Public Key Encryption

KG

October 29, 2019

Contents

1 Introduction

In this note, we consider the following problem. Alice wants to send a message to Bob via some channel. Eve has access to the channel and she may eavesdrop on anything sent over the channel. Alice does not want Eve to know the content of her message to Bob.

The obvious solution is for Alice and Bob to first run the Diffie-Hellman protocol to establish a shared secret. Then Alice can use the shared secret to encrypt her message using symmetric cryptography.

For some channels, such as mail, this is very inconvenient, since each message may take a long time to arrive, and even longer before it is read and acted upon.

Of course, Alice and Bob could establish a shared secret and then simply use that secret from then on. But Alice may need to talk to many people, not just Bob. Sharing secrets with all of them and then managing the shared secrets is inconvenient.

Alice wants to be able to send a single encrypted message to Bob or one of her other correspondents. She does not want to manage shared secrets with every correspondent, but she may be willing to manage public information, just as she is already managing names, phone numbers and adresses.

The answer to Alice's problem is public key encryption, and the basic idea is explained and defined in Section 2. Section 3 describes a public key encryption scheme based on the Diffie-Hellman protocol, and Section 4 describes public key encryption schemes based on arithmetic modulo products of large primes. The best general attack on these schemes is to factor integers, and we discuss factoring algorithms in Section 5. In Section 7 we describe several public key cryptosystems essentially based on lattices. For two of the systems, lattices are not required for describing the scheme (relying instead on linear algebra and polynomial arithmetic). Instead, lattices are required for the analysis of the schemes. We discuss algorithms for solving the related lattice problems in Section 8, while Section 6 covers basic material about lattices.

This text is intended for a reader that is familiar with mathematical language, basic number theory, basic algebra (groups, rings, fields and linear algebra) and elementary computer science (algorithms), as well as the Diffie-Hellman protocol. We do not expect the reader to be familiar with lattices, so we include a brief introduction to lattices in Section 6.

This text is informal, in particular with respect to computational complexity. Every informal claim in this text can be made precise, but the technical details are out of scope for this note.

This text uses colour to indicate who is supposed to know what. When discussing cryptography, red denotes secret information that is only known by its owner, Alice or Bob. Green denotes information that Alice and Bob want to protect, typically messages. Blue denotes information that the adversary Eve will see. Information that is assumed to be known by both Alice and Bob (as well as Eve) is not coloured.

We also use colour for theorems about computation, where blue denotes information that an algorithm gets as input and can use directly, while red denotes information that exists, but has to be computed somehow before it can be used directly. Information that is considered fixed (such as the specific group in use, group order, generator, etc.) and that the algorithm may depend on is not coloured.

2 Public Key Encryption

Alice, Bob and a number of other people want to be able to send confidential messages to each other. For various reasons, using the Diffie-Hellman protocol to establish a shared secret every time they want to send messages is not possible or practical. Furthermore, Alice does not want to manage a long-term secret for each correspondent. She is willing to manage public information for each correspondent.

In this situation, what is needed is public key encryption.

D **Definition 1.** A *public key encryption* scheme consists of three algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$:

- The *key generation* algorithm $\mathcal{K}$ takes no input and outputs an *encryption key* ek and a *decryption key* dk. To each encryption key ek there is an associated message set $\mathcal{M}_{ek}$.
- The *encryption* algorithm $\mathcal{E}$ takes as input an encryption key ek and a message $m \in \mathcal{M}_{ek}$ and outputs a ciphertext c.
- The *decryption* algorithm $\mathcal{D}$ takes as input a decryption key dk and a ciphertext c and outputs either a message m or the special symbol $\perp$ indicating decryption failure.

We require that for any key pair (ek, dk) output by $\mathcal{K}$ and any message $m \in \mathcal{M}_{ek}$

$$\mathcal{D}(dk, \mathcal{E}(ek, m)) = m.$$

Just as for symmetric encryption, the users of a system require confidentiality and some sense of integrity. However, since the encryption key is known anyone can create a ciphertext, so the informal notion of integrity does not work for public key encryption. Instead, we have a notion of non-malleability where it should be hard to modify a ciphertext in a predictable way.

Informally: A public key encryption scheme provides *confidentiality* if it is hard to learn anything at all about the decryption of a ciphertext from the ciphertext itself, possibly except the length of the decryption.

Informally: A public key encryption scheme is *non-malleable* if it is hard to create a new ciphertext based on a given ciphertext such that the decryption of the new ciphertext is not $\perp$, but predictably related to the given ciphertext.

3 Schemes Based on Diffie-Hellman

We shall develop a public key encryption scheme based on the Diffie-Hellman protocol. The initial situation is that Alice, Carol and Bob will use the Diffe-Hellman protocol to establish a shared secret, and then send the message encrypted using a symmetric encryption scheme.

Let G be a finite cyclic group of order n and let g be a generator. Let $(G, \mathcal{P}, \mathcal{C}, \mathcal{E}_s, \mathcal{D}_s)$ be a symmetric cryptosystem. Note that G is the key set for the symmetric cryptosystem.

When Alice wants to send a message $m_A \in \mathcal{P}$ to Bob, she uses Diffie-Hellman to establish a shared secret, encrypts her message using the symmetric cryptosystem and sends the ciphertext to Bob. Bob decrypts the ciphertext.

1. Alice chooses a number r_A uniformly at random from the set $\{0, 1, 2, \ldots, n-1\}$. She computes $x_A = g^{r_A}$ and sends x_A to Bob.

2. Bob receives x_A from Alice. He chooses a number b_A uniformly at random from the set $\{0, 1, \ldots, n-1\}$, computes $y_A = g^{b_A}$ and $z_A = x_A^{b_A}$.

3. Alice receives y_A from Bob. Alice computes $z_A = y_A^{r_A}$, encrypts the message as $w \leftarrow \mathcal{E}_s(z_A, m_A)$, and sends w to Bob.

The situation where both Alice and Carol send messages to Bob is illustrated in Figure 1.

But a variation on this topic is possible. Before anything happens, Bob executes part of the Diffie-Hellman protocol. He samples a random number b and computes $y = g^b$. Whenever someone contacts him, he immediately responds with y. When he receives the symmetric ciphertext, he computes the shared secret and decryptes the ciphertext. This variation is illustrated in the middle part of Figure 1.

It is possible to prove that Bob does not lose any security by doing this. The proof is out of scope for this note.

Obviously, repeatedly sending the same value y is wasteful. Bob therefore announces publicly what he is doing and publishes the value y. When Alice and Carol want to send a message to Bob, they already know y. Therefore, they do not have to wait to receive it. Instead, they can immediately compute the shared secret and encrypt the message. This final situation is shown in the bottom of Figure 1.

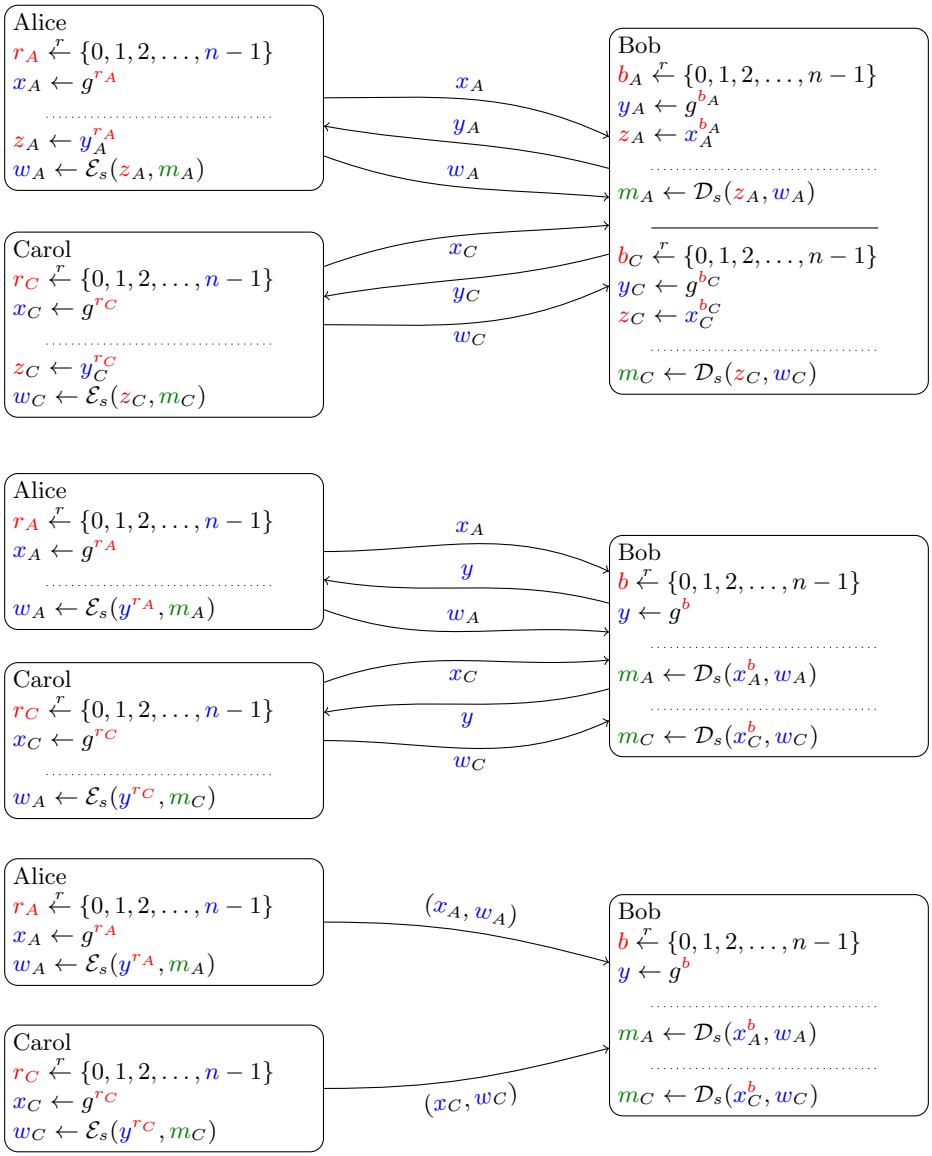

Figure 1: From Diffie-Hellman to public key encryption scheme. (top) Alice and Carol use Diffie-Hellman to establish shared secrets with Bob and send him encrypted messages. (middle) Bob uses a single random number for all the Diffie-Hellman protocol runs. (bottom) Bob publishes his Diffie-Hellman message. Alice and Carol complete the Diffie-Hellman protocol to establish a shared secret and send Bob encrypted messages.

What has happened is that we have turned the Diffie-Hellman protocol combined with a symmetric cryptosystem into a public key encryption scheme.

The public key encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is based on a finite cyclic group G of order n with generator g and a symmetric cryptosystem $(G, \mathcal{P}, \mathcal{C}, \mathcal{E}_s, \mathcal{D}_s)$.

- The *key generation* algorithm $\mathcal{K}$ samples a number b uniformly at random from the set $\{0, 1, 2, \ldots, n-1\}$. It computes $y = g^b$ and outputs $ek = y$ and $dk = b$. The message set associated to ek is $\mathcal{P}$.
- The *encryption* algorithm $\mathcal{E}$ takes as input an encryption key y and a message $m \in \mathcal{P}$. It samples a number r uniformly at random from the set $\{0, 1, 2, \ldots, n-1\}$. Then it computes $x = g^r$ and $z = y^r$, and encrypts the message as $w = \mathcal{E}_s(z, m)$. It outputs the ciphertext $c = (x, w)$.
- The *decryption* algorithm $\mathcal{D}$ takes as input a decryption key b and a ciphertext $c = (x, w)$. It computes $z = x^b$ and decrypts the message as $m = \mathcal{D}_s(z, w)$. If $\mathcal{D}_s$ outputs the special symbol $\perp$ indicating decryption failure, then $\mathcal{D}$ outputs $\perp$, otherwise it outputs m.

E *Exercise* 1. The above is an informal description of a public key encryption scheme. Implement the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$. Show that the triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a public key encryption scheme.

> **Informally:** *If it is hard to break the Diffie-Hellman protocol based on G, and $(G, \mathcal{P}, \mathcal{C}, \mathcal{E}_s, \mathcal{D}_s)$ provides confidentiality and integrity, then the above public key encryption scheme provides confidentiality and non-malleability.*

3.1 ElGamal

The security of the above scheme depends both on the security of the Diffie-Hellman protocol and the security of the symmetric cryptosystem.

In order to illustrate certain attacks, it is convenient to consider a variant of the above public key encryption scheme that uses an extremely simple symmetric cryptosystem, namely a variant of the Shift cipher given by $(G, G, G, \mathcal{E}_s, \mathcal{D}_s)$, where $\mathcal{E}_s(k, m) = mk$ and $\mathcal{D}_s(k, w) = wk^{-1}$.

The resulting scheme is known as the *ElGamal* (or *textbook ElGamal*) public key encryption scheme.

- The *key generation* algorithm is as described above.
- The *encryption* algorithm $\mathcal{E}$ takes as input an encryption key y and a message $m \in G$. It samples a number r uniformly at random from the set $\{0, 1, 2, \ldots, n-1\}$. Then it computes $x = g^r$ and $w = my^r$. It outputs the ciphertext $c = (x, w)$.
- The *decryption* algorithm $\mathcal{D}$ takes as input a decryption key b and a ciphertext $c = (x, w)$. It computes $m = wx^{-b}$.

E *Example* 1. Consider the group $\mathbb{Z}_{23}^+$ with generator $g = 5$.
Bob chooses the random number $b = 13$ and computes $y = 13 \cdot 5 = 19$.
Alice wants to encrypt the message $m = 7$. She knows Bob's encryption key $y = 19$. She chooses the random number $r = 5$ and computes $x = 5 \cdot 5 = 2$ and $w = 7 + 5 \cdot 19 = 10$. The ElGamal ciphertext is $c = (2, 10)$.
Bob computes $10 + (-13) \cdot 2 = 7$.

E *Example* 2. Consider the group $\mathbb{F}_{23}^*$ with generator $g = 5$.
Bob chooses the random number $b = 13$ and computes $y = 5^{13} = 21$.
Alice wants to encrypt the message $m = 7$. She knows Bob's encryption key $y = 21$. She chooses the random number $r = 5$ and computes $x = 5^5 = 20$ and $w = 7 \cdot 21^5 = 6$. The ElGamal ciphertext is $c = (20, 6)$.
Bob computes $6 \cdot 20^{-13} = 7$.

E *Exercise* 2. The above is an informal description of a public key encryption scheme. Implement the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$. Show that the triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a public key encryption scheme.

Information Leakage For certain groups, the ElGamal public key cryptosystem may leak information about the message. This may or may not be problematic.

E *Exercise* 3. Suppose p is a prime such that $p - 1$ is divisible by a large prime and the discrete logarithm problem is hard in $\mathbb{F}_p^*$. Consider ElGamal based on $G = \mathbb{F}_p^*$. Let y be the encryption key, and let $c = (x, w)$ be an encryption of m under y. Show that

$$\left(\frac{m}{p}\right) = \begin{cases} \left(\frac{w}{p}\right) & \text{if } \left(\frac{x}{p}\right) \text{ or } \left(\frac{y}{p}\right) \text{ is 1; and} \\ -\left(\frac{w}{p}\right) & \text{otherwise.} \end{cases}$$

E *Example* 3. Continuing Example 2, we find $\left(\frac{21}{23}\right) = -1$, $\left(\frac{20}{23}\right) = -1$ and $\left(\frac{6}{23}\right) = 1$. This means that $\left(\frac{m}{23}\right)$ should be -1, and indeed, $\left(\frac{7}{23}\right) = -1$.

E *Exercise* 4. Suppose p is a prime such that the discrete logarithm problem is hard in $\mathbb{F}_p^*$. Suppose also that $p - 1$ is divisible by a (small) known prime ℓ, and that $m \in \mathbb{F}_p^*$ has order ℓ. Show that m can be recovered from an encryption of m using essentially a small multiple of $\sqrt{\ell}$ group operations.

Malleability When the attacker only sees the encryption key and one ciphertext, we say that we have a *chosen plaintext attack*. Quite often the attacker will be able to deduce some information about the decryption of other ciphertexts. We typically consider a situation where the adversary wants to learn something about the decryption of one or more ciphertexts, and may ask for the decryption of one or more other ciphertexts. This is known as a *chosen ciphertext attack*.

We begin by showing that ElGamal is *malleable*, in that even if you do not know the decryption of a ciphertext, it is still possible to create new ciphertexts that decrypt to the same or related messages.

E *Exercise* 5. Suppose $c = (x, w)$ is an encryption of an unknown message m. Show how to create an encryption $c' = (x', w')$ of m where $x' \neq x$. Also show how to create an encryption of mm' for any $m' \in G$.

We can now use these properties of ElGamal to attack confidentiality under a chosen ciphertext attack.

E *Exercise* 6. Suppose Alice sends Bob the ciphertext $c = (x, w)$ and Eve learns this ciphertext. Suppose further that Eve is capable of tricking Bob into decrypting one ciphertext that is different from c. Show how Eve can learn the decryption of c.

The above exercises show that ElGamal, and in particular ElGamal over $\mathbb{F}_p^*$, does not provide confidentiality. Neither does ElGamal provide non-malleability.

However, if the symmetric cryptosystem used provides confidentiality and integrity, it can be proven under reasonable assumptions that the public key encryption scheme from Exercise 1 is non-malleable and provides confidentiality.

4 RSA

In this section we shall develop the famous RSA public key encryption scheme. We begin with the Pohlig-Hellman exponentiation cipher, which can be interpreted as a public key encryption scheme, albeit an insecure one. The problem is that if the group order is known, it is easy to recover the decryption key from the encryption key.

The exponentiation cipher can be adapted to a different algebraic structure, where we are able to prove that recovering a decryption key is as hard as factoring certain integers. While this does not prove that the scheme is secure if it is hard to factor those integers, it turns out that factoring seems to be the best way to attack the cryptosystem. We show a number of flaws in this cryptosystem and discuss various ways of improving the system.

4.1 Preliminaries

The Pohlig-Hellman exponentiation cipher is based on a cyclic group G of order N. The key is an integer k relatively prime to N, and the two block cipher maps are

$$(k, x) \mapsto x^k \qquad \text{and} \qquad (k, y) \mapsto y^{k^{-1}},$$

where k^{-1} is any inverse of k modulo N.

Observe now that we can turn the Pohlig-Hellman cipher into a public key encryption scheme as follows. The key generation algorithm chooses an integer e from the integers between 0 and N that are invertible modulo N. It then finds some inverse d of e modulo the group order N. The encryption key ek is e, while the decryption key dk is d.

The encryption algorithm takes as input an encryption key $ek = e$ and a message $m \in G$ and outputs the ciphertext $c = m^e$.

The decryption algorithm takes as input a decryption key $dk = d$ and a ciphertext $c \in G$ out outputs the message $m = c^d$.

E *Exercise* 7. The above is an informal description of a public key encryption scheme. Implement the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$. Show that the triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a public key encryption scheme.

Unfortunately, this public key encryption scheme is trivially insecure. An adversary that sees the encryption key e can easily compute an inverse of e modulo the group order N, which is assumed known. (It may be impossible to compute the exact inverse found by the key generation algorithm, but any inverse can be used to compute the decryption map.)

However, if the group order was unknown, there would be no obvious way to compute a decryption key from the encryption key. The first thing we do is to note that Pohlig-Hellman works equally well in any finite group.

E *Exercise* 8. Let G be a finite group of order N, and let e and d be inverses modulo (a multiple of) N. Show that the maps $x \mapsto x^e$ and $y \mapsto y^d$ are inverse maps.

The key generation algorithm needs to compute inverses modulo the group order, which usually requires that the key generation algorithm knows the group order. This means that we cannot fix a single group. Instead, the key generation algorithm must choose a group in such a way that it knows the group order. Then it must include in the encryption key a description of the group such that the group order is hard to compute from that description.

Before we continue, we observe that for the Pohlig-Hellman cipher, $ed \equiv 1 \pmod{N}$, which means that $ed - 1$ is a multiple of N. Likewise, Exercise 8 says that if we know a multiple of the group order, we can find something that is effectively a decryption key.

4.2 The RSA Cryptosystem

Before we define the famous RSA cryptosystem, we extend the result of Exercise 8 to a specific *ring*.

E *Exercise* 9. Let n be the product of two large primes p and q, and let e and d be inverses modulo (a multiple of) $\operatorname{lcm}(p-1, q-1)$. Show that the maps $x \mapsto x^e$ and $y \mapsto y^d$ on $\mathbb{Z}_n$ are inverses.

Hint: Prove that p divides the difference $x^{ed} - x$ for any integer x. Repeat for q. Then conclude that the product divides the difference.

The *textbook RSA* public key encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ works as follows.

- The *key generation* algorithm $\mathcal{K}$ chooses two large primes p and q. It computes $n = pq$, chooses e and finds d such that $ed \equiv 1 \pmod{\operatorname{lcm}(p-1, q-1)}$. Finally it outputs $ek = (n, e)$ and $dk = (n, d)$. The message set associated to ek is $\{0, 1, 2, \ldots, n-1\}$.

- The *encryption algorithm* $\mathcal{E}$ takes as input an encryption key (n, e) and a message $m \in \{0, 1, \ldots, n-1\}$. It computes $c = m^e \bmod n$ and outputs the ciphertext c.
- The *decryption algorithm* $\mathcal{D}$ takes as input a decryption key (n, d) and a ciphertext c. It computes $m = c^d \bmod n$ and outputs the message m.

Note that the encryption exponent e may be very small, even 3.

E *Exercise* 10. The above is an informal description of a public key encryption scheme. Implement (use some suitable method to choose p, q and e) the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$ are. Show that the triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a public key encryption scheme.

Hint: Use Exercise 9.

E *Example* 4. Bob chooses the two primes $p = 41$ and $q = 53$ and computes $n = 41 \cdot 53 = 2173$. He chooses $e = 3$ and computes an inverse $d = 347$ of 3 modulo $(41-1)(53-1)$. The encryption key is $(2173, 3)$, the decryption key is $(2173, 347)$.

Alice wants to encrypt the message $m = 1234$. She knows Bob's encryption key. She computes $c = 1234^3 \bmod 2173 = 884$.

Bob computes $884^{347} \bmod 2173 = 1234$.

As we saw in Section 4.1, a public key encryption scheme is obviously useless if it is easy to deduce the decryption key from the encryption key. We shall now consider this problem for the RSA cryptosystem.

Let p and q be distinct, large primes, and let $n = pq$. Consider the group $\mathbb{Z}_n^*$, which is isomorphic to $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$. It is clear that if we know p and q, we can find $N = (p-1)(q-1)$. Conversely, if we know n and N, then we can easily recover the factorization.

T **Proposition 1.** *Let n be a product of two distinct primes p and q, and let N be the order of $\mathbb{Z}_n^*$. Then p and q are zeros of the polynomial $f(X) = X^2 + (N - n - 1)X + n$.*

Proof. First note that $N = n - (p + q) + 1$, so

$$p + q = n + 1 - N.$$

Now consider the polynomial $f(X) = (X - p)(X - q) = X^2 - (p + q)X + n$. □

Note that if we know N, then we know the polynomial's coefficients, and if we know the coefficients, we can easily compute the zeros of the polynomial using the usual formula for the zeros of a quadratic polynomial.

If we only know n and a multiple of l of N, we cannot use the above result, but we can still recover the factorization of n.

E *Exercise* 11. Let n be a product of two distinct primes p and q. Let x and y be integers such that

$$x \equiv y \pmod{p} \qquad \text{and} \qquad x \not\equiv y \pmod{q}.$$

Show that $\gcd(x - y, n) = p$.

Lemma 2. *Let n be a product of two distinct large primes p and q, and let k be an odd multiple of $\operatorname{lcm}(p-1, q-1)/2$. Then for at least half of all integers z between 0 and n that are relatively prime to n,*

$$z^k \equiv \pm 1 \pmod{p} \qquad \text{and} \qquad z^k \equiv \mp 1 \pmod{q}. \tag{1}$$

Proof. Suppose first that neither $p-1$ nor $q-1$ divide k. Then $(p-1)/2$ and $(q-1)/2$ both divide k and

$$\frac{k}{(p-1)/2} \qquad \text{and} \qquad \frac{k}{(q-1)/2}$$

are both odd, so the Legendre symbol tells us that the equations

$$x^k \equiv -1 \pmod{p} \qquad \text{and} \qquad y^k \equiv -1 \pmod{q}$$

hold for half of all integers $x \in \{1, 2, \ldots, p-1\}$ and half of all integers $y \in \{1, 2, \ldots, q-1\}$.

Otherwise, suppose without loss of generality that $q-1$ divides k, while $p-1$ does not divide k. Again, the Legendre symbol tells us that

$$x^k \equiv -1 \pmod{p} \qquad \text{and} \qquad y^k \equiv 1 \pmod{q}$$

for half of all integers $x \in \{1, 2, \ldots, p-1\}$ and any integer $y \in \{1, 2, \ldots, q-1\}$.

In either case, the Chinese remainder theorem says that (1) holds for half of all integers z between 0 and n that are relatively prime to n. □

Proposition 3. *Let n be a product of two distinct primes p and q. For any $\kappa > 0$, given a multiple l of N, we can compute p and q with probability $1 - 2^{-\kappa}$ using at most $\lceil 4\kappa \log_2 l \rceil$ arithmetic operations and $\kappa \log_2 l$ gcd computations.*

Proof. Write $l = 2^t s$, $p-1 = 2^{t_p} s_p$ and $q-1 = 2^{t_q} s_q$ with s, s_p and s_q all odd. We may assume that $t_p \geq t_q$. It is then clear that $2^{t_p - 1} s$ is an odd multiple of $\operatorname{lcm}(p-1, q-1)/2$.

Lemma 2 then says that for half of all z between 0 and n that are relatively prime to n,

$$\gcd(z^{2^{t_p - 1} s} + 1, n) = p \text{ or } q.$$

For each value z chosen uniformly at random from $\{x \mid 0 < x < n, \gcd(x, n) = 1\}$ the probability that the above gcd computation does not produce a proper factor of n is $1/2$. The probability that a proper factor of n has not appeared after κ independently sampled z values is $2^{-\kappa}$.

Obviously, we do not know t_p. But for each value of z chosen, we can simply try all possible values for t_p. Since we know that $t_p \leq t < \log_2 l$, this requires at most $\log_2 l$ gcd computations per z value.

We can compute the greatest common divisor by computing $z^{2^i s}$ modulo n before computing the gcd. We can compute this using the standard recursive exponentiation algorithm using $4 \log_2 l$ arithmetic operations. By computing first $z^{2^0 s}$ first, we can compute $z^{2^1 s}, \ldots, z^{2^{t-1} s}$ successively, thereby computing all t values modulo n using just $4 \log_2 l$ arithmetic operations. Doing this for at most κ distinct z values then requires at most $4\kappa \log_2 l$ arithmetic operations. □

E *Exercise* 12. The proof of Proposition 3 essentially describes an algorithm for factoring a product of two large primes p and q given a multiple l of $\operatorname{lcm}(p-1, q-1)$. Implement this algorithm and restate Proposition 3 as a statement about the algorithm's time complexity (in terms of arithmetic operations and gcd computations, ignoring any other form of computation involved).

E *Example* 5. Consider $n = 2173$. Suppose we know that 35360 is a multiple of the order of $\mathbb{Z}_{2173}^*$. We write $35360 = 2^5 \cdot 1105$.

We begin with $z = 2$ and compute $2^{1105} \bmod 2173 = 401$. We then compute $401^2 \bmod 2173 = 2172$ and $2172^2 \bmod 2173 = 1$. This did not give us the factors of 2173.

We try again with $z = 3$ and compute $3^{1105} \bmod 2173 = 454$. We then compute $454^2 \bmod 2173 = 1854$ and $1854^2 \bmod 2173 = 1803$. At this point, we compute

$$\gcd(1803 - 1, 2173) = 53.$$

We have found that $2173 = 53 \cdot 41$.

Proposition 3 says that if anyone knows a multiple of N, then they can factor. It follows that if we believe that it is difficult to factor integers like n, then it is also hard to deduce the group order of $\mathbb{Z}_n^*$ or any multiple of it, and therefore it is hard to deduce the RSA decryption key from the RSA encryption key.

4.3 Attacks

We have seen that as long as the RSA modulus n chosen by the key generation algorithm is hard to factor, the above public key encryption scheme is not obviously useless. In fact, it turns out that it is useful.

The best strategy for recovering a completely unknown m from $c = m^e \bmod n$ — computing *eth roots* — seems to be to factor n. However, as the security definitions in Section 2 make clear, the adversary does not need to recover a completely unknown m to break the system. In this section, we shall consider a few attacks on the public key encryption scheme from Exercise 10 that work essentially without computing eth roots.

Deterministic Encryption One fundamental problem with the public key encryption scheme is that it is not randomized. The ciphertext depends only on the message.

E *Exercise* 13. Show that $\mathcal{D}(dk, c) = m$ if and only if $\mathcal{E}(ek, m) = c$.

E *Exercise* 14. Let S be a fixed and known set of messages, and let c be an encryption of some message from S. Show that we can decide what the decryption of c is using at most $|S|$ encryptions.

Information Leakage Part of the message encrypted will leak, which may or may not be a problem.

E *Exercise* 15. Show that both e and d will be odd. Show that

$$\left(\frac{m}{n}\right) = \left(\frac{c}{n}\right).$$

Malleability Just like ElGamal, RSA is *malleable* and this can be used in a *chosen ciphertext attack*.

E *Exercise* 16. Suppose c is an encryption of an unknown message m. Show how to create an encryption c' of mm' for any m' in the message space.

E *Exercise* 17. Suppose Alice sends Bob the ciphertext c and Eve learns this ciphertext. Suppose further that Eve is capable of tricking Bob into decrypting one ciphertext that is different from c. Show how Eve can learn the decryption of c.

Short Messages Just like we developed a public key encryption scheme based on Diffie-Hellman and a symmetric encryption scheme, we can combine RSA and a symmetric cryptosystem $(\mathcal{K}_s, \mathcal{P}, \mathcal{C}, \mathcal{E}_s, \mathcal{D}_s)$ to form a new public key encryption scheme.

The obvious idea is to consider the keys of the symmetric cryptosystem as integers and do as follows:

- The *key generation* algorithm is exactly the same as the RSA key generation algorithm.
- The *encryption* algorithm takes an encryption key $ek = (n, e)$ and a message $m \in \mathcal{P}$ as input. It chooses a random key k from $\mathcal{K}_s$, computes $x \leftarrow k^e \bmod n$ and $w \leftarrow \mathcal{E}_s(k, m)$. Finally, it outputs $c = (x, w)$.
- The *decryption* algorithm takes a decryption key $dk = (n, d)$ and a ciphertext $c = (x, w)$ as input. It computes $k \leftarrow x^d \bmod n$ and $m \leftarrow \mathcal{D}_s(k, w)$. If $k \notin \mathcal{K}_s$ or decryption of w fails, it outputs $\perp$. Otherwise it outputs m.

One problem is that symmetric keys are typically small compared to the RSA modulus.

E *Exercise* 18. Suppose $n \approx 2^{2000}$, $e = 3$ and $\mathcal{K}_s = \{0, 1, 2, \ldots, 2^{256} - 1\}$. Given an encryption c of an unknown key k, show how you can easily recover k.

Small Integer Roots of Polynomial Equations A first solution to this problem is to add a large, fixed integer such as 2^{1999} to the key before raising it to the eth power, and subtracting it after raising the RSA ciphertext to the dth power. That is, $x \equiv (2^{1999} + k)^e \pmod{n}$. Unfortunately, for small e it is easy to find small integer roots of modular univariate equations such as $(2^{1999} + X)^e - x \equiv 0 \pmod{n}$, and $k < 2^{256}$ is a small integer root of this equation.

A better solution is to add a random multiple of a large, fixed integer. That is, $x \equiv (r2^{256} + k)^e \pmod{n}$. We recover k by computing the remainder when divided by 2^{256} after raising x to the dth power.

While this seems to work, there are some worries. First, even if it is hard to compute eth roots in $\mathbb{Z}_n$, it may not be hard to compute parts of the root, e.g. the key k. Second, it may be possible to modify x in such a way that k does not change, only r. Since the decryption algorithm discards r, it will accept the modified ciphertext as valid, which may be a problem for certain applications.

4.4 Secure Variants

There is a simple, robust solution that uses a *random-looking* function to derive the key from a random number, which is raised to the eth power. Suppose h is a function from $\{0, 1, \ldots, n-1\}$ to $\mathcal{K}_s$. We get the following cryptosystem.

- The *key generation* algorithm is exactly the same as the RSA key generation algorithm.
- The *encryption* algorithm takes an encryption key $ek = (n, e)$ and a message $m \in \mathcal{P}$ as input. It chooses r uniformly at random from $\{0, 1, \ldots, n-1\}$, computes $k \leftarrow h(r)$, $x \leftarrow r^e \bmod n$ and $w \leftarrow \mathcal{E}_s(k, m)$. Finally, it outputs $c = (x, w)$.
- The *decryption* algorithm takes a decryption key $dk = (n, d)$ and a ciphertext $c = (x, w)$ as input. It computes $r \leftarrow x^d \bmod n$, $k \leftarrow h(r)$ and $m \leftarrow \mathcal{D}_s(k, w)$. If decryption of w fails, it outputs $\perp$. Otherwise, it outputs m.

Unless you know all of r, you know very little about the key k, since h is random-looking. Furthermore, any change in x will lead to a change in r, which will lead to an unpredictable change in k because h is random-looking. If k has changed unpredictably, it will be hard to modify w so that it is a valid ciphertext under the modified k, so it will be hard to get the decryption algorithm to say anything but $\perp$.

E *Exercise* 19. The above is an informal description of a public key encryption scheme. Implement (reuse key generation from Exercise 10) the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$. Show that the triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a public key encryption scheme.

E *Exercise* 20. Suppose you want to send the same message m to l recipients with public keys $(n_1, e_1), (n_2, e_2), \ldots, (n_l, e_l)$. One simple approach is to use the same r and compute $x_i \leftarrow r^{e_i} \bmod n_i$, $i = 1, 2, \ldots, l$, $k \leftarrow h(r)$ and $w \leftarrow \mathcal{E}_s(k, m)$. You send (x_i, w) to the ith recipient.

Suppose $e_1 = e_2 = \cdots = e_l \leq l$. Show how you can easily recover r.

Remark. Exercise 20 does not show an attack on the public key encryption scheme from Exercise 19. Instead, it illustrates one way to misuse a cryptosystem and thereby introduce weaknesses. In this case, reusing the randomness r for more than one encryption introduced the weakness. In general, it is implicitly assumed that randomness used by key generation and encryption algorithms is never reused and does not leak out of the algorithm. If those assumptions are violated, weaknesses may result as shown by the exercise.

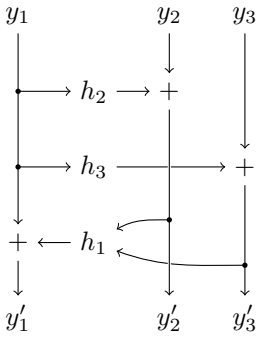

Figure 2: The permutation π can be used for RSA encryption.

> **Informally:** *If it is hard to compute eth roots modulo n for (n, e) as output by the key generation algorithm $\mathcal{K}$, and $(\mathcal{K}_s, \mathcal{P}, \mathcal{C}, \mathcal{E}_s, \mathcal{D}_s)$ provides confidentiality and integrity, then the above public key encryption scheme provides confidentiality and non-malleability.*

In situations with multiple recipients of the same message, other schemes may be more convenient. One solution is based on a Feistel-like permutation. Suppose we have three groups G_1, G_2 and G_3 such that $G_1 \times G_2 \times G_3$ can be considered a subset of $\{0, 1, \ldots, n-1\}$. Suppose also that we have three random-looking functions $h_1 : G_2 \times G_3 \to G_1$, $h_2 : G_1 \to G_2$ and $h_3 : G_1 \to G_3$. Then we can construct two permutations on $G_1 \times G_2 \times G_3$ as

$$\pi_1(y_1, y_2, y_3) = (y_1, y_2 + h_2(y_1), y_3 + h_3(y_1)) \text{ and}$$
$$\pi_2(y'_1, y'_2, y'_3) = (y'_1 + h_1(y'_2, y'_3), y'_2, y'_3).$$

The inverses of these two permutations are obvious. Our cryptosystem will use the composition $\pi = \pi_2 \circ \pi_1$ shown in Figure 2.

Suppose $\mathcal{K}_s \subseteq G_2$. Our cryptosystem works as follows.

- The *key generation* algorithm is exactly the same as the RSA key generation algorithm.
- The *encryption algorithm* takes an encryption key $ek = (n, e)$ and a message $m \in \mathcal{P}$ as input. It chooses r uniformly at random from G_1 and k uniformly at random from $\mathcal{K}_s$, computes $x \leftarrow \pi(r, k, 0)^e \bmod n$ and $w \leftarrow \mathcal{E}_s(k, m)$. Finally, it outputs $c = (x, w)$.
- The *decryption algorithm* takes a decryption key $dk = (n, d)$ and a ciphertext d as input. It computes $(r, k, y_3) \leftarrow \pi^{-1}(x^d \bmod n)$ and $m \leftarrow \mathcal{D}_s(k, w)$. If $k \notin \mathcal{K}_s$ or $y_3 \neq 0$ or the decryption of w failed, it outputs $\perp$. Otherwise it outputs m.

E *Exercise* 21. The above is an informal description of a public key encryption scheme. Implement (choose some sensible hash functions, and reuse key generation from Exercise 10) the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$ are. Show that the triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a public key encryption scheme.

Suppose we know x, that $x^d \bmod n = (y_1', y_2', y_3')$ and that $\pi^{-1}(y_1', y_2', y_3') = (r, k, 0)$.

Since $h_2(r)$ is added to the key to get y_2', it is impossible to recover k without knowing r. But h_1 is used to hide r, so it is impossible to recover all of r without knowing both y_1', y_2' and y_3'.

Furthermore, any change in y_1' will lead to a change in r, which will lead to an unpredictable change in $h_3(r)$. Any change in y_2' or y_3' will lead to an unpredictable change in r. If r has changed, it is unlikely that π^{-1} will result in 0 in the third coordinate, which means that the decryption algorithm will reject the ciphertext.

It is possible to prove that under reasonable assumptions, variants of the above public key cryptosystem provide confidentiality and non-malleability.

5 Factoring Integers

The cryptosystems described in Section 4.4 seem to be secure if it is hard to compute eth roots modulo the RSA modulus. It seems like the best method to compute eth roots modulo the RSA modulus n is to factor n.

> **Informally:** *It is conjectured that computing eth roots modulo a well-chosen RSA modulus is not (much) easier than factoring the RSA modulus.*

Under this conjecture, the study of the security of RSA-based cryptosystems like those in Section 4.4 reduces to the study of how easy it is to factor RSA moduluses.

Throughout this section, unless otherwise stated we shall consider an integer n that is the product of two large primes p and q, with $q < p$. We do this to simplify the exposition, but we note that most of the factoring algorithms we consider will work for other composite numbers, and often work much better.

We shall usually estimate the work required by our algorithms in terms of arithmetic operations. By arithmetic operations, we mean additions, subtractions, multiplications or divisions of integers of about the same size as n. We ignore additions and subtractions of small constants, as well as multiplication and division by 2.

We begin with the simplest possible factoring algorithm, so-called trial division. While it works, it is extremely slow.

T **Proposition 4.** *A factor of a composite integer n can be found using at most $\sqrt{n}$ arithmetic operations.*

Proof. The composite n must have a factor smaller than $\sqrt{n}$. We can divide n by the integers $2, 3, 4, \ldots, \lfloor \sqrt{n} \rfloor$ in order. When the remainder is zero, we have found the smallest prime divisor of n. □

E *Exercise* 22. The proof of Proposition 4 essentially describes an algorithm for finding a factor of a composite integer. Implement this algorithm and restate Proposition 4 as a statement about the algorithm's time complexity (in terms of arithmetic operations, ignoring any other form of computation involved).

Requirement 1. Let q be the smallest divisor of n. Then q arithmetic operations should be an infeasible computation.

Having sufficiently large prime divisors is obviously easy to arrange.

E *Exercise* 23. Suppose n is a product of one or more prime powers, and that the biggest prime divisor is p. Show that the prime factorization of n can be computed using at most $p + \log_2 n$ arithmetic operations.

5.1 Fermat Factoring

If p and q are numbers of roughly the same size, then p and q should be close to $\sqrt{n}$, but their average should be closer. Searching for the average close to $\sqrt{n}$ may make sense.

Let $t = (p+q)/2$ and let $s = (p-q)/2$. Then

$$t^2 - s^2 = (t+s)(t-s) = pq = n.$$

This means that we can decide if some integer τ equals t by computing $\tau^2 - n$ and checking if it is an integer square.

T **Lemma 5.** *Given an integer $k > 1$, we can compute the integer square root of k if it exists using at most $3\log_2 k + 3$ arithmetic operations.*

Proof. We use binary search and construct a sequence of intervals $(l_1, r_1), (l_2, r_2), \ldots$ as follows.

Let $l_1 = 1$ and $r_1 = k$. Let $u_i = \lfloor (l_i + r_i)/2 \rfloor$. If $u_i^2 > k$, set $(l_{i+1}, r_{i+1}) = (l_i, u_i)$. If $u_i^2 < k$, set $(l_{i+1}, r_{i+1}) = (u_i, r_i)$. If $u_i^2 = k$, set $(l_{i+1}, r_{i+1}) = (u_i, u_i)$.

If $u_i^2 = k$ for some i, then $l_j = \sqrt{k} = r_j$ for all $j > i$. Otherwise, note that $1 < \sqrt{k} < k$. And if $l_i < \sqrt{k} < r_i$ and $u_i^2 \neq k$, then $l_{i+1} < \sqrt{k} < r_{i+1}$. It follows that the intervals satisfy $l_i \leq \sqrt{k} \leq r_i$ for all i. Furthermore, if $l_i \neq r_i$, then $l_i < \sqrt{k} < r_i$.

If $r_i - l_i > 1$, then $r_{i+1} - l_{i+1} \leq \lceil (r_i - l_i)/2 \rceil$, or alternatively $r_{i+1} - l_{i+1} \leq (r_i - l_i)/2 + 1/2$. It follows that

$$r_{i+1} - l_{i+1} \leq \frac{1}{2}(r_i - l_i) + \frac{1}{2} \leq \frac{1}{2}\left(\frac{1}{2}(r_{i-1} - l_{i-1}) + \frac{1}{2}\right) + \frac{1}{2}$$

$$\leq 2^{-i}(r_1 - l_1) + \sum_{j=1}^{i} 2^{-j} < 2^{-i}k + 1.$$

It follows $r_{i+1} - l_{i+1} \leq 1$ for $i > \log_2 k$.

We now have a simple algorithm for computing the integer square root, or concluding that it does not exist. It computes pairs (l_i, r_i) until either $r_i = l_i$, in which case r_i is the integer square root of k, or $r_i = l_i + 1$, in which case k is not the square of any integer.

This algorithm will terminate after computing at most $\lceil \log_2 k \rceil$ pairs (after the first). Finally, given a pair (l_i, r_i), computing (l_{i+1}, r_{i+1}) requires 3 arithmetic operations. The claim follows. □

E *Exercise* 24. The proof of Lemma 5 essentially describes an algorithm for computing an integer square root, or proving that no such square root exists. Implement this algorithm and restate Lemma 5 as a statement about the algorithm's time complexity (in terms of arithmetic operations, ignoring any other form of computation involved).

E *Exercise* 25. Compute the square root of 16129 (by hand) using the algorithm from Exercise 24.

Remark. The algorithm implied for computing square roots implied by Lemma 5 is not the most efficient. Variants of Newtons method will typically be much faster. Furthermore, if we only want to know if a given integer is square, we can detect most non-squares very quickly by checking if they are squares modulo small primes.

T **Lemma 6.** *Let $n = pq$ where p, q are large primes. Then $(p+q)/2 - \sqrt{n} \leq |p-q|/2$.*

Proof. The claim holds if $p = q$. We may therefore assume that $p > q$. With $t = (p+q)/2$ and $s = (p-q)/2$ as above, we have that

$$t - s = \sqrt{(t-s)^2} < \sqrt{(t-s)(t+s)} = \sqrt{n},$$

from which the claim follows. □

T **Theorem 7** (Fermat factoring)**.** *Let n be a product of two large primes p, q. Then a factor of n can be found using at most $3|p-q|(2+\log_2 n)/2 + 1$ arithmetic operations.*

Proof. We are looking for an integer τ such that $\tau = (p-q)/2$. We know that $\tau > \sqrt{n}$, so we may consider the distance $\tau - \lceil \sqrt{n} \rceil$. By Lemma 6, this distance is at most $|p-q|/2$. In other words, there is an $i < |p-q|/2$ and integer σ such that

$$(\lceil \sqrt{n} \rceil + i)^2 - n = \sigma^2.$$

We can find the first integer square among the $|p-q|/2$ integers $(\lceil \sqrt{n} \rceil + i)^2 - n$ using at most $3 \log_2 n + 3 + 3$ arithmetic operations per integer, by Lemma 5.

Once we find $t = \lceil \sqrt{n} \rceil + i$ and the square root s, we find a proper factor $t - s$ of n using a single arithmetic operation. The claim follows. □

E *Exercise* 26. The proof of Theorem 7 essentially describes an algorithm for finding a factor of a composite integer. Implement this algorithm and restate Theorem 7 as a statement about the algorithm's time complexity (in terms of arithmetic operations, ignoring any other form of computation involved).

E *Example* 6. Consider $n = 1173$. We get that $\lceil \sqrt{n} \rceil = 35$.
Then $35^2 - 1173 = 52$, which is not a square.
Then $(35+1)^2 - 1173 = 123$, which is not a square.
Then $(35+2)^2 - 1173 = 196 = 14^2$. We get that

$$1173 = (37-14)(37+14) = 23 \cdot 51.$$

E *Exercise* 27. Factor 1683557 using the algorithm from the above proof.

E *Exercise* 28. We can use a variant of the Sieve of Eratosthenes to find primes quickly. The idea is that we sieve over a small integer range to quickly exclude numbers divisible by small primes. The remaining numbers are then much more likely to be prime. This reduces the number of expensive primality tests we must do before we find a prime.

We can adapt this idea to RSA key generation by sieving over a range likely to contain two primes, which will be our p and q. Explain why this is a bad idea.

We arrive at the following requirement.

Requirement 2. Let $n = pq$. Then $|p-q|$ arithmetic operations should be an infeasible computation.

E *Exercise* 29. Suppose we choose two numbers x and y independently and uniformly at random from the range $\{2^{k-1}, 2^{k-1}+1, \ldots, 2^k - 1\}$. What is the expected value of $|x - y|$?

5.2 Pollard's $p - 1$

We saw earlier that if we have a multiple of $\operatorname{lcm}(p-1, q-1)$, then we can factor n. However, it turns out that if we have a multiple of $p-1$ or $q-1$, we can usually also factor n.

T **Proposition 8.** *Let n be the product of two distinct large primes p and q, and let k be a multiple of $p-1$, but not a multiple of $q-1$. Then for at least half of all integers z between 0 and n that are relatively prime to n,*

$$z^k \equiv 1 \pmod{p} \qquad \text{and} \qquad z^k \not\equiv 1 \pmod{q}. \tag{2}$$

Proof. The first equation holds for all z that are relatively prime to n.

Let g be a generator for $\mathbb{F}_q^*$, and let

$$x = \frac{q-1}{\gcd(q-1, k)}.$$

Then g^k has order x, which means that

$$z^k \equiv 1 \pmod{q}$$

for at most $1/x$ of all integers z between 0 and n that are relatively prime to n. The claim follows. □

Note that if we have z and k such that (2) holds, then Exercise 11 allows us to factor n.

The question is, how do we find a multiple of $p-1$? One approach may be to hope that $p-1$ is only divisible by the l smallest primes $\ell_1, \ell_2, \ldots, \ell_l$. Since the largest prime power that could divide the (unknown) value $p-1$ is $\ell_i^{\lfloor \log n / \log \ell_i \rfloor}$, we could construct a multiple using

$$k = \prod_{i=1}^{l} \ell_i^{\lfloor \log n / \log \ell_i \rfloor}.$$

In practice, $k = \ell_l!$ will work just as well.

Example 7. Consider $n = 1007$. We choose $z = 2$, and try the bound $\ell_l = 6$ and $k = 6! = 720$.

We compute $2^{720} \equiv 153 \pmod{1007}$, and then

$$\gcd(153 - 1, 1007) = 19.$$

We find that $1007 = 19 \cdot 53$.

Exercise 30. Using the above approach, factor 1829.

We arrive at the following requirement.

Requirement 3. Suppose $n = pq$. Let k_1 be the largest divisor of $p-1$, and let k_2 be the largest divisor of $q-1$. Then $\min\{k_1, k_2\}$ arithmetic operations should be an infeasible computation.

One simple way to ensure this is to choose p and q as safe primes, that is, such that $(p-1)/2$ and $(q-1)/2$ are also prime.

5.3 Pollard's ρ

We assume that the reader is familiar with Pollard's ρ method for computing discrete logarithms. Our goal this time is to construct three random-looking sequences of integers, one of which will collide and provide us with our factorization.

Let s_1 be an integer between 0 and n. We construct a sequence of integers $s_1, s_2, \ldots$ using the rule

$$s_{i+1} = s_i^2 + 1 \bmod n. \tag{3}$$

Based on this sequence, we define a second sequence $s_{q,1}, s_{q,2}, \ldots$, where $s_{q,i} = s_i \bmod q$. Note that

$$s_{q,i+1} = s_{q,i}^2 + 1 \bmod q. \tag{4}$$

Lemma 9. *Let the sequences $s_1, s_2, \ldots$ and $s_{q,1}, s_{q,2}, \ldots$ be as above. Suppose indexes i, j exist such that $s_{q,i} = s_{q,j}$. Then $\gcd(s_i - s_j, n) > 1$. If $s_i \neq s_j$, then*

$$\gcd(s_i - s_j, n) = q.$$

Proof. If $s_i = s_j$, then $\gcd(s_i - s_j, n) = n$, so suppose $s_i \neq s_j$.

Since

$$s_i \equiv s_{q,i} \equiv s_{q,j} \equiv s_j \pmod{q},$$

we see that q divides the difference $s_i - s_j$. But $s_i \neq s_j$, so we cannot also have that p divides $s_i - s_j$, and the claim follows. □

Exercise 31. In this exercise, we will use our knowledge of the factors of n to better see what is happening. Let $n = 2573 = 31 \cdot 83$ and $s_1 = 2380$.

1. Compute the first 15 terms of the sequences from (3) and (4).
2. Find by inspection the first repetition in the sequence $s_{q,1}, s_{q,2}, \ldots$.
3. Use Lemma 9 and the repetition found to factor $n = 2573$.

Proposition 10. *Let $s_1, s_2, \ldots$ and $s_{q,1}, s_{q,2}, \ldots$ be defined as above. Suppose k is the smallest integer such that $s_{q,k} = s_{q,k'}$ for some $k' < k$. Then distinct indexes i, j can be found such that $\gcd(s_i - s_j, n) > 1$ using at most $9k$ arithmetic operations and k gcd computations.*

Proof. We consider the sequences $t_1, t_2, \ldots$ and $t_{q,1}, t_{q,2}, \ldots$ given by $t_j = s_{2j}$ and $t_{q,j} = s_{q,2j}$. It is clear that for some i, $t_{q,i} = s_{q,i}$, and that this i is at most k.

We can compute successively the pairs $(s_1, t_1), (s_2, t_2), \ldots$ using the rule

$$(s_{i+1}, t_{i+1}) = (s_i^2 + 1 \bmod n, (t_i^2 + 1)^2 + 1 \bmod n).$$

By computing $\gcd(s_i - t_i, n)$ we will notice when this is larger than 1, which it by Lemma 9 will be for some $i \leq k$. Computing each new pair requires 9 arithmetic operations. □

Exercise 32. The proof of Proposition 10 essentially describes an algorithm that eventually computes a greatest common divisor larger than 1. Implement this algorithm and restate Proposition 10 as a statement about the algorithm's time complexity (in terms of arithmetic operations and gcd computations).

Also, suppose the elements $s_1, s_2, \ldots, s_k$ are all distinct. Show that then the greatest common divisor eventually computed is a proper divisor of n.

Example 8. Consider $n = 1007$.

We begin with $s_1 = 2$, $t_1 = 2^2 + 1 = 5$, and compute:

$$\begin{array}{lll} s_2 = 5 & t_2 = 677 & \gcd(677 - 5, 1007) = 1 \\ s_3 = 26 & t_3 = 886 & \gcd(886 - 26, 1007) = 1 \\ s_4 = 677 & t_4 = 886 & \gcd(886 - 677, 1007) = 19 \end{array}$$

We get that $1007 = 19 \cdot 53$.

E *Exercise* 33. Use the algorithm from Exercise 32 to factor (by hand) $n = 2573$.

Let E be the event that the L first elements of $s_{q,1}, s_{q,2}, \ldots$ are all distinct, and let E' be the event that the L first elements of $s_1, s_2, \ldots$ are all distinct. Then we can define two functions

$$\theta(L, n) = \Pr[E] \qquad \text{and} \qquad \gamma(L, n) = 1 - \Pr[E'].$$

We can now prove the following result.

T **Theorem 11.** *Let n be a product of two distinct primes p and q. Then a proper factor of n can be computed using at most $9L$ arithmetic operations and L gcd computations, except with probability $\theta(L, n) + \gamma(L, n)$.*

Proof. Some integer will appear twice among the L first elements of the sequence $s_{q,1}, s_{q,2}, \ldots$ except with probability $\theta(L, n)$.

There will be no repetitions among the L first elements of the sequence $s_1, s_2, \ldots$ except with probability $\gamma(L, n)$.

By Exercise 32 there is then an algorithm that computes a proper factor of n using at most $9L$ arithmetic operations and L gcd computations. The claim follows. □

Now suppose the two sequences are "random-looking" with respect to repetitions. This means that since the elements of the second sequence come from a much smaller set, we expect a repetition in that sequence long before we have a repetition in the larger sequence. It seems plausible that the sequences we have defined above are "random-looking" with respect to repetitions. Therefore, we make the following conjecture.

T **Conjecture 12.** *The function $\theta(L, n)$ is roughly similar to*

$$\exp\left(-\frac{L(L-1)}{2q}\right)$$

and $\gamma(L, n)$ is roughly similar to

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

Based on Conjecture 12 and Theorem 11, we arrive at the following requirement.

Requirement 4. Suppose $n = pq$, with $q < p$. Then $\sqrt{q}$ arithmetic operations should be an infeasible computation.

5.4 Index Calculus

Index calculus for factoring is very similar to index calculus for discrete logarithms. We begin with the observation that knowledge of more than two square roots modulo n of the same number allows us to factor n.

T **Proposition 13.** *Let n be the product of two distinct, large primes p and q. Let z be a square modulo n with $\gcd(z, n) = 1$. Then z has four square roots modulo n.*

Suppose further that x and y are square roots of z modulo n satisfying

$$x \not\equiv \pm y \pmod{n}.$$

Then $\gcd(x - y, n)$ is a proper divisor of n.

Proof. If z is a square modulo n, then x exists such that $x^2 \equiv z \pmod{n}$, which means that x^2 is congruent to z modulo both p and q. It follows that x and $-x$ are square roots of z modulo both p and q, and they are distinct since z is relatively prime to n. The Chinese remainder theorem then says that these can be combined into four square roots of z modulo n.

Since $x^2 \equiv z \equiv y^2 \pmod{n}$, we know that n divides $x^2 - y^2 = (x - y)(x + y)$. But since $x \not\equiv \pm y \pmod{n}$, we know that n does not divide $x - y$ and $x + y$. It follows that $\gcd(x - y, n)$ is a proper divisor of n. □

E *Example* 9. Consider $n = 314791$ and the two numbers 125823 and 17500. We see that

$$125823^2 \equiv 273148 \equiv 17500^2 \pmod{314791} \quad \text{and}$$
$$125823 \not\equiv \pm 17500 \pmod{314791}.$$

A quick computation gives us

$$\gcd(125823 - 17500, 314791) = 727.$$

The next idea is that if we have a sufficient number of relations between random integers and small primes modulo n, then linear algebra will allow us to construct a square root of a product of these random integers.

T **Proposition 14.** *Let $t_1, t_2, \ldots, t_{l+1}, \ell_1, \ell_2, \ldots, \ell_l$ be integers satisfying*

$$t_i \equiv \prod_{j=1}^{l} \ell^{s_{ij}}. \tag{5}$$

Then using at most $(l + 1)^3$ arithmetic operations, we can compute $\alpha_1, \alpha_2, \ldots, \alpha_{l+1} \in \{0, 1\}$ (not all zero) such that

$$\prod_{i=1}^{l+1} t_i^{\alpha_i} \equiv \left(\prod_{j=1}^{l} \ell_j^{\frac{1}{2} \sum_{i=1}^{l+1} \alpha_i s_{ij}} \right)^2 \pmod{n}.$$

Proof. Let $\mathbf{S}$ be the $(l+1) \times l$ matrix (s_{ij}), where each of the relations defines one row. If we consider $\mathbf{S}$ as a matrix modulo 2, it has rank at most l, so there exists a non-zero vector $\boldsymbol{\alpha}$ such that $\boldsymbol{\alpha}\mathbf{S} \equiv \mathbf{0} \pmod 2$.

Given such a vector $\boldsymbol{\alpha}$, we know that the sums $\sum_{i=1}^{l+1} \alpha_i s_{ij}$ are divisible by 2, which means that raising each ℓ to these sums divided by 2 gives us a square root of $\prod_{i=1}^{l+1} t^{\alpha_i}$.

Gaussian elimination will find a vector in the kernel of $\mathbf{S}$ using at most $(l+1)^3$ arithmetic operations. □

E *Example* 10. Consider $n = 314791$. We have six relations:

$$5000 = 2^3 \cdot 5^4 \qquad 118800 = 2^4 \cdot 3^3 \cdot 5^2 \cdot 11$$
$$7425 = 3^3 \cdot 5^2 \cdot 11 \qquad 882 = 2 \cdot 3^2 \cdot 7^2$$
$$25410 = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11^2 \qquad 61250 = 2 \cdot 5^4 \cdot 7^2$$

We get the matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

We could use Gaussian elimination to find vectors in the matrix kernel, but for such a small matrix, it is easier to find them by inspection. We quickly see that $(1, 0, 0, 1, 0, 0)$ is in the matrix kernel, and we get the square

$$5000 \cdot 882 \equiv 2926 \equiv (2^2 \cdot 3 \cdot 5^2 \cdot 7)^2 \pmod{314791}.$$

Another quick look at the matrix tells us that $(0, 1, 1, 0, 0, 0)$ is in the matrix kernel, and we get the square

$$118800 \cdot 7425 \equiv 45618 \equiv (2^2 \cdot 3^3 \cdot 5^2 \cdot 11)^2 \pmod{314791}.$$

Finally, we see that $(1, 0, 0, 0, 0, 1)$ is in the matrix kernel, which says that

$$5000 \cdot 61250 \equiv 273148 \equiv (2^2 \cdot 5^4 \cdot 7)^2 \pmod{314791}.$$

We have found three squares.

Finally, if we already know squares of the random integers from our relations, we can easily find a square of the product. This second square will be independent of the one we found above. Which means that we will be able to factor n half the time.

T **Theorem 15.** *Suppose n is a product of two distinct large primes, and that a fraction σ of the squares modulo n are divisible only by the small primes $\ell_1, \ell_2, \ldots, \ell_l$.*

Then we can find a proper factor of n with probability $1/2$ using an expected

$$\sigma^{-1}(l+1)(l+\log_2 n+2)+(l+1)^3+2l^2+2(l+1)\log_2 n+2l$$

arithmetic operations and one gcd computation.

Proof. Our goal is to construct two independent square roots modulo n of the same number.

We begin by choosing random numbers r between 0 and n for each number checking if $t = r^2 \bmod n$ factors as a product of the small primes. In this way, we eventually find $l+1$ relations of the form (5).

By Proposition 14, we can find $\alpha_1, \alpha_2, \ldots, \alpha_{l+1} \in \{0,1\}$ such that

$$\left(\prod_{i=1}^{l+1} r_i\right)^2 \equiv \left(\prod_{j=1}^{l} \ell_j^{\frac{1}{2}\sum_{i=1}^{l+1}\alpha_i s_{ij}}\right)^2 \pmod{n}.$$

In other words, we have two square roots modulo n of the same number. Since the coefficients s_{ij} depend only on the square modulo n of the random numbers $r_1, \ldots, r_{l+1}$, the two square roots are also independent.

By Proposition 13 we can then factor n with probability $1/2$ using one gcd computation.

For each random number r, squaring modulo n requires 2 arithmetic operations. Checking if the square factors as a product of $\ell_1, \ldots, \ell_l$ requires at most $\lfloor l + \log_2 n \rfloor$ arithmetic operations.

We expect to find $l+1$ relations after trying $\sigma^{-1}(l+1)$ random numbers, which means that we need at most

$$\sigma^{-1}(l+1)(l+\log_2 n+2)$$

arithmetic operations to generate the relations. We then need at most $(l+1)^3$ arithmetic operations to find $\alpha_1, \ldots, \alpha_{l+1}$. (Strictly speaking, we work with a binary matrix, so these arithmetic operations are much faster.)

Finally, we need at most $2l^2$ arithmetic operations to compute the sums $\sum_{i=1}^{l+1}\alpha_i s_{ij}$, then at most $2(l+1)\log_2 n$ arithmetic operations to compute the small primes raise to these sums. Finally we require at most $2l$ multiplications to compute the two independent square roots. □

Example 11. We choose a lot of random numbers, eventually finding these relations:

$$\begin{aligned}
237625^2 &\equiv 5000 \equiv 2^3 \cdot 5^4 \pmod{314791} \\
90962^2 &\equiv 118800 \equiv 2^4 \cdot 3^3 \cdot 5^2 \cdot 11 \pmod{314791} \\
180136^2 &\equiv 7425 \equiv 3^3 \cdot 5^2 \cdot 11 \pmod{314791} \\
57593^2 &\equiv 882 \equiv 2 \cdot 3^2 \cdot 7^2 \pmod{314791} \\
228218^2 &\equiv 25410 \equiv 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11^2 \pmod{314791} \\
211193^2 &\equiv 61250 \equiv 2 \cdot 5^4 \cdot 7^2 \pmod{314791}
\end{aligned}$$

As we saw in Example 10, we can use these relations to attempt to find two non-trivial square roots of the same number. And as we see in Example 9, once we have these non-trivial square roots, an easy gcd computation gives us a factor of 314791.

Note that if we do all these calculations and fail to factor n, then we do not have to all the work over again. In practice, discarding a few of our old relations and replacing them by a few new ones will give us another go. In this way, we will quickly factor n without doing significantly more work.

We also expect l to be much larger than $\log_2 n$, so we expect to be able to factor n using approximately

$$\sigma^{-1} l^2 + l^3$$

arithmetic operations. It is reasonable to assume that random squares modulo n are as likely to be products of small primes as random integers. As we have seen, after a suitable choice for l, this means that we can factor using approximately

$$\exp\left(\sqrt{8}\sqrt{\log n \log\log n}\right)$$

arithmetic operations.

We have arrived at the following requirement.

Requirement 5. $\exp(\sqrt{8}\sqrt{\log n \log\log n})$ arithmetic operations should be an infeasible computation.

Today, we have much better algorithms for factoring. While we shall not study these algorithms, we note that the above requirement is not the final requirement.

6 Lattices

Many different mathematical concepts share the name *lattice*, but the one we are interested in is essentially integer linear combinations of linearly independent vectors in a real vector space. This notion of lattice arises naturally in many areas of mathematics, and also in many applications, leading to a rich and varied theory that was well-developed long before its use in cryptography.

Lattices have many uses in cryptography, and have for instance been used to attack many cryptographic constructions. There have been many attempts to construct cryptographic systems based on lattices, and some of these have failed. Recently, the theory of lattice-based cryptography has grown significantly, especially related to so-called *homormophic* cryptosystems.

However, in this note, we are not interested in using lattices to attack cryptosystems or these recent constructive developments, but rather the fact that there does not seem to be any fast quantum algorithms for solving difficult lattice-related problems. This makes lattice-based cryptography into a candidate for *quantum-safe* cryptography.

6.1 Lattice basics

There are several common, equivalent ways to define lattices. We have chosen a variant that is convenient for our purposes.

D **Definition 2.** A *lattice* is a subgroup of $\mathbb{R}^n$ of the form

$$\Lambda = \left\{ \sum_{i=1}^{r} a_i \mathbf{b}_i \;\middle|\; a_1, \ldots, a_r \in \mathbb{Z} \right\},$$

where $\mathbf{b}_1, \ldots, \mathbf{b}_r$ are linearly independent vectors in $\mathbb{R}^n$.

A *basis* for a lattice Λ is any set of linearly independent vectors $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{r'}$ such that $\Lambda = \{\sum_i a_i \mathbf{c}_i \mid a_i \in \mathbb{Z}\}$.

From a basis $\mathbf{b}_1, \ldots, \mathbf{b}_r$ we can construct a $r \times n$ matrix $\mathbf{B}$ by letting the ith row be $\mathbf{b}_i$, $i = 1, 2, \ldots, r$. Then

$$\Lambda = \{\mathbf{aB} \mid \mathbf{a} \in \mathbb{Z}^n\}.$$

The matrix $\mathbf{B}$ is called a basis matrix or just a basis. Since the rows of $\mathbf{B}$ are linearly independent, the rank of the matrix is equal to the number of rows.

E *Example* 12. The four vectors $(1, 0, 1, 0)$, $(1, 1, 1, 0)$, $(1, -1, -1, 1)$ and $(0, 1, -1, -1)$ are linearly independent and generate a lattice. Also, the matrix

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & -1 & -1 & 1 \\ 0 & 1 & -1 & -1 \end{pmatrix}$$

is a basis matrix for the same lattice.

E *Exercise* 34. Let Λ be the lattice generated by the basis in Example 12. Decide which of the vectors $(6, 3, -4, -1)$, $(3, 6, 9, 12)$, $(1, 2, 3, 5)$ and $(1, 2, 1, 2)$ are in the lattice.

T **Proposition 16.** *Let Λ be a lattice, and let $\mathbf{B}$ and $\mathbf{C}$ be two bases for Λ. Then $\mathbf{B}$ and $\mathbf{C}$ have the same rank r, and there exists an $r \times r$ invertible integer matrix $\mathbf{U}$ such that $\mathbf{UB} = \mathbf{C}$ and $\mathbf{U}^{-1}$ is an integer matrix.*

Proof. Every row of $\mathbf{B}$ is in Λ which is a subset of the row space of $\mathbf{C}$, so the row space of $\mathbf{B}$ is a subspace of the row space of $\mathbf{C}$. The converse also holds, so the matrices have the same row space and therefore also the same rank.

For every row $\mathbf{b}_i$ in $\mathbf{B}$, there exists an integer vector $\mathbf{a}_i \in \mathbb{Z}^r$ such that $\mathbf{b}_i = \mathbf{a}_i\mathbf{C}$. Combining these expression, we get an $r \times r$ integer matrix $\mathbf{U}$ such that $\mathbf{B} = \mathbf{UC}$. In the same way, we get an $r \times r$ integer matrix $\mathbf{V}$ such that $\mathbf{C} = \mathbf{VB}$, and $\mathbf{B} = \mathbf{UVB}$.

Since $\mathbf{B}$ and $\mathbf{C}$ have maximal rank (and therefore have right-inverses) $\mathbf{U}$ and $\mathbf{V}$ must be inverses. □

E *Exercise* 35. The matrix

$$\mathbf{C} = \begin{pmatrix} -1 & -3 & -4 & 0 \\ 2 & -6 & -3 & 4 \\ 9 & -1 & 9 & 9 \\ 6 & 7 & 12 & 3 \end{pmatrix}$$

is another basis matrix for the lattice Λ from Example 12. Find the integer matrices taking $\mathbf{B}$ to $\mathbf{C}$ and back again, and check that they are inverse integer matrices.

D **Definition 3.** The *rank* of a lattice is the number of vectors in a basis. If Λ in $\mathbb{R}^n$ has rank n, we say that Λ is a *full rank* lattice.

6.2 Hermite Normal Form

We know that matrices in triangular form are particularily useful when deciding if a given system of equations has solutions or not. This is also true for lattices, and a basis matrix that is in triangular form is convenient. We consider only full rank lattices, which simplifies the definition and the arguments slightly.

D **Definition 4.** Let $\mathbf{B} = (b_{ij})$ be an $n \times n$ integer matrix of maximal rank. We say that $\mathbf{B}$ is on *Hermite normal form* if $\mathbf{B}$ is an upper-triangular integer matrix such that $b_{ii} > 0$ and $|b_{ji}| < b_{ii}$ for $1 \leq i \leq n$ and $1 \leq j < i \leq n$.

Any lattice has a basis on Hermite normal form. Before we state the general theorem, we describe the basic step that we shall be using to construct the Hermite normal form.

T **Lemma 17.** *Let* $\mathbf{b}_1 = (b_{11}, b_{12}, \dots), \mathbf{b}_2 = (b_{21}, b_{22}, \dots) \in \mathbb{Z}^n$ *be vectors such that* $b_{11} \neq 0$. *Let* $d = \gcd(b_{11}, b_{21})$ *and let* $s, t \in \mathbb{Z}$ *be such that* $d = sb_{11} + tb_{21}$. *Let*

$$\mathbf{c}_1 = (c_{11}, c_{12}, \dots) = s\mathbf{b}_1 + t\mathbf{b}_2 \qquad \mathbf{c}_2 = (c_{21}, c_{22}, \dots) = (b_{21}/d)\mathbf{b}_1 - (b_{11}/d)\mathbf{b}_2.$$

Then $c_{11} = d$ *and* $c_{21} = 0$, *and the corresponding integer matrix*

$$\mathbf{U} = \begin{pmatrix} s & t \\ b_{21}/d & -b_{11}/d \end{pmatrix}$$

is invertible with $\det(\mathbf{U}) = -1$.

Proof. The lemma follows from direct computation. □

Theorem 18. *Let* $\mathbf{B}$ *be an* $n \times n$ *integer matrix of maximal rank. Then there exists an integer matrix* $\mathbf{U}$ *with* $\det(\mathbf{U}) = \pm 1$ *such that* $\mathbf{UB}$ *is on Hermite normal form. Furthermore, the matrix* $\mathbf{UB}$ *is unique.*

Proof. The first claim is trivially true for a 1×1 matrix.

Assume that the first claim is true for all maximal rank $(n-1) \times (n-1)$ integer matrices.

Now consider an $n \times n$ matrix $\mathbf{B}$ of maximal rank. Since the matrix is of maximal rank, there must be a row $(b_{i1}, b_{i2}, \ldots, b_{in})$ in the matrix with $b_{i1} \neq 0$. We can swap the first row and row i with a permutation matrix $\mathbf{P}$, so that our new basis matrix looks like

$$\mathbf{B}' = \left(\begin{array}{c|c} b_{i1} & \cdots \\ \hline \vdots & \mathbf{C} \end{array}\right) = \mathbf{PB}.$$

Next, we can use Lemma 17 repeatedly to get a matrix where the first row has a non-zero first column and every other row has a zero in the first column, so that our basis matrix looks like

$$\mathbf{B}'' = \left(\begin{array}{c|c} c_{11} & \cdots \\ \hline 0 & \mathbf{C}' \end{array}\right), \qquad c_{11} \neq 0.$$

Note that the matrices we get from Lemma 17 all have determinant -1, so there exists an integer matrix $\mathbf{U}'$ such that $\mathbf{B}'' = \mathbf{UB}'$ and $\det(\mathbf{U}') = \pm 1$.

Since the submatrix $\mathbf{C}'$ is of rank $n-1$, then by assumption there is an integer matrix $\mathbf{U}''$ with $\det(\mathbf{U}'') = \pm 1$ such that $\mathbf{U}''\mathbf{C}'$ is on Hermite normal form. Then

$$\mathbf{B}''' = \left(\begin{array}{c|ccc} c_{11} & c_{12} & \cdots & c_{1n} \\ \hline 0 & \multicolumn{3}{c}{\mathbf{U}''\mathbf{C}'} \end{array}\right) = \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & \mathbf{U}'' \end{array}\right) \mathbf{B}''.$$

Finally, we note that it is easy to make sure that the magnitude of the entries $(c_{12}, \ldots, c_{1n})$ in the top row are smaller than the entries on the diagonal by subtracting first an appropriate multiple of the second row, then the third row, and so on. Now the matrix is on Hermite normal form. By induction, the first claim follows.

Finally, we need to consider uniqueness. Suppose we have two maximal rank integer matrices $\mathbf{C}_1$ and $\mathbf{C}_2$ on Hermite normal form such that $\mathbf{C}_i = \mathbf{U}_i\mathbf{B}$ for some integer matrices $\mathbf{U}_i$, $i = 1, 2$, with $\det(\mathbf{U}_i) = \pm 1$.

The matrices have the same row space. Since they are both upper-triangular, the ith row in $\mathbf{C}_1$ must be an integer linear combination of the $i, i+1, \ldots, n$ rows in $\mathbf{C}_2$. It follows that the diagonals of $\mathbf{C}_1$ and $\mathbf{C}_2$ must be identical.

Next, if the ith rows first differ in the jth column, $j > i$, then their difference must lie on the row space of rows $j, j+1, \ldots, n$. Which means that the difference between the jth columns must be a non-zero multiple of the jth entry on the diagonal. But then the absolute value of the jth column value of at least one of the ith rows must be larger than the value on the diagonal, which contradicts the hypothesis that the matrices were on Hermite normal form. □

Remark. The proof of the first claim in the theorem essentially describes an algorithm for computing the Hermite normal form. It is easy to see that the number of arithmetic operations (counting a gcd computation as one arithmetic operation) required to compute this normal form is not too bad (a multiple of n^3). However, counting only arithmetic operations is misleading, since the numbers involved in these arithmetic operations can grow very large, even when the numbers in the initial lattice basis are small. However, there are more efficient algorithms for computing the Hermite normal form.

6.3 Gram-Schmidt

The Gram-Schmidt algorithm is a general inner product space algorithm, which should be well-known from linear algebra. We will need it later on, so it makes sense to repeat its properties here. In the following, $\langle \cdot, \cdot \rangle$ denotes the usual inner product on $\mathbb{R}^n$. The Gram-Schmidt algorithm takes a vector space basis $\mathbf{b}_1, \ldots, \mathbf{b}_r$ as input and constructs an orthogonal basis $\mathbf{b}_1^*, \ldots, \mathbf{b}_r^*$ recursively, beginning with $\mathbf{b}_1^* = \mathbf{b}_1$ and then computing

$$\mu_{ij} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle}, \quad 1 < i \leq r, 1 \leq j < i, \qquad \text{and} \qquad \mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{ij} \mathbf{b}_j^*. \tag{6}$$

If we rearrange the equation defining $\mathbf{b}_i^*$ as $\mathbf{b}_i = \mathbf{b}_i^* + \sum_{j=1}^{i-1} \mu_{ij} \mathbf{b}_j^*$, we get a lower-triangular matrix and the matrix equation

$$\mathbf{B} = \begin{pmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_r \end{pmatrix} = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ \mu_{ij} & & 1 \end{pmatrix} \begin{pmatrix} \mathbf{b}_1^* \\ \vdots \\ \mathbf{b}_r^* \end{pmatrix}. \tag{7}$$

E *Exercise* 36. Let $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_r \in \mathbb{R}^n$ be a lattice basis. Show that we can compute the Gram-Schmidt basis $\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_r^*$ and coefficients μ_{ij}, $1 \leq j < i \leq r$ using $2nr^2 - 1$ arithmetic operations.

E *Exercise* 37. The answer to Exercise 36 implies the existence of an algorithm to compute a Gram-Schmidt basis. Implement this algorithm.

E *Exercise* 38. Let Λ be the lattice from Example 12, and let $\mathbf{B}$ be the basis matrix from the example. Compute the Gram-Schmidt basis corresponding to this basis.

E *Exercise* 39. Let $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_r$ be a basis for a lattice Λ, let $\mathbf{B}$ be the corresponding matrix, and let $\mathbf{b}_1^*, \ldots, \mathbf{b}_r^*$ be the corresponding Gram-Schmidt orthogonal basis.

1. Show that $\mathbf{b}_1^*, \ldots, \mathbf{b}_r^*$ can be extended to an orthogonal basis $\mathbf{b}_1^*, \ldots, \mathbf{b}_r^*, \mathbf{c}_1^*, \ldots, \mathbf{c}_{n-r}^*$ for $\mathbb{R}^n$.

2. Let $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_r$ be an orthonormal basis for $\mathbb{R}^r$. Let $\mathbf{P}$ be the linear map from $\mathbb{R}^n$ to $\mathbb{R}^r$ that takes $\mathbf{b}_i^*$ to $\|\mathbf{b}_i^*\| \mathbf{e}_i$, $1 \leq i \leq r$, and $\mathbf{c}_i^*$ to $\mathbf{0}$, $1 \leq i \leq n - r$. Show

that for any vector $\mathbf{v} \in \text{span}\{\mathbf{b}_1^*, \ldots, \mathbf{b}_r^*\}$, $\|\mathbf{v}\| = \|\mathbf{v}\mathbf{P}\|$.

3. Show that the image $\Lambda\mathbf{P}$ of Λ under $\mathbf{P}$ is a full-rank lattice, and $\mathbf{b}_1\mathbf{P}, \mathbf{b}_2\mathbf{P}, \ldots, \mathbf{b}_r\mathbf{P}$ is a basis for $\Lambda\mathbf{P}$.

6.4 The Fundamental Domain

Let $\mathbf{b}_1, \ldots, \mathbf{b}_r$ be a basis for a lattice Λ, $\mathbf{B}$ in matrix form. The *fundamental domain* of the lattice is the parallelepiped

$$\left\{ \sum_{i=1}^{r} a_i \mathbf{b}_i \mid 0 \leq a_i < 1 \right\}.$$

When the lattice has full rank, it can be shown that the volume of the fundamental domain is $|\det(\mathbf{B})|$.

Since any two bases are related by an invertible integer matrix, which has determinant ± 1, it is clear that the volume of the fundamental domain is independent of the choice of basis. We call this volume the *determinant* or the *volume* of the lattice, denoted by $\det(\Lambda)$, or sometimes by $\det(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_r)$.

Since a determinant is unchanged by elementary row operations, if $\mathbf{b}_1, \ldots, \mathbf{b}_n$ is a basis for a full-rank lattice Λ, then

$$\det(\Lambda) = |\det(\mathbf{B})| = \prod_i \|\mathbf{b}_i^*\| \leq \prod \|\mathbf{b}_i\|.$$

The value

$$\frac{\prod \|\mathbf{b}_i\|}{\det(\Lambda)}$$

is called the *orthogonality defect* of the basis.

E *Exercise* 40. We continue Exercise 39. Show that $\det(\Lambda\mathbf{P}) = \det(\mathbf{B}\mathbf{P}) = \prod_{i=1}^{r} \|\mathbf{b}_i^*\|$.

When $r < n$, we define the volume of the fundamental domain to be

$$\det(\Lambda) = \sqrt{|\det(\mathbf{B}\mathbf{B}^T)|} = \prod_{i=1}^{r} \|\mathbf{b}_i^*\|.$$

Another interesting property of the fundamental domain of a full-rank lattice is that any point in space can be expressed uniquely as the sum of a lattice point and a point in the fundamental domain.

E *Exercise* 41. Let Λ be a lattice with basis $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ and let $\mathbf{z} \in \mathbb{R}^n$. Show that unique integers $a_1, a_2, \ldots, a_n$ and real numbers $\alpha_1, \alpha_2, \ldots, \alpha_n \in [0, 1)$ exist such that

$$\mathbf{z} = \sum_i a_i \mathbf{b}_i + \sum_i \alpha_i \mathbf{b}_i.$$

Note that the fundamental domain depends on the basis for the lattice. We shall use the notation $\mathbf{z} \bmod \mathbf{B}$ to denote the unique point in the fundamental domain. Of course, if two vectors $\mathbf{z}_1, \mathbf{z}_2$ satisfy $\mathbf{z}_1 \bmod \mathbf{B} = \mathbf{z}_2 \bmod \mathbf{B}$, then $\mathbf{z}_2 - \mathbf{z}_1 \in \Lambda$. We shall write both $\mathbf{z}_1 \equiv \mathbf{z}_2 \pmod{\mathbf{B}}$ and $\mathbf{z}_1 \equiv \mathbf{z}_2 \pmod{\Lambda}$.

For a full-rank lattice, the vector $\boldsymbol{\alpha}$ in the exercise is often denoted by $(\mathbf{z}\mathbf{B}^{-1}) \bmod 1$, but we shall not use this notation. Instead, we shall write $(\mathbf{z} \bmod \mathbf{B})\mathbf{B}^{-1}$.

6.5 Dual lattice

Corresponding to dual vector spaces, we shall define the dual lattice as the set of linear integer-valued functions (functionals) on the lattice, and then identify this set of functions with a particular lattice.

D **Definition 5.** Let $\Lambda \subseteq \mathbb{R}^n$ be a lattice. The *dual lattice* is $\Lambda^* = \{f : \Lambda \to \mathbb{Z} \mid f \text{ linear}\}$.

We would like to study the structure of Λ^*. It is clearly closed under addition and multiplication by integers. Recall that for any functional f on $\mathbb{R}^n$, there is a vector $\mathbf{w}_f$ such that for any $\mathbf{v} \in \mathbb{R}^n$ we have that $f(\mathbf{v}) = \mathbf{v}\mathbf{w}_f^T$. If f is a functional on a subspace, we can we choose $\mathbf{w}_f$ to be in the subspace. Since any $f \in \Lambda^*$ is also a functional on $\mathbb{R}^n$ (or $\operatorname{span}\Lambda$ if the lattice is not a full-rank lattice), we can associate the set

$$S_\Lambda = \{\mathbf{w}_f \in \operatorname{span}\Lambda \mid f \in \Lambda^*\}$$

with Λ^*.

E *Exercise* 42. Show that the set S_Λ is closed under addition and multiplication by integers, and that the natural map $f \mapsto \mathbf{w}_f$ is a bijection and respects addition and multiplication by integers.

T **Proposition 19.** *Let* $\mathbf{B}$ *be a basis matrix for* Λ. *The set* S_Λ *is a lattice with basis matrix* $(\mathbf{B}\mathbf{B}^T)^{-T}\mathbf{B}$.

Proof. Suppose $\mathbf{x}, \mathbf{y}$ be integer tuples and $\mathbf{w} = \mathbf{x}(\mathbf{B}\mathbf{B}^T)^{-T}\mathbf{B}$ and $\mathbf{v} = \mathbf{y}\mathbf{B}$. Then

$$\mathbf{v}\mathbf{w}^T = \mathbf{y}\mathbf{B}\mathbf{B}^T(\mathbf{B}\mathbf{B}^T)^{-1}\mathbf{x}^T = \mathbf{y}\mathbf{x}^T \in \mathbb{Z}$$

so the lattice generated by $(\mathbf{B}\mathbf{B}^T)^{-T}\mathbf{B}$ is a subset of S_Λ.

Now suppose $\mathbf{w}_f \in S_\Lambda$. Then $\mathbf{w}_f = \mathbf{x}\mathbf{B}$ for some $\mathbf{x} \in \mathbb{R}^r$. Observe also that since the rows of $\mathbf{B}$ are in Λ, we must have that $\mathbf{B}\mathbf{w}_f^T$ is an integer tuple. Then

$$\mathbf{w}_f = \mathbf{x}\mathbf{B} = \mathbf{x}(\mathbf{B}\mathbf{B}^T)^T(\mathbf{B}\mathbf{B}^T)^{-T}\mathbf{B} = \mathbf{x}\mathbf{B}\mathbf{B}^T(\mathbf{B}\mathbf{B}^T)^{-T}\mathbf{B} = \mathbf{w}_f\mathbf{B}^T(\mathbf{B}\mathbf{B}^T)^{-T}\mathbf{B}$$

which proves that $\mathbf{w}_f$ lies in the lattice generated by $(\mathbf{B}\mathbf{B}^T)^{-T}\mathbf{B}$. □

We identify Λ^* with the set S_Λ, so that Λ^* is obviously a lattice.

E *Exercise* 43. Show that the dual lattice of Λ^* is Λ.

T **Proposition 20.** *Let Λ be a lattice and let Λ^* be its dual. Let $\mathbf{B}$ be a basis for Λ and let $\mathbf{C}$ be a basis for Λ^*. Then $\mathbf{BC}^T$ is an integer matrix with integral inverse.*

Proof. First, note that $(\mathbf{BB}^T)^{-T}\mathbf{B}$ is a basis for Λ^*, so there is an integer matrix with integral inverse $\mathbf{U}$ such that $\mathbf{C} = \mathbf{U}(\mathbf{BB}^T)^{-T}\mathbf{B}$. But then

$$\mathbf{BC}^T = \mathbf{BB}^T(\mathbf{BB}^T)^{-1}\mathbf{U}^T = \mathbf{U}^T,$$

which proves the claim. □

6.6 p-ary lattices

Several of the lattices we shall encounter will originate with problems over prime finite fields. In these cases, it makes sense to study certain special lattices.

D **Definition 6.** Let $p\mathbb{Z}^n = \{p\mathbf{x} \mid \mathbf{x} \in \mathbb{Z}^n\}$. A lattice is a *$p$-ary lattice* if $p\mathbb{Z}^n \subseteq \Lambda \subseteq \mathbb{Z}^n$.

Let $\mathbf{M}$ be any $n \times r$ integer matrix. Then let

$$\Lambda_p(\mathbf{M}) = \{\mathbf{x} \in \mathbb{Z}^n \mid \exists \mathbf{a} \in \mathbb{Z}^r : \mathbf{aM} \equiv \mathbf{x} \pmod p\} \text{ and}$$
$$\Lambda_p^\perp(\mathbf{M}) = \{\mathbf{x} \in \mathbb{Z}^n \mid \mathbf{Mx}^T \equiv \mathbf{0} \pmod p\}.$$

Remark. We note that $\mathbf{M}$ can be considered as a generating matrix for a block code over $\mathbb{F}_p$, in which case $\mathbf{M}^T$ would be a parity check matrix for a dual code.

T **Proposition 21.** *Let Λ be a lattice. The following three conditions are equivalent:*

(i) Λ is p-ary;

(ii) there exists a matrix $\mathbf{M}$ such that $\Lambda = \Lambda_p(\mathbf{M})$; and

(iii) there exists a matrix $\mathbf{M}'$ such that $\Lambda = \Lambda_p^\perp(\mathbf{M}')$.

Before we prove the proposition, we shall need a small result about the existence of certain matrices related to subspaces of $\mathbb{F}_p^n$. (Again, this is related to linear codes and their generator and parity check matrices.)

E *Exercise* 44. Let V be any non-trivial proper subspace of $\mathbb{F}_p^n$ of dimension r. Show that there exists matrices $\mathbf{M} \in \mathbb{F}_p^{r\times n}$ and $\mathbf{M}' \in \mathbb{F}_p^{n\times(n-r)}$ such that V is the row space of $\mathbf{M}$ and the left kernel of $(\mathbf{M}')^T$.

Proof of Proposition 21. First observe that $\Lambda_p(\mathbf{M})$ and $\Lambda_p^\perp(\mathbf{M})$ are obviously p-ary lattices, so (i) is implied by both (ii) and (iii).

Let $\mathbf{B}$ be any basis for Λ. Since it is an integer matrix, we can consider it as a matrix over $\mathbb{F}_p$, and let V be its row space. Then Exercise 44 says that two matrices $\tilde{\mathbf{M}}$ and $\tilde{\mathbf{M}}'$ exist such that V is the row space of $\tilde{\mathbf{M}}$ and the left kernel of $(\tilde{\mathbf{M}}')^T$.

By construction, if we consider any integer vector $\mathbf{x} = \mathbf{aB} \in \Lambda$ as a vector in $\mathbb{F}_p^n$, it will be in V. It immediately follows that it lies in $\Lambda_p(\mathbf{M})$ and $\Lambda_p^\perp(\mathbf{M}')$.

Now suppose $\mathbf{x} \in \Lambda_p(\mathbf{M})$. Then modulo p we have that $\mathbf{x}$ is congruent to a linear combination of rows of $\mathbf{M}$. But modulo p the rows of $\mathbf{M}$ are linear combinations of the rows of $\mathbf{B}$. Which means that $\mathbf{x}$ is congruent to a lattice vector in Λ modulo p. But Λ is p-ary, so $\mathbf{x}$ is in Λ.

Finally, suppose $\mathbf{x} \in \Lambda_p^\perp(\mathbf{M}')$. Then $\mathbf{x}$ is congruent to something in the left kernel of $(\mathbf{M}')^T$, which means that it is congruent to a vector in V modulo p. But that vector is congruent to something in the row space of $\mathbf{B}$ modulo p, so $\mathbf{x}$ is congruent to a vector in Λ modulo p. But Λ is p-ary, so $\mathbf{x}$ is in Λ.

We have shown that (i) implies (ii) and (iii). □

When we consider the dual lattice $\Lambda_p^*(\mathbf{M})$ as an integer lattice, it is clear that $\Lambda_p^\perp(\mathbf{M}) \subseteq \Lambda_p^*(\mathbf{M})$. Likewise, since for any vector $\mathbf{x} \in \Lambda_p(\mathbf{M})$ and any vector $\mathbf{y} \in \Lambda_p^\perp(\mathbf{M})$ we have that

$$\mathbf{x}\mathbf{y}^T = \mathbf{aM}\mathbf{y}^T \equiv 0 \pmod{p},$$

it follows that $\frac{1}{p}\mathbf{y} \in \Lambda_p^*(\mathbf{M})$ and that

$$\frac{1}{p}\Lambda_p^\perp(\mathbf{M}) \subseteq \Lambda_p^*(\mathbf{M}).$$

In fact, we have equality, since for any $\mathbf{x} \in \Lambda_p(\mathbf{M})$ and $\mathbf{y} \in \Lambda_p^*(\mathbf{M})$ we have that $\mathbf{x}\mathbf{y}^T \in \mathbb{Z}$, which means that $\mathbf{x}(p\mathbf{y})^T \equiv 0 \pmod{p}$.

T **Proposition 22.** *Let Λ be a p-ary lattice and let $\mathbf{M}$ be such that $\Lambda = \Lambda_p(\mathbf{M})$. Then*

$$\frac{1}{p}\Lambda_p^\perp(\mathbf{M}) = \Lambda_p^*(\mathbf{M}).$$

6.7 Short Vectors

D **Definition 7.** Let Λ be a lattice. The *i-th successive minimum* $\lambda_i(\Lambda)$ is the minimal real number such that there are i linearly independent vectors of length at most $\lambda_i(\Lambda)$ in Λ.

It is immediately clear that for a lattice of rank r, there are r successive minima, that $0 < \lambda_1(\Lambda) \leq \lambda_2(\Lambda) \leq \cdots \leq \lambda_n(\Lambda)$, and that $\lambda_1(\Lambda)$ is the shortest length of a non-zero vector in Λ. (Note that $\lambda_2(\Lambda)$ does not have to be the second shortest length of a non-zero vector.)

T **Fact 23.** *There exists a constant γ_n, depending only on n, such that for any lattice Λ*

$$\lambda_1(\Lambda)^2 < \gamma_n \det(\Lambda)^{2/n}.$$

A natural question related to lattices is to ask what $\lambda_1(\Lambda)$ is. For large n, it can be shown that $n/(2\pi e) \leq \gamma_n$, and heuristic arguments suggests that an "average" lattice will not contain non-zero vectors much shorter than $\sqrt{n/(2\pi e)}\det(\Lambda)^{1/n}$.

More practically important is finding a lattice vector of length $\lambda_1(\Lambda)$.

D **Definition 8.** The *shortest vector problem* for a lattice Λ is to find a vector $\mathbf{x} \in \Lambda$ such that $\|\mathbf{x}\| = \lambda_1(\Lambda)$.

Sometimes, there will be more than one non-zero vector of minimal length, so the shortest vector problem may not have a unique answer. For some applications, merely finding a short vector is sufficient.

D **Definition 9.** The *γ-approximate shortest vector problem* for a lattice Λ is to find a vector $\mathbf{x} \in \Lambda$ such that $\|\mathbf{x}\| \leq \gamma\lambda_1(\Lambda)$.

A slightly different problem is to find a lattice vector close to some given point. It also has an approximate version.

D **Definition 10.** The *closest vector problem* for a lattice $\Lambda \subseteq \mathbb{R}^n$ and $\mathbf{z} \in \mathbb{R}^n$ is to find $\mathbf{x} \in \Lambda$ such that for any $\mathbf{y} \in \Lambda$, $\|\mathbf{x} - \mathbf{z}\| \leq \|\mathbf{y} - \mathbf{z}\|$.

The *γ-approximate closest vector problem* for a lattice $\Lambda \subseteq \mathbb{R}^n$ and $\mathbf{z} \in \mathbb{R}^n$ is to find $\mathbf{x} \in \Lambda$ such that for any $\mathbf{y} \in \Lambda$, $\|\mathbf{x} - \mathbf{z}\| \leq \gamma\|\mathbf{y} - \mathbf{z}\|$.

An exhaustive search for short or close vectors will quickly become infeasible as the lattice dimension grows. There is evidence that these lattice problems are hard in a very fundamental way. However, the hardness depends on the lattice and on how the lattice is described.

E *Exercise* 45. Suppose Λ has a given orthogonal basis $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$. Show how to find a shortest vector in Λ and all the successive minima, essentially without any arithmetic.

If the basis is nearly orthogonal, the same approach as in the exercise will find short vectors and make it easier to find a shortest vector.

E *Exercise* 46. Let $\mathbf{B}$ be an invertible $n \times n$ matrix and let $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ correspond to the n rows of $\mathbf{B}$. Let $\mathbf{z} \in \mathbb{R}^n$ and let

$$(\alpha_1, \alpha_2, \ldots, \alpha_n) = \mathbf{z}B^{-1}.$$

Show that

$$\mathbf{z} = \alpha_1\mathbf{b}_1 + \alpha_2\mathbf{b}_2 + \cdots + \alpha_n\mathbf{b}_n.$$

For the following exercise, it is convenient to introduce the following notation: $\lfloor \alpha \rceil$ is the nearest integer to α (rounding halves towards even numbers), and for a vector $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$,

$$\lfloor \boldsymbol{\alpha} \rceil = (\lfloor \alpha_1 \rceil, \lfloor \alpha_2 \rceil, \ldots, \lfloor \alpha_n \rceil).$$

E *Exercise* 47. Suppose Λ has a given orthogonal basis $\mathbf{B}$, and let $\mathbf{z} \in \mathbb{R}^n$. Suppose $\mathbf{a} = \lfloor \mathbf{z}\mathbf{B}^{-1} \rceil$. Explain why $\mathbf{aB}$ is the closest vector in Λ to $\mathbf{z}$.

If the basis is nearly orthogonal, the same approach as in the exercise will tend to find the closest vector if $\mathbf{z}$ was reasonably close to a lattice point.

E *Exercise* 48. Let Λ be the lattice from Example 12 with the basis matrix $\mathbf{B}$ from that exercise and the basis matrix $\mathbf{C}$ given in Exercise 35. Consider the vector $(2,2,2,2) \in \mathbb{R}^4$. Use the rounding strategy from Exercise 47 to find two lattice points. Are both of them "close" to $(2,2,2,2)$. Can you decide if there are lattice points closer to $(2,2,2,2)$ than you have already found?

Finding the closest lattice point in a lattice Λ has a related problem on the dual lattice Λ^*. Note that if $\mathbf{x}$ is a lattice point such that $\|\mathbf{z}-\mathbf{x}\|$ is minimal, then $\mathbf{r} = \mathbf{z}-\mathbf{x}$ has minimal length among the vectors satisfying $\mathbf{r} \equiv \mathbf{z} \pmod{\Lambda}$.

Let $\mathbf{C}$ be a basis for Λ^*. Then $\mathbf{r}\mathbf{C}^T = \mathbf{z}\mathbf{C}^T - \mathbf{x}\mathbf{C}^T$, which means that the difference between $\mathbf{r}\mathbf{C}^T$ and $\mathbf{z}\mathbf{C}^T$ is an integral vector.

7 Lattice-based cryptosystems

While the first cryptosystem we shall look at, the GGH system in Section 7.1, is phrased directly in terms of lattices, many cryptosystems do not directly use lattices, yet their security is essentially based on certain lattice problems being difficult to solve.

We now discuss three important systems, GGH, Regev's Learning-with-errors cryptosystem and NTRU. GGH is directly based on lattices. Regev's cryptosystem is based on the difficulty of solving an overdefined linear system of equations that contains errors. NTRU is based on the difficulty of expressing an element in a polynomial ring as a fraction where the numerator and denominator are both polynomials with short coefficients.

Remark. When we discussed public key encryption schemes based on Diffie-Hellman and RSA, we spent some time discussing attacks against simple variants of these schemes. Similar attacks exist against the cryptosystems in this section, or at least against simplified versions. We do not repeat the discussion.

7.1 The GGH cryptosystem

One idea for a symmetric encryption scheme based on lattices is to have a lattice Λ with a nearly orthogonal basis $\mathbf{B}$ as a secret key. To encrypt we somehow encode the message as a lattice vector $\mathbf{x} \in \Lambda$ and then add *random noise* $\mathbf{r}$ to that vector to get a ciphertext $\mathbf{z} = \mathbf{x} + \mathbf{r}$. To decrypt, we can use our nearly orthogonal basis to find the closest vector $\mathbf{x}$ to $\mathbf{z}$ (using the technique from Exercise 47), and then decode the lattice point to recover the message.

It is clear that we need to limit the magnitude of the random noise, since if it is too big, we will no longer be able to recover $\mathbf{x}$ as the closest vector. This could happen because $\mathbf{x}$ is no longer the closest vector to $\mathbf{z}$, or because our basis is not orthogonal and so does not perfectly solve the closest vector problem.

E *Exercise* 49. Let $\mathbf{B}$ be a basis for a lattice Λ. Show that with $\mathbf{x} \in \Lambda$ and $\mathbf{z} = \mathbf{x} + \mathbf{r}$, then $\lfloor \mathbf{z}\mathbf{B}^{-1} \rceil \mathbf{B} = \mathbf{x}$ if and only if $\lfloor \mathbf{r}\mathbf{B}^{-1} \rceil = \mathbf{0}$.

E *Exercise* 50. Recall that for a real vector $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, we have the norms $\|\boldsymbol{\alpha}\|_1 = \sum_i |\alpha_i|$ and $\|\boldsymbol{\alpha}\|_\infty = \max_i |\alpha_i|$.

Let $\mathbf{B}$ be a basis for a lattice Λ, let ρ be a bound on the $\|\cdot\|_1$ norm of the columns of $\mathbf{B}^{-1}$. Show that for any vector $\mathbf{r}$, we have that

$$\|\mathbf{r}\mathbf{B}^{-1}\|_\infty \leq \rho \|\mathbf{r}\|_\infty.$$

Explain how this can be used to find a bound on the random noise when encrypting, to ensure decryption still works.

It is tempting to turn this idea into a public key encryption scheme by publishing a basis for the lattice. Obviously, we cannot publish our nearly orthogonal basis $\mathbf{B}$, since this is essentially the decryption key.

Recall that any lattice Λ with basis matrix $\mathbf{B}$, if $\mathbf{U}$ is an integer matrix with determinant ± 1, then $\mathbf{C} = \mathbf{U}\mathbf{B}$ is another basis matrix for Λ.

One idea is then to create and publish a different basis for the lattice, one that is not nearly orthogonal, and therefore cannot be used to find the closest vector using the approach from Exercise 47. A natural choice is to use the Hermite normal form for $\mathbf{B}$ as $\mathbf{C}$. (The Hermite normal form is in some sense the worst possible form we can give the public basis, since any adversary could easily compute the Hermite normal form on his own.)

As usual, while we could attempt to embed the message in the vector $\mathbf{x}$, it makes more sense to use $\mathbf{x}$ and $\mathbf{r}$ as keys for a symmetric cryptosystem.

The GGH (Goldreich-Goldwasser-Halevi) public key encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is based on lattices in $\mathbb{R}^n$, a key derivation function $h : \mathbb{R}^n \times \mathbb{R}^n \to \mathcal{K}_s$ and a symmetric cryptosystem $(\mathcal{K}_s, \mathcal{P}, \mathcal{C}, \mathcal{E}_s, \mathcal{D}_s)$ and works as following.

- The *key generation* algorithm $\mathcal{K}$ chooses a lattice Λ by choosing a basis matrix $\mathbf{B}$ that is nearly orthogonal. It chooses an integer matrix $\mathbf{U}$ with determinant ± 1 and computes $\mathbf{C} = \mathbf{U}\mathbf{B}$. It then outputs $ek = \mathbf{C}$ and $dk = \mathbf{B}$.
- The *encryption* algorithm $\mathcal{E}$ takes as input an encryption key $ek = \mathbf{C}$ and a message $m \in \mathcal{P}$. It chooses a random vector $\mathbf{a} \in \mathbb{Z}^n$ and random noise $\mathbf{r}$. Then it computes $\mathbf{x} = \mathbf{a}\mathbf{C}$, $\mathbf{z} = \mathbf{x} + \mathbf{r}$, and encrypts the message as $w = \mathcal{E}_s(h(\mathbf{x}, \mathbf{r}), m)$. It outputs the ciphertext $c = (\mathbf{z}, w)$.
- The *decryption* algorithm $\mathcal{D}$ takes as input a decryption key $dk = \mathbf{B}$ and a ciphertext $c = (\mathbf{z}, w)$. It computes $\mathbf{x} = \lfloor \mathbf{z}\mathbf{B}^{-1} \rceil \mathbf{B}$ and $\mathbf{r} = \mathbf{z} - \mathbf{x}$, and decrypts the message as $m = \mathcal{D}_s(h(\mathbf{x}, \mathbf{r}), w)$. If $\mathcal{D}_s$ outputs the special symbol $\perp$ indicating decryption failure, then $\mathcal{D}$ outputs $\perp$, otherwise it outputs m.

E *Exercise* 51. The above is an informal description of a public key encryption scheme. Implement (use some simple method to choose $\mathbf{B}$, $\mathbf{U}$, $\mathbf{a}$ and $\mathbf{r}$) the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$. Show that the triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a public key encryption scheme.

Remark. Note that we have not said how to choose $\mathbf{B}$ or $\mathbf{U}$.

7.2 Regev's cryptosystem

Let p be a prime. Let $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_l \in \mathbb{F}_p^n$ be a set of randomly chosen vectors that contains n linearly independent vectors. Let $\mathbf{s} \in \mathbb{F}_p^n$, and let

$$\beta_i = \mathbf{a}_i \cdot \mathbf{s}, \qquad i = 1, 2, \ldots, l.$$

If we learn the value of these dot products, we can recover the vector $\mathbf{s}$.

E *Exercise* 52. Given $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_l$ and $\beta_1, \beta_2, \ldots, \beta_l$, show how to compute $\mathbf{s}$.

If we add a bit of randomness to the dot product, it turns out that finding the value $\mathbf{s}$ becomes much more difficult. Let χ be a probability distribution on $\mathbb{F}_p$ and let $e_1, e_2, \ldots, e_l$ be sampled independently according to χ. Let

$$b_i = \mathbf{a}_i \cdot \mathbf{s} + e_i, \qquad i = 1, 2, \ldots, l.$$

Finding $\mathbf{s}$ given $b_1, b_2, \ldots, b_l$ is known as the *learning with errors* problem (we want to learn $\mathbf{s}$ from a set of equations for $\mathbf{s}$ that have errors in them).

E *Exercise* 53. Show that if $l = n$, then the learning with errors problem is impossible to answer with (close to) certainty. (That is, any algorithm trying to solve the learning with errors problem must fail sometimes.)

We shall need one particular property for the probability distribution χ. When $e_1, e_2, \ldots, e_l$ have been sampled independently from χ, then for almost all subsets $S \subseteq \{1, 2, \ldots, l\}$ we have that

$$\sum_{i \in S} e_i = k + \langle p \rangle,$$

for some integer k with $|k| < p/4$.

It can be shown that χ can be chosen such that this requirement is satisfied, and it is still hard to solve the learning with errors problem.

We are now ready to describe a public key encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ based on learning with errors.

- The *key generation* algorithm $\mathcal{K}$ chooses random vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_l$ from $\mathbb{F}_p^n$ such that there are n linearly independent vectors. It chooses a random vector $\mathbf{s} \in \mathbb{F}_p^n$, samples errors $e_1, e_2, \ldots, e_l$ independently according to χ and computes $b_i = \mathbf{a}_i \cdot \mathbf{s} + e_i$ for $i = 1, 2, \ldots, l$. It then outputs $ek = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_l, b_1, b_2, \ldots, b_l)$ and $dk = \mathbf{s}$.

- The *encryption* algorithm $\mathcal{E}$ takes as input an encryption key $ek = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_l, b_1, b_2, \ldots, b_l)$ and a message $m \in \{0, 1\}$. It chooses a random subset $S \subseteq \{1, 2, \ldots, l\}$ and computes

  $$\mathbf{x} = \sum_{i \in S} \mathbf{a}_i \qquad \text{and} \qquad w = m\frac{p-1}{2} + \sum_{i \in S} b.$$

  It outputs the ciphertext $c = (\mathbf{x}, w)$.

- The *decryption* algorithm $\mathcal{D}$ takes as input a ciphertext $c = (\mathbf{x}, w)$. It computes

$$w - \mathbf{x} \cdot \mathbf{s} = t + \langle p \rangle, \qquad |t| < p/2.$$

If $|t| < p/4$, it outputs $m = 0$, otherwise it outputs $m = 1$.

E *Exercise* 54. The above is an informal description of a public key encryption scheme. Implement (use some simple method to sample from χ) the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$. Show that the triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a public key encryption scheme.

Remark. Note that this cryptosystem can only encrypt a single bit, so it is not at all practical. It is, however, of great theoretical interest, in particular because of the learning with errors problem.

7.2.1 Attacks

We shall consider two different ways to construct lattices that will allow us to break the LWE system.

First, we reformulate the scheme in terms of matrices. Let $\mathbf{A}$ be the matrix with rows $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_l$, let $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_l)$, $\mathbf{e} = (e_1, e_2, \ldots, e_l)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$. Then for some very short vector $\mathbf{r}$ we have

$$\mathbf{b} = \mathbf{s}\mathbf{A} + \mathbf{e}, \qquad \mathbf{x} = \mathbf{r}\mathbf{A}, \qquad w = \mathbf{r}\mathbf{b}^T + m\frac{p-1}{2}.$$

Closest vector in a p-ary lattice Let $\mathbf{z}$ be any vector in $\mathbb{Z}^n$ such that $\mathbf{z} \equiv \mathbf{b} \pmod{p}$. The matrix $\mathbf{A}$ defines a p-ary lattice $\Lambda = \Lambda_p(\mathbf{A})$. We may assume that the error distribution for the learning with errors problem is such that the length of $\mathbf{e}$ is smaller than half the length of the shortest vector in Λ. In other words, if $\mathbf{x}$ is a vector in Λ that is closest to $\mathbf{z}$, then $\mathbf{z} - \mathbf{x} \equiv \pm\mathbf{e} \pmod{p}$. Once we know the error vector, we can compute $\mathbf{s}$ using linear algebra. In other words, we can recover the secret key if we can find the closest vector.

7.3 NTRU Encrypt

Superficially, the NTRU cryptosystem is based on polynomial arithmetic. However, the best attacks on the cryptosystem comes from using lattices.

We begin by considering three rings, $R = \mathbb{Z}[X]/\langle X^n - 1\rangle$, $R_p = \mathbb{Z}_p[X]/\langle X^n - 1\rangle$ and $R_q = \mathbb{Z}_q[X]/\langle X^n - 1\rangle$. Note that there are natural homomorphisms from R to the other two rings (but not between the rings), so any element in R may be considered also to be an element of R_p and R_q.

Strictly speaking, the elements in these rings are not polynomials, but cosets. As usual, we take the lowest-degree representative whenever we need a representative. Furthermore, when we consider elements in R_q as polynomials, the coefficients are integers modulo q. We shall pull them back to R by choosing the coefficient representatives

with minimal absolute value, which means that the coefficients will be integers between $-q/2$ and $q/2$.

Consider a polynomial $s \in R$ such that s is invertible in R_p and R_q, and let g be any other polynomial in R. Let y be a polynomial such that $y = g/s$ in R_q. Suppose r, t are polynomials and that $c = pry + t$ has been computed in R_q. Then we have that

$$sc = prg + st$$

in R_q. Furthermore, if the polynomials rg and st have only "small" coefficients, then this equality also holds in R, which means that with if $a \in R$ is equal to sc computed in R_q, then $a = st$ in R_p, or $t = a/s$ in R_p.

We are now ready to describe a public key encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ based on the above equations with parameters n, p and q, a symmetric cryptosystem $(\mathcal{K}_s, \mathcal{P}, \mathcal{C}, \mathcal{E}_s, \mathcal{D}_s)$ and a key derivation function $h : R_q \times R_p \to \mathcal{K}_s$.

- The *key generation* algorithm $\mathcal{K}$ chooses polynomials g and s with "small" coefficients such that s is invertible in R_p and R_q. It computes $y \leftarrow g/s$ in R_q. It then outputs $ek = y$ and $dk = s$.
- The *encryption* algorithm $\mathcal{E}$ takes as input an encryption key $ek = y$ and a message $m \in \mathcal{P}$. It chooses polynomials r and t with "small" coefficients and computes $c \leftarrow pry + t$ in R_q, $k \leftarrow h(c, t)$ and $w \leftarrow \mathcal{E}_s(k, m)$. It outputs the ciphertext $c = (c, w)$.
- The *decryption* algorithm $\mathcal{D}$ takes as input a decryption key $dk = s$ and a ciphertext $c = (c, w)$. It computes $a = sc$ in R_q and $t = a/s$ in R_p. Then it computes $k \leftarrow h(c, t)$ and $m \leftarrow \mathcal{D}_s(k, w)$. If $\mathcal{D}_s$ outputs the special symbol $\perp$ indicating decryption failure, then $\mathcal{D}$ outputs $\perp$, otherwise it outputs m.

E *Exercise* 55. The above is an informal description of a public key encryption scheme. Implement (use some simple method to choose polynomials) the three algorithms $\mathcal{K}$, $\mathcal{E}$ and $\mathcal{D}$. Show that the triple $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a public key encryption scheme (under reasonable assumptions on what "small" means).

7.3.1 Attacking NTRU

Let y be represented by the polynomial $y_0 + y_1 X + \cdots + y_{n-1} X^{n-1} \in \mathbb{Z}[X]$. Likewise, let s and g We can now construct a matrix

$$\mathbf{B} = \begin{pmatrix} \mathbf{I} & \mathbf{H} \\ \mathbf{0} & q\mathbf{I} \end{pmatrix}$$

where $\mathbf{I}$ is a $n \times n$ identity matrix, and the matrix $\mathbf{H}$ consists of the cyclic shifts of the coefficients of above representation of the y polynomial. That is, the ith row (starting at 0) of $\mathbf{H}$ is $(y_{n-i}, y_{n-1}, y_0, y_1, \ldots, y_{n-i-1})$. This $2n \times 2n$ matrix has full rank, so it generates a full rank lattice Λ.

Now observe that if $a, b \in R_q$ are such that $b = ay$, then if a and b are represented by $a_0 + a_1X + \cdots + a_{n-1}X^{n-1} \in \mathbb{Z}[X]$ and $b_0 + b_1X + \ldots b_{n-1}X^{n-1} \in \mathbb{Z}[X]$, we have that in R

$$\left(\sum_i a_i X_i\right)\left(\sum_j y_j X_j\right) = \sum_i X_i \sum_j a_{i+j} y_{n-j} = \sum_i b_i X^i + \sum_i q w_i X^i.$$

In other words, there exists a polynomial $w \in \mathbb{Z}[X]$ such that

$$\left(\sum_i s_i X_i\right)\left(\sum_j y_j X_j\right) = \sum_i g_i X^i - \sum_i q w_i X^i.$$

It follows that with $\mathbf{a} = (s_0, s_1, \ldots, s_{n-1}, w_0, w_1, \ldots, a_{n-1})$, we have

$$\mathbf{aH} = (s_0, s_1, \ldots, s_{n-1}, g_0, g_1, \ldots, g_{n-1}).$$

Since s and g have "small" coefficients, the length of this vector is a small multiple of $\sqrt{2n}$.

We have already noted that an "average" lattice of dimension $2n$ will not have vectors much shorter than $\sqrt{2n/(2\pi e)}\det(\Lambda)^{1/(2n)}$. The determinant of our lattice is q^n, so we compare a small multiple of $\sqrt{2n}$ with $\sqrt{2nq/(2\pi e)}$. If n is not too small, our lattice will have a very short vector. It is then not too unreasonable to hope that a short vector in the lattice will be this short vector (or one closely related to it).

In other words, if we can find short vectors in the lattice Λ, we have a reasonable hope of finding the NTRU decryption key.

8 Lattice algorithms

We have now seen how we can attack several cryptosystems by either finding a closest vector in a lattice or finding a short vector in a lattice. It follows that in order to understand the security of these cryptosystems, we must understand how to find short vectors and closest vectors.

We begin by developing an algorithm for enumerating all the lattice vectors in a ball around the origin. It will turn out that this algorithm is sensitive to certain properties of the basis. We shall then study the famous LLL algorithm which will produce a basis that is (among other things) a much better starting point for the enumeration algorithm.

8.1 Enumerating short vectors

We would like to find a short vector in a lattice. One idea would simply be to enumerate all linear combinations of the basis vectors with some bound on the coefficients. Unfortunately, short vectors could in principle come from linear combinations with large

coefficients. Instead, we shall use the Gram-Schmidt basis to bound the size of the coefficients.

We shall find all points $\mathbf{x}$ in a lattice with $\|\mathbf{x}\|^2 \leq A^2$ for some bound A^2.

Let $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ be a basis for Λ, and let $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$ be the corresponding Gram-Schmidt basis. Note that for any vector $\mathbf{x} \in \Lambda$, we can write it as a linear combination of the Gram-Schmidt basis vectors $\mathbf{x} = \sum_i \alpha_i \mathbf{b}_i^*$ and then

$$\|\mathbf{x}\|^2 = \sum_{i=1}^{n} \alpha_i^2 \|\mathbf{b}_i^*\|^2.$$

Recall that $\mathbf{b}_n^*$ is the part of $\mathbf{b}_n$ that is orthogonal to all the earlier basis vectors. This means when $\mathbf{x} = \sum_i a_i \mathbf{b}_i = \sum_i \alpha_i \mathbf{b}_i^*$, then $a_n = \alpha_n$. Therefore, if $\alpha \|\mathbf{b}_n^*\| > A$ then $\|\mathbf{x}\| > A$.

We therefore begin by enumerating vectors with n-th coordinate a_n between $-\lfloor A/\|\mathbf{b}_n^*\| \rfloor$ and $+\lfloor A/\|\mathbf{b}_n^*\| \rfloor$.

Given a_n, we can now consider the possibilities for a_{n-1}. Of course, this time, the contribution in the direction of $\mathbf{b}_{n-1}^*$ is that given by $a_{n-1}\mathbf{b}_{n-1}$ and $a_n \mathbf{b}_n$, where the latter's contribution is $a_n \mu_{n,n-1} \|\mathbf{b}_{n-1}^*\|$. So given a_n, we want to enumerate all the $(n-1)$-th coordinates a_{n-1} such that

$$(a_{n-1} + a_n \mu_{n,n-1})^2 \|\mathbf{b}_{n-1}^*\|^2 + a_n^2 \|\mathbf{b}_n^*\|^2 \leq A^2.$$

In general, given $a_{i+1}, \ldots, a_n$, we consider the possibilities for a_i. Again, we want to enumerate all a_i such that

$$(a_i + \sum_{j=i+1}^{n} a_j \mu_{ji})^2 \|\mathbf{b}_i^*\|^2 + \sum_{j=i+1}^{n} (a_j + \sum_{k=j+1}^{n} a_k \mu_{k,j})^2 \|\mathbf{b}_j^*\|^2 \leq A^2.$$

For some choices of $a_{i+1}, \ldots, a_n$ there may be no possible choices for a_i, in which case we stop and continue with other choices for $a_{i+1}, \ldots, a_n$.

Whenever we find a non-empty region for a_1 and enumerate those values, we enumerate lattice vectors of length less than A. It is clear that for any lattice point $\mathbf{x}$ of length less than A, this point must be among the lattice point eventually enumerated.

This enumeration algorithm must enumerate a large number of possible coordinates. For the ith coordinate, we can upper-bound the number of possibilities for a_i by $A/\|\mathbf{b}_i^*\|$, so the total number of coordinates enumerated is bounded by

$$\prod_{i=1}^{n} \frac{A}{\|\mathbf{b}_i^*\|} = \frac{A^n}{\det(\Lambda)}.$$

E *Exercise* 56. The above discussion describes an algorithm for enumerating all the vectors in a lattice shorter than some bound. Implement this algorithm.

E *Exercise* 57. Consider the lattice given by the basis matrix $\mathbf{C}$ given in Exercise 35. Enumerate all the vectors in the lattice of length at most 3.

We can also solve the closest vector problem if we can first find an estimate for the closest vector and a reasonable upper bound on how far the estimate is from the closest point. Given this, we can enumerate all the short vectors and thereby all the lattice points close to the estimate, one of which must be closest.

There are faster algorithms for finding a shortest or closest vector.

8.2 LLL algorithm

As we saw, if we have an orthogonal basis, we can solve the closest vector problem, and if we have a nearly orthogonal basis, we can solve closest vector problem if the closest vector is close enough to a lattice point.

The natural question is how to find a reasonably good basis that will allow us to solve the closest vector problem. The first goal should be to be precise about what we mean by "reasonably good".

D **Definition 11.** Let $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ be a lattice basis, with corresponding orthogonal basis $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$ and Gram-Schmidt coefficients μ_{ij} for $1 \leq j < i \leq n$, as defined in (6). Let $\frac{1}{4} < \delta < 1$ be a real number. We say that the basis is *δ-LLL-reduced* if

$$|\mu_{ij}| \leq \frac{1}{2} \qquad \text{for all } 1 \leq j < i \leq n, \text{ and} \tag{8}$$

$$\delta \|\mathbf{b}_{i-1}^*\|^2 \leq \|\mathbf{b}_i^*\|^2 + \mu_{i,i-1}^2 \|\mathbf{b}_{i-1}^*\|^2 \qquad \text{for all } 2 \leq i \leq n. \tag{9}$$

When a basis satisfies (8) we cannot easily make the basis vectors more orthogonal. When the basis satisfies (9), the basis vectors of the Gram-Schmidt orthogonalization will be ordered roughly according to length.

E *Exercise* 58. A common choice for δ is $3/4$. Show that in this case (9) implies

$$\|\mathbf{b}_{i-1}^*\|^2 \leq 2\|\mathbf{b}_i^*\|^2 \text{ for all } 2 \leq i \leq n.$$

Hint: You may use the fact that (8) also must hold.

That an LLL-reduced basis is somehow a good basis can be seen from the following fact, which we state without proof.

T **Fact 24.** *Suppose $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ with corresponding basis matrix $\mathbf{B}$ is a $3/4$-LLL-reduced basis for a lattice Λ. Then $\|\mathbf{b}_1\| \leq 2^{(n-1)/2}\lambda_1(\Lambda)$. Also, if $\mathbf{z} \in \mathbb{R}^n$, then*

$$\|\mathbf{z} - \lfloor \mathbf{z}\mathbf{B}^{-1} \rceil \mathbf{B}\| \leq (1 + 2n(9/2)^{n/2})\|\mathbf{z} - \mathbf{x}\| \text{ for any } \mathbf{x} \in \Lambda.$$

We know that $\lambda_1(\Lambda) \leq \sqrt{\gamma}\det(\Lambda)^{1/n}$, which means that if we have an LLL-reduced basis and use $\|\mathbf{b}_1\|$ as our search bound, the enumeration approach from the previous section will have to enumerate at most

$$\frac{(2^{(n-1)/2}\gamma^{1/2}\det(\Lambda)^{1/n})^n}{\det(\Lambda)} = 2^{n(n-1)/2}\gamma^{n/2}.$$

While it does not affect the upper bound we deduced, having the Gram-Schmidt vectors not too small will decrease the total number of points the algorithm will iterate over.

We also note that the LLL-reduced basis will give us an estimate for the closest vector problem. (There are better ways to use the LLL-reduced basis.)

Having an LLL-reduced basis is therefore very useful. The question is how to find an LLL-reduced basis.

We will now discuss two ways to modify a lattice basis. The first is to use the initial basis vectors to modify the later basis vectors. This clearly results in a new basis for the same lattice, but the new basis will have the same Gram-Schmidt orthogonalization.

E *Exercise* 59. Let $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ be a lattice basis, and let $\mathbf{c}_1, \ldots, \mathbf{c}_n$ be another basis for the lattice defined by

$$\mathbf{c}_i = \mathbf{b}_i - \sum_{j=1}^{i-1} \alpha_{ij} \mathbf{b}_j,\ \alpha_{ij} \in \mathbb{Z}. \tag{10}$$

Let $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$ and $\mathbf{c}_1^*, \ldots, \mathbf{c}_n^*$ be the corresponding Gram-Schmidt orthogonalization. Show that $\mathbf{b}_i^* = \mathbf{c}_i^*$ for $i = 1, 2, \ldots, n$.

With this result in mind, we now turn to (7) which describes the relationship between a basis and its Gram-Schmidt orthogonalization,

$$\mathbf{B} = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ \mu_{ij} & & 1 \end{pmatrix} \mathbf{B}^*.$$

By adding or subtracting a suitable integer multiple of the rightmost column of the Gram-Schmidt coefficient matrix, we can ensure that $|\mu_{n,n-1}| \leq 1/2$. Then, by subtracting suitable integer multiples of the two rightmost columns, we can ensure that $|\mu_{n-1,n-2}| \leq 1/2$ and $|\mu_{n,n-2}| \leq 1/2$. In this way we can ensure that any element below the diagonal in the coefficient matrix has absolute value at most $1/2$.

Since column arithmetic in the coefficient matrix on the right-hand side of the equation corresponds to row arithmetic in the basis matrix on the left hand side, the above procedure essentially tells us how to modify a lattice basis so that its Gram-Schmidt coefficients satisfy (8). The next exercise details a more algorithmic way to do this computation.

E *Exercise* 60. Let $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ be a lattice basis, let $\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_n^*$ be the Gram-Schmidt orthogonalization with coefficients μ_{ij}, $1 < i \leq n$.

Suppose $|\mu_{ij}| \leq 1/2$ for $1 \leq j < i < k$ for some $k \leq n$. Define $\mathbf{b}_k^{(i)}$ for $1 \leq i \leq k$ by $\mathbf{b}_k^{(k)} = \mathbf{b}_k$ and

$$\mathbf{b}_k^{(i)} = \mathbf{b}_k^{(i+1)} - \left\lfloor \frac{\langle \mathbf{b}_k^{(i+1)}, \mathbf{b}_i^* \rangle}{\langle \mathbf{b}_i^*, \mathbf{b}_i^* \rangle} \right\rceil \mathbf{b}_i.$$

Consider now the new basis $\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}, \mathbf{b}_k^{(1)}, \mathbf{b}_{k+1}, \ldots, \mathbf{b}_n$.

1. Show that the new basis has the same Gram-Schmidt orthogonalization as the old basis.

2. Let μ'_{ij} be the Gram-Schmidt coefficients of the new basis. Show that $|\mu'_{ij}| \leq 1/2$ for $1 \leq j < i \leq k$.

3. Show that the above procedure essentially gives us an algorithm that for any lattice basis computes a new, equivalent basis with the same Gram-Schmidt orthogonalization that also satisfies (8).

4. Show that the resulting algorithm will compute the new basis using at most $2n^3$ arithmetic operations.

This result tells us that we can not only modify a lattice basis such that (8) holds without changing the Gram-Schmidt orthogonalization, but we can also do this relatively quickly.

Next, we want to modify our basis by changing the order of the basis vectors. Unlike the previous modification, this will change the resulting Gram-Schmidt orthogonal basis.

Before we describe and analyse this change, we need to define a new kind of volume for lattices that weights the basis vectors differently. For a basis $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ with corresponding Gram-Schmidt orthogonalization $\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_n^*$, define

$$d(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n) = \prod_{i=1}^{n} \prod_{j=1}^{i-1} \|\mathbf{b}_j^*\| = \prod_{i=1}^{n} \|\mathbf{b}_i^*\|^{n-i+1}.$$

Suppose $\mathbf{b}_1, \ldots, \mathbf{b}_n$ satisfies (8), but not (9), and i is the first index where the latter condition fails. Now suppose we change the order of the ith and $i+1$th basis vectors, so instead of considering the basis $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_i, \mathbf{b}_{i+1}, \ldots, \mathbf{b}_n$, we instead consider the basis $\mathbf{c}_1, \ldots, \mathbf{c}_n$ given by

$$\mathbf{c}_j = \begin{cases} \mathbf{b}_{i+1} & j = i \\ \mathbf{b}_i & j = i+1 \\ \mathbf{b}_j & \text{otherwise.} \end{cases}$$

E *Exercise* 61. Let $\mathbf{b}_1, \ldots, \mathbf{b}_n$ and $\mathbf{c}_1, \ldots, \mathbf{c}_n$ be as above, and let $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$ and $\mathbf{c}_1^*, \ldots, \mathbf{c}_n^*$ be the corresponding Gram-Schmidt orthogonalizations. Let μ_{ij} be the Gram-Schmidt coefficients for $\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*$.

1. Show that $\mathbf{b}_j^* = \mathbf{c}_j^*$ when $j \neq i, i+1$.

2. Show that $\mathbf{c}_i^* = \mathbf{b}_{i+1}^* + \mu_{i+1,i}\mathbf{b}_i^*$.

3. Show that

$$\prod_{j=1}^{k} \|\mathbf{b}_j^*\| = \prod_{j=1}^{k} \|\mathbf{c}_j^*\|$$

when $i \neq k$.

4. Show that

$$\frac{d(\mathbf{c}_1, \ldots, \mathbf{c}_n)}{d(\mathbf{b}_1, \ldots, \mathbf{b}_n)} \leq \sqrt{\delta}.$$

E *Exercise* 62. Show that for any lattice $\Lambda \in \mathbb{Z}^n$, then regardless of basis, there is a lower bound to $d(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ that is strictly larger than zero.

We are now ready to prove that a basis satisfying (8) and (9) can be computed relatively quickly.

T **Theorem 25.** *Let* $\mathbf{B}$ *be a basis for a lattice* $\Lambda \in \mathbb{Z}^n$*. Then a* δ*-LLL-reduced basis* $\mathbf{C}$ *can be computed using less than* $10n^3 \log d(\mathbf{B}) / \log \delta^{-1}$ *arithmetic operations.*

Proof. We begin by constructing a sequence of lattice bases $\mathbf{B}_1, \mathbf{B}_2, \ldots$ as follows.

The initial basis is $\mathbf{B}_1 = \mathbf{B}$.

We construct the $(k+1)$th basis from the kth basis $\mathbf{B}_k$ using the following two steps.

1. Use the algorithm from Exercise 60 to create a basis $\mathbf{B}'_k$.
2. If the basis $\mathbf{B}'_k$ does not satisfy (9), and i is the first index where this condition fails we change the order of the ith and the $(i+1)$th basis vectors.

It is clear that if $\mathbf{B}_k$ satisfies (8) and (9), then $\mathbf{B}_{k+1} = \mathbf{B}_k$.

Furthermore, unless we changed the order of two basis vectors when creating $\mathbf{B}_{k+1}$, then $\mathbf{B}_{k+1}$ satisfies (8) and (9).

Every time we do change the order of two basis vectors, Exercise 61 says that

$$\frac{d(\mathbf{B}_{k+1})}{d(\mathbf{B}_k)} \leq \sqrt{\delta},$$

from which we get that

$$d(\mathbf{B}_{k+1}) \leq \delta^{k/2} d(\mathbf{B}_1).$$

Since $d(\mathbf{B}_{k+1}) \geq 1$, it follows that when $k > 2 \log d(\mathbf{B}_1) / \log \delta^{-1}$ there can be no more changes of order, and we have computed a δ-LLL-reduced basis.

By Exercise 60 the first step requires at most $2n^3$ arithmetic operations. To do the second step, we need the Gram-Schmidt coefficients which we can compute using at most $2n^3$ arithmetic operations by Exercise 36. Finally, we can find the first index for which (9) does not hold using less than $3n^2$ arithmetic operations. This means that we can compute $\mathbf{B}_{k+1}$ from $\mathbf{B}_k$ using less than $5n^3$ arithmetic operations. The claim follows. □

E *Exercise* 63. The proof of Theorem 25 essentially describes an algorithm for computing an LLL-reduced basis for a lattice. Implement this algorithm and restate Theorem 25 as a statement about the algorithm's time complexity (in terms of arithmetic operations, ignoring any other form of computation involved).

We must make two remarks about this theorem and its proof. The first remark is about measuring complexity in terms of arithmetic operations. Often, this is safe, but in this case we will need to do arithmetic with rational numbers, and in some computations the size (number of digits) of the rational numbers involved will increase exponentially, even when the number of operations is limited. Fortunately, it can be proved that the

size of the rational numbers involved do not grow too badly, although it significantly affects the time required to run the algorithm.

The second remark is about the implied algorithm, which is extremly inefficient. In practice, there is no need to constantly recompute all the Gram-Schmidt coefficients. In other words, there is significant scope for optimization in the algorithm. Also, the proof significantly overestimates the cost of the implied algorithm.

E *Exercise* 64. Consider the lattice Λ given in Example 12 and the basis matrix $\mathbf{B}$ given in the example and the basis matrix $\mathbf{C}$ given in Exercise 35. For each basis, compute a 3/4-LLL-reduced basis.

9 Key Encapsulation Mechanisms

Most of the public key cryptosystems we have studied so far follows a similar pattern: a random key for a symmetric cryptosystem is encapsulated inside some object, and this key is then encrypted with the symmetric cryptosystem. The ciphertext then consists of the encapsulated key and the encrypted message. To decrypt the pair, we extract the symmetric key from the encapsulation and then use it to decrypt.

It is convenient to precisely define what a key encapsulation mechanism is.

D **Definition 12.** A *key encapsulation mechanism* (KEM) consists of three algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ and an associated symmetric key set $\mathcal{K}_s$:

- The *key generation* algorithm $\mathcal{K}$ takes no input and outputs an *encapsulation key* ek and a *decapsulation key* dk.
- The *encapsulation* algorithm $\mathcal{E}$ takes as input an encapsulation key ek and outputs an encapsulation (ciphertext) c and a key $k \in \mathcal{K}_s$.
- The *decapsulation* algorithm $\mathcal{D}$ takes as input a decapsulation key dk and an encapsulation (ciphertext) c and outputs either a key k or the special symbol $\perp$ indicating decryption failure.

We require that for any key pair (ek, dk) output by $\mathcal{K}$ and any pair (c, k) output by $\mathcal{E}$, we have that $\mathcal{D}(dk, c) = k$.

Remark. Sometimes we allow the KEM to generate keys from a different set than our symmetric encryption scheme uses. In this case, as we see in Section 7.1 and 7.3 we then use a key derivation function to get keys suitable for our symmetric encryption scheme.

The benefit of doing this is that we do not have to redesign our KEM if we change our symmetric encryption scheme. Instead, we only have to change the key derivation function, which is typically much easier.

Security for a KEM is mostly the same as for public key encryption, except that non-malleability is not immediately interesting.

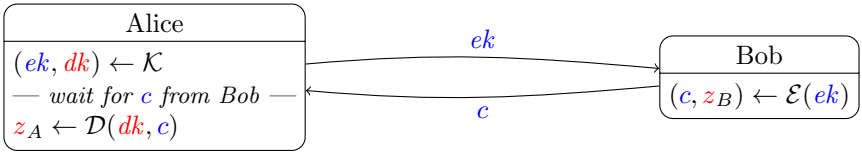

Figure 3: A key exchange protocol based on a key encapsulation mechanism $(\mathcal{K}, \mathcal{E}, \mathcal{D})$.

> **Informally:** A key encapsulation mechanism provides *confidentiality* if it is hard to learn anything at all about the decapsulation of an encapsulation from the encapsulation itself.

We shall not discuss key encapsulation mechanisms any further here, except to note that if we have a KEM we can construct a key exchange protocol, given in Figure 3, that is very similar to the Diffie-Hellman protocol.

E *Exercise* 65. Extract the key encapsulation methods used from the public key encryption schemes from Sections 3, 4.4, 7.1 and 7.3. Describe the algorithms and prove that they are key encapsulation methods.

10 The Public Key Infrastructure Problem

As we have seen, we can construct plausible cryptosystems where anyone who knows the encryption key can encrypt messages, but only those who know the decryption key can decrypt messages.

One problem remains. Alice wants to send a message to Bob. How does she get Bob's encryption key? Suppose Alice finds a key that she thinks belong's to Bob, but which in reality belongs to Eve who has the corresponding decryption key. Eve can then easily decrypt Alice's ciphertext.

One possible solution is a public key directory, modeled on a telephone directory, listing people and their public keys. It seems impractical to have printed copies of this directory, so it would have to be an online service, which begs the question: How can Alice be sure that the encryption key she just fetched came from the directory, and not from Eve?

The *public key infrastructure* problem is Alice's problem of getting hold of Bob's public key.

Another interesting problem is what happens when Bob receives Alice's ciphertext. How does he know that it comes from Alice, and not from Eve?

It turns out that both of these problems have reasonable solutions involving digital signatures.