IAS/Park City Mathematics Series
Volume 00, Pages 000–000
S 1079-5634(XX)0000-0

An Introduction to Lattices, Lattice Reduction, and Lattice-Based Cryptography

Joseph H. Silverman

Abstract. A lattice is a discrete subgroup of $\mathbb{R}^n$. We will discuss the theory of lattices and describe how they have been used to construct practical public key cryptosystems and digital signature schemes that are, at least presently, secure against attacks by quantum computers.

Contents

2010 *Mathematics Subject Classification.* Primary 94A60; Secondary 11Hxx.
Key words and phrases. lattice, shortest vector problem, closest vector problem, public key cryptosystem, digital signature, lattice-based cryptography.

Acknowledgements. The author would like to thank Mingjie Chen for many helpful suggestions and corrections to these notes, and for her assistance during the PCMI graduate summer school. The author would also like to thank the organizers for inviting him to deliver these lectures.

1. Lattices and Hard Lattice Problems

1.1. Lattices: Definitions, Notation, and Basic Properties.

Definition 1.1.1. A *lattice* L is a discrete subgroup of $\mathbb{R}^n$.

Proposition 1.1.2. *Let* L *be a lattice in* $\mathbb{R}^n$*. Then there exist vectors* $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ *that are* $\mathbb{R}$*-linearly independent and such that*

$$L = \mathbb{Z}\text{-linear span of } \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}.$$

The set $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ *is a* basis for L*, and* k *is the* rank *or* dimension *of* L*.*

Proof. Left for the reader. □

Convention: In these lectures we work almost exclusively with lattices of rank n in $\mathbb{R}^n$, i.e., with lattices of maximal rank. So unless we specify the contrary, we use the convention that[1]

$$\boxed{\textit{lattice} = \textit{lattice of maximal rank.}}$$

Thus when we say that L is a lattice, we mean that L is the $\mathbb{Z}$-linear span of an $\mathbb{R}$-basis $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$ of $\mathbb{R}^n$,

$$L = \{a_1\boldsymbol{v}_1 + a_2\boldsymbol{v}_2 + \cdots + a_n\boldsymbol{v}_n : a_1, a_2, \ldots, a_n \in \mathbb{Z}\}.$$

Important Note: A lattice will have many bases, but some bases are "better" than others.[2] This distinction is a major theme of these lectures.

Definition 1.1.3. Let L be a lattice with basis $\mathcal{B} = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$. The *fundamental domain for* L *associated to* $\mathcal{B}$ is the set[3]

$$\mathcal{F}(\mathcal{B}) = \{t_1\boldsymbol{v}_1 + t_2\boldsymbol{v}_2 + \cdots + t_n\boldsymbol{v}_n : 0 \leqslant t_i < 1\}.$$

The *determinant* (or *volume*) *of* L is[4]

$$\operatorname{Det}(L) = \operatorname{Volume}(\mathcal{F}(\mathcal{B})).$$

See Figure 1.1.4 for an example of a fundamental domain for a lattice in $\mathbb{R}^2$

Proposition 1.1.5. *Let* $L \subset \mathbb{R}^n$ *be a lattice, and let*

$$\mathcal{B} = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\} \quad \textit{and} \quad \mathcal{B}' = \{\boldsymbol{v}_1', \ldots, \boldsymbol{v}_n'\}$$

be bases for L*.*

(1) Let

$$\mathcal{M}(\mathcal{B}) = \begin{pmatrix} \boldsymbol{v}_1 \mid \boldsymbol{v}_2 \mid \cdots \mid \boldsymbol{v}_n \end{pmatrix} \in \operatorname{Mat}_{n\times n}(\mathbb{R})$$

be the matrix whose columns are the vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$*. Then*

$$\operatorname{Det}(L) = \bigl|\det \mathcal{M}(\mathcal{B})\bigr|.$$

---

[1] Lattices of non-maximal rank will come up in our proof that the LLL algroithm runs in polynomial time; see Section 2.7 and Exercise 2.8.2.

[2] Or, as Orwell's Napolean might say, "All bases are created equal, but some are more equal than others."

[3] Note the misuse of terminology, since what we have defined is really a fundamental domain for the quotient space $\mathbb{R}^n/L$. Similarly the "volume" of L is really the volume of the torus $\mathbb{R}^n/L$, which would more properly be called the co-volume of L.

[4] People also frequently work with the *discriminant of* L, which is the square of the volume; i.e., $\operatorname{Disc}(L) = \operatorname{Det}(L)^2$

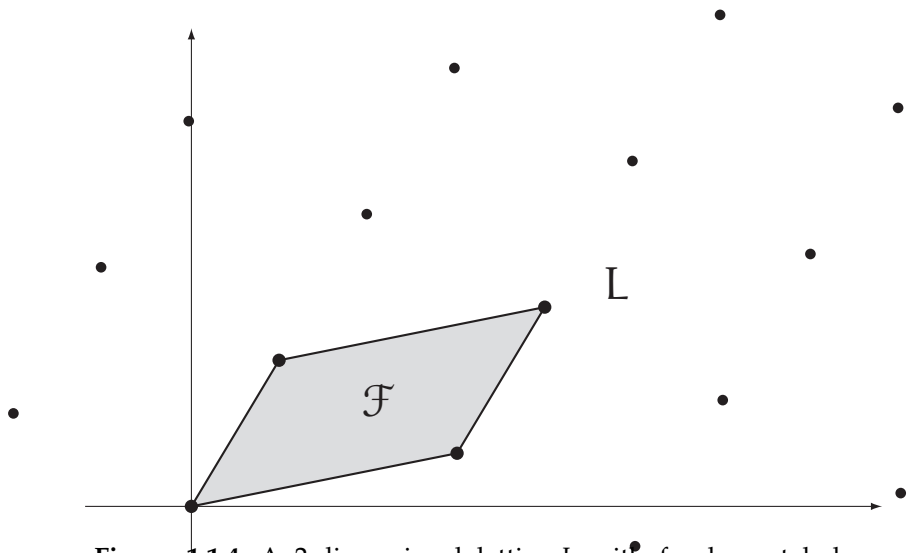


Figure 1.1.4. A 2-dimensional lattice L with fundamental domain $\mathcal{F}$

(2) There is a matrix $A \in SL_n(\mathbb{Z})$ *that transforms* $\mathcal{B}$ *to* $\mathcal{B}'$*,*

$$\mathcal{M}(\mathcal{B}') = A\mathcal{M}(\mathcal{B}).$$

(3) We have

$$\text{Volume}\big(\mathcal{F}(\mathcal{B})\big) = \text{Volume}\big(\mathcal{F}(\mathcal{B}')\big).$$

In particular, $\text{Det}(L)$ *depends only on* L*.*

Proof. Left for the reader. □

1.2. Hard Lattice Problems. The following is one of the most important and fundamental computational problems associated to a lattice.

Definition 1.2.1. Let $L \subset \mathbb{R}^n$ be a lattice. The **Shortest Vector Problem (SVP)** is the problem of finding a shortest non-zero vector in L. In other words, the SVP for L is to find a vector $\mathbf{v}_0 \in L$ satisfying

$$\|\mathbf{v}_0\| = \min_{\mathbf{v} \in L \smallsetminus \mathbf{0}} \|\mathbf{v}\|.$$

Closely related to the SVP is a second important computational lattice problem.

Definition 1.2.2. Let $L \subset \mathbb{R}^n$ be a lattice. The **Closest Vector Problem (CVP)** is the problem of finding, for a given target vector $\mathbf{t} \in \mathbb{R}^n$, a vector in L that is closest to $\mathbf{t}$. In other words, the CVP for L with target vector $\mathbf{t}$ is to find a vector $\mathbf{v}_0 \in L$ satisfying

$$\|\mathbf{v}_0 - \mathbf{t}\| = \min_{\mathbf{v} \in L} \|\mathbf{v} - \mathbf{t}\|.$$

In full generality, the exact version of both the SVP and the CVP appear to be very difficult, although for many applications it suffices to find an approximate solution, as in the following formulation.

Definition 1.2.3. Let $L \subset \mathbb{R}^n$ be a lattice. The **Approximate Closest Vector Problem (apprCVP)** is the problem of finding, for a given target vector $\mathbf{t} \in \mathbb{R}^n$, a vector in L that is "reasonably closes" to $\mathbf{t}$. More precisely, for $\kappa \geqslant 1$, the apprCVP$_\kappa$ is to find a vector $\mathbf{v}_0 \in L$ satisfying

$$\|\mathbf{v}_0 - \mathbf{t}\| \leqslant \kappa \min_{\mathbf{v} \in L} \|\mathbf{v} - \mathbf{t}\|.$$

1.3. Using a Basis to Try to Solve the CVP: Babai's Algorithm. Given a lattice L and a basis $\mathcal{B} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ for L, Babai's Algorithm uses the basis to try to solve the CVP.

Algorithm 1.3.1 (Babai's Algorithm: Intuitive Description).

Input: *A lattice* $L \subset \mathbb{R}^n$.
A basis $\mathcal{B} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ *for* L.
A target vector $\mathbf{t} \in \mathbb{R}^n$.

Compute: *Find a lattice vector* $\tilde{\mathbf{v}}$ *so that the translated fundamental domain* $\mathcal{F}(\mathcal{B}) + \tilde{\mathbf{v}}$ *contains the target point* $\mathbf{t}$.

Output: *The vertex of the closure of* $\mathcal{F}(\mathcal{B}) + \tilde{\mathbf{v}}$ *that is closest to* $\mathbf{t}$ *will be a vector in* L *that one hopes is fairly close to* $\mathbf{t}$.

See Figure 1.3.3 for an illustration in $\mathbb{R}^2$. Our high-level description of Babai's algorithm can be translated into mathematical formulas as follows:[5]

Algorithm 1.3.2 (Babai's Algorithm: Precise Formulation).

Input: *A lattice* $L \subset \mathbb{R}^n$.
A basis $\mathcal{B} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ *for* L.
A target vector $\mathbf{t} \in \mathbb{R}^n$.

Compute: $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$ *so that* $\mathbf{t} = \alpha_1\mathbf{v}_1 + \cdots + \alpha_n\mathbf{v}_n$.
$\mathbf{v}_0 \leftarrow \lfloor \alpha_1 \rceil \mathbf{v}_1 + \cdots + \lfloor \alpha_n \rceil \mathbf{v}_n \in L$.

Output: $\mathbf{v}_0$

Babai's algorithm seems like a very reasonable way to try to solve the apprCVP, and if you believe Figure 1.3.3, it should work quite well in practice. Unfortunately, Figure 1.3.3 is misleading, because we have used a fundamental domain that is fairly rectangular. (It is also misleading because pictures in 2 or 3 dimensions fail to convey the complexity of high-dimensional lattices.) In other words, the vectors in the basis $\mathcal{B}$ used to create the fundamental domain $\mathcal{F}(\mathcal{B})$ are reasonably orthogonal to one another.

What happens if we choose some other basis whose elements are not so orthogonal? Figure 1.3.4 illustrates a lattice with two bases, a "good" somewhat

[5]We write $\lfloor \alpha \rceil$ for the integer k that is closest to α; i.e., the unique $k \in \mathbb{Z}$ satisfying $k - \frac{1}{2} \leqslant \alpha < k + \frac{1}{2}$.

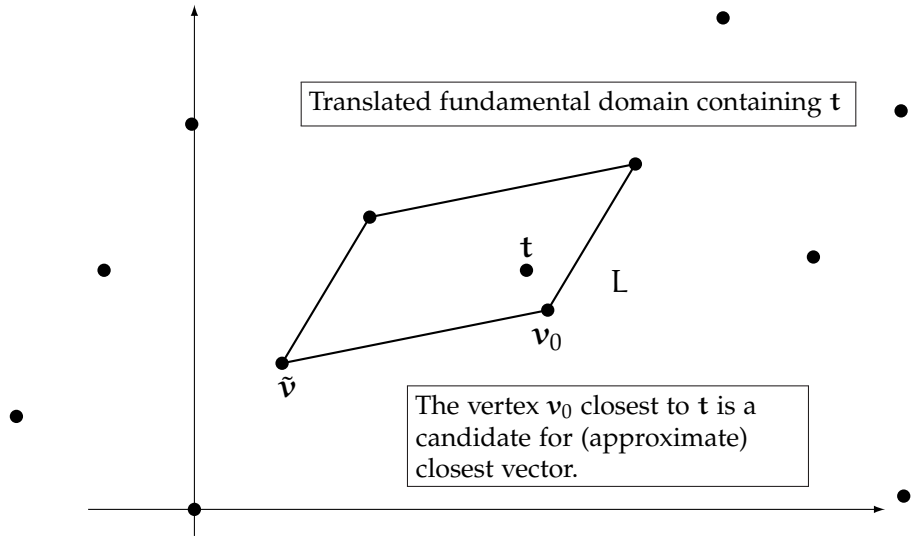


Figure 1.3.3. Babai's Algorithm: Using a Basis to Try to Solve the Closest Vector Problem

orthogonal basis $\{v_1, v_2\}$ and a "bad" highly non-orthogonal basis $\{w_1, w_2\}$. If we try to solve the CVP using the bad basis, Figure 1.3.5 shows what can go wrong. The closest lattice point in the fundamental domain is not the lattice point that is closest to the target. And as the dimension goes up, the failure of a bad basis to solve apprCVP increases exponentially.

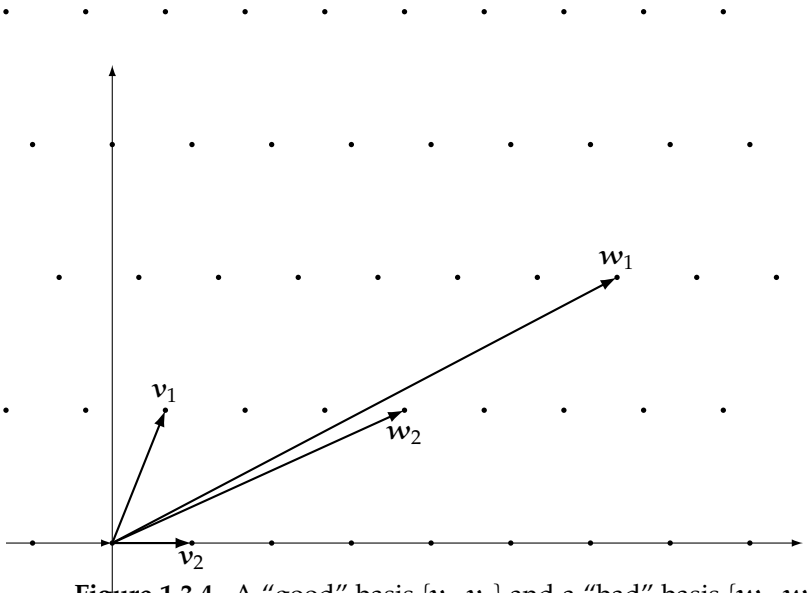


Figure 1.3.4. A "good" basis $\{v_1, v_2\}$ and a "bad" basis $\{w_1, w_2\}$

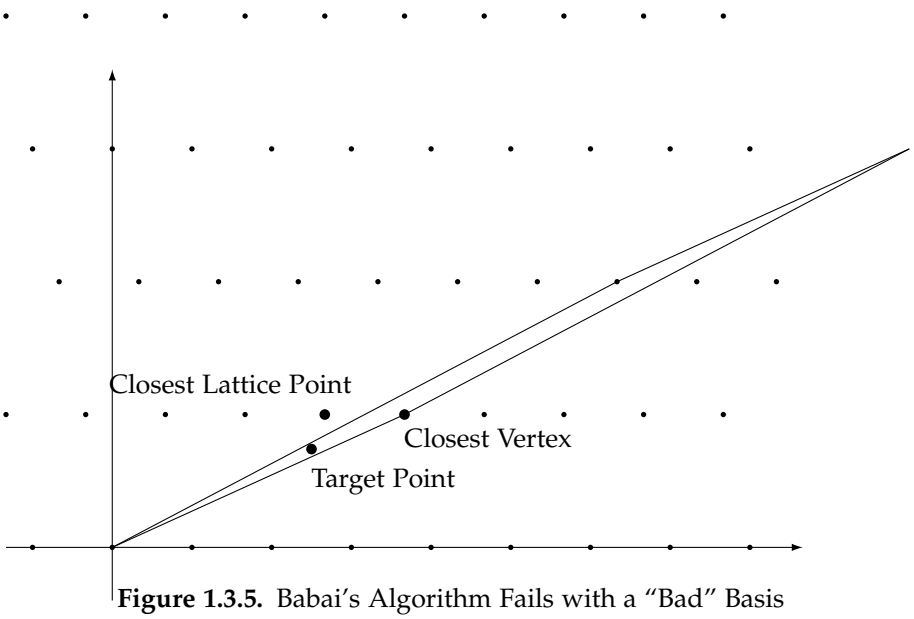


Figure 1.3.5. Babai's Algorithm Fails with a "Bad" Basis

1.4. How to Distinguish Good Bases from Bad Bases. The best possible basis would be one in which the vectors are pairwise orthogonal (perpendicular) to one another. It is not common for a lattice to have an orthogonal basis, although if it does, then its determinant is simply the product of the lengths of the basis vectors.[6] However, even for non-orthogonal bases we get an upper bound.

Theorem 1.4.1 (Hadamard's Inequality)**.** *Let* $\boldsymbol{v}_1,\dots,\boldsymbol{v}_n$ *be any basis for a lattice* L*. Then*

$$\operatorname{Det}(L) \leqslant \|\boldsymbol{v}_1\| \cdot \|\boldsymbol{v}_2\| \cdots \|\boldsymbol{v}_n\|,$$

with equality if and only if $\boldsymbol{v}_1,\dots,\boldsymbol{v}_n$ *are pairwise orthogonal.*

Proof. For $1 \leqslant k \leqslant n-1$, let $\mathcal{H}_k$ be the $\mathbb{R}$-linear subspace spanned by the vectors $\boldsymbol{v}_1,\dots,\boldsymbol{v}_k$, i.e.,

$$\mathcal{H}_k = \{t_1\boldsymbol{v}_1 + t_2\boldsymbol{v}_2 + \cdots + t_k\boldsymbol{v}_k : t_i \in \mathbb{R}\};$$

and let $\theta_k \in [0, \frac{1}{2}\pi]$ be the angle between $\mathcal{H}_k$ and $\boldsymbol{v}_{k+1}$. Then

$$\operatorname{Det}(L) = \operatorname{Volume}\mathcal{F}(\boldsymbol{v}_1,\dots,\boldsymbol{v}_n) = \prod_{k=1}^{n} \|\boldsymbol{v}_k\| \cdot \prod_{k=1}^{n-2} \sin(\theta_k).$$

Since $\bigl|\sin(\theta)\bigr| \leqslant 1$, we immdediately get Hadamard's inequality; and there is equality if and only if every $\theta_k = \frac{1}{2}\pi$, which is equivalent to pairwise orthogonality of $\boldsymbol{v}_1,\dots,\boldsymbol{v}_n$. □

Hadamard's inequality is an equality if and only if the basis vectors are orthogonal to one another. The extent to which it is not an equality measures the

[6] Think rectangular box. We also note that in this special situation, Babai's algorithm solves the CVP.

extent to which the basis is non-orthogonal. A famous theorem of Minkowski says that every lattice has at least one basis that is reasonably orthogonal, where the amount of non-orthogonality is bounded solely in terms of the dimension.[7]

Theorem 1.4.2 (Minkowski)**.** *Fix $n \geqslant 1$. There is a constant γ so that every lattice L of dimension n has a basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ satisfying*

$$\|\boldsymbol{v}_1\| \cdot \|\boldsymbol{v}_2\| \cdots \|\boldsymbol{v}_k\| \leqslant \gamma^{k/2} \operatorname{Det}(L)^{k/n} \quad \text{for all } 1 \leqslant k \leqslant n. \tag{1.4.3}$$

In particular, the inequality (1.4.3) *is true for the constant*[8]

$$\gamma = \frac{4}{\pi}\Gamma\left(\frac{1}{2}n+1\right)^{2/n} \approx \frac{2n}{\pi e}.$$

Proof. See Section 1.7 and Exercise 1.8.8. □

Definition 1.4.4. The *first minimum of* L, denoted $\lambda_1(L)$, is the length of shortest non-zero vector in L,

$$\lambda_1(L) = \inf_{\boldsymbol{v} \in L \smallsetminus \boldsymbol{0}} \|\boldsymbol{v}\|. \tag{1.4.5}$$

More generally, the k*'th successive minimum of* L, denoted $\lambda_k(L)$, is the smallest number such that L contains k vectors that are linearly independent; i.e.,

$$\lambda_k(L) = \inf\Big\{\lambda > 0 : \dim \operatorname{Span}\big\{\boldsymbol{v} \in L : \|\boldsymbol{v}\| \leqslant \lambda\big\} \geqslant k\Big\}.$$

With this notation, Theorem 1.4.2 says that

$$\lambda_1(L)\lambda_2(L)\cdots\lambda_k(L) \leqslant \gamma^{k/2} \operatorname{Det}(L)^{k/n}.$$

And the SVP may be succinctly stated as that of finding a vector $\boldsymbol{v}_0 \in L$ that satisfies $\|\boldsymbol{v}_0\| = \lambda_1(L)$.

Definition 1.4.6. Theorem 1.4.2 says in particular that there is a γ so that for every n-dimensional lattice L, the shortest non-zero vector in L has length at most $\gamma^{1/2} \operatorname{Det}(L)^{1/n}$. The smallest such γ is denoted γ_n and is called *Hermite's constant*; i.e.,

$$\gamma_n := \sup_{\text{Lattices } L \subset \mathbb{R}^n} \lambda_1(L)^2 \cdot \operatorname{Det}(L)^{-2/n}.$$

Remark 1.4.7. Theorem 1.4.2 says that $\gamma_n \lesssim 2n/\pi e$. A refined estimate due to Blichfeldt [5] says that

$$\gamma_n \leqslant \frac{2}{\pi}\Gamma\left(\frac{1}{2}n+2\right)^{2/n} \approx \frac{n}{\pi e}.$$

The exact value of γ_n is known only for $n \leqslant 8$ and $n = 24$:

n	1	2	3	4	5	6	7	8	24
γ_n^n	1	$\frac{4}{3}$	2	4	8	$\frac{64}{3}$	64	2^8	2^{48}

[7] Note that we cannot hope for an inequality of this sort to hold for every basis $\mathcal{B}$, since $\operatorname{Det}(L)$ is independent of $\mathcal{B}$, but we can make the product $\prod \|\boldsymbol{v}_i\|$ arbitrarily large by replacing $\boldsymbol{v}_1$ with $\boldsymbol{v}_1 + C\boldsymbol{v}_2$ and letting $C \in \mathbb{Z}$ go to ∞.

[8] Here $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}\,dt$ is the gamma function. The approximation, which is valid for large n, comes from Stirling's formula $\Gamma(z+1) \approx (z/e)^z \cdot \sqrt{2\pi z}$. As we will see in Sections 1.6 and 1.7, the gamma function appears because the volume of a unit ball in $\mathbb{R}^n$ is $\pi^{n/2}/\Gamma(\frac{1}{2}n+1)$.

As we will see in Section 1.6, for a "random" lattice of dimension n we expect that Theorem 1.4.2 should be true with the even smaller constant $\gamma \approx n/2\pi e$.

1.5. Theory and Practice and a Smidgeon of History. Lattices and the SVP and CVP have been intensively studied for more than 100 years, both as intrinsic mathematical problems and for applications in pure and applied mathematics, physics and cryptography. The theoretical study of lattices is often called the

Geometry of Numbers,

a name bestowed on it by Minkowski in his 1910 book *Geometrie der Zahlen*. Standard references for this subject include [6, 8, 16, 30, 34].

The practical process of finding short(est) or close(st) vectors in lattices is referred to as the theory of

Lattice Reduction.

The shortest vector problem (SVP) and the closest vector problems (CVP) are closely related, since an algorithm that solves the SVP in dimension $n+1$ can generally be used to solve the CVP in dimension n; see Exercise 1.8.11. If the dimension of the lattice L is large, then both the SVP and the CVP seem to be very hard. Indeed, in full generality, the CVP is known to be NP-hard, and the SVP is NP-hard under a randomized reduction hypothesis.

The best lattice reduction methods currently known are based on the LLL Algorithm of Lenstra, Lenstra, and Lovász [23], orginally described in *Mathematische Annalen* **261** (1982), 515-534. The LLL algorithm and some of its variants will be the featured actors in Lecture 2. They are able to find moderately short (or close) lattice vectors in polynomial time, which suffices for many applications. However, finding very short or very close vectors currently requires time that is exponential in the dimension n, making SVP and CVP suitable candidates for cryptographic applications.

Remark 1.5.1. It is worth noting that current lattice reduction algorithms such as LLL are highly sequential. Thus they are not distributable, although they are somewhat parallelizable. Further, there are currently no known quantum algorithms that solve general instances of (approximate) SVP or CVP in anything faster than exponential time.

1.6. The Gaussian Heuristic. Let $L \subset \mathbb{R}^n$ be a "random" lattice, and let $\mathbf{t} \in \mathbb{R}^n$ be a "random" target point. How close might we expect the closest vector of L to be to $\mathbf{t}$? The following heuristic gives an answer that tends to work reasonably well in practice. One might try to rigorously justify this heuristic by defining some sort of probability measure on the space of lattices and targets, but we will not pursue this in these notes.

Heuristic 1.6.1 (Gaussian Heuristic)**.** *Let n be large.*

SVP: *Let $L \subset \mathbb{R}^n$ be a random lattice. Then we expect that the smallest non-zero vector in L will satisfy*

$$\lambda_1(L) = \min_{\mathbf{0}\neq \boldsymbol{v}\in L} \|\boldsymbol{v}\| \approx \sqrt{\frac{n}{2\pi e}}\,\mathrm{Det}(L)^{1/n}.$$

CVP: *Let $L \subset \mathbb{R}^n$ be a random lattice, and let $\mathbf{t} \in \mathbb{R}^n$ be a random target point. Then we expect*

$$\min_{\boldsymbol{v}\in L} \|\boldsymbol{v}-\mathbf{t}\| \approx \sqrt{\frac{n}{2\pi e}}\,\mathrm{Det}(L)^{1/n}.$$

Heuristic "Proof". The proofs of the SVP and CVP versions are similar, so we do the latter. Let

$$\mathbb{B}_R^n(\mathbf{t}) := \big\{\boldsymbol{x} \in \mathbb{R}^n : \|\boldsymbol{x}-\mathbf{t}\| \leqslant R\big\}$$

be the ball of radius R centered at $\mathbf{t}$. The volume of $\mathbb{B}_R^n(\mathbf{t})$ is given by

$$\mathrm{Volume}\big(\mathbb{B}_R^n(\mathbf{t})\big) = \mathrm{Volume}\big(\mathbb{B}_1(\mathbf{0})\big)R^n = \frac{\pi^{n/2}}{\Gamma\big(\frac{n}{2}+1\big)}R^n \approx \left(\frac{2\pi e}{n}\right)^{n/2} R^n.$$

Here the first equality is true by homogeneity, the second equality is a standard calculus exercise using the Gamma function, and the third (approximate) equality is valid for large n using Stirling's formula

$$\Gamma(z+1) \approx \left(\frac{z}{e}\right)^z \cdot \sqrt{2\pi z} \quad \text{as} \quad z \to \infty.$$

On the other hand, we know the following facts:

- A fundamental domain $\mathcal{F}$ for L has volume $\mathrm{Det}(L)$.
- Each tranlated fundamental domain $\mathcal{F}+\boldsymbol{v}$ with $\boldsymbol{v} \in L$ contains exactly one point of L.
- The translated fundamental domains cover $\mathbb{R}^n$.

We thus expect that if we take a random (compact, convex) blob in $\mathbb{R}^n$ whose volume significantly exceeds $\mathrm{Det}(L)$, then that blob is likely to contain a point of L; but if the blob has volume significantly smaller than $\mathrm{Det}(L)$, then it probably won't contain a point of L.

A short computation shows that

$$\mathrm{Volume}\big(\mathbb{B}_R^n(\mathbf{t})\big) \approx \mathrm{Det}(L) \quad\Longleftrightarrow\quad R \approx \sqrt{n/2\pi e}\,\mathrm{Det}(L)^{1/n},$$

which leads us to expect:

$$R > \sqrt{n/2\pi e}\,\mathrm{Det}(L)^{1/n} \quad\Longrightarrow\quad L \cap \mathbb{B}_R^n(\mathbf{t}) \neq \emptyset.$$
$$R < \sqrt{n/2\pi e}\,\mathrm{Det}(L)^{1/n} \quad\Longrightarrow\quad L \cap \mathbb{B}_R^n(\mathbf{t}) = \emptyset.$$

Hence the solution to our CVP problem

$$\min\big\{\|\boldsymbol{v}-\mathbf{t}\| : \boldsymbol{v} \in L\big\}$$

is likely to be roughly equal to $\sqrt{n/2\pi e}\,\mathrm{Det}(L)^{1/n}$. □

Remark 1.6.2. Blichfeldt's upper bound for Hermite's constant (Remark 1.4.7) gives a guaranteed bound for the length $\lambda_1(L)$ of a shortest non-zero vector in L,

while the Gaussian heuristic gives a likely value for $\lambda_1(L)$ in a random lattice. It is instructive to see how they compare.

$$\frac{\begin{pmatrix}\text{Upper bound for } \lambda_1(L) \text{ guaran-}\\ \text{teed by Hermite and Blichfeldt}\end{pmatrix}}{\begin{pmatrix}\text{Expected length of } \lambda_1(L) \text{ sug-}\\ \text{gested by the Gaussian heuristic}\end{pmatrix}} = \frac{\sqrt{\frac{2}{\pi}}\cdot\Gamma\left(2+\frac{n}{2}\right)^{1/n}\operatorname{Det}(L)^{1/n}}{\sqrt{\frac{1}{\pi}}\cdot\Gamma\left(1+\frac{n}{2}\right)^{1/n}\operatorname{Det}(L)^{1/n}} \approx \sqrt{2}.$$

Thus the Gaussian heuristic suggests that the Blichfeldt–Hermite upper bound, which is guaranteed to work for all lattices, is only about 1.4 times larger than what we'd expect to need for most lattices.

1.7. Fundamental Domains: The Good, the Bad, and the Voronoi. Let L be a lattice with a given basis $\mathcal{B}$. We defined the associated fundamental domain $\mathcal{F}(\mathcal{B})$ to be the half-open parallelopiped spanned by the vectors in $\mathcal{B}$; see Definition 1.1.3. An important property of $\mathcal{F}(\mathcal{B})$ is that the natural map $\mathcal{F}(\mathcal{B}) \to \mathbb{R}^n/L$ is a bijection; i.e., every vector in L can be written uniquely as the sum of an element of L and an element of $\mathcal{F}(\mathcal{B})$.

Definition 1.7.1. A (*closed*) *half-plane* in $\mathbb{R}^n$ is a set of the form

$$\bigl\{(x_1,\dots,x_n)\in\mathbb{R}^n : a_0+a_1x_1+\cdots+a_nx_n\leqslant 0\bigr\}$$

for some $a_0,\dots,a_n\in\mathbb{R}$ with $a_1,\dots,a_n$ not all 0. A (*closed*) *polytope* in $\mathbb{R}^n$ is a compact subset $\mathcal{P}\subset\mathbb{R}^n$ that is the intersection of finitely many (closed) half-planes.

Definition 1.7.2. Let $L\subset\mathbb{R}^n$ be a lattice. A (closed polytope) *fundamental domain for* L is a closed polytope $\mathcal{P}\subset\mathbb{R}^n$ such that $\mathcal{P}$ and its interior $\mathcal{P}^\circ$ have the following properties:

$$\text{(1) } \mathcal{P}^\circ\longrightarrow\mathbb{R}^n/L \text{ is injective.} \qquad \text{(2) } \mathcal{P}\longrightarrow\mathbb{R}^n/L \text{ is surjective.}$$

A lattice L has many different fundamental domains having many different shapes. But they all have one thing in common.

Proposition 1.7.3. *Let $L\subset\mathbb{R}^n$ be a lattice. All closed polytope fundamental domains for L have the same volume. This volume is called the* determinant of L *and is denoted* $\operatorname{Det}(L)$.

Proof. Left for the reader. □

The closure of the fundamental domain $\mathcal{F}(\mathcal{B})$ associated to a basis is a closed polytope fundamental domain in the sense of Definition 1.7.2. We now describe another sort of closed polytope fundamental domain whose very definition encodes the closest vector problem.

Definition 1.7.4. Let $L\subset\mathbb{R}^n$ be a lattice, and let $\boldsymbol{v}\in L$. The (*closed*) *Voronoi cell around* $\boldsymbol{v}$ is the set

$$\mathcal{V}(\boldsymbol{v})=\Bigl\{\mathbf{t}\in\mathbb{R}^n : \|\mathbf{t}-\boldsymbol{v}\|\leqslant\|\mathbf{t}-\boldsymbol{w}\| \text{ for all } \boldsymbol{w}\in L\smallsetminus\{\boldsymbol{v}\}\Bigr\}.$$

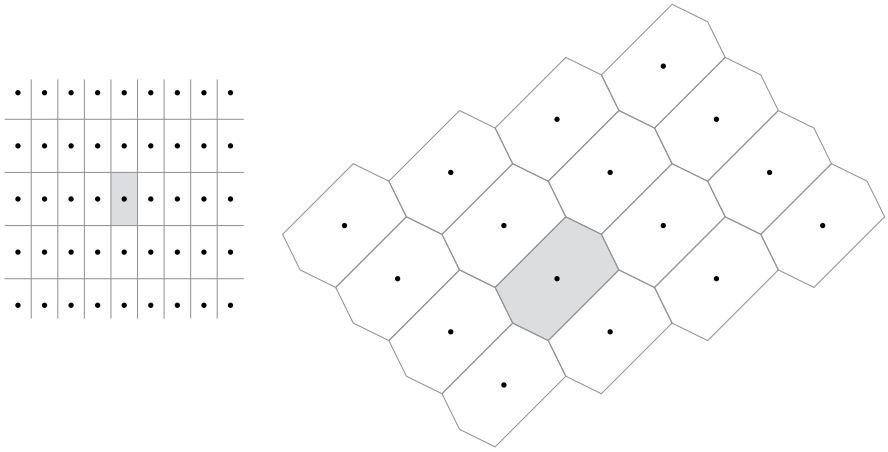


Figure 1.7.6. Two lattices and their Voronoi cells $\mathcal{V}(\boldsymbol{v})$, with one cell shaded

We note that the interior points $\mathbf{t} \in \mathcal{V}(\boldsymbol{v})^\circ$ of the Voronoi cell are exactly the target points $\mathbf{t} \in \mathbb{R}^n$ such that the CVP for $(L, \mathbf{t})$ has $\boldsymbol{v}$ as its unique solution, while target points on the boundary $\mathbf{t} \in \partial\mathcal{V}(\boldsymbol{v})$ have the property that the CVP for $(L, \mathbf{t})$ has more than one solution.

Remark 1.7.5. The boundary of a Voronoi cell is a union of finitely many $(n-1)$-dimensional polytopes lying in hyperplanes in $\mathbb{R}^n$. More concretely, let $\boldsymbol{v}, \boldsymbol{w} \in L$ be distinct lattice points whose Voronoi cells share a common boundary. The set of points that are equidistant from $\boldsymbol{v}$ and $\boldsymbol{w}$ form the hyperplane $H(\boldsymbol{v}, \boldsymbol{w})$ in $\mathbb{R}^n$ that perpendicularly bisects the line segment from $\boldsymbol{v}_1$ to $\boldsymbol{v}_2$, and the common boundary of the two Voronoi cells is a subset of this hyperplane,

$$\mathcal{V}(\boldsymbol{v}) \cap \mathcal{V}(\boldsymbol{w}) \subset H(\boldsymbol{v}, \boldsymbol{w}).$$

Figure 1.7.6 illustrates the Voronoi cells for two different lattices in $\mathbb{R}^2$. If you imagine the line segement connecting lattice points in adjacent cells, you will see that the line segment is perpendicular to the common boundary of the cells.

Proposition 1.7.7. *Let $L \subset \mathbb{R}^n$ be a lattice, and let $\boldsymbol{v} \in L$. The Voronoi cell $\mathcal{V}(\boldsymbol{v})$ containing $\boldsymbol{v}$ is a fundamental domain for L.*

Proof. First suppose that $\mathbf{t}_1, \mathbf{t}_2 \in \mathcal{V}(\boldsymbol{v})^\circ$ have the same image in $\mathbb{R}^n/L$. Our goal is to prove that $\mathbf{t}_1 = \mathbf{t}_2$, which will show that $\mathcal{V}(\boldsymbol{v})^\circ \to \mathbb{R}^n/L$ is injective. The assumption says that

$$\mathbf{t}_1 - \mathbf{t}_2 = \mathbf{u} \quad \text{for some } \mathbf{u} \in L.$$

We suppose that $\mathbf{u} \neq \mathbf{0}$ and use the following calculation to derive a contradiction:

$$\begin{aligned}
\|\mathbf{t}_1 - \boldsymbol{v}\| &< \min_{\boldsymbol{w}\in L\smallsetminus\{\boldsymbol{v}\}} \|\mathbf{t}_1 - \boldsymbol{w}\| && \text{since } \mathbf{t}_1 \in \mathcal{V}(\boldsymbol{v})^\circ,\\
&\leqslant \big\|\mathbf{t}_1 - (\mathbf{u}+\boldsymbol{v})\big\| && \text{since we are assuming that } \mathbf{u}\neq 0,\\
&&& \text{so we can take } \boldsymbol{w} = \mathbf{u}+\boldsymbol{v},\\
&= \|\mathbf{t}_2 - \boldsymbol{v}\| && \text{since } \mathbf{t}_1 - \mathbf{t}_2 = \mathbf{u},\\
&= \min_{\boldsymbol{w}\in L} \|\mathbf{t}_2 - \boldsymbol{w}\| && \text{since } \mathbf{t}_2 \in \mathcal{V}(\boldsymbol{v})^\circ,\\
&= \min_{\boldsymbol{w}\in L} \big\|\mathbf{t}_1 - (\mathbf{u}+\boldsymbol{w})\big\| && \text{since } \mathbf{t}_1 - \mathbf{t}_2 = \mathbf{u},\\
&= \min_{\boldsymbol{w}\in L} \|\mathbf{t}_1 - \boldsymbol{w}\| && \text{since } \mathbf{u}\in L, \text{ so min is the same},\\
&= \|\mathbf{t}_1 - \boldsymbol{v}\| && \text{since } \mathbf{t}_1 \in \mathcal{V}(\boldsymbol{v})^\circ.
\end{aligned}$$

This strict inequality is a contradiction, which proves that $\mathbf{u} = \mathbf{0}$, and thus $\mathbf{t}_1 = \mathbf{t}_2$.

It remain to show that $\mathcal{V}(\boldsymbol{v}) \to \mathbb{R}^n/L$ is surjective. Let $\widehat{\mathbf{t}} \in \mathbb{R}^n/L$ be an arbitrary point, and let $\widehat{U} \subset \mathbb{R}^n/L$ be an arbitrary (small) open neighborhood of $\widehat{\mathbf{t}}$. Since the map $\mathbb{R}^n \to \mathbb{R}^n/L$ is a local isomorphism (this uses the fact that L is discrete), we can find an open set $U \subset \mathbb{R}^n$ and a point $\mathbf{t} \in \mathbb{R}^n$ such that $\mathbf{t} \mapsto \widehat{\mathbf{t}}$ and such that the map $U \to \widehat{U}$ is a bijection.

As explained in Remark 1.7.5, the boundaries of the Voronoi cells in $\mathbb{R}^n$ are pieces of hyperplanes, so they cannot cover the open subset U. Hence there is a point $\mathbf{t}' \in U$ lying in some (open) Voronoi cell $\mathcal{V}(\boldsymbol{w})^\circ$. But then

$$\mathbf{t}' - \boldsymbol{w} + \boldsymbol{v} \in \mathcal{V}(\boldsymbol{v})^\circ,$$

and hence

$$\widehat{\mathbf{t}}' = (\text{image of } \mathbf{t}' \text{ in } \mathbb{R}^n/L) \in \widehat{U}.$$

To recapitulate, we have shown that for every $\widehat{\mathbf{t}} \in \mathbb{R}^n/L$ and every open neighborhood $\widehat{U} \subset \mathbb{R}^n/L$ of $\widehat{\mathbf{t}}$, the image of $\mathcal{V}(\mathbf{0})^\circ$ in $\mathbb{R}^n/L$ has a point in common with $\widehat{U}$. Hence the image of $\mathcal{V}(\mathbf{0})^\circ$ is dense in $\mathbb{R}^n/L$, so its closure is all of $\mathbb{R}^n/L$. □

We can use Voronoi cells to prove Theorem 1.4.2 for $k = 1$, thereby proving a fundamental upper bound for the shortest non-zero vectors in a lattice.

Proof of Theorem 1.4.2 *for* $k = 1$. We are given a lattice $L \subset \mathbb{R}^n$, and we want to show that there is a vector $\boldsymbol{v}_1 \in L$ satisfying

$$\|\boldsymbol{v}_1\| \leqslant \frac{2}{\sqrt{\pi}}\Gamma\left(\frac{1}{2}n+1\right)^{1/n} \operatorname{Det}(L)^{1/n}.$$

We let

$$\lambda = \lambda_1(L) = \min\big\{\|\boldsymbol{w}\| : \boldsymbol{w} \in L,\ \boldsymbol{w} \neq 0\big\}$$

be the length of the shortest non-zero vectors in L. We claim that the open ball

$$B_{\lambda/2}(\mathbf{0}) = \big\{\mathbf{t} \in \mathbb{R}^n : \|\mathbf{t}\| < \lambda/2\big\}$$

is entirely contained within the Voronoi cell $\mathcal{V}(\mathbf{0})$ around $\mathbf{0}$. To see why, suppose that

$$\mathbf{t} \in \mathrm{B}_{\lambda/2}(\mathbf{0}) \cap \mathcal{V}(\boldsymbol{v}) \quad \text{for some } \boldsymbol{v} \in \mathrm{L}.$$

Then

$$\begin{aligned}
\|\boldsymbol{v}\| &\leqslant \|\mathbf{t}\| + \|\mathbf{t}-\boldsymbol{v}\| && \text{triangle inequality,}\\
&= \|\mathbf{t}\| + \min_{\boldsymbol{w}\in\mathrm{L}} \|\mathbf{t}-\boldsymbol{w}\| && \text{since } \mathbf{t} \in \mathcal{V}(\boldsymbol{v}),\\
&\leqslant \|\mathbf{t}\| + \|\mathbf{t}\| && \text{taking } \boldsymbol{w} = \mathbf{0},\\
&< \lambda && \text{since } \mathbf{t} \in \mathrm{B}_{\lambda/2}(\mathbf{0}).
\end{aligned}$$

This strict inequality and the definition of λ tell us that $\boldsymbol{v} = \mathbf{0}$. Hence the ball $\mathrm{B}_{\lambda/2}(\mathbf{0})$ is contained in the closure of the Voronoi cell $\mathcal{V}(\mathbf{0})$.

To recapitulate, we have proven that

$$\mathrm{B}_{\lambda/2}(\mathbf{0}) \subseteq \mathcal{V}(\mathbf{0}). \tag{1.7.8}$$

Taking volumes yields

$$\begin{aligned}
\mathrm{Det}(\mathrm{L}) &= \mathrm{Volume}\big(\mathcal{V}(\mathbf{0})\big) && \text{from Proposition 1.7.3,}\\
&\geqslant \mathrm{Volume}\big(\mathrm{B}_{\lambda/2}(\mathbf{0})\big) && \text{from (1.7.8),}\\
&= (\lambda/2)^n \mathrm{Volume}\big(\mathrm{B}_1(\mathbf{0})\big).
\end{aligned}$$

Solving for λ yields the desired inequality,

$$\min_{\boldsymbol{w}\in\mathrm{L},\,\boldsymbol{w}\neq 0} \|\boldsymbol{w}\| = \lambda \leqslant \underbrace{2\mathrm{Volume}\big(\mathrm{B}_1(\mathbf{0})\big)^{1/n}}_{\text{This quantity depends only on } n.} \mathrm{Disc}(\mathrm{L})^{1/n}.$$

Hence if we set

$$\gamma = 2\mathrm{Volume}\big(\mathrm{B}_1(\mathbf{0})\big)^{1/n} = \frac{4}{\pi}\Gamma\left(\frac{1}{2}n+1\right)^{2/n} \approx \frac{2n}{\pi e},$$

then we have proven that every lattice $\mathrm{L} \subset \mathbb{R}^n$ contains a non-zero vector of length at most $\gamma\,\mathrm{Disc}(\mathrm{L})^{1/n}$. □

1.8. Exercises for Lecture 1.

Exercise 1.8.1. Prove Proposition 1.1.2, which says that every lattice (discrete subgroup) of $\mathbb{R}^n$ has a finite $\mathbb{Z}$-basis.

Exercise 1.8.2. Prove Proposition 1.1.5 which says that the value of $\mathrm{Det}(\mathrm{L})$ does not depend on the choice of a basis for L.

Exercise 1.8.3. Let $\mathrm{L} \subset \mathbb{R}^n$ be a lattice, let $\mathcal{B} = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$, let $A \in \mathrm{Mat}_{n\times n}(\mathbb{Z})$ be a matrix with $\det(A) \neq 0$, and let L' be the lattice spanned by the column vectors of $\mathcal{M}(\mathcal{B})A$.

(1) Prove that L' is a sublattice of L and satisfies

$$\mathrm{L}/\mathrm{L}' \cong \mathbb{Z}^n/A\mathbb{Z}^n \quad \text{and} \quad (\mathrm{L} : \mathrm{L}') = |\det A|.$$

(2) Prove that every sublattice $\mathrm{L}' \subseteq \mathrm{L}$ of finite index arises in this way.

Exercise 1.8.4. Write a computer program implementing Babai's algorithm and use it to attempt to solve the following CVP problems.

(1) The lattice $L \subset \mathbb{R}^4$ is spanned by

$$\boldsymbol{v}_1 = (1,-1,0,1),\ \boldsymbol{v}_2 = (0,-1,-1,1),\ \boldsymbol{v}_3 = (2,-1,1,1),\ \boldsymbol{v}_4 = (1,-1,2,0),$$

and the target vector $\mathbf{t} = (11,23,9,15)$.

(2) The lattice $L' \subset \mathbb{R}^4$ is spanned by

$$\boldsymbol{v}_1' = [-4,3,0,5],\ \boldsymbol{v}_2' = [-12,3,-3,7],\ \boldsymbol{v}_3' = [-6,5,-3,3],\ \boldsymbol{v}_4' = [0,7,-1,1],$$

and the target vector $\mathbf{t} = (11,23,9,15)$.

(3) Let $\boldsymbol{w} \in L$ and $\boldsymbol{w}' \in L'$ be the lattice vectors in (1) and (2) that Babai's algorithm says are close to the target vector $\mathbf{t}$. Compute the distances

$$\|\boldsymbol{w} - \mathbf{t}\| \quad \text{and} \quad \|\boldsymbol{w}' - \mathbf{t}\|.$$

(4) Prove that the lattice L and L' are in fact the same lattice, and use (3) to deduce that Babai's algorithm using the good basis in (1) gives a better solution to the CVP for L and $\mathbf{t}$ than using the bad basis in (2).

Exercise 1.8.5. Let L be a lattice, and let $\mathcal{B} = \{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\}$ be a basis for L.

(1) Suppose that the vectors in $\mathcal{B}$ are pairwise orthogonal; i.e., $\boldsymbol{v}_i \cdot \boldsymbol{v}_j = 0$ for all $i \neq j$. Prove that Babai's algorithm solves the CVP for every target vector.

(2) Prove the converse; i.e., if Babai's algorithm solves the CVP for every target vector, then $\mathcal{B}$ is an orthogonal basis.

Exercise 1.8.6. Compare Blichfeldt's upper bound for Hermite's constant with the actual values for $1 \leqslant n \leqslant 8$ and $n = 24$, as listed in Remark 1.4.7.

Exercise 1.8.7. Prove Proposition 1.7.3, which says that all fundamental domains for a lattice L have the same volume.

Exercise 1.8.8. Prove Theorem 1.4.2 for all k. (This is a challenging problem!)

Exercise 1.8.9. Let $L \subset \mathbb{R}^n$ be a lattice, and let $\mathcal{B} = \{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\}$ be a basis for L. The *orthogonality defect* of $\mathcal{B}$ is the ratio

$$\mathrm{OD}(\mathcal{B}) := \frac{\|\boldsymbol{v}_1\| \cdot \|\boldsymbol{v}_2\| \cdots \|\boldsymbol{v}_n\|}{\mathrm{Det}(L)}.$$

Hadamard's inequality (Theorem 1.4.1) says that $\mathrm{OD}(\mathcal{B}) \geqslant 1$, with equality if and only if $\mathcal{B}$ is an orthogonal basis, so the size of $\mathrm{OD}(\mathcal{B})$ is a measure of the non-orthogonality of the basis. We define

$$\kappa_n := \sup_{L \subset \mathbb{R}^n} \inf_{\mathcal{B} \text{ basis for } L} \mathrm{OD}(\mathcal{B}).$$

(1) Prove that κ_n is finite. What explicit uppper bound for κ_n is provided by Minkowski's theorem (Theorem 1.4.2)? How about using Blichfeldt's estimate (Remark 1.4.7)?

(2) Compute the orthogonality defects of the two bases for the lattice $L = L'$ in Exercise 1.8.4. How do they compare with the estimate for κ_4 in (1)?

Exercise 1.8.10. Let $\mathbb{B}_R^n \subset \mathbb{R}^n$ be a ball of radius R.

(1) Prove that the volume of $\mathbb{B}_R^n$ is given by the formula

$$\text{Volume}(\mathbb{B}_R^n) = \text{Volume}\big(\{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| \leqslant R\}\big) = \frac{\pi^{n/2}}{\Gamma\big(\frac{1}{2}n+1\big)} R^n,$$

where Γ is the *Gamma function*

$$\Gamma(z) = \int_0^\infty t^z e^{-t} \frac{dt}{t}.$$

(2) Prove that the Gamma function satisfies $\Gamma(z+1) = z\Gamma(z)$, so in particular if m is an integer, then $\Gamma(m+1) = m!$.

(3) Use

$$(\textbf{Stirling's Formula}) \qquad \Gamma(z+1) \approx \left(\frac{z}{e}\right)^z \cdot \sqrt{2\pi z} \text{ as } z \to \infty,$$

to prove that

$$\text{Volume}(\mathbb{B}_R^n) \approx \left(\frac{2\pi e}{n}\right)^{n/2} R^n.$$

Exercise 1.8.11. Let L be a lattice of dimension n, and let $\mathbf{t} \in \mathbb{R}^n$ be a target vector. Suppose that you have an algorithm that solves SVP in dimension $n+1$. This exercise explains how to use your SVP solver to try to solve the given CVP.

For each $\delta > 0$, we form a new lattice L_δ in dimension $n+1$ by setting

$$\boldsymbol{v}_1' = (\boldsymbol{v}_1, 0), \ldots, \boldsymbol{v}_n' = (\boldsymbol{v}_n, 0), \mathbf{t}' = (\mathbf{t}, \delta)$$

and taking the span,

$$L_\delta := \text{Span}(\boldsymbol{v}_1', \ldots, \boldsymbol{v}_n', \mathbf{t}') \subset \mathbb{R}^{n+1}.$$

(1) Prove that $\lambda_1(L_\delta) \to 0$ as $\delta \to 0$.

(2) Prove that either $\lambda_1(L_\delta) \to \lambda_1(L)$ as $\delta \to \infty$, or else that some multiple of $\mathbf{t}$ is in L.

(3) Explain why there should be some value of δ for which a smallest non-zero vector in L_δ has the form

$$m_1\boldsymbol{v}_1' + \cdots + m_n\boldsymbol{v}_n' \pm \mathbf{t}',$$

in which case $\pm(m_1\boldsymbol{v}_1 + \cdots + m_n\boldsymbol{v}_n)$ is a point in L that is close (closest?) to $\mathbf{t}$.

(4) Suppose instead that the SVP solver applied to L_δ returns the vector $m_1\boldsymbol{v}_1' + \cdots + m_n\boldsymbol{v}_n' + m_0\mathbf{t}'$ with $|m_0| \neq 1$. If $m_0 = 0$, should we increase or decrease δ before trying again? Same question if $|m_0| \geqslant 2$?

(5) Prove that $\text{Det}(L_\delta) = \text{Det}(L) \cdot \delta$.

(6) Use the Gaussian heuristic to show that a good starting value for δ is a solution to the equation

$$\frac{n+1}{2\pi e} \cdot \text{Det}(L)^{2/(n+1)} \cdot \delta^{2/(n+1)} = \frac{n}{2\pi e} \cdot \text{Det}(L)^{2/n} + \delta^2.$$

2. Lattice Reduction: The Practical Problem of Solving Hard Lattice Problems

2.1. Introduction. We recall the two fundamental lattice problems:

> **Shortest Vector Problem** (SVP)
> Find a shortest non-zero vector in L.
> **Closest Vector Problem** (CVP)
> Find a vector in L closest to a target $\mathbf{t}$.

As explained in Exercise 1.8.11, an algorithm to solve the SVP can generally be adapted to solve the CVP in one lower dimension, and we have seen that Babai's method (Section 1.3) can solve apprCVP if it is given a good basis; i.e., a basis whose vectors are reasonably orthogonal. In this lecture we thus concentrate on the computational problem of constructing a good basis for a given lattice.

Suppose that L is a lattice and that we are given a basis $\mathcal{B} = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$ for L. The Gram–Schmidt algorithm, which we now recall, transforms $\mathcal{B}$ into an orthogonal basis $\mathcal{B}^* = \{\boldsymbol{v}_1^*, \ldots, \boldsymbol{v}_n^*\}$ for $\mathbb{R}^n$.

Algorithm 2.1.1 (Gram–Schmidt Algorithm).

$$\boldsymbol{v}_1^* \leftarrow \boldsymbol{v}_1 \quad \textit{and} \quad \boldsymbol{v}_k^* \leftarrow \boldsymbol{v}_k - \sum_{i=1}^{k-1} \frac{\boldsymbol{v}_k \cdot \boldsymbol{v}_i^*}{\|\boldsymbol{v}_i^*\|^2} \boldsymbol{v}_i^* \quad \textit{for } 2 \leqslant k \leqslant n.$$

$$\textbf{Intuition:} \quad \boldsymbol{v}_k^* \leftarrow \text{Projection of } \boldsymbol{v}_k \text{ onto } \operatorname{Span}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k-1})^{\perp}.$$

Thus $\boldsymbol{v}_i^*$ is a vector created using $\boldsymbol{v}_i$ and making the largest possible angle with the span of the previous vectors.

Unfortunately, the vectors $\boldsymbol{v}_1^*, \ldots, \boldsymbol{v}_n^*$ will probably not be in L, due to those pesky $\|\boldsymbol{v}_i^*\|^2$ factors in the denominators. If we want to do a Gram–Schmidt type of process and end up with vectors in L, we can try rounding the coefficients.

Algorithm 2.1.2 (Gram–Schmidt Algorithm with Rounding).

$$\boldsymbol{v}_1^* \leftarrow \boldsymbol{v}_1 \quad \textit{and} \quad \boldsymbol{v}_k^* \leftarrow \boldsymbol{v}_k - \sum_{i=1}^{k-1} \left\lfloor \frac{\boldsymbol{v}_k \cdot \boldsymbol{v}_i^*}{\|\boldsymbol{v}_i^*\|^2} \right\rceil \boldsymbol{v}_i^* \quad \textit{for } 2 \leqslant k \leqslant n.$$

Algorithm 2.1.2 creates a basis $\{\boldsymbol{v}_1^*, \ldots, \boldsymbol{v}_n^*\}$ for L, and this new basis might be more orthogonal than the original basis. On the other hand, it might not.

It is worth observing that the output from Algorithm 2.1.2 depends on the order in which the original basis vectors are fed into the algorithm. In particular, it's usually best to take $\boldsymbol{v}_1$ to be the smallest of the original vectors. More generally, sorting the original basis by size and then feeding the vectors from small to large into the algorithm is likely to give a better output than feeding them in large to small.

The LLL algorithm intertwines Algorithm 2.1.2 (Gram–Schmidt with Rounding) with the operation of swapping pairs of input vectors, leading to a process in

which one moves up and down the list of basis vectors as they're being modified. Remarkably, with appropriate parameters, one can prove that the LLL algorithm is fast (polynomial time) and that it produces a basis that is pretty good. Further, there is an extension of LLL called LLL-BKZ in which one modifies larger blocks instead of swapping pairs. LLL-BKZ can produce better bases than LLL, albeit at the cost of added processing time.

In this lecture we describe the LLL algorithm, sketch the proof that it produces a reasonably good basis in polynomial time, and explain the idea behind LLL-BKZ. We start with dimension 2, where lattice reduction is very fast.

2.2. Lattice Reduction in Dimension 2. The following method of lattice reduction for 2-dimensional lattices is essentially due to Gauss.

> **Gauss Lattice Reduction in Dimension 2: Intuition**
> Alternately subtract multiples of one basis vector from the other until no further improvement is possible.

This leads to the following algorithm.

Algorithm 2.2.1 (Lattice Reduction in Dimension 2).

[1] Input: A basis $\{\boldsymbol{v}_1, \boldsymbol{v}_2\}$ for a lattice $L \subset \mathbb{R}^2$
[2] If $\|\boldsymbol{v}_2\| < \|\boldsymbol{v}_1\|$, then swap $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. [Swap Step]
[3] $m \leftarrow \boldsymbol{v}_1 \cdot \boldsymbol{v}_2 / \|\boldsymbol{v}_1\|^2$.
[4] If $\lfloor m \rceil = 0$
[5] Output $\{\boldsymbol{v}_1, \boldsymbol{v}_2\}$
[6] Else
[7] $\boldsymbol{v}_2 \leftarrow \boldsymbol{v}_2 - \lfloor m \rceil \boldsymbol{v}_1$ [Size Reduction Step]
[8] Go To Step 2
[9] End If

Remark 2.2.2. If we remove the closest integer sign in Step 7 of Algorithm 2.2.1, then $\boldsymbol{v}_2 - m\boldsymbol{v}_1$ is actually orthogonal to $\boldsymbol{v}_1$. So Step 7 does the best that it can, subject to the requirement that the new $\boldsymbol{v}_2$ must be in L.

Theorem 2.2.3. *The output $\{\boldsymbol{v}_1, \boldsymbol{v}_2\}$ from Algorithm 2.2.1 has the following properties:*

(1) $\boldsymbol{v}_1$ is a solution to the SVP for L.
(2) The angle between $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ satisfies $\dfrac{\pi}{3} \leqslant \theta(\boldsymbol{v}_1, \boldsymbol{v}_2) \leqslant \dfrac{2\pi}{3}$.

Proof. Left for the reader. □

Size reduction (Step 7 in Algorithm 2.2.1) is illustrated in Figure 2.2.4. In this picture, the quantity $m = \boldsymbol{v}_1 \cdot \boldsymbol{v}_2 / \|\boldsymbol{v}_1\|^2$ is the exact factor that makes

$$\boldsymbol{v}_2 - m\boldsymbol{v}_1 \text{ perpendicular to } \boldsymbol{v}_1,$$

so

$$\boldsymbol{v}_2^* = \boldsymbol{v}_2 - m\boldsymbol{v}_1 = \text{the projection of } \boldsymbol{v}_2 \text{ onto } \boldsymbol{v}_1^{\perp}.$$

Rounding m to the nearest integer, we have

$$\lfloor m \rceil \neq 0 \quad \Longrightarrow \quad \boldsymbol{v}_2 - \lfloor m \rceil \boldsymbol{v}_1 \text{ is more orthogonal to } \boldsymbol{v}_1 \text{ than is } \boldsymbol{v}_2.$$

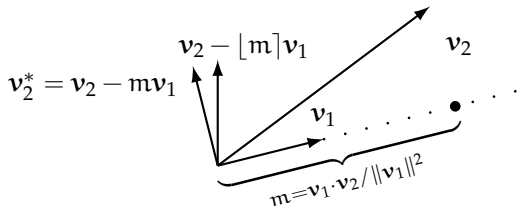


Figure 2.2.4. Size Reduction for Dimension 2 Lattice Reduction

2.3. The Size, Quasi-Orthogonality, and Lovász Conditions. If some coefficient in the Gram–Schmidt process satisfies

$$\frac{|\boldsymbol{v}_i \cdot \boldsymbol{v}_j^*|}{\|\boldsymbol{v}_j^*\|^2} > \frac{1}{2},$$

then replacing $\boldsymbol{v}_i$ by

$$\boldsymbol{v}_i - a\boldsymbol{v}_j \quad \text{for an appropriate } a \in \mathbb{Z}$$

makes the coefficient smaller, thereby increasing the angle. So Algorithm 2.1.2 (Gram–Schmidt with Rounding) helps until the basis has the following property.

Definition 2.3.1. An ordered basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ for L satisfies the *Size Condition* if the output from the Gram–Schmidt algorithm (Algorithm 2.1.1) satisfies:

$$\text{(2.3.2)} \qquad \text{Size Condition:} \qquad \frac{|\boldsymbol{v}_i \cdot \boldsymbol{v}_j^*|}{\|\boldsymbol{v}_j^*\|^2} \leqslant \frac{1}{2} \quad \text{for all } 1 \leqslant j < i \leqslant n.$$

On the other hand, we wamt the basis vectors to be at least somewhat orthogonal to one another. So we might ask that they also have the following property.

Definition 2.3.3. An ordered basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ for L satisfies the *Quasi-Orthogonality Condition* if the output from the Gram–Schmidt algorithm (Algorithm 2.1.1) satisfies:

$$\text{(2.3.4)} \qquad \text{Quasi-Orthogonality Condition:} \qquad \|\boldsymbol{v}_{i+1}^*\| \geqslant \frac{\sqrt{3}}{2}\|\boldsymbol{v}_i^*\| \quad \text{for all } 1 \leqslant i \leqslant n-1.$$

An elaboration of the proof of Theorem 1.4.2 can be used to prove the following result.

Theorem 2.3.5. *Every lattice has a basis satisfying both the Size Condition* (2.3.2) *and the Quasi-Orthogonality Condition* (2.3.4)*.*

Proof. Left for the reader. □

Unfortunately, the best known algorithms to find such a basis take time that is exponential in the dimension. The innovation of the LLL algorithm is to relax the Quasi-Orthogonality Condition.

Definition 2.3.6. An ordered basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ for L satisfies the *Lovász Condition* if the output from the Gram–Schmidt algorithm (Algorithm 2.1.1) satisfies:

$$\text{(2.3.7)} \qquad \text{Lovász Condition:} \qquad \|\boldsymbol{v}_{i+1}^*\|^2 \geqslant \left(\frac{3}{4} - \frac{|\boldsymbol{v}_{i+1} \cdot \boldsymbol{v}_i^*|^2}{\|\boldsymbol{v}_i^*\|^4}\right) \|\boldsymbol{v}_i^*\|^2.$$

Although this looks complicated, it has a natural geometrical description. The Lovász Condition says that

$$\begin{aligned} &\text{Projection of } \boldsymbol{v}_{i+1} \text{ onto } \operatorname{Span}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{i-1})^{\perp} \\ &\qquad \geqslant \frac{3}{4} \cdot \text{Projection of } \boldsymbol{v}_i \text{ onto } \operatorname{Span}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{i-1})^{\perp}. \end{aligned}$$

Definition 2.3.8. An ordered basis for L satisfying both the Size Condition (2.3.2) and the Lovász Condition (2.3.7) is called an *LLL-Reduced Basis* for L.

Theorem 2.3.9. *Let* $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$ *be an LLL-reduced basis for* L.

(1) $\|\boldsymbol{v}_1\| \leqslant 2^{(n-1)/2} \lambda_1(L)$.

(2) $\displaystyle\prod_{i=1}^{n} \|\boldsymbol{v}_i\| \leqslant 2^{n(n-1)/4} \operatorname{Det}(L)$.

Proof. See Section 2.6 for a sketch of the proof. □

Remark 2.3.10. How does an LLL-reduced basis stack up agianst the highly reduced basies that we expect from Theorem 1.4.2 or from the Gaussian heuristic? The earlier results say that every lattice L has a basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ satisfying

$$\prod_{i=1}^{n} \|\boldsymbol{v}_i\| \lessapprox n^{n/2} \operatorname{Det}(L) \asymp 2^{\frac{1}{2} n \log_2 n} \operatorname{Det}(L).$$

So an LLL-reduced basis more-or-less has an extra n in the exponent. This is typical. All known lattice reduction algorithms either take exponential time, or else they give vectors whose norms are exponentially worse than optimal in terms of the dimension.

2.4. The Basic LLL Algorithm. The LLL lattice reduction algorithm is presented in Figure 2.4.2. We have opted to give a non-optimized version that has the advantage of being easier to understand and analyze.

Theorem 2.4.3 (Lenstra, Lenstra, Lovász [23])**.** *Algorithm* 2.4.1 *in Figure* 2.4.2 *finds an LLL-reduced basis for* L *and has running time that is polynomial-time in* $n = \dim(L)$. *More precisely if the initial basis is* $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$ *and if we let* $B = \max \|\boldsymbol{v}_i\|$, *then the Swap Step* (*Step* 14) *is executed at most* $O(n^2 \log B)$ *times.*

Proof. See Section 2.7 for a sketch of the proof. □

Remark 2.4.4. Observations for the LLL Algorithm (Algorithm 2.4.1).

Algorithm 2.4.1.

[1] Input: A basis $\{\boldsymbol{v}_1,\dots,\boldsymbol{v}_n\}$ for a lattice L
[2] $k \leftarrow 2$
[3] Loop while $k \leqslant n$
[4] Loop Down $j = k-1, k-2, \dots, 2, 1$
[5] $\boldsymbol{v}_1^*,\dots,\boldsymbol{v}_k^* \leftarrow \boldsymbol{v}_1,\dots,\boldsymbol{v}_k$ [Gram–Schmidt (2.1.1)]
[6] $\mu_{k,j} \leftarrow (\boldsymbol{v}_k \cdot \boldsymbol{v}_j^*)/\|\boldsymbol{v}_j^*\|^2$
[7] $\boldsymbol{v}_k \leftarrow \boldsymbol{v}_k - \lfloor \mu_{k,j} \rceil \boldsymbol{v}_j$ [Size Reduction]
[8] End j Loop
[9] $\boldsymbol{v}_1^*,\dots,\boldsymbol{v}_k^* \leftarrow \boldsymbol{v}_1,\dots,\boldsymbol{v}_k$ [Gram–Schmidt (2.1.1)]
[10] $\mu_{k,k-1} \leftarrow (\boldsymbol{v}_k \cdot \boldsymbol{v}_{k-1}^*)/\|\boldsymbol{v}_{k-1}^*\|^2$
[11] If $\|\boldsymbol{v}_k^*\|^2 \geqslant \left(\frac{3}{4} - \mu_{k,k-1}^2\right)\|\boldsymbol{v}_{k-1}^*\|^2$ [Lovász Condition]
[12] $k \leftarrow k+1$
[13] Else
[14] Swap $\boldsymbol{v}_{k-1}$ and $\boldsymbol{v}_k$ [Swap Step]
[15] Set $k \leftarrow \max(k-1, 2)$
[16] End If
[17] End k Loop
[18] Output: The LLL-reduced basis $\{\boldsymbol{v}_1,\dots,\boldsymbol{v}_n\}$

Figure 2.4.2. The LLL Lattice Reduction Algorithm (Unoptimized)

- At the end of the loop in Steps 4–8, the vectors $\boldsymbol{v}_1,\dots,\boldsymbol{v}_k$ satisfy the size condition (2.3.2).
- At Step 11, we already know that $\boldsymbol{v}_1,\dots,\boldsymbol{v}_{k-1}$ satisfy the Lovász condition, so we just need to check if the Lovász condition is true when we adjoin $\boldsymbol{v}_k$ to the list. If it is, we can go on to $\boldsymbol{v}_{k+1}$; but if not, then in some sense $\boldsymbol{v}_k$ is "better" than $\boldsymbol{v}_{k-1}$, so we swap them. But that means that we have to reconsider the new list of values $\boldsymbol{v}_1,\dots,\boldsymbol{v}_{k-1}$.
- The big outer k-loop from Steps 3 to 17 ends when $k = n$, at which point the list of vectors $\boldsymbol{v}_1,\dots,\boldsymbol{v}_n$ has been size reduced and satisfies the Lováscz condition, so it forms an LLL-reduced basis.
- We reiterate that Step 12 helps us by incrementing k, but that there is a countervailing force in Step 15, the Swap Step, that hurts us by decrementing k.
- Later in Section 2.7 we prove that Steps 14 and 15 are executed only finitely many times, and that the number of executions is bounded by a polynomial in n. Thus LLL runs in time that is polynomial in n.

Remark 2.4.5. The LLL algorithm finds an LLL-reduced basis, so Theorem 2.3.9 tells us that LLL always finds a non-zero vector $\boldsymbol{v}$ in L that is no more than $2^{(n-2)/2}$

times as long as the shortest non-zero vector. In practice, LLL generally does better than this. But also in practice, if n is large, then $\|\boldsymbol{v}\|/\lambda_1(L)$ will be quite large.

2.5. Variants and Improvements to LLL. Many methods of improving LLL have been proposed over the years. Often they sacrifice provable polynomial time performance for improved operation on most lattices. One of the most important, which we briefly discuss, replaces the Swap Step with a more complicated procedure.

Definition 2.5.1. A basis $\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\}$ is said to be a **Korkine–Zolotareff (KZ) Reduced Basis** if it satisfies both the Size Condition and the following:

For all $1 \leqslant i \leqslant n$, the vector $\boldsymbol{v}_i^*$ is the shortest non-zero vector in the projection of L onto $\mathrm{Span}(\boldsymbol{v}_1, \dots, \boldsymbol{v}_i)$.

There are no known subexponential algorithm for creating a KZ-reduced basis for all of L. Indeed, the first vector in a KZ-reduced basis for L is a shortest non-zero vector in L. So instead we use KZ-reduced bases for small sublattices of L.

Theorem 2.5.2 (LLL-BKZ Algorithm with Blocksize β)**.** (Schnorr–Euchner [32,33]) *Rather than swapping $\boldsymbol{v}_k$ and $\boldsymbol{v}_{k-1}$ in Step* 14 *of LLL* (*Algorithm* 2.4.1)*, we instead take the sublattice spanned by a block of vectors $\boldsymbol{v}_i, \boldsymbol{v}_{i+1}, \dots, \boldsymbol{v}_{i+\beta-1}$ and replace these vectors with a KZ-reduced basis for the sublattice. The LLL-BKZ algorithm terminates in no more than $O(\beta^a n^b)$ steps (for some small constants a and b) and finds a non-zero vector $\boldsymbol{v} \in L$ satisfying*

$$\|\boldsymbol{v}\| \leqslant \left(\frac{\beta}{\pi e}\right)^{\frac{n-1}{\beta-1}} \lambda_1(L).$$

Proof. We omit the proof. □

Remark 2.5.3. Thus LLL-BKZ solves apprSVP to within an approximation factor roughly equal to $\beta^{n/\beta}$, as compared to the approximation factor $2^{n/2}$ guaranteed by LLL, but at the cost of a running time that is exponential in β. For example, if we want to solve apprSVP with an approximation factor n^d for some $d > 0$, then we need to take $\beta^{n/\beta} \approx n^d$, which forces $\beta \approx n/d$ and a running time that is exponential in n.

Remark 2.5.4. As a practical matter, it is currently possible to run BKZ-LLL with blocksizes in the 50 to 100 range, but definitely impractical to use blocksize 500. There has been a considerable amount of research, both theoretical and experimental, devoted to estimating both the time required to run BKZ-LLL with blocksize β and the expected approximation factor for the shortest non-zero vector $\boldsymbol{v}$ returned by the algorithm. We give some examples of the sorts of results that appear in the literature, and point the reader to papers such as [2,7,28,31,35] for further information.

First, how long does it take to run BKZ-β, which is our name for BKZ-LLL run using blocksize β? Conservative estimates currently suggest that running BKZ-β takes at least $2^{0.292\beta}$ operations on a classical computer and at least $2^{0.265\beta}$ operations a quantum computer.[9] Second, how good is the output from BKZ-β. This is normally measured in terms of the Hermite root factor.

Definition 2.5.5. The *Hermite root factor for BKZ-β applied to the n-dimensional lattice* L is the quantity

$$\delta(\text{BKZ-}\beta, L) = \left\{ \frac{\left(\begin{array}{c}\text{the length } \|\boldsymbol{v}\| \text{ of the shortest}\\ \text{non-zero vector found by BKZ-}\beta\end{array}\right)}{\operatorname{Det}(L)^{1/n}} \right\}^{1/(n-1)}.$$

Expermients and a geometric series heuristic suggest that for a fixed blocksize $\beta \geqslant 50$ and a random lattice $L \subset \mathbb{R}^n$ with n significantly larger than β, a reasonable estimate for the Hermite root factor for BKZ-β is given by

$$\delta(\text{BKZ-}\beta, L) \approx \left(\frac{\beta}{2\pi e} \cdot (\beta\pi)^{1/\beta} \right)^{1/(2\beta-2)}.$$

An interesting alternative method for estimating $\delta(\text{BKZ-}\beta, L)$ is the BKZ-simulator of Chen and Nguyen [7, Algorithm 2]. The input to their simulator is the list of the logarithms of the lengths of the lattice basis vectors converted into Hermite normal form.[10] However, they assume that the lattice is random in some suitable, but not precisely specified, sense. As the preceding material indicates, there is much room for further research on the operating characteristics of lattice reduction algorithms.

Remark 2.5.6. Many lattice-based cryptographic algorithms rely on hiding one or more small vectors in a lattice. What this means is that the lattice L has a non-zero vector $\boldsymbol{v}$ whose length $\|\boldsymbol{v}\|$ is significantly smaller than the shortest non-zero length predicted by the Gaussian heuristic described in Section 1.6. For example, suppose that the quantities

$$\overbrace{\lambda_1(L) := \min_{\boldsymbol{0}\neq\boldsymbol{v}\in L} \|\boldsymbol{v}\|}^{\text{actual shortest vector length}} \text{ and } \overbrace{\gamma(L) := \sqrt{n/2\pi e} \cdot (\operatorname{Det} L)^{1/n}}^{\text{expected shortest vector length}} \text{ satisfy } \lambda_1(L) \lessapprox n^{-1/2}\gamma(L).$$

It turns out that if $\lambda_1(L)$ is exponentially smaller than $\gamma(L)$, then lattice reduction algorithms such as LLL-BKZ have a fairly easy time solving the SVP; but if $\lambda_1(L)$ is only polynomially smaller than $\gamma(L)$, then it takes exponential time to solve SVP. Heuristic arguments combined with experiments suggest that for a random

[9] We will not make the notion of "operation" precise, but we note that in 2022, the world's fastest computers operate at roughly 1 exaFLOPS, where FLOPS stands for "floating point operation per second" and "exa" means 10^{18}. So in terms of floating point operations, the current fastest computers in the world can perform roughly $2^{59.8}$ operations per second. This explains why BKZ-β with $\beta = 50$ is feasible, but $\beta = 500$ is not.

[10] An invertible square matrix is in column Hermite normal form if it is lower triangular and the diagonal entries satisfy certain size comparison properties.

lattice with a hidden small vector, BKZ-β will find the small vector if its Hermite root factor satisfies either:

$$\lambda_1(L) \leqslant 3\sqrt{N/2\pi e} \cdot \delta(\text{BKZ-}\beta, L)^{-n} \cdot \operatorname{Det}(L)^{1/n}, \qquad \text{[15, Section 2.2.1];}$$
$$\lambda_1(L) \leqslant \sqrt{n/\beta} \cdot \delta(\text{BKZ-}\beta, L)^{-(n-2\beta)} \cdot \operatorname{Det}(L)^{1/n}, \qquad \text{[3, 4].}$$

As these very different estimates suggest, we still have a long way to go in understanding the behavior of lattice reduction algorithms for lattices containing a vector that is moderately shorter than expected.

2.6. LLL Bases Are Nice: Proof Sketch.

Proof Sketch of Theorem 2.3.9. We prove the second estimate, and leave the first for you; see Exercise 2.8.4. So our goal is to prove that an LLL-reduced basis satisfies

$$\prod_{i=1}^{n} \|\boldsymbol{v}_i\| \leqslant 2^{n(n-1)/4} \operatorname{Det}(L).$$

The Lovász condition and $|\mu_{i,i-1}| \leqslant \frac{1}{2}$ give

$$\|\boldsymbol{v}_i^*\|^2 \geqslant \left(\frac{3}{4} - \mu_{i,i-1}^2\right) \|\boldsymbol{v}_{i-1}^*\|^2 \geqslant \frac{1}{2}\|\boldsymbol{v}_{i-1}^*\|^2.$$

Applying repeatedly yields the useful inequality

$$\|\boldsymbol{v}_j^*\|^2 \leqslant 2^{i-j}\|\boldsymbol{v}_i^*\|^2.$$

We now compute

$$\begin{aligned}
\|\boldsymbol{v}_i\|^2 &= \left\|\boldsymbol{v}_i^* + \sum_{j=1}^{i-1} \mu_{i,j}\boldsymbol{v}_j^*\right\|^2 && \text{from Gram–Schmidt,}\\
&= \|\boldsymbol{v}_i^*\|^2 + \sum_{j=1}^{i-1} \mu_{i,j}^2 \|\boldsymbol{v}_j^*\|^2 && \text{since } \boldsymbol{v}_1^*, \dots, \boldsymbol{v}_n^* \text{ are orthogonal,}\\
&\leqslant \|\boldsymbol{v}_i^*\|^2 + \sum_{j=1}^{i-1} \frac{2^{i-j}\|\boldsymbol{v}_i^*\|^2}{4} && \text{since } |\mu_{i,j}| \leqslant \tfrac{1}{2} \text{ and } \|\boldsymbol{v}_j^*\|^2 \leqslant 2^{i-j}\|\boldsymbol{v}_i^*\|^2,\\
&= \frac{1+2^{i-1}}{2}\|\boldsymbol{v}_i^*\|^2\\
&\leqslant 2^{i-1}\|\boldsymbol{v}_i^*\|^2.
\end{aligned}$$

Multiplying over $1 \leqslant i \leqslant n$ yields

$$\prod_{i=1}^{n} \|\boldsymbol{v}_i\|^2 \leqslant \prod_{i=1}^{n} 2^{i-1}\|\boldsymbol{v}_i^*\|^2 = 2^{n(n-1)/2}\left(\prod_{i=1}^{n} \|\boldsymbol{v}_i^*\|\right)^2 = 2^{n(n-1)/2} \operatorname{Det}(L)^2.$$

□

2.7. LLL Runs in Polynomial Time: Proof Sketch.

Proof Sketch of Theorem 2.4.3. Our goal is to prove that LLL terminates in time that is polynomial in the dimension n. For ease of exposition, we assume that $L \subseteq \mathbb{Z}^n$.

The idea of the proof is to define a function

$$D : \{\text{bases } \boldsymbol{v}_1, \dots, \boldsymbol{v}_n \text{ for } L\} \longrightarrow \mathbb{R}_{\geqslant 1}$$

that measures the "complexity" of a basis and to prove that D has the following property for Lovász swap step:

$$\begin{pmatrix}\text{Lovász condi-}\\ \text{tion is false}\end{pmatrix} \implies D\begin{pmatrix}\text{basis after}\\ \text{swap step}\end{pmatrix} \leqslant \frac{\sqrt{3}}{2} \cdot D\begin{pmatrix}\text{basis before}\\ \text{swap step}\end{pmatrix}. \tag{2.7.1}$$

Applying this repeatedly, we conclude that

$$D\begin{pmatrix}\text{basis after swap step has}\\ \text{been executed } k \text{ times}\end{pmatrix} \leqslant \left(\frac{\sqrt{3}}{2}\right)^k D\begin{pmatrix}\text{original}\\ \text{basis}\end{pmatrix}.$$

But $D(\mathcal{B}) \geqslant 1$ for any basis, so

$$\begin{pmatrix}\text{number of times that}\\ \text{the swap step is executed}\end{pmatrix} \leqslant \frac{\log D(\text{original basis})}{\log(2/\sqrt{3}\,)}.$$

It remains to define a function D satisfying (2.7.1) and to prove that D of the original basis is $O(n^2 \log B)$.

Definition 2.7.2. For any basis $\mathcal{B} = \{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\}$ of L, we define a sequence of sublattices

$$L_\ell := \operatorname{Span}_{\mathbb{Z}}\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_\ell\}, \quad 1 \leqslant \ell \leqslant n.$$

We note that L_ℓ is a lattice of rank ℓ sitting in $\mathbb{Z}^n$. It follows that its absolute determinant satisfies

$$\operatorname{Det}(L_\ell)^2 \in \mathbb{N}, \quad \text{and hence in particular that} \quad \operatorname{Det}(L_\ell) \geqslant 1.$$

(See Exercise 2.8.2.) We define the *complexity of the ordered basis* $\mathcal{B}$ to be the quantity

$$D(\mathcal{B}) := \prod_{\ell=1}^{n} \operatorname{Det}(L_\ell).$$

Let $\boldsymbol{v}_1^*, \dots, \boldsymbol{v}_n^*$ be the orthogonal Gram–Schmidt basis of $\mathbb{R}^n$ associated to $\mathcal{B}$; i.e., the output of applying Algorithm 2.1.1 to $\mathcal{B}$. Then

$$\operatorname{Det}(L_\ell) = \prod_{i=1}^{\ell} \|\boldsymbol{v}_i^*\|, \quad \text{and hence} \quad D(\mathcal{B}) = \prod_{i=1}^{n} \|\boldsymbol{v}_i^*\|^{n+1-i}.$$

Since each $\operatorname{Det}(L_\ell) \geqslant 1$, we find that

$$1 \leqslant D(\mathcal{B}) \leqslant \left(\max_{1\leqslant i\leqslant n} \log \|\boldsymbol{v}_i^*\|\right)^{\frac{n(n+1)}{2}} \leqslant \left(\max_{\boldsymbol{v}\in\mathcal{B}} \log \|\boldsymbol{v}\|\right)^{\frac{n(n+1)}{2}} = B^{\frac{n(n+1)}{2}}.$$

Hence in order to complete the proof that LLL terminates in at most $O(n^2 \log B)$ steps, we need to prove the Lovász swap property described in (2.7.1).

We start with the assumption:

The Lovászc Condition is false at iteration k.

This implies that

$$\|\boldsymbol{\nu}_k^*\| < \sqrt{\frac{3}{4} - \mu_{k,k-1}^2} \cdot \|\boldsymbol{\nu}_{k-1}^*\| \leqslant \frac{\sqrt{3}}{2}\|\boldsymbol{\nu}_{k-1}^*\|.$$

When we perform the swap step, i.e., when we swap $\boldsymbol{\nu}_k$ and $\boldsymbol{\nu}_{k-1}$, the basis of L_{k-1} changes from $\boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_{k-2}, \boldsymbol{\nu}_{k-1}$ to $\boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_{k-2}, \boldsymbol{\nu}_k$. This changes the determinant as follows:

$$\begin{aligned}
\operatorname{Det}(L_{k-1}^{\text{new}}) &= \|\boldsymbol{\nu}_1^*\| \cdot \|\boldsymbol{\nu}_2^*\| \cdots \|\boldsymbol{\nu}_{k-2}^*\| \cdot \|\boldsymbol{\nu}_k^*\| \\
&= \|\boldsymbol{\nu}_1^*\| \cdot \|\boldsymbol{\nu}_2^*\| \cdots \|\boldsymbol{\nu}_{k-2}^*\| \cdot \|\boldsymbol{\nu}_{k-1}^*\| \cdot \frac{\|\boldsymbol{\nu}_k^*\|}{\|\boldsymbol{\nu}_{k-1}^*\|} \\
&= \operatorname{Det}(L_{k-1}^{\text{old}}) \cdot \frac{\|\boldsymbol{\nu}_k^*\|}{\|\boldsymbol{\nu}_{k-1}^*\|} \\
&\leqslant \frac{\sqrt{3}}{2} \operatorname{Det}(L_{k-1}^{\text{old}}).
\end{aligned}$$

The swap only affects the basis of L_{k-1}; the bases of all of the other L_i are unchanged. Hence the complexity of the new swapped basis for L is related to the complexity of the old unswapped basis by

$$\begin{aligned}
D(\mathcal{B}^{\text{new}}) &= \left(\prod_{i \neq k-1} \operatorname{Det}(L_i^{\text{old}}) \right) \cdot \operatorname{Det}(L_{k-1}^{\text{new}}) \\
&\leqslant \left(\prod_{i \neq k-1} \operatorname{Det}(L_i^{\text{old}}) \right) \cdot \frac{\sqrt{3}}{2} \operatorname{Det}(L_{k-1}^{\text{old}}) \\
&= \frac{\sqrt{3}}{2} D(\mathcal{B}^{\text{old}}).
\end{aligned}$$

This completes the proof of (2.7.1), and with it, the proof of Theorem 2.4.3. □

2.8. Exercises for Lecture 2.

Exercise 2.8.1. Prove Theorem 2.2.3, which says that the basis of a 2-dimensional lattice computed by the algorithm described in Algorithm 2.2.1 solves SVP and gives a second basis vector that is fairly orthogonal to the first basis vector .

Exercise 2.8.2. Let $\mathcal{B} = \{\boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_\ell\} \subset \mathbb{R}^n$ be a linearly independent set of vectors, and let

$$L_{\mathcal{B}} = \operatorname{Span}_{\mathbb{Z}}(\boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_\ell)$$

be the lattice that they span. N.B. We are allowing $\ell < n$. In particular, the fundamental domain

$$\mathcal{F}(\mathcal{B}) := \{t_1\boldsymbol{\nu}_1 + \cdots + t_\ell\boldsymbol{\nu}_\ell : 0 \leqslant t_i < 1\}$$

is an ℓ-dimensional paralllelopiped in $\mathbb{R}^n$, and $\operatorname{Det}(L_{\mathcal{B}})$ is the ℓ-dimensional volume of $\mathcal{F}(\mathcal{B})$.

(1) Let $\boldsymbol{v}_1^*, \dots, \boldsymbol{v}_\ell^*$ be the orthgonal vectors that are the output of Gram-Schmidt. Prove that

$$\operatorname{Det}(L_{\mathcal{B}}) = \prod_{i=1}^{\ell} \|\boldsymbol{v}_i^*\|.$$

(2) Let A be the n-by-ℓ matrix whose columns are $\boldsymbol{v}_1, \dots, \boldsymbol{v}_\ell$, and let ${}^tA$ be the transpose of A. Prove that

$$\operatorname{Det}(L_{\mathcal{B}})^2 = \det({}^tAA) = \det\Big((\boldsymbol{v}_i \cdot \boldsymbol{v}_j)_{1 \leqslant i,j \leqslant \ell}\Big).$$

(3) Deduce that if $L_{\mathcal{B}} \subseteq \mathbb{Z}^n$, i.e., if $\boldsymbol{v}_1, \dots, \boldsymbol{v}_\ell \in \mathbb{Z}^n$, then $\operatorname{Det}(L_{\mathcal{B}}) \geqslant 1$.

Exercise 2.8.3. Prove Theorem 2.3.5, which says that every lattice has a basis satisfying both the Size Condition (2.3.2) and the Quasi-Orthogonality Condition (2.3.4). (This is a challenging problem.)

Exercise 2.8.4. Let $\boldsymbol{v}_1, \dots, \boldsymbol{v}_n$ be an LLL-reduced basis for L. Prove the first part of Theorem 2.3.9:

$$\|\boldsymbol{v}_1\| \leqslant 2^{(n-1)/2} \lambda_1(L).$$

Exercise 2.8.5. Consider the Gram–Schmidt With Rounding algorithm (2.1.2). We suggested that feeding it a list of vectors in small-to-large order is likely to give a better output than large-to-small.

(1) Justify this claim rigorously for lattices in $\mathbb{R}^2$.

(2) Consider the lattice L in $\mathbb{R}^3$ whose basis $\mathcal{B}$ consists fo the three vectors

$$\boldsymbol{v}_1 = (9, 3, -11),\ \boldsymbol{v}_2 = (1, -2, 3),\ \boldsymbol{v}_3 = (-190, -87, 140).$$

Feed this basis into the Gram–Schmidt With Rounding algorithm using all six permutations of the three basis vectors and compare the outputs in various ways. For example, compare the shortest vector produced, the value of the product $\prod \|\boldsymbol{v}_i^*\|$, and/or the complexity D of the output basis.

Exercise 2.8.6. Write a computer program implementing the LLL algorithm and do some of the following experiments.

(1) Choose random vectors $\boldsymbol{v}_1, \dots, \boldsymbol{v}_{30}$ in $\mathbb{Z}^{30}$, say with coefficients randomly selected in the range $[-100, 100]$, and let L be the lattice that they span. Run LLL on this basis for L and compute the orthogonality defect (Exercise 1.8.9) of the output. What happens if you use the same basis, but input the basis vectors in a differeent order?

(2) As in (1), but after running LLL, permute the output vectors and then run them through LLL again? Do you get a better basis, as measured by the orthogonality defect?

(3) Repeat (1) and (2) for different random $\boldsymbol{v}_1, \dots, \boldsymbol{v}_{30}$ and study the extent to which the results are different.

(4) Repeat (1) and (2) for a different dimension and/or vectors whose coordinates are in chosen in some other range, and compare the results.

3. Public Key Cryptography 101: A Brief Introduction

We start with the *Fundamental Problem of Cryptography*. Two people, say Alice and Bob[11], want to exchange information so that the eavesdropper Eve cannot read their messages.

3.1. Cryptography in the (pre-1970s) Dark Ages. For thousands of years, cryptography worked as follows:

Private Key Cryptography — A Single Shared Key:

Step 1: Alice and Bob meet and share a Secret Key.

Step 2: Alice and Bob go their separate ways.

Step 3: Bob writes a message, which is called the Plaintext, that he wants to send to Alice.

Step 4: Bob uses the Secret Key and a publicly available Encryption Algorithm to encrypt his message. The encrypted message is called the Ciphertext.

Step 5: Bob sends the Ciphertext to Alice.

Step 6: Alice uses the Secret Key and a publicly available Decryption Algorithm to decrypt Bob's message.

Step 7: Eve might intercept the Ciphertext, but without the Secret Key, she cannot decrypt it to recover the Plaintext.

Remark 3.1.1. A fundamental problem with this scenario is that Bob and Alice cannot send messages until after they've met and exchanged a Secret Key. What if they've never met and have no secure way to exchange the key?

Example 3.1.2. Alice is an internet company, and Bob wants to send Alice his credit card number.

3.2. Public Key Cryptography to the Rescue. Is there some way for Alice and Bob to securely communicate even if they've never met, and even if Eve is privy to every piece of information that they exchange? Seems unlikely, but in the mid-1970s Diffie and Hellman [9] proposed creating cryptosystems that use two keys, a Private Key that Alice keeps secret and a Public Key that she publishes.[12] Here's how a two-key system might work:

Public Key Cryptography — A Tale of Two Keys:

Step 1: Alice creates a two-piece key consisting of a secret Private Key and a Public Key.

Step 2: Alice publishes her Public Key.

[11] In cryptography, Alice and Bob feature in an eternal stuggle for security against the eavesdropper Eve, the opponent Oscar, the malicious attacker Mallory, and an assortment of other evildoers!

[12] The concept of public key encryption was originally discovered by James Ellis in 1969 while working at the British Government Communications Headquarters (GCHQ), but his discoveries were classified by the British government and were not declassified and released until 1997, after his death. Two other researchers at GCHQ, Malcolm Williamson and Clifford Cocks, discovered the Diffie–Hellman key exchange algorithm and the RSA public key encryption system, respectively, before the rediscovery and public dissemination by Diffie, Hellman, Rivest, Shamir, and Adleman, but again the work Williamson and Cocks was not published until much later.

Step 3: Bob choose a Plaintext that he wants to send to Alice.

Step 4: Bob uses Alice's Public Key and a publicly available Encryption Algorithm to encrypt his message.

Step 5: Bob sends the resulting Ciphertext to Alice.

Step 6: Alice uses her secret Private Key and a publicly available Decryption Algorithm to decrypt Bob's message.

Step 7: Eve can intercept the Ciphertext, and she know's Alice's Public Key, but without knowing Alice's secret Private Key, she cannot decrypt the Ciphertext to recover the Plaintext.

This all sounds great, but Diffie and Hellman were not able to propose an explicit example of such a

Public Key Cryptosystem,

although they did propose a closely related key exchange system.

Remark 3.2.1. In these notes we will continue to refer to *private key cryptosystems* in which there is only one key and *public key cryptosystems* in which there are two keys. However, we note that they are formally known as *symmetric cryptosystems* and *asymmetric cryptosystems*, since the former have a single key that is used for both encryption and decryption, while the latter have separate keys for these purposes.

3.3. A Mathematical Formulation. Before discussing some specific examples of public key cryptosystems, let's reformulate everything in mathematical terms. Keys and texts are elements chosen from certain sets, while encryption and decryption are functions between these sets:

$$\begin{bmatrix} \{\text{Public Keys}\} \times \{\text{Plain Texts}\} & \xrightarrow{\text{Encrypt}} & \{\text{Cipher Texts}\} \\ \{\text{Private Keys}\} \times \{\text{Cipher Texts}\} & \xrightarrow{\text{Decrypt}} & \{\text{Plain Texts}\} \end{bmatrix}$$

Suppose that (PubKey, PrivKey) is a valid public/private key pair. Then for all plaintexts Msg (messgaes) we want

$$\text{Decrypt}\Big(\text{PrivKey}, \text{Encrypt}\big(\text{PubKey}, \text{Msg}\big)\Big) = \text{Msg}.$$

Further, even though Eve knows the Public Key and the Ciphertext, we insist that this information does not allow her to recover the Plaintext. Only Alice, who knows the Private Key, can do that.

We may thus view Decrypt as a function that inverts Encrypt, but in order to evaluate Decrypt, an extra piece of information, the Private Key, is needed. Here is an abstract formulation of this idea.

Definition 3.3.1. A *Trap Door Function* is an invertible function $f : X \to Y$ having the following properties:[13]

[13] In these notes we will be somewhat informal about the words "easy" and "hard". Roughly speaking, a function whose input is n bits is *easy to compute* if it can be computed in time that is polynomial

(1) $f(x)$ is easy to compute.
(2) $f^{-1}(y)$ is hard to compute.
(3) Knowledge of an extra piece of information, the "trapdoor", makes $f^{-1}(y)$ easy to compute.

3.4. A Menagerie of Functions that are Ostensibly Hard to Invert. We describe four mathematical problems that can be used to build trapdoor functions, assuming that the underlying mathematical problems are actually hard.

3.4.1. The Integer Factorization Problem (IFP). Given two large prime numbers p and q and an exponent e, the exponentiation function

$$\mathbb{Z}/pq\mathbb{Z} \longrightarrow \mathbb{Z}/pq\mathbb{Z}, \quad x \longmapsto x^e \bmod pq, \tag{3.4.1}$$

is easy to compute, but hard(?) to invert unless you know p and q.

Remark 3.4.2. We're cheating a bit, since inverting (3.4.1) isn't really the Integer Factorization Problem. We might call it the

Taking Roots Modulo pq Problem.

However, for appropriate choices of p, q, and e, the fastest known algorithm for finding roots modulo pq is to first solve for p and q.

The IFP is used to build the[14]

$$\left[\begin{array}{c}\textbf{RSA Public Key Cryptosystem}.\\ \text{Public Key} = (pq, e), \qquad \text{Private Key} = (p, q).\end{array}\right]$$

3.4.2. The Discrete Logarithm Problem (DLP). Let p be a large prime number, and let $g \in \mathbb{F}_p^*$. The powering function

$$\mathbb{Z}/(p-1)\mathbb{Z} \longrightarrow \mathbb{F}_p^*, \quad k \longmapsto g^k \bmod p,$$

is easy to compute, but hard to invert.

The DLP is used to build the

$$\left[\begin{array}{c}\textbf{Elgamal Public Key Cryptosystem}.\\ \text{Public Key} = (p, g, g^k) \qquad \text{Private Key} = k.\end{array}\right]$$

Remark 3.4.3. Again we've cheated. Elgamal is actually based on the

Diffie–Hellman Problem: Given g, g^a, g^b, compute g^{ab}.

But again, the fastest general method known for solving the Diffie–Hellman Problem is to compute a or b, i.e., to solve the DLP.

---

in n, and preferably not much more than linear in n. It is *hard to compute* if it takes time that is exponential, or at least super-polynomial, in n to compute. Depending on the desired security level, a quantity is currently considered hard to compute if its computation takes at least 2^{80}, or 2^{160}, or 2^{320} operations.

[14]For added effciency, it suffices for the RSA private key to be an integer d that satisfies the congruence $de \equiv 1 \pmod{pq - p - q + 1}$.

3.4.3. The Elliptic Curve Discrete Logarithm Problem (ECDLP). Similar to the DLP, but the multiplicative group $\mathbb{F}_p^*$ is replaced by the group of points $E(\mathbb{F}_p)$ on an elliptic curve. Thus let p be a large prime number, let $E/\mathbb{F}_p$ be an elliptic curve, and let $Q \in E(\mathbb{F}_p)$ be a point on E having large order N. Then the multiplication function

$$\mathbb{Z}/N\mathbb{Z} \longrightarrow E(\mathbb{F}_p), \qquad k \longrightarrow kQ,$$

is easy to compute, but hard to invert.

The ECDLP is used to build the

$$\left[\begin{array}{c}\textbf{Elliptic Curve Elgamal Public Key Cryptosystem.}\\ \text{Public Key} = (p, E, Q, kQ) \qquad \text{Private Key} = k.\end{array}\right]$$

Remark 3.4.4. Why bother using elliptic curves, since additon on $E(\mathbb{F}_p)$ takes a lot more effort than multiplication in $\mathbb{F}_p^*$? The answer is that the best known algorithms to solve the ECDLP are much slower than those for the IFP or the DLP. This means that keys and ciphertexts using ECDLP-based cryptosystems are smaller. Thus elliptic curve cryptography is part of the never-ending battle between contradictory cryptographic goals:

- ⋆ Be maximally efficient!
- ⋆ Be maximally secure!

Example 3.4.5. The bar codes on airline boarding passes have a limited number of bits, so they may use an ECDLP-based digital signature. Similarly, blockchain applications may need to store and transmit millions of digital signatures, so they often use ECDLP-based signatures in order to save space and bandwidth.

3.4.4. The Closest Vector Problem (CVP). Let L be a lattice and let $\mathcal{B}^{\text{bad}} = \{w_1, \dots, w_n\}$ be a bad basis for L. Then the function

$$\{0,1\}^n \longrightarrow \mathbb{R}^n,$$
$$(\epsilon_1, \dots, \epsilon_n) \longmapsto \epsilon_1 w_1 + \cdots + \epsilon_n w_n + (\text{small random vector})$$

is easy to compute, but hard to invert.

The CVP is used to build various cryptosystems, including the

$$\left[\begin{array}{c}\textbf{GH Public Key Cryptosystem.}\\ \text{Public Key} = \text{a bad basis } \mathcal{B}^{\text{bad}} \qquad \text{Private Key} = \text{a good basis } \mathcal{B}^{\text{good}}.\end{array}\right]$$

Cryptosystems built from the CVP are discussed in more detail in Lecture 4.

3.5. From Trapdoor Functions to Public Key Cryptosystems.

3.5.1. From IFP to RSA. The **RSA Cryptosystem** invented by Rivest, Shamir, and Adelman, works as follows:

- Private Key: (p,q). Public Key: (pq,e).
- Plaintext: A number M mod pq.
- Ciphertext: The number $C \equiv M^e \pmod{pq}$.

- Decryption: Compute
$$C^d \bmod pq \text{ where } de \equiv 1 \pmod{pq - p - q + 1}.$$

3.5.2. From DLP to Elgamal. The **Elgamal DLP-Based Cryptosystem** introduces randomness, a topic to which we shall return.

- Private Key: k Public Key: (p, g, g^k), where $g \in \mathbb{F}_p^*$.
- Plaintext: A number $M \bmod p$.
- Ciphertext: Choose a a random $R \bmod p - 1$. The ciphtertext is the pair of values
$$C_1 \equiv g^R \pmod{p} \quad \text{and} \quad C_2 \equiv M \cdot (g^k)^R \pmod{p}.$$
- Decryption: Compute $(C_1^k)^{-1} \cdot C_2 \bmod p$.

3.5.3. From ECDLP to Elliptic Elgamal. The **Elliptic Curve Elgamal Cryptosystem** works similarly, with the multiplicative group $\mathbb{F}_p^*$ replaced by the group of points on an elliptic curve $E(\mathbb{F}_p)$. There are also proposals for quantum secure public key cryptosystems based on maps (isogenies) between elliptic curves; see the material by Kristen Lauter in this volume.

3.5.4. From CVP to GGH and NTRU. It's relatively straightforward to create a public key cryptosystem from the CVP, where the public key is a bad basis and the private key is a good basis. However, lattice reduction algorithms such as LLL-BKZ make such systems insecure unless the dimension, and thus the key size, is quite large. The use of lattices having additional structure leads to practical lattice-based cryptosystems. Lectures 4 and 5 are devoted to an introduction to lattice-based cryptography.

3.6. Digital Signatures. Digital Signatures provide a way for Bob to electronically sign a digital document. The secure operation of the internet and its attendant businesses relies at least as much on digital signatures as it does on public key cryptosystems.

Example 3.6.1. Bob is Microsoft or Apple sending an update for Alice's computer. Should Alice install it? Only if she can verify that it came from Bob.

3.6.1. Signatures in the Dark Ages. Bob's written signature on a document affirmed that he created the document or was willing to abide by its terms. For example:

Bobolink Bank of Boston

Pay to **Alice Adams** $100.00

BOB ROBERTS

The bank can verify Bob's signature by comparing it to a copy that they have on file. But suppose that Bob wants to sign a computer file and send it to Alice over

the internet. How can he do that so that she is able verify his digital signature on the file?

3.6.2. Digital Signature Schemes: Mathematical Description. Just as with a public key cryptosystem, a

Digital Signature Scheme

consists of two functions:

$$\left[\begin{array}{rcl} \{\text{Private Keys}\}\times\{\text{Digital Docs}\} & \xrightarrow{\text{Sign}} & \{\text{Signatures}\} \\ \{\text{Public Keys}\}\times\{\text{Signatures}\}\times\{\text{Digital Docs}\} & \xrightarrow{\text{Verify}} & \{\text{Yes, No}\} \end{array}\right]$$

These functions are required to have the following properties:

- If (PubKey,PrivKey) is a valid public/private key pair, then for all documents Doc and potential signatures Sig we have
$$\text{Verify}(\text{PubKey}, \text{Sig}, \text{Doc}) = \text{Yes} \iff \text{Sig} = \text{Sign}(\text{PrivKey}, \text{Doc}).$$
- Given only PubKey and Doc, it is difficult to create a signature Sig satisfying
$$\text{Verify}(\text{PubKey}, \text{Sig}, \text{Doc}) = \text{Yes}.$$
- A list (transcript) of valid signatures
$$(\text{Sig}_i, \text{Doc}_i), \quad i = 1, 2, 3, \dots, n,$$
for the public key PubKey should not reveal useful information about the associated private key PrivKey, nor should it allow Eve to sign any documents other than those already in the list.

3.6.3. Examples of Digital Signature Schemes.

RSA Signatures

- Public Key: (pq, e).
- Private Key: d satisfying $de \equiv 1 \pmod{pq - p - q + 1}$.
- Document: A number $D \bmod pq$.
- Signing: The number $S \equiv D^d \pmod{pq}$.
- Verifying: Signature is valid if $S^e \equiv D \pmod{pq}$.

Elgamal Signatures

- Public Parameters: (p, g), where $g \in \mathbb{F}_p^*$.
- Private Key: k
- Public Key: g^k.
- Document: A number $D \bmod p$.
- Signing: Choose a random $R \bmod p-1$. The signature is the pair of values
$$S_1 \equiv g^R \pmod{p}, \qquad S_2 \equiv (D - k \cdot S_1) \cdot R^{-1} \pmod{p-1}.$$
- Verifying: Signature is valid if $(g^k)^{S_1} \cdot S_1^{S_2} \equiv g^D \pmod{p}$.

Elliptic Elgamal Signatures

- Public Parameters: (p, E, x, Q), where $Q \in E(\mathbb{F}_p)$ is a point of (large prime) order q, and where x is the x-coordinate function on a Weierstrass equation for E.
- Private Key: $k \bmod q$
- Public Key: kQ
- Document: A number $d \bmod q$.
- Signing: Choose a random $r \bmod q$. The signature is the pair of values
$$s_1 = x(rQ) \bmod q, \qquad s_2 = (d + ks_1)r^{-1} \bmod q.$$
- Verifying: Signature is valid if $x\Big(ds_2^{-1}Q + s_1 s_2^{-1}(kQ)\Big) \equiv s_1 \pmod q$.

Lattice-Based Signatures

- Lecture 5 gives an introduction to lattice-based digital signature schemes.

3.7. Cryptographically Secure Hash Functions. If Bob's document Doc is large, he could break it into pieces
$$\mathrm{Doc}_1, \mathrm{Doc}_2, \mathrm{Doc}_3, \ldots$$
and sign each piece. But that's inefficient, as well as leading to security issues. Instead, Bob signs a cryptographically secure hash of his document.

Definition 3.7.1. Intuitively, a *Hash Function* takes an arbitrary length input and creates a fixed length, deterministic, but unpredictable and random looking, output:
$$\left\{\begin{matrix}\text{arbitrary length}\\ \text{bit strings}\end{matrix}\right\} \xrightarrow{\mathrm{Hash}} \left\{\begin{matrix}\text{bit strings}\\ \text{of length } b\end{matrix}\right\}.$$
The function Hash should have (at least) the following properties:

- For $D \in \{0,1\}^*$, computing $\mathrm{Hash}(D)$ is **very** fast.
- Given $H \in \{0,1\}^b$, it is very hard to find even one
$$D \in \{0,1\}^* \quad \text{satisfying} \quad \mathrm{Hash}(D) = H.$$
- It is very hard to find distinct
$$D_1, D_2 \in \{0,1\}^* \quad \text{satisfying} \quad \mathrm{Hash}(D_1) = \mathrm{Hash}(D_2).$$
This property is called *Collision Resistance*.
- Altering even one bit of D changes $\mathrm{Hash}(D)$ unpredictably. We might formalize this as:
$$\mathrm{Prob}\left(\begin{matrix}\text{the jth bits of } \mathrm{Hash}(D)\\ \text{and } \mathrm{Hash}(D') \text{ are equal}\end{matrix}\;\middle|\;\begin{matrix}D \text{ and } D' \text{ differ}\\ \text{in exactly one bit}\end{matrix}\right) = 50\%.$$

3.8. Random Numbers in Cryptography. For RSA, Bob and Alice need to choose *random* prime numbers. Elgamal uses *random* numbers to encrypt and sign. And even a deterministic cryptosystem such as RSA tends to have security problems unless some randomness is introduced.

Example 3.8.1 (Plaintext Padding)**.** The following is only semi-realistic. When Bob wants to send the message M to Alice, rather than encrypting M directly, he chooses a random string R and instead sends Alice the concatenated string

$$M' = R \,\|\, (R \texttt{ xor } M).$$

Then, even if Eve guesses part of the message, she cannot use that knowledge to help with decryption, since the bits of M have been scrambled by R. Alice recovers the actual plaintext by first computing M', and then computing

$$(R \texttt{ xor } M) \texttt{ xor } R = M.$$

There are sources of bits that people believe are truely random:

- Quantum phenomena, such as radioactive decay.
- Micro-changes in temperature.

Such sources can be used in practice, but they tend to be inefficient.

What Bob and Alice need is a *Psuedo-Random Number Generator* (PRNG). For example, they might use a function

$$\mathrm{Rand} : \{0,1\}^N \longrightarrow \{0,1\}^N$$

and iterate it starting from a seed value σ_0. This gives a sequence of values

$$\sigma_1 = \mathrm{Rand}(\sigma_0), \quad \sigma_2 = \mathrm{Rand}(\sigma_1), \quad \sigma_3 = \mathrm{Rand}(\sigma_2), \ldots$$

and they would like this sequence to be "indistinguishable" from a sequence of values chosen randomly and uniformly from $\{0,1\}^N$.

Another way to create a pseudo-random number generator is to use a hash function and a seed value σ_0, and compute the list of pseudo-random numbers

$$\sigma_1 = \mathrm{Hash}(\sigma_0 \,\|\, 1), \quad \sigma_2 = \mathrm{Hash}(\sigma_0 \,\|\, 2), \quad \sigma_3 = \mathrm{Hash}(\sigma_0 \,\|\, 3), \ldots .$$

Of course, there is still the problem of generating a random seed σ_0 value. And it must be noted that there have been real-world security breaches arising from insufficiently random seed values!

Remark 3.8.2. Don't be fooled by the relatively innocuous definitons in Sections 3.7 and 3.8. Creating cryptographically secure hash functions and pseudo-random number generators is hard! The US National Institute of Standards and Technology (NIST) ran an open competition from 2007 to 2012 to develop an efficient and secure cryptographic hash function. The winner was publised in 2015 and is called *Secure Hash Function-3* (SHA-3), although an earlier SHA-2 is also still widely used.

3.9. How Hard are Hard Problems? So just how hard are famous "hard problems" such as the IFP, DLP, ECDLP, and CVP?

Honest Answer: No one knows!!! In the sense that we don't have a proof that any of these problems are hard.

Practical Answer: How hard are they to solve using existing algorithms on existing computers?

Problem	Steps Required to Solve the Problem	Key/Ciphertext Size to be Secure
IFP	$\approx \exp(\sqrt[3]{\log pq})$ steps	2000 to 4000 bits
DLP	$\approx \exp(\sqrt[3]{\log p})$ steps	2000 to 4000 bits
ECDLP	$\approx \sqrt{p}$ steps	300 to 400 bits
CVP	$\approx C^{\dim L}$ steps	2000 to 4000 bits

That's all great. Even 4000 bits isn't much. But you probably noticed the caveat:

Existing algorithms on **existing** computers.

3.10. Quantum Computers and Cryptography.

Definition 3.10.1. A *quantum computer* is a machine in which computation on bits (0's and 1's) is replaced by computation on *qubits*.

In the popular literature, a qubit is a bit that can take on every real value between 0 and 1. Slightly more precisely, a qubit is described by a complex number representing a superposition of 0 and 1 states with certain probabilities. A quantum computer with n qubits can "perform" a simultaneous computation on 2^n states, achieving an exponential speedup over a classical computer for certain tasks.

At present, the largest quantum computers built have around 100 qubits. But governments and businesses are investing huge sums of money to build larger ones. One might make an analogy:

- First airplane flight — 1903 — flew 852 feet.
- WW I — 1914–18 — airplanes ubiquitous.
- WW-II — 1939–45 — jets flying 500+ MPH.

Here's the bombshell paper that started all the fuss:

Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, Peter W. Shor. Proc. 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, Nov. 20–22, 1994

Shor's quantum algorithms solve the IFP and DLP (and ECDLP) in more-or-less *quadratic* time, so a full-scale quantum computer would compromise the security of most classical public key systems. This has sparked much current research on public key systems that cannot (as far as we know) be broken by a quantum computer. These include lattice-based systems discussed in Sections 4 and 5, isogeny-based systems discussed in Kristen Lauter's chapter in this volume, and

systems based on coding theory. People refer to work in this field as:

Post-Quantum Cryptography

3.10.1. What, Me Worry? If large-scale working quantum computers are decades off, why worry about them now?

- Infrastructure change is slow, time-consuming, and expensive.
- Even if post-quantum cryptosystems are not used now, we should build them into systems so that we can start using them with the flip of an (electronic) switch.
- How long do you want to protect your secrets? Your legal documents? Your blockchains? If for 10 years, then you should *probably* encrypt and sign using PQC. If 50 years, then you *definitely* should.

NIST, esimtates that by 2030 it will cost roughly $1 billion to build a quantum computer that can break 2048-bit RSA, is in the final stages of running a multi-year open competition to select and standardize post-quantum cryptosystems. NIST announced the Round 3 candidates in July 2020. It consists of 4 public key cryptosystems and 3 digital signatures, together with 8 additional alternate candidates.

3.11. Code Makers Versus Code Breakers, Or, Cryptanalysts Are Very Clever! History is littered with the invention of "unbreakable" cryptosystems that were broken. So before you start touting your own brilliant new cryptosystem, here are a few lessons that the author has painfully learned over the years:

- Cryptanalysts, those pesky folks who break cryptosystems, are *very very* clever people.
- Cryptanalysts don't play by your rules, they set their own rules. They'll try to break your algorithm, they'll try to break your software implementation, they'll try to break the hardware that you're using!
- Cryptanalysts attack the weakest part of your system or its implementation, frequently via a method that you hadn't considered.
- If you modify a cryptosystem to make it more efficient, 99 times out of 100 you'll end up compromising its security.
- It's not enough for your system to be based on a problem whose hardest case is hard. Instead you need "most" of the instances of your problem to be hard.
- Even if most instances of your problem are hard, you still need to be able to distinguish, and avoid, the instances that are not hard. And *a priori*, it's often not clear which instances are easy (or easier).
- It is important to be able to prove, under reasonable assumptions, that an oracle that decrypts or signs is also capable of solving a hard mathematical problem. These are called *security reduction proofs*. However, they come with two important caveats:

- What constitutes a "reasonable assumption"? This is a subtle question, and systems that were supposedly secure because of a security reduction proof have turned out to be insecure due to issues with the underlying assumptions.
- Since we don't yet know how to prove that cryptographically useful math problems are hard, what a security reduction proof demonstrates is that breaking the cryptosystem is equivalent to solving a mathematical problem that has been studied intensively for a many years. Thus the cryptographic security underlying today's widely deployed cryptographic constructions is ultimately based on the following principle:

Proof via lack of known algorithms despite much effort

Here are a few examples of non-obvious weaknesses in otherwise ostensibly hard mathematical problems and secure implementations, where we note that an integer is said to be *smooth* if it is a product of small primes.

- For most primes p and q, one expects the IFP for pq to be hard. But it is easier if $p-1$ or $q-1$ is sufficiently smooth.
- For most primes p and most generators $g \in \mathbb{F}_p^*$, one expects the DLP to be hard. But it is easier if $p-1$ is sufficiently smooth.
- For most primes p, most elliptic curves $E/\mathbb{F}_p$, and most points $Q \in E(\mathbb{F}_p)$, one expects the ECDLP to be hard. But letting N denote the order of Q in $\#E(\mathbb{F}_p)$, the ECDLP is:
  - Easier if N is sufficiently smooth.
  - Easier if there is a small f such that $p^f \equiv 1 \pmod{N}$.
  - Extremely easy if $\#E(\mathbb{F}_p) = p$.
- Let (pq, e) be an RSA key. If Eve can steal (or guess) half the bits of p, she can recover the other bits using lattice reduction!
- A long transcript of classical lattice-based signatures can be used to reconstruct the fundamental domain of a good basis. And it turns out that in some cases, the transcript doesn't even have to be all that long!
- Software and hardware implementations of cryptosystems, especially if optimized to avoid unnecessary calculations, can often be compromised by measuring their power consumption or execution time. These are called *power analysis and timing analysis attacks*. To see why they might succeed, note that flipping a bit takes longer and uses more energy than leaving it unchanged.

3.12. Exercises for Lecture 3.

Exercise 3.12.1. (1) Verify that RSA decryption (Section 3.5.1) works.
(2) Verify that Elgamal decryption (Section 3.5.2) works.
(3) Verify that RSA signature verification (Section 3.6.3) works.
(4) Verify that Elgamal signature verification (Section 3.6.3) works.

Exercise 3.12.2. (1) Prove that an oracle that solves the Discrete Logarithm Problem (Section 3.4.2) can be used to solve the Diffie–Hellman problem (Remark 3.4.3)

(2) Prove that an oracle that solves the Diffie–Hellman problem (Remark 3.4.3) can be used to break the Elgamal cryptosystem (Section 3.5.2).

(3) Prove that an oracle that decrypts Elgamal ciphertexts (Section 3.5.2) can be used to solve the Diffie–Hellman problem (Remark 3.4.3).

(4) It is an open question as to whether an oracle that solves the Diffie–Hellman problem can be used to solve the Discrete Logarithm Problem. Try to say something about this question.

Exercise 3.12.3. Bob wants to send the message M to Alice. He first computes the following quantities:

$$X \leftarrow M \mathbin{\texttt{xor}} \operatorname{Hash}_1(R) \quad \text{and} \quad Y \leftarrow R \mathbin{\texttt{xor}} \operatorname{Hash}_2(X).$$

He encrypts and sends Alice the concatenation $M' = X \,\|\, Y$.

(1) Show how Alice can recover M from M'; i.e., show that Aliec can recover M from X and Y.

(2) Assuming that the hash functions are appropriately secure, explain why Eve needs to know every bit of X in order to recover R from Y, and she needs to know every bit of R to recover M from X.

This is called *optimal asymmetric encryption padding*.

Exercise 3.12.4. It is asserted in Sections 3.4.1 and 3.4.2 that raising a given quantity to a given power is easy. However, when implementing systems such as RSA or Elgamal, one needs to compute a^n in a ring $\mathbb{Z}/m\mathbb{Z}$, where m, a, n may have hundreds of digits.

(1) If one naively sets $a_0 = 1$ and $a_{i+1} = a \cdot a_i$, how many multiplications are required to compute a^n?

(2) Consider the following *square-and-multiply* algorithm:

(a) Write n in binary as $n = n_0 + 2 \cdot n_1 + 4 \cdot n_2 + \cdots + 2^t \cdot n_t$, where $n_i \in \{0, 1\}$.

(b) Set $b_0 = a$. For $i = 1, 2, \ldots, t$, compute $b_i = b_{i-1}^2$.

(c) Set $a_0 = 1$. For $i = 1, 2, \ldots, t$, compute $a_i = \begin{cases} a_{i-1} & \text{if } n_{i-1} = 0, \\ a_{i-1} \cdot b_{i-1} & \text{if } n_{i-1} = 1. \end{cases}$

Prove that $a_t = a^n$, and that the computation requires at most $2\log_2(n)$ multiplications and (roughly) $\log_2(n)$ storage.

(3) Read about improvements to the square-and-multiply algorithm including: (a) Implementations requiring only $O(1)$ storage. (b) Improved performance using precomputation and/or windowing methods. (c) Using a binary expansion of n with coefficients in $\{-1, 0, 1\}$. This is especially helpful for elliptic curve cryptography, where the inversion operation $P \to -P$ is very fast.

Exercise 3.12.5 (Shanks's babystep-giantstep algorithm)**.** This exercise describes a method for solving the DLP. Let $g \in \mathbb{F}_p^*$ have order N, and let $h \in \mathbb{F}_p^*$. The goal is to find an exponent k so that $g^k = h$. Consider the following algorithm:[15]

- Set $n \leftarrow 1 + \lfloor \sqrt{N} \rfloor$.
- Compute $u \leftarrow g^n$. (If n is large, see Exercise 3.12.4.)
- Create two lists:

$$\begin{aligned} &\text{List 1:} \quad 1, g, g^2, g^3, \ldots, g^n \\ &\text{List 2:} \quad h, hu, hu^2, hu^3, \ldots, hu^n \end{aligned}$$

- Find an element that's in both lists, say $g^i = hu^j$.
- Output $i - j \bmod N$.

(1) Prove that the two lists are guaranteed to have an element in common.
(2) Prove that the output is an integer k satisfying $g^k = h$.
(3) Prove that the algorithm requires $O(\sqrt{N}))$ multiplications and $O(\sqrt{N})$ storage.[16]
(4) Read about Pollard's rho method, which also solves the DLP in $O(\sqrt{N})$ steps, but requires only $O(1)$ storage.

Exercise 3.12.6 (The Pohlig-Hellman algorithm)**.** This exercise illustrates a method for solving the DLP when the order of the element is highly composite. Let $g \in \mathbb{F}_p^*$ have order N, and let $h \in \mathbb{F}_p^*$. The goal is to an exponent k so that $g^k = h$. For ease of exposition, we suppose that

$$N = q_1 q_2 \quad \text{with } q_1 \text{ and } q_2 \text{ distinct primes of roughly the same size.}$$

- Let

$$g_1 = g^{N/q_1}, \qquad g_2 = g^{N/q_2}, \qquad h_1 = h^{N/q_1}, \qquad h_2 = h^{N/q_2}.$$

- Solve the DLP for the pairs (g_1, h_1) and (g_2, h_2), i.e., find exponents k_1 and k_2 so that

$$g_1^{k_1} = h_1 \quad \text{and} \quad g_2^{k_2} = h_2.$$

- Solve the simultaneous congruences

$$k \equiv k_1 \pmod{q_1} \quad \text{and} \quad k \equiv k_2 \pmod{q_2}.$$

(1) Prove that the output k from this algorithm satisfies $g^k = h$.
(2) Exercise 3.12.5 describes an algorithm which says that if g has order N, then we can solve the DLP for the pair (g, h) in time that is proportional to $N^{1/2}$. Prove that if we use Exercise 3.12.5 to solve each of the subsidiary DLPs in the above algorithm, then the time to solve the original DLP is proportional to $N^{1/4}$, which represents a substantial savings.

[15] The powers of g are the "babysteps," the powers of u are the "giantsteps."
[16] We note that there are sorting algorithms that allow one to find a common element in the two lists in roughly $O(\log N)$ steps, which is negligible.

(3) Generalize to the case that N is a product $q_1^{e_1} q_2^{e_2} \cdots q_t^{e_t}$ of many distinct primes raised to powers. If it roughly takes time $T(n)$ to solve the DLP for an element of order n, prove that your algorithm solves the DLP in time that is roughly the sum $\sum_{i=1}^{t} T(q_i^{e_i})$.
(4) Let q be a prime, and as in (3), suppose that it takes time $T(n)$ to solve the DLP for pairs (g, h) if g has order n. Let $N = q^e$ be a power of a prime q,and suppose that g has order N. Devise an algorithm to solve the DLP for (g, h) that takes time roughly $eT(q)$.

Exercise 3.12.7. A *smooth number* is an integer whose factorization uses only small primes, e.g., an integer is B-smooth if its factorization is a product of prime powers using only primes less than B. Many factorization and DLP algorithms exploit smooth numbers.

(1) Read about the quadratic sieve factorization method, and how it depends on finding smooth numbers of the form $x^2 - y^2 \bmod N$.
(2) Read about the number field sieve, a generalization of the quadratic sieve that is currently the fastest known general factorization method.
(3) Read about Pollard's $p-1$ factorization algorithm, which efficiently factors N if N has a prime factor p such that $p-1$ is smooth.
(4) Read about Lenstra's elliptic curve factorization algorithm, which is similar to Pollard's $p-1$ algorithm, but uses elliptic curve groups $E(\mathbb{F}_p)$ in place of the multiplicative group $\mathbb{F}_p^*$.
(5) Read about the index calculus, which is an algorithm to solve the DLP in $\mathbb{F}_p^*$ that relies on finding numbers of the form g^i and $g^j h$ whose lifts to integers in the interval $[1, p-1]$ are smooth.

Exercise 3.12.8. Read about hash functions and pseudo-random number generators. Try to implement one of them.

4. Lattice-Based Public Key Cryptosystems

4.1. Early Days and the Ajtai-Dwork Lattice-Based Cryptosystem. In 1995 Ajtai and Dwork [1] described a lattice-based public key cryptosystem whose security relies on the difficulty of solving CVP in a certain set of lattices $\mathcal{L}_{AD}$. They proved that breaking their system for a random lattice of dimension m in $\mathcal{L}_{AD}$ is as difficult as solving SVP for *all* lattices in $\mathcal{L}_{AD}$ of dimension n, where $n < m$ depends on m. This *average case-worst case equivalence* was a theoretical cryptographic milestone, but unfortunately the Ajtai-Dwork cryptosystem is impractical. Inspired by the work of Ajtai and Dwork, Goldreich–Goldwasser–Halevi [14] proposed a somewhat more practical lattice-based cryptosystem, while at the same time and working indpendently, Hoffstein–Pipher–Silverman [19] proposed an even more efficient system that they dubbed NTRU. In this chapter we describe the GGH and NTRU public key cryptosystems.

It is interesting to note that initially the major advantage of these systems was their speed, since lattice-based systems can be 10 to 100 times faster than RSA and ECC at equivalent security levels. However, as computers became faster, the speed differential became less important. Then in an unexpected turn of events, interest in lattice-based systems flourished due both to their ostensible resistance to quantum attacks and their adaptability to applications such as Gentry's homomorphic encryption scheme [12].[17]

4.2. The GGH Public Key Cryptosystem. The *GGH lattice-based cryptosystem* of Goldreich, Goldwasser, and Halevi [14] is described in Table 4.2.1.

- Note that the ciphertext vector $\boldsymbol{e}$ is not in the lattice L, but it is close to a lattice vector, since $\mathbf{r}$ is small.
- Bob uses the bad public basis $\mathcal{B}^{\text{bad}}$ and a random small vector $\mathbf{r}$ to create the ciphertext.
- Alice uses Babai's method (Section 1.3) with her good basis $\mathcal{B}^{\text{good}}$ to solve CVP. In this way she finds a vector $\boldsymbol{v} \in L$ that is close to the ciphertext. Since $\mathbf{r}$ is small, when Alice writes $\boldsymbol{v}$ in terms of $\mathcal{B}^{\text{bad}}$, with high probability it will be equal to

$$\boldsymbol{v} = \epsilon_1 \boldsymbol{w}_1 + \cdots + \epsilon_n \boldsymbol{w}_n.$$

  Form this Alice can read off $(\epsilon_1, \ldots, \epsilon_n)$, which is the plaintext.

Private Key	$=$ a good basis $\mathcal{B}^{\text{good}}$ for a lattice L
	$= \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\}$.
Public Key	$=$ a bad basis $\mathcal{B}^{\text{bad}}$ for the lattice L
	$= \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n\}$.
Plaintext	$=$ a binary vector $(\epsilon_1, \ldots, \epsilon_n)$, i.e., $\epsilon_i \in \{0, 1\}$.
Ciphertext	$= \epsilon_1 \boldsymbol{w}_1 + \cdots + \epsilon_n \boldsymbol{w}_n + \mathbf{r}$, where $\mathbf{r}$ is a small random vector.

Table 4.2.1. The GGH Lattice-Based Cryptosystem

4.3. GGH versus LLL: A Battle for Supremacy! The security of GGH comes down to the question of just how well LLL and its variants solve CVP; or alternatively, how long it takes LLL-BKZ to find a basis that is good enough to solve CVP using Babai's method.

Experimentally, one finds that if $n = \dim(\mathcal{L}) < 100$, then LLL easily finds a basis that's good enough to break GGH. And even up to $n \approx 200$, variants

[17] Homomorphic encryption is a fascinating area that we will not have time to discuss. Using a homomorphic encryption scheme, Alice can have Bob run an encrypted algorithm on encrypted input data to produce encrypted output. Current schemes are impractically slow for most applications, but research continues to improve the efficiency.

of LLL-BKZ will break GGH. On the other hand, GGH may well be secure for $500 < n < 1000$; and barring some major breakthrough in lattice-reduction methods, it is likely to be secure for quite a while for (say) $2000 < n < 4000$. However, we must then deal with the following unfortunate fact. If we implement GGH using a lattice $L \subset \mathbb{R}^n$, then the GGH public key consists of n vectors in $\mathbb{R}^n$, and hence

$$\text{Size of GGH Public Key} = O(n^2) \text{ bits.}$$

For example, if we take $n = 1000$, and if we assume that each coordinate of each vector in the GGH basis is an 8 bit integer, then a GGH public key has length roughly 8-megabits. This is quite a bit larger than standard 4000 bit RSA keys or 256 bit ECC keys!

The NTRU Public Key Cryptosystem, which we describe next, solves this problem by using a special type of lattice that has bases that can be described using roughly $\frac{1}{2} n \log_2(n)$ bits, leading to key sizes that are comparable to RSA keys.

4.4. Convolution Products and Polynomial Quotient Rings. We start with some mathematical tools that we will need.

Definition 4.4.1. The *convolution product* of two vectors[18]

$$\mathbf{a} = (a_0, a_1, \ldots, a_{N-1}) \quad \text{and} \quad \mathbf{b} = (b_0, b_1, \ldots, b_{N-1})$$

is the vector

$$\mathbf{c} = \mathbf{a} \star \mathbf{b} \quad \text{with} \quad c_k = \sum_{i+j \equiv k \pmod N} a_i b_j. \tag{4.4.2}$$

Example 4.4.3. Let $\mathbf{a} = (1, 2, 3)$ and $\mathbf{b} = (4, -5, 6)$. Then

$$\mathbf{a} \star \mathbf{b} = (1 \cdot 4 + 2 \cdot 6 - 3 \cdot 5,\ -1 \cdot 5 + 2 \cdot 4 + 3 \cdot 6,\ 1 \cdot 6 - 2 \cdot 5 + 3 \cdot 4) = (1, 21, 8).$$

Proposition 4.4.4. *Vector addition and convolution product make the set of vectors into a ring. In particular,*

$$\begin{aligned} (\mathbf{a} \star \mathbf{b}) \star \mathbf{c} &= \mathbf{a} \star (\mathbf{b} \star \mathbf{c}) && \text{Associative Law,} \\ \mathbf{a} \star (\mathbf{b} + \mathbf{c}) &= \mathbf{a} \star \mathbf{b} + \mathbf{a} \star \mathbf{c} && \text{Distributive Law,} \\ \mathbf{a} \star \mathbf{b} &= \mathbf{b} \star \mathbf{a} && \text{Commutative Law.} \end{aligned}$$

Proof. We leave the proof to the reader. □

Remark 4.4.5. There is an alternative way to describe the convolution product ring that is often useful. The idea is to identify vectors and polynomials via the association

$$\mathbf{a} = (a_0, \ldots, a_{N-1}) \quad \longleftrightarrow \quad \mathbf{a}(X) = a_0 + a_1 X + \cdots + a_{N-1} X^{N-1}.$$

We view these polynomials as living in the quotient ring

$$\mathcal{R} = \mathbb{Z}[X]/(X^N - 1),$$

[18] We use the word *vector* loosely to describe an n-tuple of elements chosen from some commutative ring such as $\mathbb{Z}$ or $\mathbb{Q}$ or $\mathbb{R}$ or $\mathbb{F}_q$.

so when we multiply polynomials in the quotient ring, we use the rule $X^N = 1$ to write the result as a polynomial of degree at most $N-1$. With this identification, we have the equivalence

$$\mathbf{c} = \mathbf{a} \star \mathbf{b} \quad \Longleftrightarrow \quad \mathbf{c}(X) \equiv \mathbf{a}(X)\mathbf{b}(X) \pmod{X^N - 1},$$

a calculation that we leave to the reader. In other words, this identification defines a ring isomorphism

$$\begin{array}{ccc} (\mathbb{Z}^N, +, \star) & \longrightarrow & (\mathcal{R}, +, \cdot), \\ \mathbf{a} & \longmapsto & \mathbf{a}(X). \end{array}$$

All of the preceding material works if we replace $\mathbb{Z}$ by another ring such as $\mathbb{Z}/q\mathbb{Z}$. We will make frequent use of the ring

$$\mathcal{R}_q = (\mathbb{Z}/q\mathbb{Z})[X]/(X^N - 1),$$

where we note that the natural map $\mathcal{R} \to \mathcal{R}_q$ obtained by reducing coefficients modulo q is a ring homomorphism.

Proposition 4.4.6. *Let q be a prime. Then there are "many" polynomials $\mathbf{a}(X) \in \mathcal{R}_q$ that have an inverse in $\mathcal{R}_q$, i.e., such that there exists a polynomial*

$$\mathbf{a}(X)^{-1} \in \mathcal{R} \quad \textit{satisfying} \quad \mathbf{a}(X)\mathbf{a}(X)^{-1} \equiv 1 \pmod{q}.$$

More precisely, we have

$$\mathbf{a}(X) \in \mathcal{R}_q^* \quad \Longleftrightarrow \quad \gcd_{(\mathbb{Z}/q\mathbb{Z})[X]}\bigl(\mathbf{a}(X), X^N - 1\bigr) = 1.$$

Proof. Suppose first that $\mathbf{a}(X) \in \mathcal{R}_q^*$, so there is some $\mathbf{b}(X) \in \mathcal{R}_q$ satisfying

$$\mathbf{a}(X)\mathbf{b}(X) = 1 \quad \text{in } \mathcal{R}_q.$$

Viewing $\mathbf{a}(X)$ and $\mathbf{b}(X)$ as polynomials in $\mathbb{Z}[X]$, this is the same as saying that

$$\mathbf{a}(X)\mathbf{b}(X) \equiv 1 \pmod{X^N - 1, q},$$

so we can find polynomials $\mathbf{c}(X), \mathbf{d}(X) \in \mathbb{Z}[X]$ so that

$$\mathbf{a}(X)\mathbf{b}(X) - 1 = (X^N - 1)\mathbf{c}(X) + q\mathbf{d}(X).$$

Reducing modulo q, we see that $\mathbf{a}(X)$ and $X^N - 1$ can have no non-trivial common factors in $(\mathbb{Z}/q\mathbb{Z})[X]$, so their gcd is 1.

Conversely, suppose that the gcd of $\mathbf{a}(X)$ and $X^N - 1$ in $(\mathbb{Z}/q\mathbb{Z})[X]$ is 1. The ring $(\mathbb{Z}/q\mathbb{Z})[X]$ is Euclidean. (This is where we use the assumption that q is prime, since it implies that $\mathbb{Z}/q\mathbb{Z}$ is a field.) The extended Euclidean algorithm, which is both effective and extremely efficient, tells us that there are elements $\mathbf{b}(X), \mathbf{c}(X) \in (\mathbb{Z}/q\mathbb{Z})[X]$ satisfying

$$\mathbf{a}(X)\mathbf{b}(X) + (X^N - 1)\mathbf{c}(X) = 1 \quad \text{in } (\mathbb{Z}/q\mathbb{Z})[X].$$

Reducing modulo $X^N - 1$, we find that $\mathbf{a}(X)\mathbf{b}(X) = 1$ in $\mathcal{R}_q$, so $\mathbf{a}(X) \in \mathcal{R}_q^*$. □

4.5. NTRUEncrypt: The NTRU Public Key Cryptosystem.

Public Parameters: Alice and Bob agree on three public parameters

$$N,\ p,\ q,$$

with N prime and with $\gcd(p, q) = 1$.[19] It is also important that q be significantly larger than p.

Key Creation: Alice chooses random polynomials $f, g \in \mathcal{R}$ with small coefficients. She computes the inverses of f modulo q and modulo p,

$$F_q \equiv f^{-1} \pmod{q} \qquad \text{and} \qquad F_p \equiv f^{-1} \pmod{p}.$$

(If either inverses doesn't exist, she discards this f and chooses a new one.) Alice computes the product

$$h = g \cdot F_q \pmod{q}.$$

In other words, she computes $h = g \cdot f^{-1}$ in $\mathcal{R}_q$. Then Alice's public/private key pair is[20]

$$\text{Public Key} = h, \qquad \text{Private Key} = f.$$

Encryption: Bob's plaintext is an element $m \in \mathcal{R}_p$, or more precisely, a polynomial with integer coefficients in the interval $(-p/2, p/2]$. Bob also chooses a random polynomial r with small integer coefficients. Bob uses these quantities to compute his ciphertext

$$\text{Ciphertext} = e \equiv p \cdot r \cdot h + m \pmod{q}.$$

Decryption: Alice computes

$$a \equiv e \cdot f \pmod{q}.$$

This computation is done in the ring $\mathcal{R}_q$, and then Alice lifts to coefficients of

$$a = a_0 + a_1 X + \cdots + a_{N-1} X^{N-1}$$

so that they are integers satisfying

$$A \leqslant a_i < A + q$$

for an appropriately chosen public quantity A. Alice then reduces the coefficients modulo p and computes the product

$$F_p \cdot a \bmod p \quad \text{in the ring } \mathcal{R}_p.$$

This quantity will equal the plaintext m.

Proposition 4.5.1. *If the parameters are chosen appropriately, then NTRU decryption gives the plaintext; i.e., NTRU works as advertised.*

Proof. The first decryption step yields the following polynomial as output:

[19] In practice, as explained in Exericse 4.10.4, it is also often convenient to take p and q to be primes that are generators for $(\mathbb{Z}/N\mathbb{Z})^*$, or at least that have large order in that ring, since then the inverses F_q and F_p will exist, either certainly or with high probability, provided $f(1) \neq 0$.

[20] For efficiency, Alice might also store F_p, but if space is an issue, she can always recompute it from f and p.

Computation (mod q)	Reason It Works
$\mathbf{a} \equiv \boldsymbol{e} \cdot \mathbf{f}$	
$\equiv (p \cdot \mathbf{r} \cdot \mathbf{h} + \mathbf{m}) \cdot \mathbf{f}$	$\boldsymbol{e} \equiv p \cdot \mathbf{r} \cdot \mathbf{h} + \mathbf{m}$
$\equiv p \cdot \mathbf{r} \cdot \mathbf{g} + \mathbf{m} \cdot \mathbf{f}$	$\mathbf{h} \cdot \mathbf{f} \equiv \mathbf{g} \cdot F_q \cdot \mathbf{f} \equiv \mathbf{g}$

The coefficients of $\mathbf{r}, \mathbf{g}, \mathbf{m}, \mathbf{f}$ are *small*, so the coefficients of

$$p \cdot \mathbf{r} \cdot \mathbf{g} + \mathbf{m} \cdot \mathbf{f}$$

will lie in an interval of length less than q.[21] Choosing an appropriate interval, the polynomial

$$\mathbf{a} \quad \text{equals} \quad p \cdot \mathbf{r} \cdot \mathbf{g} + \mathbf{m} \cdot \mathbf{f} \quad \text{exactly } \mathcal{R}, \text{ and not merely modulo in } \mathcal{R}_q.$$

Now multiply $\mathbf{a}$ by F_p and reduce modulo p to get

$$\begin{aligned} F_p \cdot \mathbf{a} &= F_p \cdot (p \cdot \mathbf{r} \cdot \mathbf{g} + \mathbf{m} \cdot \mathbf{f}) \\ &\equiv F_p \cdot \mathbf{m} \cdot \mathbf{f} \pmod{p} \\ &\equiv \mathbf{m} \pmod{p} \qquad \text{since } F_p \cdot \mathbf{f} \equiv 1 \pmod{p}. \end{aligned}$$

This shows that decryption recovers the plaintext. □

4.6. NTRU and Lattice Problems.

Definition 4.6.1. The *Convolution Modular Lattice* $L_{\mathbf{h}}$ associated to the N-dimensional vector $\mathbf{h}$ and the modulus q is the $2N$-dimensional lattice whose basis is given by the rows of the following matrix:

$$L_{\mathbf{h}} = \text{RowSpan} \left(\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{N-1} \\ 0 & 1 & \cdots & 0 & h_{N-1} & h_0 & \cdots & h_{N-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & h_1 & h_2 & \cdots & h_0 \\ \hline 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{array} \right)$$

We now given an alternative, very convenient, way to describe $L_{\mathbf{h}}$.

Proposition 4.6.2. *If we identify $\mathbb{Z}^{2N}$ with two copies of the convolution product ring $(\mathbb{Z}^N, +, \star)$, then*

$$L_{\mathbf{h}} = \big\{ (\mathbf{a}, \mathbf{b}) \in \mathbb{Z}^{2N} : \mathbf{a} \star \mathbf{h} \equiv \mathbf{b} \pmod{q} \big\}.$$

[21] For example, if f, g, r, m have coefficients in the set $\{-1, 0, 1\}$, then the coefficients of $p \cdot \mathbf{r} \cdot \mathbf{g} + \mathbf{m} \cdot \mathbf{f}$ are guaranteed to lie between $-N(p+1)$ and $N(p+1)$, so we just need to take $q > N(p+1)$ and $A = q/2$. In practice, it is highly unlikely that the coefficients will get this large, so a smaller q value will work with high probability. Added efficiency may also be gained by taking the various small polynomials to have a specified number of non-zero coefficients, which allows even smaller values of q to be used.

Proof. Suppose that $(\mathbf{a},\mathbf{b})$ satisfies

$$\mathbf{a}\star\mathbf{h}\equiv\mathbf{b}\pmod{q}.$$

This implies that

$$\mathbf{u}=\frac{\mathbf{b}-\mathbf{a}\star\mathbf{h}}{q}\in\mathbb{Z}^N$$

has coordinates in $\mathbb{Z}$. Then we see that

$$[a_0,\dots,a_{N-1},b_0,\dots,b_{N-1}]$$

is equal to the matrix product

$$[a_0,\dots,a_{N-1},u_0,\dots,u_{N-1}]\begin{pmatrix}1&\cdots&0&h_0&\cdots&h_{N-1}\\ \vdots&\ddots&\vdots&\vdots&\ddots&\vdots\\ 0&\cdots&1&h_1&\cdots&h_0\\ 0&\cdots&0&q&\cdots&0\\ \vdots&\ddots&\vdots&\vdots&\ddots&\vdots\\ 0&\cdots&0&0&\cdots&q\end{pmatrix},$$

which shows that

$$[a_0,\dots,a_{N-1},b_0,\dots,b_{N-1}]\in L_{\mathbf{h}}.$$

We leave the opposite inclusion as an exercise. □

Remark 4.6.3. An immediate consequence of Proposition 4.6.2 is that if $[\mathbf{a},\mathbf{b}]\in L_{\mathbf{h}}$, then its *rotations*

$$[X^i\star\mathbf{a},X^i\star\mathbf{b}]\text{ are also in }L_{\mathbf{h}}\text{ for all }0\leqslant i<N. \tag{4.6.4}$$

4.7. Recovering an NTRU Private Key via an SVP Problem. An NTRU public/private key pair satisfies

$$\mathbf{f}\star\mathbf{h}\equiv\mathbf{g}\pmod{q}\quad\text{where }\mathbf{f}\text{ and }\mathbf{g}\text{ are small;} \tag{4.7.1}$$

i.e., $\mathbf{f}$ and $\mathbf{g}$ are vectors in $\mathbb{Z}^N$ having small coordinates. Alternatively, we may view $\mathbf{f},\mathbf{g}\in\mathbb{Z}[X]$ as being polynomials of degree at most $N-1$ having small coefficients. Proposition 4.6.2 and (4.7.1) tell us that the lattice $L_{\mathbf{h}}$ contains the short (and likely shortest non-zero) vector

$$[\mathbf{f},\mathbf{g}]=[f_0,f_1,\dots,f_{N-1},g_0,g_1,\dots,g_{N-1}].$$

For practical choices of parameters, the vector $[\mathbf{f},\mathbf{g}]$ and its rotations (4.6.4) are likely to be the shortest non-zero vectors in $L_{\mathbf{h}}$, so solving SVP in $L_{\mathbf{h}}$ essentially recovers the private key $\mathbf{f}$.

4.8. Recovering an NTRU Plaintext via a CVP Problem. An NTRU ciphertext $\boldsymbol{e}$ has the form

$$\boldsymbol{e}=p\mathbf{r}\star\mathbf{h}+\mathbf{m}\pmod{q}\quad\text{where }\mathbf{r}\text{ and }\mathbf{m}\text{ are small.}$$

We rewrite this relation in vector form as

$$\begin{aligned}[\mathbf{0}, \boldsymbol{e}] &= [\mathbf{0},\ p\mathbf{r} \star \mathbf{h} + \mathbf{m} \mod q] \\ &\equiv [\mathbf{r},\ \mathbf{r} \star (p\mathbf{h}) \mod q] + [-\mathbf{r}, \mathbf{m}].\end{aligned}$$

The vector $[\mathbf{r},\ \mathbf{r} \star (p\mathbf{h}) \mod q]$ is in the convolution modular lattice $L_{p\mathbf{h}}$ obtained by using $p\mathbf{h}$ in place of $\mathbf{h}$. Further, the vector $[-\mathbf{r}, \mathbf{m}]$ is quite short. Hence recovering the plaintext $\mathbf{m}$ from the ciphertext $\boldsymbol{e}$ is equivalent to finding the vector in $L_{p\mathbf{h}}$ that is closest to the vector $[\mathbf{0}, \boldsymbol{e}]$. It is then a question of estimating how hard it is to solve this CVP problem.

4.9. NTRU Operating Characteristics and Variants. Table 4.9.1 gives a rough comparison of the operating characteristics for naive implementations of RSA, ECC, and NTRUEncrypt; i.e., with no implementation tricks. But there are many ways to improve the speed, including:

- Using a small RSA encryption exponent gives encryption speed $O(n^2)$.
- One can do precomputation and use windowing techniques to make ECC encryption and decryption faster, although the former requires some storage.
- Using Karatsuba multiplication or fast Fourier transform techniques, one can reduce NTRU encryption and decryption times to $O(n \log n)$.

	RSA	ECC	NTRU
Encrypt/Decrypt	$O(n^3)$	$O(n^3)$	$O(n^2)$
Key size (bits)	n	n	$\approx \frac{1}{2} n \log_2 n$
Key Create	—	$O(n^3)$	$O(n^2)$
Typical n	1024	168	502

Table 4.9.1. Operating Characteristic of Vanilla RSA, ECC, NTRU

Remark 4.9.2. There are many variants of NTRU that have been proposed over the years. For example, one might replace $X^N - 1$ with $X^N + 1$ and take $N = 2^k$ so that $X^N + 1$ is irreducible in $\mathbb{Z}[X]$. More generally, one can replace $X^N - 1$ with an arbitrary (monic) polynomial $\boldsymbol{\phi}(X) \in \mathbb{Z}[X]$, which could be irreducible and must have small coefficients. The key is the fact that if $\mathbf{a}(X)$ and $\mathbf{b}(X)$ have small coefficients, then

$$\mathbf{a}(X) \cdot \mathbf{b}(X) \bmod \boldsymbol{\phi}(X) \quad \text{has small'ish coefficients.}$$

The average size of the coefficients of the product depends on the size of the roots of $\boldsymbol{\phi}(X)$, which is why taking $\boldsymbol{\phi}(X)$ to be a cyclotomic polynomial is especially efficient. On the other hand, using $X^N - 1$ introduces symmetries into the lattice that an attacker might exploit. This hypothetical problem can be eliminated if, for example, one takes $\boldsymbol{\phi}(X) = X^N - X - 1$.

4.10. Exercises for Lecture 4.

Exercise 4.10.1. Let $f(X) \in \mathcal{R} = \mathbb{Z}[X]/(X^N - 1)$. We define the *reversal* (or *conjugate*) of $f(X)$ to be the polynomial

$$\overline{f}(X) \leftarrow f(X^{N-1}) \in \mathcal{R}.$$

Since $X^N = 1$, we may write this informally as $\overline{f}(X) = f(X^{-1})$.

(1) Let

$$f(X) = \sum_{i=0}^{N-1} a_i X^i \quad \text{and} \quad \overline{f}(X) = \sum_{i=0}^{N-1} \overline{a}_i X^i.$$

Find a formula for $\overline{a}_i$ in terms of $a_0, \ldots, a_{N-1}$.

(2) Prove that reversal is a ring homomorphism $\mathcal{R} \to \mathcal{R}$, i.e., prove that

$$\overline{f+g} = \overline{f} + \overline{g} \quad \text{and} \quad \overline{f \cdot g} = \overline{f} \cdot \overline{g}.$$

(3) Define the *sup norm* of a polyomial

$$f(X) = \sum_{i=0}^{N-1} a_i X^i \in \mathcal{R} \quad \text{to be} \quad \|f\|_\infty := \max_{0 \leqslant i < N} |a_i|,$$

i.e., $\|f\|_\infty$ is the sup-norm of the vector of coefficients of f. Let $B > 0$ be an integer. Compute the values of

$$\sum_{f \in \mathcal{R},\, \|f\|_\infty \leqslant B} f(X) \quad \text{and} \quad \sum_{f \in \mathcal{R},\, \|f\|_\infty \leqslant B} f(X) \cdot \overline{f}(X).$$

The fact that the latter average is non-zero features in various analyses of NTRU-type cryptographic constructions.

Exercise 4.10.2. Prove the associative, distributive, and commutative laws for $+$ and $\star$ as described in Proposition 4.4.4.

Exercise 4.10.3. With the identifications described in Section 4.4, prove that

$$c = a \star b \quad \Longleftrightarrow \quad c(X) = a(X) \cdot b(X) \text{ in } \mathcal{R}.$$

Exercise 4.10.4. Let N and q be primes such that q is a primitive root modulo N; i.e., assume that $q^k \equiv 1 \pmod{N}$ if and only if $N - 1 \mid k$.

(1) Prove that the splitting field of $X^N - 1$ over $\mathbb{Z}/q\mathbb{Z}$ is a field with q^N elements.
(2) Prove that $a(X) \in \mathcal{R}_q^*$ if and only if $a(1) \not\equiv 0 \pmod{q}$.
(3) Deduce in this case that $\#\mathcal{R}_q^*/\#\mathcal{R}_q = 1/q$.
(4) Suppose instead that q has exact order k in $(\mathbb{Z}/N\mathbb{Z})^*$. Modify (a,b,c) to give correct statements, and prove that those statements are correct.

Exercise 4.10.5. Suppose that the coefficients of the polynomials f and g in an NTRU private key are randomly selected so that d of the coefficients equal 1 and d

of the coefficients equal -1 and the remaining coefficients equal 0.[22] Suppose further that the coefficients of r and m lie in $\{-1, 0, 1\}$.

(1) Prove that if
$$q > 2d(p+1)$$
then NTRU decryption always works.

(2) Prove that the number of possible f polynomials is $N!/(N-d)!(d!)^2$, and similarly for g. Thus N and d must be large enough to preclude an attacker from simply checking all possible f values.[23]

(3) For a private key chosen as above, compare the length of the target vector $[f, g]$ in the NTRU lattice L_h to the Gaussian expected shortest length of a non-zero vector in L_h. (The Gaussian heuristic is described in Section 1.6.) Express you answer in terms of N, q, d. Then take $q \approx 2d(p+1)$ as in (a), and show that the target vector is likely to be unique shortest non-zero vector in L_h.[24] More precisely, show that the ratio of private key vector length to Gaussian expected shortest length is on the order of $1/\sqrt{Np}$.

Exercise 4.10.6. For a polynomial $a(X) = \sum_{i=0}^{N-1} a_i X^i \in \mathcal{R}$, we define various norms

$$\|a\|_\infty := \max |a_i|, \qquad \|a\|_1 := \sum |a_i|, \qquad \|a\|_2 := \sqrt{\sum a_i^2}. \tag{4.10.7}$$

For example, the polynomials with norm $\|a\|_\infty \leqslant 1$ are the polynomials whose coefficients are in the set $\{-1, 0, 1\}$.

(1) Let $a, b \in \mathcal{R}$. Prove that
$$\|a \cdot b\|_\infty \leqslant \|a\|_\infty \cdot \|b\|_1. \tag{4.10.8}$$
Describe the a, b for which (4.10.8) is an equality.

(2) Let $a, b \in \mathcal{R}$. Prove that
$$\|a \cdot b\|_2 \leqslant N \cdot \|a\|_\infty \cdot \|b\|_2. \tag{4.10.9}$$
Describe the a, b for which (4.10.9) is an equality.

(3) Let $b \in \mathcal{R}$. Prove that
$$\frac{1}{3^N} \sum_{\|a\|_\infty = 1} \|a \cdot b\|_2^2 = N \cdot \|b\|_2^2.$$
This suggests that for most a satisfying $\|a\|_\infty = 1$, the upper bound (4.10.9) in (2) is off by a factor of $\sqrt{N}$.

---

[22] The astute reader will realize that this will not work, since then $f(1) = 0$, which means that $f(X)$ cannot have an inverse in $\mathcal{R}_q$. In practice, one chooses $f(X)$ to have $d+1$ coefficients equal 1 and d coefficients equal -1. This has a small numerical effect, but little practical effect, on the results of this problem.

[23] There are so-called collision algorithms that give a the attacker a square-root advantage; cf. Exercise 4.10.13.

[24] This cannot be literally true, of course, since the rotations (4.6.4) of $[f, g]$ have the same length as $[f, g]$. So the correct conclusion is that these N vectors are most likely the shortest non-zero vectors in L_h.

Exercise 4.10.10. Show that NTRU decryption can be described as Babai's algorithm applied to the short partial basis consisting of $[f, g]$ and its rotations.

Exercise 4.10.11. For NTRU, one generally takes N to be prime. This exercise illustrates why by describing a potential attack of Craig Gentry [11] when N is even, say $N = 2M$. In this case the polynomial $X^N - 1$ factors, and we get a homomorphism

$$\mathbb{Z}[X]/(X^{2M}-1) \longrightarrow \mathbb{Z}[X]/(X^M-1) \times \mathbb{Z}[X]/(X^M+1). \tag{4.10.12}$$

(1) What are the kernel and cokernel of the map (4.10.12)?
(2) If a is a small element of $\mathbb{Z}[X]/(X^{2M}-1)$, show that its images in
$$\mathbb{Z}[X]/(X^M-1) \quad \text{and} \quad \mathbb{Z}[X]/(X^M+1)$$
are small. (Give a quantitative estimate for the sup norms of the coefficients.)
(3) Set up a lattice reduction search for an NTRU private key $[f, g]$ as a pair of lattice searches in two lattices of dimension M. This cuts the dimension of the relevant lattices in half compared to the lattice in Section 4.6.

Exercise 4.10.13. Any cryptographic construction in which the private key is chosen from a finite set is susceptible to a *combinatorial* or *brute-force attack* in which the attacker simply checks the possible keys. And frequently the attacker can search for "collisions" to significantly speed up the process. In this exercise we describe one such attack. We set the notation

$$\mathcal{R}_q[1] \leftarrow \bigl\{ \mathbf{a} \in \mathcal{R}_q : \|\mathbf{a}\|_\infty \leqslant 1 \bigr\}$$

for the set of polynomials in $\mathcal{R}_q$ whose coefficients have size at most 1, i.e., for the set of polynomials whose coefficients are in $\{-1, 0, 1\}$.

(1) Prove that $\#\mathcal{R}_q[1] = 3^n$.
(2) Suppose that Alice chooses random polynomials $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q[1]$ and that her public key is the polynomial $\mathbf{h} = \mathbf{g} \cdot \mathbf{f}^{-1}$ in $\mathcal{R}_q$. An attacker tries to find Alice's private key by randomly choosing polynomials $\mathbf{a} \in \mathcal{R}_q[1]$ and checking whether $\mathbf{h} \cdot \mathbf{a}$ is in $\mathcal{R}_q[1]$. Show that the expected number of attempts is $\frac{1}{2}\#\mathcal{R}_q[1]$.
(3) We define two sets
$$\mathcal{R}_q[1]_0 \leftarrow \left\{ \sum_{0 \leqslant i \leqslant n/2} a_i x^i : a_i \in \{-1,0,1\} \right\},$$
$$\mathcal{R}_q[1]_1 \leftarrow \left\{ \sum_{n/2 < i \leqslant n-1} a_i x^i : a_i \in \{-1,0,1\} \right\}.$$
Prove that
$$\#\mathcal{R}_q[1]_0 \approx \#\mathcal{R}_q[1]_1 \approx \sqrt{\#\mathcal{R}_q[1]}.$$

(4) An attacker uses Alice's public key $\mathbf{h}$ to compile the following two lists of polynomials in $\mathcal{R}_q$:

$$\mathcal{S}_0 \leftarrow \{\mathbf{h}\cdot\mathbf{a} : \mathbf{a}\in\mathcal{R}_q[1]_0\} \quad\text{and}\quad \mathcal{S}_1 \leftarrow \{\mathbf{h}\cdot\mathbf{a} : \mathbf{a}\in\mathcal{R}_q[1]_1\}.$$

Prove that there exists an $\mathbf{h}\cdot\mathbf{a}_0\in\mathcal{S}_0$ and an $\mathbf{h}\cdot\mathbf{a}_1\in\mathcal{S}_1$ that are very close to one another; more precisely, they satisfy

$$\|\mathbf{h}\cdot\mathbf{a}_0 - \mathbf{h}\cdot\mathbf{a}_1\|_\infty \leqslant 1.$$

Prove that $\mathbf{a}_0-\mathbf{a}_1$ is (almost certainly) equal to $x^j\cdot\mathbf{f}$ for some $0\leqslant j<n$, an attacker who can compute this "almost collision" between the sets $\mathcal{S}_0$ and $\mathcal{S}_1$, has broken Alice's system.

(5) Show that after sorting $\mathcal{S}_0$ in some reasonable way, e.g., lexigraphically by coordinates, it is possible to check whether an element of $\mathcal{S}_1$ has an almost collision in $\mathcal{S}_0$ in roughly $O(\log n)$ steps. Deduce that using this collision-style attack, an attacker can break Alice's system in roughly $O\big(\sqrt{\#\mathcal{R}_q[1]}\cdot\log n\big)$ steps, which represents a substantial improvement on the brute force search described in (2).

Exercise 4.10.14. This exercise asks you to explore Remark 4.9.2.

(1) Generate some data to illustrate Remark 4.9.2. For example, generate a lot of random polynomials $\mathbf{a}(X),\mathbf{b}(X)$ of degree at most $N-1$ and compare the largest coefficient of the product $\mathbf{a}(X)\cdot\mathbf{b}(X)$ in the rings

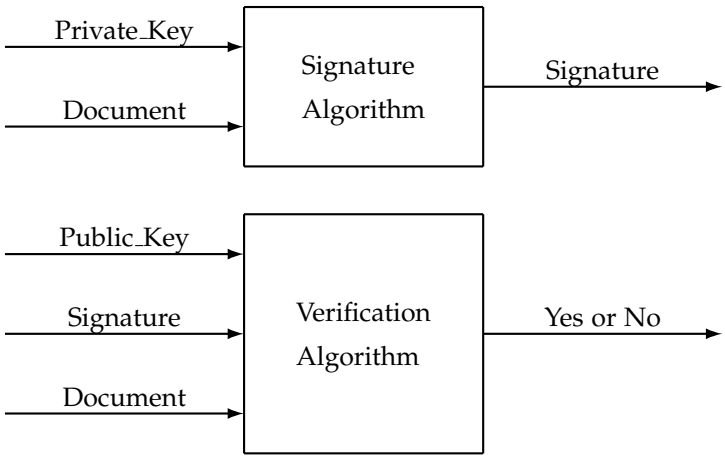
$$\mathbb{Z}[X]/(X^N-1),\qquad \mathbb{Z}[X]/(X^N+1),\qquad \mathbb{Z}[X]/(X^N-X-1).$$

Do this for, say, $N=7$ and/or $N=23$.

(2) Let $\boldsymbol{\phi}(X)\in\mathbb{Z}[X]$ be a monic polynomial of degree N. Give an heuristic argument as to why the largest coefficient of a product $\mathbf{a}(X)\mathbf{b}(X)$ in $\mathbb{Z}[X]/\boldsymbol{\phi}(X)\mathbb{Z}[X]$ should depend on the largest root of $\boldsymbol{\phi}(X)$ in $\mathbb{C}$. Make your estimate as quantitative as you can.

5. Lattice-Based Digital Signatures and Rejection Sampling

5.1. Digital Signatures. We recall that a *Digital Signature Scheme* consists of a signing function and a verification function:



If (PubKey, PrivKey) is a valid public/private key pair and Doc is a document and Sig is a purported signature, then we want the following to be true:

$$\text{Verify}(\text{PubKey}, \text{Sig}, \text{Doc}) = \text{Yes} \quad \Longleftrightarrow \quad \text{Sig} = \text{Sign}(\text{PrivKey}, \text{Doc}).$$

5.2. CVP Digital Signatures — GGH. In Section 3.6.3 we briefly discussed digital signature schemes based on the integer factorization problem (IFP), the discrete logarithm problem (DLP), and the elliptic discrete logarithm problem (ECDLP). In this section we discuss lattice-based digital signature schemes whose security relies on the shortest and/or closest vector problem (SVP/CVP).

The prototypical example of a lattice-based digital signature is the *GGH digital signature scheme*, which is described in Table 5.2.2. Its public and private keys are the same as that of the GGH lattice-based cryptosystem:, abd the signature is a solution to an approximate CVP problem:

$$\text{Private Key} = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n\} = \text{a good basis } \mathcal{B}^{\text{good}} \text{ for a lattice } \mathcal{L}.$$

$$\text{Public Key} = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n\} = \text{a bad basis } \mathcal{B}^{\text{bad}} \text{ for the lattice } \mathcal{L}.$$

$$\text{Document} = \text{a vector } \mathbf{d} \text{ that is } \underline{\text{not}} \text{ in } \mathcal{L}.$$

Remark 5.2.1. In practice, the "document" $\mathbf{d}$ that Alice signs is the output of a hash function applied to her actual document concatenated or otherwise combined with some random bits; cf. Section 3.7, Example 3.8.1, and Exercise 3.12.3

5.3. Security of GGH and other CVP-based Digital Signatures. Just as with CVP-based encryption schemes, there are many security issues that must be addressed for CVP-based digital signatures:

Cominatorial Security: The spaces of keys and signature must to be large enough so that an attacker cannot check all their elements. Further, one must take into account the fact that collision search algorithms often run in time $O(\sqrt{K})$ on sets of size K; cf. Exercises 4.10.13 and 5.8.10.

- Alice uses Babai's algorithm (Section 1.3) with $\mathcal{B}^{\text{good}}$ to find a lattice vector close to $\mathbf{d}$. To do this, she first writes
$$\mathbf{d} = \delta_1 \boldsymbol{v}_1 + \cdots + \delta_n \boldsymbol{v}_n \quad \text{with } \delta_1, \ldots, \delta_n \in \mathbb{R}.$$
Then she rounds the coefficients to get a lattice vector
$$\mathbf{s} = \lfloor \delta_1 \rceil \boldsymbol{v}_1 + \cdots + \lfloor \delta_n \rceil \boldsymbol{v}_n \in \mathcal{L}$$
that is close to $\mathbf{d}$.
- Alice writes $\mathbf{s}$ using the bad basis $\mathcal{B}^{\text{bad}}$,
$$\mathbf{s} = s_1 \boldsymbol{w}_1 + \cdots + s_n \boldsymbol{w}_n.$$
Her signature on $\mathbf{d}$ is the n-tuple $(s_1, \ldots, s_n)$.
- Bob uses the n-tuple $(s_1, \ldots, s_n)$ and the bad public basis $\mathcal{B}^{\text{bad}}$ to reconstruct $\mathbf{s}$ using the formula
$$\mathbf{s} = s_1 \boldsymbol{w}_1 + \cdots + s_n \boldsymbol{w}_n.$$
The vector $\mathbf{s}$ is automatically in $\mathcal{L}$, and Bob verifies that the signature is valid by checking that $\mathbf{s}$ is sufficiently close to $\mathbf{d}$.

Table 5.2.2. The GGH Lattice-Based Digital Signature Scheme

Lattice Security: Just as with CVP-based public key cryptosystems, one needs to check that latttice reduction algorithms such as LLL-BKZ cannot solve the approximate CVP well enough to forge signatures.

Practicality: The GGH digital signature scheme has impractically large public keys, since a public key is a complete basis of the lattice. One might try to instead use an NTRU-type construction, although it is not entirely clear how to do that, since an NTRU lattice only provides a good basis for a half-dimensional sublattice. However, such a construction is possible; see [20] for details.

Transcript Security: For digital signature schemes, here is

Yet Another Thing to Worry About!

- Each signature potentially reveals some information about the private key.
- So a long transcript of signatures might compromise security.

5.4. A Transcript Attack on the GGH Digital Signature Scheme. The possibility of a transcript attack is not an idle threat. A number of people, including Gentry–Szydlo [13] and Nguyen–Regev [26,27], developed practical transcript attacks on GGH and other early CVP-based signature schemes. We describe roughly how one might use a GGH transcript to recover the private key. A GGH signature $\mathbf{s}$ on a document $\mathbf{d}$ reveals a vector in the centered fundamental domain spanned by the good basis:
$$\mathbf{s} - \mathbf{d} \in \mathcal{F}^{\text{good}} := \left\{ t_1 \boldsymbol{v}_1 + \cdots + t_n \boldsymbol{v}_n : -\frac{1}{2} \leqslant t_i \leqslant \frac{1}{2} \right\}.$$

As illustrated in Figure 5.4.1 on page 58, a transcript consisting of only a few signatures is relatively innocuous, but a long transcript of signatures reveals a lot of information about the fundamental domain spanned by the good secret basis.

Note that Alice, Bob, and Carl each have their own private good bases, and that their bases define different fundamental domains. So transcripts of their signatures fill out their individual private fundamental domains, as illustrated in Figure 5.4.2 on page 59.

5.5. Rejection Sampling to the Rescue. *Rejection sampling* is a technique from statistics in which one starts with a random process whose output has a known distribution, and by judiciously throwing away (rejecting) some of the output values, creates a new random process whose output has some other desired distribution.

Lyubashevsky [24, 25] constructed a lattice-based identification scheme and digital signature using a technique that he called *aborting*, which is a version of rejection sampling.

The basic idea is that Alice generates a signature on her document and checks if it will leak information about her individual private key. If it will leak information, then she rejects that signature and generates another one. She continues to do this until finding a safe signature, which she publishes.

First Issue: How does Alice generate multiple signatures on the same document?

Remember that Alice isn't really signing her document, she's signing a hash of her document. In fact, for various reasons, it is advisable that she signs something like

$$\mathrm{Hash}\begin{pmatrix}\text{Alice's actual document concate-}\\ \text{nated with, say, 160 random bits}\end{pmatrix}.$$

So if Alice generates a signature and decides to reject it, she simply selects a new random 160 bits and tries again. Her signature would then include the 160 random bits that she uses in the signature that she doesn't reject.

How might this work for GGH? Alice's signatures give a set of points that are uniformly distributed in her fundamental domain $\mathcal{F}_{\text{Alice}}^{\text{good}}$. Similarly, Bob's signatures give points uniformly distributed in $\mathcal{F}_{\text{Bob}}^{\text{good}}$, and the same for Carl. But suppose that they fix a region that's common to all three fundamental domains; i.e., that is contained in $\mathcal{F}_{\text{Alice}}^{\text{good}}$, $\mathcal{F}_{\text{Bob}}^{\text{good}}$, and $\mathcal{F}_{\text{Carl}}^{\text{good}}$. Then they reject signatures that fall outside the common region. This is illustrated in Figure 5.5.1 on page 60, where the shaded box is common to everyone's centered fundamental domain. Hence when Alice, Bob, and Carl reveal sets of equidistributed points in the shaded box, they are yielding no information about their private bases.

5.6. Transcript Security — At A Cost. Assuming that the hash function and pseudo-random number generator work as advertised, Alice's and Bob's and Carl's transcripts will have exactly the same distribution, namely they will be

1. The public parameters include a cut-off value B with the property that everyone's centered fundamental domain $\mathcal{F}^{good}$ contains the centered box
$$\left\{(x_1, \ldots, x_n) \in \mathbb{R}^n : -\frac{1}{2}B \leqslant x_i \leqslant \frac{1}{2}B\right\}.$$
2. Alice computes a vector $\mathbf{d}$ to sign:
$$\mathbf{d} \leftarrow \mathrm{Hash}(\text{Alice's Doc} \,\|\, \text{Random Bits}).$$
3. Alice uses her good basis $\mathcal{B}^{good}$ to find a lattice vector $\mathbf{s}$ that is close to $\mathbf{d}$.
4. If
$$\|\mathbf{s} - \mathbf{d}\|_\infty > B,$$
then Alice rejects $\mathbf{s}$, returns to Step 2, and selects new random bits.
5. Otherwise Alice accepts and publishes the signature $\mathbf{s}$ and the random bits that she used in Step 2. Note that it is only at this step that $\mathbf{s}$ becomes public, so an attacker never gets to see the rejected signatures.

Table 5.5.2. The GGH Digital Signature Scheme with Rejection Sampling

equidistributed over the lattice points in the common box. Thus the transcripts provide no useful information about the private key.

However, we have the added cost of rejection sampling. This leads to the following imporatant question:

How many signatures are rejected before one of them is accepted?

For a naive implementation of GGH, the volume of the common box will be very small compared to the volume of the fundamental domain. So using rejection sampling is at best inefficient, and at worst, completely impractical.

Lyubashevsky and others use various methods to make it easier to find non-rejected signatures, while maintaining the property that the distribution of signatures is independent of the private key. For example, a signature scheme called BLISS devised by Ducas, Durmus, Lepoint, and Lyubashevsky [10] makes use of an NTRU-like key generation procedure and a bimodal discrete Gaussian noise distribution to produce compact equidistributed signatures. The efficiency of the scheme is quite good, especially considering the complexity of sampling discrete Gaussian distributions.

There are also lattice-based signature schemes for which the simple box rejection method is practical. These include a variant of NTRU described in [20], and another older scheme based on partial evaluation of polynomials called PASS [22]. In the next section we use PASS to illustrate how to prove that rejection sampling gives transcript security, although we will limit ourselves to describing the transcript aspects of PASS and leave a full description of the PASS signature scheme to the exercises.

5.7. An Example of a Lattice Recovery Problem and Rejection Sampling In this section we describe how a short basis of a certain type of lattice can be recovered from a long list of CVP solutions, and how applying rejection sampling to the list of solutions prevents recovery of the short basis. In the exercises we explain how these lattices can be used to create a digital signature schemee.

We recall from Remark 4.4.5 that we defined a quotient ring

$$\mathcal{R} = \mathbb{Z}[X]/(X^N - 1),$$

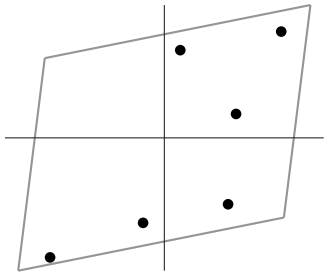
and that we identified each element of $\mathcal{R}$ with its vector of coefficients in $\mathbb{Z}^N$. We also defined the *sup norm* of a polyomial to be the maximum of its coefficients, i.e.,

$$\text{the sup norm of} \quad f(X) = \sum_{i=0}^{N-1} a_i X^i \in \mathcal{R} \quad \text{is} \quad \|f\|_\infty := \max_{0 \leqslant i < N} |a_i|.$$

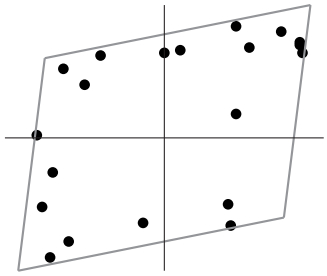
We let

$$\mathcal{R}[b] = \bigl\{ f \in \mathcal{R} : \|f\|_\infty \leqslant b \bigr\},$$

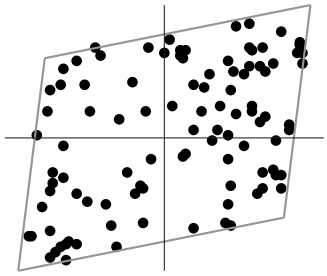
so for example, the polynomials in $\mathcal{R}[1]$ have coefficients in the set $\{-1, 0, 1\}$.



A few signatures aren't much help in reconstructing $\mathcal{F}^{good}$.

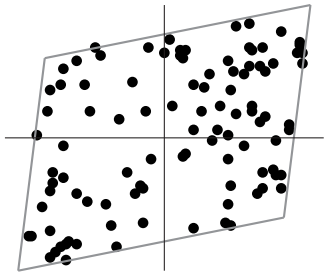


With a moderate number of signatures, the fundamental domain $\mathcal{F}^{good}$ starts to emerge.

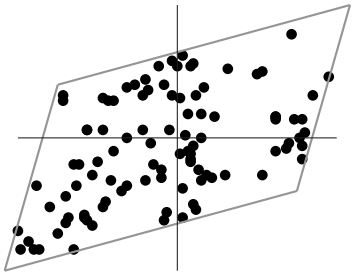


With lots of signatures, one may be able to reconstruct the fundamental domain $\mathcal{F}^{good}$.

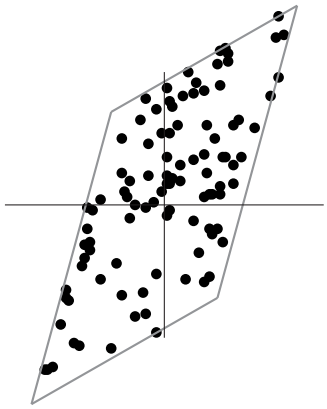
Figure 5.4.1. Illustrating a Transcript Attack



A Fundamental Domain $\mathcal{F}^{\text{good}}_{\text{Alice}}$ for Alice's Private Basis



A Fundamental Domain $\mathcal{F}^{\text{good}}_{\text{Bob}}$ for Bob's Private Basis



A Fundamental Domain $\mathcal{F}^{\text{good}}_{\text{Carl}}$ for Carl's Private Basis

Figure 5.4.2. Different Private Good Bases Have Different Transcripts

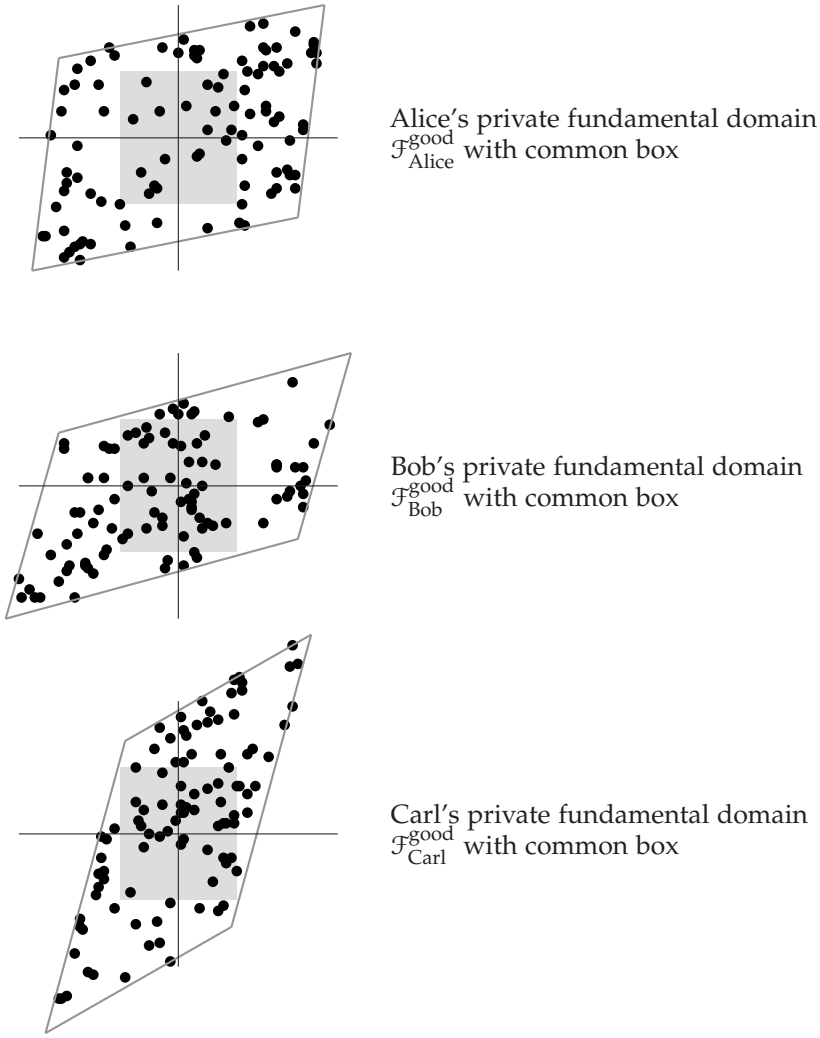


Figure 5.5.1. Selecting a Region Common to All Fundamental Domains

Algorithm 5.7.1 (Prototypical Rejection Sampling Algorithm).

1. **Parameter Selection:** *Fix a lattice dimension parameter* N *and a norm bound* k.
2. **Secret Lattice Creation:** *Alice chooses a small polynomial* $\mathbf{f} \in \mathcal{R}[1]$ *that determines her secret lattice.*
3. **Random Polynomial Selection:** *Alice chooses a random polynomial* $\mathbf{y} \in \mathcal{R}[k]$.
4. **Hash Function:** *A hash function is applied to certain quantities associated to* $\mathbf{f}$ *and* $\mathbf{y}$. *The output from the hash function is a polynomial* $\mathbf{c} \in \mathcal{R}[1]$ *that depends randomly on the inputs.*[25]
5. **Signature Creation:** *Alice computes the polynomial*
$$\mathbf{s} = \mathbf{f} \star \mathbf{c} + \mathbf{y} \quad \textit{in the ring } \mathcal{R}.$$
6. **Rejection Sampling:** *If*
$$\|\mathbf{s}\|_\infty \geqslant k - N,$$
then Alice goes back to Step 3 *and selects a new value for* $\mathbf{y}$.
5. **Publication:** *Alice publishes the pair of polynomials* $(\mathbf{s}, \mathbf{c})$ *as her signature.*

Suppose that Alice uses Algorithm 5.7.1 to create a long list of signatures

$$(\mathbf{s}_1, \mathbf{c}_1),\ (\mathbf{s}_2, \mathbf{c}_2),\ (\mathbf{s}_3, \mathbf{c}_3),\ \dots\ .$$

We are going to prove that this transcript reveals no information about Alice's private key $\mathbf{f}$, but that if Alice were to skip the rejection sampling step, then information would be revealed. We also estimate the probability that a proposed signature makes it through the rejction step, since as noted in Section 5.6, rejection sampling comes a cost, and an excessively high rejection rate could make the scheme impractical. We also note that the constraint $k > N^2$ in Proposition 5.7.2 must be balanced against the underlying CVP when the system is turned into a full-fledged digital signature scheme; see Figure 5.8.7 for details.

Proposition 5.7.2. *Let*

$$(\mathbf{s}_1, \mathbf{c}_1),\ (\mathbf{s}_2, \mathbf{c}_2),\ (\mathbf{s}_3, \mathbf{c}_3),\ \dots\ . \tag{5.7.3}$$

be a transcript of signatures created using Algorithm 5.7.1.

(1) If the $\mathbf{c}_i$ *are uniformly and randomly distributed in* $\mathcal{R}[1]$,[26] *then the transcript* (5.7.3) *reveals no information about the private key* $\mathbf{f}$.

(2) If instead the transcript is created skipping the rejection sampling step in Algorithm 5.7.1, then the transcript will contain information allowing recovery of the private key $\mathbf{f}$.

[25] Since we are only interested in transcripts at the moment, we will not explain exactly how $\mathbf{c}$ depends on $\mathbf{f}$ and $\mathbf{y}$. It suffices to know that $\mathbf{c}$ behaves like a random variable uniformly distributed in $\mathcal{R}[1]$. See Figure 5.8.7 for further details.

[26] We are being somewhat informal here. In a formal security proof we would assume that the algorithm that creates the $\mathbf{c}_i$ may be modeled as a random oracle whose output is completely unpredictable.

(3) If $k > N^2$, then the probability that a signature passes the rejection sampling step in Algorithm 5.7.1 (roughly) satisfies

$$\text{Prob}\Big(\|\mathbf{s}\|_\infty < k-N\Big) \gtrapprox \left(1-\frac{N}{2k+1}\right)^N \approx e^{-N^2/(2k+1)}.$$

Proof. (1) What does it mean to say that the transcript reveals no information about $\mathbf{f}$? The $\mathbf{c}_i$ polynomials are assumed to be randomly distributed, so the question is to what extent the $\mathbf{s}_i$ values contain information about $\mathbf{f}_i$. We claim that the probability that any particular valid $\mathbf{s}$ shows up in a signature does not depend on the polynomial $\mathbf{f}$. We can formulate this as a statement in conditional probability as follows:

Claim. For all $\mathbf{f}_0 \in \mathcal{R}[1]$ and all $\mathbf{s}_0 \in \mathcal{R}[k-N]$,

$$\text{Prob}\left(\begin{matrix}\text{a signature } (\mathbf{s},\mathbf{c}) \text{ created}\\ \text{using } \mathbf{f} \text{ satisfies } \mathbf{s}=\mathbf{s}_0\end{matrix}\;\middle|\;\mathbf{f}=\mathbf{f}_0\right) \quad \text{does not depend on } \mathbf{f}_0.$$

The probability is computed over the space of randomly chosen polynomials $\mathbf{c} \in \mathcal{R}[1]$ and $\mathbf{y} \in \mathcal{R}[k]$, so we compute

$$\begin{aligned}\operatorname*{Prob}_{\substack{\mathbf{c}\in\mathcal{R}[1]\\ \mathbf{y}\in\mathcal{R}[k]}}\big(\mathbf{s}=\mathbf{s}_0 \;\big|\; \mathbf{f}=\mathbf{f}_0\big) &= \frac{1}{\#\mathcal{R}[1]\cdot\#\mathcal{R}[k]}\cdot\#\Big\{(\mathbf{c},\mathbf{y})\in\mathcal{R}[1]\times\mathcal{R}[k] \;\Big|\; \mathbf{s}_0=\mathbf{c}\cdot\mathbf{f}_0+\mathbf{y}\Big\}\\ &= \frac{1}{\#\mathcal{R}[1]\cdot\#\mathcal{R}[k]}\cdot\#\Big\{\mathbf{c}\in\mathcal{R}[1] \;\Big|\; \mathbf{s}_0-\mathbf{c}\cdot\mathbf{f}_0\in\mathcal{R}[k]\Big\}. \qquad (5.7.4)\end{aligned}$$

We next use the fact that $\mathbf{c}$ and $\mathbf{f}_0$ are in $\mathcal{R}[1]$ to bound the coefficients of their product. More generally, for $\mathbf{a},\mathbf{b}\in\mathcal{R}$, we can use the triangle inequality to prove the bound[27]

$$\|\mathbf{a}\star\mathbf{b}\|_\infty = \max_{0\leqslant n<N}\left|\sum_{i+j\equiv n \pmod N} a_i b_j\right| \leqslant N\|\mathbf{a}\|_\infty\cdot\|\mathbf{b}\|_\infty.$$

Hence for all $\mathbf{s}_0\in\mathcal{R}[k-N]$ and all $\mathbf{c},\mathbf{f}_0\in\mathcal{R}[1]$ we have

$$\begin{aligned}\big\|\mathbf{s}_0-\mathbf{c}\cdot\mathbf{f}_0\big\|_\infty &\leqslant \big\|\mathbf{s}_0\big\|_\infty + \big\|\mathbf{c}\cdot\mathbf{f}_0\big\|_\infty\\ &\leqslant \big\|\mathbf{s}_0\big\|_\infty + N\big\|\mathbf{c}\big\|_\infty\cdot\big\|\mathbf{f}_0\big\|_\infty\\ &\leqslant (k-N)+N\\ &= k.\end{aligned}$$

Thus the condition $\mathbf{s}_0-\mathbf{c}\star\mathbf{f}_0\in\mathcal{R}[k]$ in the conditional probability (5.7.4) is vacuous. This yields

$$\operatorname*{Prob}_{\substack{\mathbf{c}\in\mathcal{R}[1]\\ \mathbf{y}\in\mathcal{R}[k]}}\big(\mathbf{s}=\mathbf{s}_0 \;\big|\; \mathbf{f}=\mathbf{f}_0\big) = \frac{1}{\#\mathcal{R}[1]\cdot\#\mathcal{R}[k]}\cdot\#\big\{\mathbf{c}\in\mathcal{R}[1]\big\} = \frac{1}{\#\mathcal{R}[k]}.$$

In particular, the probability does not depend on the choice of $\mathbf{f}_0\in\mathcal{R}[1]$, and indeed, the probability is also independent of $\mathbf{s}_0\in\mathcal{R}[k-N]$.

[27] We remark that this estimate also follows from Exercise 4.10.6 combined with the trivial triangle inequality bound $\|\mathbf{b}\|_1 \leqslant N\|\mathbf{b}\|_\infty$.

(2) Exercise 4.10.1 in Lecture 4 defines the *reversal* (or *conjugate*) of a polynomial $\mathbf{a}(X) \in \mathcal{R}$ to be the polynomial

$$\overline{\mathbf{a}}(X) \leftarrow \mathbf{a}(X^{N-1}) \in \mathcal{R}.$$

In that exercise you verified that the map

$$\mathcal{R} \longrightarrow \mathcal{R}, \quad \mathbf{a} \longmapsto \overline{\mathbf{a}},$$

is a ring automorphism, and you computed the average values

$$\frac{1}{\#\mathcal{R}[B]} \sum_{\mathbf{a} \in \mathcal{R}[B]} \mathbf{a} = \mathbf{0} \qquad \text{and} \qquad \frac{1}{\#\mathcal{R}[B]} \sum_{\mathbf{a} \in \mathcal{R}[B]} \mathbf{a} \star \overline{\mathbf{a}} = \frac{2N}{3}(B+1). \tag{5.7.5}$$

We note in particular that the latter average is a non-zero constant polynomial in $\mathcal{R}$.

Suppose that we have a transcript that contains T different signatures. For each signature $(\mathbf{s}_i, \mathbf{c}_i)$ in the transcript, we multiply $\mathbf{s}_i$ by $\overline{\mathbf{c}}_i$, and then we take the average over the signatures in the transcript. This yields

$$\begin{aligned}
\frac{1}{T}\sum_{i=1}^{T} \mathbf{s}_i \star \overline{\mathbf{c}}_i &= \frac{1}{T}\sum_{i=1}^{T} (\mathbf{f} \star \mathbf{c}_i + \mathbf{y}_i) \star \overline{\mathbf{c}}_i \\
&= \mathbf{f} \star \left(\frac{1}{T}\sum_{i=1}^{T} \mathbf{c}_i \star \overline{\mathbf{c}}_i \right) + \frac{1}{T}\sum_{i=1}^{T} \mathbf{y}_i \star \mathbf{c}_i \\
&\approx \mathbf{f} \star \left(\frac{1}{\#\mathcal{R}[1]} \sum_{\mathbf{c} \in \mathcal{R}[1]} \mathbf{c} \star \overline{\mathbf{c}} \right) + \left(\frac{1}{\#\mathcal{R}[k]} \sum_{\mathbf{y} \in \mathcal{R}[k]} \mathbf{y} \right) \star \left(\frac{1}{\#\mathcal{R}[1]} \sum_{\mathbf{c} \in \mathcal{R}[1]} \overline{\mathbf{c}} \right) \\
&\qquad \text{since the } \mathbf{c}_i \text{ are uniformly distributed in } \mathcal{R}[1], \text{ the } \mathbf{y}_i \\
&\qquad \text{are uniformly distributed in } \mathcal{R}[k], \text{ and the } \mathbf{c}_i \text{ and } \mathbf{y}_i \\
&\qquad \text{are independent random variables,} \\
&= \mathbf{f} \star \frac{4N}{3} + \mathbf{0} \quad \text{applyiing (5.7.5) with } (\mathbf{a}, B) = (\mathbf{c}, 1) \text{ and } (\mathbf{y}, k).
\end{aligned}$$

Hence if T is sufficiently large, then rounding the coefficients of

$$\frac{3}{4NT}\sum_{i=1}^{T} \mathbf{s}_i \star \overline{\mathbf{c}}_i$$

to the nearest integer is likely to yield the secret polynomial $\mathbf{f}$.

(3) The idea is that the coefficients of $\mathbf{c} \star \mathbf{f}$ are more-or-less a random sum of N terms, where each term is a product of two independent random variables that are uniformly distributed in $\{-1, 0, 1\}$. Hence each coefficient of $\mathbf{c} \star \mathbf{f}$ is an N-step random walk, where each step has a $\frac{5}{9}$ probability of being 0 and a $\frac{2}{9}$ probability of being either $+1$ or -1. Hence it is unlikely that there will be a coefficient whose magnitude is much larger than a small multiple $\sqrt{N}$. Since we are assuming that k is considerably larger than N, the probability that $\|\mathbf{f} \star \mathbf{c} + \mathbf{y}\|_\infty \leqslant k - N$ is roughly the same as the probability that $\mathbf{y}$ itself satisfies $\|\mathbf{y}\|_\infty \leqslant k - N$. This allows us to

estimate

$$\begin{aligned}
\operatorname*{Prob}_{\substack{\mathbf{c}\leftarrow\mathcal{R}[1]\\ \mathbf{y}\leftarrow\mathcal{R}[k]}}\Big(\|\mathbf{c}\star\mathbf{f}+\mathbf{y}\|_\infty \leqslant k-N\Big)
&\approx \operatorname*{Prob}_{\mathbf{y}\leftarrow\mathcal{R}[k]}\Big(\|\mathbf{y}\|_\infty \leqslant k-N\Big)\\
&\approx \prod_{i=0}^{N-1} \operatorname*{Prob}_{\mathbf{y}\leftarrow\mathcal{R}[k]}\Big(\big|i\text{th coefficient of }\mathbf{y}\big| \leqslant k-N\Big)\\
&\qquad \text{since the coefficients are more-or-less independent,}\\
&= \prod_{i=0}^{N-1} \frac{2(k-N)+1}{2k+1}\\
&= \left(1-\frac{N}{2k+1}\right)^N.
\end{aligned}$$

We have proven the first estimate, and for the second we use $(1-t)^N \approx e^{-tN}$, which is valid provided that tN is small. □

Remark 5.7.6. In the proof of Proposition 5.7.2(3) we ignored the $\mathbf{c}\star\mathbf{f}$ term as being negligible. For those who object to this cavalier attitude, we explain in more detail what happens if we include the $\mathbf{c}\star\mathbf{f}$ term. As noted in the proof, the coefficients of $\mathbf{c}\star\mathbf{f}$ may be modeled by an $\frac{4}{9}N$-step random walk in which each step is equally likely to be $+1$ and -1. If we write W for the distance from the origin at the end of the walk, then a well-known calculation says that the expected values of the powers of W are

$$\mathbb{E}(W)=0,\quad \mathbb{E}(W^2)=\frac{4}{9}N,\quad \mathbb{E}(W^3)=0,\quad \mathbb{E}(W^4)\approx\frac{16}{27}N^2;$$

see Exercise 5.8.1. So roughly speaking, we expect that the absolute value of a random coefficient of $\mathbf{c}\star\mathbf{f}$ to be around $0.67\sqrt{N}$, and the standard deviation of this value to be around $0.63N$. The probability that a coefficient has magnitude that is (say) 8 standard deviations from the mean is vanishingly small. So we make the reasonable assumption that every coefficient of $\mathbf{c}\star\mathbf{f}$ is at most $5\sqrt{N}$. Then a small modification of our computation yields

$$\operatorname*{Prob}_{\substack{\mathbf{c}\leftarrow\mathcal{R}[1]\\ \mathbf{y}\leftarrow\mathcal{R}[k]}}\Big(\|\mathbf{c}\star\mathbf{f}+\mathbf{y}\|_\infty \leqslant k-N\Big) \gtrapprox = \left(1-\frac{N+5\sqrt{N}}{2k+1}\right)^N.$$

In practice, the dimension is taken to satisfy at least $N \geqslant 1000$, and we can see how little difference the the $\mathbf{c}\star\mathbf{f}$ terms makes by computing

$$N=1000 \text{ and } K=N^2$$
$$\Longrightarrow\quad \left(1-\frac{N}{2k+1}\right)^N \approx 0.606 \quad\text{and}\quad \left(1-\frac{N+5\sqrt{N}}{2k+1}\right)^N \approx 0.560.$$

As a practical matter, it makes little difference whether Alice has a 60% success rate or a 56% success rate in her signature generation routine.

5.8. Exercises for Lecture 5.

Exercise 5.8.1. Let $X_1, X_2, X_3, \ldots, X_n$ be independent random variables with the property that

$$\text{Prob}(X_i = 1) = \frac{1}{2} \quad \text{and} \quad \text{Prob}(X_i = -1) = \frac{1}{2}.$$

Let

$$W = X_1 + X_2 + \cdots + X_n.$$

Compute the expected values of W, W^2, W^3, and W^4 as functions of n.

Exercise 5.8.2. This exercise illustrates an elementary example of rejection sampling. Suppose that a pair of dice is rolled many times, that we write R_k for the value of the kth roll, and that we create a value A_k by the rule

$$A_k = \begin{cases} H & \text{if } R_k \leqslant 6, \\ T & \text{if } R_k \geqslant 8, \\ \text{Reject} & \text{if } R_k = 7. \end{cases}$$

(1) Prove that the sequence $A_1, A_2, A_3, \ldots$ is uniformly distributed on the set $\{H, T\}$. We have thus converted a non-uniformly distributed random variable R into a uniformly distributed random variable A.
(2) About how many times would we need to sample R in order to expect to get n samples of A? In other words, about how many dice rolls would we need in order to simulate n coin flips?
(3) Generalize as much as you can!

Series of Exercises for the PASS Digital Signature Scheme

Exercises 5.8.4–5.8.8 describes a transcript secure digital signature scheme called PASS[28] that is based on partial evaluation of polynomials that have small coefficients. The transcript security was proved in Section 5.7, so we concentrate on describing how PASS works and why there is an underlying lattice problem. For those interested in the history of such schemes, the original idea of using partial evaluation of small polynomials appears in [17], and simplifications and methods to improve efficiency were described in [18, 21]. Then, after the introduction of rejection sampling methods to achieve transcript security in lattice-based signature schemes, it was noted in [22] that such methods are particularly easy to implement for PASS.

We assume for Exercises 5.8.4–5.8.8 that the following (public) parameters have been fixed:

$$\begin{aligned} N &= \text{a moderate size prime, say } 500 < N < 5000 \\ q &= \text{a prime satisfying } q \equiv 1 \pmod{N} \\ \zeta &= \text{a generator (primitive } N\text{th root of unity) in } \mathbb{F}_q^* \end{aligned}$$

[28]For those who are curious, PASS is an acronymn for Polynomial Authentication and Signature Scheme.

We recall from Definition 4.4.2 that we defined the convolution product of two vectors by the formula (4.4.2), and that Proposition 4.4.4 says that coordinate-wise addition and convolution product give a ring structure to $\mathbb{F}_q^N$, i.e.,

$$(\mathbb{F}_q^N, +, \star) \text{ is a ring.}$$

Alternatively, Exercise 4.10.3 says that if we identify the elements of $\mathbb{F}_q^N$ with the coefficients of polynomials of degree $N-1$, then the convolution product is given by polynomial multiplication in the quotient ring

$$\mathcal{R}_q = \mathbb{F}_q[X]/(X^N - 1).$$

We observe that there is another, easier, way to make $\mathbb{F}_q^N$ into a ring, namely via coordinate-wise multiplication. We define

$$\mathbf{a} \odot \mathbf{b} = (a_0 b_0,\, a_1 b_1,\, \ldots,\, a_{N-1} b_{N-1}),$$

and then

$$(\mathbb{F}_q^N, +, \odot) \text{ is a ring.} \tag{5.8.3}$$

We use the generator $\zeta \in \mathbb{F}_q^*$ to define the *discrete Fourier transform* (DFT) map

$$\mathcal{F} : R_q \longrightarrow \mathbb{F}_q^N, \quad \mathcal{F}\big(\mathbf{f}(X)\big) \leftarrow \big(\mathbf{f}(\zeta^i)\big)_{0 \leqslant i < N}.$$

Exercise 5.8.4. Prove that $\mathcal{F}$ is a ring isomorphism, where $\mathbb{F}_q^N$ is given a ring structure via (5.8.3).

The idea underlying PASS is to that the secret key is a polynomial $\mathbf{f}$ having small coefficients, and the public key is a subset of the coordinates of the DFT of $\mathbf{f}$. There will be many elements of $\mathcal{R}_q$ having those DFT coordinates, but very few of them will themselves have small coefficients. This leads to the following problem

Definition 5.8.5. Fix a subset $\Omega \subset \{0, 1, \ldots, N-1\}$. The *small partial DFT inversion problem* is as follows: For a given set of values $(b_i)_{i \in \Omega}$, find a polynomial $\mathbf{f}(X) \in \mathcal{R}_q$ satisfying

$$\mathbf{f} \text{ has small coefficients} \quad \text{and} \quad \mathbf{f}(\zeta^i) = b_i \quad \text{for all } i \in \Omega,$$

or prove that no such $\mathbf{f}(X)$ exists. To ease notation, we let

$$t = \#\Omega, \quad \text{and we write} \quad \mathcal{F}_\Omega : \mathcal{R}_q \longrightarrow \mathbb{F}_q^t, \quad \mathcal{F}_\Omega(\mathbf{f}) = \big(\mathbf{f}(\zeta^i)\big)_{i \in \Omega},$$

for the *partial DFT associated to the set* $\S$.

Exercise 5.8.6.

(1) Prove that

$$\mathcal{F}_\Omega : \mathcal{R}_q = \mathbb{F}_q^N \longrightarrow \mathbb{F}_q^t$$

is a surjective $\mathbb{F}_q$-linear transformation

(2) We compose $\mathcal{F}_\Omega$ with the reduction modulo q map and consider the function

$$\overline{\mathcal{F}}_\Omega : \mathbb{Z}^N \xrightarrow[\text{modulo } q]{\text{reduction}} \mathbb{F}_q^N = \mathcal{R}_q \xrightarrow{\mathcal{F}_\Omega} \mathbb{F}_q^t.$$

Prove that

$$\mathrm{Null}(\overline{\mathcal{F}}_\Omega) := \bigl\{\mathbf{f} \in \mathbb{Z}^N : \overline{\mathcal{F}}_\Omega(\mathbf{f}) = \mathbf{0}\bigr\}$$

is an N-dimensional lattice in $\mathbb{Z}^N$. (*Hint*. Note that $\mathrm{Null}(\overline{\mathcal{F}}_\Omega)$ contains the lattice $(q\mathbb{Z})^N$.)

(3) Prove that

$$\mathrm{Det}\bigl(\mathrm{Null}(\overline{\mathcal{F}}_\Omega)\bigr) = q^t.$$

(4) Use (3) and (4) and the Gaussian heuristic (Section 1.6) to estimate the length of the shortest non-zero vector in $\mathrm{Null}(\overline{\mathcal{F}}_\Omega)$

Private Key Creation: Alice's private key is a small polynomial $\mathbf{f} \in \mathcal{R}_q$, say satisfying $\|\mathbf{f}\|_\infty \leqslant 1$.
Public Key Creation: Alice's public key is $\mathbf{F} := \mathcal{F}_\Omega(\mathbf{f}) \in \mathbb{F}_q^t$, the partial DFT of $\mathbf{f}$.
Signing: Alice wants to sign the document Doc for her public key $\mathbf{F}$.

(1) (Commitment) Alice chooses a random small polynomial $\mathbf{y} \in \mathcal{R}_q$, say satisfying $\|\mathbf{y}\|_\infty \leqslant k$.

(2) (Challenge) Alice uses a hash function to compute a small polynomial

$$\mathbf{c} := \mathrm{Hash}\bigl(\mathrm{Doc}, \mathcal{F}_\Omega(\mathbf{y}), \mathbf{F}\bigr),$$

where say $\|\mathbf{c}\|_\infty \leqslant 1$. (Note how $\mathbf{c}$ serves to tie Alice's signature to her document, her public key, and her random commitment.)

(3) (Response) Alice computes

$$\mathbf{s} := \mathbf{c} \star \mathbf{f} + \mathbf{y}.$$

(4) (Rejection Sampling) If $\|s\|_\infty$ is large, say $\|s\|_\infty \geqslant k - N$, then Alice goes back to Step (2) and chooses a new random commitment polynomial.

(5) Alice publishes her signature

$(\mathbf{s}, \mathbf{c})$ on the document Doc for the public key $\mathbf{F}$.

Verification: Bob wants to check the validity of the signature $(\mathbf{s}, \mathbf{c})$ on the document Doc for Alice's public key $\mathbf{F}$.

(1) Bob computes

$$\mathbf{Y} := \mathcal{F}_\Omega(\mathbf{s}) - \mathcal{F}_\Omega(\mathbf{c}) \odot \mathbf{F}.$$

(2) Bob accepts the signature as valid if the following two statements are true:

- $\mathbf{s}$ is small, say $\|\mathbf{s}\|_\infty < k - N$.
- $\mathbf{c}$ is equal to $\mathrm{Hash}(\mathrm{Doc}, \mathbf{Y}, \mathbf{F})$.

Figure 5.8.7. The PASS Digital Signature Scheme

Exercise 5.8.8. The PASS Digital Signature Scheme is described in Figure 5.8.7. Prove that if Alice uses the signing procedure in Figure 5.8.7, then Bob will accept her signature as valid.

Exercise 5.8.9. This exercise explains how to formulate the small partial DFT inversion problem (Definition 5.8.5) as a closest vector problem in a certain lattice

of dimension N and discriminant q^t. So we assume that we are given a vector $\mathbf{F} \in \mathbb{F}_q^t$, and that we are told that there is a vector $\mathbf{f} \in \mathbb{Z}^N$ satisfying

$$\|\mathbf{f}\|_\infty \leqslant 1 \quad \text{and} \quad \overline{\mathcal{F}}_\Omega(\mathbf{f}) = \mathbf{F},$$

where $\overline{\mathcal{F}}_\Omega$ is the map described in Exercise 5.8.6. Our goal is to find $\mathbf{f}$.

(1) Explain why it is easy to find a vector $\mathbf{f}_0 \in \mathbb{Z}^N$ satisfying $\overline{\mathcal{F}}_\Omega(\mathbf{f}_0) = \mathbf{F}$, provided that we do not require that $\mathbf{f}_0$ be small.

(2) Prove that

$$\bigl\{\mathbf{g} \in \mathbb{Z}^N : \overline{\mathcal{F}}_\Omega(\mathbf{g}) = \mathbf{F}\bigr\} = \mathbf{f}_0 + \operatorname{Null}(\overline{\mathcal{F}}_\Omega).$$

(3) Suppose that the length of the shortest non-zero vector in $\operatorname{Null}(\overline{\mathcal{F}}_\Omega)$ is given by the Gaussian heuristic; see Exercise 5.8.6. Prove that $\mathbf{f}_0 - \mathbf{f}$ solves the CVP for the lattice $\operatorname{Null}(\overline{\mathcal{F}}_\Omega)$ and the target vector $\mathbf{f}_0$.

Exercise 5.8.10. This exercise describes a collision algorithm to solve the small partial DFT inversion problem; cf. Exericse 4.10.13. We suppose that there is an unknown polynomial $\mathbf{f}(X) \in \mathcal{R}[1]$, i.e., having coefficients in $\{-1, 0, 1\}$, and we further suppose that we are given the value of $\mathcal{F}_\Omega(\mathbf{f})$. We define two sets

$$\mathcal{R}[1]_0 := \biggl\{ \sum_{0 \leqslant i \leqslant n/2} a_i x^i : a_i \in \{-1, 0, 1\} \biggr\},$$

$$\mathcal{R}[1]_1 := \biggl\{ \sum_{n/2 < i \leqslant n-1} a_i x^i : a_i \in \{-1, 0, 1\} \biggr\},$$

whose size is roughly $\sqrt{\#\mathcal{R}}$, and we use them to create two lists

$$\mathcal{S}_0 := \bigl\{\mathcal{F}_\Omega(\mathbf{a}) : \mathbf{a} \in \mathcal{R}[1]_0\bigr\} \quad \text{and} \quad \mathcal{S}_1 := \bigl\{\mathcal{F}_\Omega(\mathbf{f}) - \mathcal{F}_\Omega(\mathbf{a}) : \mathbf{a} \in \mathcal{R}[1]_1\bigr\}.$$

(1) Prove that $\mathcal{S}_0 \cap \mathcal{S}_1 \neq \emptyset$.

(2) Prove that if

$$\mathcal{F}_\Omega(\mathbf{a}_0) \in \mathcal{S}_0 \quad \text{and} \quad \mathcal{F}_\Omega(\mathbf{f}) - \mathcal{F}_\Omega(\mathbf{a}_1) \in \mathcal{S}_1$$

satisfy

$$\mathcal{F}_\Omega(\mathbf{a}_0) = \mathcal{F}_\Omega(\mathbf{f}) - \mathcal{F}_\Omega(\mathbf{a}_1),$$

then (almost certainly) $\mathbf{f} = \mathbf{a}_0 + \mathbf{a}_1$.

(3) For a given element of $\mathcal{S}_1$, explain how to check whether it is in the set $\mathcal{S}_0$ in roughly $O(\log n)$ steps. Deduce that one can recover $\mathbf{f}$ in roughly $O\bigl(\sqrt{\#\mathcal{R}[1]}\log n\bigr)$ steps.

References

[1] M. Ajtai and C. Dwork, *A public-key cryptosystem with worst-case/average-case equivalence*, STOC '97 (El Paso, TX), ACM, New York, 1999, pp. 284–293. MR1715640 ←41

[2] M. R. Albrecht, L. Ducas, G. Herold, E. Kirshanova, E. W. Postlethwaite, and M. Stevens, *The general sieve kernel and new records in lattice reduction*, Advances in cryptology—EUROCRYPT 2019. Part II, Lecture Notes in Comput. Sci., vol. 11477, Springer, Cham, 2019, pp. 717–746. MR3964651 ←22

[3] M. R. Albrecht, F. Göpfert, F. Virdia, and T. Wunderer, *Revisiting the expected cost of solving uSVP and applications to LWE*, Advances in cryptology—ASIACRYPT 2017. Part I, Lecture Notes in Comput. Sci., vol. 10624, Springer, Cham, 2017, pp. 297–322. MR3747701 ←24

[4] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwab, *Post-quantum key exchange: A new hope*, 25th USENIX Conference on Security Symposium, SEC16, USENIX Association, 2016, pp. 327343. ←24

[5] H. F. Blichfeldt, *The minimum value of quadratic forms, and the closest packing of spheres*, Math. Ann. **101** (1929), no. 1, 605–608, DOI 10.1007/BF01454863. MR1512555 ←8

[6] J. W. S. Cassels, *An introduction to the geometry of numbers*, Classics in Mathematics, Springer-Verlag, Berlin, 1997. Corrected reprint of the 1971 edition. MR1434478 ←9

[7] Y. Chen and P. Q. Nguyen, *BKZ 2.0: better lattice security estimates*, Advances in cryptology—ASIACRYPT 2011, Lecture Notes in Comput. Sci., vol. 7073, Springer, Heidelberg, 2011, pp. 1–20. MR2934994 ←22, 23

[8] J. H. Conway and N. J. A. Sloane, *Sphere packings, lattices and groups*, 3rd ed., Grundlehren der mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], vol. 290, Springer-Verlag, New York, 1999. With additional contributions by E. Bannai, R. E. Borcherds, J. Leech, S. P. Norton, A. M. Odlyzko, R. A. Parker, L. Queen and B. B. Venkov. MR1662447 ←9

[9] W. Diffie and M. E. Hellman, *New directions in cryptography*, IEEE Trans. Inform. Theory **IT-22** (1976), no. 6, 644–654, DOI 10.1109/tit.1976.1055638. MR437208 ←28

[10] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, *Lattice signatures and bimodal Gaussians*, Advances in cryptology—CRYPTO 2013. Part I, Lecture Notes in Comput. Sci., vol. 8042, Springer, Heidelberg, 2013, pp. 40–56. MR3126416 ←56

[11] C. Gentry, *Key recovery and message attacks on NTRU-composite*, Advances in cryptology—EUROCRYPT 2001 (Innsbruck), Lecture Notes in Comput. Sci., vol. 2045, Springer, Berlin, 2001, pp. 182–194. MR1895433 ←51

[12] C. Gentry, *Fully homomorphic encryption using ideal lattices*, STOC'09—Proceedings of the 2009 ACM International Symposium on Theory of Computing, ACM, New York, 2009, pp. 169–178. MR2780062 ←42

[13] C. Gentry and M. Szydlo, *Cryptanalysis of the revised NTRU signature scheme*, Advances in cryptology—EUROCRYPT 2002 (Amsterdam), Lecture Notes in Comput. Sci., vol. 2332, Springer, Berlin, 2002, pp. 299–320. MR1975541 ←54

[14] O. Goldreich, S. Goldwasser, and S. Halevi, *Public-key cryptosystems from lattice reduction problems*, Advances in cryptology—CRYPTO '97 (Santa Barbara, CA, 1997), Lecture Notes in Comput. Sci., vol. 1294, Springer, Berlin, 1997, pp. 112–131, DOI 10.1007/BFb0052231. MR1630399 ←41, 42

[15] F. Göpfert, *Securely Instantiating Cryptographic Schemes Based on the Learning with Errors Assumption*, 2016. Thesis (Ph.D.)–Technische Universität Darmstadt. ←24

[16] P. M. Gruber and C. G. Lekkerkerker, *Geometry of numbers*, 2nd ed., North-Holland Mathematical Library, vol. 37, North-Holland Publishing Co., Amsterdam, 1987. MR893813 ←9

[17] J. Hoffstein, B. S Kaliski Jr, D. B. Lieman, M. J. B. Robshaw, and Y. L. Yin, *Secure user identification based on constrained polynomials*, Filed October 1997, Issued June 2000. US Patent 6,076,163. ←65

[18] J. Hoffstein, D. Lieman, and J. H. Silverman, *Polynomial rings and efficient public key authentication*, International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC '89) (Hong Kong, 1989). ←65

[19] J. Hoffstein, J. Pipher, and J. H. Silverman, *NTRU: a ring-based public key cryptosystem*, Algorithmic number theory (Portland, OR, 1998), Lecture Notes in Comput. Sci., vol. 1423, Springer, Berlin, 1998, pp. 267–288, DOI 10.1007/BFb0054868. MR1726077 ←41

[20] J. Hoffstein, J. Pipher, J. M. Schanck, J. H. Silverman, and W. Whyte, *Transcript secure signatures based on modular lattices*, Post-quantum cryptography, Lecture Notes in Comput. Sci., vol. 8772, Springer, Cham, 2014, pp. 142–159. MR3278400 ←54, 56

[21] J. Hoffstein and J. H. Silverman, *Polynomial rings and efficient public key authentication. II*, Cryptography and computational number theory (Singapore, 1999), Progr. Comput. Sci. Appl. Logic, vol. 20, Birkhäuser, Basel, 2001, pp. 269–286. MR1944732 ←65

[22] J. Hoffstein, J. Pipher, J. M. Schanck, J. H. Silverman, and W. Whyte, *Practical signatures from the partial Fourier recovery problem*, Applied cryptography and network security, Lecture Notes in Comput. Sci., vol. 8479, Springer, Cham, 2014, pp. 476–493. MR3219007 ←56, 65

[23] A. K. Lenstra, H. W. Lenstra Jr., and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann. **261** (1982), no. 4, 515–534, DOI 10.1007/BF01457454. MR682664 ←9, 20

[24] V. Lyubashevsky, *Lattice-based identification schemes secure under active attacks*, Public key cryptography—PKC 2008, Lecture Notes in Comput. Sci., vol. 4939, Springer, Berlin, 2008, pp. 162–179. MR2570228 ←55

[25] V. Lyubashevsky, *Fiat-Shamir with aborts: applications to lattice and factoring-based signatures*, Advances in cryptology—ASIACRYPT 2009, Lecture Notes in Comput. Sci., vol. 5912, Springer, Berlin, 2009, pp. 598–616. MR2593089 ←55

[26] P. Q. Nguyen and O. Regev, *Learning a parallelepiped: cryptanalysis of GGH and NTRU signatures*, J. Cryptology **22** (2009), no. 2, 139–160. MR2496387 ←54

[27] P. Q. Nguyen and O. Regev, *Learning a parallelepiped: cryptanalysis of GGH and NTRU signatures*, Advances in cryptology—EUROCRYPT 2006, Lecture Notes in Comput. Sci., vol. 4004, Springer, Berlin, 2006, pp. 271–288. MR2423548 ←54

[28] P. Q Nguyen, *Hermite's constant and lattice algorithms*, The LLL algorithm, 2010, pp. 19–69. MR2722178 ←22

[29] P. Q. Nguyen and B. Vallée (eds.), *The LLL algorithm*, Information Security and Cryptography, Springer-Verlag, Berlin, 2010. Survey and applications. MR2722178 ←

[30] W. M. Schmidt, *Diophantine approximation*, Lecture Notes in Mathematics, vol. 785, Springer, Berlin, 1980. MR568710 ←9

[31] C. P. Schnorr, *Lattice reduction by random sampling and birthday methods*, STACS 2003, Lecture Notes in Comput. Sci., vol. 2607, Springer, Berlin, 2003, pp. 145–156. MR2066588 ←22

[32] C.-P. Schnorr and M. Euchner, *Lattice basis reduction: improved practical algorithms and solving subset sum problems*, Math. Programming **66** (1994), no. 2, Ser. A, 181–199, DOI 10.1007/BF01581144. MR1297061 ←22

[33] C.-P. Schnorr and M. Euchner, *Lattice basis reduction: improved practical algorithms and solving subset sum problems*, Fundamentals of computation theory (Gosen, 1991), Lecture Notes in Comput. Sci., vol. 529, Springer, Berlin, 1991, pp. 68–85. MR1136071 ←22

[34] C. L. Siegel, *Lectures on the geometry of numbers*, Springer-Verlag, Berlin, 1989. Notes by B. Friedman; Rewritten by Komaravolu Chandrasekharan with the assistance of Rudolf Suter; With a preface by Chandrasekharan. MR1020761 ←9

[35] Y. Yu and L. Ducas, *Second order statistical behavior of LLL and BKZ*, Selected areas in cryptography—SAC 2017, Lecture Notes in Comput. Sci., vol. 10719, Springer, Cham, 2018, pp. 3–22. MR3775576 ←22

Department of Mathematics, Box 1917, Brown University, Providence, RI 02912 USA
Email address: joseph_silverman@brown.edu