m_2 r_1 - m_1 r_2) \equiv 0 \pmod{n},$$

in which the only quantities that are not known to the adversary are the two per-message secrets k_1 and k_2. This may be written more compactly as

$$k_1 + t k_2 + u \equiv 0 \pmod{n}$$

where

$$t = -(s_1 r_2)^{-1} s_2 r_1 \bmod n \quad \text{and} \quad u = (s_1 r_2)^{-1}(m_2 r_1 - m_1 r_2) \bmod n.$$

Equivalently, the congruence can be expressed as the equation

$$k_1 + t k_2 + u + n x = 0,$$

where $x \in \mathbb{Z}$. The values t and u can be computed by the adversary.

Now, consider the integer lattice $L \subseteq \mathbb{Z}^3$ with basis matrix

$$B = \begin{bmatrix} n & t & u \\ 0 & 1 & 0 \\ 0 & 0 & K \end{bmatrix}.$$

Observe that

$$v = B[x, k_2, 1]^T = [xn + tk_2 + u, k_2, K]^T = [-k_1, k_2, K]^T,$$

so $v = (-k_1, k_2, K) \in L$. Moreover, the length of v satisfies

$$\|v\| = \sqrt{k_1^2 + k_2^2 + K^2} \leq \sqrt{3K^2} = \sqrt{3}K.$$

The lattice L has volume $\det(B) = nK$. By the Gaussian heuristic (8),

$$\lambda_1(L) \approx \sqrt{3/2\pi e}\,(nK)^{1/3} \approx (nK)^{1/3}.$$

Consequently, if $\|v\| \ll (nK)^{1/3}$, which reduces to the condition $K \ll \sqrt{n}$, then v is expected to be an unusually short lattice vector and, in particular, to be a shortest nonzero vector of L. In this case, the adversary can plausibly recover v by applying the LLL algorithm to compute a reduced basis for L, and checking whether $\pm v$ appears among the basis vectors. Once v is known, the values of k_1 and k_2 are immediately revealed, and the long-term private key a can then be recovered.

Leakage with many signatures. We now turn to a more realistic setting in which an adversary obtains several ECDSA signed messages (M_i, r_i, s_i), $1 \leq i \leq N$, and also learns the ℓ least significant bits of each of the corresponding per-message secrets k_i. Writing $k_i = \alpha_i 2^\ell + \beta_i$ where $\beta_i < 2^\ell$, we assume the values β_i are known to the adversary, while the α_i remain unknown. Let $K = \lceil n/2^\ell \rceil$, so $\alpha_i < K$.

For each i, $1 \leq i \leq N-1$, eliminate a from the two ECDSA signing equations (108) corresponding to message M_i and the last message M_N. This yields a linear congruence of the form

$$k_i + t_i' k_N + u_i' \equiv 0 \pmod{n},$$

where the only quantities that are not known to the adversary are k_i and k_N. Substituting $k_i = \alpha_i 2^\ell + \beta_i$ and $k_N = \alpha_N 2^\ell + \beta_N$ gives

$$(\alpha_i 2^\ell + \beta_i) + t_i'(\alpha_N 2^\ell + \beta_N) + u_i' \equiv 0 \pmod{n}.$$

Multiplying both sides by the inverse of $2^\ell \bmod n$, simplifying, and eliminating the "mod n" operator, results in $N-1$ equations

$$\alpha_i + t_i \alpha_N + u_i + x_i n = 0,$$

where the unknown quantities are the α_i for $1 \leq i \leq N$ and the integers x_i for $1 \leq i \leq N-1$.

Next, consider the $(N+1)$-dimensional integer lattice $L \subseteq \mathbb{Z}^{N+1}$ with basis matrix

$$B = \begin{bmatrix} n & 0 & \cdots & 0 & t_1 & u_1 \\ 0 & n & \cdots & 0 & t_2 & u_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & n & t_{N-1} & u_{N-1} \\ 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & K \end{bmatrix}_{(N+1)\times(N+1)}.$$

Observe that

$$v = B[x_1, x_2, \ldots, x_{N-1}, \alpha_N, 1]^T = [-\alpha_1, -\alpha_2, \ldots, -\alpha_{N-1}, \alpha_N, K]^T$$

is a lattice vector of length

$$\|v\| = \sqrt{\alpha_1^2 + \cdots + \alpha_N^2 + K^2} \leq \sqrt{N+1}\,K \approx \sqrt{N}K.$$

The lattice L has volume $\det(B) = n^{N-1}K$. By the Gaussian heuristic (8),

$$\lambda_1(L) \approx \sqrt{(N+1)/2\pi e}\,(n^{N-1}K)^{1/(N+1)} \approx N^{1/2}n^{(N-1)/(N+1)}K^{1/(N+1)}.$$

For the attack to succeed, we require that $\|v\|$ be significantly smaller than this estimate for $\lambda_1(L)$, meaning

$$N^{1/2}K \ll N^{1/2}n^{(N-1)/(N+1)}K^{1/(N+1)},$$

which simplifies to the condition $K^N \ll n^{N-1}$.

Let t denote the bitlength of n, so that $\log_2 K \approx t - \ell$. By taking logarithms of both sides, the inequality $K^N \ll n^{N-1}$ then becomes $N(t-\ell) \ll Nt - t$, or equivalently, $N \gg t/\ell$. Hence, if the adversary obtains signed messages for which the ℓ least significant bits of the per-message secrets are known across $N \gg t/\ell$ messages, one can expect v to be an exceptionally short lattice vector and, in particular, to be a shortest nonzero vector in L. In this case, applying LLL to compute a reduced basis for L and checking whether $\pm v$ appears among the basis vectors is likely to recover v. From this, the adversary can determine the values of k_i and subsequently recover the long-term private key a.

Example 10.6 (*side-channel attack on ECDSA*) Select $n = 935878837$ which is a $t = 30$-bit prime number, $\ell = 6$, and $N = 10$. We have $K = \lceil n/2^\ell \rceil = 29246214$. Suppose that the adversary has obtained 10 ECDSA-signed messages and knows the 6 least significant bits of each of the corresponding per-message secrets. Assume that the unknown portions α_i of the per-message secrets are:

$\alpha_1 = 9550684$ $\alpha_2 = 11443958$ $\alpha_3 = 8533292$ $\alpha_4 = 505823$ $\alpha_5 = 9809889$
$\alpha_6 = 14944555$ $\alpha_7 = 16279651$ $\alpha_8 = 1484325$ $\alpha_9 = 21735620$ $\alpha_{10} = 3788058$.

Also, assume that the quantities t_i that the adversary computed are:

$t_1 = 201354591$ $t_2 = 891235047$ $t_3 = 922939225$ $t_4 = 934992634$ $t_5 = 676943009$
$t_6 = 889775274$ $t_7 = 471852626$ $t_8 = 206613081$ $t_9 = 215952988$.

Let $u_i = -\alpha_i - t_i\alpha_{10}$ for $1 \leq i \leq 9$, so:

$u_1 = 245087712$ $u_2 = 884448799$ $u_3 = 274011166$ $u_4 = 926348469$ $u_5 = 199874056$
$u_6 = 477691203$ $u_7 = 847522394$ $u_8 = 33208359$ $u_9 = 802959243$.

As shown above, the adversary is able to compute the u_i without knowledge of the α_i. The adversary's task is to determine the α_i, given n, t_i and u_i.

The associated lattice is $L = L(B) \subseteq \mathbb{Z}^{11}$ where

$$B = \begin{bmatrix} n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 201354591 & 245087712 \\ 0 & n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 891235047 & 884448799 \\ 0 & 0 & n & 0 & 0 & 0 & 0 & 0 & 0 & 922939225 & 274011166 \\ 0 & 0 & 0 & n & 0 & 0 & 0 & 0 & 0 & 934992634 & 926348469 \\ 0 & 0 & 0 & 0 & n & 0 & 0 & 0 & 0 & 676943009 & 199874056 \\ 0 & 0 & 0 & 0 & 0 & n & 0 & 0 & 0 & 889775274 & 477691203 \\ 0 & 0 & 0 & 0 & 0 & 0 & n & 0 & 0 & 471852626 & 847522394 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & n & 0 & 206613081 & 33208359 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & n & 215952988 & 802959243 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 29246214 \end{bmatrix}_{11\times 11}.$$

Applying the LLL algorithm to this basis produces the reduced basis shown in Figure 10.1. The first vector of the basis is the desired one $(-\alpha_1, \ldots, -\alpha_9, \alpha_{10}, K)$. The adversary can identify this vector by noticing that K appears in its last component, then extracting the α_i, computing the k_i (using the known values β_i) and thereafter the long-term private key a, and verifying that $A = aP$.

Extensive experimental results indicate that the attack can succeed even when as few as $\ell = 2$ bits of per-message secrets are leaked. Moreover, this approach can be adapted to settings in which the adversary learns a small number of most significant bits, or even a few middle bits, of the per-message secrets. In practice, these techniques have proven effective against poor implementations of ECDSA systems, including real-world Bitcoin applications.

10.6 Finding small roots of a polynomial modulo N

Univariate polynomials with integer coefficients can be efficiently factored over the integers or, equivalently, over the rational numbers. In fact, the first major application of the LLL lattice basis reduction algorithm was a polynomial-time method for this task. There are also randomized algorithms for factoring polynomials modulo a prime p that run in expected polynomial time. In contrast, no efficient algorithm is known for factoring polynomials modulo a composite number N; indeed, this problem is as hard as factoring N itself.

In 1996, Coppersmith demonstrated that the LLL algorithm can be used to find all sufficiently small roots of a polynomial $f(x) \in \mathbb{Z}_N[x]$, where N is a composite number whose factorization is unknown. More precisely, for a degree-d polynomial $f(x)$, the method recovers all roots x_0 satisfying $|x_0| \leq N^{1/d}$.

In §10.6.1, we present Howgrave-Graham's formulation of Coppersmith's algorithm, which is nonetheless commonly referred to as Coppersmith's method. Then, in §10.6.2 and §10.6.3, we describe two applications that exploit weaknesses in some padding mechanisms for the RSA public-key encryption scheme.

$$\left[\begin{array}{rrrrrr}
-9550684 & -11443958 & -8533292 & -505823 & -9809889 & -14944555 \\
-16279651 & -1484325 & -21735620 & 3788058 & 29246214 & \\
\hline
-41311850 & 16900714 & 41413030 & -93099186 & -8553329 & -2542146 \\
1419659 & -25764384 & -3240651 & -584960 & 29246214 & \\
\hline
34135239 & -6867183 & 36587811 & -33449287 & 27568406 & 2322804 \\
-67573114 & 12645502 & 13904351 & 73069518 & 0 & \\
\hline
44355001 & -3193014 & 12572852 & 227503 & 3773040 & -81867128 \\
32886850 & -8674033 & 39863888 & 71775813 & 0 & \\
\hline
-15272374 & 42386155 & -35766423 & -13087977 & -32724161 & -33155395 \\
-42424262 & 30657732 & 19663684 & 69776738 & -58492428 & \\
\hline
12219651 & -89373135 & 42673924 & -64744357 & -40234085 & 19955255 \\
-22397523 & -34546996 & -11644606 & 6163219 & -29246214 & \\
\hline
19729402 & -7687782 & -2384161 & 55173765 & -81415633 & 31377161 \\
77473667 & -55455163 & 5685632 & 38448777 & 0 & \\
\hline
55358855 & 34048060 & -48308186 & -38930222 & 13619789 & 7485867 \\
-30183366 & 34740289 & -83396757 & -32176865 & -58492428 & \\
\hline
8548959 & 4082825 & -10754428 & -1310998 & 15376068 & 24531739 \\
-33765648 & -89629917 & 44660893 & -32612045 & 58492428 & \\
\hline
-16512091 & 5472975 & 62360251 & 45786710 & -32495195 & -49558677 \\
20009308 & -113304986 & 15649897 & -34101103 & -29246214 & \\
\hline
-9597542 & 29413244 & 20704559 & -36375836 & 55150417 & -6234725 \\
-18905490 & -64921672 & -60601488 & 55864289 & -29246214 &
\end{array}\right]$$

Figure 10.1: The LLL-reduced basis in Example 10.6. Due to space constraints, each row of the matrix spans two lines.

10.6.1 The Coppersmith/Howgrave-Graham method

Let $f(x) \in \mathbb{Z}[x]$ be a monic polynomial of degree d, and let N be a composite number with unknown factorization. Suppose that x_0 is a small root of $f(x)$ modulo N. Coppersmith's method begins by applying the LLL algorithm to construct a polynomial $G(x)$ with small integer coefficients such that

$$G(x_0) \equiv 0 \pmod{N^\alpha}$$

for some $\alpha \geq 1$. If both x_0 and the coefficients of G are sufficiently small, then one expects that

$$-N^\alpha < G(x_0) < N^\alpha,$$

which forces $G(x_0) = 0$. Consequently, x_0 can be recovered by computing all integer roots β of $G(x)$ and checking which of them satisfy $f(\beta) \equiv 0 \pmod{N}$.

Algorithm description. Fix an integer $h \geq 2$ (the choice of h will be discussed later), and set $n = dh$. Define the solution bound

$$X = \left\lceil 2^{-1/2}\, N^{(h-1)/(n-1)}\, n^{-1/(n-1)} \right\rceil - 1. \tag{109}$$

Note that $X \longrightarrow N^{1/d}/\sqrt{2}$ as $h \longrightarrow \infty$. The goal is to efficiently determine all integers $x_0 \in [-X, X]$ such that $f(x_0) \equiv 0 \pmod{N}$.

For each $i \in [0, h-1]$ and $j \in [0, d-1]$, define the polynomial

$$g_{i,j}(x) = N^{h-1-i} x^j f(x)^i.$$

Then $\deg(g_{i,j}) = di + j \leq n - 1$, and any root x_0 of $f(x)$ modulo N satisfies

$$g_{i,j}(x_0) \equiv 0 \pmod{N^{h-1}}.$$

Index these polynomials as $G_\ell(x)$ for $1 \leq \ell \leq n$ by setting

$$G_\ell(x) = g_{i,j}(x), \quad \text{where } j = (\ell - 1) \bmod d \ \text{ and } \ i = \lfloor (\ell - 1)/d \rfloor.$$

Next, let B be the $n \times n$ matrix whose ℓth row consists of the coefficients of $G_\ell(xX)$. That is, if $G_\ell(x) = \gamma_0 + \gamma_1 x + \cdots + \gamma_{n-1} x^{n-1}$, then the ℓth row of B is

$$[\gamma_0,\ \gamma_1 X,\ \gamma_2 X^2,\ \ldots,\ \gamma_{n-1} X^{n-1}].$$

Consider the integer lattice $L = L(B) \subseteq \mathbb{Z}^n$ spanned by the rows of B. Since B is a lower triangular matrix with diagonal entries

$$\begin{array}{cccc}
N^{h-1}, & N^{h-1}X, & \ldots, & N^{h-1}X^{d-1}, \\
N^{h-2}X^{d}, & N^{h-2}X^{d+1}, & \ldots, & N^{h-2}X^{2d-1}, \\
N^{h-3}X^{2d}, & N^{h-3}X^{2d+1}, & \ldots, & N^{h-3}X^{3d-1}, \\
\cdots & \cdots & \cdots & \cdots \\
X^{(h-1)d}, & X^{(h-1)d+1}, & \ldots, & X^{hd-1},
\end{array}$$

we have

$$\mathrm{vol}(L) = |\det(B)| = N^{n(h-1)/2} X^{n(n-1)/2}. \tag{110}$$

Each row of B corresponds to a polynomial $G_\ell(x)$ that vanishes at x_0 modulo N^{h-1}. Therefore, since every lattice vector is an integer linear combination of the rows of B, it follows that every lattice vector corresponds to a polynomial $G(x)$ satisfying

$$G(x_0) \equiv 0 \pmod{N^{h-1}}.$$

Applying the LLL algorithm yields a nonzero such polynomial with small coefficients.

Now, let $\tilde{B} = [\tilde{b}_1, \ldots, \tilde{b}_n]$ be an LLL-reduced basis of L. By Theorem 8.17, we have

$$\begin{aligned}
\|\tilde{b}_1\| &\le 2^{(n-1)/4} \mathrm{vol}(L)^{1/n} \\
&= 2^{(n-1)/4} N^{(h-1)/2} X^{(n-1)/2} \quad \text{by (110)} \\
&< 2^{(n-1)/4} N^{(h-1)/2} \left(2^{-1/2} N^{(h-1)/(n-1)} n^{-1/(n-1)}\right)^{(n-1)/2} \quad \text{by (109)} \\
&= N^{h-1}/\sqrt{n}.
\end{aligned} \tag{111}$$

Let $G(x)$ be the polynomial corresponding to $\tilde{b}_1$; namely, $G(x) = \sum_{i=0}^{n-1} r_i x^i$ where $\tilde{b}_1 = (r_0, r_1 X, r_2 X^2, \ldots, r_{n-1} X^{n-1})$. Then for all integers x_0 with $|x_0| \le X$, we have

$$|G(x_0)| = \left|\sum_{i=0}^{n-1} r_i x_0^i\right| \le \sum_{i=0}^{n-1} |r_i x_0^i| \le \sum_{i=0}^{n-1} |r_i| X^i. \tag{112}$$

By the Cauchy-Schwarz inequality[18], we have

$$\left|\sum_{i=0}^{n-1} |r_i| X^i\right|^2 \le n \cdot \|G(xX)\|^2,$$

where the length of a polynomial $a(x) = \sum_{i=0}^{n-1} a_i x^i$ is $\|a(x)\| = (\sum_{i=0}^{n-1} a_i^2)^{1/2}$. Taking square roots of both sides yields

$$\sum_{i=0}^{n-1} |r_i| X^i \le \sqrt{n} \cdot \|G(xX)\| = \sqrt{n}\, \|\tilde{b}_1\|. \tag{113}$$

Combining the bounds (111), (112) and (113) gives

$$|G(x_0)| \le \sum_{i=0}^{n-1} |r_i| X^i \le \sqrt{n}\, \|\tilde{b}_1\| < N^{h-1}.$$

Finally, for each x_0 satisfying $|x_0| \le X$ and $f(x_0) = 0 \pmod N$, we have $G(x_0) \equiv 0 \pmod{N^{h-1}}$. Since $|G(x_0)| < N^{h-1}$, we must have $G(x_0) = 0$.

In summary, the small roots of $f(x)$ modulo N can be found by first computing all integer roots x_0 of $G(x)$ with $|x_0| \le X$, and then checking which of them satisfy $f(x_0) \equiv 0 \pmod N$.

[18] Let $u = (|r_0|, |r_1| X, \ldots, |r_{n-1}| X^{n-1})$ and $v = (1, 1, \ldots, 1)$, and use the Cauchy-Schwarz inequality to conclude that $|\langle u, v\rangle|^2 \le \|u\|^2 \cdot \|v\|^2$.

Example 10.7 (*finding small roots of a polynomial modulo N*) Consider the degree-4 polynomial

$$f(x) = x^4 + 41383461x^3 + 67085051x^2 + 66463060x + 67200429 \in \mathbb{Z}[x].$$

We wish to determine all small roots of f modulo the composite number

$$N = 1781197896431.$$

Select $h = 2$, whereby $n = 8$ and $X = 6$. The associated lattice is $L = L(B) \subseteq \mathbb{Z}^8$ where the matrix B is given in Figure 10.2. Applying the LLL algorithm to this basis produces the

$$\left[\begin{array}{rrrr}
77123141 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\hline
0 & 462738846 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\hline
0 & 0 & 2776433076 & 0 \\
0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 16658598456 \\
0 & 0 & 0 & 0 \\
\hline
67200429 & 398778360 & 2415061836 & 8938827576 \\
1296 & 0 & 0 & 0 \\
\hline
0 & 403202574 & 2392670160 & 14490371016 \\
53632965456 & 7776 & 0 & 0 \\
\hline
0 & 0 & 2419215444 & 14356020960 \\
86942226096 & 321797792736 & 46656 & 0 \\
\hline
0 & 0 & 0 & 14515292664 \\
86136125760 & 521653356576 & 1930786756416 & 279936
\end{array}\right]$$

Figure 10.2: The 8×8 basis matrix B in Example 10.7.

reduced basis shown in Figure 10.3. Dividing the entries of the first row of $\tilde{B}$ by powers of X yields the polynomial

$$g(x) = 2x^7 + 2x^6 - 103x^5 - 9x^4 + 776x^3 - 8074x^2 + 16066x + 164520 \in \mathbb{Z}[x],$$

whose factorization over $\mathbb{Z}$ is

$$g(x) = (x - 5)(x + 4)(2x^5 + 4x^4 - 59x^3 + 12x^2 - 392x - 8226).$$

The roots of $g(x)$ in $[-X, X]$ are $x_0 = 5$ and $x_0 = -4$, which one can check are also roots of $f(x)$ modulo N.

$$\left[\begin{array}{rrrr}
164520 & 96396 & -290664 & 167616 \\
-11664 & -800928 & 93312 & 559872 \\
\hline
606720 & -80544 & -702144 & 703944 \\
-583200 & -93312 & -419904 & 0 \\
\hline
-107360 & -289248 & 125496 & 1027080 \\
-3888 & -746496 & 373248 & -559872 \\
\hline
-637180 & -519234 & 1141380 & 441288 \\
-427680 & -365472 & 186624 & 1119744 \\
\hline
172900 & -8010 & -849744 & -238464 \\
-104976 & 46656 & 1772928 & 0 \\
\hline
-557640 & -774972 & 211608 & 733752 \\
1122336 & -124416 & -279936 & 839808 \\
\hline
54621401 & -9758682 & 27377964 & -1241568 \\
15505344 & 2146176 & 9611136 & 2799360 \\
\hline
-6042579 & 288967410 & 44872020 & 168121008 \\
52578720 & 101772288 & 45442944 & 64665216
\end{array}\right]$$

Figure 10.3: The LLL-reduced basis $\tilde{B}$ in Example 10.7.

Analysis. The overall complexity of the Coppersmith/Howgrave-Graham method is governed primarily by the cost of the LLL reduction step. For a lattice $L \subseteq \mathbb{Z}^n$ with basis $B = [b_1, \ldots, b_n]$, the LLL algorithm computes an LLL-reduced basis in time $O(n^6 \log^3 Y)$ where $Y = \max_i \|b_i\|$ (Theorem 8.30). By reducing the coefficients of $f(x)$ modulo N, we may assume without loss of generality that they lie in the range $[0, N-1]$. It follows that

$$\max_{\ell} \|G_\ell(xX)\| = \|G_n(xX)\| \leq d^{h-1} N^h,$$

which implies that $\log Y \leq h \log N$. Consequently, the running time of the Coppersmith/Howgrave-Graham method is

$$O(d^6 h^9 \log^3 N). \tag{114}$$

Recall that $X \to N^{1/d}/\sqrt{2}$ as $h \to \infty$. Increasing h is therefore beneficial, as it enlarges the bound X within which solutions may be found. However, this comes at the cost of a rapidly increasing running time, as indicated by (114). In practice, one typically starts with $h = 2$ and incrementally increases h until the sought-after solution is found.

Remark 10.8 The Coppersmith/Howgrave-Graham method can be refined to find all the roots x_0 with $|x_0| \leq N^{1/d}$ of a degree-d polynomial $f(x)$ over $\mathbb{Z}_N$ in polynomial time.

10.6.2 RSA encryption with fixed padding

When utilizing RSA with a small public exponent e, one must be careful when encrypting short messages. Consider the case of encryption exponent $e = 3$. Suppose the scheme is used to encrypt a symmetric key K whose bitlength is much smaller than that of the RSA modulus N. For example, N might be 2048 bits long whereas K is only 256 bits. The resulting ciphertext is $c = K^3 \bmod N$. But since $K^3 < N$, reduction modulo N does not occur, so $c = K^3$. Consequently, K can be recovered simply by computing the integer cube root of c.

To mitigate this weakness, suppose that K is padded on the left with a string of bits, all of which are 1, to form a message m whose bitlength is just one less than that of N. The ciphertext is then $c = m^3 \bmod N$.

However, if the bitlength of K is less than one-third that of N, the key can still be efficiently recovered from c. Writing $m = P + K$, we have $c = (P + K)^3 \bmod N$. Thus, K is a root of the polynomial

$$f(x) = x^3 + 3Px^2 + 3P^2x + P^3 - c \pmod{N}.$$

Since $K < N^{1/3}$, it can be recovered using the Coppersmith/Howgrave-Graham method.

10.6.3 RSA encryption with random padding

To achieve security against chosen-ciphertext attacks, a plaintext message should be randomized prior to applying the RSA encryption functions. One way to introduce this randomness is through random padding. The following example illustrates an attack on RSA with random right-padding.

Let N be an RSA modulus and let $e = 3$ be the encryption exponent. Suppose a message M is encrypted by first padding it on the right with a randomly selected bitstring R of length k. Thus, the ciphertext corresponding to M is $c = m^3 \bmod N$ where $m = 2^k M + R$.

Now suppose a second encryption of the same message M is performed using independent randomness $R' \in_R \{0,1\}^k$, yielding $m' = 2^k M + R'$ and $c' = (m')^3 \bmod N$. Assume an adversary obtains N, c and c'. We show that the adversary can efficiently recover M.

Let $r = R' - R$, so that $m' = m + r$. Consider the bivariate polynomials

$$f_1(x, y) = y^3 - c \text{ and } f_2(x, y) = (x + y)^3 - c' \in \mathbb{Z}_N[x, y].$$

The resultant of f_1 and f_2 with respect to y, denoted $\mathrm{Res}_y(f_1, f_2)$, is a degree-9 polynomial in $\mathbb{Z}_N[x]$:

$$f(x) = \mathrm{Res}_y(f_1, f_2) = x^9 + (3c - 3c')x^6 + (3c^2 + 21cc' + 3(c')^2)x^3 + (c - c')^3.$$

The resultant has the following property: if, for some $\alpha \in \mathbb{Z}_N$, the univariate polynomials $f_1(\alpha, y)$ and $f_2(\alpha, y)$ have a common root in $\mathbb{Z}_N$, then $f(\alpha) = 0$. Now, $f_1(r, y)$ and

$f_2(r, y)$ share the root $y = m$ and hence $f(r) \equiv 0 \pmod{N}$. If $|r| < N^{1/9}$, then the Coppersmith/Howgrave-Graham method can be used to efficiently determine r. Once r is known, one can easily recover m and m', and hence M (Exercise 10.11).

10.7 Exercises

Exercise 10.1 Let $m = 6$, $n = 4$, $q = 98947$, and $B = 300$. Use the primal attack to solve the ss-LWE(m, n, q, B) instance (A, t) where

$$A = \begin{bmatrix} 24487 & 21871 & 9749 & 28102 \\ 90929 & 26089 & 60954 & 46363 \\ 87262 & 40841 & 30905 & 55441 \\ 8122 & 70592 & 74652 & 26958 \\ 83978 & 14387 & 91339 & 17846 \\ 27139 & 89719 & 90428 & 81851 \end{bmatrix} \quad \text{and} \quad t = \begin{bmatrix} 32578 \\ 45126 \\ 74973 \\ 32418 \\ 89962 \\ 6941 \end{bmatrix}.$$

Exercise 10.2 Consider the polynomial ring $R_q = \mathbb{Z}_{97}[x]/(x^4 + 1)$. Let $k = 4$, $\ell = 3$ and $B = 2$. Use the primal attack to solve the ss-MLWE instance (A, t) where:

$$A = \begin{bmatrix} 77 + 96x + 10x^2 + 86x^3 & 58 + 36x + 80x^2 + 22x^3 & 44 + 39x + 60x^2 + 39x^3 \\ 43 + 12x + 55x^2 + 2x^3 & 24 + 71x + 45x^2 + 29x^3 & 21 + 48x + 7x^2 + 33x^3 \\ 57 + 65x + 16x^2 + 93x^3 & 96 + 71x + 44x^2 + 70x^3 & 58 + 25x + 29x^2 + 73x^3 \\ 57 + 40x + 78x^2 + 88x^3 & 25 + 44x + 67x^2 + 88x^3 & 5 + 83x + 90x^2 + 33x^3 \end{bmatrix} \in R_q^{4\times 3}$$

and

$$t = \begin{bmatrix} 31 + 38x + 7x^2 + 71x^3 \\ 95 + 55x + 12x^2 + 87x^3 \\ 79 + 34x + 6x^2 + 34x^3 \\ 17 + 39x + 74x^2 + 43x^3 \end{bmatrix} \in R_q^4.$$

Exercise 10.3 Let $n = 7$, $m = 14$, $q = 997$, and $B = 35$. Use the dual attack to solve the following $\mathrm{SIS}_2(n, m, q, B)$ instance:

$$A = \begin{bmatrix} 301 & 352 & 400 & 269 & 585 & 916 & 438 & 560 & 510 & 108 & 14 & 157 & 265 & 620 \\ 886 & 348 & 376 & 178 & 581 & 111 & 717 & 789 & 212 & 238 & 542 & 288 & 585 & 272 \\ 751 & 758 & 760 & 641 & 438 & 400 & 702 & 205 & 81 & 615 & 816 & 455 & 237 & 521 \\ 28 & 4 & 376 & 755 & 213 & 145 & 375 & 559 & 122 & 538 & 756 & 659 & 453 & 281 \\ 474 & 349 & 139 & 507 & 8 & 223 & 584 & 948 & 45 & 124 & 640 & 843 & 985 & 292 \\ 499 & 740 & 268 & 274 & 195 & 834 & 897 & 835 & 443 & 686 & 337 & 139 & 414 & 162 \\ 352 & 523 & 981 & 765 & 540 & 403 & 473 & 920 & 724 & 836 & 523 & 663 & 277 & 194 \end{bmatrix}.$$

Exercise 10.4 Consider the polynomial ring $R_q = \mathbb{Z}_{97}[x]/(x^5+1)$. Let $k = 3$, $\ell = 4$ and $B = 50$. Use the dual attack to solve the following MSIS_2 instance:

$$A = \left[\begin{array}{cc} 77+96x+10x^2+86x^3+58x^4 & 36+80x+22x^2+44x^3+39x^4 \\ 60+39x+43x^2+12x^3+55x^4 & 2+24x+71x^2+45x^3+29x^4 \\ \hline 21+48x+7x^2+33x^3+57x^4 & 65+16x+93x^2+96x^3+71x^4 \\ 44+70x+58x^2+25x^3+29x^4 & 73+57x+40x^2+78x^3+88x^4 \\ \hline 25+44x+67x^2+88x^3+5x^4 & 83+90x+33x^2+88x^3+56x^4 \\ 81+85x+3x^2+65x^3+36x^4 & 52+28x+21x^2+51x^3+68x^4 \end{array}\right] \in R_q^{3\times 4}.$$

Note: The size of a polynomial $a = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} \in \mathbb{Z}_q[x]/(x^n+1)$ is $\|a\|_2 = \sqrt{a_0^2 + a_1^2 + \cdots + a_{n-1}^2}$. The size of a polynomial vector $b = (b_1, b_2, \ldots, b_k) \in R_q^k$ is $\|b\|_2 = \sqrt{\|b_1\|_2^2 + \|b_2\|_2^2 + \cdots + \|b_k\|_2^2}$.

Exercise 10.5 Let $L = L(B) \subseteq \mathbb{Z}^n$ be a full-rank integer lattice, and let $t \in \mathbb{Z}^n$ be a target vector. Babai's *rounding algorithm* for approximate CVP proceeds as follows. First, compute an LLL-reduced basis $\tilde{B} = [\tilde{b}_1, \ldots, \tilde{b}_n]$ of L. Next, express $t = \sum_{i=1}^n t_i \tilde{b}_i$ with $t_i \in \mathbb{R}$. The algorithm then outputs the lattice vector $x = \sum_{i=1}^n \lfloor t_i \rceil \tilde{b}_i$.
(a) Suppose that L is given with an orthogonal basis. Show that in this case, Babai's rounding algorithm solves CVP exactly, i.e., with approximation factor $\gamma = 1$.
(b) For each of the following pairs (B, t), apply Babai's rounding algorithm to find a lattice vector $x \in L(B)$ that is close to t.
(i)

$$B = \begin{bmatrix} -84 & 7 & 41 & -80 \\ 85 & -57 & -23 & 36 \\ -4 & 86 & -53 & -80 \\ 81 & -69 & 1 & 57 \end{bmatrix}, \quad t = \begin{bmatrix} -1576 \\ 1520 \\ -1276 \\ 1718 \end{bmatrix}.$$

(ii)

$$B = \begin{bmatrix} 10 & 14 & 14 & -81 & 93 \\ 72 & 43 & 71 & -61 & 39 \\ -30 & -35 & 24 & 6 & 29 \\ -49 & 25 & 36 & 36 & -83 \\ -22 & -34 & 29 & 64 & -27 \end{bmatrix}, \quad t = \begin{bmatrix} 275 \\ 209 \\ 26 \\ 204 \\ -311 \end{bmatrix}.$$

(iii)

$$B = \begin{bmatrix} 2 & -14 & -83 & 99 & 49 & -11 \\ -15 & 87 & 73 & 76 & 94 & -56 \\ 95 & -28 & 88 & 16 & -41 & -92 \\ -88 & -19 & 83 & 2 & -36 & -49 \\ -35 & -59 & 48 & 1 & 15 & -32 \\ -11 & -43 & 47 & 50 & 22 & -32 \end{bmatrix}, \quad t = \begin{bmatrix} -608 \\ -1323 \\ 686 \\ -809 \\ -408 \\ -530 \end{bmatrix}.$$

Exercise 10.6 (*Kannan's embedding technique*) Let $L = L(B) \subseteq \mathbb{Z}^n$ be a full-rank integer lattice, and let $t \in \mathbb{Z}^n$ be a target vector. Let $v \in L$ be a lattice vector that is closest to t, and define $d = v - t$. Let $Q = \|d\|$ and suppose that $Q < \lambda_1(L)/\sqrt{2}$.
(a) Prove that (d, Q) is a shortest nonzero vector in the lattice $L' \subseteq \mathbb{Z}^{n+1}$ with basis matrix

$$B' = \begin{bmatrix} B & -t \\ 0 & Q \end{bmatrix}.$$

(b) The result in (a) suggests the following algorithm which, given B and t, attempts to find a vector $v \in L(B)$ that is close to t.
For $M = 1, 2, 3, \ldots$, let L_M be the lattice with basis matrix

$$B_M = \begin{bmatrix} B & -t \\ 0 & M \end{bmatrix}.$$

Apply the LLL algorithm to find a reduced basis $\tilde{B}_M$ for L_M. If $\tilde{B}_M$ has a vector of the form (d, M), then output $v = d + t$ (and stop).
Show that $v \in L$, and explain why v can be expected to be close to t when the algorithm terminates.
(c) For the (B, t) pairs listed in Exercise 10.5(b), use the method in (b) to find a vector $v \in L(B)$ that is close to t. Compare your results with the vectors found using Babai's rounding method.

Exercise 10.7 Use the lattice method described in §10.3 to "discover" the following known Machin-like formulae. You will need to experiment with increasing values of scaling factor Q and precision for the real number approximations.
(a) $\frac{\pi}{4} = 4\arctan\left(\frac{1}{5}\right) - \arctan\left(\frac{1}{239}\right)$.
(b) $\frac{\pi}{4} = \arctan\left(\frac{1}{2}\right) + \arctan\left(\frac{1}{5}\right) + \arctan\left(\frac{1}{8}\right)$.
(c) $\frac{\pi}{4} = 12\arctan\left(\frac{1}{38}\right) + 20\arctan\left(\frac{1}{57}\right) + 7\arctan\left(\frac{1}{239}\right) + 24\arctan\left(\frac{1}{268}\right)$.

Exercise 10.8 A real number α is said to be *algebraic* if there exists a nonzero polynomial $f(x) \in \mathbb{Z}[x]$ such that $f(\alpha) = 0$. Such a polynomial of degree n immediately yields an integer relation for $\{1, \alpha, \alpha^2, \ldots, \alpha^n\}$. For the following algebraic numbers α, use the lattice method described in §10.3 to find a polynomial $f(x) \in \mathbb{Z}[x]$ of degree n such that $f(\alpha) = 0$. You will need to experiment with increasing values of scaling factor Q and precision for the real number approximations.
(a) $\alpha = \sqrt{2}, \quad n = 2$.
(b) $\alpha = \sqrt{2} + \sqrt{3}, \quad n = 4$.
(c) $\alpha = \sqrt{2} + \sqrt{3} + \sqrt{5}, \quad n = 8$.
(d) $\alpha = \sqrt{2} + \sqrt[3]{3}, \quad n = 6$.
(e) $\alpha = \sqrt{2} + \sqrt[5]{3}, \quad n = 10$.

Exercise 10.9 Use the LLL algorithm to find small rational approximations with common denominator for the following lists of irrational numbers (see §10.4).

(a) $\sqrt{2}$, $\sqrt{3}$, $\sqrt{5}$, $\sqrt{7}$, $\sqrt{11}$ with $\varepsilon = \frac{1}{2}$.
(b) $\sqrt{2}$, $\sqrt{3}$, $\sqrt{5}$, $\sqrt{7}$, $\sqrt{11}$, $\sqrt{13}$, $\sqrt{17}$, $\sqrt{19}$, $\sqrt{23}$, $\sqrt{29}$ with $\varepsilon = \frac{2}{3}$.

Exercise 10.10 Consider the RSA encryption scheme with public exponent $e = 3$ and 128-bit RSA modulus

$$N = 340282360475865017927401647640365095477.$$

A 25-bit symmetric key K is encrypted as follows: first, K is padded on the left with 102 bits, all equal to 1, and the resulting integer is then encrypted using RSA. The resulting ciphertext is

$$c = 176903397640877314846558520499830023827.$$

Determine K.

Exercise 10.11 Let N be an RSA modulus and let the public exponent be $e = 3$. Consider two plaintext messages $m_1, m_2 \in [0, N-1]$, and define $r = m_2 - m_1$. Their corresponding ciphertexts are $c_1 = m_1^3 \bmod N$ and $c_2 = m_2^3 \bmod N$. Show that

$$m_1 = \frac{r(c_2 + 2c_1 - r^3)}{c_2 - c_1 + 2r^3} \bmod N.$$

Deduce that both m_1 and m_2 can be efficiently recovered given N, c_1, c_2 and r.

Exercise 10.12 Consider the RSA encryption scheme with public exponent $e = 3$ and the following 2048-bit RSA modulus N:

```
17976931348623159077293051907890247336179769789423065727343008115773267580550096312228546417590770834220379891873327844190344217453348003906629230956773116093836340601039326663645086095468006803268162101021499409405174265966496695312077337304231298404292169466823984779491404759726124807446005014176719061778 9.
```

A plaintext M is encrypted by first padding it on the right with a randomly selected bitstring of length 64, and then applying the RSA encryption function. Suppose that a plaintext M is encrypted twice, yielding the following two ciphertexts:

```
97281555518276697162604105158477538979439741228077012917326189769539364253679939979863686525056475054455510161581583470636525831513643948888278928335024922353272601990949151203823324665374417549676879637832690366633662476699456248810119474827328559120921420121209593874882619562446563958936994737739896921 7
```

and

```
94681391986701340753942638430259451175338345862013952535356696085293122953024485192589633028180170107195791328142154763072668832677175471604014741884284842084865632439460420479366769578931101173863440487761806293366293583044071817359017533690507319378098728610644764061643469188016001601755692567007555924679.
```

Determine M.

Exercise 10.13 The *subset sum problem* is the following. Fix positive integers k and n. Select $a_i \in_R [1, 2^k - 1]$ and $e_i \in_R \{0, 1\}$ for $1 \leq i \leq n$, and define $s = \sum_{i=1}^{n} e_i a_i$. Given $a = (a_1, \ldots, a_n)$ and s, the problem is to determine $x_1, \ldots, x_n \in \{0, 1\}$ such that $\sum_{i=1}^{n} a_i x_i = s$.
Note: The *density* of a is $d = n/k$. If $d \ll 1$, then one expects that the subset sum problem has a unique solution, namely $e = (e_1, \ldots, e_n)$.
Suppose now that $d < 0.6463$. Then the subset sum problem can be solved (with high probability) using the lattice-based method outlined below.
We will suppose, without loss of generality, that $0 < \sum_{i=1}^{n} e_i \leq \frac{1}{2}n$. Select an integer $N > \sqrt{n}$, and define the $(n+1)$-dimensional integer lattice $L = L(B)$ with basis matrix

$$B = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ Na_1 & Na_2 & Na_3 & \cdots & Na_n & Ns \end{bmatrix}_{(n+1)\times(n+1)}.$$

(a) Let $v = (e_1, \ldots, e_n, 0)$. Prove that $v \in L$ and $\|v\| \leq \sqrt{n/2}$.
(b) Using the Gaussian Heuristic, give an informal argument that v can be expected to be a lattice vector of shortest nonzero length. Hence, one can expect that v appears as a basis vector in an LLL-reduced basis for L.
(c) Use this LLL-based method to solve the following instances of subset sum:
(i) $a = (491761, 618427, 742119, 526658, 100492, 607921, 945509, 570458, 654371, 381711)$, $s = 1899948$.
(ii) $a = (551012, 435750, 764101, 642331, 28000, 882705, 994024, 530149, 192346, 711419)$, $s = 2584681$.

Exercise 10.14 Let p be an odd prime. Lagrange's Theorem states that p can be written as a sum of four squares (and thus, so can every positive integer). This exercise develops an efficient method to compute this representation.
(a) Prove that there exist integers $u, v \in [0, p-1]$ such that $u^2 + v^2 \equiv -1 \pmod{p}$.
Hint: Show that the sets $U = \{u^2 + 1 : u \in \mathbb{Z}_p\}$ and $V = \{v^2 : v \in \mathbb{Z}_p\}$ have a non-trivial intersection.
(b) Let $u, v \in [0, p-1]$ with $u^2 + v^2 \equiv -1 \pmod{p}$. Consider the lattice $L \subseteq \mathbb{Z}^4$ with basis matrix

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ u & v & p & 0 \\ -v & u & 0 & p \end{bmatrix}.$$

Prove that every lattice vector $w \in L$ satisfies $\|w\|^2 \equiv 0 \pmod{p}$.
(c) Let $\tilde{B} = \{\tilde{b}_1, \tilde{b}_2, \tilde{b}_3, \tilde{b}_4\}$ be an $\frac{8}{9}$-LLL-reduced basis for B. Use (71) to prove that $\|\tilde{b}_1\|^2 = p$.
(d) Describe and analyze a polynomial-time algorithm for expressing a prime p as a sum

of four squares.
(e) Write each of the following primes as the sum of four squares.
(i) $p = 7919 \quad (u = 4, v = 3211)$.
(ii) $p = 59359 \quad (u = 4, v = 41632)$.
(iii) $p = 104743 \quad (u = 2, v = 59657)$.

11 The Number-Theoretic Transform

Contents

The Number-Theoretic Transform (NTT) is a finite-field analogue of the Discrete Fourier Transform (DFT). It enables the multiplication of two polynomials in the ring $R_q = \mathbb{Z}_q[x]/(x^n+1)$ using $O(n \log n)$ $\mathbb{Z}_q$-operations, in contrast to the $O(n^2)$ $\mathbb{Z}_q$-operations required by naive polynomial multiplication. The NTT is used in both Kyber and Dilithium to accelerate polynomial multiplication in R_q, which is the dominant operation in those schemes. The NTT is introduced in §11.1 and §11.2, while the specific NTT variants used in Dilithium and Kyber are described in §11.3 and §11.4, respectively.

11.1 NTT definition

Let q be a prime number and let $n \geq 2$ be an integer. Denote by $\mathbb{Z}_q[x]$ the set of all polynomials in the indeterminate x with coefficients in $\mathbb{Z}_q$. The quotient ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ consists of all polynomials in $\mathbb{Z}_q[x]$ of degree at most $n-1$, with multiplication performed modulo the polynomial x^n+1. A polynomial $a(x) = a_0+a_1x+\cdots+a_{n-1}x^{n-1} \in R_q$ can be identified with its coefficient vector $(a_0, a_1, \ldots, a_{n-1})$, giving a natural one-to-one correspondence between R_q and the vector space $\mathbb{Z}_q^n$.

Classical polynomial multiplication. Let $a(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1}$ and $b(x) = b_0 + b_1x + \cdots + b_{n-1}x^{n-1}$ be elements of R_q. The classical method for computing their product $c(x) = a(x)b(x) \bmod (x^n + 1)$ in R_q proceeds as follows:

1. Compute the ordinary polynomial product in $\mathbb{Z}_q[x]$:
$$d(x) = a(x) \cdot b(x) = d_0 + d_1x + \cdots + d_{2n-2}x^{2n-2}.$$
2. Reduce $d(x)$ modulo $x^n + 1$ by substituting $x^n = -1$, $x^{n+1} = -x$, …, and $x^{2n-2} = -x^{n-2}$, yielding
$$(d_0 + d_1x + \cdots + d_{n-1}x^{n-1}) + (-d_n - d_{n+1}x - \cdots - d_{2n-2}x^{n-2}).$$
3. Add the two polynomials above to obtain $c(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$.

Steps 1 and 3 require $O(n^2)$ and $O(n)$ arithmetic operations[19], respectively, for an overall running time of $O(n^2)$. In contrast, the NTT allows multiplication in R_q to be carried out in time $O(n \log n)$.

[19] An arithmetic operation is an addition, subtraction, multiplication, or division in $\mathbb{Z}_q$.

Definition 11.1 Let $n = 2^k$, and let q be a prime such that $q - 1$ is divisible by $2n$. Let $\zeta \in \mathbb{Z}_q^*$ be an element of order[20] $2n$, so that $\zeta^n = -1$. The *Number-Theoretic Transform* (NTT) is the map $\text{NTT} : R_q \longrightarrow \mathbb{Z}_q^n$ defined by

$$\text{NTT}(a) = (a(\zeta), a(\zeta^3), a(\zeta^5), \ldots, a(\zeta^{2n-1})).$$

For notational convenience, we will often write $\hat{a}$ in place of $\text{NTT}(a)$.

The NTT of $a(x)$ evaluates the polynomial at the odd powers of ζ. These odd powers are precisely the elements of order $2n$ in $\mathbb{Z}_q^*$, also known as the primitive $(2n)$-th roots of unity modulo q. A key property of the NTT is that it is invertible; its inverse function $\text{NTT}^{-1} : \mathbb{Z}_q^n \longrightarrow R_q$ can be interpreted as polynomial interpolation. Consequently, $\text{NTT}^{-1}(\text{NTT}(a)) = a$ for all $a \in R_q$.

Addition and subtraction in R_q. Let $a(x), b(x) \in R_q$ and let $c(x) = a(x) \pm b(x)$. For any odd integer i, we have

$$c(\zeta^i) = a(\zeta^i) \pm b(\zeta^i),$$

and so $\hat{c} = \text{NTT}(c) = \hat{a} \pm \hat{b}$. Thus, addition and subtraction in NTT form is componentwise.

Fast multiplication in R_q. Let $a(x), b(x) \in R_q$ and let $c(x)$ be their product in R_q, i.e., $c(x) = a(x)b(x) \bmod (x^n + 1)$. There exists a polynomial $\ell(x) \in \mathbb{Z}_q[x]$ such that

$$c(x) = a(x)b(x) + \ell(x)(x^n + 1).$$

For any odd integer i, we have

$$c(\zeta^i) = a(\zeta^i)b(\zeta^i) + \ell(\zeta^i)(\zeta^{in} + 1) = a(\zeta^i)b(\zeta^i),$$

since $\zeta^{in} = (\zeta^n)^i = (-1)^i = -1$. Therefore

$$\text{NTT}(c) = \hat{c} = (a(\zeta)b(\zeta),\ a(\zeta^3)b(\zeta^3),\ \ldots,\ a(\zeta^{2n-1})b(\zeta^{2n-1})) = \hat{a} \circ \hat{b},$$

where $\hat{a} \circ \hat{b}$ denotes the componentwise multiplication of vectors in $\mathbb{Z}_q^n$.

In §11.2, we show that both NTT and NTT^{-1} can be computed in time $O(n \log n)$. This leads to an alternative procedure for multiplying $a(x)$ and $b(x)$ in R_q:

1. Compute $\hat{a} = \text{NTT}(a)$ and $\hat{b} = \text{NTT}(b)$.
2. Compute the componentwise product $\hat{c} = \hat{a} \circ \hat{b}$.
3. Recover $c = \text{NTT}^{-1}(\hat{c})$.

Since the componentwise multiplication requires only $O(n)$ operations, the overall running time of this method is $O(n \log n)$, a substantial improvement over the $O(n^2)$ running time of the classical method.

[20] The *order* of a nonzero element α in $\mathbb{Z}_q$ is the smallest positive integer t such that $\alpha^t \equiv 1 \pmod{q}$. For each positive divisor t of $q - 1$, there exists an element of order t in $\mathbb{Z}_q^*$.

11.2 NTT computation

Recall that $\mathrm{NTT}(a) = (a(\zeta), a(\zeta^3), \ldots, a(\zeta^{2n-1}))$. We can view the value $a(\zeta^i)$ as the remainder obtained by dividing $a(x)$ by the linear polynomial $x - \zeta^i$. The algorithm for computing the NTT uses the following result to recursively compute these remainders by first reducing modulo higher-degree factors.

Theorem 11.2 Let a, f, g be polynomials in $\mathbb{Z}_q[x]$. If f divides g, then

$$a(x) \bmod f(x) = (a(x) \bmod g(x)) \bmod f(x).$$

In other words, the remainder upon dividing $a(x)$ by $f(x)$ can be obtained by first dividing $a(x)$ by $g(x)$, and then dividing the resulting remainder by $f(x)$.

NTT computation for $n = 8$. To illustrate the algorithm, consider the case $n = 8$. (Generalizing the algorithm to arbitrary $n = 2^k$ is straightforward.) Let q be a prime such that $q - 1$ is divisible by $2n = 16$, and let $\zeta \in \mathbb{Z}_q^*$ be an element of order 16. Then $\zeta^8 = -1$ and the polynomial ring of interest is

$$R_q = \mathbb{Z}_q[x]/(x^8 + 1).$$

The NTT computation is based on the identities (115), (116) and (117) derived as follows:

$$\begin{aligned}
x^8 + 1 &= x^8 - \zeta^8 \\
&= (x^4 - \zeta^4)(x^4 + \zeta^4) && (115)\\
&= (x^4 - \zeta^4)(x^4 - \zeta^{12}) \\
&= (x^2 - \zeta^2)(x^2 + \zeta^2)(x^2 - \zeta^6)(x^2 + \zeta^6) && (116)\\
&= (x^2 - \zeta^2)(x^2 - \zeta^{10})(x^2 - \zeta^6)(x^2 - \zeta^{14}) \\
&= (x - \zeta)(x + \zeta)(x - \zeta^5)(x + \zeta^5)(x - \zeta^3)(x + \zeta^3)(x - \zeta^7)(x + \zeta^7). && (117)
\end{aligned}$$

Let $a \in R_q$, so $a(x)$ is a polynomial in $\mathbb{Z}_q[x]$ of degree ≤ 7. The computation $\mathrm{NTT}(a)$ proceeds recursively down the levels of the *NTT computation tree* shown in Figure 11.1. More specifically, one first computes

$$a_0 = a \bmod (x^4 - \zeta^4) \quad \text{and} \quad a_1 = a \bmod (x^4 + \zeta^4), \tag{118}$$

then

$$a_{00} = a_0 \bmod (x^2 - \zeta^2), \quad a_{01} = a_0 \bmod (x^2 + \zeta^2), \tag{119}$$
$$a_{10} = a_1 \bmod (x^2 - \zeta^6), \quad a_{11} = a_1 \bmod (x^2 + \zeta^6), \tag{120}$$

and finally

$$a_{000} = a_{00} \bmod (x - \zeta), \quad a_{001} = a_{00} \bmod (x + \zeta), \tag{121}$$
$$a_{010} = a_{01} \bmod (x - \zeta^5), \quad a_{011} = a_{01} \bmod (x + \zeta^5), \tag{122}$$
$$a_{100} = a_{10} \bmod (x - \zeta^3), \quad a_{101} = a_{10} \bmod (x + \zeta^3), \tag{123}$$
$$a_{110} = a_{11} \bmod (x - \zeta^7), \quad a_{111} = a_{11} \bmod (x + \zeta^7). \tag{124}$$

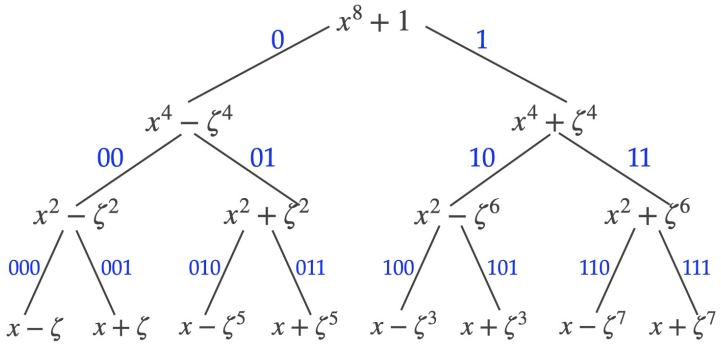


Figure 11.1: The NTT computation tree ($n = 8$).

The subscripts i in the polynomials a_i correspond to the labellings of the branches in Figure 11.1.

The polynomials a_i are computed as follows. Let $a_L(x)$ and $a_R(x)$ denote the left and right portions of $a(x)$. More precisely, we have $a(x) = a_L(x) + a_R(x)x^4$ where $a_L(x), a_R(x)$ each have degree ≤ 3. Then

$$a_0(x) = a(x) \bmod (x^4 - \zeta^4) = a_L(x) + \zeta^4 a_R(x), \tag{125}$$

and

$$a_1(x) = a(x) \bmod (x^4 + \zeta^4) = a_L(x) - \zeta^4 a_R(x), \tag{126}$$

so $a_0(x)$ and $a_1(x)$ can be computed simply by multiplying the right half $a_R(x)$ by ζ^4 and adding/subtracting the result to/from $a_L(x)$.

Similarly, writing $a_0(x) = a_{0L}(x) + a_{0R}(x)x^2$ and $a_1(x) = a_{1L}(x) + a_{1R}(x)x^2$ where $a_{0L}, a_{0R}, a_{1L}, a_{1R}$ each have degree ≤ 1, we have

$$a_{00}(x) = a_0(x) \bmod (x^2 - \zeta^2) = a_{0L}(x) + \zeta^2 a_{0R}(x), \tag{127}$$
$$a_{01}(x) = a_0(x) \bmod (x^2 + \zeta^2) = a_{0L}(x) - \zeta^2 a_{0R}(x), \tag{128}$$
$$a_{10}(x) = a_1(x) \bmod (x^2 - \zeta^6) = a_{1L}(x) + \zeta^6 a_{1R}(x), \tag{129}$$
$$a_{11}(x) = a_1(x) \bmod (x^2 + \zeta^6) = a_{1L}(x) - \zeta^6 a_{1R}(x). \tag{130}$$

Finally, writing $a_{00}(x) = a_{00L} + a_{00R}x$, $a_{01}(x) = a_{01L} + a_{01R}x$, $a_{10}(x) = a_{10L} + a_{10R}x$, and $a_{11}(x) = a_{11L} + a_{11R}x$, where each a_{ijL} and a_{ijR} is in $\mathbb{Z}_q$, we have

$$a_{000} = a_{00L} + \zeta a_{00R}, \quad a_{001} = a_{00L} - \zeta a_{00R}, \tag{131}$$
$$a_{010} = a_{01L} + \zeta^5 a_{01R}, \quad a_{011} = a_{01L} - \zeta^5 a_{01R}, \tag{132}$$
$$a_{100} = a_{10L} + \zeta^3 a_{10R}, \quad a_{101} = a_{10L} - \zeta^3 a_{10R}, \tag{133}$$
$$a_{110} = a_{11L} + \zeta^7 a_{11R}, \quad a_{111} = a_{11L} - \zeta^7 a_{11R}. \tag{134}$$

Thus,

$$\mathrm{NTT}(a) = (a_{000}, a_{100}, a_{010}, a_{110}, a_{001}, a_{101}, a_{011}, a_{111}). \tag{135}$$

NTT running time. Let $n = 2^k$, and let $T(n)$ denote the time required to evaluate NTT. We have the recursive formula

$$T(n) = 2T(n/2) + 3n/2.$$

Indeed, at each level of the NTT computation tree, a total of $\frac{n}{2}$ multiplications and n additions are required to carry out the reduction steps. Moving to the next level doubles the number of reduction steps, but each reduction operates on polynomials of half the degree. Solving this recurrence yields $T(n) = O(n \log n)$, which is the running time of the NTT algorithm.

NTT^{-1} computation for $n = 8$. The inverse NTT is computed by traversing the NTT computation tree in reverse, starting from the leaves and working back to the root. Thus, given

$$\text{NTT}(a) = (a_{000}, a_{100}, a_{010}, a_{110}, a_{001}, a_{101}, a_{011}, a_{111}),$$

we first combine a_{000} and a_{001} to recover $a_{00}(x)$, then combine a_{010} and a_{011} to recover $a_{01}(x)$, and so on, until the original polynomial $a(x)$ is reconstructed from $a_0(x)$ and $a_1(x)$.

As an illustration, recall that $a_0(x) = a_L(x) + \zeta^4 a_R(x)$ and $a_1(x) = a_L(x) - \zeta^4 a_R(x)$. Solving for $a_L(x)$ and $a_R(x)$ gives

$$a_L(x) = \frac{a_0(x) + a_1(x)}{2} \quad \text{and} \quad a_R(x) = \frac{a_0(x) - a_1(x)}{2\zeta^4},$$

and hence $a(x) = a_L(x) + a_R(x)x^4$ can be recovered from $a_0(x)$ and $a_1(x)$. Applying this procedure recursively yields the full set of inverse NTT formulas, which are listed below. For notational simplicity, the argument "(x)" is omitted.

$$\begin{aligned}
a_{00} &= a_{00L} + a_{00R}x, \quad \text{where } a_{00L} = (a_{000} + a_{001})/2 \text{ and } a_{00R} = (a_{000} - a_{001})/(2\zeta),\\
a_{01} &= a_{01L} + a_{01R}x, \quad \text{where } a_{01L} = (a_{010} + a_{011})/2 \text{ and } a_{01R} = (a_{010} - a_{011})/(2\zeta^5),\\
a_{10} &= a_{10L} + a_{10R}x, \quad \text{where } a_{10L} = (a_{100} + a_{101})/2 \text{ and } a_{10R} = (a_{100} - a_{101})/(2\zeta^3),\\
a_{11} &= a_{11L} + a_{11R}x, \quad \text{where } a_{11L} = (a_{110} + a_{111})/2 \text{ and } a_{11R} = (a_{110} - a_{111})/(2\zeta^7),\\
a_0 &= a_{0L} + a_{0R}x^2, \quad \text{where } a_{0L} = (a_{00} + a_{01})/2 \text{ and } a_{0R} = (a_{00} - a_{01})/(2\zeta^2),\\
a_1 &= a_{1L} + a_{1R}x^2, \quad \text{where } a_{1L} = (a_{10} + a_{11})/2 \text{ and } a_{1R} = (a_{10} - a_{11})/(2\zeta^6),\\
a &= a_L + a_R x^4, \quad \text{where } a_L = (a_0 + a_1)/2 \text{ and } a_R = (a_0 - a_1)/(2\zeta^4).
\end{aligned}$$

The inverse NTT map can also be computed in time $O(n \log n)$.

Example 11.3 (*NTT computation*) Consider $q = 241$ and $n = 8$, whence the polynomial ring of interest is $R_q = \mathbb{Z}_{241}[x]/(x^8 + 1)$. Note that $2n = 16$ divides $q - 1 = 240 = 16 \times 15$. The element $\zeta = 76 \in \mathbb{Z}_{241}^*$ has order 16. Let

$$a(x) = 71 + 152x + 4x^3 + 74x^5 + 133x^6 + 200x^7 \in R_q$$

and

$$b(x) = 199 + x^2 + x^3 + 11x^4 + 44x^5 + 165x^6 + 114x^7 \in R_q.$$

We wish to compute $c(x) = a(x)b(x) \bmod (x^8 + 1)$ using NTT.

Let us first compute $\hat{a} = \mathrm{NTT}(a)$. We have

$$a_L(x) = 71 + 152x + 4x^3 \quad \text{and} \quad a_R(x) = 74x + 133x^2 + 200x^3.$$

We compute

$$a_0(x) = a_L(x) + \zeta^4 a_R(x) = 71 + 68x + 77x^2 + 31x^3$$

and

$$a_1(x) = a_L(x) - \zeta^4 a_R(x) = 71 + 236x + 164x^2 + 218x^3,$$

then

$$a_{00}(x) = 178 + 61x,\ a_{01}(x) = 205 + 75x,\ a_{10}(x) = 212 + 203x,\ a_{11}(x) = 171 + 28x,$$

and finally

$$\begin{aligned} a_{000}(x) = 235,\ \ a_{001}(x) = 121,\ \ a_{010}(x) = 131,\ \ a_{011}(x) = 38,\\ a_{100}(x) = 180,\ \ a_{101}(x) = 3,\ \ a_{110}(x) = 196,\ \ a_{111}(x) = 146. \end{aligned}$$

Thus,

$$\hat{a} = (235, 180, 131, 196, 121, 3, 38, 146).$$

Similarly, we compute

$$b_0(x) = b_L(x) + \zeta^4 b_R(x) = 180 + 165x + 198x^2 + 67x^3$$

and

$$b_1(x) = b_L(x) - \zeta^4 b_R(x) = 218 + 76x + 45x^2 + 176x^3,$$

then

$$b_{00}(x) = 42 + 111x,\ b_{01}(x) = 77 + 219x,\ b_{10}(x) = 73 + 98x,\ b_{11}(x) = 122 + 54x,$$

and finally

$$\begin{aligned} b_{000}(x) = 43,\ \ b_{001}(x) = 41,\ \ b_{010}(x) = 73,\ \ b_{011}(x) = 81\\ b_{100}(x) = 16,\ \ b_{101}(x) = 130,\ \ b_{110}(x) = 153,\ \ b_{111}(x) = 91. \end{aligned}$$

Thus,

$$\hat{b} = (43, 16, 73, 153, 41, 130, 81, 91),$$

and

$$\hat{c} = \hat{a} \circ \hat{b} = (224, 229, 164, 104, 141, 149, 186, 31).$$

Finally, we compute (details omitted)

$$c(x) = \mathrm{NTT}^{-1}(\hat{c}) = 33 + 3x + 239x^2 + 37x^3 + 71x^4 + 43x^5 + 143x^6 + 69x^7.$$

11.3 Dilithium NTT

The FIPS 204 standard that specifies the Dilithium signature scheme fixes the parameters $n = 256$ and $q = 2^{23} - 2^{13} + 1 = 8,380,417$, chosen so that $2n$ divides $q-1$, and the order-$2n$ element $\zeta = 1753$ in $\mathbb{Z}_q^*$. The associated Dilithium ring $R_q = \mathbb{Z}_q[x]/(x^{256}+1)$ consists of polynomials of degree at most 255 with coefficients in $\mathbb{Z}_{8380417}$.

Dilithium NTT definition. The NTT used in Dilithium, as specified in FIPS 204, differs slightly from the transform introduced in Definition 11.1. In that earlier definition, the components of $\hat{a} = \text{NTT}(a)$ correspond to the consecutive odd powers of ζ, namely $\hat{a}_i = a(\zeta^{2i-1})$ for $1 \le i \le n$. In contrast, the Dilithium NTT permutes these powers of ζ so that they appear in the same order as the leaves of the underlying NTT computation tree. This avoids having to permute the values corresponding to the leaves of the NTT tree as was done in (135).

This ordering of leaves can be described succinctly using *bit reversal*. Suppose that $n = 2^k$. For an integer $v \in [0, n-1]$, let $\text{brv}(v)$ denote the k-bit integer obtained by reversing the k-bit binary representation of v. For example, when $k = 8$, we have $\text{brv}(113) = 142$ since the 8-bit binary representation of 113 is 01110001 whose reversal is 10001110.

Definition 11.4 Let $n = 2^k$, and let q be a prime such that $q-1$ is divisible by $2n$. Let $\zeta \in \mathbb{Z}_q^*$ be an element of order $2n$, and define

$$\zeta_i = \zeta^{2\text{brv}(i)+1} \bmod q \quad \text{for } 0 \le i \le n-1.$$

The *Dilithium NTT* is the map $\text{NTT} : R_q \longrightarrow \mathbb{Z}_q^n$ given by

$$\text{NTT}(a) = (a(\zeta_0), a(\zeta_1), a(\zeta_2), \ldots, a(\zeta_{n-2}), a(\zeta_{n-1})).$$

A nice feature of NTT and its inverse is that both can be computed *in-place*. As a result, only minimal auxiliary storage is required for NTT computations, which is especially advantageous for hardware implementations. We next illustrate the in-place computation for the case $n = 8$.

In-place NTT computation. Let $n = 8$, and recall that $\zeta^8 = -1$. The brv function is:

i	0	1	2	3	4	5	6	7
$\text{brv}(i)$	0	4	2	6	1	5	3	7.

The Dilithium NTT of a polynomial $A(x) = A_0 + A_1 x + \cdots + A_7 x^7 \in R_q$ is

$$\text{NTT}(A) = (A(\zeta), A(\zeta^9), A(\zeta^5), A(\zeta^{13}), A(\zeta^3), A(\zeta^{11}), A(\zeta^7), A(\zeta^{15})).$$

Here are the steps of the computation of $\text{NTT}(A)$:

1. A length-8 array is initialized with the coefficients of $A(x)$:

$$[A_0, A_1, A_2, A_3, A_4, A_5, A_6, A_7].$$

2. The array entries are divided into two halves $[a_L, a_R]$, and then updated to $[a_0, a_1]$ where $a_0 = a_L + t$ and $a_1 = a_L - t$ and $t = \zeta^4 a_R$; see equations (125) and (126).

3. Next, the array entries are divided into four pieces and updated to $[a_{00}, a_{01}, a_{10}, a_{11}]$; see equations (127)–(130).

4. Finally the array entries are updated to $[a_{000}, a_{001}, a_{010}, a_{011}, a_{100}, a_{101}, a_{110}, a_{111}]$ according to equations (131)–(134), yielding $\text{NTT}(A)$.

Each step updates the array in place, requiring no additional arrays beyond the original length-n input.

Dilithium key generation. To generate a Dilithium key pair, a user randomly selects a $k \times \ell$ matrix of polynomials $A \in_R R_q^{k\times\ell}$, together with two polynomial vectors $s_1 \in_R S_\eta^\ell$ and $s_2 \in_R S_\eta^k$. Here, S_η denotes the set of all polynomials in R_q whose coefficients lie in the interval $[-\eta, \eta]$. The user then computes the $k \times 1$ polynomial vector $t = As_1 + s_2$. Her public key is (A, t), while her private key is (s_1, s_2).

NTT is used to accelerate the $k\ell$ polynomial multiplications needed to compute the matrix-vector product As_1. To minimize the number of NTT and NTT^{-1} applications, the matrix A is generated directly in NTT form. Specifically, each entry of $\hat{A}$ is a randomly selected vector $\hat{a} \in_R \mathbb{Z}_q^n$; the corresponding polynomial $a = \text{NTT}^{-1}(\hat{a}) \in R_q$ is not needed and so is never explicitly computed.

Key generation proceeds as follows.

1. Select $\hat{A} \in_R (\mathbb{Z}_q^n)^{k\times\ell}$.

2. Select $s_1 \in_R S_\eta^\ell$ and $s_2 \in_R S_\eta^k$ and compute $\hat{s}_1 = \text{NTT}(s_1)$.
(Here, $\text{NTT}(s_1)$ means applying the NTT to each polynomial in s_1.)

3. Compute $\hat{b} = \hat{A} \circ \hat{s}_1$.
(Here, $\hat{A}\circ\hat{s}_1$ means the matrix-vector product of $\hat{A}$ and $\hat{s}_1$, where each multiplication within the product is performed componentwise.)

4. Compute $b = \text{NTT}^{-1}(\hat{b})$.

5. Compute $t = b + s_2$.

Overall, NTT is applied only ℓ times (to compute $\hat{s}_1$ in step 2), while NTT^{-1} is applied k times (to compute b in step 4). The $k\ell$ multiplications needed in step 3 are fast componentwise multiplications in $\mathbb{Z}_q^n$.

Dilithium signature generation. The dominant computation in signature generation is the evaluation of $w = Ay$, where A is the user's public key matrix and y is a randomly selected $\ell \times 1$ polynomial vector in $\tilde{S}_{\gamma_1}^{\ell}$. Here, $\tilde{S}_{\gamma_1}$ denotes the set of all polynomials in R_q whose coefficients lie in the interval $(-\gamma_1, \gamma_1]$. This computation is performed using the NTT as follows:

1. Select $y \in_R \tilde{S}_{\gamma_1}^{\ell}$ and compute $\hat{y} = \text{NTT}(y)$.
2. Compute $\hat{w} = \hat{A} \circ \hat{y}$.
3. Compute $w = \text{NTT}^{-1}(\hat{w})$.

Each of the ℓ NTT applications in step 1 and the k applications of NTT^{-1} in step 3 requires $\frac{1}{2} n \log_2 n$ $\mathbb{Z}_q$-multiplications[21]. The $k\ell$ componentwise multiplications in step 2 each require n $\mathbb{Z}_q$-multiplications. Consequently, the total cost is $\frac{1}{2}(\ell + k) n \log_2 n + k\ell n$ multiplications in $\mathbb{Z}_q$. In contrast, computing Ay using conventional polynomial multiplication would require n^2 $\mathbb{Z}_q$-multiplications for each of the $k\ell$ products, for a total of $k\ell n^2$ multiplications in $\mathbb{Z}_q$.

The ML-DSA-87 Dilithium parameter set specified in FIPS 204 uses $k = 8$ and $\ell = 7$. For these parameters (and with $n = 256$), computing w using the NTT requires approximately $2^{14.9}$ $\mathbb{Z}_q$-multiplications, whereas a naive approach using conventional polynomial multiplication would require approximately $2^{21.8}$ $\mathbb{Z}_q$-multiplications. Thus, the use of NTT yields a speedup of roughly a factor $2^{6.9} \approx 122$, representing a very substantial performance improvement.

11.4 Kyber NTT

The Kyber NTT parameters are $n = 256$ and $q = 3329$, so the associated polynomial ring is $R_q = \mathbb{Z}_{3329}[x]/(x^{256} + 1)$. Since $q - 1 = 3328 = 2^8 \times 13$, the integer $q - 1$ is divisible by n but not by $2n$. As a result the standard NTT algorithm must be slightly modified: specifically, the height of the NTT computation tree is reduced by one level. The modified transform is called NTT_2.

NTT_2 definition. Let $n = 2^k$, and let q be a prime such that $q - 1$ is divisible by n but not by $2n$. Let $\zeta \in \mathbb{Z}_q^*$ be an element of order n, so that $\zeta^{n/2} = -1$.

Lemma 11.5 For every odd integer $i \in [1, n-1]$, the polynomial $x^2 - \zeta^i$ is irreducible over $\mathbb{Z}_q$.

Proof: Suppose that $x^2 - \zeta^i$ were reducible over $\mathbb{Z}_q$. Then it would have a root c in $\mathbb{Z}_q$, implying $c^2 = \zeta^i$. Raising both sides to the n-th power gives $(c^2)^n = (\zeta^i)^n = (\zeta^n)^i = 1$, so the order of c divides $2n$. On the other hand, $c^n = (c^2)^{n/2} = (\zeta^i)^{n/2} = (\zeta^{n/2})^i = (-1)^i = -1$, which shows that the order of c does not divide n. Therefore, c has order $2n$,

[21] We are not counting additions, subtractions, and divisions by 2 in $\mathbb{Z}_q$, since these operations are less costly than a multiplication in $\mathbb{Z}_q$.

which is impossible since $2n$ does not divide $q-1$. This contradiction proves that $x^2-\zeta^i$ is irreducible. □

For each odd integer $i \in [1, n-1]$, define the *quadratic ring of polynomials*

$$Q_i = \mathbb{Z}_q[x]/(x^2-\zeta^i).$$

Elements of Q_i are polynomials of the form c_0+c_1x with coefficients in $\mathbb{Z}_q$; such a polynomial can be represented by its coefficient pair $(c_0, c_1) \in \mathbb{Z}_q^2$. Multiplication is performed modulo $x^2-\zeta^i$. The addition, subtraction and multiplication operations in Q_i are the following: if $a(x) = a_0 + a_1x$, $b(x) = b_0 + b_1x \in Q_i$, then

$$a(x) \pm b(x) = (a_0 \pm b_0) + (a_1 \pm b_1)x,$$

and

$$a(x) \cdot b(x) = (a_0b_0 + a_1b_1\zeta^i) + (a_0b_1 + a_1b_0)x.$$

Define the ring T_q as the product of the $\frac{n}{2}$ quadratic rings $Q_1, Q_3, \ldots, Q_{n-1}$:

$$T_q = Q_1 \times Q_3 \times \cdots \times Q_{n-1}.$$

Elements of T_q can be represented as vectors in $\mathbb{Z}_q^n$.

Definition 11.6 Let $n = 2^k$, and let q be a prime such that $q-1$ is divisible by n but not by $2n$. Let $\zeta \in \mathbb{Z}_q^*$ be an element of order n, and let $T_q = Q_1 \times Q_3 \times \cdots \times Q_{n-1}$ where $Q_i = \mathbb{Z}_q[x]/(x^2-\zeta^i)$. The *Quadratic Number-Theoretic Transform* (NTT_2) is the map $\mathrm{NTT}_2 : R_q \longrightarrow T_q$ defined by

$$\mathrm{NTT}_2(a) = (a \bmod (x^2-\zeta),\ a \bmod (x^2-\zeta^3),\ \ldots,\ a \bmod (x^2-\zeta^{n-1})).$$

For notational convenience, we will often write $\hat{a}$ in place of $\mathrm{NTT}_2(a)$.

The map $\mathrm{NTT}_2 : R_q \longrightarrow T_q$ is a ring isomorphism and is therefore invertible.

Addition and subtraction in R_q. Let $a(x), b(x) \in R_q$, and let $c(x) = a(x) \pm b(x)$. For any odd integer i, we have

$$c \bmod (x^2-\zeta^i) = a \bmod (x^2-\zeta^i) \pm b \bmod (x^2-\zeta^i),$$

and so $\hat{c} = \mathrm{NTT}_2(c) = \hat{a} \pm \hat{b}$. Consequently, addition and subtraction in NTT_2 form are performed componentwise.

Fast multiplication in R_q. Let $a(x), b(x) \in R_q$, and let $c(x)$ be their product in R_q, i.e., $c(x) = a(x)b(x) \bmod (x^n + 1)$. There exists a polynomial $\ell(x) \in \mathbb{Z}_q[x]$ satisfying

$$c(x) = a(x)b(x) + \ell(x)(x^n + 1).$$

For any odd integer i, we have

$$\begin{aligned} x^n + 1 \bmod (x^2 - \zeta^i) &= (x^2)^{n/2} + 1 \bmod (x^2 - \zeta^i) \\ &= (\zeta^i)^{n/2} + 1 = (\zeta^{n/2})^i + 1 = (-1)^i + 1 = 0 \end{aligned}$$

and so

$$\begin{aligned} c(x) \bmod (x^2 - \zeta^i) &= a(x)b(x) + \ell(x)(x^n + 1) \bmod (x^2 - \zeta^i) \\ &= a(x)b(x) \bmod (x^2 - \zeta^i). \end{aligned}$$

Thus, $\mathrm{NTT}_2(c)$ is

$$\begin{aligned} \hat{c} &= (a(x)b(x) \bmod (x^2 - \zeta),\ a(x)b(x) \bmod (x^2 - \zeta^3),\ \ldots,\ a(x)b(x) \bmod (x^2 - \zeta^{n-1})) \\ &= \hat{a} \circ \hat{b}, \end{aligned}$$

where $\hat{a} \circ \hat{b}$ denotes componentwise multiplication in T_q.

As with the standard NTT, both NTT_2 and NTT_2^{-1} can be computed in time $O(n \log n)$. This yields an alternative procedure for multiplying $a(x)$ and $b(x)$ in R_q:

1. Compute $\hat{a} = \mathrm{NTT}_2(a)$ and $\hat{b} = \mathrm{NTT}_2(b)$.
2. Compute the componentwise product $\hat{c} = \hat{a} \circ \hat{b}$.
3. Recover $c = \mathrm{NTT}_2^{-1}(\hat{c})$.

Since the componentwise multiplication requires only $O(n)$ operations, the overall complexity is $O(n \log n)$, a substantial improvement over the $O(n^2)$ running time of the classical method.

We next present an explicit example of an NTT_2 computation in the case $n = 8$. Full descriptions of $O(n \log n)$-time algorithms for computing NTT_2 and its inverse NTT_2^{-1} for the general case $n = 2^k$ are left to the reader (Exercise 11.5).

NTT_2 computation for $n = 8$. To illustrate the transform, consider $n = 8$. Let q be a prime such that $q - 1$ is divisible by $n = 8$ but not by $2n = 16$, and let $\zeta \in \mathbb{Z}_q^*$ be an element of order 8. Then $\zeta^4 = -1$. The polynomial ring under consideration is $R_q = \mathbb{Z}_q[x]/(x^8+1)$, and the transform is $\mathrm{NTT}_2 : R_q \longrightarrow T_q$, where $T_q = Q_1 \times Q_3 \times Q_5 \times Q_7$ and $Q_i = \mathbb{Z}_q[x]/(x^2 - \zeta^i)$.

The NTT_2 computation is based on the identities (136) and (137) derived as follows:

$$\begin{aligned} x^8 + 1 &= x^8 - \zeta^4 \\ &= (x^4 - \zeta^2)(x^4 + \zeta^2) \qquad (136) \\ &= (x^4 - \zeta^2)(x^4 - \zeta^6) \\ &= (x^2 - \zeta)(x^2 + \zeta)(x^2 - \zeta^3)(x^2 + \zeta^3). \qquad (137) \end{aligned}$$

Let $a \in R_q$, so that $a(x) \in \mathbb{Z}_q[x]$ has degree at most 7. The computation of $\mathrm{NTT}_2(a)$ proceeds recursively down the levels of the *NTT_2 computation tree* shown in Figure 11.2. Specifically, the first step is to compute

$$a_0 = a \bmod (x^4 - \zeta^2) \quad \text{and} \quad a_1 = a \bmod (x^4 + \zeta^2), \qquad (138)$$

and then

$$a_{00} = a_0 \bmod (x^2 - \zeta), \quad a_{01} = a_0 \bmod (x^2 + \zeta), \qquad (139)$$

$$a_{10} = a_1 \bmod (x^2 - \zeta^3), \quad a_{11} = a_1 \bmod (x^2 + \zeta^3). \qquad (140)$$

The final output of the transform is then

$$\mathrm{NTT}_2(a) = (a_{00}, a_{10}, a_{01}, a_{11}). \qquad (141)$$

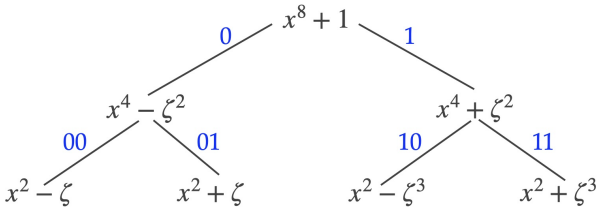


Figure 11.2: The NTT_2 computation tree ($n = 8$).

Example 11.7 (*NTT_2 computation*) Let $q = 233$ and $n = 8$, so the polynomial ring of interest is $R_q = \mathbb{Z}_{233}[x]/(x^8 + 1)$. Note that n divides $q - 1 = 8 \times 29$ but $2n = 16$ does not. The element $\zeta = 12 \in \mathbb{Z}_{233}^*$ has order 8. Consider the polynomials

$$a(x) = 92 + 152x + 57x^3 + 29x^5 + 133x^6 + 200x^7 \in R_q$$

and

$$b(x) = 6 + 45x^2 + 112x^3 + 4x^4 + 8x^5 + 165x^6 + 212x^7 \in R_q.$$

Our goal is to compute $c(x) = a(x)b(x) \bmod (x^8 + 1)$ using NTT_2.

We begin by computing $\hat{a} = \mathrm{NTT}_2(a)$. Write

$$a_L(x) = 92 + 152x + 57x^3 \quad \text{and} \quad a_R(x) = 29x + 133x^2 + 200x^3.$$

Then

$$a_0(x) = a_L(x) + \zeta^2 a_R(x) = 92 + 134x + 46x^2 + 198x^3$$

and

$$a_1(x) = a_L(x) - \zeta^2 a_R(x) = 92 + 170x + 187x^2 + 149x^3.$$

Reducing further yields

$$a_{00}(x) = 178 + 180x, \quad a_{01}(x) = 6 + 88x, \quad a_{10}(x) = 57 + 177x, \quad a_{11}(x) = 127 + 163x.$$

Thus,

$$\hat{a} = (178 + 180x,\ 57 + 177x,\ 6 + 88x,\ 127 + 163x).$$

Applying the same procedure to $b(x)$, we obtain

$$\hat{b} = (118 + 226x,\ 183 + 140x,\ 114 + 214x,\ 75 + 119x).$$

Hence, componentwise multiplication gives

$$\hat{c} = \hat{a} \circ \hat{b} = (59 + 189x,\ 211 + 62x,\ 11 + 132x,\ 171 + 77x).$$

Finally, applying the inverse transform (details omitted) gives

$$c(x) = \mathrm{NTT}_2^{-1}(\hat{c}) = 113 + 115x + 114x^2 + 119x^3 + 48x^4 + 205x^5 + 51x^6 + 18x^7.$$

Kyber NTT_2. The FIPS 203 standard that specifies Kyber fixes the parameters $n = 256$ and $q = 3329$. The order-256 element $\zeta = 17 \in \mathbb{Z}_{3329}^*$ is used. The corresponding Kyber ring $R_q = \mathbb{Z}_{3329}[x]/(x^{256} + 1)$ consists of polynomials of degree at most 255 with coefficients in $\mathbb{Z}_{3329}$.

The NTT_2 transform employed in Kyber, as specified in FIPS 203, differs slightly from the transform defined in Definition 11.6. In that earlier definition, the i-th component of $\hat{a} = \mathrm{NTT}_2(a)$ is $\hat{a}_i = a(x) \bmod (x^2 - \zeta^{2i-1})$ for $1 \le i \le \frac{n}{2}$. In contrast, the Kyber NTT_2 permutes these powers of ζ so that they appear in the same order as the leaves of the underlying NTT_2 computation tree. This avoids having to permute the values corresponding to the leaves of the NTT tree as was done in (141).

This ordering of leaves is concisely described using *bit reversal*. For an integer $v \in [0, 127]$, let $\mathrm{BitRev}(v)$ denote the 7-bit integer obtained by reversing the 7-bit binary representation of v. For example, we have $\mathrm{BitRev}(113) = 71$.

Definition 11.8 For $0 \le i \le 127$, define $\zeta_i = \zeta^{2\mathrm{BitRev}(i)+1} \bmod q$. The Kyber NTT_2 is the map $\mathrm{NTT}_2 : R_q \longrightarrow T_q$ defined by

$$\mathrm{NTT}_2(a) = (a \bmod (x^2 - \zeta_0),\ a \bmod (x^2 - \zeta_1),\ \ldots,\ a \bmod (x^2 - \zeta_{127})).$$

Kyber key generation. To generate a Kyber-KEM key pair, a user randomly selects $\rho \in_R \{0,1\}^{256}$ and constructs the matrix $A \in R_q^{k \times k}$ from ρ using a pseudorandom function: $A = \text{Expand}(\rho)$. The user selects $s \in_R S_{\eta_1}^k$ and $e \in_R S_{\eta_1}^k$, where S_{η_1} denotes the set of polynomials in R_q all of whose coefficients lie in the interval $[-\eta_1, \eta_1]$. The user computes the $k \times 1$ polynomial vector $t = As + e$. Their public key is (A, t), and their private key is s.

NTT_2 is used to accelerate the k^2 polynomial multiplications needed to compute the matrix-vector product As. To minimize the number of NTT_2 and NTT_2^{-1} applications, the matrix A is generated directly in NTT_2 form. Key generation proceeds as follows.

1. Compute $\hat{A} = \text{Expand}(\rho)$.
2. Select $s \in_R S_{\eta_1}^k$ and $e \in_R S_{\eta_1}^k$, and compute $\hat{s} = \text{NTT}_2(s)$ and $\hat{e} = \text{NTT}_2(e)$.
3. Compute $\hat{t} = \hat{A} \circ \hat{s} + \hat{e}$.
4. The user's public key is $(\rho, \hat{t})$, and their private key is $\hat{s}$.

The transforms NTT_2 and NTT_2^{-1} are also used during key encapsulation and key decapsulation.

11.5 Exercises

Exercise 11.1 Let $n = 2^k$. For each $0 \le i \le n-1$, prove that the leaf of the NTT computation tree indexed by i is $x - \zeta^{2\text{brv}(i)+1}$ (cf. Fig. 11.1).

Exercise 11.2 Consider the NTT parameters $q = 97$, $n = 4$ and $\zeta = 64$, so the polynomial ring of interest is $R_q = \mathbb{Z}_{97}[x]/(x^4 + 1)$.
(a) Write pseudocode for evaluating NTT and NTT^{-1}.
(b) Let $a(x) = 1 + 71x + 52x^2 + 90x^3 \in R_q$, $b(x) = 83 + 31x + 13x^2 + 54x^3 \in R_q$, and let $c(x) = a(x)b(x) \bmod (x^4 + 1)$. Verify the following computations:
(i) $\hat{a} = (4, 19, 55, 23)$.
(ii) $\hat{b} = (9, 50, 50, 29)$.
(iii) $\hat{c} = (36, 77, 34, 85)$.
(iv) $c = \text{NTT}^{-1}(\hat{c}) = 58 + 6x + 21x^2 + 68x^3$.

Exercise 11.3 Consider the NTT parameters $q = 241$, $n = 8$ and $\zeta = 115$, so the polynomial ring of interest is $R_q = \mathbb{Z}_{241}[x]/(x^8 + 1)$. Let $a(x) = 11x + 22x^2 + 33x^3 + 44x^4 + 55x^5 + 66x^6 + 77x^7 \in R_q$, $b(x) = 77x + 66x^2 + 55x^3 + 44x^4 + 33x^5 + 22x^6 + 11x^7 \in R_q$, and let $c(x) = a(x)b(x) \bmod (x^8 + 1)$. Verify the following computations:
(a) $\text{NTT}(a) = (149, 228, 183, 57, 37, 128, 176, 6)$.
(b) $\text{NTT}(b) = (235, 65, 113, 204, 184, 58, 13, 92)$.
(c) $\text{NTT}(c) = (70, 119, 194, 60, 60, 194, 119, 70)$.
(d) $c = \text{NTT}^{-1}(\text{NTT}(c)) = 171 + 185x + 202x^2 + 221x^3 + 20x^5 + 39x^6 + 56x^7$.

Exercise 11.4 Understand and then implement the Dilithium NTT and NTT^{-1} in-place algorithms described in Section 7.5 of FIPS 204.

Exercise 11.5 Let $n = 2^k$, and let q be a prime that is divisible by n but not by $2n$. Describe $O(n \log n)$-time algorithms for evaluating NTT_2 and NTT_2^{-1} with respect to the polynomial ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$.

Exercise 11.6 Consider the NTT_2 parameters $q = 569$, $n = 8$ and $\zeta = 277$, so the polynomial ring of interest is $R_q = \mathbb{Z}_{569}[x]/(x^8 + 1)$.
(a) Write pseudocode for evaluating NTT_2 and NTT_2^{-1}.
(b) Let $a(x) = 199 + 352x^2 + 157x^3 + 2x^4 + 129x^5 + 343x^6 + 499x^7 \in R_q$,
$b(x) = 16 + 73x + 145x^2 + 332x^3 + 114x^4 + 128x^5 + 164x^6 + 312x^7 \in R_q$,
and let $c(x) = a(x)b(x) \bmod (x^8 + 1)$. Verify the following computations:
(i) $\text{NTT}_2(a) = (126 + 332x,\ 368 + 222x,\ 497 + 240x,\ 374 + 344x)$.
(ii) $\text{NTT}_2(b) = (166 + 45x,\ 264 + 402x,\ 173 + 276x,\ 30 + 138x)$.
(iii) $\text{NTT}_2(c) = (475 + 468x,\ 486 + 566x,\ 125 + 26x,\ 566 + 480x)$.
(iv) $c = \text{NTT}_2^{-1}(\text{NTT}_2(c)) = 413 + 385x + 28x^2 + 156x^3 + 524x^4 + 81x^5 + 327x^6 + 475x^7$.

Exercise 11.7 Consider the NTT_2 parameters $q = 241$, $n = 16$ and $\zeta = 111$, so the polynomial ring of interest is $R_q = \mathbb{Z}_{241}[x]/(x^{16} + 1)$.
(a) Write pseudocode for evaluating NTT_2 and NTT_2^{-1}.
(b) Let $a(x) = 92 + 238x + 121x^2 + 44x^3 + 223x^4 + 5x^5 + 225x^6 + 186x^7 + 43x^8 + 99x^9 + 37x^{10} + 196x^{11} + 26x^{12} + 95x^{13} + 16x^{14} + 231x^{15} \in R_q$,
$b(x) = 228 + 89x + 161x^2 + 17x^3 + 179x^4 + 55x^5 + 170x^6 + 82x^7 + 24x^8 + 89x^9 + 92x^{10} + 59x^{11} + 220x^{12} + 141x^{13} + 49x^{14} + 122x^{15} \in R_q$,
and let $c(x) = a(x)b(x) \bmod (x^{16} + 1)$. Verify the following computations:
(i) $\text{NTT}_2(a) = (129 + 98x,\ 181 + 36x,\ 143 + 32x,\ 61 + 64x,\ 153 + 130x,\ 31 + 94x,\ 21 + 171x,\ 17 + 74x)$.
(ii) $\text{NTT}_2(b) = (222+119x,\ 141+217x,\ 109+122x,\ 210+134x,\ 95+7x,\ 171+208x,\ 126+219x,\ 27 + 168x)$.
(iii) $\text{NTT}_2(c) = (30+234x,\ 154+9x,\ 186+208x,\ 166+165x,\ 44+166x,\ 158+109x,\ 201+117x,\ 89 + 34x)$.
(iv) $c = \text{NTT}_2^{-1}(\text{NTT}_2(c)) = 8 + 70x + 195x^2 + 211x^3 + 39x^4 + 48x^5 + 22x^6 + 145x^7 + 116x^8 + 131x^9 + 183x^{10} + 216x^{11} + 213x^{12} + 101x^{13} + 176x^{14} + 65x^{15}$.

Exercise 11.8 Understand and then implement the Kyber NTT_2 and NTT_2^{-1} in-place algorithms described in Section 4.3 of FIPS 203.

12 Notes and further references

§1 Introduction

Lattices are fundamental structures that play a central role in numerous areas of mathematics and computer science, including the geometry of numbers, coding theory, optimization, and cryptography. The *geometry of numbers* is a branch of number theory that applies geometric techniques, particularly those involving lattices, to problems concerning integers and algebraic numbers. The subject was founded by Hermann Minkowski in the late nineteenth century; a classical reference is the book by Cassels [38]. Notable applications of lattices include H. Lenstra's polynomial-time algorithm [101] for integer linear programming in fixed dimension n, Viazovska's solution [156] of the sphere packing problem in dimension 8, the work of Kannan and Lovász [86] on the covering radius of convex bodies, and the capacity-achieving lattice coding scheme for the additive white Gaussian noise (AWGN) channel developed by Erez and Zamir [61]. More recently, in 2023, Reis and Rothvoss [142] introduced a $(\log(2n))^{O(n)}$-time randomized algorithm for solving integer programs in n variables.

These notes focus on developing a thorough understanding of the lattice-based cryptographic schemes Kyber and Dilithium, which were standardized by NIST. As part of its post-quantum cryptography standardization initiative, NIST has also finalized standards for the stateful hash-based signature schemes LMS and XMSS in SP 800-208 [118], as well as the stateless hash-based signature scheme SPHINCS+ in FIPS 205 [121]. Menezes [111] gives an overview of these schemes.

§2 Lattices

The first monograph devoted to algorithmic problems in lattices and their computational complexity is the 2002 book by Micciancio and Goldwasser [114]. Part IV of Galbraith's book [65] introduces lattices and explores their applications in cryptanalysis. Nguyen [124] explains the connections between the Hermite constants, lattice algorithms, and sphere packings. Peikert [136] provides a comprehensive survey of the major developments in lattice-based cryptography up to 2015, including fully homomorphic encryption (FHE) and attribute-based encryption (ABE). More recently, Lyubashevsky [107] offers a detailed exposition of the mathematical foundations and design principles underlying SIS- and LWE-based cryptosystems, including Kyber and Dilithium. Kyber and Dilithium are fully specified in the NIST standards FIPS 203 [120] and FIPS 204 [119].

Conway and Sloane's book [48] is a rich source of information on the Leech lattice Λ_{24}. The basis matrix for Λ_{24} presented in Example 2.13 is taken from Figure 4.12 of that book. Bonnecaze and Solé [28] gave a construction of the Leech lattice from the binary $(23, 12, 7)$-Golay code. The *sphere packing* problem is to find an arrangement of non-overlapping unit balls in $\mathbb{R}^n$ so that the proportion of space they cover is as large as possible. Cohn, Kumar, Miller, Radchenko and Viazovska [47] proved that the densest sphere packing in $\mathbb{R}^{24}$ is attained with unit balls centered at the points of the Leech lattice. This work builds on Viazovska's proof [156] that the densest sphere packing in

$\mathbb{R}^8$ is attained by placing unit balls at the points of the scaled lattice $\sqrt{2}E_8$ (the E_8 lattice is defined in Exercise 8.11).

In 1998, Ajtai [4] proved that SVP is NP-hard under randomized reductions. Subsequently, Khot [87] proved that the approximate version SVP_γ remains NP-hard for constant approximation factors γ. The claim that SVP_γ is unlikely to be NP-hard for $\gamma > \sqrt{n}$ is supported by a result of Aharonov and Regev [2] that there exists $c > 0$ for which $\mathrm{GapSVP}_{c\sqrt{n}}$ lies in NP $\cap$ coNP (GapSVP_β is a decisional version of SVP). Blömer and Seifert [24] proved that SIVP_γ is NP-hard for constant approximation factors γ. Laarhoven, van de Pol and de Weger [94] and Stephens-Davidowitz [152] provide comprehensive overviews of the relationships between the main lattice problems. Stephens-Davidowitz [152] gave a proof that $\mathrm{SIVP}_{\gamma\sqrt{n}} \leq \mathrm{SVP}_\gamma$ for every $\gamma \geq 1$.

Micciancio [112] gave an elementary proof that CVP is NP-hard. Micciancio [113] gave polynomial-time reductions CVP $\leq$ SIVP and $\mathrm{SIVP}_\gamma \leq \mathrm{CVP}_\gamma$ for every $\gamma \geq 1$. Goldreich, Micciancio, Safra and Seifert [73] showed that for any $\gamma \geq 1$, SVP_γ polynomial-time reduces to CVP_γ. Conversely, Stephens-Davidowitz [152] proved that $\mathrm{CVP}_{\gamma^2\sqrt{n}}$ polynomial-time reduces to SVP_γ.

Babai [16] introduced two conceptually simple polynomial-time algorithms for CVP_γ. His *nearest plane algorithm* achieves an approximation factor of $\gamma = 2^{n/2}$, while his *rounding algorithm* (see Exercise 10.5) attains $\gamma = 1 + 2n(\frac{9}{2})^{n/2}$. The nearest plane algorithm has found applications in several areas beyond cryptography. In particular, Chen et al. [41] showed in 2025 that GPTQ, a widely used method for quantizing the weights of large language models (LLMs), can be viewed as an application of the nearest plane algorithm.

§3 Short Integer Solution problem

The SIS problem was introduced by Ajtai [3], who provided a worst-case to average-case reduction from approx-SIVP to SIS. The approximation factor in Ajtai's reduction was later improved by Micciancio and Regev [115]. A proof of Theorem 3.9 can be found in §1.6 of Stewart and Tall's book [153].

Gentry, Peikert and Vaikuntanathan [68, 69] introduced a framework for constructing hash-and-sign signature schemes. This framework was later employed in the design of the Falcon signature scheme by Fouque et al. [62]. Falcon (Fast-Fourier Lattice-Based Compact Signatures over NTRU) derives its security from the hardness of SIS on NTRU lattices. Falcon is currently undergoing standardization by NIST, with a draft standard, FIPS 206, anticipated in 2026.

Cash, Hofheinz, Kiltz and Peikert [37] introduced the concept of lattice-based *bonsai trees*, using it to construct a stateless SIS-based signature scheme whose security proof does not depend on the random oracle model.

§4 Learning With Errors problem

The LWE problem was introduced by Regev [141], who gave a worst-case to average-case quantum reduction from approx-SIVP to LWE. For a concrete analysis of the tightness gap in Regev's reduction, see the work of Chatterjee, Koblitz, Menezes and Sarkar [40].

The Lindner-Peikert public-key encryption scheme [103] forms the basis for the Kyber public-key encryption scheme specified in FIPS 203. LWE also forms the basis for a wide variety of cryptographic protocols, including the key encapsulation mechanism FrodoKEM of Glabush et al. [71], the identity-based encryption scheme of Gentry, Peikert and Vaikuntanathan [68, 69], the hierarchical identity-based encryption scheme of Cash, Hofheinz, Kiltz and Peikert [37], the attribute-based encryption scheme of Gorbunov, Vaikuntanathan and Wee [74], and the fully-homomorphic encryption scheme of Cheon, Kim, Kim and Song [45].

The embedding technique used in the primal attack on LWE described in §4.4.1 was originally introduced by Kannan [85] to reduce approx-CVP to SVP. Albrecht [7] proposed a variant of the dual attack tailored to short-secret LWE. Howgrave-Graham [82] introduced *hybrid attacks*, which combine lattice basis reduction with meet-in-the-middle techniques.

The method of transforming an LWE instance into a system of high-degree multivariate polynomial equations that can then be solved via linearization was introduced by Arora and Ge [14]. Another known attack on LWE is the *BKW attack* introduced by Blum, Kalai and Wasserman [25]. This attack has been refined by many authors, including Kirchner and Fouque [89], Budroni, Guo, Johansson, Mårtensson and Wagner [34], and Guo, Johansson, Mårtensson and Wagner [75]. However, these attacks are not effective against Kyber and Dilithium, as they require an impractically large number of LWE samples. Machine learning-based approaches have also been investigated, most recently by Alfarano, Saxena, Wenger, Charton and Lauter [10], although these methods have so far achieved only limited success.

The *Learning With Rounding (LWR) problem* is a variant of LWE in which the random error term is replaced by deterministic rounding. Let integers $q \geq p \geq 2$ be fixed. Define the *rounding function* $\lfloor \cdot \rfloor_p : \mathbb{Z}_q \to \mathbb{Z}_p$ by $\lfloor x \rfloor_p = \lfloor (p/q) \cdot x \rfloor$. This function naturally extends to vectors in $\mathbb{Z}_q^m$ by applying it componentwise. The LWR problem is then defined as follows: given a uniformly random matrix $A \in_R \mathbb{Z}_q^{m \times n}$ and a vector $b = \lfloor As \rfloor_p$, where $s \in_R \mathbb{Z}_q^n$ is chosen uniformly at random, determine s. The LWR problem was introduced by Banerjee, Peikert and Rosen [18], and a tighter reduction from LWE to LWR was later provided by Alwen, Krenn, Pietrzak and Wichs [12]. One advantage of LWR over LWE is that it requires fewer random bits, since the noise is deterministic (via rounding) rather than randomly sampled. Notable KEMs based on the module version of the LWR assumption include Saber, designed by D'Anvers, Karmakar, Roy and Vercauteren [53], and SMAUG-T [44], which is currently undergoing standardization in South Korea.

§5 Module-SIS and Module-LWE

Ring-SIS, an analogue of SIS that employs certain polynomial rings, was formulated by Lyubashevsky and Micciancio [108]. The module-based analogue, Module-SIS (MSIS), which employs vectors of polynomials, was developed by Langlois and Stehlé [97].

Ring-LWE, a variant of LWE that employs certain polynomial rings, was formulated by Lyubashevsky, Peikert and Regev [109], who also provided a worst-case to average-case quantum reduction from approx-SIVP in ideal lattices to Ring-LWE. The module variant, Module-LWE (MLWE), was first introduced by Brakerski, Gentry and Vaikuntanathan [32], and a worst-case to average-case quantum reduction was developed by Langlois and Stehlé [97]. A concrete analysis of the tightness gaps in the Ring-LWE and MLWE reductions was carried out by Koblitz, Samajder, Sarkar and Singha [91], who argued that the approximation factor in the SIVP problem is too large to yield meaningful security guarantees in practice. Wenger et al. [157] provide concrete benchmarks for several lattice-based attacks on LWE with parameter choices from Kyber and a Ring-LWE-based homomorphic encryption scheme.

§6 Kyber (ML-KEM)

Kyber-PKE was developed by Bos et al. [31]. The Fujisaki-Okamoto transform was introduced in 1999 [64]. A variant of it, due to Hofheinz, Hövelmanns and Kiltz [80], was used by Bos et al. [31] in the design of Kyber-KEM. For a comprehensive description of Kyber-KEM, including the specifications of the hash functions G, H, and J, the eXtendable Output Function (XOF) used to generate A, and the pseudorandom function (PRF) used to generate s, e, r, e_1 and e_2, see FIPS 203 [120].

D'Anvers et al. [52] showed how information leaked through Kyber-KEM decapsulation failure can be exploited to formulate an MLWE instance with a secret that is smaller than the original pair (s, e), thereby making the problem easier to solve. They then applied standard cost estimates for solving MLWE to approximate the time required to recover the secret key. Their analysis indicates that the attack is ineffective when the number of decapsulation queries is limited to 2^{64}. In subsequent work, D'Anvers, Rossi and Virdia [54] proposed a technique called *directional failure boosting*, in which a previously identified failing ciphertext is leveraged to dramatically accelerate the discovery of additional ciphertexts that also fail decapsulation.

Most PQC deployments adopt a *hybrid* approach. In this model, traditional elliptic-curve key agreement such as elliptic-curve Diffie-Hellman (ECDH) is combined with a quantum-resistant key establishment mechanism such as Kyber-KEM. The final shared secret key is derived by combining the outputs of both schemes via a key derivation function. Hybrid constructions mitigate *harvest-now, decrypt-later* attacks by ensuring that recorded traffic remains secure even if large-scale quantum computers become available in the future. At the same time, the inclusion of a classical and well-studied primitive such as ECDH provides a hedge against the possibility that unforeseen weaknesses are later discovered in the quantum-resistant component. This dual protection, however, is lost if cryptographically relevant quantum computers become available *and* major weak-

nesses are discovered in the quantum-resistant scheme.

Giacon, Heuer and Poettering [70] studied key combiners for hybrid key agreement that remain secure provided that at least one of the constituent KEMs is secure. NIST's SP 800-227 [122] recommends the following generic key combiner. For $i \in \{1, 2\}$, let ek_i, c_i and K_i denote the encapsulation key, ciphertext and shared secret key for two KEMs. The combined key is then computed as $K = H(K_1, K_2, c_1, c_2, ek_1, ek_2, \text{domain\_sep})$, where "domain_sep" is a string that uniquely identifies the composite KEM scheme in use. A hybrid KEM tailored to combine X25519 elliptic curve key agreement with ML-KEM-768 was proposed by Barbosa et al. [20]; this construction achieves improved efficiency compared to generic combiners.

Side-channel attacks against Kyber implementations continue to be an active area of research. A comprehensive survey on side-channel attacks on Kyber implementations was presented in 2024 by Ravi, Chattopadhyay, D'Anvers and Baksi [140]. Recent notable contributions include the works of Hermelink, Ning, Petri and Strieder [78], Bernstein et al. [22], and Fouque, Marion, Nguyen and Wallet [63].

§7 Dilithium (ML-DSA)

Dilithium was designed by Ducas, Kiltz, Lepoint, Lyubashevsky, Schwabe and Seiler [59] and draws on ideas from several earlier works, most notably the papers of Lyubashevsky [105, 106] and Bai and Galbraith [17]. It was heavily optimized to reduce public-key and signature sizes, while ensuring fast and constant-time implementations. For a full description of Dilithium, including specifications of ExpandA, ExpandS, ExpandMask, H, and SampleInBall, see FIPS 204 [119]. Formal reductionist security proofs for Dilithium are quite intricate; see Barbosa et al. [19] and Devevey et al. [57].

Azevedo-Oliveira, Calle Viera, Cogliati and Goubin [15] developed a practical linear-programming method that recovers the private key component t_0 from a Dilithium public key (ρ, t_1) together with a collection of signed messages. Approximately 200,000 signatures were needed for the ML-DSA-44 parameter set, while about 500,000 signatures were required for the ML-DSA-65 and ML-DSA-87 parameter sets. These results support the statement in the FIPS 204 document that "The low-order bits of t can be reconstructed from a small number of signatures and, therefore, need not be regarded as secret."

The vulnerability of Dilithium implementations to side-channel attacks remains an active area of research. A comprehensive survey of such attacks was presented in 2024 by Ravi, Chattopadhyay, D'Anvers and Baksi [140]. Recent notable works include attacks exploiting potential information leakage arising from rejection sampling by Zhou, Wang, Sun and Yu [162] and by Liu, Wang, Wei, Ding, Zhang, Liu and Zhu [104], as well as fault attacks studied by Ulitzsch, Marzougui, Bagia, Tibouchi and Seifert [154], Krahmer, Pessl, Land and Güneysu [93], and Kosuge and Xagawa [92].

Threshold signature schemes distribute a secret signing key among multiple parties so that a valid signature can only be produced by a coalition of parties exceeding a given size threshold. In 2026, multi-party computation techniques were used by Bienstock, de Castro, Escudero, Polychroniadou and Takahashi [23], and by Celi, del Pino, Espitau,

Niot and Prest [39], to construct threshold signing protocols for Dilithium.

§8 The LLL lattice basis reduction algorithm


Carl Friedrich Gauss

Gauss described his lattice basis reduction algorithm in 1801. The algorithm was, in fact, first discovered by Lagrange in 1773, but this was not well known until quite recently, and so the algorithm is commonly called Gauss's algorithm. The polynomial-time complexity of Gauss's algorithm was established by Lagarias [95]. Vallée [155] derived the best possible upper bound on the number of iterations, and explored the connection between Gauss's algorithm and Euclid's algorithm for computing the greatest common divisor of two integers. Nguyen and Stehlé [128] showed that the running time of Gauss's algorithm can be improved to $O(\log^2 X)$ bit operations, where X is the length of the longer of the two input basis vectors.

The LLL algorithm was introduced by Lenstra, Lenstra and Lovász [100] in their landmark 1982 paper, which also derived its $O(n^6 \log^3 X)$ bit complexity stated in Theorem 8.30, and presented the first polynomial-time algorithm for factoring polynomials over the rational numbers. The book *The LLL Algorithm: Survey and Applications*, edited by Nguyen and Vallée [131], contains a collection of articles covering algorithmic aspects of lattice reduction, applications to algorithmic number theory, integer programming, and cryptography, as well as the complexity of lattice problems. Chapter 1 provides an interesting historical account of the development of the LLL algorithm.

Stehlé [151] studied practical and theoretical aspects of floating-point variants of LLL. The L^2 variant of LLL was introduced by Nguyen and Stehlé [127]. Nguyen and Stehlé [126] reported extensive experimental results on the average-case behaviour of L^2, aiming to explain the significant gap between its practical performance and the proven worst-case bounds. Specifically, they studied the algorithm's behaviour on rank-d integer lattices $L \subseteq \mathbb{R}^n$ with $n = \Theta(d)$, whose bases are randomly generated with vector lengths bounded by X, where $d = O(\log X)$. Their experiments indicate that the average-case estimate of the *Hermite factor*, defined as $\|b_1\| / \operatorname{vol}(L)^{1/d}$ for the first vector b_1 of the output basis, is approximately 1.02^d. This represents a substantial improvement over the worst-case bound of $(4/3)^{d/4} \approx 1.075^d$ for $\delta \approx 1$; see equation (71). Most LLL-reduced bases do attain this worst-case upper bound, as shown by Kim and Venkatesh [88]. Furthermore, while the worst-case analysis requires about $1.58d$ bits of numerical precision, the average-case analysis suggests that about $0.25d$ bits suffice in practice. These experimental findings were further corroborated by Gama and Nguyen [66]. A recent LLL implementation incorporating several theoretical improvements is described by Ducas, Pulles and Stevens [60].

§9 The BKZ lattice basis reduction algorithm

Yasuda [158] surveys algorithms for solving SVP, including LLL, BKZ, sieving, and enumeration. A more recent survey that covers the latest advances in LLL, BKZ, enumeration, and sieving is provided by Shen and Li [149].

The notion of a β-BKZ-reduced basis was introduced by Schnorr [144]. The upper bound in Theorem 9.5 on the length of the first vector in a β-BKZ-reduced basis $B = [b_1, \ldots, b_n]$ was proved by Schnorr [145], who also showed that $\|b_i\| \leq \sqrt{(i+3)/4} \cdot \gamma_\beta^{(n-1)/(\beta-1)} \lambda_i(L)$ for $2 \leq i \leq n$. The BKZ algorithm is due to Schnorr and Euchner [147]. The modified-LLL algorithm, which produces an LLL-reduced basis when the input is a linearly dependent spanning set rather than a true basis, was first described by Pohst [137]; see also Section 2.6.4 of Cohen [46]. Hanrot, Pujol and Stehlé [76] proved that a slightly modified version of the BKZ algorithm terminates after $O(\beta^n)$ tours, which implies the existence of β-BKZ-reduced bases. In 2011, Chen and Nguyen [43] introduced BKZ 2.0, a refined version of BKZ designed with early abort. See also the article by Albrecht, Player and Scott [9], which performs a detailed analysis of BKZ 2.0 and gives concrete estimates for various families of LWE instances. The first rigorous analysis of BKZ was given by Li and Nguyen [102]. Running-time estimates for BKZ were used by Bos et al. [31] to assess the security of Kyber, and by Ducas et al. [59] to assess the security of Dilithium.

The basic enumeration algorithm was described by Kannan [85]. The algorithm can be optimized to have running time $O(2^{n \log n})$. Although enumeration algorithms are asymptotically slower than sieving, they require only modest amounts of memory. Schnorr and Euchner [147] introduced a method for pruning the search tree, and later an improved pruning method was proposed by Gama, Nguyen and Regev [67].

Sieving methods are the fastest algorithms known for solving SVP exactly. Sieving was originally proposed by Ajtai, Kumar and Sivakumar [6], who also showed that approx-SVP with approximation factor $\gamma > 2^{(n \log\log n)/\log n}$ is solvable in polynomial time. The sieving algorithm presented in §9.5, together with Theorem 9.10, is due to Nguyen and Vidick [132]. In Nguyen and Vidick's sieving algorithm, the vectors in the original list S_0 are selected according to a Gaussian distribution using Klein's algorithm [90]. Nguyen and Vidick give a more careful analysis than the one given in §9.5, and also provide experimental evidence for the validity of Heuristic Assumption 1. Theorem 9.11 appears in Cai, Fan and Jiang [35].

Micciancio and Voulgaris [116] proposed GaussSieve, which performs well in practice. The fastest classical sieving algorithm, due to Becker, Ducas, Gama and Laarhoven [21], uses *locality sensitive hashing* to speed up the search for nearby lattice vectors, and runs in time $2^{0.292n+o(n)}$. The fastest quantum variant, described by Bonnetain, Chailloux, Schrottenloher and Shen [29], runs in time $2^{0.256n+o(n)}$. While these running times are heuristic, the fastest known SVP solver with a rigorously proven running time is the $2^{0.63269n+o(n)}$-time algorithm of Pouly and Shen [139]. Aggarwal and Stephens-Davidowitz [1] described a conceptually simple sieving algorithm that provably runs in time $2^{n+o(n)}$ and space $2^{n+o(n)}$.

A notable drawback of sieving-based methods is that they require both exponential time and exponential space. Zhao, Ding and Yang [161] report on their solution in 2024 of a dimension-183 SVP challenge using sieving; their computation required 30 days on a 112-core server with 0.87 Terabytes of RAM. More recently, Zhao and Ding [160] report on their solution of a dimension-200 SVP challenge using sieving; their computation required 30 days on a server with eight RTX 4090 GPUs, 1 Terabyte of RAM, and 30 Terabytes of disk space.

The strategy of using the primal attack together with BKZ reduction to evaluate the concrete security of Kyber was first proposed by Alkim, Ducas, Pöppelman and Schwabe [11] in their work introducing the Ring-LWE-based key agreement protocol NewHope. This paper also introduced the notion of core-SVP hardness. The primal attack strategy was developed further by Bos et al. [31] in the CRYSTALS-Kyber submission to the NIST post-quantum standardization process. A method for recovering the target vector x from the projected vector x' was described and analyzed by Albrecht, Göpfert, Virdia and Wunderer [8]. They also gave experimental evidence supporting the accuracy of the security estimates, showing that the practical behaviour of the primal attack closely matches the theoretical predictions.

The Geometric Series Assumption (92) was introduced by Schnorr [146]. The asymptotic limit (96) for the root Hermite factor of β-BKZ-reduced bases of random lattices is due to Chen [42]. An efficient algorithm for simulating the lengths of the Gram-Schmidt basis vectors after each tour in the BKZ algorithm was presented by Chen and Nguyen [43].

Module-BKZ is a generalization of BKZ to module lattices, such as those arising in the study of MLWE. It was introduced by Lee, Pellet-Mary, Stehlé and Wallet [98], and by Mukherjee and Stephens-Davidowitz [117]. The average-case behaviour of module-BKZ was later analyzed by Ducas, Engelberts and de Perthuis [58] and by de Perthuis and Trenkić [56]. These works did not find any improvement in primal attacks on MLWE compared to the corresponding attacks on LWE. More recently, Ogilvie [135] analyzed a code-based hybrid dual attack introduced by Carrier, Meyer-Hilfiger, Shen and Tillich [36], showing that the attack performs slightly better against MLWE than against LWE. Based on this analysis, Ogilvie estimated that the core-SVP hardness of standardized Kyber is approximately 3–4 bits lower than previously believed.

§10 LLL applications

Early applications of lattice basis reduction in cryptography are surveyed by Nguyen and Stern [130]. Lattice-based attacks on the subset sum problem were first devised by Lagarias and Odlyzko [96] and later improved by Coster et al. [51]. Exercise 10.13 uses the Lagarias-Odlyzko attack. In 1997, Ajtai and Dwork [5] introduced a public-key encryption scheme and proved its security based solely on the worst-case hardness of the unique shortest vector problem (uSVP). Shortly thereafter, Nguyen and Stern [129] presented lattice attacks showing that any secure implementation of the Ajtai-Dwork scheme would require prohibitively large keys, rendering it impractical. Also in 1997,

Goldreich, Goldwasser and Halevi [72] proposed another lattice-based public-key encryption scheme. This scheme was broken by Nguyen [123] in 1999, who showed that decryption could be reduced to solving easy instances of the Closest Vector Problem. In 1985, Odlyzko and te Riele [134] used the LLL algorithm to disprove the Mertens conjecture, which posited that $|\sum_{n \le x} \mu(n)| < x^{1/2}$ for $x \ge 2$, where μ denotes the Möbius function. This conjecture was of interest in part because its validity would have implied the Riemann hypothesis.

Lenstra, Lenstra and Lovász [100] were the first to propose the use of lattice basis reduction for discovering integer relations. Borwein and Lisoněk [30] presented a number of examples of relations obtained using integer relation-finding algorithms, including LLL and PSLQ. Håstad, Just, Lagarias and Schnorr [77] introduced an integer relation-finding algorithm, together with a detailed complexity analysis. Lehmer [99] compiled a list of Machin-like formulae. The lattice-based method for finding simultaneous Diophantine approximations described in §10.4 is also due to Lenstra, Lenstra and Lovász [100].

The first lattice-based attack on ECDSA that exploited the partial leakage of per-message secrets was proposed by Howgrave-Graham and Smart [83] and further analyzed by Nguyen and Shparlinski [125]. The exposition of the ECDSA side-channel attack in §10.5 is largely based on the work of De Micheli and Heninger [55], which offers a comprehensive overview of lattice-based attacks on both RSA and elliptic curve cryptosystems. Breitner and Heninger [33] and Rowe, Breitner and Heninger [143] study weak implementations of ECDSA signing in cryptocurrencies, including Bitcoin. Aranha et al. [13] devised a side-channel attack targeting weaknesses in an implementation of the Montgomery ladder used for ECDSA point multiplication. Remarkably, their lattice-based technique succeeded even under extremely limited leakage—an average of less than one bit of information about each per-message secret.

May [110] surveyed applications of the Coppersmith [49] and Howgrave-Graham [81] techniques for finding small solutions to polynomial equations modulo a composite number. Among these applications are an attack by Boneh and Durfee [26] that breaks RSA when the private exponent d has bitlength less than 0.292 times that of the RSA modulus N; an algorithm by Boneh, Durfee and Howgrave-Graham [27] for factoring integers of the form $p^r q$ with large r; Shoup's [150] security proof for RSA-OAEP with encryption exponent $e = 3$; and the deterministic polynomial-time algorithm of Coron and May [50] for factoring an RSA modulus N given the private exponent d. The flaws in specific RSA padding mechanisms described in §10.6.2 and §10.6.3 were demonstrated by Coppersmith [49].

§11 The Number-Theoretic Transform

The Number-Theoretic Transform (NTT) was introduced independently in 1971 by Nicholson [133], Pollard [138] and Schönhage and Strassen [148]. It can be viewed as a finite-field analogue of the Discrete Fourier Transform (DFT), operating on sequences of elements from a finite field rather than on sequences of complex numbers. The ear-

liest use of the NTT in lattice-based cryptography appears in the work of Hoffstein, Pipher and Silverman [79] on NTRU, the first truly practical lattice-based cryptosystem. There, the authors proposed using NTT to accelerate polynomial multiplication in the ring $\mathbb{Z}_q[x]/(x^n - 1)$ underlying NTRU.

The NTT variant described in §11 for polynomial multiplication in $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ is tailored for so-called negacyclic convolution. The NTT used in Dilithium was introduced by Ducas et al. [59]. A complete description, along with in-place algorithms for evaluating the Dilithium NTT and its inverse, is provided in FIPS 204 [119]. The NTT employed in Kyber first appeared in version 2.0 of the Kyber submission to NIST's post-quantum cryptography standardization project. A complete description, along with in-place algorithms for evaluating the Kyber NTT and its inverse, is provided in FIPS 203 [120]. Huang et al. [84] and Zhang, Yan, Huang and Koç [159] report on software implementations of Kyber and Dilithium on ARM and RISC processors.

13 Answers to computational exercises

2.9 $\lambda_1(L) = 2$.

2.10 $\lambda_1(L) = \sqrt{2}$.

3.1 $z = \pm(-1, 4, 3, -1, 1, 0)$, $z = \pm(-5, -4, -4, 3, 4, 4)$, $z = \pm(1, 2, -4, -1, 5, -1)$.

3.2 $z = (-2, -4, -1, -5, 2)$, $z = (-2, 2, 0, 4, 5)$, $z = (0, -3, -4, 3, -5)$
$z = (1, 4, -2, -5, 2)$, $z = (2, -1, -2, -2, 3)$, $z = (3, 5, -5, 3, -5)$.

4.1 $s = (3, 14, 1)$, $e = (-1, 1, -2, 1, 1)$; $s = (9, 47, 25)$, $e = (1, 1, 0, 1, 0)$;
$s = (22, 32, 44)$, $e = (0, 1, -2, 1, 2)$; $s = (24, 11, 45)$, $e = (-1, 1, 0, 1, -2)$;
$s = (28, 12, 15)$, $e = (2, 1, 0, 1, 1)$; $s = (37, 49, 11)$, $e = (-2, 1, -2, 1, 0)$;
$s = (43, 29, 35)$, $e = (0, 1, 0, 1, -1)$; $s = (49, 9, 6)$, $e = (2, 1, 2, 1, -2)$.

4.2 $s = (-3, -2, -1, -1)$.

5.2 The three solutions up to multiplication by ± 1, $\pm x$, $\pm x^2$ are $z = 1 - 8x + 6x^2 + 4x^3$, $z = 10 + 10x - 7x^2 - 5x^3$, and $z = -7 - 7x - 6x^2 + x^3$.

5.5 The unique solution is:

$$s = \begin{bmatrix} 36 + 35x + 26x^2 + 2x^3 \\ 17 + 18x + 20x^2 + 19x^3 \end{bmatrix}, \quad e = \begin{bmatrix} 1 - x \\ 1 - x + x^2 - x^3 \\ -1 - x + x^3 \end{bmatrix}.$$

8.7 (i) $\lambda_1(L) = 1$, $\lambda_2(L) = 2$. (ii) $\lambda_1(L) = \sqrt{5}$, $\lambda_2(L) = \sqrt{17}$.
(iii) $\lambda_1(L) = \sqrt{10146890}$, $\lambda_2(L) = \sqrt{19059138}$.

8.8(d) (i) $12373 = 42^2 + 103^2$. (ii) $555557 = 239^2 + 706^2$. (iii) $44444453 = 2783^2 + 6058^2$.

8.10 The following are $\frac{3}{4}$-reduced LLL bases.
(a) $(0, 0, -1, 1, 0)$, $(1, -1, 0, 1, 1)$, $(0, -1, 0, 0, -2)$, $(0, -1, -2, -1, 1)$, $(3, 1, -1, 0, 0)$.
(b) $(1, 1, 3, -3, 7)$, $(4, 8, -4, 1, 4)$, $(-6, 2, -2, -6, -4)$, $(-3, -1, -5, 5, 6)$, $(3, -5, -10, -8, 3)$.
(c) $(3, 3, -3, 2, 2)$, $(6, -1, 9, -6, 5)$, $(-8, -5, -13, -9, 10)$, $(-14, 6, 8, 19, 11)$,
$(96, -256, -31, 144, 55)$.

8.11 A basis matrix for E_8 is

$$\frac{1}{2}\begin{bmatrix} -2 & 2 & -2 & -2 & -2 & -2 & 1 & -1 \\ 2 & 2 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 2 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 2 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 2 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 2 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}.$$

10.1 $s = (-154, -51, 68, -176)$, $e = (-54, 135, -257, 108, -121, -196)$.

10.2 The MLWE solution is

$$s = \begin{bmatrix} 2x + x^2 + 2x^3 \\ -1 + x - 2x^3 \\ -2 + 2x^2 - x^3 \end{bmatrix}, \quad e = \begin{bmatrix} x \\ -x - 2x^2 - x^3 \\ 2 + 2x - 2x^2 + x^3 \\ -2 + 2x \end{bmatrix}.$$

10.3 Three SIS_2 solutions are $(-6, -15, -6, 5, 10, 4, 14, -5, -3, -8, -6, 1, 15, 1)$,
$(1, -1, -23, 6, -3, 4, 4, -15, -16, 4, -7, -2, -2, 5)$, and
$(4, 9, 10, -1, -8, -21, 8, -3, 3, 5, 5, 11, -3, 16)$.

10.4 One MSIS_2 solution is

$$s = \begin{bmatrix} -5 + 5x - 7x^2 - 27x^3 + 11x^4 \\ 3 - 5x - 6x^2 + 17x^3 + 4x^4 \\ -16 - 13x - 6x^2 + 4x^3 + 10x^4 \\ -16 - 10x - 9x^2 - 5x^3 - x^4 \end{bmatrix}.$$

10.5(b) (i) $x = (-1571, 1529, -1281, 1710)$. (ii) $x = (259, 242, 8, 194, -307)$.
(iii) $x = (-589, -1343, 711, -837, -391, -527)$.

10.6(c) (i) Kannan embedding with $M = 1$ yields $v = (-1571, 1529, -1281, 1710)$.
(ii) Kannan embedding with $M = 3$ yields $v = (275, 192, 38, 281, -282)$.
(iii) Kannan embedding with $M = 1$ yields $v = (-589, -1343, 711, -837, -391, -527)$.

10.8 (a) $x^2 - 2$. (b) $x^4 - 10x^2 + 1$. (c) $x^8 - 40x^6 + 352x^4 - 960x^2 + 576$.
(d) $x^6 - 6x^4 - 6x^3 + 12x^2 - 36x + 1$. (e) $x^{10} - 10x^8 + 40x^6 - 6x^5 - 80x^4 - 120x^3 + 80x^2 - 120x - 23$.

10.9 (a) One solution is $\frac{48}{34}, \frac{59}{34}, \frac{76}{34}, \frac{90}{34}, \frac{113}{34}$.

(b) One solution is $\frac{929}{657}, \frac{1138}{657}, \frac{1469}{657}, \frac{1738}{657}, \frac{2179}{657}, \frac{2369}{657}, \frac{2709}{657}, \frac{2864}{657}, \frac{3151}{657}, \frac{3538}{657}$.

10.10 $K = 9550684$.

10.12 The plaintext is $M =$
22222222222222222333333333333333333333333333444444444444444444
44444444445555555555555555555555555555556666666666666666666666
77777777777777777777777888888888888888888888888.

10.13(c) (i) $s = a_2 + a_4 + a_5 + a_9$. (ii) $s = a_1 + a_3 + a_5 + a_8 + a_{10}$.

10.14(e) (i) $7919 = 5^2 + 37^2 + 42^2 + 69^2$. (ii) $59359 = 10^2 + 67^2 + 77^2 + 221^2$.
(iii) $104743 = 79^2 + 151^2 + 151^2 + 230^2$.

References

[1] D. Aggarwal and N. Stephens-Davidowitz. Just take the average! — An embarrassingly simple 2^n-time algorithm for SVP (and CVP). In *Symposium on Simplicity of Algorithms (SOSA)*, pages 12:1–12:19, 2018.

[2] D. Aharonov and O. Regev. Lattice problems in NP ∩ coNP. *Journal of the ACM*, 52:749–765, 2005.

[3] M. Ajtai. Generating hard instances of lattice problems. In *Symposium on Theory of Computing (STOC)*, pages 99–108, 1996.

[4] M. Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions. In *Symposium on Theory of Computing (STOC)*, pages 10–19, 1998.

[5] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Symposium on Theory of Computing (STOC)*, pages 284–293, 1997.

[6] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Symposium on Theory of Computing (STOC)*, pages 601–610, 2001.

[7] M. Albrecht. Lattice attacks against small-secret parameter choices in HElib and SEAL. In *EUROCRYPT*, pages 103–129, 2017.

[8] M. Albrecht, F. Göpfert, F. Virdia, and T. Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. In *ASIACRYPT*, pages 297–322, 2017.

[9] M. Albrecht, R. Player, and S. Scott. On the concrete hardness of Learning with Errors. *J. Mathematical Cryptology*, 9:169–203, 2015.

[10] A. Alfarano, E. Saxena, E. Wenger, F. Charton, and K. Lauter. Improving ML attacks on LWE with data repetition and stepwise regression, 2026. arXiv:2604.03903.

[11] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange — a new hope. In *USENIX Security*, pages 327–343, 2016.

[12] J. Alwen, S. Krenn, K. Pietrzak, and D. Wichs. Learning with rounding, revisited: New reductions, properties and applications. In *CRYPTO*, pages 57–74, 2013.

[13] D. Aranha, F. Novaes, A. Takahashi, M. Tibouchi, and Y. Yarom. LadderLeak: Breaking ECDSA with less than one bit of nonce leakage. In *ACM Comp. & Comm. Security (CCS)*, pages 225–242, 2020.

[14] S. Arora and R. Ge. New algorithms for learning in presence of errors. In *ICALP*, pages 403–415, 2011.

[15] P. Azevedo-Oliveira, A. Calle Viera, B. Cogliati, and L. Goubin. Uncompressing Dilithium's public key. In *CRYPTO*, pages 417–443, 2025.

[16] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 1:1–13, 1986.

[17] S. Bai and S. Galbraith. An improved compression technique for signatures based on learning with errors. In *The Cryptographers' Track at RSA Conference (CT-RSA)*, pages 28–47, 2014.

[18] A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In *EUROCRYPT*, pages 719–737, 2012.

[19] M. Barbosa, G. Barthe, C. Doczkal, J. Don, S. Fehr, B. Grégoire, Y. Huang, A. Hülsing, Y. Lee, and X. Wu. Fixing and mechanizing the security proof of Fiat-Shamir with aborts and Dilithium. In *CRYPTO*, pages 358–389, 2023.

[20] M. Barbosa, D. Connolly, J. Duarte, A. Kaiser, P. Schwabe, K. Varner, and B. Westerbaan. X-Wing: The hybrid KEM you've been looking for. *IACR Communications in Cryptology*, 1, 2024.

[21] A. Becker, L. Ducas, N. Gama, and T. Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 10–24, 2016.

[22] D. Bernstein, K. Bhargavan, S. Bhasin, A. Chattopadhyay, T. Chia, M. Kannwischer, F. Kiefer, T. Paiva, P. Ravi, and G. Tamvada. KyberSlash: Exploiting secret-dependent division timings in Kyber implementations. *IACR Trans. Cryptographic Hardware and Embedded Systems*, pages 209–234, 2025.

[23] A. Bienstock, L. de Castro, D. Escudero, A. Polychroniadou, and A. Takahashi. Quorus: Efficient, scalable threshold ML-DSA signatures from MPC. In *USENIX Security*, 2026.

[24] J. Blömer and J. Seifert. On the complexity of computing short linearly independent vectors and short bases in a lattice. In *Symposium on Theory of Computing (STOC)*, pages 711–720, 1999.

[25] A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM*, 50(4):506–519, 2003.

[26] D. Boneh and G. Durfee. Cryptanalysis of RSA with private key less than $N^{0.292}$. *IEEE Trans. Info. Theory*, 46:1339–1349, 2000.

[27] D. Boneh, G. Durfee, and N. Howgrave-Graham. Factoring $N = p^r q$ for large r. In *CRYPTO*, pages 326–337, 1999.

[28] A. Bonnecaze and P. Solé. Quaternary constructions of formally self-dual binary codes and unimodular lattices. In *Algebraic Coding*, pages 194–205, 1994.

[29] X. Bonnetain, A. Chailloux, A. Schrottenloher, and Y. Shen. Finding many collisions via reusable quantum walks: Application to lattice sieving. In *EUROCRYPT*, pages 221–251, 2023.

[30] J. Borwein and P. Lisoněk. Applications of integer relation algorithms. *Discrete Mathematics*, 217:65–82, 2000.

[31] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In *IEEE Symp. Security and Privacy*, pages 353–367, 2018.

[32] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory*, 7(3), 2014.

[33] J. Breitner and N. Heninger. Biased nonce sense: Lattice attacks against weak ECDSA signatures in cryptocurrencies. In *Financial Crypto*, pages 3–20, 2019.

[34] A. Budroni, Q. Guo, T. Johansson, E. Mårtensson, and P. Wagner. Making the BKW algorithm practical for LWE. In *INDOCRYPT*, pages 417–439, 2020.

[35] T. Cai, J. Fan, and T. Jiang. Distributions of angles in random packings in spheres. *J. Machine Learning Research*, 14:1837–1864, 2013.

[36] K. Carrier, C. Meyer-Hilfiger, Y. Shen, and J. Tillich. Assessing the impact of a variant of MATZOV's dual attack on Kyber. In *CRYPTO*, pages 444–476, 2025.

[37] D. Cash, D. Hofheinz, E. Kiltz, and C. Peikert. Bonsai trees, or how to delegate a lattice basis. *J. Cryptology*, 25:601–639, 2012.

[38] J. Cassels. *An Introduction to the Geometry of Numbers*. Springer, 1997.

[39] S. Celi, R. del Pino, T. Espitau, G. Niot, and T. Prest. Efficient threshold ML-DSA. In *USENIX Security*, 2026.

[40] S. Chatterjee, N. Koblitz, A. Menezes, and P. Sarkar. Another look at tightness II: Practical issues in cryptography. In *Paradigms in Cryptography (Mycrypt)*, pages 21–55, 2017.

[41] J. Chen, Y. Shabanzadeh, E. Crnčević, T. Hoefler, and D. Alistarh. The geometry of LLM quantization: GPTQ as Babai's nearest plane algorithm, 2025. arXiv:2507.18553.

[42] Y. Chen. *Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe*. PhD thesis, Université Paris Diederot (France), 2017.

[43] Y. Chen and P. Nguyen. BKZ 2.0: Better lattice security estimates. In *ASIACRYPT*, pages 1–20, 2011.

[44] J. Cheon, H. Choe, J. Seo, and H. Seong. SMAUG(-T), revisited: Timing-secure, more compact, less failure. *IEEE Access*, 12:188386–188397, 2024.

[45] J. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT*, pages 409–437, 2017.

[46] H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer, 1993.

[47] H. Cohn, A. Kumar, S. Miller, D. Radchenko, and M. Viazovska. The sphere packing problem in dimension 24. *Annals of Mathematics*, 185:1017–1033, 2017.

[48] J. Conway and N. Sloane. *Sphere Packings, Lattices and Groups*. Springer, 3rd edition, 1999.

[49] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, 10:233–260, 1997.

[50] J. Coron and A. May. Deterministic polynomial-time equivalence of computing the RSA secret key and factoring. *J. Cryptology*, 20:39–50, 2007.

[51] M. Coster, A. Joux, B. LaMacchia, A. Odlyzko, C. Schnorr, and J. Stern. Improved low-density subset sum algorithms. *Computational Complexity*, 2:111–128, 1992.

[52] J. D'Anvers, Q. Guo, T. Johansson, A. Nilsson, F. Vercauteren, and I. Verbauwhede. Decryption failure attacks on IND-CCA secure lattice-based schemes. In *Public Key Cryptography (PKC)*, pages 565–598, 2019.

[53] J. D'Anvers, A. Karmakar, S. Roy, and F. Vercauteren. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In *AFRICACRYPT*, pages 282–305, 2018.

[54] J. D'Anvers, M. Rossi, and F. Virdia. (one) failure is not an option: Bootstrapping the search for failures in lattice-based encryption schemes. In *EUROCRYPT*, pages 3–33, 2020.

[55] G. De Micheli and N. Heninger. Survey: Recovering cryptographic keys from partial information, by example. *IACR Communications in Cryptology*, 1, 2024.

[56] P. de Perthuis and F. Trenkć. Refined modelling of the primal attack, and variants against module-LWE. In *Post-Quantum Cryptography (PQCrypto)*, pages 342–376, 2026.

[57] J. Devevey, P. Fallahpour, A. Passelègue, D. Stehlé, and K. Xagawa. A detailed analysis of Fiat-Shamir with abort. In *CRYPTO*, pages 327–357, 2023.

[58] L. Ducas, L. Engelberts, and P. de Perthuis. Predicting module-lattice reduction. In *ASIACRYPT*, pages 133–166, 2026.

[59] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, and G. Seiler. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptographic Hardware and Embedded Systems*, pages 238–268, 2018.

[60] L. Ducas, L. Pulles, and M. Stevens. Towards a modern LLL implementation. In *ASIACRYPT*, pages 65–99, 2026.

[61] U. Erez and R. Zamir. Achieving $\frac{1}{2}\log(1+\mathrm{SNR})$ on the AWGN channel with lattice encoding and decoding. *IEEE Trans. Info. Theory*, 50:2293–2314, 2004.

[62] P. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Sieler, W. Whyte, and Z. Zhang. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU, 2020. v1.2 of submission to NIST PQC standardization project, available at https://falcon-sign.info/.

[63] P. Fouque, D. Marion, Q. Nguyen, and A. Wallet. Avengers assemble! Supervised learning meets lattice reduction: A single trace attack against CRYSTALS-Kyber key generation. *IACR Trans. Cryptographic Hardware and Embedded Systems*, pages 409–436, 2025.

[64] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO*, pages 537–554, 1999.

[65] S. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012.

[66] N. Gama and P. Nguyen. Predicting lattice reduction. In *EUROCRYPT*, pages 31–51, 2008.

[67] N. Gama, P. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In *EUROCRYPT*, pages 257–278, 2010.

[68] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions, 2007. IACR Cryptology ePrint Archive, https://eprint.iacr.org/2007/432.

[69] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Symposium on Theory of Computing (STOC)*, pages 197–206, 2008.

[70] F. Giacon, F. Heuer, and B. Poettering. KEM combiners. In *Public Key Cryptography (PKC)*, pages 190–218, 2018.

[71] L. Glabush, P. Longa, M. Naehrig, C. Peikert, D. Stebila, and F. Virdia. FrodoKEM: A CCA-secure Learning With Errors key encapsulation mechanism. *IACR Communications in Cryptology*, 2, 2025.

[72] O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. In *CRYPTO*, pages 112–131, 1997.

[73] O. Goldreich, D. Micciancio, S. Safra, and J. Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattices vectors. *Information Processing Letters*, 71:51–61, 1999.

[74] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Attribute-based encryption for circuits. *Journal of the ACM*, 62(6):229–246, 2015.

[75] Q. Guo, T. Johansson, E. Mårtensson, and P. Wagner. Coded-BKW with sieving. In *ASIACRYPT*, pages 323–346, 2017.

[76] G. Hanrot, X. Pujol, and D. Stehlé. Terminating BKZ, 2011. IACR Cryptology ePrint Archive, https://eprint.iacr.org/2011/198.

[77] J. Håstad, B. Just, J. Lagarias, and C. Schnorr. Polynomial time algorithms for finding integer relations among real numbers. *SIAM J. Computing*, 5:859–881, 1989.

[78] J. Hermelink, K. Ning, R. Petri, and E. Strieder. The insecurity of masked comparisons: SCAs on ML-KEM's FO-transform. In *ACM Comp. & Comm. Security (CCS)*, pages 2430–2444, 2024.

[79] J. Hoffstein, J. Pipher, and J. Silverman. NTRU: A ring-based public key cryptosystem. In *Algorithmic Number Theory (ANTS)*, pages 267–288, 1998.

[80] D. Hofheinz, K. Hövelmanns, and E. Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In *Theory of Cryptography Conference (TCC)*, pages 341–371, 2017.

[81] N. Howgrave-Graham. Finding small roots of univariate modular equations revisited. In *Cryptography and Coding*, pages 131–142, 1997.

[82] N. Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In *CRYPTO*, pages 150–169, 2007.

[83] N. Howgrave-Graham and N. Smart. Lattice attacks on digital signature schemes. *Designs, Codes and Cryptography*, 23:283–290, 2001.

[84] J. Huang, A. Adomnicăi, J. Zhang, W. Dai, Y. Liu, C. Cheung, Ç. Koç, and D. Chen. Revisiting Keccak and Dilithium implementations on ARMv7-M. *IACR Trans. Cryptographic Hardware and Embedded Systems*, pages 1–24, 2024.

[85] R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12:415–440, 1987.

[86] R. Kannan and L. Lovász. Covering minima and lattice-point-free convex bodies. *Annals of Mathematics*, 188:577–602, 1988.

[87] S. Khot. Hardness of approximating the shortest vector problem in lattices. *Journal of the ACM*, 52:789–808, 2005.

[88] S. Kim and A. Venkatesh. The behaviour of random reduced bases. *International Mathematics Research Notices*, 2018:6442–6480, 2017.

[89] P. Kirchner and P. Fouque. Improved BKW algorithm for LWE applications to cryptography and lattices. In *CRYPTO*, pages 43–62, 2015.

[90] P. Klein. Finding the closest lattice vector when it's unusually close. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 937–941, 2000.

[91] N. Koblitz, S. Samajder, P. Sarkar, and S. Singha. Concrete analysis of approximate ideal-SIVP to decision ring-LWE reduction. *Advances in Mathematics of Communications*, 18:1216–1258, 2024.

[92] H. Kosuge and K. Xagawa. The security of ML-DSA against fault-injection attacks. In *ASIACRYPT*, pages 641–674, 2025.

[93] E. Krahmer, P. Pessl, G. Land, and T. Güneysu. Correction fault attacks on randomized CRYSTALS-Dilithium. *IACR Trans. Cryptographic Hardware and Embedded Systems*, pages 174–199, 2024.

[94] T. Laarhoven, J. van de Pol, and B. de Weger. Solving hard lattice problems and the security of lattice-based cryptosystems, 2012. IACR Cryptology ePrint Archive, https://eprint.iacr.org/2012/533.

[95] J. Lagarias. Worst-case complexity bounds for algorithms in the theory of integral quadratic forms. *J. Algorithms*, 1:142–186, 1980.

[96] J. Lagarias and A. Odlyzko. Solving low-density subset sum problems. *Journal of the ACM*, 32:229–246, 1985.

[97] A. Langlois and D. Stehlé. Worst-case to average-case reduction for module lattices. *Designs, Codes and Cryptography*, 75:565–599, 2015.

[98] C. Lee, A. Pellet-Mary, D. Stehlé, and A. Wallet. An LLL algorithm for module lattices. In *ASIACRYPT*, pages 59–90, 2019.

[99] D. Lehmer. On arccotangent relations for π. *The American Mathematical Monthly*, 45:657–664, 1938.

[100] A. Lenstra, H. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[101] H. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.

[102] J. Li and P. Nguyen. A complete analysis of the BKZ lattice reduction algorithm. *J. Cryptology*, 38, 2025.

[103] R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. In *The Cryptographers' Track at RSA Conference (CT-RSA)*, pages 319–339, 2011.

[104] Z. Liu, A. Wang, C. Wei, Y. Ding, J. Zhang, A. Liu, and L. Zhu. Release the power of rejected signatures: An efficient side-challenge attack on the ML-DSA cryptosystem. *IEEE Trans. Info. Forensics and Security*, 20:13356–13369, 2025.

[105] V. Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *ASIACRYPT*, pages 598–616, 2009.

[106] V. Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT*, pages 738–755, 2012.

[107] V. Lyubashevsky. Basic lattice cryptography: The concepts behind Kyber (ML-KEM) and Dilithium (ML-DSA), 2024. IACR Cryptology ePrint Archive, https://eprint.iacr.org/2024/1287.

[108] V. Lyubashevsky and D. Micciancio. Generalized compact knapsacks are collision resistant. In *ICALP*, pages 144–155, 2006.

[109] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM*, 60(6), 2013.

[110] A. May. Using LLL-reduction for solving RSA and factorization problems, 2010. Chapter 10 in [131].

[111] A. Menezes. Post-quantum cryptography, 2026. Chapter 15 in *Textbook of Applied Cryptography*, https://sites.google.com/view/textbook-appliedcrypto.

[112] D. Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Trans. Info. Theory*, 47:1212–1215, 2001.

[113] D. Micciancio. Efficient reductions among lattice problems. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 84–93, 2008.

[114] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems: A Cryptographic Perspective*. Springer, 2002.

[115] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM J. Computing*, 37:267–302, 2007.

[116] D. Micciancio and P. Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1468–1480, 2010.

[117] T. Mukherjee and N. Stephens-Davidowitz. Lattice reduction for modules: or how to reduce ModuleSVP to ModuleSVP. In *CRYPTO*, pages 213–242, 2020.

[118] National Institute of Standards and Technology. Recommendation for stateful hash-based signature schemes, 2020. SP 800-208.

[119] National Institute of Standards and Technology. Module-lattice-based digital signature standard, 2024. FIPS 204.

[120] National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard, 2024. FIPS 203.

[121] National Institute of Standards and Technology. Stateless hash-based digital signature standard, 2024. FIPS 205.

[122] National Institute of Standards and Technology. Recommendation for key-encapsulation mechanisms, 2025. SP 800-227.

[123] P. Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from Crypto'97. In *CRYPTO*, pages 299–304, 1999.

[124] P. Nguyen. Hermite's constant and lattice algorithms, 2010. Chapter 2 in [131].

[125] P. Nguyen and I. Shparlinski. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Designs, Codes and Cryptography*, 30:201–217, 2003.

[126] P. Nguyen and D. Stehlé. LLL on the average. In *Algorithmic Number Theory (ANTS)*, pages 238–256, 2006.

[127] P. Nguyen and D. Stehlé. An LLL algorithm with quadratic complexity. *SIAM J. Computing*, 39:874–903, 2009.

[128] P. Nguyen and D. Stehlé. Low-dimensional lattice basis reduction revisited. *ACM Transactions on Algorithms*, 5, 2009.

[129] P. Nguyen and J. Stern. Cryptanalysis of the Ajtai-Dwork cryptosystem. In *CRYPTO*, pages 223–242, 1998.

[130] P. Nguyen and J. Stern. The two faces of lattices in cryptography. In *Cryptography and Lattices*, pages 146–180, 2001.

[131] P. Nguyen and B. Vallée, editors. *The LLL Algorithm: Survey and Applications*. Springer, 2010.

[132] P. Nguyen and T. Vidick. Sieve algorithms for the shortest vector problem are practical. *J. Mathematical Cryptology*, 2:181–207, 2008.

[133] P. Nicholson. Algebraic theory of Finite Fourier Transforms. *J. Computer and Systems Sciences*, 5:524–547, 1971.

[134] A. Odlyzko and H. te Riele. Disproof of the Mertens conjecture. *J. für die Reine und Angewandte Mathematik*, 357:138–160, 1985.

[135] T. Ogilvie. On the concrete hardness gap between MLWE and LWE, 2026. IACR Cryptology ePrint Archive, https://eprint.iacr.org/2026/279.

[136] C. Peikert. A decade of lattice cryptography, 2015. IACR Cryptology ePrint Archive, https://eprint.iacr.org/2015/939.

[137] M. Pohst. A modification of the LLL reduction algorithm. *J. Symbolic Computation*, 4:123–127, 1987.

[138] J. Pollard. The Fast Fourier Transform in a finite field. *Mathematics of Computation*, 25:365–374, 1971.

[139] A. Pouly and Y. Shen. Solving the shortest vector problem in $2^{0.63269n+o(n)}$ time on random lattices. In *EUROCRYPT*, pages 92–123, 2026.

[140] P. Ravi, A. Chattopadhyay, J. D'Anvers, and A. Baksi. Side-channel and fault-injection attacks over lattice-based post-quantum schemes (Kyber, Dilithium): Survey and new results. *ACM Transactions on Embedded Computing Systems*, 23(2), 2024.

[141] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):1–40, 2009.

[142] V. Reis and T. Rothvoss. The subspace flatness conjecture and faster integer programming. In *Symposium on Foundations of Computer Science (FOCS)*, pages 974–988, 2023.

[143] D. Rowe, J. Breitner, and N. Heninger. The curious case of the half-half Bitcoin ECDSA nonces. In *AFRICACRYPT*, pages 273–284, 2023.

[144] C. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Comput. Sci.*, 53:201–224, 1987.

[145] C. Schnorr. Block reduced lattice bases and successive minima. *Combinatorics, Probability and Computing*, 3:507–522, 1994.

[146] C. Schnorr. Lattice reduction by random sampling and birthday methods. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 145–156, 2003.

[147] C. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66:181–199, 1994.

[148] A. Schönhage and V. Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7:281–292, 1971.

[149] T. Shen and X. Li. A survey on classical lattice algorithms. *Cryptography*, 10(2), 2026.

[150] V. Shoup. OAEP reconsidered. *J. Cryptology*, 15:223–249, 2002.

[151] D. Stehlé. Floating-point LLL: Theoretical and practical aspects, 2010. Chapter 5 in [131].

[152] N. Stephens-Davidowitz. Dimension-preserving reductions between lattice problems, 2016. unpublished manuscript.

[153] I. Stewart and D. Tall. *Algebraic Number Theory and Fermat's Last Theorem.* A K Peters, 3rd edition, 2002.

[154] V. Ulitzsch, S. Marzougui, A. Bagia, M. Tibouchi, and J. Seifert. Loop aborts strike back: Defeating fault countermeasures in lattice signatures with ILP. *IACR Trans. Cryptographic Hardware and Embedded Systems*, pages 367–392, 2023.

[155] B. Vallée. Gauss' algorithm revisited. *J. Algorithms*, 12:556–572, 1991.

[156] M. Viazovska. The sphere packing problem in dimension 8. *Annals of Mathematics*, 185:991–1015, 2017.

[157] E. Wenger, E. Saxena, M. Malhou, E. Thieu, and K. Lauter. Benchmarking attacks on Learning with Errors. In *IEEE Symp. Security and Privacy*, pages 279–297, 2025.

[158] M. Yasuda. A survey of solving SVP algorithms and recent strategies for solving the SVP challenge. In *International Symposium on Mathematics, Quantum Theory, and Cryptography*, pages 189–207, 2021.

[159] J. Zhang, Y. Yan, J. Huang, and Ç. Koç. Optimized software implementation of Keccak, Kyber, and Dilithium on RV{32,64}IM{B}{V}. *IACR Trans. Cryptographic Hardware and Embedded Systems*, pages 632–655, 2025.

[160] Z. Zhao and J. Ding. Towards large-scale lattice attack: New lattice records by disk-based sieving. In *EUROCRYPT*, pages 154–182, 2026.

[161] Z. Zhao, J. Ding, and B. Yang. Sieving with streaming memory access. *IACR Trans. Cryptographic Hardware and Embedded Systems*, pages 362–384, 2025.

[162] Y. Zhou, W. Wang, Y. Sun, and Y. Yu. Rejected signatures' challenges pose new challenges: Key recovery of CRYSTALS-Dilithium via side-channel attacks. *IACR Trans. Cryptographic Hardware and Embedded Systems*, pages 817–847, 2025.