A Brief Introduction to Symmetric Cryptography

KG

October 24, 2019

Contents

1 Introduction

In this note, we consider the following problem. Alice wants to send messages to Bob via some communications channel. Eve has access to the channel and she may eavesdrop on and possibly tamper with anything sent over the channel.

Alice does not want Eve to be able to eavesdrop on her messages. She wants to communicate *confidentially*. Also, when Bob receives a message that looks like it came from Alice, then Bob wants to be sure that Alice really sent the message and that Eve did not tamper with it. Alice and Bob want *integrity*.

Alice and Bob share a secret, called the *key*. Cryptography where the sender and the receiver (the honest users) have the same knowledge is called *symmetric cryptography*, where the word symmetry refers to the symmetry of knowledge.

We define what a *symmetric cryptosystem* is and what the *security requirements* are for such cryptosystems in Section 2.

To illustrate standard *attacks*, it is useful to study some historic cryptosystems and how those systems can be attacked. This is done in Section 3, which alternates between discussing historical ciphers and discussing interesting attacks that apply to the historical ciphers. (Note that Section 3 is no history of cryptography.)

Section 3 contains a few constructions that provide confidentiality against eavesdroppers. These constructions do not provide integrity, nor do they provide confidentiality if Eve is willing to tamper with ciphertexts.

The main tool for providing integrity is discussed in Section 4. How to combine the constructions provided in Sections 3 and 4 into cryptosystems providing both integrity and confidentiality even when Eve tampers with the ciphertexts is discussed in Section 5.

This text is intended for a reader that is familiar with mathematical language, basic algebra (groups, rings, fields, linear algebra and polynomials), elementary probability theory and elementary computer science (algorithms).

This text is very informal. Every concept and result mentioned in the text can be made precise, but the technical details are out of scope for this text.

While modern high-level constructions are discussed in this note, low-level constructions are out of scope. This explains why this note defines what a block cipher is and gives an informal explanation of what it means for a block cipher to be secure, but does not contain a single example of a modern block cipher.

Another topic that is out of scope is proving the security of the modern constructions discussed. We only include proofs for *information theoretically secure* constructions.

This text uses colour to indicate who is supposed to know what. Red denotes secret information (typically keys) known only by Alice and Bob. Green denotes information that Alice and Bob want to protect, typically messages. Blue denotes information that the eavesdropper will see.

2 Basic Definitions

We begin with the definition of a symmetric cryptosystem and what it means for a cryptosystem to be *secure*.

D **Definition 1.** A *symmetric cryptosystem* consists of

- a set $\mathcal{K}$ of *keys*;
- a set $\mathcal{P}$ of *plaintexts*;
- a set $\mathcal{C}$ of *ciphertexts*;
- an *encryption algorithm* $\mathcal{E}$ that on input of a key and a plaintext outputs a ciphertext; and
- a *decryption algorithm* $\mathcal{D}$ that on input of a key and a ciphertext outputs either a plaintext or the special symbol $\perp$ (indicating an invalid ciphertext).

For any key k and any plaintext m, we have that

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

The set $\mathcal{P}$ will usually be a set of finite sequences of letters from an *alphabet*.

We shall assume that the key Alice and Bob share has been chosen uniformly at random from the set of keys.

3 Confidentiality Against Eavesdroppers

In this section we shall consider the situation where Eve is eavesdropping on Alice and Bob. Eve's goal is to understand what Alice is saying to Bob.

We shall briefly discuss some historic cryptosystems. We do this to give a gentle introduction to the basic concepts in cryptograpy and provide some insight into important attack strategies.

The presentation in this section alternates between describing a cryptosystem and describing how to attack that cryptosystem, until we reach systems that will provide confidentiality.

> **Informally:** A symmetric cryptosystem provides *confidentiality* if it is – without knowledge of the key – hard to learn anything at all about the decryption of a ciphertext from the ciphertext itself, except possibly the length of the decryption.

3.1 Shift Cipher

The shift cipher is also known as the Cæsar cipher.

Suppose first that we give our alphabet G some group structure. There is a natural bijection between the English alphabet $\{\text{A}, \text{B}, \text{C}, \ldots, \text{Z}\}$ and the group $\mathbb{Z}_{26}^{+}$, given by $0 \leftrightarrow \text{A}$, $1 \leftrightarrow \text{B}$, etc. We add F and G by applying the bijection to get 5 and 6, adding them to 11, and then applying the inverse bijection to get L.

The *plaintext* m is a sequence of letters $m_1 m_2 \ldots m_L$ from the alphabet. The *key* is an element k from G. We *encrypt* the message by adding the key to each letter, that is, the ith ciphertext letter is

$$c_i = m_i + k, \qquad 1 \leq i \leq L. \tag{1}$$

The ciphertext c is the sequence of letters $c_1 c_2 \ldots c_L$.

To *decrypt* a ciphertext $c = c_1 \ldots c_L$, we subtract the key from each ciphertext letter, that is, the ith plaintext letter is

$$m_i = c_i - k, \qquad 1 \leq i \leq L.$$

E *Exercise* 1. The above is an informal description. Write down carefully what the three sets $\mathcal{K}$, $\mathcal{P}$ and $\mathcal{C}$ are, and implement the two algorithms $\mathcal{E}$ and $\mathcal{D}$. Show that $(\mathcal{K}, \mathcal{P}, \mathcal{C}, \mathcal{E}, \mathcal{D})$ is a symmetric cryptosystem.

E *Exercise* 2. How many different keys are there for the shift cipher when the alphabet has 26 elements?

E *Example* 1. If we want to encrypt BABOONSAREFUNNY using the shift cipher with the key $k = \text{D}$ (D corresponds to the number 3), we get the following computations:

B	A	B	O	O	N	S	A	R	E	F	U	N	N	Y
+D	+D	+D	+D	+D	+D	+D	+D	+D	+D	+D	+D	+D	+D	+D
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
E	D	E	R	R	Q	V	D	U	H	I	X	Q	Q	B

The ciphertext is then EDERRQVDUHIXQQB.

3.1.1 Attack: Exhaustive Search

The easiest attack on the shift cipher is an *exhaustive search* for the key, or a *brute force attack*. The two assumptions required for this attack is that only one key will give a reasonable decryption, and that we will be able to recognize that decryption. Both of these assumptions are almost always true.

If there are few keys, we can decrypt with all possible keys in reasonable time. The correct key will be the one that gives a reasonable decryption.

E *Exercise* 3. Find all the possible decryptions of HGHUUT. How many are English words? What about the possible decryptions of MBQ?

E *Exercise (for groups)* 4. Choose a key for the Shift cipher at random and encrypt some message. Give the ciphertext to someone else in the group and have them decrypt it without knowing the key.

3.2 Affine Cipher

Now we give our alphabet R a ring structure, say like $\mathbb{Z}_{26}$. We add as before. We multiply F and G by applying the bijection to get 5 and 6, multiplying them to get 30 which is 4 modulo 26, and then applying the inverse bijection to get E.

The plaintext m is a sequence of letters $m_1 m_2 \dots m_L$ from the alphabet. The key is a pair (k_1, k_2) of ring elements, the first of which must be invertible. We encrypt the message letterwise using the formula

$$c_i = k_1 m_i + k_2, \qquad 1 \le i \le L. \tag{2}$$

The ciphertext c is the sequence of letters $c_1 c_2 \dots c_L$.

To decrypt a ciphertext $c = c_1 \dots c_L$, we compute the ith plaintext letter using the formulas

$$m_i = k_1^{-1}(c_i - k_2), \qquad 1 \le i \le L.$$

E *Exercise* 5. The above is an informal description. Write down carefully what the three sets $\mathcal{K}$, $\mathcal{P}$, $\mathcal{C}$ are, and implement the two algorithms $\mathcal{E}$ and $\mathcal{D}$. Show that $(\mathcal{K}, \mathcal{P}, \mathcal{C}, \mathcal{E}, \mathcal{D})$ is a symmetric cryptosystem.

E *Exercise* 6. How many different keys are there for the affine cipher when the alphabet has 26 elements?

3.2.1 Attack: Known Plaintext

Suppose Eve knows that Alice always begins her messages with HI. One ciphertext starts with the letters UB. When the attacker knows the plaintext corresponding to a piece of ciphertext, that is called *known plaintext*.

Eve knows that Alice used the affine cipher, which means that equation (2) was used to encrypt H to U and I to B. She gets the following two equations:

$$\begin{aligned} \mathrm{U} &= k_1 \mathrm{H} + k_2, \\ \mathrm{B} &= k_1 \mathrm{I} + k_2. \end{aligned} \tag{3}$$

This is a linear system of equations with two equations and two unknowns. As long as the difference $\mathrm{H} - \mathrm{I}$ is invertible in the ring (which it is), we can solve the system and recover the key (k_1, k_2).

E *Exercise* 7. Solve the linear system of equations given in (3) to find the key.

E *Exercise (for groups)* 8. Choose a key for the affine cipher at random and encrypt some message. Give the ciphertext along with some known plaintext to someone else in the group and have them decrypt it without knowing the key.

E *Exercise* 9. Develop a similar known plaintext attack for the Shift cipher from Section 3.1.

3.3 Substitution Cipher

The formulas (1) and (2) define bijections on the alphabet. We can generalize these schemes by using any bijection or *permutation* on our alphabet. Let our alphabet be a set S.

The plaintext m is a sequence of letters $m_1 m_2 \dots m_L$ from the alphabet. The key is a permutation π on S. We encrypt the message letterwise using the formula

$$c_i = \pi(m_i), \qquad 1 \le i \le L. \tag{4}$$

The ciphertext c is the sequence of letters $c_1 c_2 \dots c_L$.

To decrypt a ciphertext $c = c_1 \dots c_L$, we compute the ith plaintext letter using the formula

$$m_i = \pi^{-1}(c_i), \qquad 1 \le i \le L.$$

E *Exercise* 10. The above is an informal description. Write down carefully what the three sets $\mathcal{K}$, $\mathcal{P}$, $\mathcal{C}$ are, and implement two algorithms $\mathcal{E}$ and $\mathcal{D}$. Show that $(\mathcal{K}, \mathcal{P}, \mathcal{C}, \mathcal{E}, \mathcal{D})$ is a symmetric cryptosystem.

E *Exercise* 11. How many different keys are there for the substitution cipher when the alphabet has 26 elements?

E *Exercise* 12. Explain how we can recover part of the key (a *partial key*) from known plaintext, but not necessarily the full key.

3.3.1 Attack: Frequency Analysis

Known plaintext will reveal part of the key. But there are stronger attacks on the substitution cipher, based on the number of times the various ciphertext letters appear.

If the permutation takes the plaintext letter A to the ciphertext letter Z, the number of Z's in the ciphertext will be the same as the number of A's in the plaintext. This means that the relative frequencies of the ciphertext letters will be the same as the relative frequencies of the plaintext letters, up to permutation.

For most long English texts, the relative frequency of the various letters is constant. This means that for encryptions of long English texts, the relative frequencies of ciphertext letters is a simple permutation of the relative frequencies of letters in English text. It will be a simple matter of matching plaintext letters and ciphertext letters and thereby recovering the key and thus the plaintext.

For English texts of moderate length, the relative frequencies of the less common letters will vary a lot, and reliable matching of plaintext letters to ciphertext letters will be impossible. However, some letters, E in particular, are so common in English that they will usually be the most common letters, even for fairly short texts.

E *Exercise* 13. Gather a collection of English texts of varying topic and length. Compute the frequency distributions. Use these distributions to estimate how long a text must be before we can expect to identify with reasonable certainty (a) E, (b) the five most frequent letters, and (c) the ten most frequent letters.

This means that even though we cannot reliably match every plaintext letter to every ciphertext letter, we can match a few plaintext letters to a few ciphertext letters. This gives us a partial key and a partial decryption.

The next step is to pretend that this partial decryption is a crossword puzzle, and guess some plaintext words that fit with the partial decryption. We then treat these guesses as known plaintext and recover more of the key. This gives us a better partial decryption. If the new partial decryption does not make sense or is impossible, we guessed wrong. We backtrack and guess again.

If, on the other hand, the new partial decryption makes sense, we probably guessed right. Now we treat the new partial decryption as a crossword puzzle. We repeat this process of *guessing* and *verifying* until we have the complete decryption.

E *Exercise (for groups)* 14. Choose a key for the substitution cipher and encrypt some sufficiently long message. Give the ciphertext to someone else in the group and have them decrypt it without knowing the key. You may also give them a small amount of known plaintext.

3.4 Towards Block Ciphers

One approach to preventing frequency analysis is to use a permutation on pairs of letters. That is, our permutation acts on the set S of all pairs of letters, not the set of letters.

E *Exercise* 15. For a substitution cipher based on permutations on pairs, write down carefully what the three sets $\mathcal{K}$, $\mathcal{P}$, $\mathcal{C}$ are, and implement the two algorithms $\mathcal{E}$ and $\mathcal{D}$. Show that $(\mathcal{K}, \mathcal{P}, \mathcal{C}, \mathcal{E}, \mathcal{D})$ is a symmetric cryptosystem.

Unfortunately, the frequencies of pairs are uneven, which means that frequency analysis still works, although it is less effective. A permutation on triples of letters would be better, but still not perfect.

Even better would be l-tuples. The number l is called the *block length*. Unfortunately, representing a random permutation over a large set is impractical. (Merely writing down a permutation requires at least $\log_2(|S|^l!) \approx |S|^l(\ln|S|^l - 1)/\ln 2$ binary digits.)

One idea would be to use not a random permutation, but instead use a random member of some family of permutations.

The Hill cipher is an example of such a family of permutations, namely the permutations described by invertible matrices. We give our alphabet R a ring structure, say like $\mathbb{Z}_{26}$. We denote a l-tuple of letters as $\mathbf{m} \in R^l$. An invertible $l \times l$ matrix $\mathbf{K}$ acts upon such l-tuples in the obvious fashion, and we denote this action by $\mathbf{Km}$.

The plaintext m is a sequence of l-tuples of letters $\mathbf{m}_1\mathbf{m}_2\dots\mathbf{m}_L$. The key is an invertible $l \times l$ matrix $\mathbf{K}$. We encrypt the message using the formula

$$\mathbf{c}_i = \mathbf{K}\mathbf{m}_i, \qquad 1 \le i \le L.$$

The ciphertext c is the sequence of l-tuples $\mathbf{c}_1\mathbf{c}_2\dots\mathbf{c}_L$.

To decrypt a ciphertext $c = \mathbf{c}_1\dots\mathbf{c}_L$, we compute the ith plaintext tuple using the formula

$$\mathbf{m}_i = \mathbf{K}^{-1}\mathbf{c}_i, \qquad 1 \le i \le L.$$

E *Exercise* 16. The above is an informal description. Write down carefully what the three sets $\mathcal{K}$, $\mathcal{P}$, $\mathcal{C}$ are, and implement the two algorithms $\mathcal{E}$ and $\mathcal{D}$. Show that $(\mathcal{K}, \mathcal{P}, \mathcal{C}, \mathcal{E}, \mathcal{D})$ is a symmetric cryptosystem.

E *Exercise* 17. How many different keys are there for the Hill cipher with block length 2 when the alphabet has 29 elements?

E *Exercise* 18. How many blocks of ciphertext-plaintext correspondences do you need to recover $\mathbf{K}$ with reasonable probability, when the block length is 2 and the alphabet has 29 elements? (For the purposes of this exercise only, you may assume that the known plaintext consists of random letters from the alphabet.)

E *Example* 2. If we want to encrypt ABABOONISFUNNY using Hill cipher encryption with the key $\mathbf{K} = \begin{pmatrix} \text{B} & \text{D} \\ \text{A} & \text{F} \end{pmatrix}$, we get the following computations:

AB	AB	OO	NI	SF	UN	NY
↓	↓	↓	↓	↓	↓	↓
K·	**K**·	**K**·	**K**·	**K**·	**K**·	**K**·
↓	↓	↓	↓	↓	↓	↓
DF	DF	ES	LO	HZ	HN	HQ

The ciphertext is then DFDFESLOHZHNHQ. Observe the repetition in the first two plaintext blocks reflecting in the ciphertext, while the final three blocks all start with H, which is not correlated to any particular plaintext feature.

E *Exercise (for groups)* 19. Choose a key for the Hill cipher and encrypt some message. Give the ciphertext along with some known plaintext to someone else in the group and have them decrypt it without knowing the key.

A second example of a family of permutations is the Pohlig-Hellman exponentiation cipher. This time, we do not consider tuples of letters, but rather a very large alphabet. We give our large alphabet G the structure of a cyclic group of order n.

The plaintext m is a sequence of group elements $m_1m_2\dots m_L$. The key is an integer k between 0 and n that is relatively prime to n. We encrypt the message using the formula

$$c_i = km_i, \qquad 1 \le i \le L.$$

The ciphertext c is the sequence of group elements $c_1c_2\dots c_L$.

To decrypt a ciphertext $c = c_1c_2\dots c_L$, we compute the i plaintext tuple using the formula

$$m_i = k^{-1}c_i, \qquad 1 \le i \le L,$$

where the inverse k^{-1} of k is computed modulo n.

Note that the expression km_i where k is an integer and m_i is a group element is very different from the expression km_i when k and m_i are elements in a ring. The former notation is short for $m_i + m_i + \cdots + m_i$, where the sum contains k terms.

E *Exercise* 20. The above is an informal description. Write down carefully what the three sets $\mathcal{K}$, $\mathcal{P}$, $\mathcal{C}$ are, and implement the two algorithms $\mathcal{E}$ and $\mathcal{D}$. Show that $(\mathcal{K}, \mathcal{P}, \mathcal{C}, \mathcal{E}, \mathcal{D})$ is a symmetric cryptosystem.

It is generally believed that if the group G is carefully chosen, it is hard to find the key, even with known or chosen plaintext.

3.4.1 Attack: Distinguishing

The permutations used in the Hill cipher (linear, invertible maps) are very different from most permutations. For any two l-tuples $\mathbf{m}, \mathbf{m}'$ of letters from the alphabet, an invertible matrix $\mathbf{K}$ satisfies the equation

$$\mathbf{Km} + \mathbf{Km}' = \mathbf{K}(\mathbf{m} + \mathbf{m}').$$

The same observation holds for the permutations used in the Pohlig-Hellman cipher. For any two elements $m, m' \in G$, we get that

$$km + km' = k(m + m').$$

Most permutations would not satisfy these equations. This means that the permutations do not look like randomly chosen permutations. It is easy to *distinguish* the Hill cipher and Pohlig-Hellman cipher permutations from random permutations.

We can use this property to make simple deductions about plaintext based only on ciphertext properties.

E *Exercise* 21. Consider the Hill cipher. Suppose c, c' and c'' are ciphertexts such that $\mathbf{c}_i + \mathbf{c}'_i = \mathbf{c}''_i$ for one or more indexes i. What can you say about the corresponding plaintexts?

3.5 Block Ciphers

We shall now work with a set S, typically the set of l-tuples of letters from our alphabet.

D **Definition 2.** A *block cipher* is a pair of maps $\pi, \pi^{-1} : \mathcal{K} \times S \to S$ such that for all $k \in \mathcal{K}$ and $s \in S$ we have that

$$\pi(k, \pi^{-1}(k, s)) = s \text{ and } \pi^{-1}(k, \pi(k, s)) = s.$$

In other words, a block cipher is a family of permutations on a set S indexed by a key set $\mathcal{K}$.

E *Exercise* 22. The Hill cipher is based on a block cipher. Identify the block cipher by explaining what the key set $\mathcal{K}$, the set S and the functions π, π^{-1} are.

E *Exercise* 23. The Pohlig-Hellman cipher is based on a block cipher. Identify the block cipher by explaining what the key set $\mathcal{K}$, the set S and the functions π, π^{-1} are.

Despite the name, a block cipher by itself is not a cryptosystem. But we construct a cryptosystem based on a block cipher.

The plaintext m is a sequence of elements $m_1 m_2 \dots m_L$ from the set S. The key is an element k in $\mathcal{K}$. We encrypt the message elementwise using the formula

$$c_i = \pi(k, m_i), \qquad 1 \leq i \leq L.$$

The ciphertext c is the sequence of set elements $c_1 c_2 \dots c_L$.

To decrypt a ciphertext $c = c_1 \dots c_L$, we compute the ith plaintext element using the formula

$$m_i = \pi^{-1}(k, c_i), \qquad 1 \leq i \leq L.$$

E *Exercise* 24. The above is an informal description of a block cipher used in *electronic code book (ECB) mode*. Write down carefully what the three sets $\mathcal{K}$, $\mathcal{P}$, $\mathcal{C}$ are, and implement the two algorithms $\mathcal{E}$ and $\mathcal{D}$. Use the block cipher from Exercise 16. Show that $(\mathcal{K}, \mathcal{P}, \mathcal{C}, \mathcal{E}, \mathcal{D})$ is a symmetric cryptosystem.

Our hope is that this scheme will be both practical and as good as the substitution cipher when used on l-tuples. But as we have seen, the Hill cipher is easy to attack.

> **Informally:** A block cipher is *secure* if it is hard to distinguish a pair of randomly chosen inverse permutations (π, π^{-1}) from the pair of inverse permutations $(\pi(k, \cdot), \pi^{-1}(k, \cdot))$, where k has been chosen uniformly at random from $\mathcal{K}$.
>
> Note that whoever is trying to distinguish is only allowed to see the functions evaluated at various points. He is never allowed to see the permutation π or the key k.

The idea is that if it is hard to distinguish the block cipher's permutations from "average" permutations, we may as well use the block cipher with a random key instead of a random permutation. If there is an attack on the block cipher cryptosystem that does not work on the substitution cipher, that will be one way to distinguish the block cipher.

We note that for a block cipher to be secure, the key set must be very large. Otherwise, one can recognize the block cipher permutations with high probability by enumerating all the keys and observing how the corresponding permutation affects one or two elements of the set.

3.5.1 Sketch: Feistel Ciphers

How to construct secure block ciphers is out of scope of this note. However, we shall very briefly discuss one popular design for block ciphers, the *Feistel cipher*.

Block ciphers are typically built by repeatedly applying one or more simple block ciphers called *rounds*. A single round will be very easy to break, but the composition

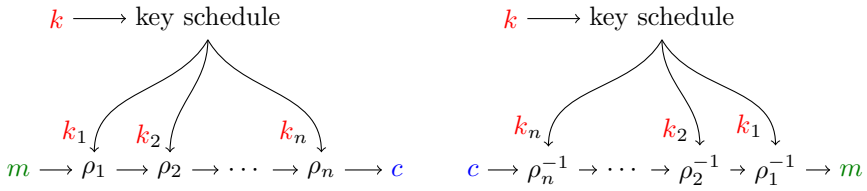

Figure 1: Typical high-level block cipher design.

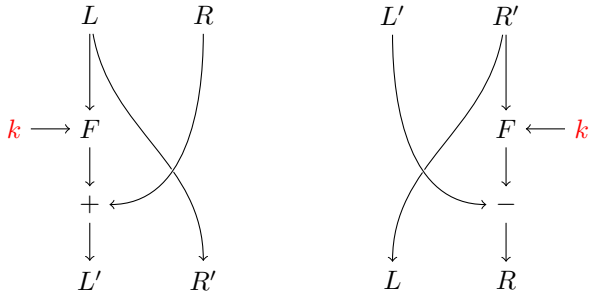

Figure 2: The Feistel round and its inverse.

of sufficiently many rounds may be hard to break. Given the rounds $\rho_1, \rho_2, \ldots, \rho_n$ with corresponding inverse rounds, we get a block cipher π by composition:

$$\pi(k, m) = \rho_n(k_n, \cdots \rho_2(k_2, \rho_1(k_1, m)) \cdots) \qquad \text{and}$$
$$\pi^{-1}(k, m) = \rho_1^{-1}(k, \cdots \rho_{n-1}^{-1}(k_{n-1}, \rho_n^{-1}(k_n, m)) \cdots)$$

The *round keys* $k_1, k_2, \ldots, k_n$ are derived from the key k. Using the same key for each round is problematic. Using independent keys such that $k = (k_1, k_2, \ldots, k_n)$ leads to impractically large keys. Instead, a *key schedule* is usually used to derive round keys from the block cipher key. This high-level design is shown in Figure 1. We note that for good ciphers, the key schedule and the rounds are highly dependent on eachother.

One convenient way to design rounds is the *Feistel round*. The construction assumes that $S = G \times G$ for some finite group G. It uses a *round function* $F : \mathcal{K} \times G \to G$ to construct the the permutation

$$\rho(k, (L, R)) = (R, L + F(k, R)).$$

It is easy to see that the inverse permutation is

$$\rho^{-1}(k, (L', R')) = (R' - F(k, L'), L').$$

Choosing a suitable round function is a hard problem, especially when the goal is to find a round function that can be computed very quickly and that does not require many rounds. Again, this problem is out of scope for this note.

3.5.2 Practical: Padding

The plaintext set for the cryptosystem from Exercise 24 is the set of finite sequences of elements from S, where each set element is typically an l-tuple of letters from the alphabet. In other words, the plaintext set is the set of letter sequences whose length is divisible by l.

But when l is large, it is unreasonable to expect message lengths to be a multiple of l. We usually need to encrypt arbitrary sequences of letters. Since we need to decrypt correctly, we cannot just append some fixed letter until the sequence length is a multiple of l.

We extend a cryptosystem to accept sequences of any length by constructing a suitable injective function, a so-called *padding scheme*.

D **Definition 3.** Let $\mathcal{P}$ and $\mathcal{P}'$ be sets. A *padding scheme for $\mathcal{P}$ and $\mathcal{P}'$* consists of two functions $\iota : \mathcal{P} \to \mathcal{P}'$ and $\lambda : \mathcal{P}' \to \mathcal{P} \cup \{\perp\}$ satisfying

$$\lambda(\iota(m)) = m \text{ for all } m \in \mathcal{P}.$$

E *Exercise* 25. Suppose you have a cryptosystem $(\mathcal{K}, \mathcal{P}', \mathcal{C}, \mathcal{E}', \mathcal{D}')$ and a padding scheme (ι, λ) for $\mathcal{P}$ and $\mathcal{P}'$. Based on the padding scheme and the cryptosystem, construct a new cryptosystem $(\mathcal{K}, \mathcal{P}, \mathcal{C}, \mathcal{E}, \mathcal{D})$. Show that it is indeed a cryptosystem.

Typically, the alphabet will be $\{0, 1\}$ and our set is $S = \{0, 1\}^l$, bit strings of length l. The plaintext set $\mathcal{P}'$ will then be bit strings of length divisible by l.

One padding scheme is the following: We first add one 1-bit, then we add 0-bits until the total length is divisible by l. If the block size l is 8, the bit string 10 10 1 will become 10 10 11 00. If the block size l is 5, the bit string 01 01 0 becomes 01 01 01 00 00.

To remove the padding, we remove up to $l-1$ trailing 0-bits and exactly one 1-bit. If the block size l is 8, the string 01 01 01 01 01 becomes the bit string 01 01 01 01 0. If the block size l is 5, the string 01 01 0 becomes 01 0, while the string 10 10 10 00 00 cannot be decoded and therefore becomes $\perp$.

If decoding fails as in the last example, we have a *padding error*. Padding errors have subtle effects on the security of cryptographic protocols.

3.5.3 Attack: Block Repetitions

We make two observations about ECB mode (the cryptosystem from Exercise 24):

- Any repetition among plaintext blocks will cause corresponding repetitions among ciphertext blocks.
- If we encrypt the same message twice, we get the same ciphertext.

Both of these observations allow an eavesdropper to learn something about the message from the ciphertext, and thereby break confidentiality. Both observations are independent of which block cipher we use. The problem is not with the concept of block cipher, but with how we use the block cipher.

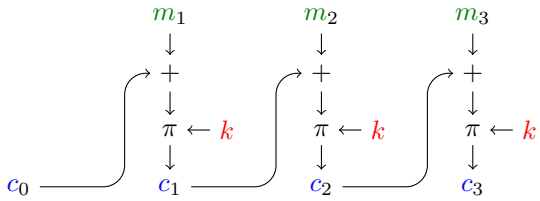

Figure 3: Cipherblock chaining (CBC) mode encryption diagram using the block cipher (π, π^{-1}). To get the decryption diagram, reverse the direction of the vertical arrows, replace $+$ by $-$ and π by π^{-1}.

E *Exercise* 26. Suppose a message has been encrypted with ECB mode using a block cipher with block length 4, and that you know that the message is either

SELL THE HOUSE NOW DO NOT SELL THE CABIN

or

SELL THE HOUSE AND EVERYTHING ELSE NOW.

(Ignore spaces and punctuation.) Explain how you can decide which message is the decryption by only looking at the ciphertext.

3.6 A Correct Use of a Block Cipher

We now allow our block cipher to operate on a group G instead of a set. (Note that our permutations will still act on the set of group elements. Since most permutations on G do not respect the group operation, neither should the block cipher permutations.)

The plaintext m is a sequence of elements $m_1 m_2 \dots m_L$ from the group G. The key is an element k in $\mathcal{K}$. We encrypt the message elementwise by first choosing a random group element c_0 and then using the formula

$$c_i = \pi(k, m_i + c_{i-1}), \qquad 1 \le i \le L.$$

The ciphertext c is the sequence of set elements $c_0 c_1 c_2 \dots c_L$.

To decrypt a ciphertext $c = c_0 c_1 \dots c_L$, we compute the ith plaintext element using the formula

$$m_i = \pi^{-1}(k, c_i) - c_{i-1}, \qquad 1 \le i \le L.$$

E *Exercise* 27. The above is an informal description of a block cipher used in *cipherblock chaining (CBC) mode*. Write down carefully what the three sets $\mathcal{K}$, $\mathcal{P}$, $\mathcal{C}$ are, and implement the two algorithms $\mathcal{E}$ and $\mathcal{D}$. Show that $(\mathcal{K}, \mathcal{P}, \mathcal{C}, \mathcal{E}, \mathcal{D})$ is a symmetric cryptosystem.

What happens when we encrypt is that we start at a random group element. This random element is added to the first message block, which is then permuted, resulting in

essentially a random-looking group element. This element is added to the second message block, which is again permuted, resulting in essentially a second random-looking group element. This process continues, producing a ciphertext that consists of a sequence of random-looking group elements.

Remark. The initial random group element c_0 is often called an *initialization vector*.

It is possible to prove a precise variant of the following statement. Its proof is out of scope for this note.

> **Informally:** *A secure block cipher used in CBC mode provides confidentiality against eavesdroppers.*

E *Exercise* 28. Consider the attacks discussed in previous sections. Explain why they fail against a secure block cipher used in CBC mode.

E *Exercise* 29. CBC mode is insecure if the initialization vector c_0 is predictable. Suppose the group G is $\mathbb{Z}^+_{2^{128}}$. We play the following game.

1. You get an encryption $c^* = c_0^* c_1^*$ of a known plaintext $m^* \in \mathbb{Z}^+_{2^{128}}$.
2. You choose $m \in \mathbb{Z}^+_{2^{128}}$.
3. You are given an encryption c of $m + \Delta$, $\Delta \in \{0, 1\}$, where the initial random element c_0 was not chosen at random, but rather as c_1^*. That is, $c = c_0 c_1$ with $c_0 = c_1^*$.

Show how you can play this game and be able to determine Δ by choosing m carefully.

3.7 Vigenère Cipher

Frequency analysis worked very well against the substitution cipher from Section 3.3. One approach to preventing frequency analysis might be to encrypt different plaintext letters with different substitution ciphers.

The idea is that the frequencies produced by the encryption will be the average of the frequencies produced by the different substitution ciphers, which should tend towards a uniform distribution, thereby preventing frequency analysis.

Again, we let our alphabet be a group G, such as $\mathbb{Z}^+_{26}$.

The plaintext m is a sequence of elements $m_1 m_2 \dots m_L$ from the group G. The key is a sequence of elements $k_1, k_2, k_3, \dots, k_l$ from G. We encrypt the message elementwise with the formula

$$c_i = m_i + k_j, \text{ where } 1 \le i \le L,\ 1 \le j \le l \text{ and } j \equiv i \pmod{l}.$$

The ciphertext c is the sequence of elements $c_1 c_2 \dots c_L$.

To decrypt a ciphertext $c = c_1 c_2 \dots c_L$, we compute the ith plaintext element using the formula

$$m_i = c_i - k_j, \text{ where } 1 \le i \le L,\ 1 \le j \le l \text{ and } j \equiv i \pmod{l}.$$

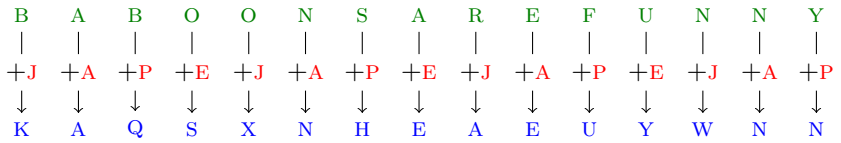

Figure 4: Example of Vigenère cipher encryption with the key $k = \text{JAPE}$.

E *Exercise* 30. The above is an informal description. Write down carefully what the three sets $\mathcal{K}$, $\mathcal{P}$, $\mathcal{C}$ are, and implement the two algorithms $\mathcal{E}$ and $\mathcal{D}$. Show that $(\mathcal{K}, \mathcal{P}, \mathcal{C}, \mathcal{E}, \mathcal{D})$ is a symmetric cryptosystem.

E *Exercise (for groups)* 31. Choose a key for the Vigenère cipher and encrypt some message. Give the ciphertext along with sufficient known plaintext to someone else in the group and have them decrypt it without knowing the key.

3.7.1 Attack: Frequency Analysis II

There is an easy attack against the Vigenère cipher if we have known plaintext. If we subtract the known plaintext from the corresponding ciphertext, we get the key repeated over and over. However, there are stronger attacks based on frequency analysis.

We begin with an English text and create a subsequence of letters by starting at the ith letter and then adding every lth letter. It so happens that such subsequences tend to have the same frequency distribution as the entire text.

E *Exercise* 32. In Exercise 13, you gathered a collection of English texts. Extract subsequences as above and compute the frequency distributions. Compare these distributions to the frequency distribution of the whole text, and estimate how long subsequences must be before we can expect to identify with reasonable certainty (a) E, (b) the five most frequent letters, and (c) the ten most frequent letters.

What happens to such subsequences when we encrypt the text with the Vigenère cipher using a key of length l? The letters in the subsequence are encrypted by adding the same letter from the key to it. In other words, the subsequence is encrypted using a shift cipher.

Earlier, we attacked the shift cipher by exhaustive search, but recognizing subsequences of English text is more difficult than recognizing English text. A better approach is to use frequency analysis. We know that E will likely be the most common letter in the plaintext subsequence, which corresponds to the most common letter in the ciphertext subsequence.

To recover a key of length l, all we have to do is run l frequency analysis attacks against the shift cipher.

E *Exercise (for groups)* 33. Choose a key for the Vigenère cipher and encrypt some sufficiently long message. Give the ciphertext along with key length to someone else in the group and have them decrypt it without knowing the key.

There is one minor problem: The key length may vary and we do not know it. The simplest approach is to try every possible length, beginning with $l = 1$. When we have the wrong key length, the attack will fail to produce a sensible decryption.

If everything has to be done by hand, there are faster ways to determine the key length. The oldest method relies on repetitions in the plaintext affecting the ciphertext. A newer method uses the so-called *index of coincidence*.

E *Exercise (for groups)* 34. Choose a key for the Vigenère cipher and encrypt some sufficiently long message. Give the ciphertext to someone else in the group and have them use the index of coincidence to determine the key length.

3.8 One Time Pad

Suppose the key for the Vigenère cipher is completely random, that is, each key letter is sampled from the uniform distribution and each letter is independent of the other letters. What happens if the key is at least as long as the message and used for only one message? When used like this, the cipher is known as the *one time pad*.

Again, our alphabet is a group G, such as $\mathbb{Z}_{26}^{+}$.

The (single) plaintext m is a sequence of L group elements $m_1 m_2 \dots m_L \in G^L$. The key k is a sequence of L group elements $k_1 k_2 \dots k_L \in G^L$. We encrypt the message elementwise using the formula

$$c_i = m_i + k_i, \text{ where } 1 \leq i \leq L.$$

The ciphertext c is the sequence of L group elements $c_1 c_2 \dots c_L$.

To decrypt a ciphertext $c = c_1 c_2 \dots c_L$, we compute the ith plaintext element using the formula

$$m_i = c_i - k_i, \text{ where } 1 \leq i \leq L.$$

E *Exercise* 35. The above is an informal description. Write down carefully what the three sets $\mathcal{K}$, $\mathcal{P}$, $\mathcal{C}$ are, and implement the two algorithms $\mathcal{E}$ and $\mathcal{D}$. Show that $(\mathcal{K}, \mathcal{P}, \mathcal{C}, \mathcal{E}, \mathcal{D})$ is a symmetric cryptosystem.

We shall prove that Eve's knowledge about the message after she saw the ciphertext, is the same as the knowledge she had before she saw the ciphertext.

Even before Alice has sent her ciphertext, Eve has some information about what Alice's plaintext will be. We can model this information as a random variable M, which means that from Eve's point of view, we assign a probability for Alice's plaintext being a specific message m. We denote this probability by $\Pr[M = m]$.

Likewise, we model Eve's information about the ciphertext as a random variable C. Again, from Eve's point of view, we can now assign a probability to the likelihood of seeing a given ciphertext. We denote that probability by $\Pr[C = c]$. We can also consider, from Eve's point of view, the conditional probability of seeing a particular ciphertext, given a particular plaintext. We denote that conditional probability by $\Pr[C = c \mid M = m]$.

Once Eve has seen the ciphertext, she has perfect knowledge of it, so there is no longer any uncertainty. Now, however, her information about the message may have changed. We denote the probability of Alice's message being a specific message conditioned on the observed ciphertext by $\Pr[M = m \mid C = c]$.

T **Theorem 1.** *If the above scheme has been used to encrypt exactly one message, then*

$$\Pr[M = m_1 m_2 \dots m_L \mid C = c_1 c_2 \dots c_L] = \Pr[M = m_1 m_2 \dots m_L].$$

Proof. The assumption on the key means that from Eve's point of view before seeing the ciphertext, the key letters are uniformly and independently distributed, which is expressed as the statement

$$\Pr[K = k_1 k_2 \dots k_L] = |G|^{-L}.$$

Again, K is a random variable describing Eve's knowledge about the key.

We first compute the probabilities

$$\begin{aligned}\Pr[C = c_1 c_2 \dots c_L \mid M = m_1 m_2 \dots m_L] \\ = \Pr[K = (c_1 - m_1)(c_2 - m_2) \dots (c_L - m_L)] = |G|^{-L}\end{aligned}$$

and

$$\begin{aligned}\Pr[C = c] &= \sum_m \Pr[C = c \mid M = m] \Pr[M = m] \\ &= |G|^{-L} \sum_m \Pr[M = m] = |G|^{-L}.\end{aligned}$$

Then we compute the a posteriori probability

$$\begin{aligned}\Pr[M = m \mid C = c] &= \frac{\Pr[M = m \wedge C = c]}{\Pr[C = c]} \\ &= \frac{\Pr[C = c \mid M = m] \Pr[M = m]}{\Pr[C = c]} \\ &= \Pr[M = m],\end{aligned}$$

which completes the proof. □

The a posteriori probability is the same as the a priori probability, which means that Eve has learned nothing new by observing the ciphertext. We have proved that the one time pad provides confidentiality against eavesdroppers.

> **Informally:** *If the key is used to encrypt only once, the one time pad provides confidentiality against eavesdroppers.*

Note that this claim is *unconditional*, unlike the corresponding claim for CBC mode in Section 3.6, which is conditional on the use of a secure block cipher. This is good. Unfortunately, we must note that the one time pad is impractical in almost every application.

E *Exercise (for groups)* 36. The one time pad is not secure if the key is used more than once. Choose a key of sufficient length for the one time pad and encrypt two different messages using the same key. Give the ciphertext along with part of one message to someone else in the group and have them decrypt it without knowing the key. To ease decryption, the messages should consist of mostly long words.

3.9 Stream Ciphers

There are many possible definitions of stream ciphers, but we shall consider only the notion of *synchronious* or *additive* stream ciphers. The idea is that a *key stream generator* expands a key and an *initialization vector* into something that looks like a key for the one time pad, which is then used to encrypt the message.

D **Definition 4.** A *key stream generator* is a function $f : \mathcal{K} \times \mathcal{I} \to G^N$.

> **Informally:** A key stream generator is *secure* if it is hard to distinguish the function values $f(k, iv_1), f(k, iv_2), \dots, f(k, iv_n)$ from random values when k has been chosen uniformly at random from $\mathcal{K}$ and the values $iv_1, iv_2, \dots, iv_n$ have been chosen uniformly at random from $\mathcal{I}$.

Again, our alphabet is a group G, such as $\mathbb{Z}_{26}^+$.

The plaintext m is a sequence of group elements $m_1 m_2 \dots m_L$ of length $L \leq N$. The key k is an element in $\mathcal{K}$. We encrypt the message by first choosing iv uniformly at random from $\mathcal{I}$, then computing the L first elements $z_1 z_2 \dots z_L$ of $f(k, iv) = z_1 z_2 \dots z_N$. We encrypt the plaintext elementwise using the formula

$$w_i = m_i + z_i, \text{ where } 1 \leq i \leq L.$$

The ciphertext c is the pair $(iv, w_1 w_2 \dots w_L)$.

To decrypt a ciphertext $c = (iv, w_1 w_2 \dots w_L)$, we first compute the L first elements $z_1 z_2 \dots z_L$ of $f(k, iv)$ and then compute the ith plaintext element using the formula

$$m_i = w_i - z_i, \text{ where } 1 \leq i \leq L.$$

E *Exercise* 37. The above is an informal description of a stream cipher based on a key stream generator. Write down carefully what the three sets $\mathcal{K}, \mathcal{P}, \mathcal{C}$ are, and implement the two algorithms $\mathcal{E}$ and $\mathcal{D}$. Show that $(\mathcal{K}, \mathcal{P}, \mathcal{C}, \mathcal{E}, \mathcal{D})$ is a symmetric cryptosystem.

It is possible to prove a precise variant of the following statement. Its proof is out of scope for this note.

> **Informally:** *A stream cipher using a secure key stream generator provides confidentiality against eavesdroppers.*

3.9.1 Key Stream Generators from Block Ciphers

Let $\pi, \pi^{-1} : \mathcal{K} \times G \to G$ be a block cipher. Suppose the set $\mathcal{I} \times \{1, 2, \ldots, N\}$ is a subset of the set of group elements of G. We shall use this block cipher to construct two key stream generators, $f_{\mathrm{OFB}/\pi} : \mathcal{K} \times G \to G^N$ and $f_{\mathrm{CTR}/\pi} : \mathcal{K} \times \mathcal{I} \to G^N$.

For any $iv \in G$ and $k \in \mathcal{K}$, let

$$z_1 = \pi(k, iv) \text{ and } z_i = \pi(k, z_{i-1}), \text{ where } 2 \leq i \leq N.$$

Then $f_{\mathrm{OFB}/\pi}(k, iv) = z_1 z_2 \ldots z_N$.

For any $iv \in \mathcal{I}$ and $k \in \mathcal{K}$, let

$$z_i = \pi(k, (iv, i)), \text{ where } 1 \leq i \leq N.$$

Then $f_{\mathrm{CTR}/\pi}(k, iv) = z_1 z_2 \ldots z_N$.

It is possible to prove a precise variant of the following statement. Its proof is out of scope for this note.

> **Informally:** *Output feedback mode $f_{\mathrm{OFB}/\pi}$ and counter mode $f_{\mathrm{CTR}/\pi}$ using a secure block cipher are secure key stream generators.*

From a practical point of view, counter mode is very easy to parallelize and can therefore be made very fast. Output feedback mode is inherently unparallelizable.

4 Integrity

In this section we shall consider the situation where Eve controls the communications channel between Alice and Bob. Eve's goal is to tamper with the messages sent by Alice so that Bob receives a different message, *without noticing*. For the moment, we shall not care about confidentiality.

The main tool we shall use is the *message authentication code*, where Alice adds an *authentication tag* to her message that allows Bob the verify that the message is unchanged.

D **Definition 5.** A *message authentication code* is a function $\mu : \mathcal{K} \times \mathcal{P} \to \mathcal{T}$.

> **Informally:** A message authentication code is *secure* if it is hard to guess the function value $\mu(k, m)$ for any m, even after seeing the values $\mu(k, m_1), \mu(k, m_2), \ldots, \mu(k, m_n)$ for any $m_1, m_2, \ldots, m_n \in \mathcal{P}$. Here, k has been chosen uniformly at random from $\mathcal{K}$, and $m \neq m_i$ for $i = 1, 2, \ldots, n$.

When Alice wants to send a message m to Bob, she sends the pair $(m, \mu(k, m))$. When Bob receives the pair (m', t), he checks that $\mu(k, m') = t$. If it is, he accepts that the message came from Alice. Otherwise, he discards the message.

> **Informally:** *A secure message authentication code used as above provides integrity.*

Note that nothing prevents Eve from *replaying* old messages by sending them to Bob. Defending against such attacks is out of scope for this note.

One of the most popular MAC constructions – HMAC – is based on so-called *hash functions*, which are out of scope for this note. We shall discuss two constructions of message authentication codes.

4.1 Polynomial evaluation MACs

We begin our discussion with a *one-time* polynomial-evaluation MAC. This MAC is insecure if it is used on more than one message. Such a MAC is impractical, so we shall also discuss how a block cipher can be used to make a more practical variant.

Our alphabet is a finite field $\mathbb{F}$, say a field with a prime p number of elements.

The plaintext m is a sequence of field elements $m_1 m_2 \ldots m_L$. The key (k_1, k_2) is a pair of field elements. The function μ_{OTPE} is computed as

$$\mu_{\mathrm{OTPE}}(k_1, k_2, m) = k_2 + \sum_{i=1}^{L} m_i k_1^i + k_1^{L+1}. \tag{5}$$

E *Exercise* 38. The above is an informal description. Write down carefully what the three sets $\mathcal{K}$, $\mathcal{P}$, $\mathcal{T}$ are, and implement an algorithm computing the function μ_{OTPE}. Show that μ_{OTPE} is a message authentication code.

We shall first prove the following statement.

T **Theorem 2.** *Let m, m' be two messages of length at most L. Let $t, t' \in \mathbb{F}$. The probability that $\mu_{\mathrm{OTPE}}(k_1, k_2, m') = t'$ given that $\mu_{\mathrm{OTPE}}(k_1, k_2, m) = t$ is at most $(L + 1)/p$.*

Proof. For simplicity, we shall assume that both messages have length exactly L. We want to compute the probability

$$\Pr[\mu_{\mathrm{OTPE}}(k_1, k_2, m') = t' \mid \mu_{\mathrm{OTPE}}(k_1, k_2, m) = t].$$

We can do that by computing how many pairs (k_1, k_2) satisfy

$$t = k_2 + k_1^{L+1} + \sum_{i=1}^{L} m_i k_1^i \tag{6}$$

and how many also satisfy

$$t' = k_2 + k_1^{L+1} + \sum_{i=1}^{L} m'_i k_1^i. \tag{7}$$

It is clear that for every value of k_1, there is exactly one value of k_2 that satisfies (6), so there are exactly p pairs that we need to consider.

Combining the two equations, we get that k_1 must satisfy the equation

$$t' - t = \sum_{i=1}^{L} (m'_i - m_i) k_1^i.$$

A solution to this equation is a zero of a polynomial equation of degree at most L, which means that there are at most L solutions.

The conclusion is that out of p possible keys (k_1, k_2) satisfying (6), there are at most L pairs that also satisfy (7). It now follows that we have a bound on the probability.

When the messages have different length, the exact same argument applies, but the polynomial we consider has degree at most $L + 1$. □

Note that this means that when Alice sends a single message m to Bob with the tag $t = \mu_{\text{OTPE}}(k_1, k_2, m)$, and Bob receives the message $m' \neq m$ and the tag t', the probability that Bob accepts that message as coming from Alice is at most $(L + 1)/p$. This proves the following claim.

> **Informally:** *If the key is used to create a MAC tag only once, the one-time polynomial evaluation MAC is a secure message authentication code.*

E *Exercise* 39. Show that the scheme is not secure if we

1. replace k_1^i with k_1^{i-1} in the sum in (5);
2. remove the term k_2 from (5); or
3. remove the term k_1^{L+1} from (5).

The above construction is just a one-time MAC, which is usually impractical. However, if we use a block cipher $\pi, \pi^{-1} : \mathcal{K}' \times S \to S$ with $\mathbb{F} \subseteq S$, we can construct an alternative polynomial-evaluation MAC that can be used more than once, using

$$\mu_{\text{PE}/\pi}(k_1, k_2, m) = \pi(k_2, k_1^{L+1} + \sum_{i=1}^{L} m_i k_1^i).$$

E *Exercise* 40. The above is an informal description. Write down carefully what the three sets $\mathcal{K}$, $\mathcal{P}$, $\mathcal{T}$ are, and implement an algorithm computing the function $\mu_{\text{PE}/\pi}$. Show that $\mu_{\text{PE}/\pi}$ is a message authentication code.

4.2 Block-cipher-based MACs

One simple MAC based on a block cipher $\pi, \pi^{-1} : \mathcal{K} \times G \to G$ is CBC-MAC, which is somewhat similar to Cipherblock Chaining mode from Section 3.6.

Fix some element $t_0 \in G$. The plaintext m is a sequence of group elements $m_1 m_2 \dots m_L$. The key k is an element of $\mathcal{K}$. Let

$$t_i = \pi(k, t_{i-1} + m_i), \text{ where } 1 \leq i \leq L.$$

Then $\mu_{\text{CBC}}(k, m) = t_L$.

E *Exercise* 41. The above is an informal description. Write down carefully what the three sets $\mathcal{K}$, $\mathcal{P}$, $\mathcal{T}$ are, and implement an algorithm computing the function μ_{CBC}. Show that μ_{CBC} is a message authentication code.

This MAC is only secure when restricted to messages of fixed length. If messages of different lengths are allowed, it is not secure.

E *Exercise* 42. Find an attack against this MAC when messages of different length are allowed.

There are many secure variants of this MAC. One variant is based on a block cipher $\pi, \pi^{-1} : \mathcal{K} \times \mathbb{F} \to \mathbb{F}$ over a field $\mathbb{F}$. The idea is to modify the final plaintext block with an unpredictable value.

Fix a non-zero element $g \in \mathbb{F}$. The plaintext m is a sequence of field elements $m_1 m_2 \dots m_L$. The key k is an element of $\mathcal{K}$. Let $t_0 = 0$, $h = \pi(k, 0)$ and

$$t_i = \pi(k, t_{i-1} + m_i), \text{ where } 1 \leq i \leq L - 1.$$

Then $\mu_{\text{CBC}'}(k, m) = \pi(k, hg + t_{L-1} + m_L)$.

> **Informally:** *The modified CBC-MAC using a secure block cipher is a secure MAC.*

5 Confidentiality and Integrity

Finally, we consider the situation where Eve controls the communications channel between Alice and Bob. Eve's goal is both to read Alice's messages to Bob and tamper with them.

We shall combine the secure cryptosystems we saw in Section 3 with the message authentication codes we saw in Section 4 into cryptosystems that provide both confidentiality and integrity.

> **Informally:** A symmetric cryptosystem provides *integrity* if it is hard to create a ciphertext that decrypts to anything other than $\perp$ without knowledge of the key.

First we consider a general principle in cryptography: *Never use the same key for two different things.* This means that we should use different keys for the cryptosystem and the MAC when we combine them. That leaves us with three obvious ways of combining a cryptosystem with a MAC:

Encrypt-then-MAC First encrypt the message, then MAC the ciphertext:

$$w = \mathcal{E}(k_e, m);\ t = \mu(k_m, w);\ c = (w, t).$$

MAC-then-encrypt MAC the message, then encrypt the message and the tag:

$$t = \mu(k_m, m);\ c = \mathcal{E}(k_e, (m, t)).$$

Encrypt-and-MAC Encrypt the message and attach a MAC tag of the message:

$$w = \mathcal{E}(k_e, m);\ t = \mu(k_m, m);\ c = (w, t).$$

In all three cases, the decryption algorithm verifies the MAC and if the verification fails, the output of the decryption algorithm is $\perp$.

Encrypt-and-MAC is in general insecure. If you encrypt the same message twice, you will always get the same tag, something that Eve will notice. Therefore, it fails confidentiality.

MAC-then-encrypt is often secure, but there are special cases where it is not secure. In particular, such schemes will often fail when combined with padding schemes.

Encrypt-then-MAC is always secure, which means that this is the best choice.

> **Informally:** *A cryptosystem that provides confidentiality against eavesdroppers and a secure message authentication code combined using Encrypt-then-MAC provides both confidentiality and integrity.*

Remark. Practice has shown that Encrypt-then-MAC seems to be hard to do right. In practice, there is also a need to protect the integrity of more than the encrypted message, so-called *associated data*, which is actually non-trivial to get right. Modern practice has therefore moved towards *authenticated encryption with associated data* (AEAD). We do not discuss this here.

Note that nothing prevents Eve from *replaying* old ciphertexts by sending them to Bob. Defending against such attacks is out of scope for this note.

Diffie-Hellman and Discrete Logarithms

KG

October 24, 2019

Contents

1 Introduction

In this note, we consider the following problem. Alice and Bob wants to establish a shared secret known only by them by communicating via some communications channel. Eve has access to the channel and she may eavesdrop on anything sent over the channel.

Establishing a shared secret is a prerequisite for using the theory of symmetric cryptography to communicate securely over insecure channels. Traditionally, this was done by meeting in person, using couriers or relying on trusted third parties. But as networks grow and the number of connections increase, the traditional approaches become impractical or introduce unpleasant trust assumptions.

The Diffie-Hellman protocol is a *cryptographic protocol* for establishing a shared secret. Conjecturally, if the underlying mathematical structure is carefully chosen, running the protocol requires Alice and Bob to do relatively little work to establish a shared value, but the eavesdropper Eve will have to do infeasibly much work to deduce the shared value. In other words, the shared value will remain a secret known (fully) only to Alice and Bob.

This note is an introduction to this protocol and the study of its security. Section 2 describes the protocol and its mathematical foundations, namely finite cyclic groups.

As shown in Section 3.1, not every finite cyclic group is suitable for use in the Diffie-Hellman protocol. Section 3 discusses various necessary requirements for a cylic group to be suitable. Two plausible families of cyclic groups based on finite fields and elliptic curves are discussed in Section 5 and 6, while Section 4 show how to distinguish prime numbers from composite numbers, something that we will need to do.

This text is intended for a reader that is familiar with mathematical language, basic algebra (groups, rings, fields and linear algebra) and elementary computer science (algorithms).

This text is sometimes 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 known only by Alice or Bob. Blue denotes information that the eavesdropper will see. Information that is assumed to be known by both Alice and Bob (as well as Eve) is not coloured.

We also 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.) is not coloured.

2 The Diffie-Hellman Protocol

The *Diffie-Hellman protocol* is a *cryptographic protocol* that allows Alice and Bob to establish a shared value. Alice is the *initiator* in the sense that she sends the first message in the protocol. Bob is the *responder* since he responds to Alice's message.

We first prove that the protocol is *complete*, in the sense that running the protocol without an adversary establishes a shared value. Next, we consider how much work Alice and Bob must do to execute the protocol.

We want to use the protocol to establish a shared *secret*, in the sense that the eavesdropper Eve should not know the shared value.

> **Informally:** A *key exchange* protocol is a two-party protocol between an *initiator* and a *responder*. The players eventually output either a *shared secret* or $\perp$.

> **Informally:** A key exchange protocol is *secure* if someone who sees the protocol messages, but does not actually participate in the protocol, cannot learn the agreed-upon shared secret.

The protocol is based on a finite cyclic group, and the study of its security turns out to involve the study of an interesting computational problem in finite cyclic groups, computing so-called *discrete logarithms*.

Before we can describe the Diffie-Hellman protocol, we must establish the underlying abstract mathematical structure. We begin with a bit of notation.

D **Definition 1.** *Exponentiation* in a group is defined as

$$x^a = \underbrace{x \cdot x \cdots x}_{a \text{ terms}}$$

for any group element x and any integer $a > 0$. Then we define x^0 to be the identity element and $x^a = (x^{-1})^{-a}$ when $a < 0$.

When using additive notation for the group operation, we get the notation

$$a \cdot x = \underbrace{x + x + \cdots + x}_{a \text{ terms}}$$

for any group element x and any integer $a > 0$, with $0 \cdot x = 0$ and $a \cdot x = (-a) \cdot (-x)$ for $a < 0$.

D **Definition 2.** A group G is *cyclic* if there exists an element $g \in G$ such that $G = \{g^a \mid a \in \mathbb{Z}\}$.

Let G be a finite cyclic group of order n, and let g be a generator. The Diffie-Hellman protocol works as follows (see also Figure 1).

Both Alice and Bob know the group G, the group order n and the generator g.

1. Alice chooses a number a uniformly at random from the set $\{0, 1, 2, \ldots, n-1\}$. She computes $x = g^a$ and sends x to Bob.
2. Bob receives x from Alice. He chooses a number b uniformly at random from the set $\{0, 1, 2, \ldots, n-1\}$, computes $y = g^b$ and $z_B = x^b$, and sends y to Alice.
3. Alice receives y from Bob. Alice computes $z_A = y^a$.

Remark. The structures $\mathbb{Z}_n$ and $\mathbb{F}_p$, which we will use for many examples, are typically constructed as factor rings. As such, their elements are really cosets and should be denoted as $k + \langle n \rangle$ or $k + \langle p \rangle$. In principle, any integer in the coset can be used to represent the coset.

We now make two remarks;

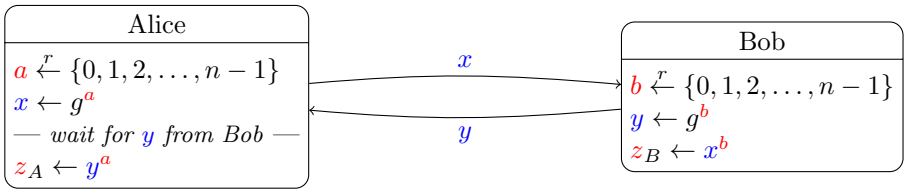

Figure 1: The Diffie-Hellman protocol.

1. In the interest of brevity, we will simply write k instead of $k + \langle n \rangle$ and $k + \langle p \rangle$ whenever the meaning is clear. So in $\mathbb{Z}_{13}^+$ we will write $3 \cdot 10 = 4$, instead of the cumbersome $(3 + \langle 13 \rangle)(10 + \langle 13 \rangle) = (4 + \langle 13 \rangle)$.

2. When Alice and Bob compute group elements, their choice of representatives for the coset is usually important, especially that they make the same choice. Typically, they choose the unique smallest non-negative integer in the coset. Other choices are sometimes convenient, in particular for intermediate results.

E *Example* 1. Alice and Bob use Diffie-Hellman with the group $G = \mathbb{Z}_{13}^+$ and generator $g = 2$. Note the additive notation.

Alice chooses $a = 3$ and computes $x = a \cdot g = 3 \cdot 2 = 6$. She sends 6 to Bob.

Bob receives 6. He chooses $b = 5$ and computes $y = b \cdot g = 5 \cdot 2 = 10$ and $z_B = b \cdot x = 5 \cdot 6 = 4$. He sends 10 to Alice.

Alice receives 10 from Bob. She computes $z_A = a \cdot y = 3 \cdot 10 = 4$.

Remark. Note that for $\mathbb{Z}_n^+$, the exponentiation coincides with ring multiplication. As we shall see, this means that $\mathbb{Z}_n^+$ cannot be used safely with Diffie-Hellman.

E *Example* 2. Alice and Bob use Diffie-Hellman with the group $G = \mathbb{F}_{13}^*$ and generator $g = 2$.

Alice chooses $a = 3$ and computes $x = g^a = 2^3 = 8$. She sends 8 to Bob.

Bob receives 8. He chooses $b = 5$ and computes $y = g^b = 2^5 = 6$ and $z_B = x^b = 8^5 = 8$. He sends 6 to Alice.

Alice receives 6 from Bob. She computes $z_A = y^a = 6^3 = 8$.

E *Exercise (for groups)* 1. Agree on a cyclic group and a generator for the group, then use the Diffie-Hellman protocol to agree on a secret.

We begin by proving completeness, that is, Alice and Bob arrive at a shared value.

T **Proposition 1.** *In the above protocol $z_A = z_B$.*

Proof. We compute that $z_A = y^a = (g^b)^a = (g^a)^b = x^b = z_B$. □

We have shown that the Diffie-Hellman protocol establishes a single shared value, which we shall denote by z.

Next, we must consider how much work Alice and Bob must do to execute the protocol. It is clear that the only non-trivial computations involved are the two *exponentiations* done by each of them. From the definition, it is clear that Alice and Bob can do their two exponentiations using less than $2n$ group operations:

$$2^5 = 2 \cdot 2^4 = 4 \cdot 2^3 = 8 \cdot 2^2 = 16 \cdot 2 = 32.$$

However, there is a better way to do these computations.

T **Proposition 2.** *For any $x \in G$ and integer $a > 0$, the group element x^a can be computed using at most $2 \log_2 a$ group operations.*

Proof. Since

$$x^{2^{i+1}} = (x^{2^i})^2,$$

we can compute the $l + 1$ group elements $x = x^{2^0}, x^{2^1}, x^{2^2}, \ldots, x^{2^l}$ using l group operations.

When these elements have been computed, we can write a in binary as

$$a = \sum_{i=0}^{l} a_i 2^i, \qquad a_i \in \{0, 1\}$$

and compute the product

$$\prod_{i=0}^{l} (x^{2^i})^{a_i} = x^{\sum_{i=0}^{l} a_i 2^i} = x^a.$$

Computing this product requires at most l group operations.

We have done at most $2l$ group operations. If $a > 0$, we may assume that $a_l = 1$, and then we have that $l \leq \log_2 a$. The claim follows. □

E *Example* 3. We can compute 2^{13} in $\mathbb{Z}$ by observing that $13 = 2^0 + 2^2 + 2^3$, computing

$$2^{2^0} = 2 \qquad 2^{2^1} = (2^{2^0})^2 = 2^2 = 4$$
$$2^{2^2} = (2^{2^1})^2 = 4^2 = 16 \qquad 2^{2^3} = (2^{2^2})^2 = 16^2 = 256$$

and

$$2^{13} = 2^{2^0 + 2^2 + 2^3} = 2^{2^0} 2^{2^2} 2^{2^3} = 2 \cdot 16 \cdot 256 = 32 \cdot 256 = 8192.$$

This required only 5 multiplications, while using the definition would require 12 multiplications.

E *Example* 4. We can compute 2^{257} in $\mathbb{F}_{19}^*$ by computing successively $2^2 = 4$, $2^{2^2} = 4^2 = 16$, $2^{2^3} = 16^2 = 9$, $2^{2^4} = 9^2 = 5$, $2^{2^5} = 5^2 = 6$, $2^{2^6} = 6^2 = 17$, $2^{2^7} = 17^2 = 15$ and $2^{2^8} = 15^2 = 16$, and finally

$$2^{257} = 2 \cdot 16 = 13.$$

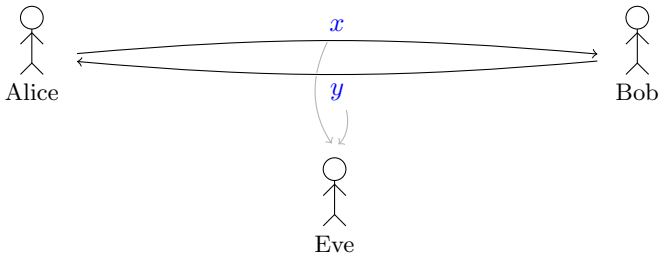

Figure 2: Eve is listening and learns x and y, but cannot see Alice' and Bob's secret values or computations. She already knows G, g and n.

Note that by choosing small coset representatives for the intermediate results, we avoid computing with large integers.

E *Exercise* 2. The proof of Proposition 2 essentially describes an algorithm for computing x^a. Implement this algorithm and restate Proposition 2 as a statement about the algorithm's time complexity (in terms of group operations, ignoring any other form of computation involved).

E *Exercise* 3. Use the algorithm from Exercise 2 to compute (by hand) 5^{582} in the cyclic group $\mathbb{F}^*_{10007}$.

E *Exercise* 4. Note that we can define x^a for $a \geq 0$ using the rules that $x^0 = 1$ and

$$x^a = \begin{cases} (x^{a/2})^2 & \text{when } a \text{ is even, and} \\ (x^{(a-1)/2})^2 x & \text{when } a \text{ is odd.} \end{cases}$$

Use this fact to give an alternative proof of Proposition 2 leading to an alternative algorithm.

We have shown that as long as group operations can be computed in reasonable time, Alice and Bob can execute the Diffie-Hellman protocol in reasonable time, even when the group order is very large.

3 Discrete Logarithms

Our goal is for Alice and Bob to establish a shared *secret*. We must consider what the eavesdropper Eve can do to learn the established shared value.

The Diffie-Hellman protocol is supposed to work even when Alice and Bob have had no previous communication. We must therefore assume that Eve knows both the group G, the generator g and the group order n. When Alice and Bob run the protocol, Eve additionally learns x and y. She wants to know z.

D **Definition 3.** The *Diffie-Hellman problem* in a cyclic group G of order n with generator g is to find $z = g^{ab}$ given $x = g^a$ and $y = g^b$, when a and b has been chosen independently and uniformly at random from $\{0, 1, 2, \ldots, n-1\}$.

It is more or less clear that solving the Diffie-Hellman problem is more or less the same as attacking the Diffie-Hellman key exchange protocol.

We must choose the cyclic group that Alice and Bob shall use. They want to spend as little effort as possible to establish the shared secret, both with respect to computation and communication. Which means that computing the group operation should be reasonably fast, and group elements should have a reasonably compact representation.

At the same time, Alice and Bob want the established shared value to be secret. It cannot be secret unless solving the Diffie-Hellman problem for the group Alice and Bob use requires more computational effort than Eve can manage.

It is clear that if Eve can find a or b, then she can easily compute $z = y^a = x^b$.

D **Definition 4.** The *discrete logarithm* of x to the base g is the smallest non-negative integer a such that $x = g^a$. We write $\log_g x = a$.

The *discrete logarithm problem* in a cyclic group G is to find the discrete logarithm of x to the base g, when x has been chosen uniformly at random from the group.

As we see, if Eve can compute discrete logarithms, she can easily compute the shared value established by Alice and Bob. Conversely, it is generally believed (and there is evidence to suggest that this is the case) that if Eve can compute the shared value, she will be able to compute discrete logarithms as well.

> **Informally:** *It is conjectured that solving the Diffie-Hellman problem in a group G is not (much) easier than computing discrete logarithms in G.*

Under this conjecture, the study of the security of the Diffie-Hellman protocol reduces to the study of how easy it is to compute discrete logarithms in the group we want to use.

We first begin with the observation that no choice of generator will make discrete logarithm computations harder.

E *Exercise* 5. Let x be a group element of order m and a, b be integers. Prove that $x^a = x^b$ if and only if $a \equiv b \pmod{m}$.

T **Proposition 3.** *Let g_1 and g_2 be generators, and suppose $\log_{g_1} g_2 = a$. Then $\log_{g_2} g_1 \equiv a^{-1} \pmod{n}$.*

Proof. It is given that $g_2 = g_1^a$, and since g_2 is a generator, there exists an integer b such that $g_1 = g_2^b$. We get that $g_1 = g_1^{ab}$. Exercise 5 then says that

$$ab \equiv 1 \pmod{n},$$

and our claim follows. □

The following exercise will justify that we do not much care about which base we use and often omit it from theorems.

Exercise 6. Show that if you can compute discrete logarithm with some generator g as base, then you can compute discrete logarithms with any generator as base, at roughly twice the cost.

Exercise 7. Any cyclic group G of order n is isomorphic to $\mathbb{Z}_n^+$. Let $\lambda : G \to \mathbb{Z}_n^+$ be a group isomorphism taking a generator g to $1 + \langle n \rangle$. Show that computing discrete logarithms to the base g is essentially the same as computing the group isomorphism λ.

We now begin by establishing a level of effort that will certainly be sufficient to compute discrete logarithms. This will lead to our first requirement for a group to be suitable for Diffie-Hellman.

Proposition 4. *Let G be a cyclic group of order n. The discrete logarithm of a group element $x \in G$ can be computed using less than n group operations.*

Proof. Let g be a generator. Since $g^a g = g^{a+1}$, we can compute the elements of the sequence $g^0, g^1, g^2, \ldots, g^{n-1}$ using $n-1$ group operations. Clearly, we can also keep track of the discrete logarithm of each element as we compute it, simply by counting how many group operations we have done.

Since g is a generator, x must be one of the elements in the sequence, and we can recognize it when we reach it. The claim follows. □

The algorithm for computing discrete logarithms implied by the proof of the proposition does more than just group operations, it also compares group elements and counts how many group operations it has done. But the group operations will dominate the computational effort. It therefore makes sense to focus on the number of group operations. This will be the case for all the algorithms in this section.

Exercise 8. The proof of Proposition 4 essentially describes an algorithm for computing $\log_g x$. Implement this algorithm and restate Proposition 4 as a statement about the algorithm's time complexity (in terms of group operations, ignoring everything else).

Example 5. Consider the group $G = \mathbb{F}_{13}^*$ with generator $g = 2$. We want to compute the discrete logarithm of $x = 6$.

We compute $2^0 = 1$, $2^1 = 2$, $2^2 = 2 \cdot 2 = 4$, $2^3 = 4 \cdot 2 = 8$, $2^4 = 8 \cdot 2 = 3$ and $2^5 = 3 \cdot 2 = 6$. We find that $\log_2 6 = 5$.

Exercise 9. Use the algorithm from Exercise 8 to compute (by hand) the discrete logarithm of 3972 to the base 5 in the group $\mathbb{F}_{10007}^*$.

The structure of our study of the discrete logarithm computation is to study various ways of solving or simplifying the computation. Every time we improve our ability to compute discrete logarithms in various groups, we better understand what kind of group Alice and Bob can use for Diffie-Hellman.

Based on Proposition 4, we arrive at the following requirement.

Requirement 1. If n is the group order, n group operations must be an infeasible computation.

3.1 An Unsuitable Group

It is easy to find cyclic groups with large group order. If n is any large number, then $\mathbb{Z}_n^+$ is a cyclic group of order n. A natural generator is $1 + \langle n \rangle$, but the coset of any integer relatively prime to n will do.

Note that this group is written additively. The Diffie-Hellman protocol then looks like:

1. Alice chooses a number a uniformly at random from the set $\{0, 1, 2, \ldots, n-1\}$. She computes $x = a \cdot (1 + \langle n \rangle) = a + \langle n \rangle$ and sends x to Bob.

2. Bob receives x from Alice. He chooses a number b uniformly at random from the set $\{0, 1, 2, \ldots, n-1\}$, computes $y = b \cdot (1 + \langle n \rangle)$ and $z = b \cdot x$, and sends y to Alice.

3. Alice receives y from Bob. Alice computes $z = a \cdot y$.

From the description, we see that x is essentially a, and it is immediately obvious that this group is unsuitable for Diffie-Hellman.

Exercise 10. Show that the *Extended Euclidean algorithm* can be used to compute multiplicative inverses modulo n quickly (in time roughly proportional to $\log_2 n$ integer divisions and multiplications). Implement this algorithm.

Exercise 11. Alice's x is essentially equal to a since we used $1 + \langle n \rangle$ as a generator. Use Proposition 3 and the previous exercise to show that any other generator would be equally insecure.

Exercise 12. In Example 1, Alice and Bob use the group $G = \mathbb{Z}_{13}^+$ and generator $g = 2$. Alice sends $x = 6$ to Bob, and Bob replies with $y = 10$. Use only this information, together with the result from the previous exercise, to compute Alice's secret value a, Bob's secret value b and the shared secret z.

This example shows us that a large group is necessary, but not sufficient for our purposes.

3.2 Pohlig-Hellman I

There are many ways to describe and analyse the first part of the Pohlig-Hellman algorithm for computing discrete logarithms. We shall rely on the algebraic structure of cyclic groups, and begin with the observation that computing a discrete logarithm in G is the same as computing the isomorphism from G to $\mathbb{Z}_n^+$ taking g to $1 + \langle n \rangle$. If G has a suitable group structure, we can use that to simplify the computation of the isomorphism.

Suppose for the remainder of this section that $n = n_1 n_2$ with $\gcd(n_1, n_2) = 1$. Define the two sets

$$H_1 = \{x^{n_2} \mid x \in G\} \text{ and } H_2 = \{x^{n_1} \mid x \in G\}.$$

E *Exercise* 13. Prove that the sets H_1, H_2 are subgroups of G of order n_1, n_2, respectively.

Next, define $\pi_1 : G \to H_1$, $\pi_2 : G \to H_2$ and $\pi : G \to H_1 \times H_2$ by

$$\pi_1(x) = x^{n_2}, \quad \pi_2(x) = x^{n_1} \quad \text{and} \quad \pi(x) = (\pi_1(x), \pi_2(x)).$$

E *Exercise* 14. Prove that the maps π_1, π_2 are well-defined, that they are group homomorphisms and that they are surjective.

E *Exercise* 15. Prove that the map π is a group isomorphism by giving an inverse homomorphism.

It is interesting to compare the above results with the situation for $\mathbb{Z}_n^+$. We begin by stating a version of the Chinese remainder theorem without proof.

T **Theorem 5.** *Let $n = n_1 n_2$ with $\gcd(n_1, n_2) = 1$. Then as rings*

$$\mathbb{Z}_n \simeq \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2},$$

and the unique ring isomorphism and its inverse are easy to compute.

The map $\mathbb{Z}_n^+ \to \mathbb{Z}_{n_1}^+ \times \mathbb{Z}_{n_2}^+$ corresponding to π above is given by $k + \langle n \rangle \mapsto (kn_2 + \langle n_1 \rangle, kn_1 + \langle n_2 \rangle)$. This map is a group isomorphism, but not a ring isomorphism, unlike the Chinese remainder theorem ring isomorphism which takes $k + \langle n \rangle$ to $(k + \langle n_1 \rangle, k + \langle n_2 \rangle)$.

Let $\lambda : G \to \mathbb{Z}_n^+$, $\lambda_1 : H_1 \to \mathbb{Z}_{n_1}^+$ and $\lambda_2 : H_2 \to \mathbb{Z}_{n_2}^+$ be the group isomorphisms satisfying $\lambda(g) = 1 + \langle n \rangle$, $\lambda_1(\pi_1(g)) = 1 + \langle n_1 \rangle$ and $\lambda_2(\pi_2(g)) = 1 + \langle (n_2 \rangle$. Note that λ_1 and λ_2 are carefully chosen, and correspond to discrete logarithms to the bases $\pi_1(g)$ and $\pi_2(g)$.

T **Proposition 6.** *The diagram*

$$\begin{array}{ccc} G & \xrightarrow{\quad\lambda\quad} & \mathbb{Z}_n^+ \\ {\scriptstyle\pi}\downarrow & & \uparrow{\scriptstyle CRT} \\ H_1 \times H_2 & \xrightarrow{\quad\lambda_1 \times \lambda_2\quad} & \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \end{array}$$

is commutative. (Here $\lambda_1 \times \lambda_2$ is the group isomorphism taking (x_1, x_2) to $(\lambda_1(x_1), \lambda_2(x_2))$.

Proof. It is sufficient to consider what happens to a group generator.

By definition, we have that $\lambda_1(\pi_1(g)) = 1 + \langle n_1 \rangle$ and $\lambda_2(\pi_2(g)) = 1 + \langle n_2 \rangle$.

Since $\gcd(n_1, n_2) = 1$, the Chinese remainder theorem says that $\mathbb{Z}_{n_1}^+ \times \mathbb{Z}_{n_2}^+$ is isomorphic to $\mathbb{Z}_n^+$. Moreover, the ring isomorphism given by the Chinese remainder theorem takes the ring identity $(1 + \langle n_1 \rangle, 1 + \langle n_2 \rangle)$ to the ring identity $1 + \langle n \rangle$, which concludes the proof. □

Now that we have established the suitable group structure, all we need to do is to ensure that we can use it.

T **Proposition 7.** *Let G be a cyclic group of order $n = n_1 n_2$, $\gcd(n_1, n_2) = 1$. Let H_1 and H_2 be the subgroups of order n_1 and n_2, respectively. The discrete logarithm of any group element $x \in G$ can be computed by computing one discrete logarithm in H_1 and one discrete logarithm in H_2, and also two exponentiations of x and two exponentiations of g.*

Proof. Consider the diagram from Proposition 6. Computing the maps π_1 and π_2 costs one exponentiation each, and we need to apply the maps to g and x in order to know what the λ_1 and λ_2 maps are. Computing discrete logarithms in H_1 and H_2 is the same as computing the maps λ_1 and λ_2. Computing the ring isomorphism described by the Chinese remainder theorem is cheap, so we can ignore that cost.

Computing discrete logarithms in G is equivalent to computing λ. Our claim follows by Proposition 6 and the above accounting. □

E *Exercise* 16. The proof of Proposition 7 essentially describes an algorithm for computing discrete logarithms, using a different algorithm for computing discrete logarithms in subgroups. Implement this algorithm and restate Proposition 7 as a statement about the algorithm's time complexity (in terms of group operations and computing discrete logarithms in subgroups, ignoring any other form of computation involved).

E *Example* 6. Consider the group $G = \mathbb{Z}_{72}^+$ with generator $g = 1$, and let $x = 23$. We want to compute $\log_g x$. (The answer is of course obvious, but we do not actually care about the answer, we just want to understand how the algorithm works.)

Since $n = 72 = 8 \cdot 9$, we let $n_1 = 8$ and $n_2 = 9$.

We compute $\pi_1(g) = 9 \cdot 1 = 9$ and $\pi_1(x) = 9 \cdot 23 = 63$. We can now observe that $\pi_1(x) = 63 = 7 \cdot 9 = 7 \cdot \pi_1(g)$, so $\lambda_1(\pi_1(x)) = 7$.

Next, we compute $\pi_2(g) = 8 \cdot 1 = 8$ and $\pi_2(x) = 8 \cdot 23 = 40$. Again, we observe that $\pi_1(x) = 40 = 5 \cdot 8 = 5 \cdot \pi_2(g)$, so $\lambda_2(\pi_2(x)) = 5$.

Finally, the canonical CRT map takes $(7, 5)$ to 23, as expected.

E *Exercise* 17. Observe that $72 = 8 \cdot 9$. Use the algorithm from Exercise 16 to compute (by hand) the discrete logarithm of 70 to the base 5 in the group $\mathbb{F}_{73}^*$.

An alternative, more computational approach to proving the theorem is to observe that if $\log_g x = a$ then

$$x^{n_2} = g^{an_2} = (g^{n_2})^a,$$

which by Exercise 5 means that

$$a \equiv \log_{g^{n_2}} x^{n_2} \pmod{n_1}.$$

In the same way, we get that

$$a \equiv \log_{g^{n_1}} x^{n_1} \pmod{n_2}.$$

It follows that if we can compute the discrete logarithms to the bases g^{n_1} and g^{n_2}, we can use the Chinese remainder theorem to recover the logarithm to the base g.

E *Example* 7. We redo Example 6.

We compute $n_2 \cdot g = 9 \cdot 1 = 9$ and $n_2 \cdot x = 9 \cdot 23 = 63$. We compute that $\log_9 63 = 7$, which means that $a \equiv 7 \pmod 8$.

Next, we compute $n_1 \cdot g = 8 \cdot 1 = 8$ and $n_1 \cdot x = 8 \cdot 23 = 40$. We compute that $\log_8 40 = 5$, which means that $a \equiv 5 \pmod 9$.

Finally, we use CRT to find that $a = 23$.

E *Exercise* 18. State a variant of Proposition 7 and prove it using the above computational approach.

T **Theorem 8** (Pohlig-Hellman I)**.** *Let G be a cyclic group of order $n = \prod_{i=1}^{l} \ell_i^{r_i}$, where $\ell_i \neq \ell_j$ when $i \neq j$. The discrete logarithm of a group element $x \in G$ can be computed by computing one discrete logarithm in each of the subgroups of order $\ell_i^{r_i}$, and also l exponentiations of x and l exponentiations of g.*

Proof. We apply Proposition 7 repeatedly and the theorem follows. □

E *Exercise* 19. The proof of Theorem 8 essentially describes an algorithm for computing discrete logarithms. Implement this algorithm and restate Theorem 8 as a statement about the algorithm's time complexity (in terms of group operations and computing discrete logarithms in subgroups, ignoring any other form of computation involved).

You may use the algorithm and statement from Exercise 16 as a subroutine and lemma, respectively.

Based on Proposition 4 and Theorem 8, we arrive at the following requirement.

Requirement 2. If $n = \prod_{i=1}^{l} \ell_i^{r_i}$ is the group order and $\ell_l^{r_l}$ is the largest prime power dividing n, $\ell_l^{r_l}$ group operations must be an infeasible computation.

3.3 Pohlig-Hellman II

In the previous section, we saw how computing discrete logarithms can be reduced to computing discrete logarithms in subgroups whose orders are prime powers. We shall now see how computing discrete logarithms in a group whose order is a prime power can be reduced to computing discrete logarithms in a prime-ordered subgroup.

Suppose for the remainder of this section that the group order n is a prime power ℓ^r for some prime ℓ.

Define the sets

$$H_i = \{x^{\ell^i} \mid x \in G\}, \qquad 0 \leq i \leq r.$$

Note that $H_r = \{1\}$.

E *Exercise* 20. Prove that the sets $H_0, H_1, \ldots, H_r$ are subgroups of G such that $H_0 \supsetneq H_1 \supsetneq \cdots \supsetneq H_r$. Prove also that H_{r-1} is isomorphic to $\mathbb{Z}_\ell^+$.

Now we consider the maps $\pi_i : H_i \to H_{r-1}$ defined by

$$\pi_i(y) = y^{\ell^{r-i-1}}, \qquad 0 \leq i \leq r-1.$$

E *Exercise* 21. Prove that the π_i maps are surjective group homomorphisms.

E *Exercise* 22. Show that if $y \in H_i$, then $y^{\ell^j} \in H_{i+j}$ for any $j \geq 0$. Show also that $\pi_{i+j}(y^{\ell^j}) = \pi_i(y)$.

E *Exercise* 23. Prove that the kernel of π_i is H_{i+1}.

T **Proposition 9.** *Let g be a generator for G, let $y \in H_i$ and let $a = \log_{\pi_0(g)} \pi_i(y)$. Then*

$$y g^{-\ell^i a} \in H_{i+1}.$$

Proof. Note first that $g^{\ell^i} \in H_i$, and that $\pi_i(y) = \pi_0(g)^a$. Using Exercise 22, we compute

$$\pi_i(y g^{-\ell^i a}) = \pi_i(y)\pi_i(g^{\ell^i})^{-a} = \pi_0(g)^a \pi_0(g)^{-a} = 1.$$

The claim now follows from Exercise 23. □

This suggests a recursive algorithm for computing discrete logarithms.

T **Theorem 10** (Pohlig-Hellman II)**.** *Let G be a cylic group of order $n = \ell^r$, where ℓ is prime. The discrete logarithm of a group element $x \in G$ can be computed by computing r discrete logarithms in the subgroup of order ℓ (all to the same base), and also $r-1$ exponentiations of group elements, $r-1$ group operations and r exponentiation of g.*

Proof. We construct a sequence of group elements $y_0, y_1, \ldots, y_r$ and integers $a_0, a_1, \ldots, a_{r-1}$ as follows. We begin with $y_0 = x$, and compute

$$a_i = \log_{\pi_0(g)} \pi_i(y_i) \text{ and } y_{i+1} = y_i g^{-\ell^i a_i}, \text{ for } 0 \leq i < r.$$

By Proposition 9, this sequence is well-defined and $y_r = 1$. Which means that

$$1 = y_{r-1} g^{-\ell^{r-1} a_{r-1}} = \cdots = x g^{-\sum_{i=0}^{r-1} a_i \ell^i}.$$

Note that $0 \leq a_i < \ell$ for $0 \leq i < r$, which means that $0 \leq \sum_{i=0}^{r-1} a_i \ell^i < \ell^r$. By Exercise 5, we get that

$$\log_g x = \sum_{i=0}^{r-1} a_i \ell^i,$$

Each iteration requires the computation of one discrete logarithm in the subgroup H_{r-1}, and also one evaluation of π_i (one exponentiation of a group element), one exponentiation of g and one group operation. The latter can be ignored for the final iteration, of course.

There is a total of r iterations. We also have to compute $\pi_0(g)$ (one exponentiation of g) once. □

E *Exercise* 24. The proof of Theorem 10 essentially describes an algorithm for computing discrete logarithms. Implement this algorithm and restate Theorem 10 as a statement about the algorithm's time complexity (in terms of group operations and computing discrete logarithms in subgroups, ignoring any other form of computation involved).

E *Example* 8. Consider the group $G = \mathbb{Z}_{27}^+$ with generator $g = 1$, and let $x = 23$. We want to compute $\log_g x$.

Since $n = 27 = 3^3$, we find that the subgroup H_1 is all the multiples of 3, while $H_2 = \{0, 9, 18\}$ is the multiples of $9 = 3^2$. The map π_0 then maps $y \in G = H_0$ to $3^2 \cdot y \in H_2$, while π_1 maps $y \in H_1$ to $3 \cdot y \in H_2$.

First we see that $\pi_0(g) = 9 \cdot 1 = 9$.

With $y_0 = x = 23$, we get $\pi_0(x) = 9 \cdot 23 = 18 = 2 \cdot 9$, so $a_0 = 2$.

We compute $y_1 = x_0 - a_0 \cdot g = 23 - 2 \cdot 1 = 21$. We then get $\pi_1(21) = 3 \cdot 21 = 9 = 1 \cdot 9$, so $a_1 = 1$.

Finally, we compute $y_2 = y_1 - (3a_1) \cdot g = 21 - (3 \cdot 1) \cdot 1 = 18 = 2 \cdot 9$, so $a_2 = 2$.

We get $\log_1 23 = 2 + 1 \cdot 3 + 2 \cdot 3^2 = 23$.

E *Exercise* 25. Use the algorithm from Exercise 24 to compute the discrete logarithm of 85 to the base 11 in the group $\mathbb{Z}_{125}^+$. (Note that we are talking about the additive group here, where the group operation is addition modulo 125 and exponentiation is multiplication modulo 125.)

E *Exercise* 26. Observe that $72 = 2^3 3^2$.

1. In $\mathbb{F}_{73}^*$, 2 generates a subgroup of order 3^2. Use the algorithm from Exercise 24 to compute (by hand) the discrete logarithm of 64 to the base 2 in the subgroup of $\mathbb{F}_{73}^*$.
2. In $\mathbb{F}_{73}^*$, 10 generates a subgroup of order 2^3. Use the algorithm from Exercise 24 to compute (by hand) the discrete logarithm of 27 to the base 10 in the subgroup of $\mathbb{F}_{73}^*$.
3. Use the algorithm from Exercise 16 together with the algorithm from Exercise 24 to compute (by hand) the discrete logarithm of 70 to the base 5 in the group $\mathbb{F}_{73}^*$. You should use the results of the previous two exercises in your computation.

An alternative, more computational approach to proving Proposition 10 begins by writing $a = \sum a_i \ell^i$ and then observing that if

$$y = g^{\sum_{i=j}^{r-1} a_i \ell^i},$$

then

$$y^{\ell^{j-1}} = g^{a_j \ell^{r-1}}.$$

E *Exercise* 27. Give a proof of Theorem 10 using the above computational approach.

E *Example* 9. Consider the group $G = \mathbb{Z}_{27}^+$ with generator $g = 1$, and let $x = 23$. We want to compute $\log_g x$.

We write $\log_g x = a_0 + 3a_1 + 3^2 a_2$.

First we compute $9 \cdot g = 9 \cdot 1 = 9$.

Let $y_0 = x = (a_0 + 3a_1 + 3^2 a_2) \cdot 1$. We compute $9 \cdot 23 = 18 = 2 \cdot 9$, which tells us that

$$9(a_0 + 3a_1 + 3^2 a_2) \equiv 9 \cdot 2 \pmod{27} \quad \Rightarrow \quad a_0 = 2.$$

We get $y_1 = y_0 - 2 \cdot 1 = 21 = (3a_1 + 3^2 a_2) \cdot 1$. We compute $3 \cdot y_1 = 3 \cdot 21 = 9 = 1 \cdot 9$, which tells us that

$$3(3a_1 + 3^2 a_2) \equiv 9 \cdot 1 \pmod{27} \quad \Rightarrow \quad a_1 = 1.$$

We get $y_2 = y_1 - (3 \cdot 1) \cdot 1 = 18 = (3^2 a_2) \cdot 1$. We compute $y_2 = 2 \cdot 9$, which tells us that

$$3^2 a_2 \equiv 9 \cdot 2 \pmod{27} \quad \Rightarrow \quad a_2 = 2.$$

We get that $\log_g 23 = 2 + 3 \cdot 1 + 3^2 \cdot 2 = 23$.

Based on Proposition 4, Theorem 8 and Theorem 10, we arrive at the following requirement.

Requirement 3. If $n = \prod_{i=1}^{l} \ell_i^{r_i}$ is the group order and ℓ_l is the largest prime dividing n, ℓ_l group operations must be an infeasible computation.

3.4 Shank's Baby-step Giant-step

In this section, we shall improve on Proposition 4 by trading reduced computational effort for increased memory use. Let G be a cyclic group of order n. We must assume that there is some *total order* $\preceq$ on the group elements, such that the following two claims hold when $L \lll n$:

Sorting With effort comparable to computing L group operations, a list of pairs of group elements and integers $(x_1, a_1), (x_2, a_2), \ldots, (x_L, a_L)$ can be rearranged into a new list $(y_1, b_1), (y_2, b_2), \ldots, (y_L, b_L)$ satisfying $y_i \preceq y_j$ when $i \leq j$.

Searching With effort comparable to computing one group operation, we can decide if a group element x is present in a list of pairs of group elements and integers $(y_1, b_1), (y_2, b_2), \ldots, (y_L, b_L)$ satisfying $y_i \preceq y_j$ when $i \leq j$. If the element is present, we also learn what its corresponding number b is.

The algorithm we are interested in begins with the observation that $\log_g x = bL + b'$, where $0 \leq b' < L$.

T **Theorem 11.** *Let G be a cyclic group of order n. For any positive integer $L \lll n$, the discrete logarithm of an element $x \in G$ can be computed using memory to hold L group elements and $L + \lceil n/L \rceil$ group operations.*

Proof. First we construct a list of pairs of group elements and their discrete logarithms

$$(1,0),(g,1),(g^2,2),(g^3,3),\ldots,(g^{L-1},L-1).$$

Computing this list requires less than L group operations and memory to hold L group elements. By assumption, we can sort the list into the list

$$(y_1,b_1),(y_2,b_2),(y_3,b_3),\ldots,(y_L,b_L)$$

using effort comparable to what it took to create the list. Note that we can now quickly decide if the discrete logarithm of any group element is less than L, and if so, what its discrete logarithm is.

Recall that we can write $\log_g x = bL + b'$, where $0 \leq b' < L$. Therefore

$$0 \leq \log_g x(g^{-L})^b < L.$$

We can find b and b' by computing successively

$$x, xg^{-L}, x(g^{-L})^2, x(g^{-L})^3, \ldots$$

and for each element check if it is in our list. Note that computing the next element requires one group operation, while the check requires comparable effort.

We know that we will find an element in our list before computing $\lceil n/L \rceil$ elements. It follows that we will find the discrete logarithm of x with at most $L + \lceil n/L \rceil$ group operations. □

E *Exercise* 28. The proof of Theorem 11 essentially describes an algorithm for computing discrete logarithms. Implement this algorithm and restate Theorem 11 as a statement about the algorithm's time complexity (in terms of group operations, ignoring any other form of computation involved).

E *Example* 10. Consider the group $G = \mathbb{Z}_{29}^+$ with generator $g = 1$, and let $x = 23$. We want to compute $\log_g x$.

We first compute the list of group elements, choosing $L = 5$. Sorting is easy for this group.

$i \cdot g$	0	1	2	3	4
i	0	1	2	3	4

We also compute $-5 \cdot g = 24$.

We see that 23 is not in the table, so we compute $23 + 24 = 18$. This is not in the table, so we compute $18 + 24 = 13$. This is not in the table, so we compute $13 + 24 = 8$. Again, this is not in the table, so we compute $8 + 24 = 3$.

We find that 3 is in the table, with discrete logarithm 3. We have added $-5 \cdot g$ to 23 four times. It follows that

$$\log_g 23 = 4 \cdot 5 + 3.$$

E *Exercise* 29. What value of L minimizes computational effort?

E *Exercise* 30. Using the algorithm from Exercise 28 to compute (by hand) the discrete logarithm of 51 to the base 2 in the group $\mathbb{F}_{59}^*$.

E *Exercise* 31. Decide the optimal value of L if we need to compute the discrete logarithm of k group elements $x_1, x_2, \ldots, x_k$.

E *Exercise* 32. Suppose we know that $\log_g x < k$. Show that for any $L > 0$ we can compute this discrete logarithm using at most $L + \lceil k/L \rceil$ group operations.

E *Exercise* 33. Suppose we know that $k_1 \leq \log_g x < k_2$. Show that for any $L > 0$ we can compute this discrete logarithm using at most $L + \lceil (k_2 - k_1)/L \rceil$ group operations.

Based on Theorems 8, 10 and 11, we arrive at the following requirement.

Requirement 4. If n is the group order and ℓ is the largest prime dividing n, then $L + \lceil \ell/L \rceil$ group operations using memory for L group elements must be an infeasible computation.

3.5 Pollard's ρ

In this section, we shall consider a group G with a prime n number of elements. In the previous section, we saw that we could trivially recover the discrete logarithm of x given an equation of the form $x(g^{-L})^b = g^{b'}$. Pollard's ρ method will try to generate an equation of the form

$$g^a x^b = g^{a'} x^{b'}, \tag{1}$$

with $b \not\equiv b' \pmod{n}$, after which it is easy to see that $\log_g x \equiv (a - a')(b' - b)^{-1} \pmod{n}$.

Pollard's ρ method relies on connecting two separate ideas with a conjecture to arrive at an algorithm for finding a relation like (1) above. The first idea is that in sequences of randomly chosen elements we will quickly see repetitions. The second idea is that in certain sequences we can quickly find cycles. The conjecture is that, with respect to repetitions, certain non-random sequences "look random". It is the repetitions in this sequence that will cause cycles, which we can quickly find and which will give us the required relation.

We begin with the first idea and a sequence of elements chosen at random. What is the probability that there are no repetitions within the first L elements of the sequence?

T **Proposition 12.** *Suppose we have a sequence of elements $s_1, s_2, s_3, \ldots$, the elements chosen independently and uniformly at random from a set S with n elements. Let E be the event that the L first elements are all distinct. Then*

$$1 - \frac{L(L-1)}{2n} \leq \Pr[E] \leq \exp\left(-\frac{L(L-1)}{2n}\right).$$

Proof. If $L \geq n$, then $\Pr[E] = 0$ and the claim holds. So we may assume $L < n$.

Choosing elements one after another, we see that we have n choices for the first element, $n-1$ choices for the second element, $n-2$ choices for the third element, and so forth. By independence and uniformity, we get that

$$\Pr[E] = (1-1/n)(1-2/n)\cdots(1-(L-1)/n).$$

Note that $1-\epsilon \le \exp(-\epsilon)$ for any ϵ, so

$$\Pr[E] \le e^{-1/n}e^{-2/n}\cdots e^{-(L-1)/n} = \exp\left(-\sum_{i=1}^{L-1} i/n\right) = \exp\left(-\frac{L(L-1)}{2n}\right).$$

For the lower bound, we consider the complementary event. Let F_i be the event that the ith chosen element coincides with at least one of the previous elements. Then $\Pr[F_i] \le (i-1)/n$, and we get that

$$\begin{aligned}1-\Pr[\neg E] &= 1-\Pr[F_1 \vee F_2 \vee \cdots \vee F_L] \\ &\ge 1-\sum_{i=1}^{L}\Pr[F_i] \ge 1-\sum_{i=1}^{L}\frac{i-1}{n} \ge 1-\frac{L(L-1)}{2n},\end{aligned}$$

which concludes the proof. □

Note that the sequence we shall consider next is far from random, and this proposition and its proof do not apply. But many non-random sequences are still "random-looking" with respect to repetitions in the sequence, which means that the bounds in the proposition apply in practice. This will be sufficient for our purposes.

Now we shall consider the problem of finding cycles in the special kind of infinite sequences generated by a starting point $s_1 \in S$, a function $f : S \to S$ and the rule $s_{i+1} = f(s_i)$. The sequence eventually becomes cyclic (for some integers i, j, k, $s_{j+k} = s_j$ when $j \ge i$) when S is finite.

T **Proposition 13.** *Let $s_1, s_2, \ldots$ be the sequence defined by s_1 and the rule $s_{i+1} = f(s_i)$. Suppose k is the smallest integer such that $s_k = s_{k'}$ for some $k' < k$. Then distinct indexes i, j can be found such that $s_i = s_j$ using at most $3k$ evaluations of f.*

Proof. We consider the sequence $t_1, t_2, \ldots$ given by $t_j = s_{2j}$. It is clear that for some i, $t_i = s_i$, 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

$$(s_{i+1}, t_{i+1}) = (f(s_i), f(f(t_i))).$$

We will notice when $s_i = t_i$, at which point we have found $s_i = s_{2i}$. Computing each new pair requires evaluating f three times. The claim follows. □

E *Exercise* 34. The proof of Proposition 13 essentially describes an algorithm for finding two integers. Implement this algorithm and restate Proposition 13 as a statement about the algorithm's time complexity (in terms of evaluations of the function f).

Now we are ready to construct a sequence that should allow us to find an equation like (1) and thereby compute discrete logarithms.

Our set will be the group G, of course. Suppose $\{S_1, S_2, S_3\}$ is a partition of $G - S_1$, S_2 and S_3 are pairwise disjoint sets whose union is G – where the three subsets have approximately the same cardinality. Suppose also that it is easy to check which subset an element is in.

Finally, we are ready to construct a sequence $y_1, y_2, \ldots$ based on S_1, S_2, S_3, a generator g and a group element x. We let $y_1 = x$ and

$$y_{i+1} = \begin{cases} y_i g & y_i \in S_1, \\ y_i^2 & y_i \in S_2, \text{ or} \\ y_i x & y_i \in S_3. \end{cases} \tag{2}$$

Based on the sequence $y_1, y_2, \ldots$, we can define a sequence of integer pairs $(a_1, b_1), (a_2, b_2)$, starting with $(a_1, b_1) = (0, 1)$ and using the rule

$$(a_{i+1}, b_{i+1}) = \begin{cases} (a_i + 1 \bmod n, b_i) & y_i \in S_1, \\ (2a_i \bmod n, 2b_i \bmod n) & y_i \in S_2, \text{ or} \\ (a_i, b_i + 1 \bmod n) & y_i \in S_3. \end{cases} \tag{3}$$

E *Exercise* 35. Show that the above two sequences defined by (2) and (3) satisfy $y_i = g^{a_i} x^{b_i}$.

E *Example* 11. Consider $G = \mathbb{Z}_{31}^+$ with generator $g = 1$, $x = 7$ and the (randomly chosen) partition:

S_1	4, 9, 11, 14, 15, 16, 21, 24, 29, 30
S_2	1, 6, 7, 8, 10, 18, 19, 20, 25, 27
S_3	0, 2, 3, 5, 12, 13, 17, 22, 23, 26, 28

Given the starting point $y_1 = x = 7$, we get the sequences

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
j		1		2		3		4		5		6		7		8
y_i	7	14	15	16	17	24	25	19	7	14	15	16	17	24	25	19
	S_2	S_1	S_1	S_1	S_3	S_1	S_2	S_2	S_2	S_1	S_1	S_1	S_3	S_1	S_2	S_2
a_i	0	0	1	2	3	3	4	8	16	1	2	3	4	4	5	10
b_i	1	2	2	2	2	3	3	6	12	24	24	24	24	25	25	19

By inspection, we see that $y_9 = y_1 = 7$, while the algorithm from Exercise 34 finds the repetition $y_8 = y_{2\cdot 8}$.

E *Exercise* 36. Consider the subgroup G of $\mathbb{F}_{107}^*$ with 53 elements. Use the generator $g = 11$ and $x = 85$. Also use the partition where $y \in S_i \Leftrightarrow y \equiv i - 1 \pmod 3$.

1. Compute (by hand) the 20 first terms of the sequences from (2) and (3).
2. Find by inspection the first repetition in the sequence $y_1, y_2, \ldots$.
3. Use the algorithm from Exercise 34 to find a repetition in this sequence.

Exercise 37. Let $y_1, y_2, \ldots$ and $(a_1, b_1), (a_2, b_2), \ldots$ be the above sequences defined by (2) and (3) and suppose k is the smallest integer such that $y_k = y_{k'}$ for some $k' < k$. Prove that distinct indexes i, j along with corresponding pairs (a_i, b_i) and (a_j, b_j) can be found such that $y_i = y_j$ using at most $3k$ group operations and $6k$ additions and multiplications modulo n.

Let E be the event that the L first elements of $y_1, y_2, \ldots$ are all distinct, and let E' be the event that the L first pairs of $(a_1, b_1), (a_2, b_2), \ldots$ are all distinct. Then we can define the two functions

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

We can now prove the following result.

Theorem 14. *Let G be a cyclic group of order n. The discrete logarithm of an element $x \in G$ can be computed using $3L$ group operations and $6L + 3$ arithmetic operations modulo n, except with probability at most $\theta(L, n) + \gamma(L, n)$.*

Proof. Some element will appear twice among the L first elements of the sequence described by (2) except with with probability $\theta(L, n)$.

The corresponding pairs (a_i, b_i) and (a_j, b_j) will be distinct except with probability $\gamma(L, n)$.

This repetition $y_i = y_j$ gives us an equation of the form (1), namely

$$g^{a_i} x^{b_i} = g^{a_j} x^{b_j}.$$

When $b_i \neq b_j$, we can compute the discrete logarithm of x, since n is prime.

By Exercise 37 we can find the indexes of the repetition, while at the same time keeping track of the sequence described by (3), using at most $3L$ group operations and $6L$ arithmetic operations modulo n. The claim follows. □

Exercise 38. The proof of Theorem 14 essentially describes an algorithm for computing discrete logarithms. Implement this algorithm and restate the claim in Theorem 14 as a statement about the algorithm's time complexity (in terms of group operations and arithmetic operations modulo n) and success probability (in terms of $\theta(L, n)$ and $\gamma(L, n)$).

Example 12. Continuing from Example 11, we use the algorithm from Exercise 37 to find a repetition in the sequence $y_1, y_2, \ldots$, and find that $y_{16} = y_8$.

From the already computed values, we find that

$$8 \cdot 1 + 6 \cdot x = 10 \cdot 1 + 19 \cdot x \quad \Rightarrow \quad \log_g x \equiv \frac{8 - 10}{19 - 6} \equiv (-2)(12) \equiv 7 \pmod{31}.$$

Exercise 39. Use the algorithm from Exercise 38 to compute (by hand) the discrete logarithm of 85 to the base 11 in the group $\mathbb{F}^*_{107}$.

When we choose a reasonable partition $\{S_1, S_2, S_3\}$, it seems plausible that something similar to the claims of Proposition 12 should hold for the two sequences, and in practice, this seems to be true. We phrase this in terms of a conjecture. Note that in this conjecture the probability is taken over the choice of the element x.

Conjecture 15. *For reasonable partitions $\{S_1, S_2, S_3\}$, the function $\theta(L, n)$ is roughly similar to*

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

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

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

Based on Conjecture 15 and Theorems 8, 10 and 14, we arrive at the following requirement.

Requirement 5. If n is the group order and ℓ is the largest prime dividing n, then $\sqrt{\ell}$ group operations must be an infeasible computation.

4 Primality Testing

Throughout this section, we shall take n to be a large odd integer.

We want to be able to distinguish prime integers from composite integers. The following proposition is obvious, since a composite integer must have a proper divisor smaller than the square root of the integer and we can in principle check every possible divisor.

Proposition 16. *We can decide if a number n is prime using at most $\sqrt{n}$ integer divisions.*

Unfortunately, the algorithm implied by this proposition is useless for large integers, and therefore for our purposes. However, having divisors is just one difference in behaviour between composite and prime integers.

To distinguish primes from composites, we shall find a subset S of $\mathbb{Z}^*_n$ with three properties:

1. It is easy to check if an element is in S or not.
2. If n is prime, then $S = \mathbb{Z}^*_n$.
3. If n is composite, then S contains at most half of all the elements of $\mathbb{Z}^*_n$.

If our three properties are satisfied, we can recognize primes as follows. First, we choose k elements from $\mathbb{Z}_n^*$ uniformly and independently at random. Then we check if all of these elements are in the subset S. If any element is not in the subset, we conclude that n is composite. Otherwise, we conclude that n *may be prime*.

If one or more of the elements are not in the subset, we have proved that S is a proper subset, from which it follows that n is composite. Any element that is not in the subset is a *witness* for the fact that n is not prime.

The probability that all of the random elements lie inside S is of course $(|S|/|\mathbb{Z}_n^*|)^k$. For moderately large k, if $|S|/|\mathbb{Z}_n^*| < 1/2$, this probability will be very small. It follows logically that if all of the random elements lie inside S, it is most likely that n is prime. Note that *it is not certain that n is prime*, just likely.

4.1 Fermat's Test

Our first attempt is based on Fermat's little theorem.

T **Theorem 17.** *If n is prime, then for any integer a not divisible by n,*

$$a^{n-1} \equiv 1 \pmod{n}. \tag{4}$$

We now define our subset, which turns out to also be a subgroup.

E *Exercise* 40. Let $G_n \subseteq \mathbb{Z}_n^*$ be the set

$$G_n = \{x \in \mathbb{Z}_n^* \mid x^{n-1} = 1\}. \tag{5}$$

Show that G_n is a subgroup of $\mathbb{Z}_n^*$, and that if n is prime then $G_n = \mathbb{Z}_n^*$.

E *Exercise* 41. Show that there is an algorithm that can decide if an element $x \in \mathbb{Z}_n^*$ is in G_n or not using at most one exponentiation.

We have now shown that we have a subset G_n and that we can distinguish its members easily. Fermat's little theorem says that if n is prime, then $G_n = \mathbb{Z}_n^*$. Furthermore, if G_n is a proper subgroup, then it will contain at most half of all the elements of $\mathbb{Z}_n^*$.

E *Exercise* 42. Exercise 41 together with the strategy described above informally describes an algorithm that decides if the subgroup G_n is a proper subgroup of $\mathbb{Z}_n^*$. Implement this algorithm. Formulate and prove a statement about the algorithm's time complexity (in terms of exponentiations) and success probability (the probability that the algorithm decides correctly).

Note the careful phrasing. The exercise only says that the algorithm can decide if G_n is likely to be a proper subgroup, not if n is likely to be prime. Of course, if G_n is a proper subgroup, we know that n is composite, so we can attempt to use this to prove that numbers are composites.

E *Example* 13. Consider $n = 55$. First we try 34 and compute that

$$34^{54} \equiv 1 \pmod{55}.$$

This means that 34 does not disprove that 55 is prime. Next we try 21 and compute that

$$21^{54} \equiv 1 \pmod{55}.$$

Again, this means that 21 does not disprove that 55 is prime. Next we try 27 and compute that

$$27^{54} \equiv 9 \pmod{55},$$

which proves that 55 is composite. Note that we have not found any factors of 55.

E *Exercise* 43. Use the algorithm from Exercise 42 to prove two of the three numbers 767, 13457 and 83693 composite.

Unfortunately, our strategy will fail for some composite numbers.

D **Definition 5.** A *Carmichael number* is a composite integer n such that $G_n = \mathbb{Z}_n^*$.

The algorithm from Exercise 42 does not distinguish prime numbers from composite numbers, but rather prime and Carmichael numbers from non-Carmichael composites.

E *Exercise* 44. Use the algorithm from Exercise 42 to prove one of the two numbers 294409 and 951253 composite. What about the other number?

Carmichael numbers not only exist, but there are infinitely many of them. Fortunately, Carmichael numbers are rare, in the sense that we are unlikely to run into one in most cryptographic applications. This means that for most cryptographic purposes, this test would be good enough. But it is easy to do better.

4.2 Soloway-Strassen Test

The Soloway-Strassen test relies on a simple way to decide if a number is a square modulo a prime. Before we can explain the test, we require a bit of number theory.

D **Definition 6.** Let p be a prime. For any integer a, the Legendre symbol is defined to be

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if there exists } b \not\equiv 0 \pmod{p} \text{ such that } b^2 \equiv a \pmod{p}, \\ 0 & p \text{ divides } a, \text{ and} \\ -1 & \text{otherwise.} \end{cases}$$

Let n be an integer with prime factorization $n = \prod_i p_i^{r_i}$. For any integer a, the Jacobi symbol is defined to be

$$\left(\frac{a}{n}\right) = \prod_i \left(\frac{a}{p_i}\right)^{r_i}.$$

Note that if n is prime, the Jacobi symbol coincides with the Legendre symbol.

We can consider the Legendre symbol to be a map from $\mathbb{F}_p^*$ to its subgroup $\{\pm 1\}$. The inverse image of 1 is the set of all group elements that are squares.

Likewise, we can consider the Jacobi symbol $\left(\frac{\cdot}{n}\right)$ to be a map from $\mathbb{Z}_n^*$ to its subgroup $\{\pm 1\}$.

E *Exercise* 45. Let $H_n \subseteq \mathbb{Z}_n^*$ be the set

$$H_n = \left\{ x \in \mathbb{Z}_n^* \mid x^{\frac{n-1}{2}} = \left(\frac{x}{n}\right) \right\}$$

Show that H_n is a subgroup of $\mathbb{Z}_n^*$.

We have defined our subset. We now need to be able to decide if an element lies in H_n or not, which means that we need to be able to compute the Jacobi symbol quickly. We can do this using quadratic reciprosity as well as other properties of the Jacobi symbol. We state the properties of the Jacobi symbol without proof.

T **Fact 18.** *Let n be an odd integer and let a, b be integers that are relatively prime to n. For the final point, a should also be odd. Then the following hold:*

1. $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$ *if* $a \equiv b \pmod{n}$.
2. $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$.
3. $\left(\frac{2}{n}\right) = \begin{cases} 1 & \text{if } n \text{ is congruent to } 1 \text{ or } 7 \text{ modulo } 8, \\ -1 & \text{otherwise.} \end{cases}$
4. $\left(\frac{a}{n}\right) = \begin{cases} -\left(\frac{n}{a}\right) & \text{if } n \text{ and } a \text{ are both congruent to } 3 \text{ modulo } 4, \\ \left(\frac{n}{a}\right) & \text{otherwise.} \end{cases}$

E *Exercise* 46. Show that using the above properties of the Jacobi symbol, we can construct an algorithm for quickly computing the Jacobi symbol. Implement this algorithm.

E *Example* 14. We want to compute the Jacobi symbol $\left(\frac{34}{55}\right)$. We get

$$\left(\frac{34}{55}\right) = \left(\frac{2}{55}\right)\left(\frac{17}{55}\right) = \left(\frac{17}{55}\right) = \left(\frac{55}{17}\right) = \left(\frac{4}{17}\right) = 1.$$

We use the second, third, fourth, first and third properties of the Jacobi symbol, respectively.

E *Example* 15. We want to compute the Jacobi symbol $\left(\frac{34}{385}\right)$. We get

$$\begin{aligned}\left(\frac{34}{385}\right) &\overset{2.}{=} \left(\frac{2}{385}\right)\left(\frac{17}{385}\right) \overset{3.}{=} \left(\frac{17}{385}\right) \overset{4.}{=} \left(\frac{385}{17}\right) \overset{1.}{=} \left(\frac{11}{17}\right) \overset{4.}{=} \left(\frac{17}{11}\right) \\ &\overset{1.}{=} \left(\frac{6}{11}\right) \overset{2.}{=} \left(\frac{2}{11}\right)\left(\frac{3}{11}\right) \overset{3.}{=} -\left(\frac{3}{11}\right) \overset{4.}{=} \left(\frac{11}{3}\right) \overset{1.}{=} \left(\frac{2}{3}\right) \overset{3.}{=} -1.\end{aligned}$$

The Jacobi symbol property used is marked above each equality sign.

E *Exercise* 47. Compute the Jacobi symbol $\left(\frac{7}{294409}\right)$.

The next theorem shows that $H_n = \mathbb{Z}_n^*$ when n is prime.

T **Theorem 19.** *Let n be an odd prime and let $a \in \mathbb{Z}_n^*$. Then*

$$\left(\frac{a}{n}\right) = a^{\frac{n-1}{2}}. \tag{6}$$

Proof. Recall that the Legendre symbol of a group element is 1 if and only if that element is a square in the group.

When n is prime, Proposition 21 says that $\mathbb{Z}_n^*$ is cyclic of order $n-1$. Let g be any generator. When the group order $n-1$ is even, the squares are the elements of the form g^{2i} for integers i, while the non-squares are of the form g^{2i+1}. We see that

$$(g^{2i})^{\frac{n-1}{2}} = (g^{n-1})^i = 1 \qquad \text{and} \qquad (g^{2i+1})^{\frac{n-1}{2}} = (g^{n-1})^i g^{\frac{n-1}{2}} = -1,$$

which proves the theorem. □

We are now very close to an algorithm that decides if a number is prime or not. All that remains is to show that whenever n is composite, H_n is a proper subgroup of $\mathbb{Z}_n^*$.

T **Theorem 20.** *Let n be an odd composite number. Then there exists $a \in \mathbb{Z}_n^*$ such that (6) does not hold.*

Proof. We consider two cases, whether or not n is square-free.

First, suppose there is a prime p such that p^2 divides n. Let $n_2 = n/p$, and let a be the integer $1 + n_2$, which is not congruent to 1 modulo n, but is congruent to 1 modulo n_2 and p. The Jacobi symbol is

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{n_2}\right) = \left(\frac{1}{p}\right)\left(\frac{1}{n_2}\right) = 1.$$

We shall show that a has order p modulo n. Since p does not divide $n-1$, it will follow that $a^{(n-1)/2}$ is not congruent to 1 modulo n, and therefore that $a + \langle n \rangle$ does not satisfy (6).

From the binomial theorem, it follows that

$$a^p \equiv (1 + n_2)^p \equiv \sum_{i=0}^{p} \binom{p}{i} n_2^i \pmod{n}.$$

It is clear that n divides every other term of the sum except the first term (which is 1) and possibly the second term. But since p divides $\binom{p}{1}$, n divides the second term too, and a^p has order p modulo n.

Next, suppose that no square of any prime divides n. Let p be any prime dividing n and let $n_2 = n/p$. Let also b be any integer that is not a square modulo p with $\gcd(b, n) = 1$. By the Chinese remainder theorem, we can find an integer a satisfying

$$\begin{aligned} a &\equiv 1 \pmod{n_2}, \\ a &\equiv b \pmod{p}. \end{aligned}$$

Note that the first equation implies that

$$a^{(n-1)/2} \equiv 1^{(n-1)/2} \equiv 1 \pmod{n_2},$$

which means that $a^{(n-1)/2} \not\equiv -1 \pmod{n}$. Finally, we compute the Jacobi symbol of a as

$$\left(\frac{a}{n}\right) = \left(\frac{b}{p}\right)\left(\frac{1}{n_2}\right) = (-1) \cdot 1 = -1.$$

It follows that (6) does not hold for $a + \langle n \rangle$. □

It is now clear that our strategy will work.

E *Exercise* 48. Exercise 45, Exercise 46 and our strategy can be combined into an algorithm that decides if the subgroup H_n is a proper subgroup of $\mathbb{Z}_n^*$. Implement this algorithm and formulate and prove a statement about the algorithm's time complexity (in terms of exponentiations) and success probability.

Exercise 48 allows us to decide if H_n is a proper subgroup of $\mathbb{Z}_n^*$ or not. By Theorems 19 and 20, we know that it is a proper subgroup if and only if n is a composite number. In other words, we have an efficient way to decide if a number is prime or not.

E *Example* 16. Consider $n = 55$. First we try 34 and compute that

$$34^{27} \equiv 34 \not\equiv \pm 1 \pmod{55},$$

which proves that 55 is composite. Note that we have not found any factors of 55.

E *Example* 17. Consider $n = 385$. First we try 309 and compute that

$$309^{192} \equiv 1 \equiv \left(\frac{309}{385}\right) \pmod{385}.$$

This means that 309 does not disprove that 385 is prime. Next, we try 34 and compute that

$$34^{192} \equiv 1 \not\equiv \left(\frac{34}{385}\right) \pmod{385},$$

which proves that 385 is composite. Note that we have not found any factors of 385.

E *Exercise* 49. Use the algorithm from Exercise 48 to prove the two numbers 294409 and 951253 composite.

E *Exercise* 50. Choose several random numbers between 10^6 and 10^7. Use the algorithm from Exercise 48 to decide primality for each number with reasonable certainty. For the composites, how many random numbers do you need to test before disproving their primality.

Redo the exercise for random numbers between 10^{20} and 10^{21}.

Are the results surprising? Should they be?

5 Finite Fields

The non-zero elements of a finite field with q form an abelian group under multiplication, and we denote this group by $\mathbb{F}_q^*$. The question we shall now investigate is if this group is suitable for use in Diffie-Hellman.

We begin by showing that this group is cyclic. Note that we may still want to use a subgroup and not the whole group.

T **Proposition 21.** *The group $\mathbb{F}_q^*$ is cyclic.*

Proof. Let n be the maximal order of any element in $\mathbb{F}_q^*$. We know that the order of any element in a group divides the group order, so specifically $n \leq q - 1$. Furthermore, the order of any element in $\mathbb{F}_q^*$ must divide n.

For any $x \in \mathbb{F}_q^*$ we therefore have that $x^n = 1$ or $x^n - 1 = 0$.

Now consider the polynomial $X^n - 1$. As we have just seen, every non-zero field element is a zero of this polynomial. We know that a polynomial of degree $d > 0$ over any field has at most d zeros, so $n \geq q - 1$. We can conclude that $n = q - 1$.

Since we have an element of order $q - 1$, the group order, the group is cyclic. □

Requirement 5 says that the order of any group we use in Diffie-Hellman should be divisible by a large prime. In other words, we need a large prime power q such that $q - 1$ is divisible by a large prime.

It turns out (for reasons that we shall not investigate) that computing discrete logarithms in extension fields is easier than in prime fields of comparable size. Therefore we restrict our study to prime fields.

Before we go on to discuss how we can find suitable primes, we first discuss the details of the group operation.

5.1 Group Operation

Let p be a prime. Mathematically, the finite field $\mathbb{F}_p$ consists of a set of p elements, two of which are distinguished (0 and 1), along with two binary operations $+$ and $\cdot$.

The field is isomorphic to the factor ring $\mathbb{Z}/\langle p \rangle$, where it is easy to compute. To add, subtract or multiply, we simply add, subtract or multiply representatives of the cosets to get new representatives. To divide by $\xi + \langle p \rangle$, we first find an inverse ζ of ξ modulo p, then multiply by $\zeta + \langle p \rangle$ (again by multiplying representatives).

This is unproblematic mathematically, but computationally it is awkward because the size of the representatives tends to grow very quickly, making arithmetic very slow. However, we know that two integers represent the same coset if and only if they are congruent modulo p. Which means that after we do some arithmetic operation on the representatives, we may divide the (possibly large) new representative by p and use the remainder as our representative for the result coset.

What happens is that we represent the field elements using the integers $\{0, 1, \ldots, p-1\}$ and do arithmetic as integer arithmetic followed by taking the remainder after division by p.

It follows that a group operation costs two arithmetic operations, while finding an inverse in the group is somewhat more expensive.

E *Exercise* 51. Find reasonable algorithms for addition, subtraction, multiplication, division. We can use the Extended Euclidian algorithm to compute inverses. Implement any of these that are missing in your favourite programming language. Compare these algorithms in terms of run-time.

5.2 Finding Suitable Primes

The prime number theorem says that in any given range of integers, primes are fairly common (inversely proportional to the number of digits). Since we can efficiently recognize primes, we can fairly quickly find large primes simply by choosing random large numbers until we find a prime.

By Requirement 5, we need a cyclic group such that the group order is divisible by a large prime. This means that we are not just looking for primes, we are looking for a prime p such that $p-1$ is divisible by a large prime.

We can do this by first choosing a sufficiently large prime ℓ and then choosing random numbers k until $2k\ell+1$ is prime. In practice, this algorithm performs as well as a search for an arbitrary prime.

E *Example* 18. Testing integers sequentially starting at 2000, we find that $\ell = 2003$ is prime. We then test multiples starting with 101 and find that when $k = 103$ the number $p = 2k\ell + 1 = 412619$ is prime.

Sometimes, we want the $p-1$ to be twice a prime ℓ. In this case, p is called a *safe prime* and ℓ is called a *Sophie-Germain prime*. This time, the need for ℓ and $2\ell+1$ to be prime simultaneously means that we need to look at many more candidates before we find a suitable prime. This will be slow, but there are techniques to speed up the search.

E *Example* 19. Again, testing integers sequentially starting at 2003, we find that $\ell = 2039$ is prime at the same time as $p = 2\ell + 1 = 4079$.

E *Exercise* 52. About one third of all candidates will be divisible by 3. Checking that a number is not divisible by 3 is much faster than using the primality testing algorithms from Section 4. Expand on this idea and explain how we can use so-called trial division by small primes to exclude most candidate primes before finally using the algorithms from Section 4.

E *Exercise* 53. The Sieve of Eratosthenes can be used to quickly find the first small primes. Adapt the sieving idea to quickly find the most likely prime candidates from an integer range.

5.3 Index Calculus

All of the algorithms from Section 3 will work for $\mathbb{F}_p^*$. But it turns out that we can do very much better. We shall develop the ideas of index calculus in a general setting, and then show how the properties of prime fields allow us to apply the ideas to get a more efficient algorithm for computing discrete logarithms.

We begin with an observation about abstract cyclic groups, which is an extension of (1). Let G be a cyclic group of order n. Let g be some generator and let x be a group element. Suppose we have ν pairs of integers $(r_1, t_1), (r_2, t_2), \ldots, (r_\nu, t_\nu)$ and integers $\alpha_1, \alpha_2, \ldots, \alpha_\nu$ (not all congruent to zero modulo n) such that

$$\prod_{i=1}^{\nu} \left(x^{t_i} g^{r_i}\right)^{\alpha_i} = 1. \tag{7}$$

This will give us the equation

$$x^{\sum_i \alpha_i t_i} g^{\sum_i \alpha_i r_i} = 1,$$

which is of the same form as (1). As long as $\sum_i \alpha_i t_i$ is invertible modulo n, we can recover $\log_g x$ from the equation using $2\nu+2$ arithmetic operations ($\nu+1$ multiplications, ν additions and one inversion).

We first consider the case when the group order n is prime.

E *Example* 20. Consider the prime $p = 1019$ with the elements $g = 3$, generating a subgroup G of $\mathbb{F}_p^*$ of order 509. Let $x = 11$.

With the relations

$$y_1 = g^{112}x^{239} = 576 \quad y_2 = g^{477}x^{274} = 70 \quad y_3 = g^{378}x^{248} = 180 \quad y_4 = g^{80}x^{66} = 42$$
$$y_5 = g^{331}x^{488} = 720$$

and integers $\alpha_1 = 145$, $\alpha_2 = 436$, $\alpha_3 = 72$, $\alpha_4 = 73$ and $\alpha_5 = 1$, we get that

$$y_1^{145} x_2^{436} y_3^{72} y_4^{73} y_5 = 1.$$

This means that

$$\log_g x \equiv -\frac{145 \cdot 112 + 436 \cdot 477 + 72 \cdot 378 + 73 \cdot 80 + 331}{145 \cdot 239 + 436 \cdot 274 + 72 \cdot 180 + 73 \cdot 66 + 488} \equiv -\frac{45}{149} \equiv 191 \pmod{509}.$$

A quick calculation proves that this is the correct discrete logarithm.

Exercise 54. Suppose n is prime. Consider (7). Let

$$y_i = x^{t_i} g^{r_i},\ i = 1, 2, \ldots, \nu.$$

Suppose further that the coefficients $r_1, r_2, \ldots, r_\nu$, $t_1, t_2, \ldots, t_\nu$ have been chosen uniformly at random, but the coefficients $\alpha_1, \alpha_2, \ldots, \alpha_\nu$ only depend on the group elements $y_1, y_2, \ldots, y_\nu$. Show that $\sum_i \alpha_i t_i$ is divisible by n with probability $1/n$.

Proposition 22. *Suppose n is prime and that $\bar{\ell}_1, \bar{\ell}_2, \ldots, \bar{\ell}_l$ are elements of G. Given $l+1$ distinct, non-trivial relations of the form*

$$y_i = \prod_{j=1}^{l} \bar{\ell}_j^{s_{ij}}, \qquad i = 1, 2, \ldots, l+1,$$

we can compute $\alpha_1, \alpha_2, \ldots, \alpha_{l+1}$ satisfying

$$\prod_{i=1}^{l+1} y_i = 1$$

using at most $(l+1)^3$ arithmetic operations.

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 over $\mathbb{F}_n$, it has rank at most l, so there exists a non-zero vector $\boldsymbol{\alpha}$ such that $\boldsymbol{\alpha}\mathbf{S} = \mathbf{0}$. Given such a vector, we get

$$\prod_{i=1}^{l+1} y_i^{\alpha_i} = \prod_{i=1}^{l+1} \left(\prod_{j=1}^{l} \bar{\ell}_j^{s_{ij}} \right)^{\alpha_i} = \prod_{j=1}^{l} \bar{\ell}_j^{\sum_{i=1}^{l+1} \alpha_i s_{ij}} = 1.$$

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

Example 21. Continuing with the discrete logarithm problem and relations from Example 20, we have the elements 2, 3, 5 and 7 in the subgroup of $\mathbb{F}_p^*$ and the relations:

$$y_1 = 576 = 2^6 \cdot 3^2 \qquad y_2 = 70 = 2 \cdot 5 \cdot 7 \qquad y_3 = 180 = 2^2 \cdot 3^2 \cdot 5$$
$$y_4 = 42 = 2 \cdot 3 \cdot 7 \qquad y_5 = 720 = 2^4 \cdot 3^2 \cdot 5$$

This gives us the matrix

$$\mathbf{S} = \begin{pmatrix} 6 & 2 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 2 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 4 & 2 & 1 & 0 \end{pmatrix}.$$

The Gaussian elimination in $\mathbb{F}_{509}$ proceeds as follows:

$$\left(\begin{array}{cccc|ccccc} 6 & 2 & 0 & 0 & 1 & & & & \\ 1 & 0 & 1 & 1 & & 1 & & & \\ 2 & 2 & 1 & 0 & & & 1 & & \\ 1 & 1 & 0 & 1 & & & & 1 & \\ 4 & 2 & 1 & 0 & & & & & \end{array}\right) \sim \left(\begin{array}{cccc|ccccc} 6 & 2 & 0 & 0 & 1 & & & & \\ & 339 & 1 & 1 & 424 & 1 & & & \\ & 171 & 1 & 0 & 339 & & 1 & & \\ & 340 & 0 & 1 & 424 & & & 1 & \\ & 340 & 1 & 0 & 169 & & & & \end{array}\right)$$

$$\sim \left(\begin{array}{cccc|ccccc} 6 & 2 & 0 & 0 & 1 & & & & \\ & 339 & 1 & 1 & 424 & 1 & & & \\ & & 5 & 4 & 508 & 4 & 1 & & \\ & & 2 & 3 & 254 & 2 & & 1 & \\ & & 3 & 2 & 508 & 2 & & & \end{array}\right) \sim \left(\begin{array}{cccc|ccccc} 6 & 2 & 0 & 0 & 1 & & & & \\ & 339 & 1 & 1 & 424 & 1 & & & \\ & & 5 & 4 & 508 & 4 & 1 & & \\ & & & 205 & 458 & 204 & 305 & 1 & \\ & & & 305 & 305 & 305 & 203 & & \end{array}\right)$$

$$\sim \left(\begin{array}{cccc|ccccc} 6 & 2 & 0 & 0 & 1 & & & & \\ & 339 & 1 & 1 & 424 & 1 & & & \\ & & 5 & 4 & 508 & 4 & 1 & & \\ & & & 205 & 458 & 204 & 305 & 1 & \\ & & & & 145 & 436 & 72 & 73 & \end{array}\right).$$

We find $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ in the bottom row of the right half of the matrix.

Proposition 23. *Suppose n is prime and that $\bar{\ell}_1, \bar{\ell}_2, \ldots, \bar{\ell}_l$ are elements of G. Suppose further that for a group element y chosen uniformly at random from G we can find a relation of the form*

$$y = \prod_{j=1}^{l} \bar{\ell}_j^{s_j} \tag{8}$$

with probability σ, using at most τ arithmetic operations.

Then, except with probability $1/n$, we can compute $\log_g x$ using an expected

$$\sigma^{-1}(l+1)(\tau + 2\chi) + (l+1)^3 + 2l + 3$$

arithmetic operations, where χ is the number of arithmetic operations required for an exponentiation in G.

Proof. We shall choose random group elements and for each element use at most τ arithmetic operations to try to find a relation of the form (8). When we have found $l+1$ relations, we shall stop.

We choose random elements of the form $x^t g^r$, ensuring that the coefficients s_j will be independent of the exact random numbers t, r chosen, satisfying the requirements of

Exercise 54.

Now we can apply Proposition 22 to compute a relation among the random group elements, which by Exercise 54 will allow us to compute the discrete logarithm except with probability $1/n$.

We expect to find $l+1$ relations after choosing $\sigma^{-1}(l+1)$ random group elements.

Sampling a random element costs 2χ arithmetic operations, and trying to find relations costs τ arithmetic operations per group element tested.

The linear algebra will require $(l+1)^3$ arithmetic operations, and computing $\log_g x$ will cost at most $2l+3$ arithmetic operations. The claim follows. □

If n is a prime power q^k then Proposition 22 does not apply. If q is small, the algorithms from Section 3 suffice to compute the discrete logarithm quickly. The case where q is not small can be handled as well, but since this is almost never the case in cryptography, we do not discuss the details.

Finally, we consider the case of a more general group order.

T **Proposition 24.** *Let $n = n_1 n_2$, such that n_1 is a large prime that does not divide n_2, let g be a generator of G, let $\bar{\ell}_1, \bar{\ell}_2, \ldots, \bar{\ell}_l$ be elements of G, and let H be the subgroup of G of order n_1. Also, let d be an inverse of n_2 modulo n_1.*

Suppose that for a group element y chosen uniformly at random from G, we can find a relation of the form

$$y = \prod_{j=1}^{l} \bar{\ell}_j^{s_j}$$

with probability σ, using at most τ arithmetic operations.

Then for a group element y' chosen uniformly at random from H, we can find a relation of the form

$$y' = \prod_{j=1}^{l} (\bar{\ell}_j^{n_2 d})^{s_j}$$

with probability σ, using at most $\tau + \chi$ arithmetic operations, where χ is the number of arithmetic operations required for an exponentation in G.

Proof. If y' has been chosen uniformly at random from H, and b has been chosen uniformly at random from $\{0, 1, \ldots, n_2 - 1\}$, then

$$y = y' g^{n_1 b}$$

has been chosen uniformly at random from G.

By assumption, with probability σ we can find $s_1, s_2, \ldots, s_l$ such that

$$y = \prod_{j=1}^{l} \bar{\ell}_j^{s_j}. \tag{9}$$

Then

$$y' = y^{n_2 d} = \prod_{j=1}^{l} \bar{\ell}_j^{s_j n_2 d} = \prod_{j=1}^{l} (\bar{\ell}_j^{n_2 d})^{s_j}.$$

We have a relation of the required form, and we succeed with probability σ.

Generating y requires χ arithmetic operations. Finding (9) requires τ arithmetic operations. □

This result says that if we can find relations in the big group, we can move that relation into a subgroup. By Proposition 23 we can then compute discrete logarithms in the subgroup. Theorem 8 now applies. (In fact, we can do even better by reusing relations that we find in the big group for each of the subgroups, and not finding new relations for each subgroup.)

Now we let $G = \mathbb{F}_p$. We want a way to find relations of the form (8) for randomly chosen group elements.

Let $\ell_1, \ell_2, \ldots, \ell_l$ be the l smallest primes (listed in order, so that $\ell_1 = 2$, $\ell_2 = 3$, etc.), and let $\bar{\ell}_1, \bar{\ell}_2, \ldots, \bar{\ell}_l$ be the corresponding group elements in $\mathbb{F}_p^*$. Define the sets

$$P = \left\{ \prod_{j=1}^{l} \ell_j^{s_j} < p \;\middle|\; s_1, s_2, \ldots, s_l \geq 0 \right\} \quad \text{and} \quad \bar{P} = \{k + \langle p \rangle \mid k \in P\}.$$

It is clear that for any $y \in \mathbb{F}_p^*$, we can choose a smallest non-negative coset representative and test if $y \in \bar{P}$ by repeated trial division.

E *Exercise* 55. Show that we can decide if a group element is in $\bar{P}$ or not using at most $\tau = \lfloor l + \log_2 p \rfloor$ integer divisions.

E *Exercise* 56. Show that we can use the trial division algorithm implied by the previous exercise to find relations of the form (8) for a fraction $\sigma = |\bar{P}|/(p-1)$ of all elements in $\mathbb{F}_p^*$ using at most $\tau = \lfloor l + \log_2 p \rfloor$ arithmetic operations.

E *Exercise* 57. A group operation in $\mathbb{F}_p^*$ requires two arithmetic operations (one multiplication and one division). Use Proposition 2 to show that we can compute an exponentiation (with an exponent smaller than the group order) in $\mathbb{F}_p^*$ using at most $4 \log_2 p$ arithmetic operations.

E *Example* 22. Consider $p = 272003$ and the subgroup of $\mathbb{F}_p^*$ of order 307, generated by $g = 231904$, and the group element $x = 4644$. We want to compute $\log_g x$. (Note that $p = 2 \cdot 443 \cdot 307 + 1$.)

We find that $h = 255754$ has order $2 \cdot 443$ modulo p. We choose to use the small primes 2, 3, 5, 7 and 11. There are 1647 integers between 1 and p who have no other prime factors, so we expect to find the 6 relations we need after about 1000 tries. A

computer-aided search quickly finds:

$$\begin{aligned} g^{238}x^4h^{20} &= 1089 && = 3^2 \cdot 11^2 \\ g^{217}x^{264}h^{142} &= 180075 && = 3 \cdot 5^2 \cdot 7^4 \\ g^{206}x^{305}h^{355} &= 128 && = 2^7 \\ g^{259}x^{261}h^{464} &= 26244 && = 2^2 3^8 \\ g^{303}x^{231}h^{64} &= 4158 && = 2 \cdot 3^3 \cdot 7 \cdot 11 \\ g^{106}x^{125}h^{793} &= 13122 && = 2 \cdot 3^8 \end{aligned}$$

We get the matrix

$$\mathbf{S} = \begin{pmatrix} 0 & 2 & 0 & 0 & 2 \\ 0 & 1 & 2 & 4 & 0 \\ 7 & 0 & 0 & 0 & 0 \\ 2 & 8 & 0 & 0 & 0 \\ 1 & 3 & 0 & 1 & 1 \\ 1 & 8 & 0 & 0 & 0 \end{pmatrix}.$$

It is now easy to see that with $\boldsymbol{\alpha} = (0, 0, 1, -7, 0, 7)$, we have $\boldsymbol{\alpha}\mathbf{S} = \mathbf{0}$.

We quickly verify by checking that $128 \cdot 26244^{-7} \cdot 13122^7 = 1$.

We then compute

$$\log_g x \equiv -\frac{206 - 7 \cdot 259 + 7 \cdot 106}{305 - 7 \cdot 261 + 7 \cdot 125} \equiv 11 \pmod{307}.$$

For simplicity, we consider a single large prime divisor q of $p-1$ such that q^2 does not divide $p-1$. With the above results, Proposition 23 and Exercises 55 and 57 say that we can compute logarithms modulo q using

$$\sigma^{-1}(l+1)(l + \log_2 p + 8\log_2 p) + (l+1)^3 + 2l + 3$$

arithmetic operations, where $\sigma = |\bar{P}|/(p-1)$. We expect l to be much larger than $\log_2 p$, so we get an approximate cost

$$\sigma^{-1}l^2 + l^3. \tag{10}$$

It is now clear that while we can make the fraction σ large by making l large, that will increase the cost of generating relations and may actually increase the total cost. Making l too large may be counterproductive, and we need to find a good value.

We must estimate $\sigma^{-1} = (p-1)/|P| \approx p/|P|$. We begin by taking logarithms in the equation $\prod_j \ell_j^{s_j} < p$ and observing that the number of integers in P is the same as the number of non-negative integer solutions to the inequality

$$\sum_{j=1}^{l} s_j \log \ell_j < \log p.$$

Observing that most of our small primes have approximately the same logarithm, we can instead count the number of non-negative integer solutions to the inequality

$$\sum_{j=1}^{l} s_j < \frac{\log p}{\log \ell_l}.$$

Letting $u = \log p / \log \ell_l$, we get that

$$|P| \approx \binom{\lfloor u \rfloor + l}{l} = \frac{(\lfloor u \rfloor + l)!}{\lfloor u \rfloor! l!}.$$

Taking logarithms, using Stirling's approximation and replacing $\lfloor u \rfloor$ by u, we get that

$$\log |P| \approx (u+l)\log(u+l) - (u+l) - (u \log u - u) - (l \log l - l).$$

If we additionally assume that u is much smaller than l, we get

$$\log |P| \approx (u+l)\log l - u - l - u \log u + u - l \log l + l = u \log l - u \log u.$$

The prime number theorem says that $l \approx \ell_l / \log \ell_l$, so we get that

$$\begin{aligned} \log \sigma^{-1} \approx \log p - \log |P| &\approx u \log \ell_l - u \log l + u \log u \\ &= u \log \ell_l - u \log \ell_l + u \log \log \ell_l + u \log \log p - u \log \log \ell_l \\ &= \frac{\log p}{\log \ell_l} \log \log p. \end{aligned}$$

We get the rough estimate

$$\sigma^{-1} \approx \exp\left(\frac{\log p}{\log \ell_l} \log \log p \right).$$

The prime number theorem says that

$$l \approx \ell_l / \log \ell_l = \exp\left(\log \ell_l - \log \log \ell_l \right),$$

which means that (10) gives us an approximate cost of

$$\exp\left(\frac{\log p \log \log p}{\log \ell_l} + 2 \log \ell_l - 2 \log \log \ell_l \right) + \exp\left(3 \log \ell_l - 3 \log \log \ell_l \right).$$

Ignoring the $\log \log \ell_l$ terms, we want to choose ℓ_l so as to make the sum

$$\exp\left(\frac{\log p \log \log p}{\log \ell_l} + 2 \log \ell_l \right) + \exp\left(3 \log \ell_l \right)$$

as small as possible. The sum is dominated by the exponential with the biggest exponent. The second exponent increases monotonously with increasing ℓ_l. Since the first exponent has its minimum $\sqrt{8}\sqrt{\log p \log \log p}$ at

$$\log \ell_l = \sqrt{\log p \log \log p / 2}$$

where the second exponent takes the smaller value $\sqrt{9/2}\sqrt{\log p \log\log p}$, we see that taking this value for $\log \ell_l$ should approximate a minimum. In particular, by using this value we should be able to compute discrete logarithms using approximately

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

arithmetic operations.

We have arrived at the following requirement.

Requirement 6. If G is any subgroup of $\mathbb{F}_p^*$, then $\exp(\sqrt{8\log p \log\log p})$ arithmetic operations must be an infeasible computation.

Today, we have much better algorithms for computing discrete logarithms in finite fields. While we shall not study these algorithms, we note that the above requirement is not the final requirement.

E *Exercise* 58. Suppose we want to compute a number (say hundreds) of discrete logarithms in a prime-order subgroup of $\mathbb{F}_p^*$. That is, suppose we have $x_1, \ldots, x_\nu \in G$, $G \subseteq \mathbb{F}_p$, and want to compute $\log_g x_1, \ldots, \log_g x_\nu$.

First, show that given ν relations

$$g^{v_i} \prod_i x_i^{u_i} = 1,$$

you can compute the discrete logarithms. Next, show that the above index calculus algorithms can be adapted to find such relations. Finally, show that this computation is much faster than computing the ν discrete logarithms separately.

Hint: You may make optimistic assumptions about linear independence when needed.

6 Elliptic Curves

Elliptic curves have been studied for a long time by number theorists and a rich and varied theory has been developed. We are interested in elliptic curves because the points on an elliptic curve over a finite field forms a group that is suitable for use in cryptography.

From a mathematical point of view, studying elliptic curves over any field is interesting. From a cryptographic point of view, our groups come from elliptic curves over finite fields, which must therefore be our main interest. To simplify our presentation, we shall restrict ourselves to elliptic curves of a special form defined over prime fields. We note that essentially all of the theory we discuss works equally well for elliptic curves defined over other fields, though sometimes with minor modifications.

Even though we only discuss curves over finite prime fields, it is still convenient to use drawings of curves over the real numbers to illustrate ideas.

We begin by considering the algebraic curve C defined over the field $\mathbb{F}_p$, p a large prime, given by the polynomial equation

$$Y^2 = X^3 + AX + B, \qquad A, B \in \mathbb{F}_p. \tag{11}$$

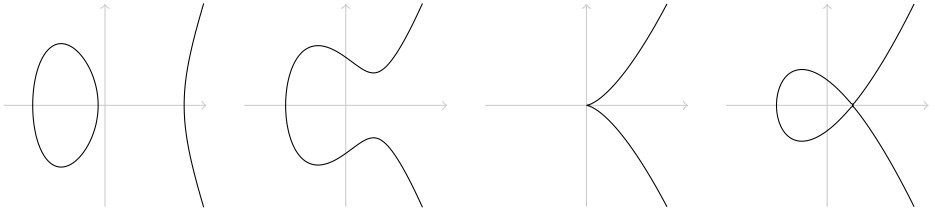

Figure 3: Four cubic curves. From left to right: $Y^2 = X^3 - 6X - 1$, $Y^2 = X^3 - 2X + 2$, $Y^2 = X^3$ and $Y^2 = X^3 - \frac{3}{4}X + \frac{1}{4}$.

The *points* on the curve are the points in the plane that satisfy the curve equation. However, we cannot restrict the coordinates of the points to be elements of $\mathbb{F}_p$. We fix an algebraic closure $\bar{\mathbb{F}}$ of $\mathbb{F}_p$ and consider the points on the curve to be all the pairs $(x, y) \in \bar{\mathbb{F}}^2$ satisfying the curve equation.

A point on a curve is $\mathbb{F}_p$-*rational* (or just *rational*) if its coordinates lie in $\mathbb{F}_p$.

E *Example* 23. Consider the curve over $\mathbb{F}_{13}$ defined by $Y^2 = X^3 + X + 2$. The points in $\mathbb{F}_p^2$ satisfying the equation are

$$(1,2), (1,11), (2,5), (2,8), (6,4), (6,9), (7,1), (7,12), (9,5), (9,8), (12,0).$$

Observe that $11 = -2$, $8 = -5$, etc.

A curve is *smooth* if its partial derivatives never vanish all at the same time for points on the curve. Figure 3 shows four cubic curves, of which two are smooth.

E *Example* 24. Consider the curve over $\mathbb{F}_{13}$ defined by $Y^2 = X^3 + X + 3$. The point $(2, 0)$ is on the curve. Since both partial derivatives vanish at $(2, 0)$, there is no well-defined tangent at that point.

The slope of a line passing through two distinct points on the curve is well-defined. Over a finite field, the tangent line no longer has a natural definition in terms of limits, but we can still use formal partial derivatives to define tangent lines.

E *Exercise* 59. Show that a smooth curve has a well-defined, unique tangent line at any point on the curve.

T **Proposition 25.** *An algebraic curve defined by* (11) *is smooth if and only if the polynomial* $X^3 + AX + B$ *has three distinct zeros.*

Proof. We first compute the partial derivative with respect to Y to get

$$2Y = 0.$$

Any point on the curve where both partial derivatives vanish must therefore have Y-coordinate 0.

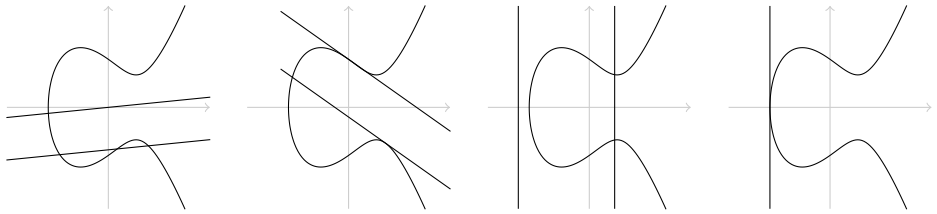

Figure 4: Intersection of lines and elliptic curves. From left to right: three distinct intersection points, possibly with complex Y-coordinates; tangent lines with one or two distinct intersection points; two distinct intersection points, possibly with complex Y-coordinates; and a tangent line with one intersection point.

It follows that the X-coordinate of a point where both partial derivatives vanish will be a zero of both $X^3 + AX + B$ and its derivative $3X^2 + A$. The only way a polynomial and its derivative can have a common zero is if the polynomial has a double or triple zero. We may conclude that the curve is smooth if and only if the polynomial $X^3 + AX + B$ has three distinct zeros. □

T **Fact 26.** *A polynomial $aX^3 + bX^2 + cX + d$ has three distinct zeros if and only if its* discriminant *$b^2c^2 - 4ac^3 - 4b^3d - 27a^2d^2 + 18abcd$ is non-zero.*

In general, an elliptic curve is a smooth cubic curve with one chosen point, but we shall restrict attention to curves of the form we have already discussed.

D **Definition 7.** An *elliptic curve* E over the field $\mathbb{F}_p$ is a cubic curve over $\mathbb{F}_p$ given by (11) with $4A^3 + 27B^2 \neq 0$.

E *Example* 25. Over $\mathbb{F}_{13}$, it is easy to verify that the curve defined by $Y^2 = X^3 + X + 2$ is an elliptic curve, while the curve defined by $Y^2 = X^3 + X + 3$ is not.

We need to study the points of intersection between elliptic curves and straight lines. The situation can be neatly illustrated over the real numbers as in Figure 4. We see that lines can intersect the curve in zero, one, two or three points. This is untidy, and we want a better understanding of what is going on.

T **Proposition 27.** *Let E be an elliptic curve defined by $Y^2 = X^3 + AX + B$.*
If L is a line defined by $Y = \alpha X + \beta$, then the zeros of the polynomial

$$X^3 - \alpha^2 X^2 + (A - 2\alpha\beta)X + B - \beta^2 \tag{12}$$

are the X-coordinates of the points of intersection.
If L is a line defined by $X = \beta$, then the zeros of the polynomial

$$Y^2 - \beta^3 - A\beta - B \tag{13}$$

are the Y-coordinates of the points of intersection.

Proof. We begin with a line L of the form $Y = \alpha X + \beta$. We want to compute the intersection of this line and an elliptic curve defined by $Y^2 = X^3 + AX + B$. Using the line equation to eliminate Y from the curve equation, we find

$$\alpha^2 X^2 + 2\alpha\beta X + \beta^2 = X^3 + AX + B.$$

The solutions to this equation corresponds to the zeros of the cubic polynomial (12). If x is any zero of this polynomial, we know that the point $(x, \alpha x + \beta)$ is on both the line and the curve and therefore a point of intersection.

Next we consider a line L' of the form $X = \beta$. This time, we use the line equation to eliminate X from the curve equation and get

$$Y^2 = \beta^3 + A\beta + B.$$

The solutions to this equation corresponds to the zeros of the quadratic polynomial (13). If y is any zero of this polynomial, we know that the point (β, y) is on both the line and the curve, and therefore a point of intersection. □

E *Example* 26. Consider the elliptic curve over $\mathbb{F}_{13}$ defined by $Y^2 = X^3 + X + 2$, and the six lines $X = 2$, $X = 3$, $X = 12$, $Y = X + 3$ and $Y = X + 4$ and $Y = 2X$.

The line $X = 2$ gives us the equation $Y^2 = 2^3 + 2 + 2 = 12$. We see that $5^2 = 25 = 12$, so the zeros of this polynomial equation are $Y = \pm 5$, giving us the intersection points $(2, 5)$ and $(2, -5) = (2, 8)$.

The line $X = 3$ gives us the equation $Y^2 = 3^3 + 3 + 2 = 6$, but 6 is not a square modulo 13, so the line does not intersect the curve in any rational points.

The line $X = 12$ gives us the equation $Y^2 = 12^3 + 12 + 2 = 0$, so 0 is a double zero of the polynomial. This gives us a single intersection point $(12, 0)$.

The line $Y = X + 3$ gives us the equation $X^2 + 6X + 9 = X^3 + X + 2$, or $X^3 - X^2 - 5X - 7 = 0$. A quick search tells us that the left hand side is $(X - 2)(X - 6)^2$, which means that there are two intersection points, $(2, 5)$ and $(6, 9)$, where $(6, 9)$ is a double zero and the line is a tangent.

The line $Y = X + 4$ gives us the equation $X^2 + 8X + 3 = X^3 + X + 2$, or $X^3 - X^2 - 7X - 1 = 0$. A quick computation tells us that this equation has no rational zeros (solutions in $\mathbb{F}_p$).

The line $Y = 2X$ gives us the equation $4X^2 = X^3 + X + 2$, or $X^3 - 4X^2 + X + 2 = 0$. A quick search tells us that the left hand side factors as $(X - 1)(X - 7)(X - 9)$, which gives us the intersection points $(1, 2)$, $(7, 1)$ and $(9, 5)$.

T **Fact 28.** *Let $\mathbb{F}$ be a field and let $\bar{\mathbb{F}}$ be an algebraic closure of $\mathbb{F}$. Counting multiplicities, a polynomial of degree d over $\mathbb{F}$ has d zeros in $\bar{\mathbb{F}}$.*

D **Definition 8.** Let E be an elliptic curve, P a point on E and L a line intersecting E in P. The *multiplicity* of the point of intersection P is the multiplicity of the corresponding zero of the polynomials in (12) or (13).

*Exercise** 60. Let E be an elliptic curve, P a point on E and L a line intersecting E in P. Show that the multiplicity of the intersection point is greater than 1 if and only if L is tangent to E at P.

Exercise 61. Let E be an elliptic curve. Show that E has 0, 1 or 3 rational points with vertical tangents.

Example 27. The elliptic curve over $\mathbb{F}_{13}$ defined by $Y^2 = X^3 + X + 2$ has just one rational point with vertical tangent, namely $(12, 0)$.

The elliptic curve over $\mathbb{F}_{13}$ defined by $Y^2 = X^3 + 2X + 6$ has three rational points with vertical tangents, namely $(3, 0)$, $(4, 0)$ and $(6, 0)$.

The elliptic curve over $\mathbb{F}_{13}$ defined by $Y^2 = X^3 + 2X + 4$ has no rational points with vertical tangents.

Returning to Figure 4, in the left drawing we see two lines, one of which clearly has three real points of intersection. In this case, the polynomial (12) has three real solutions. The second line seemingly has only one point of intersection. In this case, (12) has only one real zero. But it also has two complex zeros, and these zeros correspond to points of intersection, points with complex coordinates. Over a general field, all three points of intersection may have X-coordinates in an extension field.

The second drawing in Figure 4 again has two lines, one of which has two points of intersection and one which has a single point of intersection. The corresponding cubic polynomial (12) does not have complex zeros, but double or triple zeros. The intersection point has multiplicity equal to the multiplicity of the corresponding zero. We do not have three distinct points of intersection, but if we count multiplicity, we have three points of intersection.

In the third drawing in Figure 4, by considering complex coordinates, we see that both lines have two distinct points of intersection. In the fourth drawing, by counting multiplicity, we get two points of intersection. But we never get three points of intersection, because (13) is a polynomial of degree two.

It is very inconvenient that vertical lines have only two points of intersection, while non-vertical lines have three points of intersection. It would have been nice if every line intersected the curve in three points, counting multiplicity. The proper solution to this issue lies in projective geometry, but in this text we shall choose a much simpler solution.

We declare that one extra point $\mathcal{O}$ exists with the following properties:

1. The point $\mathcal{O}$ does not lie in the plane (hence has no coordinates), but lies on the curve and is a rational point.
2. Any line of the form $X = \beta$ intersects the elliptic curve in $\mathcal{O}$ with multiplicity one. No line of the form $Y = \alpha X + \beta$ intersects the curve in $\mathcal{O}$.
3. The curve has a tangent line in $\mathcal{O}$, and that line intersects the curve only in $\mathcal{O}$, with multiplicity three.

The special point $\mathcal{O}$ is often called the *point at infinity*.

As we saw in the discussion of Figure 4, considering coordinates in an algebraic closure and the notion of multiplicity somewhat simplifies the situation with regard to intersection points. Now that we have introduced the point at infinity, we get the following nice results.

Proposition 29. *Let E be an elliptic curve defined over $\mathbb{F}_p$, and let L be a line. Counting multiplicities, L intersects E in exactly three points. If two of the intersection points are rational, then the third point is also rational.*

Proof. The first claim follows trivially from Proposition 27 and the properties of $\mathcal{O}$.

If the two points are both $\mathcal{O}$, then by $\mathcal{O}$-2 and $\mathcal{O}$-3 the line must be the tangent line to E at $\mathcal{O}$, which means that the third point of intersection is again $\mathcal{O}$, which is rational.

If only one point is $\mathcal{O}$, then by $\mathcal{O}$-2 the line takes the form $X = \beta$. By (13), the other two points of intersection must be $(\beta, \pm y)$ for some $y \in \bar{\mathbb{F}}_p$. It is then clear that either both or none of these points are rational.

Finally, suppose neither of the two rational points is $\mathcal{O}$. We consider two cases. If the line is of the form $X = \beta$, then by $\mathcal{O}$-2 the third point of intersection is $\mathcal{O}$, which is rational.

Otherwise, the line is of the form $Y = \alpha X + \beta$. If the line passes through two distinct rational points, we know that $\alpha, \beta \in \mathbb{F}_p$. If the multiplicity of the intersection point is 2 or greater, Exercise 60 says that the line is a tangent, which means that $\alpha, \beta \in \mathbb{F}_p$.

This means that the polynomial (12) has coefficients in $\mathbb{F}_p$. Furthermore, two of its zeros lie in $\mathbb{F}_p$, which means that the third zero also is in $\mathbb{F}_p$. This means that the X-coordinate of the third point of intersection lies in $\mathbb{F}_p$, and since it lies on the line $Y = \alpha X + \beta$, the third point of intersection is rational. □

Proposition 30. *Let P, Q be points (not necessarily distinct) on an elliptic curve E. Then there exists a unique line L and a unique point R such that P, Q and R are the points of intersection of E and L.*

Proof. If $P = Q = \mathcal{O}$ then $\mathcal{O}$-2 and $\mathcal{O}$-3 say that $R = \mathcal{O}$.

If P and Q are distinct points and one of them is $\mathcal{O}$, then the vertical line through the other point is determined by that point. By Proposition 29 this line intersects the curve in one more point, though not necessarily distinct.

If P and Q are distinct points and neither of them is $\mathcal{O}$, the line through the points is uniquely determined. By Proposition 29 this line intersects the curve in one more point, though not necessarily distinct.

If $P = Q \neq \mathcal{O}$, then the line we are looking for must be a tangent line. By Exercise 59 the tangent line is unique, and by Proposition 29 this line intersects the curve in one more point, though not necessarily distinct. □

Example 28. Consider the elliptic curve over $\mathbb{F}_{13}$ defined by $Y^2 = X^3 + X + 2$.

The points $(1, 2)$ and $(7, 1)$ both lie on the curve. The line $Y = 2X$ is uniquely

determined by the two points. The line also intersects the curve in a third point $(9,5)$.

The point $(6,9)$ lies on the curve. The line $Y = X + 3$ is the unique tangent to the curve at $(6,9)$. The line also intersects the curve in the point $(2,5)$.

The points $\mathcal{O}$ and $(2,5)$ both line on the curve. The line $X = 2$ is, by $\mathcal{O}$-2, the unique line determined by the two points. The line also intersects the curve in a third point $(2,-5)$.

We conclude this section by saying precisely what the points on an elliptic curve are.

D **Definition 9.** The *points* on an elliptic curve E over $\mathbb{F}_p$ defined by

$$Y^2 = X^3 + AX + B, \qquad 4A^3 + 27B^2 \neq 0,$$

are the points with coordinates in the algebraic closure $\bar{\mathbb{F}}$ of $\mathbb{F}_p$ satisfying the curve equation, plus the special point $\mathcal{O}$:

$$E(\bar{\mathbb{F}}) = \{(x,y) \in \bar{\mathbb{F}}^2 \mid y^2 = x^3 + Ax + B\} \cup \{\mathcal{O}\}.$$

The $\mathbb{F}_p$-*rational* (or just rational) points on E are the points with coordinates in $\mathbb{F}_p$, plus the special point $\mathcal{O}$:

$$E(\mathbb{F}_p) = \{(x,y) \in \mathbb{F}_p^2 \mid y^2 = x^3 + Ax + B\} \cup \{\mathcal{O}\}.$$

E *Example* 29. Consider the elliptic curve over $\mathbb{F}_{13}$ defined by $Y^2 = X^3 + X + 2$. The rational points on the curve are

$$\mathcal{O}, (1,2), (1,11), (2,5), (2,8), (6,4), (6,9), (7,1), (7,12), (9,5), (9,8), (12,0).$$

Except for $\mathcal{O}$, this is exactly as in Example 23

Since the rational points on an elliptic curve will form a group that we will use for cryptography, we are very interested in how many rational points there are on an elliptic curve.

T **Fact 31** (Hasse's theorem)**.** *Let E be an elliptic curve defined over $\mathbb{F}_p$. The number of rational points on E is*

$$|E(\mathbb{F}_p)| = p + 1 - t,$$

where $|t| \leq 2\sqrt{p}$.

E *Example* 30. Consider the elliptic curve over $\mathbb{F}_{13}$ defined by $Y^2 = X^3 + X + 2$. We saw in Example 29 that there are 12 rational points on this curve, including $\mathcal{O}$. We get that $12 = 13 + 1 - 2$, so in this case $t = 2$.

E *Example* 31. Consider the elliptic curve over $\mathbb{F}_{13}$ defined by $Y^2 = X^3 + 4$. An exhaustive computation shows that there are 21 rational points on this curve, including $\mathcal{O}$. We get that $21 = 13 + 1 - (-7)$, so in this case $t = -7$.

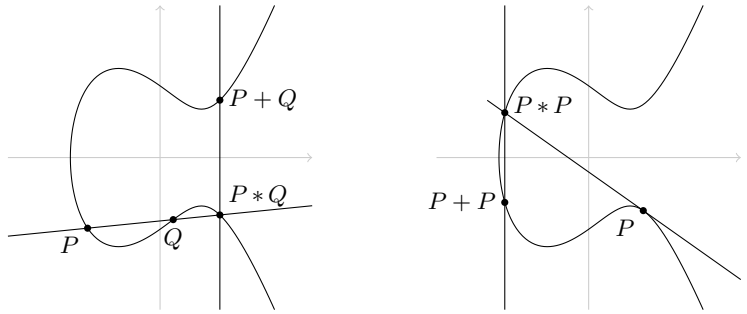

Figure 5: The group operation on elliptic curves. Addition of distinct points is shown on the left, point doubling is shown on the right.

6.1 Group Operation

We are now ready to turn the set of points on an elliptic curve into a group. We begin by defining a binary operation on the points of the elliptic curve, and then define the actual group operation in terms of the binary operation.

D **Definition 10.** Let E be an elliptic curve. We define two binary operations $*$ and $+$ on E as follows: $P * Q$ is the unique point identified by Proposition 30, and $P + Q = (P * Q) * \mathcal{O}$.

E *Example* 32. Consider the elliptic curve over $\mathbb{F}_{13}$ defined by $Y^2 = X^3 + X + 2$.

Since the unique line through $(1,2)$ and $(7,1)$ intersects the curve in a third point $(9,5)$, we find that $(1,2) * (7,1) = (9,5)$.

The line through $(9,5)$ and $\mathcal{O}$ intersects the curve in $(9,-5)$, so according to the definition $(1,2) + (7,1) = ((1,2) * (7,1)) * \mathcal{O} = (9,-5)$.

The unique line through $(6,9)$ that is tangent to the curve intersects the curve in the point $(2,5)$, so $(6,9) * (6,9) = (2,5)$.

The line through $(2,5)$ and $\mathcal{O}$ intersects the curve in $(2,-5)$, so $(6,9) + (6,9) = (2,-5)$.

The unique line through $(2,5)$ and $(2,-5)$ intersects the curve in $\mathcal{O}$. The line through $\mathcal{O}$ and $\mathcal{O}$ is the tangent at $\mathcal{O}$, which intersects only there with multiplicity 3. In other words, $(2,5) + (2,-5) = \mathcal{O}$.

We shall show that there exists an identity element for $+$, there exists inverses for $+$ and that it is commutative. There are many ways to show that $+$ is associative, but they are either tedious or advanced, so we do not prove associativity.

E *Exercise* 62. Let E be an elliptic curve and let P, Q be points on E. Show that $P * Q = Q * P$, and consequently that $P + Q = Q + P$.

T **Proposition 32.** *Let E be an elliptic curve and let P be a point on E. Then $P + \mathcal{O} = \mathcal{O} + P = P$.*

Proof. First suppose $P = \mathcal{O}$. By $\mathcal{O}$-2 and $\mathcal{O}$-3, we see that the third point of intersection identified by Proposition 30 must be $\mathcal{O}$. It follows that $\mathcal{O} * \mathcal{O} = \mathcal{O}$ and that $\mathcal{O} + \mathcal{O} = \mathcal{O}$.

Next, suppose $P \neq \mathcal{O}$. By $\mathcal{O}$-2 the line through $\mathcal{O}$ and P intersects the curve in some point $Q \neq \mathcal{O}$ and $P * \mathcal{O} = Q$. It then follows that the line through P and Q has $\mathcal{O}$ as its third point of intersection, which means that $P + \mathcal{O} = P$. □

T **Proposition 33.** *Let E be an elliptic curve and let $P = (x, y)$ be a point on E. Let $Q = (x, -y)$. Then Q is also on the curve, $P * \mathcal{O} = Q$ and $P + Q = \mathcal{O}$.*

Proof. It is immediately clear that if P is on the curve, then so is Q.

If $y = 0$, then $P = Q$ and the tangent in that point is a vertical line, which intersects the curve in $\mathcal{O}$ by $\mathcal{O}$-2, so $P * Q = \mathcal{O}$ and $P + Q = \mathcal{O}$.

If $y \neq 0$, the line through P and Q is vertical, so by $\mathcal{O}$-2 intersects the curve in $\mathcal{O}$. It follows that $P * Q = \mathcal{O}$, that $P + Q = \mathcal{O}$ and that $P * \mathcal{O} = Q$. □

T **Fact 34.** *Let E be an elliptic curve and let P, Q, R be points on E. Then $(P+Q)+R = P+(Q+R)$.*

T **Theorem 35.** *Let E be an elliptic curve. The set of points on E is a commutative group under $+$. The set of rational points is a subgroup.*

Proof. The fact that the set of points is a group follows from Propositions 32 and 33 and Fact 34. Commutativity follows from Exercise 62.

That the rational points form a subgroup follows from Proposition 30. □

We conclude this section by developing explicit formulas for computing $P + Q$.

By Proposition 32, if either point to be added is $\mathcal{O}$, the answer is the other point. Otherwise, we may assume that $P = (x_1, y_1)$ and $Q = (x_2, y_2)$.

By Proposition 33, if $x_1 = x_2$ and $y_1 = -y_2$, then the answer is $\mathcal{O}$.

If the answer has not been found, we need to find the third point of intersection of the line through P and Q. We begin by finding the slope α of the line. If $x_1 = x_2$, then $P = Q$ and we must use the tangent. The tangent line has slope

$$\alpha = \frac{3x_1^2 + A}{2y_1}.$$

If $x_1 \neq x_2$, then the slope is

$$\alpha = \frac{y_2 - y_1}{x_2 - x_1}.$$

In either case, the line's constant term is $\beta = y_1 - \alpha x_1$.

The X-coordinates of the intersection points of this line and the elliptic curve are zeros of (12). We know two of the zeros, namely x_1 and x_2, and we need to find the third zero x_3. The monic polynomial in (12) should be equal to the polynomial

$(X - x_1)(X - x_2)(X - x_3)$. Comparing the coefficients of the X^2 term, we get that $-\alpha^2 = -x_1 - x_2 - x_3$, or

$$x_3 = \alpha^2 - x_1 - x_2.$$

The Y-coordinate of the third point of intersection is $\alpha x_3 + y_1 - \alpha x_1 = \alpha(x_3 - x_1) + y_1$. By Proposition 33, the Y-coordinate of $P + Q$ is the negative of this, which is

$$y_3 = \alpha(x_1 - x_3) - y_1.$$

We summarize the above in the following result.

T **Proposition 36.** *Let E be an elliptic curve and P, Q be points on E.*

- *If $P = \mathcal{O}$, then $P + Q = Q$.*
- *If $Q = \mathcal{O}$, then $P + Q = P$.*

If neither point is $\mathcal{O}$, then let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$.

- *If $x_1 = x_2$ and $y_1 = -y_2$, then $P + Q = \mathcal{O}$.*
- *Otherwise, $P + Q = (x_3, y_3)$ with*

$$x_3 = \alpha^2 - x_1 - x_2 \qquad \text{and} \qquad y_3 = \alpha(x_1 - x_3) - y_1,$$

where

$$\alpha = \begin{cases} \frac{3x_1^2 + A}{2y_1} & x_1 = x_2, \\ \frac{y_2 - y_1}{x_2 - x_1} & x_1 \neq x_2. \end{cases}$$

E *Example* 33. Consider the elliptic curve over $\mathbb{F}_{13}$ defined by $Y^2 = X^3 + X + 2$.

We want to add $(1, 2)$ and $(7, 1)$. We compute the slope as $-1/6 = 2$, and then $x_3 = 2^2 - 1 - 7 = 9$ and $y_3 = 2(1 - 9) - 2 = 8$, that is

$$(1, 2) + (7, 1) = (9, -5).$$

We want to add $(6, 9)$ to itself (doubling). We compute the slope as $(3 \cdot 6^2 + 1)/(2 \cdot 9) = 1$, and then $x_3 = 1^2 - 6 - 6 = 2$ and $y_3 = 6 - 2 - 9 = -5$, that is

$$(6, 9) + (6, 9) = (2, -5).$$

We conclude by noting that the *exponentiation* we have studied in Section 2 and 3 now corresponds to a *point multiplication*

$$aP = \underbrace{P + P + \cdots + P}_{a \text{ terms}},$$

since the group of points on an elliptic curve is written additively instead of multiplicatively.

Even though the notation we have chosen for the elliptic curve group operation suggests addition and not multiplication, we use the words from Definition 4 and say

that the discrete logarithm of a point Q to the base P is the smallest non-negative integer a such that $Q = aP$. We write $\log_P Q = a$.

E *Exercise* 63. Redo Exercise 2 for point multiplications. Is anything different? What about Exercise 4?

6.2 Finding Suitable Curves

We have shown that the points on an elliptic curve form a commutative group, and the set of rational points is a finite commutative group. As we saw in Section 3, we need a cyclic group whose order is divisible by a large prime.

E *Exercise* 64. Show that an elliptic curve has 0, 1 or 3 rational points of order 2.
Hint: Use Exercise 61.

E *Exercise* 65. Let E be the curve defined by $Y^2 = X^3 + 12$ over the field $\mathbb{F}_{13}$. Show that E is not cyclic.

In general, the group of rational points is not cyclic, but it must contain a large cyclic subgroup.

T **Fact 37.** *Let E be an elliptic curve defined over $\mathbb{F}_p$. Then there exists n_1, n_2, where n_1 divides both n_2 and $p-1$, such that*

$$E(\mathbb{F}_p) \simeq \mathbb{Z}^+_{n_1} \times \mathbb{Z}^+_{n_2}.$$

But a large cyclic subgroup is not sufficient for our purposes, we also need to know that its order is divisible by a large prime.

T **Proposition 38.** *Let p be a prime congruent to 2 modulo 3, and let E be an elliptic curve defined by $Y^2 = X^3 + B$ over $\mathbb{F}_p$. Then $|E(\mathbb{F}_p)| = p + 1$.*

Proof. Since $p \equiv 2 \pmod 3$, 3 will be invertible modulo $p-1$. Then the map $\zeta \mapsto \zeta^3$ is invertible. If k is an inverse of 3 modulo $p-1$, then $\zeta \mapsto \zeta^k$ is the inverse map.

This means that for any value γ, the equation $\gamma = X^3 + B$ has a unique solution in $\mathbb{F}_p$, and that solution is $(\gamma - B)^k$. We conclude that for every possible Y-coordinate y,

$$((y^2 - B)^k, y)$$

is a point on the curve, and it is the only point with that Y-coordinate. There are p such points and these are all the points with coordinates, which when counting $\mathcal{O}$ makes for $p+1$ points on the curve. □

Unfortunately, curves of the form $Y^2 = X^3 + B$ are so-called *supersingular*, which for reasons we shall not consider are less suitable for our purposes than ordinary elliptic curves.

For a randomly chosen curve, the number of points is close to evenly distributed within the range given by Hasse's theorem. Since a curve with a prime (or close to prime) number of points would be reasonable, if we can determine the number of rational points on a curve, we could easily find a suitable curve.

We shall briefly sketch Schoof's algorithm for counting points on an elliptic curve. The story begins with the *Frobenius map*. Let E be an elliptic curve defined over $\mathbb{F}_p$ and define a map $\phi : E(\bar{\mathbb{F}}) \to E(\bar{\mathbb{F}})$ by

$$\phi(P) = \begin{cases} \mathcal{O} & P = \mathcal{O}, \text{ and} \\ (x^p, y^p) & P = (x, y). \end{cases}$$

E *Exercise* 66. Show that ϕ is a well-defined map.

E *Exercise* 67. Show that ϕ is a group homomorphism.

The fixed field of the map $\bar{\mathbb{F}} \to \bar{\mathbb{F}}$ given by $\alpha \mapsto \alpha^p$ is $\mathbb{F}_p$. It follows that the set of fixed points of ϕ is the set of rational points on E.

Hasse's theorem (Fact 31) says that there is a number t such that the number of rational points is $p + 1 - t$. It turns out that this number is closely related to the Frobenius map.

T **Fact 39.** *Let E be an elliptic curve defined over $\mathbb{F}_p$, and let t be the integer such that the number of rational points on E is $p + 1 - t$. Then*

$$\phi^2(P) - t\phi(P) + pP = \mathcal{O} \tag{14}$$

for any point $P \in E(\bar{\mathbb{F}})$.

It turns out that the action of the Frobenius map on points of small order is important.

D **Definition 11.** Let ℓ be an integer greater than 0. The set of *ℓ-torsion* points is

$$E[\ell] = \{P \in E(\bar{\mathbb{F}}) \mid \ell P = \mathcal{O}\}.$$

If $P \in E[\ell]$, then (14) becomes

$$\phi^2(P) - t_\ell \phi(P) + pP = \mathcal{O},$$

where $t_\ell = t \bmod \ell$. Reordering terms, we get

$$\phi^2(P) + pP = t_\ell \phi(P).$$

This is nothing more than a discrete logarithm problem, but note that the subgroup generated by $\phi(P)$ has order ℓ. If ℓ is not too big and computations on ℓ-torsion points are not too expensive, it will be possible to recover t_ℓ.

Note that $t \equiv t_\ell \pmod \ell$. If we can recover t_ℓ for many different, small primes ℓ whose product is greater than $4\sqrt{p}$, we can recover t and thereby the group order.

It still remains to show that we can find ℓ-torsion points and compute with them.

Fact 40. *Let E be an elliptic curve. There exists an efficiently computable sequence of polynomials (called* division polynomials*) $\psi_1(X,Y), \psi_2(X,Y), \ldots$ such that*

- *for any $P = (x,y) \in E(\bar{\mathbb{F}})$, $\psi_\ell(x,y) = 0$ if and only if $P \in E[\ell]$; and*
- *for odd ℓ, $\psi_\ell(X,Y)$ is a polynomial in X only, and has degree $(\ell^2-1)/2$.*

Fact 41. *Let E be an elliptic curve over $\mathbb{F}_p$. If p does not divide ℓ, then $E[\ell] \simeq \mathbb{Z}_\ell^+ \times \mathbb{Z}_\ell^+$.*

We now have a polynomial that characterizes all the X-coordinates of the ℓ-torsion points. Suppose that ℓ is an odd prime and smaller than p.

We can decide if $\psi_\ell(X)$ has rational zeros by computing $\gcd(X^p - X, \psi_\ell(X))$. It must either have 0, $(\ell-1)/2$ or $(\ell^2-1)/2$ rational zeros. Since each X-coordinate gives rise to two points, for the latter two cases we immediately know that $p+1-t$ is congruent to 0 modulo ℓ or ℓ^2, respectively.

More usually, $\psi_\ell(X)$ will not have any rational zeros. But it will usually not be irreducible. Let $f(X)$ be an irreducible factor of $\psi_\ell(X)$ of degree d. Then by constructing the extension field

$$\mathbb{F}_{p^d} \simeq \mathbb{F}_p[X]/\langle f(X)\rangle,$$

we know that the element $x \in \mathbb{F}_{p^d}$ corresponding to $X + \langle f(X)\rangle$ is a zero of $f(X)$ and hence of $\psi_\ell(X)$ and hence the X-coordinate of an ℓ-torsion point.

We now have the X-coordinate of an ℓ-torsion point, but we do not have a Y-coordinate. We can find a Y-coordinate by computing a square root of $x^3 + Ax + B$. Note that sometimes, we have to go to yet another field extension to find a square root, but this is unproblematic.

Factoring $\psi_\ell(X)$ is possible, but costly. Instead of finding an irreducible factor of $\psi_\ell(X)$, we can compute in the factor ring

$$\mathbb{F}_p[X]/\langle \psi_\ell(X)\rangle.$$

If $\psi_\ell(X)$ factors into distinct irreducibles as $f_1(X)f_2(X)\cdots f_l(X)$, then

$$\mathbb{F}_p[X]/\langle \psi_\ell(X)\rangle \simeq \mathbb{F}_p[X]/\langle f_1(X)\rangle \times \cdots \times \mathbb{F}_p[X]/\langle f_l(X)\rangle.$$

Computing in this ring is just a simultaneous computation in every possible field extension where we could have found X-coordinates for ℓ-torsion points.

Since our computations involve divisions, and our ring contains non-invertible elements, the computation may not be possible to complete. However, when we cannot find an inverse of some element, we find a divisor of $\psi_\ell(X)$. If this happens, we simply restart the computation using one of the factors of $\psi_\ell(X)$. Eventually, we must either complete the computation or find an irreducible factor of $\psi_\ell(X)$.

Going into an extension of these rings to find the Y-coordinate of the ℓ-points is also unproblematic.

The above algorithm sketch works, but is very inefficient and mostly impractical. A more careful algorithm will work significantly faster.

The most important improvement to the algorithm is that for some small primes it is relatively easy to find an irreducible factor of $\psi_\ell(X)$. Since this factor has much

smaller degree, we can work in a small field extension where arithmetic is much faster than in $\mathbb{F}_p[X]/\langle \psi_\ell(X)\rangle$. The resulting algorithm is significantly faster.

The upshot is that there are feasible algorithms that are able to compute the number of points on an elliptic curve. We will not have to count the number of points of many curves until we find one with a suitable number of points.

6.3 Discrete Logarithms

In the group $\mathbb{F}_p^*$ the group operation requires two arithmetic operations (one integer multiplication and one integer division), while finding inverses is much more costly (using the extended Euclidian algorithm). Note that division in a finite field is usually done by multiplying with inverses.

In the group $E(\mathbb{F}_p)$, Proposition 36 says that adding distinct non-inverse points requires one inversion, three multiplications and six additions. Adding a point to itself requires one inversion, two multiplications by small constants, four multiplications, one addition of a constant and four additions.

At first glance, it would seem odd to consider the elliptic curve group, since the group operation there is much more complicated than the group operation in $\mathbb{F}_p^*$. But there is one more variable to consider: the size of the underlying field. Recall that we choose the size of the group such that the discrete logarithm problem in the group is sufficiently difficult.

The methods in Section 3 work for essentially any group. For $\mathbb{F}_p^*$ we also have index calculus methods, which become significantly better than the methods in Section 3 as p grows. For most elliptic curves, there are no equivalents of the small primes, so there are no obvious, useful index calculus methods.

We briefly summarize the state of the art in elliptic curve discrete logarithm computations.

- For so-called *anomalous elliptic* curves, curves defined over $\mathbb{F}_p$ with p elements, there are very efficient algorithms for computing discrete logarithms. These curves are completely unsuitable for use in cryptography.
- For certain subgroups G, H of $E(\bar{\mathbb{F}})$, there are so-called *bilinear maps* $e : G \times H \to \mathbb{F}_{p^d}^*$ satisfying
$$e(aP, bQ) = e(P,Q)^{ab}.$$
When the field extension degree d is small, these maps can be computed easily and can sometimes be used to move a discrete logarithm problem from an elliptic curve into a finite field. Since index calculus methods can be used in finite fields, curves with low extension degree must be defined over larger finite fields, making arithmetic slower.

  So-called *supersingular curves* is one class of curves with very low extension degree.

  Note that bilinear maps can sometimes be used constructively in cryptography. But unless we need easily computable bilinear maps, curves with such maps are not useful for cryptography.

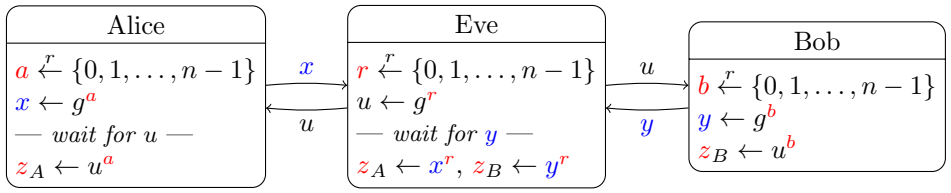

Figure 6: A man-in-the-middle attack on the Diffie-Hellman protocol, where Eve establishes separate shared secrets with Alice and Bob, who do not notice the attack.

- For many other small families of elliptic curves, there are algorithms capable of computing discrete logarithms faster than the methods from Section 3.

For most elliptic curves, the best algorithms for computing discrete logarithms are those from Section 3. Compared to $\mathbb{F}_p^*$, our elliptic curves can therefore be defined over much smaller fields where arithmetic is much faster, so even if we have to do more arithmetic operations, each group operation may be faster.

We should mention that elliptic curves over non-prime finite fields have been studied extensively. In certain cases, such fields may be more convenient than prime fields, but there have been many more advances in computing discrete logarithms for such curves. Usually, the advantage gained is outweighed by the uncertainty.

7 Active Attacks

The only attackers we have considered so far are eavesdroppers. For some communications channels, this is correct. But for most channels in use today, an attacker that can eavesdrop can also tamper with communications.

As we see in Figure 6, Diffie-Hellman on its own will not be secure in practice. We shall consider how Diffie-Hellman can be secured later.

Public Key Encryption

KG

October 24, 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.

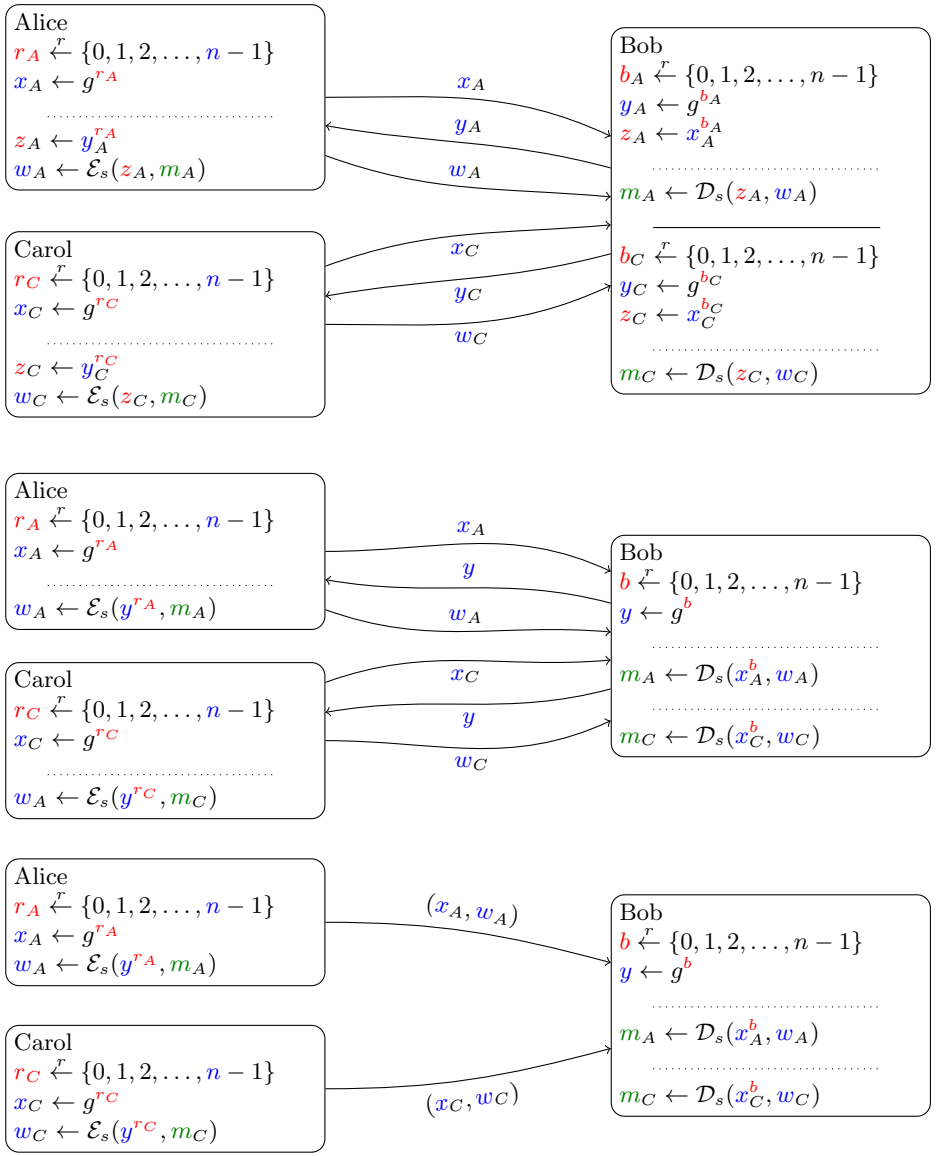

Figure 1: From Diffie-Hellman to public key encryption scheme. (top) Alice and Carol use Diffie-Hellman to establish shared secrets with Bob and send him encrypted messages. (middle) Bob uses a single random number for all the Diffie-Hellman protocol runs. (bottom) Bob publishes his Diffie-Hellman message.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

T **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. □

T **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 $\text{lcm}(p-1, q-1)$. 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.

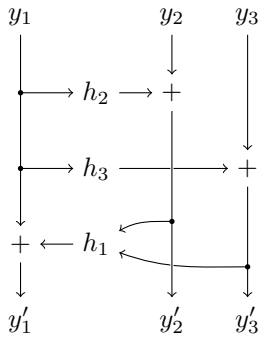

Figure 2: The permutation π can be used for RSA encryption.

> **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.

E *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 $1007 = 19 \cdot 53$.

E *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

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

T **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. □

E *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 $n = 2573$.

T **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. □

E *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.

E *Example* 8. Consider $n = 1007$.

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

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

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:

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

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.

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.

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.

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.

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. □

T **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 $(b_{i1}, b_{i2}, \dots, b_{in})$ 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

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

Finally, we note that it is easy to make sure that the magnitude of the entries $(c_{12}, \dots, c_{1n})$ 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 $i, i+1, \dots, n$ 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 $j, j+1, \dots, n$. 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 $\mathbf{v} \in \operatorname{span}\{\mathbf{b}_1^*, \ldots, \mathbf{b}_r^*\}$, $\|\mathbf{v}\| = \|\mathbf{v}\mathbf{P}\|$.
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 $\mathbf{B}\mathbf{C}^T$ is an integer matrix with integral inverse.*

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

$$\mathbf{B}\mathbf{C}^T = \mathbf{B}\mathbf{B}^T(\mathbf{B}\mathbf{B}^T)^{-1}\mathbf{U}^T = \mathbf{U}^T,$$

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

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

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 $\mathbf{x} = \mathbf{a}\mathbf{B} \in \Lambda$ 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

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

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 $\mathbf{a}\mathbf{B}$ 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

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

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 $a_0 + a_1 X + \cdots + a_{n-1} X^{n-1} \in \mathbb{Z}[X]$ and $b_0 + b_1 X + \ldots b_{n-1} X^{n-1} \in \mathbb{Z}[X]$, 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 $\mathbf{c}_i^* = \mathbf{c}_{i+1}^* + \mu_{i+1,i}\mathbf{b}_i^*$.
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.

> **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

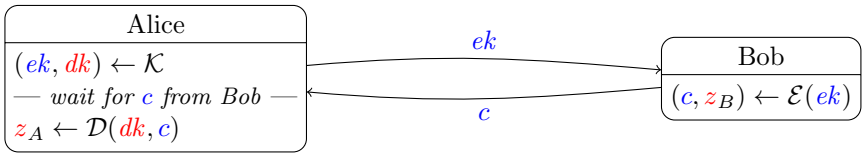

Figure 3: A key exchange protocol based on a key encapsulation mechanism $(\mathcal{K}, \mathcal{E}, \mathcal{D})$.

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.

Factoring Using Quantum Computers

KG

October 24, 2019

Contents

1 Introduction

In this note, we shall describe a machine for quickly factoring an RSA modulus n, which is the product of two large primes p and q. Let $g \in \mathbb{Z}$ be relatively prime to n.

Consider the function $f : \mathbb{Z} \to \mathbb{Z}$ given by $a \mapsto g^a \bmod n$. This function is *periodic*, and the period is the same as the multiplicative order of $g + \langle n \rangle$ in $\mathbb{Z}_n^*$. The period of this function therefore divides $(p-1)(q-1)$, and for most n and g the period will be close to $(p-1)(q-1)$.

We have previously seen that we can easily factor n if we know a multiple of $(p-1)(q-1)$. This means that if we can find the a multiple of the period of such a function, then by adapting our previous results, we can factor n.

So to meet our goal of factoring, it suffices to construct a machine that is capable of finding (a multiple of) the period of a periodic function $f : \mathbb{Z} \to \mathbb{Z}$. In the following, r will be the period of f.

2 Background

2.1 Rational Approximations

A *rational approximation* of a real number is a fraction that is close to the real number. Continued fractions are a useful tool to find rational approximations.

T **Fact 1.** *Let a, b, c, d be integers such that $|a/b - c/d| \leq 1/(2d^2)$. Then c/d is a convergent for the continued fraction expansion of a/b, and this convergent can be found using the Euclidian algorithm.*

Let N be an integer satisfying $N > n^2 > r^2$. Suppose we also have an integer k that is close to a multiple of the fraction N/r, that is, satisfying

$$k = l\frac{N}{r} + \delta_k \qquad \text{where } l, N, r \in \mathbb{Z} \text{ and } |\delta_k| \leq 1/2. \tag{1}$$

Then

$$\left|\frac{k}{N} - \frac{l}{r}\right| \leq \frac{1}{2N}.$$

This means that l/r is a rational approximation of the rational number k/N. This rational approximation can be found quickly using the Euclidian algorithm, but note that as there may be factors in common between l and r, we may not find r exactly.

This means that if we know N and find k satisfying (1), then we can often find a rational approximation l/r, which will give us r (up to cancellation of common factors).

E *Example* 1. Consider the rational number $k/N = 1365/2048$. In a procedure which is essentially the Euclidian algorithm, we get

$$\frac{1365}{2048} = \frac{1}{1 + \frac{683}{1365}} \qquad \frac{683}{1365} = \frac{1}{1 + \frac{682}{683}} \qquad \frac{682}{683} = \frac{1}{1 + \frac{1}{682}}$$

from which we get the continued fraction

$$\frac{1365}{2048} = \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \frac{1}{682}}}}.$$

We get the convergents

$$0, \frac{1}{1}, \frac{1}{2}, \frac{2}{3}, \frac{1365}{2048}$$

where we see that $2/3$ is a good rational approximation. Indeed, so would $4/6$, $8/12$, etc. also be.

E *Example* 2. Consider the real number $k/N = 1195/2048$. Following the procedure from the previous example we get

$$\frac{1195}{2048} = \frac{1}{1 + \frac{853}{1195}} \qquad \frac{853}{1195} = \frac{1}{1 + \frac{342}{853}} \qquad \frac{342}{853} = \frac{1}{2 + \frac{169}{342}}$$
$$\frac{169}{342} = \frac{1}{2 + \frac{4}{169}} \qquad \frac{4}{169} = \frac{1}{42 + \frac{1}{4}}$$

from which we get the continued fraction

$$\frac{1195}{2048} = \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{2 + \frac{1}{42 + \frac{1}{4}}}}}}.$$

We get the convergents

$$0, \frac{1}{1}, \frac{1}{2}, \frac{3}{5}, \frac{7}{12}, \frac{297}{509}, \frac{1195}{2048}$$

from which we see that $7/12$ and $297/509$ are good rational approximations.

2.2 Discrete Fourier Transform

Let $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_{N-1}) \in \mathbb{C}^N$. Then a normalized *discrete Fourier transform* of $\boldsymbol{\alpha}$ is $\boldsymbol{\beta} \in \mathbb{C}^N$ given by

$$\beta_k = \frac{1}{\sqrt{N}} \sum_j \alpha_j \exp\left(2\pi i \frac{kj}{N}\right), \qquad 0 \leq k < N.$$

Since it is normalized, it preserves the norm of vectors, so $\|\boldsymbol{\alpha}\| = \|\boldsymbol{\beta}\|$. In fact, it is a linear map given by a unitary matrix (a normalized Vandermonde matrix).

E *Exercise* 1. Let $\omega = \exp(2\pi i/N) \in \mathbb{C}$, which is a primitive Nth root of unity. Let $V = (v_{kj})$ be the Vandermonde matrix with $v_{kj} = \omega^{kj}$, $0 \leq k, j < N$.

1. Show that the $\overline{\omega^j} = \omega^{N-j}$, where $\bar{z}$ denotes complex conjugation.
2. Show that $\sum_{j=0}^{N-1} (\omega^k)^j = 0$ for any $k \not\equiv 0 \pmod{N}$.
3. Show that $VV^* = NI$, where V^* denotes the conjugate transpose of V.
4. Show that when $\boldsymbol{\beta}$ is the discrete Fourier transform of $\boldsymbol{\alpha}$, then
$$\boldsymbol{\beta} = \frac{1}{\sqrt{N}} V\boldsymbol{\alpha}.$$

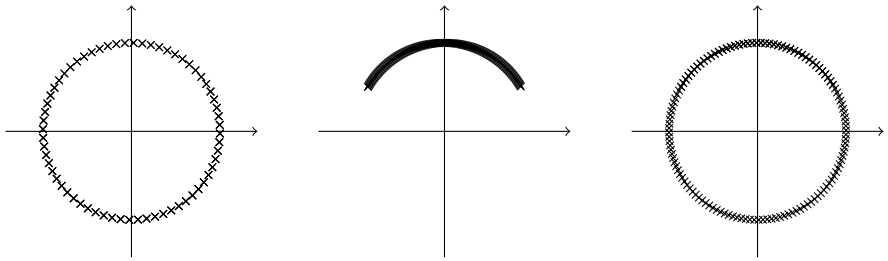

Figure 1: Plot of the terms in the final sum in (3) for $N = 2048$, $r = 12$ and $t_0 = 7$, and for $k = 1192, 1195, 1196$.

Exercise 2. Let U be an $N \times N$ unitary matrix. Show that for any vector $\boldsymbol{\alpha} \in \mathbb{C}^N$,

$$\|U\boldsymbol{\alpha}\| = \|\boldsymbol{\alpha}\|.$$

We want to study the Fourier coefficients of a very special complex vector. Let t_0 be an integer such that $0 \le t_0 < r$, and let m be minimal such that $mr + t_0 \ge N$. Let $\boldsymbol{\alpha} \in \mathbb{C}^N$ be given by

$$\alpha_k = \begin{cases} \frac{1}{\sqrt{m}} & k = t_0 + jr, \text{ and} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Note that $\|\boldsymbol{\alpha}\| = 1$ and that $1 - mr/N = 1 - (N - t_0)/N \le r/N \le 1/r$.

Applying the discrete Fourier transform gives us

$$\begin{aligned} \beta_k &= \frac{1}{\sqrt{N}} \sum_{j=0}^{m-1} \frac{1}{\sqrt{m}} \exp\left(2\pi i \frac{k(t_0 + jr)}{N}\right) \\ &= \frac{1}{\sqrt{N}} \frac{1}{\sqrt{m}} \exp\left(2\pi i \frac{kt_0}{N}\right) \sum_{j=0}^{m-1} \exp\left(2\pi i \frac{kjr}{N}\right). \end{aligned} \tag{3}$$

Example 3. Consider the vector $\boldsymbol{\alpha}$ from (2) for $N = 2048$, $r = 12$ and $t_0 = 7$. The terms of the sum in the expression for β_k from (3) has been plotted in the complex plane for $k = 1192, 1195, 1196$ in Figure 1.

Since the Fourier coefficient β_k is the sum of all the plotted complex numbers, it is easy to imagine that the sum of these complex numbers will tend to cancel out in the left and right cases, while for the middle case with $k = 1195$ there will be no cancellation. In other words, the amplitude of β_{1192} and β_{1196} will be small, while β_{1195} will be large.

We shall only care about the absolute value, which means that we shall only care about the sum factor. Furthermore, we shall only care about those k that are close to a multiple of the fraction N/r, that is those k satisfying (1). For such a k, our sum factor

becomes

$$\begin{aligned} \sum_{j=0}^{m-1} \exp\left(2\pi i \frac{kjr}{N}\right) &= \sum_{j=0}^{m-1} \exp\left(2\pi i \frac{jr}{N}\left(l\frac{N}{r} + \delta_k\right)\right) \\ &= \sum_{j=0}^{m-1} \exp(2\pi i \underbrace{jl}_{\in\mathbb{Z}}) \exp\left(2\pi i \frac{jr}{N}\delta_k\right) \\ &= \sum_{j=0}^{m-1} \exp\left(2\pi i \frac{jr}{N}\delta_k\right) = \sum_{j=0}^{m-1} \left(\exp\left(2\pi i \frac{r}{N}\delta_k\right)\right)^j \\ &= \frac{1 - \exp\left(2\pi i \frac{r}{N}\delta_k m\right)}{1 - \exp\left(2\pi i \frac{r}{N}\delta_k\right)} = S. \end{aligned}$$

Exercise 3. Let $z \in \mathbb{R}$.

1. Prove that $|1 - \exp(2\pi i z)|^2 = 4\sin^2(\pi z)$.
2. If $0 \le |z| \le \pi/2$, prove that
$$\frac{\sin z}{z} \ge \frac{2}{\pi}.$$

Recall that when z is a very small real number, $|\sin z| \le |z|$, while for $|z| < \pi/2$ we know that $|\sin z|$ is increasing. Since N is large relative to r, r/N will be small. Furthermore, $(m-1)r + t_0 \le N < mr + t_0$, which means that mr/N is slightly smaller than 1. Using Exercise 3 we get

$$|S|^2 = \frac{\sin^2\left(\pi \frac{r}{N}\delta_k m\right)}{\sin^2\left(\pi \frac{r}{N}\delta_k\right)} \ge \frac{\sin^2(\pi\delta_k)}{\sin^2\left(\pi \frac{r}{N}\delta_k\right)} \ge \frac{\sin^2(\pi\delta_k)}{(\pi\delta_k r/N)^2} = \frac{N^2}{r^2}\left(\frac{\sin\pi\delta_k}{\pi\delta_k}\right)^2 \ge \frac{N^2}{r^2}\left(\frac{2}{\pi}\right)^2.$$

This means that for the indexes k satisfying (1), the discrete Fourier coefficient satisfies

$$|\beta_k|^2 \ge \frac{1}{N}\frac{1}{m}\frac{N^2}{r^2}\frac{4}{\pi^2} = \frac{1}{mr}\frac{N}{r}\frac{4}{\pi^2} \ge \frac{1}{mr}\frac{mr}{N}\frac{N}{r}\frac{4}{\pi^2} = \frac{1}{r}\frac{4}{\pi^2}.$$

Next, we want to sum these values for all k satisfying (1), and since there are at least $r - 1$ such k, we get an approximate lower bound

$$(r-1)\frac{1}{r}\frac{4}{\pi^2} \approx \frac{4}{\pi^2}. \tag{4}$$

This result essentially says that when we sum the squared absolute values of all the Fourier coefficients, those corresponding to values of k satisfying (1) constitute a relatively large fraction of the total sum.

Example 4. Figure 2 shows a plot of the amplitude of the discrete Fourier transform of a function of the form (2). The locations of the amplitude peaks for the discrete fourier transform are given in Table 1.

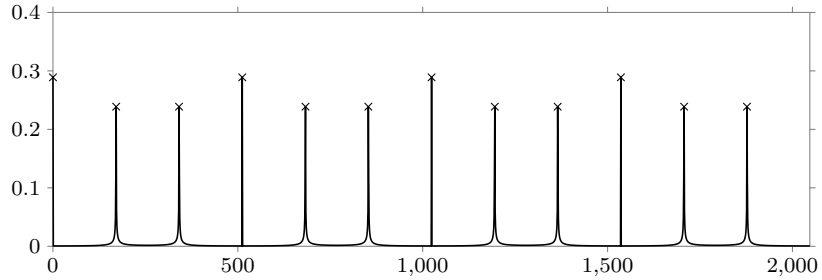

Figure 2: The amplitude of the discrete Fourier transform of a function of the form (2) with period $r = 12$ and $N = 2048$.

Table 1: The locations of the maximal amplitudes from Figure 2, with reference to (1).

k	$\lfloor k/(N/r) \rceil$	$\lvert\delta\rvert$	rat. approx.
0	0.0	0.0	-
171	1.0	0.002	1/12
341	2.0	0.002	1/6
512	3.0	0.0	1/4
683	4.0	0.002	1/3
853	5.0	0.002	5/12
1024	6.0	0.0	1/2
1195	7.0	0.002	7/12
1365	8.0	0.002	2/3
1536	9.0	0.0	3/4
1707	10.0	0.002	5/6
1877	11.0	0.002	11/12

The cumulative probability of these values is approximately 0.79. We see that for only a few of these values will the rational approximation of k/N give us the period $r = 12$, but for all except $k = 0$, it is within a small multiple. (The explanation is that the multiple l and the period r have common factors, which cancel.)

Note, however, that we cannot compute the discrete Fourier transform, since N is too large, and even if we could, we could not easily find out which values of k are included in the sum. So the discrete Fourier transform does not immediately help us find a k satisfying (1).

However, if we could arrange some probabilistic process in such a way that it will sample an integer from $\{0, 1, \ldots, N-1\}$ with probability according to the squared absolute value of the Fourier coefficients, then the above results says that it is reasonably likely that we will sample a k satisfying (1).

3 Quantum Computation

A *quantum bit* (often called *qubit*) is a physical system with two states. We interpret the two states as 0 and 1. A quantum system can be in a *superposition* of its two states, which is described by two complex numbers α and β such that $|\alpha|^2 + |\beta|^2 = 1$. The two values $|\alpha|^2$ and $|\beta|^2$ are the probabilities of measuring 0 and 1, respectively. We write the state as

$$\alpha|0\rangle + \beta|1\rangle.$$

A physical system can have more than two states, of course. If we number the states from 0 and to $N-1$, a superposition of states is described a complex vector $\boldsymbol{\alpha}$ with norm 1 and we write the state as

$$\sum_{j=0}^{N-1} \alpha_j |j\rangle.$$

Two physical systems (with N and N' states, respectively) can be *entangled*, which means that the outcome of measuring one of the systems will influence the outcome of measuring the other system. We write the system state as

$$\sum_{k=0}^{N-1} \sum_{j=0}^{N'-1} \alpha_{kj} |k\rangle|j\rangle.$$

If we measure the second system, we will get the outcome j_0 with probability

$$\sum_{k=0}^{N-1} |\alpha_{kj_0}|^2$$

and if j_0 was the outcome, we will get the quantum state

$$\sum_{k=0}^{N-1} \frac{\alpha_{kj_0}}{\sqrt{\sum_{k=0}^{N-1} |\alpha_{kj_0}|^2}} |k\rangle|j_0\rangle.$$

When later measuring the first system, the probability of outcome k_0 will be

$$\frac{|\alpha_{k_0 j_0}|^2}{\sum_{k=0}^{N-1} |\alpha_{kj_0}|^2}.$$

Multiple quantum bits can be entangled, which means that the outcome of measuring one bit may influence the outcome of measuring the other bits. In this case, a system of l qubits will have $N = 2^l$ states. A system of multiple entangled qubits is called a *quantum register*.

3.1 Computing on Quantum Registers

Computations on quantum bits can be done using *quantum gates*. Any such computation can be described by an invertible linear map that preserves the norm, that is, a unitary matrix U. That is, the transition

$$\sum_{j=0}^{N-1} \alpha_j |j\rangle \qquad \stackrel{U}{\longmapsto} \qquad \sum_{j=0}^{N-1} \beta_j |j\rangle$$

is given by by $\boldsymbol{\beta} = U\boldsymbol{\alpha}$.

Example 5. Consider a system with two states. We want to compute the function $f(z) = 1 - z$, the negation function, on this state. The unitary matrix

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

will compute exactly this function, mapping the quantum state (α_0, α_1) to (α_1, α_0).

Example 6. Consider a system of three qubits. We want to compute the function $f(z_1, z_2, z_3) = (z_1, z_2, z_1 z_2 + z_3 \bmod 2)$, that is, negate z_3 if and only if both z_1 and z_2 are set. If we consider the natural basis $|000\rangle, |001\rangle, \ldots |110\rangle, |111\rangle$, the matrix

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

will compute exactly this function, swapping the quantum states 110 and 111, while leaving all other states unchanged. (It can be shown that this gate, the Toffoli gate, is universal for classical computations. Which means that this gate is sufficient in order to do any classical computation.)

Example 7. Consider a system with two states. The gate given by the unitary matrix

$$U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

defines an operation that maps the state $|0\rangle$ to a superposition of $|0\rangle$ and $|1\rangle$, where each outcome will be equally likely. (Such gates can be used to create superpositions.)

Exercise 4. Show that the above three matrices are unitary.

Since unitary matrices are invertible, computations on quantum bits must also be invertible. But with a bit of care and effort any classical computation can be repeated on quantum bits.

In particular, we can compute the function f discussed above, even though it is not invertible. Typically, we will have two quantum registers and map $|k\rangle|j\rangle$ to $|k\rangle|j + f(k)\rangle$, which is an invertible computation, at least if we use modular addition.

If we compute on a superposition, our result will be a superposition of the function values. In the example above

$$\sum_{k=0}^{N-1} \sum_{j=0}^{N'-1} \alpha_{kj} |k\rangle|j\rangle \qquad \longmapsto \qquad \sum_{k=0}^{N-1} \sum_{j=0}^{N'-1} \alpha_{kj} |k\rangle|j + f(k)\rangle.$$

3.2 Quantum Fourier Transform

Example 6 and the discussion in the previous section show that any classical computation can be replicated on a quantum computer. However, Example 7 shows that there are operations on quantum bits that do not correspond directly to classical computations. One such operation that will be important for us is the quantum Fourier transform.

The quantum Fourier transform on a system with N states is defined by

$$\sum_{j=0}^{N-1} \alpha_j |j\rangle \qquad \stackrel{QFT}{\longmapsto} \qquad \sum_{j=0}^{N-1} \beta_j |j\rangle$$

where

$$\beta_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \alpha_j \exp(2\pi i j k / N).$$

This is of course exactly the same as our normalized discrete Fourier transform.

Unlike the discrete Fourier transform, it turns out that a quantum Fourier transform of a quantum system made up of l quantum bits (with $N = 2^l$) can be implemented using about l^2 elementary quantum gates. The reason for this is that the classical discrete Fourier transform must be computed using operations that involve all the complex numbers $\alpha_0, \alpha_1, \ldots$. The quantum Fourier transform is computed by manupulating the l quantum bits, which only indirectly manipulates the quantum probabilities $\alpha_0, \alpha_1, \ldots$.

4 Factoring Using a Quantum Computer

Our goal is now to set up a quantum system with an amplitude distribution like (2). We can then apply the quantum Fourier transform, which will result in a quantum system where we know that the likelihood of measuring a k satisfying (1) is at least $4/\pi^2$. And if we measure such a k, computing a rational approximation will give us the period of the function f, which will allow us to factor n.

We begin with a two entangled quantum registers, one with $\log_2 N$ qubits and the other with $\lceil \log_2 n \rceil$ qubits. The first register should contain a superposition of the integers $0, 1, 2, \ldots, N-1$, all with the same amplitude, while the second register should contain 0. We have the state

$$\sum_{j=0}^{N-1} \frac{1}{\sqrt{N}} |j\rangle|0\rangle.$$

Using a quantum circuit, we compute f on the first register, storing the result in the second register. Since any classical computation can be done on a quantum computer, computing f is easy. Our two quantum registers now contain a superposition of $(k, f(k))$ for $k = 0, 1, \ldots, N-1$, all being equally likely, and we have the new state

$$\sum_{j=0}^{N-1} \frac{1}{\sqrt{N}} |j\rangle|f(j)\rangle.$$

Next, we measure the second register. Suppose we measure the value s. Since we have not yet measured the first register, we do not know what it contains, but since it was entangled with the second register, every value we could possible measure must be consistent with s. Suppose t_0 is the smallest non-negative integer such that $f(t_0) = s$. Then the only values we can measure in the first register are the integers $t_0 + jr$.

Since we had a uniform probability before we measured the second register, we must have a uniform probability after measuring. We now have m possible states, so we have the state

$$\sum_{j=0}^{m-1} \frac{1}{\sqrt{m}} |t_0 + jr\rangle|s\rangle.$$

In other words, if we ignore the second register which is constant, our first register now has exactly the amplitudes given by (2).

We then use a second quantum circuit to compute the Quantum Fourier Transform on the first register. And then we measure the first register. As discussed, we will with significant probability measure a k satisfying (1), which will allow us to factor n.

Example 8. Consider $n = 35$. We choose $g = 2$ and want to compute (something close to) the period of $a \mapsto 2^a \bmod 35$. To this end, we have a quantum computer with two registers, the first of which has 2048 qubits.

We initialise the quantum registers with a superposition of all possible values in the first register and zero in the second register. We apply the exponentiation function and measure the function value in the second register. Suppose we get the answer 23, which implies that the amplitude of the collapsed first register corresponds to that from Example 3.

We then compute the quantum Fourier transform of the first register, and the end result is the amplitude shown in Example 4.

Then we measure the first register. Suppose we get the answer 1195. Rational approximation as in Example 2 gives us a period of 12.

We now compute

$$2^3 \equiv 8 \pmod{35}$$

and we immediately get that $\gcd(8 - 1, 35) = 7$.

(Even if we had measured 1365 and gotten the wrong period from the rational approximation in Example 1, we would still have factored 35. But that would have been just lucky.)

As of today, quantum computers with a few qubits capable of running this algorithm have been demonstrated. It is unclear if we will ever be able to build larger quantum computers, but for applications requiring long-term security, quantum computers will at least constitute a source of uncertainty for a long time.

Digital Signatures

KG

October 24, 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 tamper with anything sent over the channel, and even introduce her own messages. Alice wants her message to Bob to arrive without modification, or if it has been tampered with, Bob should notice.

Various solutions using public key encryption schemes are possible, but a different primitive is more convenient, namely digital signatures. The basic idea is explained and defined in Section 2. Section 3 discusses hash functions and how they can be used to make some signature schemes more convenient. Section 4 and Section 5 discuss digital signature schemes based on the RSA problem and the discrete logarithm problem, respectively. Finally, in Section 7 we use digital signatures to construct a secure version of the Diffie-Hellman protocol.

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, discrete logarithms and the RSA public key cryptosystem.

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.

2 Digital Signatures

Alice, Bob and a number of other people want to be able to send messages to each other, and they want to notice any tampering with those messages. Alice does not want to manage a long-term secret for each correspondent, so symmetric key techniques such as message authentication codes cannot be used. Alice is willing to manage public information for each correspondent.

In this situation, what is needed is digital signatures.

D **Definition 1.** A *digital signature* scheme consists of three algorithms $(\mathcal{K}, \mathcal{S}, \mathcal{V})$.

- The *key generation* algorithm $\mathcal{K}$ takes no input and outputs a *signing key* sk and a *verification key* vk. To each key pair there is an associated message set denoted by $\mathcal{M}_{sk}$ or $\mathcal{M}_{vk}$.
- The *signing* algorithm $\mathcal{S}$ takes as input a signing key sk and a message $m \in \mathcal{M}_{sk}$ and outputs a signature σ.
- The *verification* algorithm $\mathcal{V}$ takes as input a verification key vk, a message $m \in \mathcal{M}_{vk}$ and a signature σ, and outputs either 0 or 1.

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

$$\mathcal{V}(vk, m, \mathcal{S}(sk, m)) = 1.$$

We interpret a 1 from the verification algorithm as a *valid* signature, and a 0 as

an *invalid* signature. A valid signature that was created without the signing key is a *forgery*.

> **Informally:** A signature scheme is *secure* if it is hard to create a valid signature on a message without the signing key, even when you can see valid signatures on many different messages.

3 Hash Functions

A vital component for designing practical digital signature schemes is the hash function. The idea is that we can easily build signature schemes, but often we get a scheme with a very small message space. Moreover, many of the schemes we build suffer from a weakness where it is very easy to come up with signatures on random messages, even without the signing key.

If we combine our primitive signature schemes with a suitable hash function, we can extend the message space and protect against these designed-in weaknesses.

The idea for the construction is that instead of signing the message itself, we shall sign a hash of the message. Let $(\mathcal{K}, \mathcal{S}', \mathcal{V}')$ be a signature scheme and let $h : S \to T$ be a hash function such that T is a subset of the signature scheme's message space. We construct a new signature scheme $(\mathcal{K}, \mathcal{S}, \mathcal{V})$ with message space S as follows. The key generation algorithm is unchanged. The signing algorithm creates a signature of a message m under the signing key sk by computing $\mathcal{S}'(sk, h(m))$. On input of vk, m and σ, the verification algorithm outputs $\mathcal{V}'(vk, h(m), \sigma)$.

Signing a hash of the message could be a security problem if we could find two messages that have the same hash. A signature on one of the messages would also be a signature on the other message, which would be bad. Ideally, we would like the hash function to be injective, but that would not allow us to expand the message space. Instead, we shall settle for a hash function that merely "looks" injective.

D **Definition 2.** Let $h : S \to T$ be a function. A *preimage* of $t \in T$ is an element $s \in S$ such that $h(s) = t$. A *second preimage* for $s_1 \in S$ is an element $s_2 \in S$ such that $s_1 \neq s_2$ while $h(s_1) = h(s_2)$. A *collision* for h is a pair of distinct elements $s_1, s_2 \in S$ such that $h(s_1) = h(s_2)$.

The following two definitions are informal. It is possible to give precise definitions, but this is out of scope for this note.

> **Informally:** Let $h : S \to T$ be a function. We say that it is *one-way* if it is an infeasible computation to find a preimage of a random $t \in T$ and to find a second preimage for a random $s \in S$.

> **Informally:** Let $h : S \to T$ be a function. We say that it is *collision resistant* if it is an infeasible computation to find collisions for h and to find a second preimage for a random $s \in S$.

We quickly note that if you can find second preimages, you can also find collisions. The converse does not have to be true. It follows that if finding a collision is an infeasible computation, the hash function will be collision resistant and it will behave like an injective function in practice.

It would be natural if the ability to find preimages implied the ability to find second preimages. This is not true, as implied by the following exercise.

E *Exercise* 1. Let $h : S \to T$ be a hash function, and suppose that $T \subseteq S$, but $4|T| = |S|$. Let $h' : S \to \{0,1\} \times T$ be the hash function defined by

$$h'(s) = \begin{cases} (0, s) & s \in T, \\ (1, h(s)) & \text{otherwise.} \end{cases}$$

Show that for 1/4th of all elements of $\{0,1\} \times T$, it is easy to find preimages, but for none of those preimages there exists a second preimage.

We see that if our hash function is collision resistant, the hash function behaves as an injective function and the above signature scheme $(\mathcal{K}, \mathcal{S}, \mathcal{V})$ is no less secure than the original scheme $(\mathcal{K}, \mathcal{S}', \mathcal{V}')$.

If our hash function is one-way, the above scheme may actually be more secure than the original scheme.

We note that there is a different security notion for hash functions, *random-looking*, which is that the hash function should in some sense look like a typical "random" function. We do not discuss this notion further, except to note that this is different from the above notions.

3.1 Attacks

The main generic attack on hash functions is to hash random messages until a collision is found. Let $h : S \to T$ be a hash function. Choose a large number of random messages $s_1, s_2, \ldots, s_l$ and compute their hashes. We store the messages and their hashes in a list sorted by their hash values. By the birthday paradox, as soon as l is roughly $\sqrt{|T|}$, we should have a reasonable likelyhood of finding a collision.

This result gives us a minimal size for the set T, namely that $\sqrt{|T|}$ should be an infeasible computation. However, the attack described above requires a lot of memory.

E *Exercise* 2. Let $l = 10\lfloor\sqrt{|T|}\rfloor$. Let $g : T \to S$ be an injective function, and let $f : T \to T$ be the function defined by $f(t) = h(g(t))$. Let t_0 and t_0' be two distinct elements of T. Define two sequences by the equations

$$t_i = f(t_{i-1}) \qquad \text{and} \qquad t_i' = f(t_{i-1}').$$

1. Imagine that the two sequences are really sequences of random elements. Argue that with reasonable probability, $t_j' = t_i$ for some i, j smaller than l.

2. Suppose h is "random-looking". Argue that the above result should apply even when the sequences are determined solely by the random choice of t_0 and t_0', respectively.

3. Suppose $t'_j = t_l$ for some $j < 2l$, and $t'_j \neq t_0$ for any $j < l$. Show how you can find, given j, a collision in h using at most $3l$ evaluations of g and h.

4. Suppose h is "random-looking". Argue that with reasonable probablity, you can find a collision in h using about $6l$ hash evaluations.

3.2 Compression Functions

Typically, the domain of a hash function is much larger than the range. For example, $\log_2 |T|$ will typically be between one hundred and a few thousand, while $\log_2 |S|$ is 2^{64} or higher. A hash function where the domain is larger than the range, but not by much, is called a *compression function*.

We are interested in compression functions for two reasons. First of all, it is probably easier to construct compression functions than large-domain hash functions. And second, we have efficient constructions that turn secure compression functions into secure hash functions.

We begin by discussing the latter construction. So let $f : S' \to T$ be a compression function. Suppose further that there is a set $\mathcal{A}$ such that $\{0,1\} \times \mathcal{A} \times T$ is either a subset of S' or trivially injects into S'. Then we can consider the restriction of f to $\{0,1\} \times \mathcal{A} \times T$ instead.

We shall now construct a hash function $h : S \to T$. The domain S is the set of all finite sequences of elements from $\mathcal{A}$, denoted by $\mathcal{A}^*$. Let $t_0 \in T$ be a fixed element of T.

The value s we want to hash is a sequence $s_1 s_2 \dots s_L$ of elements from $\mathcal{A}$. The function h is computed using the formula

$$t_1 = f(1, s_1, t_0), \qquad t_i = f(0, s_i, t_{i-1}), \qquad 2 \leq i \leq L.$$

Then $h(s) = t_L$.

The cost (in terms of compression function evaluations) of computing h is linear in the length of the message to be hashed. If it is easy to compute f, then computing h is quite efficient.

If the compression function is one-way and collision resistant, we would like the above defined hash function to be both one-way and collision resistant.

T **Theorem 1.** *Given a collision (s, s') in the above constructed hash function, we can find a collision in the compression function f using at most $2L$ evaluations of f, where the length of s and s' is at most L.*

Proof. Let $s = s_1 s_2 \dots s_L$ and $s' = s'_1 s'_2 \dots s_{L'}$, with $L \leq L'$.

We know that

$$f(0, s_L, t_{L-1}) = f(0, s'_{L'}, t'_{L'-1}).$$

Either we have found our collision, or $s_L = s_{L'}$ and $t_{L-1} = t'_{L'-1}$. The latter means that

$$f(0, s_{L-1}, t_{L-2}) = f(0, s'_{L'-1}, t'_{L'-2}).$$

We continue in this way until either we find a collision or we reach the beginning of s. Then if $L = L'$, we must have that $s_1 \neq s'_1$ since $s \neq s'$, which gives us a collision since

$$f(1, s_1, t_0) = f(1, s'_1, t_0).$$

If $L < L'$, we must have that

$$f(1, s_1, t_0) = f(0, s'_{L'-L+1}, t'_{L'-L}),$$

which will also be a collision.

We can find this collision by first computing $t'_1, t'_2, \dots, t'_{L'-L}$, then computing the pairs $(t_1, t'_{L'-L+1}), (t_2, t'_{L'-L+2}), \dots, (t_L, t'_{L'})$. One of these pairs will be our collision. The claim follows. □

The theorem says that from any collision in the constructed hash function h, it is easy to find a collision in the compression function f. Which means that if our compression function is collision-resistant, the constructed hash function is also collision-resistant.

E *Exercise* 3. Consider the above construction. Suppose you have a "magic box" that for any $t \in T$ will provide you with a reasonable-length preimage of t under h. Show that you can use this oracle to find preimages for any $t \in T$ under f.

As for collision resistance, the consequence of the above exercise is that if we can construct a compression function where finding preimages is an infeasible computation, we can construct a hash function where finding preimages is an infeasible computation.

To conclude that the hash function is one-way, we must also consider second preimages. Unlike for preimages and collision, being able to find second preimages for h does not seem to imply the ability to find second preimages of f. But the ability to find second preimages for h implies the ability to find collisions for h, which implies the ability to find collisions for f. That is, if we can find second preimages for h, then we can find collisions in f.

This means that if f is one-way and collision-resistant, then h is one-way and collision-resistant hash function.

Note that the construction uses the 0 and the 1 to differentiate the start of the iteration. This is important in the proof, since without this differentiation, we could have run into problems when messages of different length collided.

There are other ways to construct hash functions from compression functions.

E *Exercise* 4. Suppose we have a compression function $f : \mathcal{A} \times T \to T$, and that $\{0, 1, \dots, 2^{64} - 1\}$ is a subset of $\mathcal{A}$. Let t_0 be a fixed element of T.

We define two hash functions for messages that are sequences of elements from $\mathcal{A}$ of length less than 2^{64}, using the two recursive formulas

$$t_1 = f(L, t_0), \qquad t_{i+1} = f(s_i, t_i), \qquad 1 \leq i \leq L,$$

and

$$t_i = f(s_i, t_{i-1}), \qquad 1 \leq i \leq L, \qquad t_{L+1} = f(L, t_L).$$

In either case, the hash of the message is t_{L+1}.

For each hash function, state and prove a result similar to that of Theorem 1 for that hash function.

Note that to use the first construction in Exercise 4, you must know the length of the message before you begin hashing it. There are reasonable cases where you want to begin hashing a message before you know the entire message, and in particular before you know the length of the entire message.

Hash functions are used for many things in cryptography, and frequently the security requirements are different from what we need for digital signatures.

One example is to use a hash function as a message authentication code, simply by computing $\mu(k, m) = h(k||m)$, where $k||m$ denotes the concatenation of the key and the message. As the following exercise shows, our construction cannot be used like this.

E *Exercise* 5. Let $h : \mathcal{A}^* \to T$ be the hash function constructed at the start of the section. Suppose you are given a value t such that $t = h(m)$. Show that you can easily compute $h(m||m')$ for any m', even when you do not know m, only t. (Here, $m||m'$ denotes the concatenation of m and m'.)

3.3 Constructing a Compression Function

Let G be a cyclic group of prime order n, and let x and y be non-zero elements. Then we can construct a compression function $f_{x,y} : \{0, 1, 2, \ldots, n-1\} \times \{0, 1, 2, \ldots, n-1\} \to G$ as

$$f_{x,y}(u, v) = x^u y^v.$$

When x and y are clear from context, we shall write simply f for $f_{x,y}$.

If it is hard to compute discrete logarithms in G, then it is hard to find both preimages and collisions for this compression function, provided x and y have been chosen at random from G.

T **Theorem 2.** *Suppose we know a collision $((u, v), (u', v'))$ for f. Then we can compute $\log_x y$ using 3 arithmetic operations.*

Proof. If we have a collision, we know that

$$x^u y^v = x^{u'} y^{v'}.$$

Since $(u, v) \neq (u', v')$ and the above equation holds, we have that $u \neq u'$ and $v \neq v'$. This means that

$$y = x^{-(u-u')(v-v')^{-1}}.$$

The claim follows. □

E *Exercise* 6. Suppose you have a "magic box" that for any x, y, z of your choice will find one preimage of z under the hash function $f_{x,y}$. Explain how you can use this "magic box" to compute discrete logarithms in G.

Using this compression function and the construction from the previous section, we have a one-way and collision-resistant hash function. However, this hash function is of theoretical interest only, since we have much faster constructions that also have other interesting properties.

4 RSA Signatures

We briefly recall the textbook RSA public key cryptosystem. The key generation algorithm chooses two primes p and q and finds e and d such that $ed \equiv 1 \pmod{\text{lcm}(p-1, q-1)}$. The encryption key is (n, e), where $n = pq$ and the decryption key is (n, d).

To encrypt a message $m \in \{0, 1, \ldots, n-1\}$, we compute $c = m^e \bmod n$. To decrypt a ciphertext c, we compute $m = c^d \bmod n$.

It turns out that we can construct a very simple signature scheme based on this. The *textbook RSA* signature scheme $(\mathcal{K}, \mathcal{S}, \mathcal{V})$ 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 $ed \equiv 1 \pmod{\text{lcm}(p-1, q-1)}$. Finally it outputs $vk = (n, e)$ and $sk = (n, d)$. The message set associated to vk is $\{0, 1, 2, \ldots, n-1\}$.
- The *signing* algorithm $\mathcal{S}$ takes as input a signing key (n, d) and a message $m \in \{0, 1, 2, \ldots, n-1\}$. It computes $\sigma = m^d \bmod n$ and outputs the signature σ.
- The *verification* algorithm $\mathcal{V}$ takes as input a verification key (n, e), a message $m \in \{0, 1, 2, \ldots, n-1\}$ and a signature $\sigma \in \{0, 1, 2, \ldots, n-1\}$. It outputs 1 if $\sigma^e \equiv m \pmod n$, otherwise 0.

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

4.1 Attacks

As for textbook RSA public key encryption, as long as the RSA modulus n chosen by the key generation algorithm is hard to factor, the above digital signature scheme is useful. But it is not entirely secure.

E *Exercise* 8. Suppose $(n, 3)$ is a verification key. Forge a signature on the message 8.

E *Exercise* 9. Suppose (n, e) is a verification key. Explain how to create a random message with a forged signature.

Malleability Just like the RSA encryption scheme, the RSA signature scheme is malleable, and this can be used to create forgeries.

E *Exercise* 10. Suppose you have two messages m, m' and signatures σ, σ' on those messages under the verification key (n, e). Show how to construct a signature on the product $mm' \bmod n$.

E *Exercise* 11. Suppose Eve wants to have Alice' signature on a message m. Suppose also that she is capable of getting Alice to sign any other message. Show how Eve can use this to forge a signature on m.

Short Messages We want to use the idea from Section 3 with a hash function $h : S \to T$ that is both collision resistant and one-way, and where T is a set of integers all smaller than any RSA modulus we choose. This *hashed RSA* signature scheme works as follows.

- The *key generation* algorithm is exactly the same as the RSA key generation algorithm.
- The signing algorithm $\mathcal{S}$ takes a signing key $sk = (n, d)$ and a message $m \in S$ as input. It computes $\sigma = (h(m))^d \bmod n$ and outputs the signature σ.
- The verification algorithm $\mathcal{V}$ takes as input a verification key (n, e), a message $m \in S$ and a signature $\sigma \in \{0, 1, 2, \ldots, n-1\}$. It outputs 1 if $\sigma^e \equiv h(m) \pmod n$, otherwise 0.

E *Exercise* 12. Explain why the attacks from Exercises 8 and 9 fail against this scheme.
Hint: The hash function is one-way.

E *Exercise* 13. Suppose that the hash function used is also random-looking. Explain why the attacks from Exercises 10 and 11 become much more difficult.

However, most practical hash functions have outputs that are very short relative to an RSA modulus, and this allows us to develop an attack.

E *Exercise* 14. Let (n, e) be a verification key with corresponding signing key (n, d). Suppose you have messages $m_1, m_2, \ldots, m_l$, and integers $\ell_1, \ell_2, \ldots, \ell_l$, $\sigma_1, \sigma_2, \ldots, \sigma_l$ and s_{ij}, $1 \le i, j \le l$, such that

$$h(m_i) = \prod_{j=1}^{l} \ell_j^{s_{ij}}, \qquad 1 \le i \le l \text{ and}$$
$$h(m_i) \equiv \sigma_i^e \pmod n.$$

We shall assume that the matrix $\mathbf{S} = (s_{ij})$ is invertible modulo e, and that $R = (r_{ki})$ is an inverse modulo e.

1. Show that
$$\sum_{i=1}^{l} r_{ki} s_{ij} = \delta_{kj} + t_{kj} e$$
where $\delta_{kj} = 1$ if $k = j$, otherwise $\delta_{kj} = 0$.

2. Show that
$$\prod_{i=1}^{l} \sigma_i^{r_{ki}} \equiv \ell_k^d \prod_{j=1}^{l} \ell_j^{t_{kj}} \pmod n.$$

3. Explain how we can easily compute $\ell_k^d \bmod n$ from the above.

4. Suppose you are given a message m such that
$$h(m) = \prod_{i=1}^{l} \ell_i^{u_i}.$$
Explain how you, given the above, can easily compute $h(m)^d \bmod n$.

If we let $\ell_1, \ell_2, \ldots, \ell_l$ be small primes, then if our hash function h has small integers as output, we can quickly find messages $m, m_1, m_2, \ldots, m_l$ such that their hashes are products of powers of our small primes. Which means that if we get signatures on $m_1, m_2, \ldots, m_l$, we can construct a forgery on m.

4.2 Secure Variants

It turns out that it is quite easy to fix the hashed RSA signature scheme discussed above. All we need is a random-looking, one-way, collision-resistant hash function whose domain is almost all of $\{0, 1, 2, \ldots, n-1\}$. In which case the hashed RSA scheme is secure, and is known as the *full domain hashed RSA* signature scheme, or *RSA-FDH*.

E *Exercise* 15. Explain why the attack from Exercise 14 fails against RSA-FDH.

5 Schnorr Signatures

In this section, we shall develop the well-known Schnorr signature scheme. While the Schnorr scheme is not used much as a signature scheme, its development is interesting and many other signature schemes are very similar to the Schnorr system.

For the remainder of this section, let G be a group of prime order n, and let g be a generator.

5.1 How to Prove That You Know a Secret

We begin with a very different question. Suppose Alice knows a secret number $a \in \{0, 1, 2, \ldots, n-1\}$. Bob does not know the secret number, but he knows $x \in G$ such that $x = g^a$. Alice wants to convince Bob that she really knows a.

Of course there is an adversary. Eve may want to cheat Bob by pretending to know a. Or Eve may want to cheat Alice by somehow learning a.

The latter point explains why Alice cannot convince Bob simply by revealing a to Bob. While Bob is honest, Alice may at some point in time also want to convince Eve that she knows a, after which Eve could cheat Bob by pretending to know a.

One thing Alice could do was to choose a random number r, compute $\alpha = g^r$ and $\gamma = r + a \bmod n$, and then show Bob α and γ. Bob accepts that Alice knows a if

$$g^\gamma \stackrel{?}{=} \alpha x.$$

The idea is that the above protocol does not reveal a to Bob, because Alice just as well could choose a random γ and compute α as $g^\gamma x^{-1}$.

E *Exercise* 16. Explain how Eve can choose α and γ to convince Bob that she knows a, even though she only knows x.

Hint: Use the above explanation of why the scheme does not reveal a.

We can improve on this procedure as follows. Instead of showing Bob both α and γ at once, Alice first shows Bob α. Then Bob is allowed to choose if he wants to see $\gamma = r$ or $\gamma = r + a \bmod n$. He accepts that Alice knows a if

$$g^\gamma \stackrel{?}{=} \alpha \qquad \text{or} \qquad g^\gamma \stackrel{?}{=} \alpha x, \text{ respectively.}$$

Note that we can encode Bob's choice β as a 0 or a 1, in which case the above formulas can be reduced to

$$\gamma = r + \beta a \bmod n \qquad \text{and} \qquad g^\gamma \stackrel{?}{=} \alpha x^\beta. \tag{1}$$

E *Exercise* 17.

1. Suppose Bob tells Eve what his choice β will be before Eve chooses α. Explain how Eve can choose α and γ to convince Bob that she knows a, even though she only knows x.

   If Bob does not tell Eve his choice early, explain why Eve can guess his choice, proceed as above and successfully cheat Bob, all with probability 1/2.

2. Suppose that Eve has chosen α and that she knows the correct response to make Bob accept, regardless of Bob's choice. That is, Eve knows γ_0 and γ_1 such that $g^{\gamma_\beta} = \alpha x^\beta$. Show that Eve can easily compute a from γ_0 and γ_1.

We wanted to ensure that if Alice runs this protocol with Eve, then Eve learns nothing about Alice's secret. We shall argue that if Eve can learn something from Alice, she can learn the same thing without Alice.

So suppose Eve gets α from Alice, chooses β, receives γ and from that exchange learns something about a.

Now Eve decides to do without Alice. Instead, she makes a guess β' at what challenge she will choose upon seeing α. Then she proceeds according to the first part of Exercise 17. With probability 1/2, she will guess her choice of challenge correctly, in which case her conversation would proceed exactly as if she were talking to Alice, which means that she would learn something about a. Of course, with probability 1/2, Eve will not guess correctly, so she may not learn anything, but in this case she can just try again.

We also wanted to ensure that Eve cannot cheat Bob. From the above exercise we know that Eve can successfully pretend to know a with probability 1/2, but unless she knows a, she cannot succeed with any greater probability.

It follows that Alice can convince Bob that she almost certainly knows a by repeating the above protocol many times. For each repetition, Eve would have probability 1/2 of cheating successfully, but for k repetitions her success probability sinks to 2^{-k}.

Doing k repetitions is quite inefficient, of course. Instead, we can do something slightly different. The problem is that Eve can guess Bob's choice and choose α based on that. But note that in (1), there is nothing that forces β to be just 0 or 1.

The protocol can then work as follows.

1. Alice chooses r uniformly at random from $\{0, 1, 2, \ldots, n-1\}$, computes $\alpha = g^r$, and sends α to Bob.

2. Bob chooses β uniformly at random from $\{0, 1, 2, \ldots, 2^k - 1\}$ and sends β to Alice.

3. Alice computes $\gamma = r + \beta a \bmod n$ and sends γ to Bob.

Bob accepts that Alice knows a if

$$g^\gamma \stackrel{?}{=} \alpha x^\beta.$$

By the above arguments, the probability that Eve cheats Bob should not be much larger than 2^{-k}. One problem is that our argument for why Eve does not learn anything from Alice was exactly the argument that proved that Eve could cheat Bob. Which means that strictly speaking, we no longer have an argument for why Eve will not learn anything by talking to Alice.

However, if Eve chooses her challenge without looking at Alice's α, then the argument from the first part of Exercise 17 applies, and we can use it to show that in this case Eve cannot learn anything about Alice's message.

Unfortunately, for the same reason, Bob cannot reveal his challenge before Alice reveals her α. The question is, how can we force Bob to choose his challenge before Alice reveals her α, but without Bob revealing his challenge?

E *Exercise* 18. Let $h : S \to T$ be a hash function such that $\{0, 1, 2, \ldots, 2^{2k} - 1\} \times \{0, 1, 2, \ldots, 2^k - 1\}$ is a subset of the domain.

Suppose Bob chooses t and β and computes $\omega = h(t, \beta)$. He sends ω to Alice. The protocol then proceeds by Alice sending α to Bob, who responds with his already chosen β and the random number t. Alice verifies that ω equals $h(t, \beta)$ and responds with the correct γ, which Bob verifies as usual.

Under reasonable assumptions on the hash function h, it can be shown that ω does not reveal anything about β, and that Bob cannot find different t', β' such that $h(t', \beta') = \omega$.

Argue why Eve cannot cheat Alice (to learn something about a) or Bob (to convince him that Eve knows a) using the above protocol.

A different approach can be used if we have a "random-looking" hash function $h : S \to T$ where $G \times G$ is a subset of S and $T = \{0, 1, \ldots, 2^k - 1\}$. Instead of choosing a random challenge, Bob can instead compute the challenge as $\beta = h(x, \alpha)$.

Since Eve no longer chooses her challenge when trying to cheat Alice, Eve will not be looking at α before deciding on her challenge, so as we have argued above, she should not learn anything new.

When Eve is trying to cheat Bob, however, she can know what challenge Bob will choose without actually sending α to Bob. She cannot know β until after she has chosen α, but she will still have the ability to look at many αs with corresponding βs, before she sends one α to Bob. While this does give her increased power, it can easily be neutralized by increasing k.

Of course, if Eve can compute β before sending α to Bob, so can Alice. We can therefore greatly simplify the process. Alice chooses r, computes $\alpha = g^r$, $\beta = h(x, \alpha)$, and $\gamma = r + \beta a \bmod n$. She then sends α, β and γ to Bob. Bob verifies that

$$\beta \stackrel{?}{=} h(x, \alpha) \qquad \text{and} \qquad g^\gamma \stackrel{?}{=} \alpha x^\beta.$$

An equivalent verification equation can be

$$\alpha \stackrel{?}{=} g^\gamma x^{-\beta}.$$

If the hash function is collision resistant (and a "random-looking" hash function should be), this equation will hold in practice if and only if

$$h(x, g^\gamma x^{-\beta}) \stackrel{?}{=} \beta.$$

This gives us further scope for simplification and making the formulas tidier. The process now works as follows. Alice chooses r, computes $\alpha = g^r$, $\beta = h(x, \alpha)$ and $\gamma = r - \beta a \bmod n$. She then sends β and γ to Bob. Bob verifies that

$$h(x, g^\gamma x^\beta) \stackrel{?}{=} \beta.$$

5.2 Schnorr Signatures

The Schnorr signature scheme is based on the ideas on how to prove that you know something, but the proofs are augmented by including something extra in the hash that generates the challenge.

Suppose $\mathcal{P}$ is a set of messages and we have a "random-looking" hash function $h : S \to T$, where $G \times G \times \mathcal{P}$ is a subset of S and $T = \{0, 1, \ldots, 2^k - 1\}$.

The Schnorr signature scheme $(\mathcal{K}, \mathcal{S}, \mathcal{V})$ works as follows.

- The *key generation* algorithm $\mathcal{K}$ samples a number a uniformly at random from the set $\{0, 1, 2, \ldots, n-1\}$. It computes $x = g^a$ and outputs $vk = x$ and $sk = a$. The message set associated to vk is $\mathcal{P}$.
- The *signing* algorithm $\mathcal{S}$ takes as input a signing key a and a message $m \in \mathcal{P}$. It samples a number r uniformly at random from the set $\{0, 1, 2, \ldots, n-1\}$. It computes $\alpha = g^r$, $\beta = h(x, \alpha, m)$ and $\gamma = r - \beta a \bmod n$. It outputs the signature (β, γ).
- The *verification* algorithm $\mathcal{V}$ takes as input a verification key x, a message $m \in \mathcal{P}$ and a signature (β, γ). It outputs 1 if

  $$h(x, g^\gamma x^\beta, m) \stackrel{?}{=} \beta,$$

  otherwise 0.

E *Exercise* 19. The above is an informal description of a signature scheme. Implement the three algorithms $\mathcal{K}$, $\mathcal{S}$ and $\mathcal{V}$. Show that the triple $(\mathcal{K}, \mathcal{S}, \mathcal{V})$ is a signature scheme.

The Schnorr signatures and related schemes are famously sensitive to random number generation, as the following exercise shows.

E *Exercise* 20. Let x be a verification key for the Schnorr signature scheme. Suppose that under this verification key, (β, γ) is a valid signature on m, and (β', γ') is a valid signature on m', $m \neq m'$. Suppose further that

$$g^\gamma x^\beta = g^{\gamma'} x^{\beta'}.$$

Explain why the existence of these two signatures suggest a malfunction in random number generation, and show how to recover the signing key corresponding to x using only a handful of arithmetic operations.

5.3 The Digital Signature Algorithm

The Digital Signature Algorithm is a variant of the Schnorr signature scheme. We shall describe two variants of DSA, to show again how a hash function can improve both the practicality and security of a signature scheme.

Let $f : G \to \{0, 1, \ldots, n-1\}$ be a "random-looking" hash function. Our first signature scheme $(\mathcal{K}, \mathcal{S}, \mathcal{V})$ works as follows.

- The *key generation* algorithm $\mathcal{K}$ is the same as for the Schnorr signature scheme.
- The *signing algorithm* $\mathcal{S}$ takes as input a signing key a and a message $m \in \{0, 1, \ldots, n-1\}$. It samples a number r uniformly at random from the set $\{0, 1, \ldots, n-1\}$. It computes $\beta = f(g^r)$ and

  $$\gamma = r^{-1}(m + \beta a) \bmod n.$$

  It outputs the signature (β, γ).
- The *verification algorithm* $\mathcal{V}$ takes as input a verification key x, a message $m \in \{0, 1, \ldots, n-1\}$ and a signature (β, γ). It outputs 1 if

  $$f(g^{m\gamma^{-1}} x^{\beta\gamma^{-1}}) \stackrel{?}{=} \beta,$$

  otherwise 0. (Note that the exponent arithmetic happens modulo n.)

E *Exercise* 21. The above is an informal description of a signature scheme. Implement the three algorithms $\mathcal{K}$, $\mathcal{S}$ and $\mathcal{V}$. Show that the triple $(\mathcal{K}, \mathcal{S}, \mathcal{V})$ is a signature scheme.

E *Exercise* 22. Show how you can create a forgery for a random message for this scheme.

Hint: Choose u and v. Compute $\beta = f(g^u x^v)$. Now solve $v \equiv \beta\gamma^{-1} \pmod n$ and $u \equiv m\gamma^{-1} \pmod n$.

E *Exercise* 23. Modify the above scheme to accept messages from S, by using a hash function $h : S \to \{0, 1, \ldots, n-1\}$. Suppose h is one-way and collision resistant. Explain how this stops the attack from Exercise 22.

6 Hash-based Signatures

Signature schemes based on factoring (Section 4) or discrete logarithms (Section 5) are all vulnerable to Shor's algorithm if someone ever builds a sufficiently large and reliable quantum computer. For encryption or key exchange algorithms, potential future quantum computers will often be a problem, since today's adversaries may be storing today's intercepted ciphertexts so that they can read the corresponding messages after they have built a sufficiently large quantum computer. The quantum computer threat is less acute for many applications of digital signatures, since being able to forge a signature in a decade or two will not compromise today's use of the system. However, sufficiently large quantum computers may be built, so it makes sense to prepare by designing quantum-safe signature schemes.

6.1 Lamport's One-time Signatures

We begin by describing Lamport's *one-time* signature scheme, based on a hash function $h : S \to T$. Lamport's one-time signature scheme $(\mathcal{K}_1, \mathcal{S}_1, \mathcal{V}_1)$ works as follows.

- The *key generation* algorithm $\mathcal{K}_1$ samples L pairs of elements (s_{i0}, s_{i1}) from S^2 , $i = 1, 2, \ldots, L$. It computes $t_{ij} \leftarrow h(s_{ij})$ for $i \in \{1, 2, \ldots, L\}$, $j \in \{0, 1\}$. It then outputs $vk = ((t_{10}, t_{11}), \ldots, (t_{L0}, t_{L1}))$ and $sk = ((s_{10}, s_{11}), \ldots, (s_{L0}, s_{L1}))$.
- The *signing* algorithm $\mathcal{S}_1$ takes as input a signing key $sk = ((s_{10}, s_{11}), \ldots, (s_{L0}, s_{L1}))$ and a message $m = m_1 m_2 \ldots m_L \in \{0, 1\}^L$. The signature is $(s_{1,m_1}, s_{2,m_2}, \ldots, s_{L,m_L})$.
- The verification algorithm $\mathcal{V}_1$ takes as input a verification key $vk = ((t_{10}, t_{11}), \ldots, (t_{L0}, t_{L1}))$, a message $m = m_1 m_2 \ldots m_L \in \{0, 1\}^L$ and a signature $(u_1, u_2, \ldots, u_L)$. It outputs 1 if $t_{i,m_i} = h(u_i)$ for $i = 1, 2, \ldots, L$, otherwise 0.

Remark. Obviously, the scheme can be augmented with another hash function $h' : \mathcal{P} \to \{0, 1\}^L$ to increase the size of the plaintext set. This is secure if the hash function h' is collision resistant.

E *Exercise* 24. A common hash function will have $T = \{0, 1\}^{256}$. If we use this hash function with Lamport's scheme, what is the length of a signature (in bits)? And what is the length of the secret key (in bits)? Compare with the length of the signatures of RSA signatures (suppose $n \approx 2^{2048}$) and Schnorr signatures (suppose $p \approx 2^{256}$).

E *Exercise* 25. The above is an informal description of a signature scheme. Implement the three algorithms $\mathcal{K}_1$, $\mathcal{S}_1$ and $\mathcal{V}_1$. Show that the triple $(\mathcal{K}_1, \mathcal{S}_1, \mathcal{V}_1)$ is a signature scheme.

E *Exercise* 26. Lamport's scheme is a one-time scheme. Show that if you see signatures on two messages that differ in at least two positions, you can create a forgery.

Remark. The Lamport scheme's verification key is quite large, since two hash values are needed to verify a single bit. One way to reduce its size is to use a hash function $h : T \to T$ and a pair of *hash chains* of length k to sign an integer in $\{0, 1, \ldots, k\}$.

An ith preimage of t is a value s such that

$$\underbrace{(h \circ h \circ \cdots \circ h)}_{i \text{ times}}(s) = t.$$

Given two hash values t and t', we can encode an integer i as an ith preimage of t and an $(k - i)$th preimage of t'.

In this way, at the cost of $2k$ hash computations during key generation and k hash computations during signing and verification, we can sign k distinct values which makes signatures much shorter.

Remark. A one-time scheme $(\mathcal{K}_1, \mathcal{S}_1, \mathcal{V}_1)$ is of limited use. One way to make this scheme more useful is to use a tree of signatures. Each leaf node contains a key pair. Each internal node in the tree contains a key pair and a signature on the verification keys of the child nodes.

The verification key would be the root node's verification key. To sign a message, we sign it using one of the leaf node signing keys. The total signature consists of the one-time signature along with all the verification keys and one-time signatures needed to connect the message to the root verification key.

There are a number of problems with this approach. Even though the depth of the tree is logarithmic in the total number of messages we want to sign, signatures are very long. Also, we need to keep track of which keys have been used, which means that the system is stateful. We also need to keep track of many signing keys, which means that the system state is large.

6.2 Merkle Signatures

A one-time scheme $(\mathcal{K}_1, \mathcal{S}_1, \mathcal{V}_1)$ is of limited use. However, we can use a *Merkle tree* to turn the one-time scheme into a somewhat more practical scheme that can sign many messages. The idea is to create a hash tree (*Merkle tree*) where each internal node contains the hash of its children, while each leaf node contains the hash of a verification key.

We need to define an indexing scheme for the nodes in a binary tree. We shall index each node by its level and another integer, and we define the indexing recursively. Let $(0, 0)$ be the index of the root node. Then if a node has index (i, j), its left child will have index $(i + 1, 2j)$, while its right child will have index $(i + 1, 2j + 1)$. An index (i, j) will therefore satisfy $0 \leq j < 2^i$.

Note that the index (i, j) describes the path to the node from the root node, by considering the j written as an i-digit binary number. We begin at the most significant digit and interpret a 0 as "left" and a 1 as "right". The indexes of the nodes in the path to (k, j) will be $(0, 0), (1, \lfloor j/2^{k-1} \rfloor), (2, \lfloor j/2^{k-2} \rfloor), \ldots, (k - 1, \lfloor j/2 \rfloor), (k, j)$.

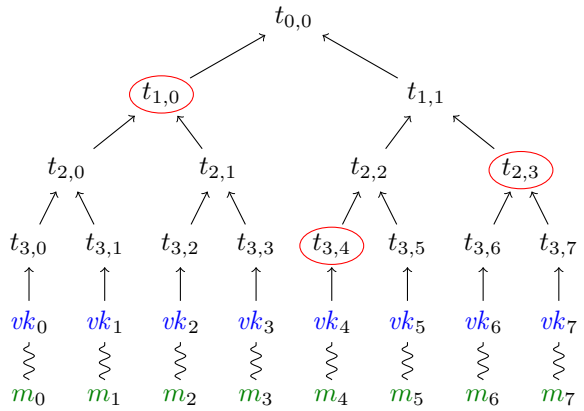

Figure 1: A Merkle tree with depth 3, allowing for up to 8 signed messages, signed with the one-time signature keys in the leaf nodes. Here $t_{3,j} = h(vk_j)$ and $t_{i,j} = h(t_{i+1,2j}, t_{i+1,2j+1})$. The hash values needed to verify that the one-time verification key vk_5 is part of a tree with root $t_{0,0}$ are circled. Given these hash values, the hash values on the path to the root can be computed.

It is convenient to define some notation for the tree. We denote the node on the ith level in the path to (k, j) as $j_i = \lfloor j/2^{k-i} \rfloor$. Since the left child always has an even index, while the right child has an odd index, the index of a nodes' sibling, as well as the left and right node in a sibling pair, is given by

$$sib(j) = \begin{cases} j-1 & j \text{ odd}, \\ j+1 & j \text{ even}, \end{cases} \qquad left(j) = \begin{cases} j & j \text{ even}, \\ j-1 & j \text{ odd}, \end{cases} \qquad right(j) = left(j) + 1.$$

If the hash function is collision resistant, then the root node in practice defines which verification keys are attached to the leaf nodes. In order to verify that the verification key vk_j is attached to the leaf node (k, j), we need only compute the hashes on the path from the root to (k, j) and verify that this hash chain is consistent with the hash on the root node. However, to recompute the hash chain, we need the hashes stored on the siblings of the nodes in the path.

Merkle's signature scheme $(\mathcal{K}_2, \mathcal{S}_2, \mathcal{V}_2)$ based on a hash function h and a one-time signature scheme $(\mathcal{K}_1, \mathcal{S}_1, \mathcal{V}_1)$ works as follows for tree depth k.

- The *key generation* algorithm $\mathcal{K}_2$ uses $\mathcal{K}_1$ to generate 2^k key pairs $(vk_0, sk_0), \ldots, (vk_{2^k-1}, sk_{2^k-1})$. It then computes the hashes on the internal nodes in the Merkle tree as

$$t_{i,j} = \begin{cases} h(t_{i+1,2j}, t_{i+1,2j+1}) & i = 0, 1, \ldots, k-2, \\ h(vk_{2j}, vk_{2j+1}) & i = k-1. \end{cases} \tag{2}$$

  It then outputs $vk = t_{0,0}$ and $sk = ((vk_0, sk_0), \ldots, (vk_{2^k-1}, sk_{2^k-1}))$.

- The *signing* algorithm $\mathcal{S}_2$ takes as input a signing key $sk = ((vk_0, sk_0), \ldots, (vk_{2^k-1}, sk_{2^k-1}))$ and a message m. It chooses an index j and signs the message m as

$$\sigma_1 = \mathcal{S}_1(sk_j, m).$$

  Then it recomputes the hash values in the Merkle tree using (2). The signature is $(j, \sigma_1, vk_j, t_{1,sib(j_1)}, t_{2,sib(j_2)}, \ldots, t_{k-1,sib(j_{k-1})}, t_{k,sib(j)})$.

- The *verification* algorithm $\mathcal{V}_2$ takes as input a verification key $vk = t_{0,0}$, a message m and a signature $\sigma = (j, \sigma_1, vk, s_1, s_2, \ldots, s_k)$. It verifies that $\mathcal{V}_1(vk_j, m, \sigma_1) = 1$, and then computes a subset of the Merkle tree hashes as $t'_{k,j} = h(vk)$, $t'_{i,sib(j)} = s_i$ and $t'_{i-1,j_{i-1}} = h(t'_{i,left(j_i)}, t'_{i,right(j_i)})$. Finally, it outputs 1 if $t'_{0,0} = t_{0,0}$.

E *Exercise* 27. Suppose we have a hash function will have $T = \{0,1\}^{256}$, and that we use this hash function and Lamport's one-time signatures (using the same hash function) with the above Merkle signatures. How long would a signature be (in bits), as a function of the number of messages that can be signed? How many hash function computations would be required during key generation. What would be the size of the secret key?

E *Exercise* 28. The above is an informal description of a signature scheme. Implement the three algorithms $(\mathcal{K}_2, \mathcal{S}_2, \mathcal{V}_2)$, up to the underlying one-time signature scheme. Show that the triple $(\mathcal{K}_2, \mathcal{S}_2, \mathcal{V}_2)$ is a signature scheme.

E *Exercise* 29. The Merkle signature scheme is not a one-time scheme. Show that you can create a forgery if you see two signatures for distinct messages, but where both signatures have the same j.

If there are sufficiently many potential messages to be signed in this system, we could just choose leaf nodes at random and expect to avoid collisions. However, this is impractical because of the effort involved in generating the key pair, which is essentially proportional to the number of leaf nodes.

Remark. In practice, the effort required to generate a key pair is proportional to the number of messages to be signed over the life of the key. This is impractical. However, there are a number of options for improving the Merkle tree, including the ideas from the final two remarks in Section 6.1.

- Instead of having one large Merkle tree, we can create a tree of Merkle trees, attaching the root of one tree to the leaf nodes of the parent tree. This would allow a trade-off between reducing key generation time and increasing the length of the signatures.

- We could use n-ary trees instead of binary trees, which would shorten signatures, allowing a trade-off between computational requirements and signature length.

- Signing key material could be generated by a pseudo-random generator, which means that the secret key would not have to include all the signing keys, which would make the signing key much smaller, at little computational cost.

- Instead of keeping track of which leaf nodes had been used, we could derive which leaf node to use from a hash of the message. If the hash function was collision resistant, this would be secure. This would allow us to operate the signature scheme without a state.

There are a number of interesting constructions for such hash functions.

7 Securing Diffie-Hellman

As we have seen, the Diffie-Hellman protocol is subject to a man-in-the-middle attack, where Eve essentially runs the Diffie-Hellman protocol separately with Alice and Bob. Since Alice and Bob cannot distinguish each other's bits from Eve's bits, they will be cheated.

Signatures are one tool Alice and Bob can use to protect their Diffie-Hellman key exchange. In this case, Alice has a signing key pair (sk_A, vk_A) and Bob has a signing key pair (sk_B, vk_B), and they both know the other's verification key.

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

2. Bob receives x from Alice. He chooses a number b uniformly at random from the set $\{0, 1, 2, \ldots, n-1\}$ and computes $y = g^b$ and $z_B = x^b$. He computes $\sigma_B = \mathcal{S}(sk_B, m)$, where m is a message containing Alice' and Bob's names, that Alice initiated the key exchange, and x and y. Bob then sends y and σ_B to Alice.

3. Alice receives y and σ_B from Bob. Alice verifies σ_B and computes $z_A = y^a$. She also computes $\sigma_A = \mathcal{S}(sk_A, m')$, where m' is a message containing Alice' and Bob's names, that Alice initiated the key exchange, and x, y and σ_B. Alice then sends σ_B to Bob.

4. Bob receives σ_A from Alice. Bob verifies σ_A.

If either party's signature verification fails, that party stops immediately.

We note, without giving further details, that digital signatures can also be used with public key encryption to solve the problem of who sent a given ciphertext.

8 The Public Key Infrastructure Problem Revisited

Before asymmetric encryption was invented, a shared secret was required for secure communication over insecure channels. As we have seen, the Diffie-Hellman key exchange, public key encryption and digital signatures have removed the need for a preexisting shared secret, but public keys (for encryption or signature verification) still need to be exchanged before communicating.

A *public key infrastructure* is an infrastructure set up to move public keys from Alice to Bob in such a way that Bob can be sure that the public key he receives really belongs to Alice and is her current key, even if Alice and Bob have never communicated before.

As is often said, nothing will come of nothing, so Alice and Bob cannot hope to solve this problem on their own. One possible solution is the so-called *web of trust*. In this scheme, Alice and all her friends sign each other's public keys together with their unique names. Alice' public key, her name and her friend's signature is often called a *certificate*. The certificate is interpreted as Alice' friend saying that the public key belongs to the named person, namely Alice.

Alice' friends in turn sign their friends' public keys, and so forth. If we consider people as vertices in a graph, with edges between friends who have signed each other's public keys, Alice and Bob need to find a path between themselves in this graph.

A more practical system relies on a trusted third party, usually called a *certificate authority*. Again, the trusted third party signs Alice' public key along with her unique name. If Alice and Bob both trust each other's certificate authorities, they can simply send their certificate to the other party, and then use an appropriate public key protocol.

In practice, private keys are sometimes compromised, which means that Eve learns the key. When Alice discovers that someone knows her private key (and can thus impersonate her), Alice would like her certificate to stop working. She notifies her certificate authority that the certificate has been compromised.

The certificate authority was not involved when Alice and Bob communicated, so somehow Bob must be told that Alice' certificate has been revoked. The traditional approach is for the certificate authority to maintain a list of revoked certificates (a *certificate revocation list*). Anyone who relies on the certificate authority will periodically fetch an updated list of revoked certificates.

Since certificate revocation lists are fetched only periodically, there will typically be some time between Alice notifies her certificate authority until Bob stops accepting the certificate. Another problem with certificate revocation lists is that if there are many certificate authorities, managing the revocation lists becomes impractical.

One popular solution is for a certificate authority to provide a *certificate status service*. Any user may ask for the status of a given certificate. The certificate authority will reply with a signed message. If the certificate is valid (that is, not revoked), the message contains a statement to that effect and the current time. If the certificate has been revoked, the message contains a statement to that effect and the time of revocation.