1 Modular Arithmetic

Definition 1.1 (A divides B)**.**
Let $A, B \in \mathbb{Z}$. We say that A *divides* B (or A is a *divisor* of B), denoted $A|B$, if there is a number $C \in \mathbb{Z}$ such that $B = AC$. ■

Definition 1.2 (Prime number)**.**
Let $P \in \mathbb{N}$. We say that P is a *prime number* if $P \geq 2$ and the only divisors of P are 1 and P. ■

Definition 1.3 (Congruence modulo N)**.**
We denote by $A \bmod N$ the remainder you get when you divide A by N. Note that $A \bmod N \in \{0, 1, 2, \cdots, N-1\}$. We say that A and B are congruent modulo N, denoted $A \equiv_N B$ (or $A \equiv B \bmod N$), if $A \bmod N = B \bmod N$. ■

Exercise 1.4. Show that $A \equiv_N B$ if and only if $N|(A - B)$.

Remark. The above characterization of $A \equiv_N B$ can be taken as the definition of $A \equiv_N B$. Indeed, it is used a lot in proofs.

Notation 1.5. We write $\gcd(A, B)$ to denote the greatest common divisor of A and B. Note that for any A, $\gcd(A, 1) = 1$ and $\gcd(A, 0) = A$.

Definition 1.6 (Relatively prime)**.**
We say that A and B are relatively prime if $\gcd(A, B) = 1$. ■

Below, in Section 1.1, we first give the definitions of basic modular operations like addition, subtraction, multiplication, division and exponentiation. We also explore some of their properties. Later, in Section 1.2, we look at the computational complexity of these operations. Being able to compute these operations efficiently is crucial for applications.

1.1 Modular operations: Basic definitions and properties

1.1.1 Addition and subtraction

Notation 1.7. We let $\mathbb{Z}_N$ denote the set $\{0, 1, 2, \ldots, N-1\}$.

Definition 1.8 (Addition in $\mathbb{Z}_N$)**.**
For $A, B \in \mathbb{Z}_N$, we define the *addition* of A and B, denoted $A +_N B$, as $(A + B) \bmod N$. When N is clear from the context, we can drop the subscript N from $+_N$ and write $+$. For $N = 5$, we can represent the addition operation in $\mathbb{Z}_5$ using the following table.

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

In $\mathbb{Z}_N$, the element 0 is called the *additive identity*. It has the property that for any $A \in \mathbb{Z}_N$, $A +_N 0 = 0 +_N A = A$. ■

Exercise 1.9.

- Show that if $A \equiv_N B$ and $A' \equiv_N B'$, then $A + A' \equiv_N B + B'$.
- Show that for any $A, B \in \mathbb{Z}$,

$$(A + B) \bmod N = (A \bmod N) +_N (B \bmod N).$$

Definition 1.10 (Additive inverse)**.**
Let $A \in \mathbb{Z}_N$. The *additive inverse* of A, denoted $-A$, is defined to be an element in $\mathbb{Z}_N$ such that $A +_N -A = 0$. ■

Exercise 1.11. Show that every element of $\mathbb{Z}_N$ has a unique additive inverse.

Definition 1.12 (Subtraction in $\mathbb{Z}_N$)**.**
Let $A, B \in \mathbb{Z}_N$. We define "A minus B", denoted $A -_N B$, as $A +_N -B$. ■

Exercise 1.13. Show that in the addition table of $\mathbb{Z}_N$, every row and column is a permutation of the elements $\mathbb{Z}_N$.

1.1.2 Multiplication and division

Definition 1.14 (Multiplication in $\mathbb{Z}_N$)**.**
For $A, B \in \mathbb{Z}_N$, we define the *multiplication* of A and B, denoted $A \cdot_N B$, as $AB \bmod N$. If N is clear from the context, we can drop the subscript N from $\cdot_N$ and write $\cdot$. Furthermore, we can even drop $\cdot$ and represent $A \cdot_N B$ as simply AB. ■

Exercise 1.15.

- Show that if $A \equiv_N B$ and $A' \equiv_N B'$, then $AA' \equiv_N BB'$.
- Show that for any $A, B \in \mathbb{Z}$,

$$AB \bmod N = (A \bmod N) \cdot_N (B \bmod N).$$

Definition 1.16 (Multiplicative inverse)**.**
Let $A \in \mathbb{Z}_N$. The *multiplicative inverse* of A, denoted A^{-1}, is defined to be an element in $\mathbb{Z}_N$ such that $A \cdot_N A^{-1} = 1$. ■

Proposition 1.17. *Let $A, N \in \mathbb{N}$. The multiplicative inverse of A in $\mathbb{Z}_N$ exists if and only if* $\gcd(A, N) = 1$.

Definition 1.18 (Division in $\mathbb{Z}_N$)**.**
Let $A, B \in \mathbb{Z}_N$, where B has a multiplicative inverse B^{-1}. Then we define "A divided by B", denoted $A/_N B$, as $A \cdot_N B^{-1}$. ■

Notation 1.19. We let $\mathbb{Z}_N^*$ denote the set $\{A \in \mathbb{Z}_N : \gcd(A, N) = 1\}$. In other words, $\mathbb{Z}_N^*$ is the set of all elements of $\mathbb{Z}_N$ that have a multiplicative inverse.

Exercise 1.20. Show that $\mathbb{Z}_N^*$ is *closed* under multiplication, i.e., $A, B \in \mathbb{Z}_N^* \implies A \cdot_N B \in \mathbb{Z}_N^*$.

Remark. Similar to an addition table for $\mathbb{Z}_N$, one can consider a multiplication table for $\mathbb{Z}_N^*$. For example, $\mathbb{Z}_8^* = \{1, 3, 5, 7\}$, and the multiplication table is as below:

•	1	3	5	7
1	1	3	5	7
3	3	1	7	5
5	5	7	1	3
7	7	5	3	1

Exercise 1.21. Show that in the multiplication table of $\mathbb{Z}_N^*$, every row and column is a permutation of the elements $\mathbb{Z}_N^*$.

Definition 1.22 (Euler totient function)**.**
The Euler totient function $\varphi : \mathbb{N} \to \mathbb{N}$ is defined as $\varphi(N) = |\mathbb{Z}_N^*|$. By convention, $\varphi(0) = 0$. ■

Exercise 1.23. Show that for P a prime number, $\varphi(P) = P - 1$. Also show that for P and Q distinct prime numbers, $\varphi(PQ) = (P - 1)(Q - 1)$.

1.1.3 Exponentiation

Definition 1.24 (Exponentiation in $\mathbb{Z}_N$)**.**
Let $A \in \mathbb{Z}_N$ and $E \in \mathbb{Z}$. We write A^E to denote

$$\underbrace{A \cdot_N A \cdot_N \cdots \cdot_N A}_{E \text{ times}}.$$

■

Theorem 1.25 (Euler's Theorem)**.** *For any $A \in \mathbb{Z}_N^*$, $A^{\varphi(N)} = 1$. Equivalently, for any $A, N \in \mathbb{Z}$ with $\gcd(A, N) = 1$, $A^{\varphi(N)} \equiv_N 1$.*

Proof. (In this proof we drop the subscript N from the multiplication notation.) Take an arbitrary $A \in \mathbb{Z}_N^*$. Let $B_1, B_2, \ldots, B_k$ be the elements of $\mathbb{Z}_N^*$, where $k = \varphi(N)$. By Exercise 1.21, $\{AB_1, AB_2, \ldots, AB_k\} = \mathbb{Z}_N^*$. The product of all the elements in the first set can be written as $(AB_1)(AB_2)\cdots(AB_k)$. This must be equal to the product $B_1B_2\cdots B_k$, i.e.

$$(AB_1)(AB_2)\cdots(AB_k) = B_1B_2\cdots B_k.$$

Dividing both sides by $B_1B_2\cdots B_k$ (i.e. multiplying both sides by the inverse of $B_1B_2\cdots B_k$), we get $A^k = 1$, as desired. □

Remark. When N is a prime number, then Euler's Theorem is known as Fermat's Little Theorem.

Exercise 1.26. Let $A \in \mathbb{Z}_N^*$ and $E \in \mathbb{Z}$. Show that $A^E \equiv_N A^{E \bmod \varphi(N)}$.

Exercise 1.27. Compute by hand $102^{98} \bmod 7$.

Remark. What the previous two exercises demonstrate is that if we are exponentiating an element $A \in \mathbb{Z}_N^*$, then we can effectively think of the exponent as living in the set $\mathbb{Z}_{\varphi(N)}$. This will be important to keep in mind when we cover Cryptography later.

Definition 1.28 (Generator in $\mathbb{Z}_N^*$)**.**
Let $A \in \mathbb{Z}_N^*$. We say that A is a *generator* if

$$\{A^E : E \in \mathbb{Z}_{\varphi(N)}\} = \mathbb{Z}_N^*.$$

■

Theorem 1.29. *If N is a prime number, then $\mathbb{Z}_N^*$ contains a generator.*

1.2 Modular operations: Computational complexity

In this section, we will look at the computational complexities of doing the basic modular operations discussed in the previous section. We will use the fact that addition, subtraction, multiplication and division operations can be computed efficiently in $\mathbb{Z}$. Note that $A \bmod N$ is easy to compute by dividing A by N and seeing what the remainder is.

1.2.1 Addition and subtraction

In order to compute $A +_N B$ in $\mathbb{Z}_N$, we can simply add A and B in $\mathbb{Z}$ and then take the sum modulo N. To compute $A -_N B$, we can do $A + (N - B)$ in $\mathbb{Z}$ and then take the result modulo N.

1.2.2 Multiplication and division

In order to compute $A \cdot_N B$ in $\mathbb{Z}_N$, we can multiply A and B in $\mathbb{Z}$ and then take the product modulo N. To compute $A/_N B = A \cdot_N B^{-1}$, we first need to figure out whether B has a multiplicative inverse. Recall that B^{-1} exists if and only if B and N are relatively prime, i.e. $\gcd(B, N) = 1$. The following algorithm, known as *Euclid's Algorithm*, efficiently computes the greatest common divisor of two numbers.

$\gcd(A, B)$:

- If $B = 0$, then return A.
- Else return $\gcd(B, A \bmod B)$.

Exercise 1.30. Show that if $A \geq B$, $\gcd(A, B) = \gcd(A - B, B)$. Use this to show that Euclid's Algorithm correctly computes the greatest common divisor of two numbers.

Exercise 1.31. Suppose A and B can be represented with at most n bits each. Give an upper bound on the number of recursive calls Euclid's Algorithm makes in terms of n.

Using Euclid's Algorithm, we can check if $\gcd(B, N) = 1$ and determine if B has a multiplicative inverse. It turns out that a slight modification of Euclid's Algorithm also allows us to compute B^{-1} if it exists. In order to show this, we first need a definition.

Definition 1.32 (Miix)**.**
Let $A, B, C \in \mathbb{N}$. We say that C is a *miix* of A and B if

$$C = kA + \ell B$$

for some $k, \ell \in \mathbb{Z}$. ■

Exercise 1.33. Let $A, B, C \in \mathbb{N}$. Show that if C is a miix of A and B then C is a multiple of $\gcd(A, B)$.

Exercise 1.34. Let $A, B, C \in \mathbb{N}$. Show that if C is any multiple of $\gcd(A, B)$, then C is a miix of A and B.
Hint: Show how to modify Euclid's algorithm so that it outputs k and ℓ such that $\gcd(A, B) = kA + \ell B$.

Suppose B has a multiplicative inverse modulo N, i.e. $\gcd(B, N) = 1$. Then by the previous exercise, we can obtain k and ℓ such that $1 = kB + \ell N$. If we take this equation modulo N, we get that $kB \equiv_N 1$. Therefore k is the multiplicative inverse of B.

To sum up, if we want to compute $A/_N B = A \cdot_N B^{-1}$, we can first compute B^{-1} and then compute $A \cdot_N B^{-1}$.

Exercise 1.35. Prove Proposition 1.17 using the previous two exercises.

1.2.3 Exponentiation

Given $N \in \mathbb{N}$, $A \in \mathbb{Z}_N$ and $E \in \mathbb{N}$, we can compute $A^E \bmod N$ efficiently. Assume that A, E and N can be represented using at most n bits each. The algorithm below is known as *fast modular exponentiation*. To understand how the algorithm works, see the example following the algorithm.

FME(A, E, N):

- Repeatedly square A to obtain
  $A^2 \bmod N$, $A^4 \bmod N$, $A^8 \bmod N$, ..., $A^{2^n} \bmod N$.
- Multiply together (modulo N) the powers of A so that the product is A^E.
  To figure out which powers to multiply, look at the binary representation of E.

Consider the example of computing $2337^{53} \bmod 100$. The first step of the algorithm computes

$$\begin{aligned} 2337^2 &\bmod 100 \\ 2337^4 &\bmod 100 \\ 2337^8 &\bmod 100 \\ 2337^{16} &\bmod 100 \\ 2337^{32} &\bmod 100 \end{aligned}$$

by squaring 2337 modulo 100 5 times. The binary representation of 53 is 110101. This implies that

$$53 = 1 + 4 + 16 + 32.$$

Therefore to calculate $2337^{53} \bmod 100$, the second step of the algorithm does:

$$(2337 \bmod 100) \cdot (2337^4 \bmod 100) \cdot (2337^{16} \bmod 100) \cdot (2337^{32} \bmod 100).$$

Exercise 1.36. Suppose A, E and N are integers that can be represented using at most n bits. Give an upper bound on the running time of the above algorithm in terms of n.

1.2.4 Taking roots

Consider the following computational problem. You are given $A, E, N \in \mathbb{N}$ with $A \in \mathbb{Z}_N^*$. The goal is to output $B \in \mathbb{N}$ such that $B^E \equiv_N A$. In other words, the goal is to find the E'th root of A in $\mathbb{Z}_N^*$ (if it exists). Many experts believe (but cannot prove) that this problem cannot be computed in polynomial time. The assumed hardness of this problem is used in the famous RSA cryptosystem (see the section on Cryptography for details).

1.2.5 Taking logarithms

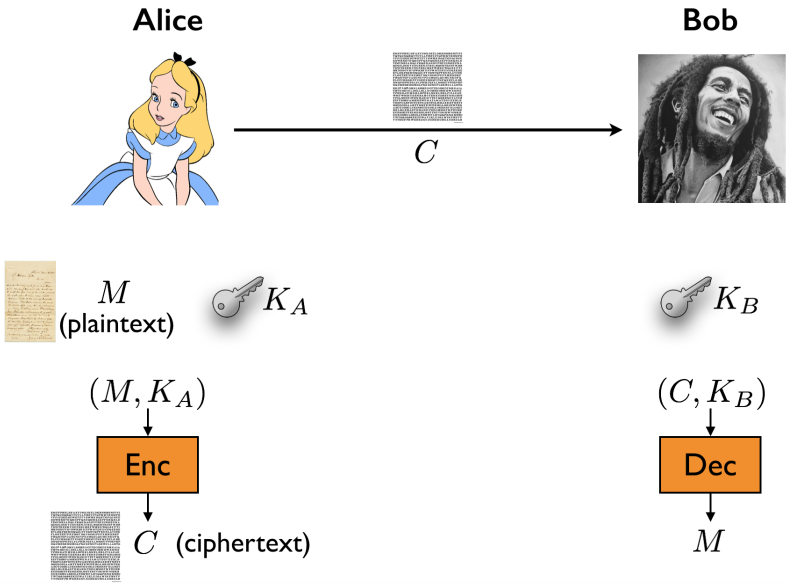
Consider the following computational problem which is known as the Discrete Log Problem in $\mathbb{Z}_P^*$. You are given $A, B, P \in \mathbb{N}$, where P is a prime number, $A \in \mathbb{Z}_P^*$, and $B \in \mathbb{Z}_P^*$ is a generator (Theorem 1.29 tells us that $\mathbb{Z}_P^*$ always contains a generator). The goal is to output $X \in \mathbb{N}$ such that $B^X \equiv_P A$. In a sense, this is like trying to compute $\log_B A$ in $\mathbb{Z}_N^*$. Many experts believe (but cannot prove) that this problem cannot be computed in polynomial time. The assumed hardness of this problem is used in the famous Diffie-Hellman secret key exchange protocol (see the section on Cryptography for details).

2 Cryptography

In this section, we will be interested in the following setting. There are three parties named Alice, Bob and Eve. Alice's goal is to send a private message to Bob over some channel. In particular, Alice wants only Bob to know what her message is, but unfortunately, any message she sends over the channel can be intercepted by the eavesdropper Eve. Therefore she would like to first encrypt her message (which is also known as the *plaintext*) and then send the encrypted message (the *ciphertext*) to Bob over the channel. Bob should be able to decrypt the ciphertext and recover the plaintext. If the system is secure, then Eve learns no information about the plaintext by seeing the ciphertext. If the system is not secure, we will call it broken. We assume that Eve knows the encryption and decryption algorithms used by Alice and Bob respectively.

2.1 Private-key cryptographic system

In a private-key cryptographic system, a protocol is designed as follows. We assume Alice has a private key K_A and Bob has a private key K_B. These keys help Alice and Bob encrypt and decrypt messages. In particular, if M is the plaintext that Alice wants to send, she encrypts it using an encryption algorithm Enc that takes M and K_A as input, and produces a ciphertext C as output: $\text{Enc}(M, K_A) = C$. This ciphertext C is sent to Bob, and Bob uses a decryption algorithm Dec that takes a ciphertext C and his private key K_B as input, and produces a plaintext M: $\text{Dec}(C, K_B) = M$. (We assume that M, C, K_A, K_B are encoded as strings over some finite alphabet Σ.)



A very simple example of a private-key protocol is the well-known Caesar shift.[1] In this protocol, $0 \leq K_A = K_B < 25$ and a plaintext is encrypted by replacing each letter with a letter which is K_A positions down the alphabet. This system is easy to break since the number of possibilities for the key is very small. Therefore one can try each possible key value one by one.

There are much more sophisticated private-key protocols. We now present a simple one that is provably perfectly secure.

2.1.1 One-time pad

In this section, we assume that the plaintext $M \in \{0,1\}^n$ is a binary string of length n. Then we choose $K_A = K_B \in \{0,1\}^n$ uniformly at random (and denote it by K). The encryption algorithm takes the bit-wise xor of M and K to produce an n-bit ciphertext C. Or in other words, for each i, the i'th bit of C, $C[i]$, is defined to be $M[i] \oplus K[i]$. Below is an example.

Encryption:

$$\begin{aligned} M &= 0101101011110101000001111 \\ \oplus\ K &= 1110011000101011111000101 \\ \hline C &= 1001011010111101111000010 \end{aligned}$$

The decryption algorithm is exactly the same as the encryption algorithm. It takes C and K as input, and produces M by taking the bit-wise xor of C and K. Observe

[1] See https://en.wikipedia.org/wiki/Caesar_cipher for details on the Caesar shift.

that for all i,

$$C[i] \oplus K[i] = (M[i] \oplus K[i]) \oplus K[i] = M[i] \oplus (K[i] \oplus K[i]) = M[i],$$

so the decryption algorithm correctly recovers the original message M.

For any plaintext $M \in \{0,1\}^n$, if $K \in \{0,1\}^n$ is chosen uniformly at random, then the cipertext C is a uniformly random element of $\{0,1\}^n$. This means that Eve learns nothing about M by seeing C, so the system is perfectly secure. The downside is that Alice and Bob have to share a key that is as long as the message itself.[2]

It is natural to ask whether there is any perfectly secure system like one-time pad that uses a shorter key. Claude Shannon proved that the answer is "no". To state his result informally, he showed that if K is shorter than M and Eve is computationally unbounded, then Eve can learn some information about the message M.

Given this, we let computational complexity come to our rescue. It is completely reasonable to assume that Eve is indeed computationally *bounded*. So from now on, we will assume that Eve is a polynomial-time agent.

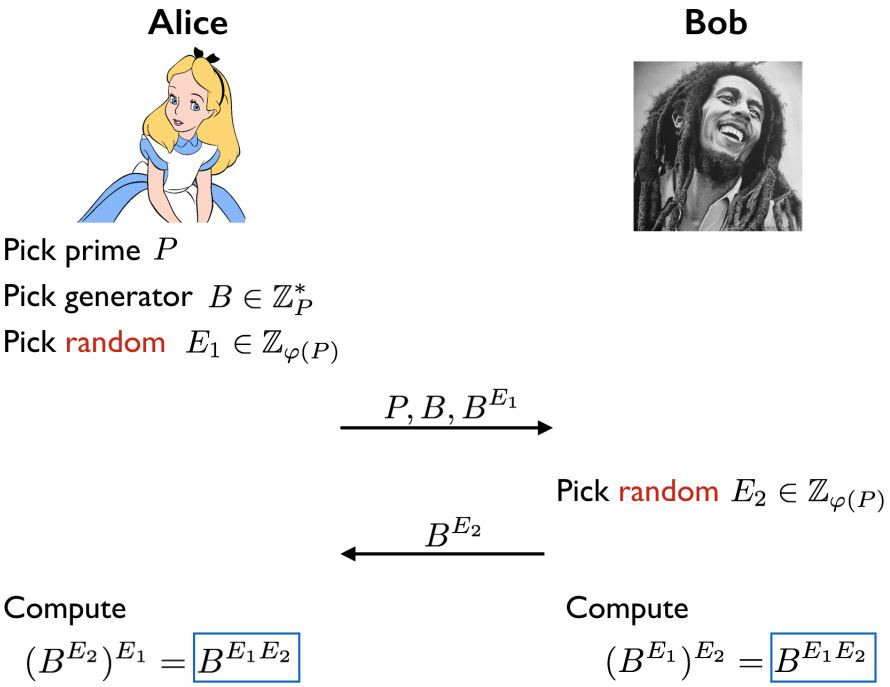
2.1.2 Diffie-Hellman secret key exchange

We present a protocol for Alice and Bob to agree on a secret key K by communicating publicly. The protocol is secure if Eve has no information about K even though she sees all the communication between Alice and Bob. This sounds like an impossible task, but it is actually believed to be feasible.

The protocol makes use of the assumption that the Discrete Log problem is computationally hard (see Section 1.2.5), and it goes as follows. Alice picks privately a (sufficiently large) random prime number P, a generator B in $\mathbb{Z}_P^*$, and a random exponent $E_1 \in \mathbb{Z}_{\varphi(P)}$.[3] She computes B^{E_1} in $\mathbb{Z}_P^*$ and sends over to Bob P, B, B^{E_1}. Bob privately picks a random exponent $E_2 \in \mathbb{Z}_{\varphi(P)}$ and computes B^{E_2} in $\mathbb{Z}_P^*$. He sends B^{E_2} to Alice. At this point both players can privately compute $S = B^{E_1 E_2}$ in $\mathbb{Z}_P^*$, which is defined as the secret key that they now share. The protocol is illustrated below.

[2]For the system to remain perfectly secure, you should not reuse the same key for more than one message. This is the reason for the name "one-time pad".

[3]Why is the exponent chosen from $\mathbb{Z}_{\varphi(P)}$? Recall that thanks to Euler's Theorem, if we are exponentiating an element $A \in \mathbb{Z}_N^*$, then we can effectively think of the exponent as living in the set $\mathbb{Z}_{\varphi(N)}$.



There are two important questions related to this protocol. First, are all the operations done by Alice and Bob polynomial-time computable? Second, how secure is the system?

Even though we won't explicitly discuss it, every computation done by Alice and Bob can indeed be done in polynomial time. For the security, observe that we definitely need the Discrete Log problem to be computationally hard, because otherwise, Eve can compute E_1 from B^{E_1} and E_2 from B^{E_2}. Then it is easy for her to compute the "secret" $B^{E_1E_2}$ (since she also knows B). To be more careful though, we want that given P, B, B^{E_1}, B^{E_2} (i.e. what Eve sees), it is computationally hard to compute $B^{E_1E_2}$. This is known as the Diffie-Hellman assumption. Unfortunately we cannot prove this assumption since if we could, we would be also proving $\mathsf{P} \neq \mathsf{NP}$. Even if the Diffie-Hellman assumption holds, we should not be satisfied. Not only we don't want Eve to compute the secret $B^{E_1E_2}$, but we don't want her to gain *any* information about $B^{E_1E_2}$ (e.g. not even the first bit of it). The assumption that Eve learns nothing about the secret is known as the Decisional Diffie-Hellman assumption.[4]

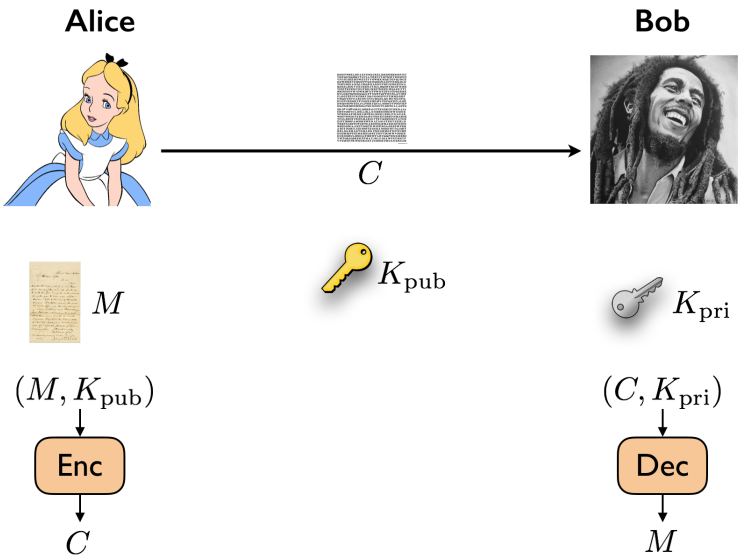
2.2 Public-key cryptographic system

In a public-key cryptographic system, our goal is to design a protocol that allows Alice to send a message to Bob without the need of having to exchange messages in order to share a secret key. For example, if a Nigerian prince wants to send you a private email

[4]Decisional Diffie-Hellman assumption turns out to be false in the group $\mathbb{Z}_P^*$ but there are other cyclic groups for which experts believe the assumption should hold.

notifying you that you have inherited a million dollars, then he should be able to do so without needing your authorization.

In order to establish this, a public-key cryptographic system uses the following general strategy. Bob generates a tuple of keys $(K_{\text{pri}}, K_{\text{pub}})$, where K_{pri} is called the *private key* and is kept private to him, and K_{pub} is called the *public key* and is published to the world. If someone (e.g. Alice) wants to send a message to Bob, they use the public key to encrypt their message M. That is, the ciphertext C is produced by running an encryption algorithm $\text{Enc}(M, K_{\text{pub}})$. Once Bob receives C, he decrypts it using his private key by running a decryption algorithm $\text{Dec}(C, K_{\text{pri}})$.



We now present different instantiations of this idea.

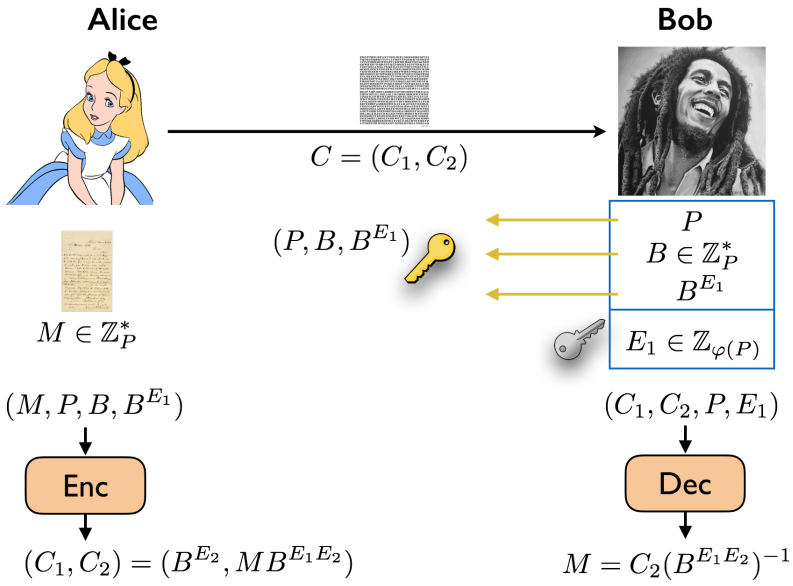
2.2.1 ElGamal public-key cryptographic system

The first public-key protocol we present is similar in nature to the Diffie-Hellman secret-key exchange protocol. However, to make the correspondence clear, we reverse the roles of Alice and Bob. Below are the details.

In the ElGamal protcol, Bob picks a (sufficiently large) prime number P, a generator $B \in \mathbb{Z}_P^*$, and a random exponent $E_1 \in \mathbb{Z}_{\varphi(P)}$. He computes B^{E_1} in $\mathbb{Z}_P^*$. The private key is $K_{\text{pri}} = E_1$ and the public key is $K_{\text{pub}} = (P, B, B^{E_1})$.

The message M that Alice wants to send is viewed as an element of $\mathbb{Z}_P^*$ (it is easy to agree on an encoding scheme to do this). To encrypt her message, Alice does the following. She picks a random exponent $E_2 \in \mathbb{Z}_{\varphi(P)}$, and computes B^{E_2}, $B^{E_1E_2}$ and $MB^{E_1E_2}$ in $\mathbb{Z}_P^*$. Then the ciphertext she sends over to Bob is $C = (C_1, C_2) = (B^{E_2}, MB^{E_1E_2})$. Once Bob receives this message, using C_1 he first computes $C_1^{E_1} = B^{E_1E_2}$ in $\mathbb{Z}_P^*$. Note that this is the *secret key* from the Diffie-Hellman protocol. Let's call it S. He computes S^{-1} in $\mathbb{Z}_P^*$. Then he computes $C_2S^{-1} = M$ to recover the original message M.

Even though this may seem like a non-simple protocol, it has a simple summary once you understand the Diffie-Hellman secret-key exchange protocol. Effectively, Alice and Bob are sharing the private secret $S = B^{E_1 E_2}$ as in the Diffie-Hellman secret-key exchange protocol. Alice "masks" her message M using S (by multiplying M and S). Since Bob also knows S, he can compute its inverse, and therefore recover M.



As in the Diffie-Hellman secret-key exchange protocol, all the computation done by Alice and Bob can be carried out in polynomial time. Furthermore, the security is based on the same assumptions as in the Diffie-Hellman secret-key exchange protocol. The details are omitted.

2.2.2 RSA public-key cryptographic system

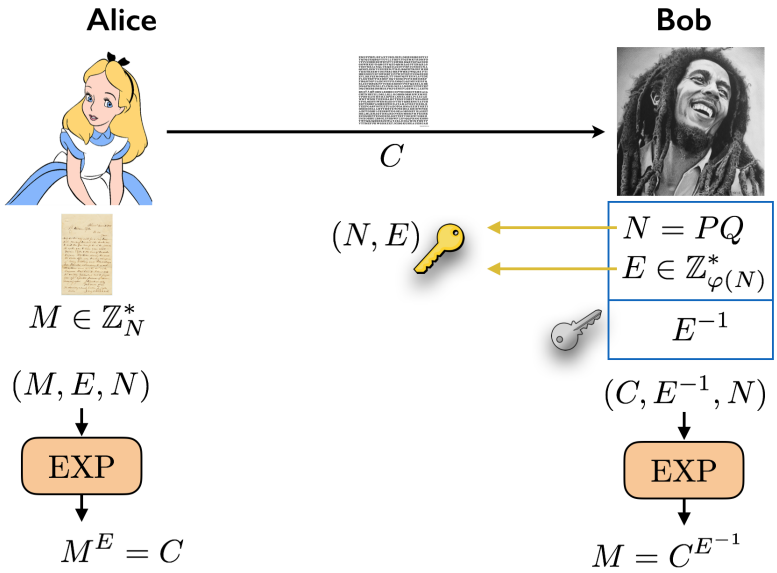
The RSA cryptographic system uses the assumption that taking roots in the modular universe is a computationally hard problem (see Section 1.2.4). Notice that taking roots is the inverse of the exponentiation function. And in RSA, the encryption is indeed done using the exponentiation function. Below are the details.

First, Bob picks two (sufficiently large) distinct prime numbers P and Q. He multiplies them together to obtain $N = PQ$.[5] He picks an exponent $E \in \mathbb{Z}_{\varphi(N)}^*$.[6] He computes E^{-1} in $\mathbb{Z}_{\varphi(N)}^*$ and keeps that as his private key, $K_{\text{pri}} = E^{-1}$. The public key is $K_{\text{pub}} = (N, E)$.

[5] Why is N chosen to be a product of primes and not just a prime number? We will explore this question after we describe the protocol.

[6] Why is the exponent chosen from $\mathbb{Z}_{\varphi(N)}^*$? Once again, if we are exponentiating an element $A \in \mathbb{Z}_N^*$, then we can effectively think of the exponent as living in the set $\mathbb{Z}_{\varphi(N)}$. In the RSA protocol, we will need the exponent to have an inverse in $\mathbb{Z}_{\varphi(N)}$ and therefore we pick it from $\mathbb{Z}_{\varphi(N)}^*$.

The message M that Alice wants to send to Bob is viewed as an element of $\mathbb{Z}_N^*$. To encrypt her message, she computes $C = M^E$ in $\mathbb{Z}_N^*$, and sends it over to Bob. The decryption algorithm happens to be exactly the same as the encryption algorithm. Once Bob receives C, he computes $C^{E^{-1}}$ in $\mathbb{Z}_N^*$, and recovers M since $C^{E^{-1}} = (M^E)^{E^{-1}} = M^{EE^{-1}} = M$.



As before, it can be shown that all the computation done by Alice and Bob is polynomial-time. We now make a few comments about the security of the system. What is the advantage that Bob has over Eve that allows him to decrypt the message? If Eve could compute E^{-1} herself, then she would be able to decrypt the message as well. To compute E^{-1}, you need to know $\varphi(N)$ since E^{-1} lives in $\mathbb{Z}_{\varphi(N)}^*$. Bob's advantage is that he can easily compute $\varphi(N)$ because $\varphi(N) = (P-1)(Q-1)$. In other words, the advantage that Bob has is that he knows the prime factorization of N. If Eve could factor N efficiently, then she could also easily compute $\varphi(N) = (P-1)(Q-1)$. It turns out that factoring N and computing $\varphi(N)$ are computationally equivalent in the following sense. Clearly, as we argued, if we can factor N in polynomial time, then we can compute $\varphi(N)$ in polynomial time. Furthermore, if we can compute $\varphi(N)$ in polynomial time, then we can factor N in polynomial time (we leave this as an exercise to the reader).

One might ask if computing $\varphi(N)$ is the only way to crack RSA. We don't know the answer to this question. So we cannot rule out that there might be some other devious way of recovering the message M without computing $\varphi(N)$ (or factoring N).

Solutions to Selected Exercises

Exercise 1.4

If $A \equiv_N B$, then by definition, A and B have the same remainder R when they are divided by N. So we can write $A = Q \cdot N + R$ for some $Q \in \mathbb{Z}$ and $B = Q' \cdot N + R$ for some $Q' \in \mathbb{Z}$. Then $A - B = (Q - Q') \cdot N$, and therefore $N|(A - B)$.

Suppose $N|(A-B)$. Write $A = Q \cdot N + R$ for some $Q \in \mathbb{Z}$ and $R \in \{0, 1, \ldots, N-1\}$. Also write $B = Q' \cdot N + R'$ for some $Q' \in \mathbb{Z}$ and $R' \in \{0, 1, \ldots, N-1\}$. Then $A - B = (Q - Q') \cdot N + (R - R')$. Since $N|(A - B)$, it must be the case that $N|(R - R')$. Since $R - R'$ is an integer between $-(N-1)$ and $(N-1)$, the only way N can divide $R - R'$ is if $R = R'$, i.e., $A \equiv_N B$.

Exercise 1.9

Part 1: Since $A \equiv_N B$, we have $N|(A - B)$, and since $A' \equiv_N B'$, we have $N|(A' - B')$. This implies $N|(A - B) + (A' - B')$, or in other words, $N|(A + A') - (B + B')$. And this is equivalent to $A + A' \equiv_N B + B'$.

Part 2: Follows from the previous part and the definition of $+_N$.

Exercise 1.11

The additive inverse of $A \in \mathbb{Z}_N$ is 0 if $A = 0$, and is $N - A$ otherwise.

To show uniqueness, assume that $-A$ and $-A'$ are both additive inverses of A. Then $A +_N -A = A +_N -A' = 0$, which implies $-A = -A'$.

Exercise 1.13

We argue that every row contains distinct elements of $\mathbb{Z}_N$, which implies that every row is a permutation of $\mathbb{Z}_N$. Take an arbitrary row, which corresponds to some element $A \in \mathbb{Z}_N$. Suppose for the sake of contradiction that two entries of this row are the same. Then there exists B and B' in $\mathbb{Z}_N$, $B \neq B'$, such that $A +_N B = A +_N B'$. But then if we add $-A$ to both sides of the equality, we get $B = B'$, a contradiction.

The argument for the columns is the same.

Exercise 1.15

Part 1: Hint: Write the elements as $QN + R$ for some $Q \in \mathbb{Z}$ and remainder $R \in \{0, 1, \ldots, N-1\}$.

Part 2: Follows from Part 1 and the definition of $\cdot_N$.

Exercise 1.20

If A and B are in $\mathbb{Z}_N^*$, then they have inverses $A^{-1}, B^{-1} \in \mathbb{Z}_N^*$. To show $A \cdot_N B$ is in $\mathbb{Z}_N^*$, we need to show that it has an inverse modulo N. And indeed, $B^{-1} \cdot_N A^{-1}$ is the inverse of $A \cdot_N B$ since $A \cdot_N B \cdot_N B^{-1} \cdot_N A^{-1} = 1$.

Exercise 1.21

We argue that every row contains distinct elements of $\mathbb{Z}_N^*$, which implies that every row is a permutation of $\mathbb{Z}_N^*$. Take an arbitrary row, which corresponds to some element $A \in \mathbb{Z}_N^*$. Suppose for the sake of contradiction that two entries of this row are the same. Then there exists B and B' in $\mathbb{Z}_N^*$, $B \neq B'$, such that $A \cdot_N B = A \cdot_N B'$. But then if we multiply both sides of the equality by A^{-1} we get $B = B'$, a contradiction.

The argument for the columns is the same.

Exercise 1.23

We know that $\varphi(N)$ is the number of elements in $\{0, 1, 2, \ldots, N-1\}$ that are relatively prime to N. If P is a prime number, then for any $A \in \{1, 2, \ldots, P-1\}$, we have $\gcd(A, P) = 1$. And $\gcd(0, P) = P$. So $\varphi(P) = P - 1$.

When $N = PQ$ for distinct primes P and Q, we need to determine the number of elements in $\{0, 1, 2, \ldots, PQ-1\}$ that are relatively prime to PQ. The elements that are **not** relatively prime to PQ are 0 and

$$P, 2P, 3P, \ldots, (Q-1)P,$$
$$Q, 2Q, 3Q, \ldots, (P-1)Q.$$

In total there are $1 + (Q-1) + (P-1) = Q + P - 1$ of these elements. Then the number of elements that *are* relatively prime to PQ is $PQ - (Q + P - 1) = PQ - Q - P + 1 = (P-1)(Q-1)$.

Exercise 1.26

For the first part, we can write $E = \varphi(N) \cdot Q + R$ where Q is some integer and $R \in \{0, 1, \ldots, \varphi(N) - 1\}$ is the remainder. So $E \equiv_{\varphi(N)} R$. Then

$$A^E = A^{\varphi(N) \cdot Q + R} = \left(A^{\varphi(N)}\right)^Q \cdot A^R = A^R,$$

where for the last equality, we used Euler's Theorem (Theorem 1.25).

Exercise 1.27

Since $\gcd(102, 7) = 1$, we can use the previous exercise and reduce the exponent modulo 6. So $102^{98} \equiv_7 102^2$. Furthermore, 102 can be reduced modulo 7. So $102^2 \equiv_7 4^2$. And 16 modulo 7 is 2, which is the answer.

Exercise 1.30

If x divides A and B, then it must divide $A - B$. In particular, $\gcd(A, B)$ divides both B and $A - B$, and therefore $\gcd(A - B, B) \geq \gcd(A, B)$.

If x divides $A - B$ and B, then it must divide A. In particular, $\gcd(A - B, B)$ divides both A and B, and therefore $\gcd(A, B) \geq \gcd(A - B, B)$.

So we can conclude $\gcd(A, B) = \gcd(A - B, B)$.

When $A \geq B$, if we iteratively use the equality $\gcd(A, B) = \gcd(A - B, B)$ to subtract B from the larger number A, then we will eventually arrive at $\gcd(A, B) = \gcd(A \bmod B, B)$. For example:

$$\begin{aligned}\gcd(6004, 6) &= \gcd(5998, 6)\\ &= \gcd(5992, 6)\\ &= \gcd(5986, 6)\\ &\cdots\\ &= \gcd(4, 6).\end{aligned}$$

So $\gcd(6004, 6)$ eventually ends up at $\gcd(6004 \bmod 6, 6)$.

$\gcd(A \bmod B, B)$ is obviously equal to $\gcd(B, A \bmod B)$. So one can show that Euclid's algorithm is correct by an induction argument, where the base case corresponds to the base case of the recursive algorithm, and the induction step corresponds to the recursive call.

Exercise 1.31

We claim that $A \bmod B \leq A/2$. To see this, we case on whether $A \geq 2B$. If $A \geq 2B$, then the claim is true because $A \bmod B$ is always less than B. If $A < 2B$, then the claim is true because $A \bmod B = A - B < A - A/2 = A/2$.

Now when we call the algorithm with input (A, B) and make a recursive call, the next pair of inputs is $(B, A \bmod B)$. Using the claim above, we have $(A \bmod B) \cdot B \leq AB/2$. So in each recursive call, the product of the inputs is going down by a factor of at least 2. And this implies the number of recursive calls is at most $\log_2(AB)$, which is $O(n)$ if A and B are at most n-bits.

Exercise 1.33

Let $G = \gcd(A, B)$. If C is a miix of A and B, then $C = kA + \ell B$. Furthermore, we know we can write $A = xG$ for some integer x, and $B = yG$ for some integer y. Therefore

$$C = kA + \ell B = kxG + \ell yG = G(kx + \ell y),$$

which shows C is a multiple of G.

Exercise 1.34

Here is the extended Euclid's algorithm.

Extended-Euclid(A, B):

- if B divides A, return $(B, 0, 1)$
- else:
  - $(G, k, \ell) \leftarrow$ Extended-Euclid$(B, A \bmod B)$
  - return $(G, \ell, (k - \ell \cdot \lfloor A/B \rfloor))$

Here, we have modified Euclid's algorithm to return a tuple of variables; Extended-Euclid$(A, B) = (G, k, \ell)$ where $G = \gcd(A, B)$ and $G = k \cdot A + \ell \cdot B$. By the correctness of Euclid's algorithm, we can conclude that the returned value G is the gcd (both algorithms do the same calculations for G). We use induction on the number of steps to argue correctness of the returned values (k, ℓ).

Base Case: If B divides A, the algorithm correctly returns $k = 0$ and $\ell = 1$.

Induction Step: Suppose the algorithm ran for $s > 1$ steps. By the induction hypothesis, we can assume that $G = k \cdot B + \ell \cdot (A \bmod B)$. Since we can write $A = q \cdot B + (A \bmod B)$ where $q = \lfloor A/B \rfloor$, we can say $G = k \cdot B + \ell(A - q \cdot B)$. Thus, the returned tuple $(G, \ell, k - \ell \cdot \lfloor A/B \rfloor)$ is correct.

Let's come back to the main question. The above algorithm shows that $\gcd(A, B)$ is a miix of A and B. So if C is a multiple of $\gcd(A, B)$, it is also a miix of A and B.

Exercise 1.35

The previous two exercises imply that C is a miix of A and B if and only if C is a multiple of $\gcd(A, B)$. Then,

$$\begin{aligned} A^{-1} \text{ exists} &\iff \exists k \text{ such that } kA \equiv_N 1 \\ &\iff \exists k \text{ such that } N \text{ divides } kA - 1 \\ &\iff \exists k, q \text{ such that } kA - 1 = qN \\ &\iff \exists k, q \text{ such that } 1 = kA + (-q)N \\ &\iff 1 \text{ is a miix of } A \text{ and } N \\ &\iff \gcd(A, N) = 1. \end{aligned}$$