OLR

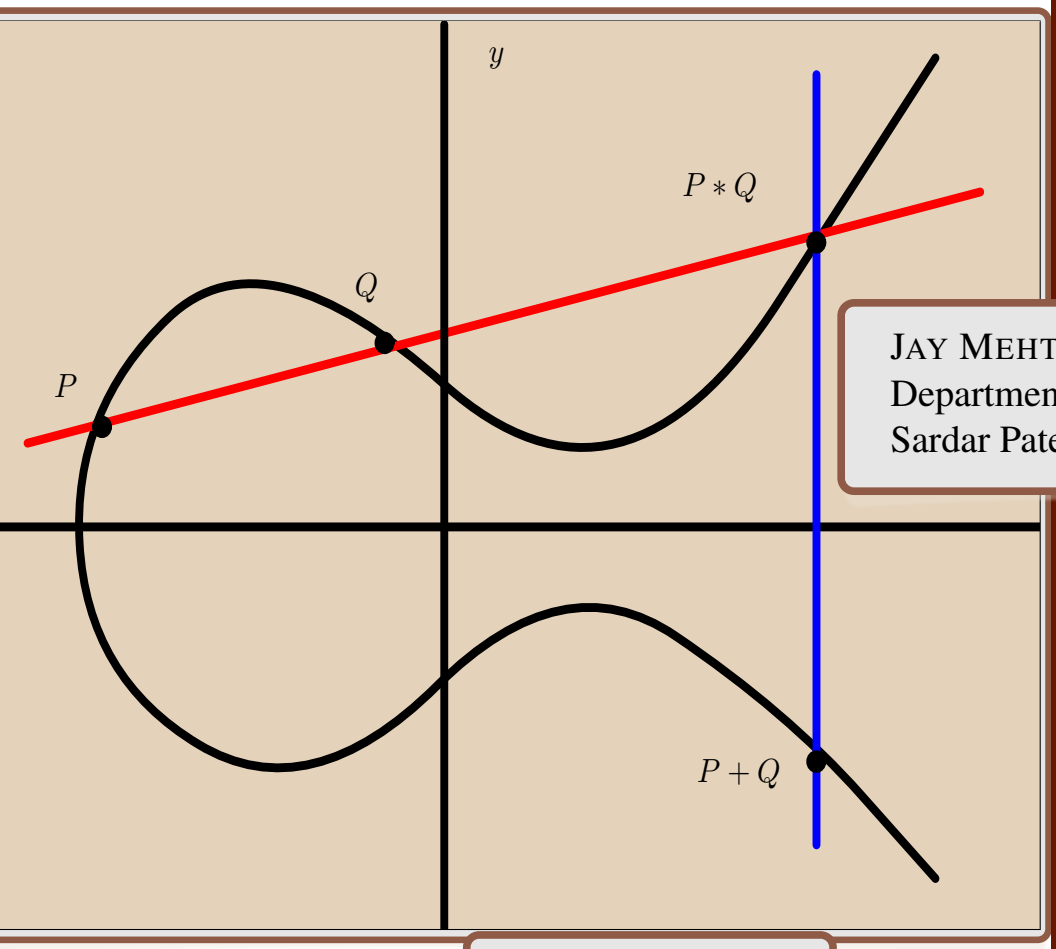
Lecture notes on

NUMBER THEORY AND CRYPTOGRAPHY

PS04EMTH59

DLP

$\mathcal{O}(N)$



JAY MEHTA
Department of Mathematics,
Sardar Patel University.

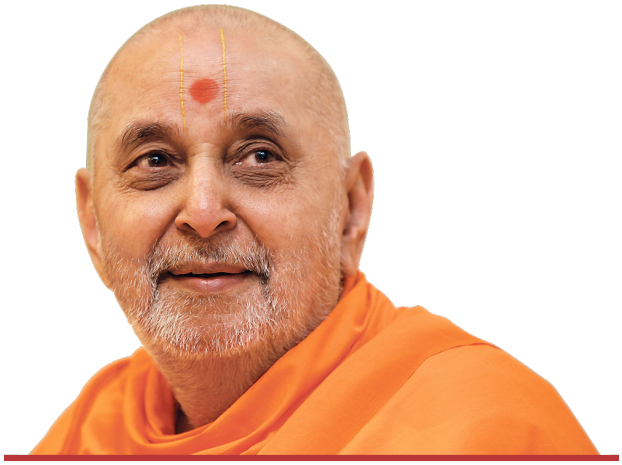
RSA

$E(\mathbb{F}_p)$

SEMESTER - IV
2022-23

Dedication

Dedicated to my Guru



HH PRAMUKH SWAMI MAHARAJ

on His Centenary

Pramukh Swami Maharaj Shatabdi Mahotsav
1921 - 2022

Contents

Preface and Acknowledgments

This lecture note of the course "Number Theory and Cryptography" offered to the M.Sc. (Semester - III and Semester IV) students at Department of Mathematics, Sardar Patel University, 2022-23 is aimed to provide a reading material to the students, in addition to the references mentioned in the university syllabus, so as to save time of the teacher and the students in writing on the board and copying in the notebooks, respectively. These notes are tailor-made for the "Number Theory and Cryptography" (PS03EMTH55/PS04EMTH59) syllabus of M.Sc. (Semester-III/IV) of the University and do not cover all the topics of Cryptography.

This note is prepared from the recommended reference books, and it is not the original work of the author. We mostly followed the text book by "An Introduction to Mathematical Cryptography" by J. Hoffstein, J. Pipher, and J. H. Silverman.

This is the first version of the note on Cryptography. We have strictly followed the text in our syllabus and except a few examples, all the examples are taken from the book. Since most of the examples, in this course, have very large numerical values, it requires computer programming or calculator to compute the values and solve them. In the subsequent revisions, we will try to frame new examples and exercises for which computations can be done easily using calculators if not by hand. We also request the students and the readers to create their own examples which can be easily computed and understood instead of using large values.

Most of the exercises were separately given to students as seminars and assignments and are not included here. Students are advised and encouraged to solve more and more exercises from the reference book(s). There may be a few errors/typos in this reading material. The students and interested readers are welcomed to give their valuable suggestions, comments or point out errors, if and whenever, they find any.

Finally, for the students who are also studying a course in computer programming, it would be good to write (python) programs for the algorithms and cryptosystems discussed in this course for computational purpose and for practice.

JAY MEHTA

Date: April 5, 2023

Syllabus

PS04EMTH59: Number Theory and Cryptography

Unit I: Number Theory and Discrete Logarithm Problem: Simple substitution ciphers (except cryptanalysis), divisibility and GCD, modular arithmetic, prime numbers, unique factorization and finite fields, primitive roots in finite fields. The discrete logarithm problem.

Unit II: DLP based cryptosystems: The Diffie-Hellman key exchange, the ElGamal public key cryptosystem, difficulty of discrete log problem (DLP), a collision algorithm for the DLP, the Chinese remainder theorem, the Pohlig-Hellman algorithm.

Unit III: The RSA Algorithm: Euler's formula and roots modulo pq, the RSA public key cryptosystem, implementation and security issues, primality testing, Pollard's p-1 factorization algorithm.

Unit IV: Elliptic curve cryptography: Elliptic curves, elliptic curve over finite fields, the elliptic curve discrete logarithm problem, elliptic curve cryptography.

References

1. Hoffstein J., Pipher J., Silverman J. H., An Introduction to Mathematical Cryptography, Undergraduate Texts in Mathematics, Springer, New York, (2008).
2. Douglas R. Stinson, Cryptography: Theory and Practice, Second Edition, Chapman and Hall/CRC, (2005).
3. N. Koblitz, A Course in Number Theory and Cryptography, Springer (1994).
4. J. A. Buchmann, Introduction to Cryptography, Second Edition, Undergraduate Texts in Mathematics, Springer-Verlag, New York, (2004).

CHAPTER 1

Number Theory and Discrete Log Problem

1.1 Simple substitution ciphers

As Julius Caesar was surveying a battle from his hilltop post, he received a courier containing the following gibberish message:

j s j r d k f q q n s l g f h p g w j f p y m w t z l m n r r n s j s y q z h n z x

Reading this message, Julius realised that there was a temporary gap in the formation of opponent's troops and to take this opportunity he immediately sent his troops there. How did such gibberish junk text convey such an important message to Caesar?

In cryptography, *plaintext* is the text in English or our language in original readable form and *ciphertext* is the gibberish junk text which is not readable to others easily. The process of converting plaintext to ciphertext is called *encryption* and the process of converting ciphertext to get back the plaintext is called *decryption*.

Julius Caesar shifted each letter in the message, i.e., in the ciphertext by five letters up the alphabet. For example, j is shifted to e in the plaintext as e is followed in the alphabet by f, g, h, i, j. Applying this procedure to the entire ciphertext, we get

j	s	j	r	d	k	f	q	q	n	s	l	g	f	h	p	g	w	j	f	p	y	m	w	t	z	l	m	n	r	r	n	s	j	s	y	q	z	h	n	z	x
e	n	e	m	y	f	a	l	l	i	n	g	b	a	c	k	b	r	e	a	k	t	h	r	o	u	g	h	i	m	m	i	n	e	n	t	l	u	c	i	u	s

The second line is the decrypted plaintext on which applying proper punctuation, we read the plaintext message as

Enemy falling back. Breakthrough imminent. Lucius.

How to address the letter d in the ciphertext? There is no letter appearing five letters before d in the alphabet. To answer this, we must wrap around to the end of the alphabet. Thus, d is replaced by y as y is followed by z, a, b, c, d. This wrap-around effect can be seen by placing the alphabet around a circle. The plaintext is placed on the outer wheel and is written while the ciphertext alphabet are written in the inner wheel which is rotated by 5 as shown in the figure below. The ciphertext is written in capital letters to distinguish it from the plaintext. To decrypt a letter, simply find it on the inner wheel and read the the corresponding plaintext letter from

the outer wheel. Similarly, to encrypt a letter, find that letter in the outer plaintext wheel and look for the corresponding ciphertext letter in the inner wheel.

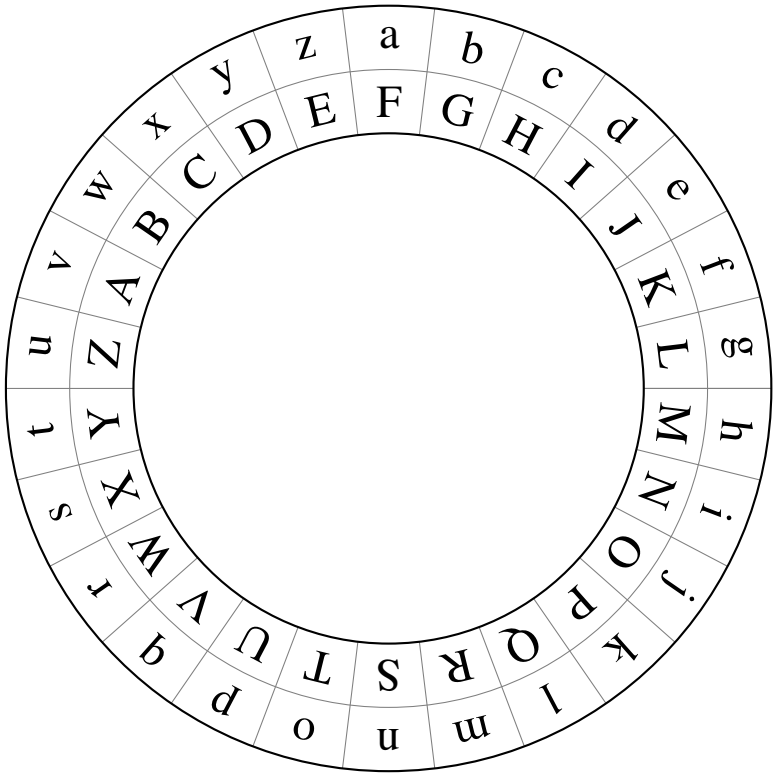


Figure 1.1: A cipher wheel with an offset of five letters

Although it is unlikely that Romans would be communicating in modern English back then but it is believed that Julius Caesar employed this early method of cryptography which is called the *Caesar cipher*. It is sometimes also called the *shift cipher* since each letter in the alphabet is either shifted up or down.

Cryptography is the methodology of concealing (hiding or keeping secret) the content of messages. It has Greek origin and comes from the two Greek words kryptos which means hidden and graphikos which means writing. The modern study of cryptography is also known as *cryptology*.

In a basic communication scenario, there are two parties, Alice and Bob who wants to communicate with each other. A third party Eve is a potential eavesdropper, a villain. In the Caesar cipher, each letter is replaced by one specific substitute letter. However, if Bob encrypts a message for Alice using Caesar cipher and if the encrypted message is accessible to Eve, then it will take very little time for Eve to decrypt it as there are only 26 possible shifts.

Bob can make the encryption better by using more complicated replacement scheme which makes the message harder to attack or reveal. For example, he could replace every occurrence of a by z and every occurrence of y by b, and so on. This is an example of a *substitution cipher* in which every letter is replaced by another letter (or a symbol). Caesar cipher is an example of a simple substitution cipher. A substitution cipher may be considered as a rule or a function

$$\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \ldots, \mathsf{x}, \mathsf{y}, \mathsf{z}\} \longrightarrow \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}, \ldots, \mathsf{X}, \mathsf{Y}, \mathsf{Z}\}$$

assigning each plaintext letter in the domain to a different ciphertext letter in the range. Note

that, for the decryption to work, the encryption function must have the property that no two plaintext letters go to the same ciphertext letter. A function with this property is called a *one-one* function or an *injective* function.

A convenient way to describe the encryption function is to create a table by writing the plaintext alphabet in the top row and putting each ciphertext letter below the corresponding plaintext letter.

Example 1.1.1. A simple substitution encryption table is given below. The ciphertext alphabet is a randomly chosen permutation of 26 letters.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
C	I	S	Q	V	N	F	O	W	A	X	M	T	G	U	H	P	B	K	L	R	E	Y	D	Z	J

Table 1.1: Simple substitution encryption table

Let us encrypt the plaintext

Four score and seven years ago.

Look up plaintext letter in the encryption table and write the corresponding ciphertext letter below.

f	o	u	r	s	c	o	r	e	a	n	d	s	e	v	e	n	y	e	a	r	s	a	g	o
N	U	R	B	K	S	U	B	V	C	G	Q	K	V	E	V	G	Z	V	C	B	K	C	F	U

It is then customary to write the ciphertet in five-letter blocks:

NURBK SUBVC GQKVE VGZVC BKCFU.

Decryption is a similar process. Suppose that we received the message

SCRKV WMUEV ZURNU BWGNW GWLZ

which we know was encrypted using Table 1.1. We can decrypt it by reversing the encryption process, i.e., by finding each ciphertext in the second row of Table 1.1 and writing down the corresponding letter from the top row. Since all the letters in the second row of the table are mixed up and not in order, this process of decryption is somewhat inefficient. Hence, it is better to make a decryption table in which ciphertext letters are in the lower row and are listed in the alphabetical order and the corresponding plaintext letters written in the upper row are mixed up. See the table below.

j	r	a	x	v	g	n	p	b	z	s	t	l	f	h	q	d	u	c	m	o	e	i	k	w	y
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Table 1.2: Simple substitution decryption table

Using Table 1.2, we can easily decrypt the message as below.

S	C	R	K	V	W	M	U	E	V	Z	U	R	N	U	B	W	G	N	W	G	W	L	Z
c	a	u	s	e	i	l	o	v	e	y	o	u	f	o	r	i	n	f	i	n	i	t	y

Putting in the appropriate word breaks and punctuations, we get our plaintext as follows.

Cause I love you for infinity.

■

We skip the cryptanalysis of the substitution cipher in this note as it is beyond the scope of our syllabus. We next move to some number theory part for now. Some of the topics in this chapter is just a revision of basic number theory studied in school or college level.

1.2 Divisibility and greatest common divisors

Definition 1.2.1

Let a and b be integers $b \neq 0$. We say that b divides a, or that a is divisible by b, if there is an integer c such that

$$a = bc.$$

We denote by $b \mid a$ to indicate that b divides a. We denote by $b \nmid a$ to indicate that b does not divide a.

Example 1.2.2. We have $847 \mid 485331$ as $485331 = 847 \cdot 573$. On the other hand $355 \nmid 259943$ as when we divide 259943 by 355, we get a remainder of 83, i.e., $259943 = 355 \cdot 732 + 83$. ■

Remark 1.2.3. Note that every integer is divisible by 1. The integers that are divisible by 2 are called the *even integers*, and the integers that are not divisible by 2 are called the *odd integers*.

Every integer $a \neq 0$ is divisible by itself.

There are many properties of divisibility. Some are listed in the following proposition, the proof of which is left as an exercise for students as it is elementary.

Proposition 1.2.4

Let $a, b, c \in \mathbb{Z}$ be integers.

(a) If $a \mid b$ and $b \mid c$, then $a \mid c$.
(b) If $a \mid b$ and $b \mid a$, then $a = \pm b$.
(c) If $a \mid b$ and $a \mid c$, then $a \mid (b+c)$ and $a \mid (b-c)$.
In fact, given any integers x and y, $a \mid (bx+cy)$. In other words, if a divides b and c, then it divides any integer linear combination of them.

Proof. Left as an exercise. □

Definition 1.2.5: GCD

A *common divisor* of two integers a and b is a positive integer d that divides both a and b. The *greatest common divisor* of a and b is the largest positive integer d such that $d \mid a$ and $d \mid b$. The greatest common divisor of a and b is denoted by $\gcd(a,b)$ or sometimes just by (a,b).

That is $d = \gcd(a,b)$ if

1. $d \mid a$ and $d \mid b$.
2. If $c \mid a$ and $c \mid b$, then $c \leq d$.

Example 1.2.6. The greatest common divisor of 12 and 18 is 6, since $6 \mid 12$ and $6 \mid 18$ and there is no larger number with this property. Similarly,

$$\gcd(748, 2024) = 44.$$

One way to check whether this is correct or not is to list all the positive divisors of 748 and 2044.
Divisors of $748 = \{1,2,4,11,17,22,34,44,68,187,374,748\}$.
Divisors of $2024 = \{1,2,4,8,11,22,23,44,46,88,92,184,253,506,1012,2024\}$.

From the above two lists, we can see that the greatest common divisor is 44. However, it is clear that this is not an efficient method to compute the GCD. *Division with remainder* called the *long division* is the key to compute the GCD efficiently. For example,

$$\begin{array}{r} 13 \\ 17\overline{)230} \\ \underline{17} \\ 60 \\ \underline{51} \\ 9 \end{array}$$

So 230 when divided by 17 gives a quotient of 13 with a remainder 9. It means 230 can be written as

$$230 = 17 \cdot 13 + 9,$$

where the remainder 9 is strictly smaller than the divisor 17. ∎

We state the result below known as the division algorithm and leave the proof as exercise.

Definition 1.2.7: Division Algorithm

Let a and b be positive integers. Then a divided by b has quotient q and remainder r means that

$$a = b \cdot q + r \qquad \text{with } 0 \leq r < b.$$

The values of q and r are uniquely determined by a and b.

Exercise 1.1

Given integers a and b, with $b > 0$, show that there exists unique integers q and r satisfying

$$a = qb + r, \qquad 0 \leq r < b.$$

Exercise 1.2

Using above exercise prove the following.
If a and b are integers, with $b \neq 0$, then there exists unique integers q and r satisfying

$$a = qb + r, \qquad 0 \leq r < |b|.$$

Suppose we want to find the greatest common divisor of a and b. First we divide a by b to get

$$a = b \cdot q + r \qquad \text{with } 0 \leq r < b. \tag{1.1}$$

Note that if $d \mid a$ and $d \mid b$, then $d \mid a - bq$, i.e., $d \mid r$. Thus, if d is a common divisor of a and b, then d is a common divisor of b and r. On the other hand, if e is a common divisor of b and r, then from equation (1.1), it follows that $e \mid a$. Thus, a common divisor of b and r is also a common divisor of a and b. Hence,

$$\gcd(a,b) = \gcd(b,r).$$

We repeat the process, dividing b by r to get quotient q' and remainder r' to get

$$b = r \cdot q' + r' \qquad \text{with } 0 \leq r' < r.$$

By the reason same as above, we have

$$\gcd(b,r) = \gcd(r,r').$$

Continuing this process, the remainders become smaller and smaller, until eventually we get remainder 0 at which the final value $\gcd(s,0) = s$ is equal to $\gcd(a,b)$.

We illustrate this process first with an example before describing the general method known as the *Euclidean Algorithm*.

Example 1.2.8. We compute $\gcd(748, 2044)$ using the Euclidean algorithm, which is nothing but repeated application of the division algorithm. Notice how the quotient and remainder at each step become the new a and b on the subsequent step.

$$\begin{aligned}
2024 &= 748 \cdot 2 + 528 \\
748 &= 528 \cdot 1 + 220 \\
528 &= 220 \cdot 2 + 88 \\
220 &= 88 \cdot 2 + 44 \qquad \leftarrow \boxed{\gcd = 44} \\
88 &= 44 \cdot 2 + 0
\end{aligned}$$

■

Theorem 1.2.9: The Euclidean Algorithm

Let a and b be positive integers with $a \geq b$. The following algorithm computes $\gcd(a,b)$ in a finite number of steps.

(1) Let $r_0 = a$ and $r_1 = b$.

(2) Set $i = 1$.

(3) Divide r_{i-1} by r_i to get a quotient q_i and remainder r_{i+1},

$$r_{i-1} = r_i \cdot q_i + r_{i+1} \qquad \text{with } 0 \le r_{i+1} < r_i.$$

(4) If the remainder $r_{i+1} = 0$, then $r_i = \gcd(a,b)$ and the algorithm terminates.

(5) Otherwise, $r_{i+1} > 0$, so set $i = i+1$ and go to Step 3.

The division step (Step 3) is executed at most

$$2\log_2(b) + 1 \quad \text{times.}$$

Proof. The Euclidean algorithm consists of a sequence of divisions with remainders as shown in the below. We have set $r_0 = a$ and $r_1 = b$.

$$\begin{aligned}
a &= b \cdot q_1 + r_2 && \text{with } 0 \le r_2 < b,\\
b &= r_2 \cdot q_2 + r_3 && \text{with } 0 \le r_3 < r_2,\\
r_2 &= r_3 \cdot q_2 + r_4 && \text{with } 0 \le r_4 < r_3,\\
r_3 &= r_4 \cdot q_2 + r_5 && \text{with } 0 \le r_5 < r_4,\\
&\vdots \qquad \vdots && \vdots\\
r_{t-2} &= r_{t-1} \cdot q_{t-1} + r_t && \text{with } 0 \le r_t < r_{t-1},\\
r_{t-1} &= r_t \cdot q_t
\end{aligned}$$

Then $r_t = \gcd(a,b)$.

Figure 1.2: The Euclidean algorithm step by step

The r_i values are strictly decreasing and as soon as they reach zero, the algorithm terminates. This proves that the algorithm finishes in a finite number of steps. Further, at the Step 3, we have an equation of the form

$$r_{i-1} = r_i \cdot q_i + r_{i+1}.$$

This equation implies that any common divisor of r_{i-1} and r_i is also a divisor of r_{i+1} and similarly any common divisor of r_i and r_{i+1} is also a divisor of r_{i-1}. Hence,

$$\gcd(r_{i-1}, r_i) = \gcd(r_i, r_{i+1}) \qquad \text{for all } i = 1,2,3,\ldots. \tag{1.2}$$

However, eventually we get to an r_i that is zero, say $r_{t+1} = 0$. Then $r_{t-1} = r_t \cdot q_t$. So

$$\gcd(r_{t-1}, r_t) = \gcd(r_t \cdot q_t, r_t) = r_t.$$

But by equation (1.2), $\gcd(r_{t-1}, r_t) = \gcd(r_0, r_1) = \gcd(a,b)$. This completes the proof that the last nonzero remainder in the Euclidean algorithm is equal to the greatest common divisor of a and b.

Now, we estimate the efficiency of the algorithm. As remarked earlier, since the values r_i are strictly decreasing, the algorithm terminates. Since $r_1 = b$, it terminates in at most b steps.

However, this is not an efficient bound. We claim that after every two iterations in Step 3, the value of remainder is at least cut in half. That is:

$$\textbf{Claim. } r_{i+2} < \tfrac{1}{2} r_i \quad \text{for all } i = 0, 1, 2, \ldots.$$

We prove the claim by considering two cases.
Case I: $r_{i+1} \leq \frac{1}{2} r_i$
Since the values of r_i are strictly decreasing,

$$r_{i+2} < r_{i+1} \leq \frac{1}{2} r_i.$$

Case II: $r_{i+1} > \frac{1}{2} r_i$ Since the value of r_{i+1} is so large, when we divide r_i by r_{i+1}, we get quotient as 1 and remainder r_{i+2} to write

$$r_i = r_{i+1} \cdot 1 + r_{i+2} \qquad \text{with } \; r_{i+2} = r_i - r_{i+1} < \frac{1}{2} r_i.$$

Hence, we have proved our claim that is $r_{i+2} < \frac{1}{2} r_i$ for all i. Using this inequality repeatedly, we get

$$r_{2k+1} < \frac{1}{2} r_{2k-1} < \frac{1}{4} r_{2k-3} < \frac{1}{8} r_{2k-5} < \cdots < \frac{1}{2^k} r_1 = \frac{1}{2^k} b.$$

Hence if $2^k \geq b$, then $r_{2k+1} < 1$ which forces r_{2k+1} equal to 0 and the algorithm to terminate. In terms of Figure 1.2, the value of r_{t+1} is 0, so we have $t+1 \leq 2k+1$ and thus $t \leq 2k$. Furthermore, there are exactly t division performed in Figure 1.2, so the Euclidean algorithm terminates in at most $2k$ iterations. Choose the smallest such k, so $2^k \geq b > 2^{k-1}$.
Then $(k-1)\log 2 < \log b \leq k \log 2$. Dividing by $\log 2$, we get $k - 1 < \frac{\log_{10} b}{\log_{10} 2} < k$. Then by change of base, we have $k - 1 < \log_2 b < k$. Therefore,

$$\#\text{ of iterations } \leq 2k = 2(k-1) + 2 < 2\log_2(b) + 2.$$

Thus, the number of divisions (at Step 3) is executed at most $2\log_2(b) + 1$ times. This completes the proof of the theorem. □

Remark 1.2.10. We proved that the Euclidean algorithm applied to a and b with $a \geq b$ requires at most $2\log_2(b) + 1$ iterations to compute $\gcd(a, b)$. It has been proved that this estimate was improved to $1.45\log_2(b) + 1.68$ iterations.

From the Euclidean algorithm applied on a and b, we can work our way back up the process to obtain an interesting formula involving a, b, and $\gcd(a, b)$. Let us first illustrate it with the following example before coming to the general result.

Example 1.2.11. Recall the Euclidean algorithm in Example 1.2.8 to compute $\gcd(2024, 748)$.

$$\begin{aligned}
2024 &= 748 \cdot 2 + 528 \\
748 &= 528 \cdot 1 + 220 \\
528 &= 220 \cdot 2 + 88 \\
220 &= 88 \cdot 2 + 44 \qquad \leftarrow \boxed{\gcd = 44} \\
88 &= 44 \cdot 2 + 0
\end{aligned}$$

We let $a = 2024$ and $b = 748$. Then from the first step of the algorithm, we get

$$528 = a - 2b.$$

We substitute this in the second line to get

$$b = (a - 2b) \cdot 1 + 220, \qquad \text{so} \qquad 220 = -a + 3b.$$

Next we substitute the expressions $528 = a - 2b$ and $220 = -a + 3b$ into the third line to get

$$a - 2b = (-a + 3b) \cdot 2 + 88, \qquad \text{so} \qquad 88 = 3a - 8b.$$

Finally, we substitute the expressions $220 = -a + 3b$ and $88 = 3a - 8b$ into the last line to get

$$-a + 3b = (3a - 8b) \cdot 2 + 44, \qquad \text{so} \qquad 44 = -7a + 19b.$$

In other words,

$$-7 \cdot 2024 + 19 \cdot 748 = 44 = \gcd(2024, 748).$$

Thus, we have found a way to write $\gcd(a,b)$ as an integer linear combination of a and b using the integer coefficients. ∎

In general, $\gcd(a,b)$ can be written as an integer linear combination of a and b. We have the following result called the Extended Euclidean Algorithm. It is also called Bezout's identity (the first part of the result) and it has many important consequences.

Theorem 1.2.12: Extended Euclidean Algorithm

Let a and b be positive integers. Then the equation

$$au + bv = \gcd(a,b)$$

always has a solution in integers u and v.

If (u_0, v_0) is any one solution, then every solution has the form

$$u = u_0 + \frac{b \cdot k}{\gcd(a,b)} \quad \text{and} \quad v = v_0 + \frac{a \cdot k}{\gcd(a,b)} \qquad \text{for some } k \in \mathbb{Z}.$$

Proof. Recall Figure 1.2 which illustrates the Euclidean algorithm step by step. From the fist step, we get $r_2 = a - b \cdot q_1$. Substituting this in the second line we get

$$b = (a - b \cdot q_1) \cdot q_2 + r_3, \qquad \text{so} \qquad r_3 = -a \cdot q_2 + b \cdot (1 + q_1 q_2).$$

Next substitute the expression for r_2 and r_3 into the third line to get

$$a - b \cdot q_1 = (-a \cdot q_2 + b(1 + q_1 q_2))q_3 + r_4.$$

After rearranging the terms, we get

$$r_4 = a \cdot (1 + q_2 q_3) - b \cdot (q_1 + q_3 + q_1 q_2 q_3).$$

Thus, $r_4 = a \cdot u + b \cdot v$, where u and v are integers. Continuing this way, at each stage we find that r_i is the sum of an integer multiple of a and an integer multiple of b. Eventually, we get

$$r_t = a \cdot u + b \cdot v.$$

But $r_t = \gcd(a,b)$. This completes the proof of the first part.

The proof of the second part is left as a *seminar exercise*. □

A particular and important case of the extended Euclidean algorithm arises when the greatest common divisor of a and b in 1. In this case, a and b are given a special name.

Definition 1.2.13

Let a and b be integers. We say that a and b are *relatively prime* if $\gcd(a,b) = 1$.

Note that any equation of the form

$$Au + Bv = \gcd(A,B)$$

can be reduced to the case of relatively prime numbers by dividing both sided by $\gcd(A,B)$. Thus,

$$\frac{A}{\gcd(A,B)}u + \frac{B}{\gcd(A,B)}v = 1,$$

where $a = A/\gcd(A,B)$ and $b = B/\gcd(A,B)$ are relatively prime and satisfying $au + bv = 1$. For example, $\gcd(2024,748) = 44$ satisfying

$$-7 \cdot 2024 + 19 \cdot 748 = 44.$$

Dividing both the sides by 44, we get

$$-7 \cdot 46 + 19 \cdot 17 = 1.$$

Thus, $\frac{2024}{44} = 46$ and $\frac{748}{44} = 17$ are relatively prime numbers with $u = -7$ and $v = 19$ as coefficients of their integer linear combination which is equal to 1.

Exercise 1.3

Let a and b be positive integers.

(a) Suppose that there are integers u and v satisfying $au + bv = 1$. Prove that $\gcd(a,b) = 1$.

(b) Suppose that there are integers u and v satisfying $au + bv = 6$. Is it necessarily true that $\gcd(a,b) = 6$? If not, give a specific counterexample, and describe in general all of the possible values of $\gcd(a,b)$.

(c) Suppose that (u_1, v_2) and (u_2, v_2) are two solutions in integers to the equation $au + bv = 1$. Prove that a divides $v_2 - v_1$ and b divides $u_2 - u_1$.

(d) More generally, let $g = \gcd(a,b)$ and let (u_0, v_0) be a solution in integers to $au + bv = g$. Prove that every other solution has the form $u = u_0 + kb/g$ and $v = v_0 - ka/g$ for some integer k. (This is the second part of Theorem 1.2.12.)

Just as in Example 1.2.8, by means of an example, we describe another algorithm to compute the coefficients of the integer linear combinations of a and b in the case when $\gcd(a,b) = 1$.

Example 1.2.14. Take $a = 73$ and $b = 25$. The Euclidean algorithm gives

$$\begin{aligned} 73 &= 25 \cdot 2 + 23 \\ 25 &= 23 \cdot 1 + 2 \\ 23 &= 2 \cdot 11 + 1 \\ 2 &= 1 \cdot 2 + 0. \end{aligned}$$

We set up a box, using the sequence of quotients $2, 1, 11$, and 2 as follows:

		2	1	11	2
0	1	$*$	$*$	$*$	$*$
1	0	$*$	$*$	$*$	$*$

The rule to fill in the remaining entries in as follows:

> New Entry = (Number of Top) · (Number to the Left)
> + (Number Two Spaces to the Left).

Thus, the two leftmost $*$'s (in the third column) are

$$2 \cdot 1 + 0 = 2 \qquad \text{and} \qquad 2 \cdot 0 + 1 = 1,$$

so now our box looks as follows:

		2	1	11	2
0	1	2	$*$	$*$	$*$
1	0	1	$*$	$*$	$*$

Then the next two leftmost $*$'s (in the fourth column) are

$$1 \cdot 2 + 1 = 3 \qquad \text{and} \qquad 1 \cdot 1 + 0 = 1,$$

and then the next column entries are

$$11 \cdot 3 + 2 = 35 \qquad \text{and} \qquad 11 \cdot 1 + 1 = 12,$$

and the final entries are

$$2 \cdot 35 + 3 = 73 \qquad \text{and} \qquad 2 \cdot 12 + 1 = 25.$$

The completed box becomes

		2	1	11	2
0	1	2	3	35	73
1	0	1	1	12	25

Note that the last column repeats the values of a and b. More importantly, the next to last column gives the values of $-v$ and u in that order. Thus, in the example, we find that

$$73 \cdot 12 - 25 \cdot 35 = 1.$$

■

The general algorithm of the above example is given in Figure 1.2.

If a and b are relatively prime and if $q_1, q_2, \ldots, q_t$ is the sequence of quotients obtained from applying the Euclidean algorithm to a and b (as in Figure 1.2), then the box has the following form.

		q_1	q_2	$\cdots$	q_{t-1}	q_t
0	1	P_1	P_2	$\cdots$	P_{t-1}	a
1	0	Q_1	Q_2	$\cdots$	Q_{t-1}	b

The entries in the box are calculated using the initial values

$$P_1 = q_1, \quad Q_1 = 1, \quad P_2 = q_2 \cdot P_1 + 1, \quad Q_2 = q_2 \cdot Q_1,$$

and then, for $i \geq 3$, using the formulas

$$P_i = q_i \cdot P_{i-1} + P_{i-2} \quad \text{and} \quad Q_i = q_i \cdot Q_{i-1} + Q_{i-2}.$$

The final four entries in the box satisfy (**How?**)

$$a \cdot Q_{t-1} - b \cdot P_{t-1} = (-1)^t.$$

Multiplying both sides by $(-1)^t$ gives the solution $u = (-1)^t Q_{t-1}$ and $v = (-1)^{t+1} P_{t-1}$ to the equation $au + bv = 1$.

Figure 1.3: Solving $au + bv = 1$ using the Euclidean algorithm

1.3 Modular arithmetic

Definition 1.3.1

Let $m \geq 1$ be an integer. We say that the integers a and b are *congruent modulo m* if their difference $a - b$ is divisible by m, i.e., $m \mid a - b$. We denote it by

$$a \equiv b \pmod{m}.$$

The number m is called the *modulus*.

Example 1.3.2. We have

$$17 \equiv 7 \pmod{5}, \qquad \text{since 5 divides } 10 = 17 - 7.$$

On the other hand,

$$19 \not\equiv 6 \pmod{11}, \qquad \text{since 11 does not divide } 13 = 16 - 6.$$

We convert the time/hours in our digital clock using congruences modulo 12 as

$$6+9 = 15 \equiv 3 \pmod{12} \qquad \text{and} \qquad 2-3 = -1 \equiv 11 \pmod{12}.$$

■

Note the numbers satisfying $a \equiv 0 \pmod{m}$ are the numbers that are divisible by m, i.e., the multiples of m.
The following result states some of the properties of congruences which indicates that congruences behave like equalities.

Proposition 1.3.3

Let $m \geq 1$ be an integer.

(a) If $a_1 \equiv a_2 \pmod{m}$ and $b_1 \equiv b_2 \pmod{m}$, then

$$a_1 \pm b_1 \equiv a_2 \pm b_2 \pmod{m} \quad \text{and } a_1 \cdot b_1 \equiv a_2 \cdot b_2 \pmod{m}.$$

(b) Let a be an integer. Then

$$a \cdot b \equiv 1 \pmod{m} \text{ for some integer } b \text{ if and only if } \gcd(a,m) = 1.$$

Further if $a \cdot b_1 \equiv a \cdot b_2 \equiv 1 \pmod{m}$, then $b_1 \equiv b_2 \pmod{m}$. We say that b is the (multiplicative) inverse of a modulo m. (We say "the" inverse instead of "an" inverse, because any two inverses are congruent modulo m.)

Proof. (a) Seminar exercise.

(b) First assume that $\gcd(a,m) = 1$. Then by the Extended Euclidean algorithm, there exists integers u and v such that $au + mv = 1$. Then $au - 1 = -mv$. Then $au - 1$ is divisible by m. So by the definition of congruence, $au - 1 \equiv 0 \pmod{m}$ or $au \equiv 1 \pmod{m}$. Thus, we can take $b = u$.

Conversely, assume that a has an inverse modulo m, say b. That is, $a \cdot b \equiv 1 \pmod{m}$. This means that $ab - 1 = cm$ for some integer c. Then $ab - cm = 1$. Let $d = \gcd(a,m)$. Then $d \mid a$ and $d \mid m$. Then $d \mid ab - cm$, i.e., $d \mid 1$. Since d is a positive integer, $d = 1$. Thus, $\gcd(a,m) = 1$.

It remains to show that the inverse b is unique modulo m. Suppose that $a \cdot b_1 \equiv a \cdot b_2 \equiv 1 \pmod{m}$. Then

$$b_1 \equiv b_1 \cdot 1 \equiv b_1 \cdot (a \cdot b_2) \equiv (b_1 \cdot a) \cdot b_2 \equiv 1 \cdot b_2 \equiv b_2 \pmod{m}.$$

□

By Proposition 1.3.3 (b), if $\gcd(a,m) = 1$, then there exists an inverse b of a modulo m. Thus, $a^{-1} = \frac{1}{a}$ makes sense the in integers modulo m. In face, a^{-1} is the unique integer b modulo m satisfying the congruence $a \cdot b \equiv 1 \pmod{m}$.

Example 1.3.4. Take $m = 5$ and $a = 2$. Since $\gcd(2,5) = 1$, there exists inverse of 2 modulo 5. The inverse of 2 modulo 5 is 3, since $2 \cdot 3 \equiv 1 \pmod 5$. So $2^{-1} \equiv 3 \pmod 5$. Similarly, $\gcd(4,15) = 1$ and so 4^{-1} exists modulo 15. In fact, $4 \cdot 4 \equiv 1 \pmod{15}$, i.e., 4 is its own inverse modulo 15.
We can also work with fractions $\frac{a}{d}$ modulo m provided that the denominator is relatively prime to m. For example, we can compute $5/7$ modulo 11. Observe that $7 \cdot 8 \equiv 1 \pmod{11}$ and so $7^{-1} \equiv 8 \pmod{11}$. Then

$$\frac{5}{7} = 5 \cdot 7^{-1} \equiv 5 \cdot 8 \equiv 40 \equiv 7 \pmod{11}.$$

■

Remark 1.3.5. In the above example, we computed the inverse modulo m by trial and error method. However, if m is large, then computing a^{-1} modulo m can be more challenging. Note that we showed that inverses exists by using the extended Euclidean algorithm. But in order to actually determine integers u and v such that $au + mv = \gcd(a,m)$, we can apply the Euclidean algorithm or we can use the more efficient box method.
If a divided by m has quotient q and remainder r, then it can be written as

$$a = m \cdot q + r \qquad \text{with } 0 \leq r < m.$$

This show that $a \equiv r \pmod m$ for some integer r between 0 and $m-1$. Thus, if we work with integers modulo m then it enough to use the integers $0 \leq r < m$ and we have the following definition.

Definition 1.3.6

We write

$$\mathbb{Z}/m\mathbb{Z} = \{0, 1, 2, \ldots, m-1\}$$

and call $\mathbb{Z}/m\mathbb{Z}$ the *ring of integers modulo m*.
Note that whenever we perform addition or multiplication in the ring $\mathbb{Z}/m\mathbb{Z}$, we divide the result by m and take the remainder to obtain an element in $\mathbb{Z}/m\mathbb{Z}$.

Figure 1.4 illustrates the ring $\mathbb{Z}/5\mathbb{Z}$ by giving the addition and multiplications tables modulo 5.

+	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

·	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Figure 1.4: Addition and multiplication tables modulo 5

Definition 1.3.7

Proposition 1.3.3 (b) tells that a has an inverse modulo m if and only if $\gcd(a,m) = 1$. Numbers that have inverses are called *units*. We denote the set of all units by

$$\begin{aligned}(\mathbb{Z}/m\mathbb{Z})^* &= \{a \in \mathbb{Z}/m\mathbb{Z} : \gcd(a,m) = 1\} \\ &= \{a \in \mathbb{Z}/m\mathbb{Z} : a \text{ has an inverse modulo } m\}.\end{aligned}$$

The set $(\mathbb{Z}/m\mathbb{Z})^*$ is called the *group of units modulo m*.

Note that if a_1 and a_2 are units modulo m, then a_1a_2 is also a unit (**Why?**). So when we multiple two units, we always get a unit. However, this is not true with addition. That is, if we add two units, then we often do not get a unit.

Example 1.3.8. The group of units modulo 24 is

$$(\mathbb{Z}/24\mathbb{Z})^* = \{1,5,7,11,13,17,19,23\}.$$

The multiplication table for $(\mathbb{Z}/24\mathbb{Z})^*$ is shown in Figure 1.5. ∎

Example 1.3.9. The group of units modulo 7 is

$$(\mathbb{Z}/7\mathbb{Z})^* = \{1,2,3,4,5,6\},$$

since every number between 1 and 6 is relatively prime to 7. The multiplication table for $(\mathbb{Z}/7\mathbb{Z})^*$ is shown in Figure 1.5. ∎

·	1	5	7	11	13	17	19	23
1	1	5	7	11	13	17	19	23
5	5	1	11	7	17	13	23	19
7	7	11	1	5	19	23	13	17
11	11	7	5	1	23	19	17	13
13	13	17	19	23	1	5	7	11
17	17	13	23	19	5	1	11	7
19	19	23	13	17	7	11	1	5
23	23	19	17	13	11	7	5	1

Unit groups modulo 24

·	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

Unit groups modulo 7

Figure 1.5: The unit groups $(\mathbb{Z}/24\mathbb{Z})^*$ and $(\mathbb{Z}/7\mathbb{Z})^*$

The number of elements in the unit group modulo m is very important in many of the cryptosystems that we will study. Therefore this quantity is important and so we give it a name in the following definition.

Definition 1.3.10

Euler's phi function also sometimes known as *Euler's totient function* is the function $\phi(m)$ defined by the rule

$$\phi(m) = \#(\mathbb{Z}/m\mathbb{Z})^* = \#\{0 \leq a < m : \gcd(a,m) = 1\}.$$

For example, as seen in Examples 1.3.8 and 1.3.9, we have $\phi(24) = 8$ and $\phi(7) = 6$.

1.3.1 Modular Arithmetic and Shift Ciphers

Recall that the Caesar (or shift) cipher works by shifting each letter in the alphabet by a fixed number of letters. We describe a shift cipher mathematically by assigning a number to each letter as in Table 1.3.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Table 1.3: Assigning numbers to letters

Then a shift cipher with shift k takes a plaintext letter corresponding to number p and assigns it to the ciphertext letter corresponding to the number $p+k \mod 26$. Note that the use of modular arithmetic simplifies the description of the shift cipher. The shift amount serves as both the encryption and the decryption key. Encryption is given by the formula

$$\text{(Ciphertext Letter)} \equiv \text{(Plaintext Letter) + (Secret Key)} \pmod{26},$$

and the decryption works by shifting in the opposite direction.

$$\text{(Plaintext Letter)} \equiv \text{(Ciphertext Letter) - (Secret Key)} \pmod{26}.$$

If we take

$$p = \text{Plaintext Letter}, \qquad c = \text{Ciphertext Letter}, \qquad k = \text{Secret Key},$$

then

$$\underbrace{c \equiv p + k \pmod{26}}_{\text{Encryption}} \quad \text{and} \quad \underbrace{p \equiv c - k \pmod{26}}_{\text{Decryption}}.$$

1.3.2 The Fast Powering Algorithm

In certain cryptosystems like RSA and Diffie-Hellman cryptosystems, Alice and Bob are required to compute large powers of some number g modulo some other number N, where N may have hundreds of digits. The naive way to compute g^A is by repeated multiplication of g. That is

$$g_1 \equiv g \pmod{N}, \qquad g_2 \equiv g \cdot g_1 \pmod{N}, \qquad g_3 \equiv g \cdot g_2 \pmod{N},$$

$$g_4 \equiv g \cdot g_3 \pmod{N}, \qquad g_5 \equiv g \cdot g_4 \pmod{N}, \ldots.$$

It is clear that $g_A \equiv g^A \pmod{N}$. But if A is large, then this algorithm is impractical to implement. Thus, we need a better way to compute $g^A \pmod{N}$. The idea is to use the binary expansion of the exponent A to convert the calculation of g^A into a succession of squarings and multiplications. Before we describe the method, consider the following example for a better idea and understanding.

Example 1.3.11. Compute $3^{218} \pmod{1000}$. ▪

Solution. The fist step is to write 218 are a sum of powers of 2.

2	218			
2	109	0		
2	54	1		2^1
2	27	0		
2	13	1		2^3
2	6	1		2^4
2	3	0		
2	1	1		2^6
2	0	1	↑	2^7

Thus, $218 = (11011010)_2$ and so we can write

$$218 = 2 + 2^3 + 2^4 + 2^6 + 2^7.$$

Then 2^{218} becomes

$$3^{218} = 3^{2+2^3+2^4+2^6+2^7} = 3^2 \cdot 3^{2^3} \cdot 3^{2^4} \cdot 3^{2^6} \cdot 3^{2^7}. \tag{1.3}$$

Note that it is relatively easy to compute the sequence of values

$$3, 3^2, 3^{2^2}, 3^{2^3}, 3^{2^4}, \ldots,$$

since each number in the sequence is the square of the preceding one. Further, since we only need these values modulo 1000, we need not store more than the last three digits. Table 1.4 lists all the powers of 3 modulo 1000 upto 3^{2^7}. It requires only 7 multiplications although $3^{2^7} = 3^{128}$ is a very large exponent. This is because each successive entry in the table is equal to the square of the previous entry.

i	0	1	2	3	4	5	6	7
$3^{2^i} \pmod{1000}$	3	9	81	561	721	841	281	961

Table 1.4: Successive square powers of 3 modulo 1000

We use expression (1.3) to determine which powers from Table 1.4 are needed to compute 3^{218}. Thus,

$$\begin{aligned} 3^{218} &= 3^2 \cdot 3^{2^3} \cdot 3^{2^4} \cdot 3^{2^6} \cdot 3^{2^7} \\ &\equiv 9 \cdot 561 \cdot 721 \cdot 281 \cdot 961 \pmod{1000} \\ &\equiv 489 \pmod{1000}. \end{aligned}$$

□

Note that in computing the product $9 \cdot 561 \cdot 721 \cdot 281 \cdot 961$, after each multiplication we reduce it to modulo 100 so that we never have to deal with very large numbers. Also note that it took us only 11 multiplications to compute $3^{218} \pmod{1000}$ which is very much efficient as compared to the naive method.
The method described in Example 1.3.11 is known as the *Fast Powering Algorithm* and the *Square-and-Multiply Algorithm*. We now describe the algorithm formally as follows.

The Fast Powering Algorithm

Step 1. Compute the binary expansion of A as

$$A = A_0 + A_1 \cdot 2 + A_2 \cdot 2^2 + A_3 \cdot 2^3 + \cdots + A_r \cdot 2^r \qquad \text{with } A_0, \ldots, A_r \in \{0,1\},$$

where we may assume that $A_r = 1$.

Step 2. Compute the powers $g^{2^i} \pmod{N}$ for $0 \leq i \leq r$ by successive squaring,

$$\begin{aligned}
a_0 &\equiv g && \pmod{N} \\
a_1 &\equiv a_0^2 \equiv g^2 && \pmod{N} \\
a_2 &\equiv a_1^2 \equiv g^{2^2} && \pmod{N} \\
a_3 &\equiv a_2^2 \equiv g^{2^3} && \pmod{N} \\
&\vdots \equiv \vdots && \vdots \\
a_r &\equiv a_{r-1}^2 \equiv g^{2^r} && \pmod{N}.
\end{aligned}$$

Each term is the square of the previous one, so this requires r multiplications.

Step 3. Compute $g^A \pmod{N}$ using the formula

$$\begin{aligned}
g^A &= g^{A_0 + A_1 \cdot 2 + A_2 \cdot 2^2 + A_3 \cdot 2^3 + \cdots + A_r \cdot 2^r} \\
&= a^{A_0} \cdot (g^2)^{A_1} \cdot \left(g^{2^2}\right)^{A_2} \cdot \left(g^{2^3}\right)^{A_3} \cdots \left(g^{2^r}\right)^{A_r} \\
&\equiv a_0^{A_0} \cdot a_1^{A_1} \cdot a_2^{A_2} \cdot a_3^{A_3} \cdots a_r^{A_r} \pmod{N}. \qquad (1.4)
\end{aligned}$$

Note that the quantities $a_0, a_1, \ldots, a_r$ were computed in Step 2. Thus, the product (1.4) can be computed by looking up the values of the a_i's whose exponent A_i is 1 and then multiplying them together. This requires at most another r multiplications.

Running Time. It takes at most $2r$ multiplications modulo N to compute g^A. Since $A \geq 2^r$, we see that it takes at most $2\log_2(A)$ multiplications modulo N to compute g^A. Thus, even if A is very large, say $A \approx 2^{1000}$, it is easy for a computer to do the approximately 2000 multiplications needed to compute 2^A modulo N.

Efficiency Issues. There are various ways in which the square-and-multiply algorithm can be made somewhat more efficient, in particular regarding eliminating storage requirements.

1.4 Prime numbers, unique factorization, and finite fields

We have seen that we can add, subtract, and multiply integers modulo m. However, division by a is possible in $\mathbb{Z}/m\mathbb{Z}$ only if $\gcd(a,m) = 1$. But if m is prime, then we can divide by every nonzero element of $\mathbb{Z}/m\mathbb{Z}$.

Definition 1.4.1

An integer p is called a *prime* if $p \geq 2$ and if the only positive integers dividing p are 1 and p.

For example, the first ten primes are $2,3,5,7,11,13,17,19,23,29$, while the thousand-th prime is 7919 and the million-th prime is 15485863. The fact that there are infinitely many primes was known in ancient Greece and appears as a theorem in Euclid's *Elements*.
A number greater 1 is called *composite* if it is not a prime. A prime p is defined in terms of numbers that divide p. The following proposition gives a useful property of primes.

Proposition 1.4.2

Let p be a prime number, and suppose that p divides the product ab of two integers a and b. Then p divides at least one of a and b.
More generally, if p divides a product of integers, say

$$p \mid a_1 a_2 \cdots a_n,$$

then p divides at least one of the individual a_i.

Proof. If $p \mid a$, then we are done. If $p \nmid a$, then $\gcd(a,p) = 1$. Then by the extended Euclidean algorithm (Theorem 1.2.12), there exist integers u and v such that $au + pv = 1$. Multiplying both sides of the equation by b, we get

$$abu + pbv = b. \tag{1.5}$$

By our assumption $p \mid ab$. Also $p \mid pbv$. So p divides both the terms on the left hand side of equation (1.5). Hence, it divides the right hand side, i.e., $p \mid b$.
Now we prove the general statement. Let $p \mid a_1 a_2 \cdots a_n$. We write the product as $a_1(a_2 \cdots a_n)$ and let $a = a_1$ and $b = a_2 \cdots a_n$. Then by the above argument, $p \mid a_1$ or $p \mid a_2 \cdots a_n$. If $p \mid a_1$, then we are done. If $p \mid a_2 \cdots a_n$, then write the product as $a_2(a_3 \cdots a_n)$. Again by the above argument, $p \mid a_2$ or $p \mid a_3 \cdots a_n$. Continuing in this way, we eventually find some a_i such that $p \mid a_i$. □

Note that the above proposition is not true for the composite numbers. For example, $6 \mid 3 \cdot 10$ but neither 6 divides 3 nor 6 divides 10.
As an application of Proposition 1.4.2, we prove that every integer greater than 1 can be essentially uniquely expressed as a product of primes.

Theorem 1.4.3: The Fundamental Theorem of Arithmetic

Let $a \geq 2$ be an integer. Then a can be factored as a product of prime numbers

$$a = p_1^{e_1} \cdot p_2^{e_2} \cdot p_2^{e_3} \cdots p_r^{e_r}.$$

Further, other than rearranging the order of primes, this factorization into prime powers is unique.

Proof. Prove (as an exercise) that every $a \geq 2$ can be written as a product of primes.

We prove the uniqueness. Suppose a has two factorizations into product of primes, say,

$$a = p_1 p_2 \cdots p_s = q_1 q_2 \cdots q_t, \tag{1.6}$$

where p_i and q_j are all primes, not necessarily distinct, and without the loss of generality assume $s \leq t$. Since $p_1 \mid a$, p_1 divides the product $q_1 q_2 \cdots q_t$. Since p_1 is prime (by Proposition 1.4.2), p_1 divides q_i for some i. Rearranging the order of the q_i if necessary, we may assume that $p_1 \mid q_1$. Since p_1 and q_1 are both prime, we must have $p_1 = q_1$. Cancelling them from both the sides in equation (1.6), we get

$$p_2 p_3 \cdots p_s = q_2 q_3 \cdots q_t.$$

Repeating this process s times, we get the equation of the form

$$1 = q_{t-s} q_{t-s+1} \cdots q_t.$$

Since q_j's are primes, this is not possible. Hence, it follows that $t = s$ and that $t = s$ and the factorizations of a in (1.6) were identical up to rearranging the order of the factors. □

Definition 1.4.4

The fundamental theorem of arithmetic (Theorem 1.4.3) says that in the factorization of positive integer a into primes, each prime p appears to a particular power. We denote this power by $\mathrm{ord}_p(a)$ and call it the *order (or exponent) of p in a*.
For convenience, we set $\mathrm{ord}_p(1) = 0$ for all primes.

For example, the factorization of 1728 is $1728 = 2^6 \cdot 3^3$, so $\mathrm{ord}_2(1728) = 6$, $\mathrm{ord}_3(1728) = 3$, and $\mathrm{ord}_p(1728) = 0$ for all primes $p \geq 5$.
Using the ord_p notation, the factorization of a can be written as

$$a = \prod_{\text{primes } p} p^{\mathrm{ord}_p(a)}.$$

Note that the product makes sense (that is, it is finite) since $\mathrm{ord}_p(a)$ is zero for all but finitely many primes.
It is useful to view ord_p as a function

$$\mathrm{ord}_p : \{1, 2, 3, \ldots\} \to \{0, 1, 2, 3, \ldots\}. \tag{1.7}$$

Now we prove that if p is a prime, then every nonzero number modulo p has an inverse.

Proposition 1.4.5

Let p be a prime. Then every nonzero element a in $\mathbb{Z}/p\mathbb{Z}$ has a multiplicative inverse, that is, there is a number b satisfying

$$ab \equiv 1 \pmod{p}.$$

We denote this value of b by $a^{-1} \mod p$, or if p has already been specified, then simply by a^{-1}.

Proof. This proposition is a special case of Proposition 1.3.3 (b) where the modulus m is the prime p, since if $a \in \mathbb{Z}/p\mathbb{Z}$ is nonzero, then $\gcd(a,p) = 1$.
Then by Proposition 1.3.3 (b), it has an inverse (modulo p). □

The above proposition can be restated by saying that if p is prime, then

$$(\mathbb{Z}/p\mathbb{Z})^* = \{1,2,3,\ldots,p-1\}.$$

In other words, all nonzero elements in $(\mathbb{Z}/p\mathbb{Z})$ are units and when 0 is removed from $(\mathbb{Z}/p\mathbb{Z})$, then the remaining elements are closed under multiplication.

Remark 1.4.6. The extended Euclidean algorithm gives us an efficient computational method to compute $a^{-1} \mod p$. We simply solve the equation

$$au + pv = 1$$

in integers u and v, and then $u = a^{-1} \mod p$.

Definition 1.4.7

If p is prime, then the set $\mathbb{Z}/p\mathbb{Z}$ of integers modulo p with its addition, subtraction, multiplication, and division rules is an example of a *field*.
The field $\mathbb{Z}/p\mathbb{Z}$ of integers modulo p has only finitely many elements. It is a *finite field* and is often denoted by $\mathbb{F}_p$. Thus, $\mathbb{F}_p$ and $\mathbb{Z}/p\mathbb{Z}$ denote the same object. Similarly, we denote by $\mathbb{F}_p^*$, the group of units $(\mathbb{Z}/p\mathbb{Z})^*$.

Although $\mathbb{Z}/p\mathbb{Z}$ and $\mathbb{F}_p$ are used to denote the same object, the equality of elements is different in both of them. For $a,b \in \mathbb{F}_p$, they are equal if $a = b$, while for $a,b \in \mathbb{Z}/p\mathbb{Z}$, by equality of a and b we mean equivalence modulo p, i.e., $a \equiv b \pmod{p}$.

1.5 Powers and Primitive Roots in Finite Fields

In cryptography, it is often required to raise the elements of the finite field $\mathbb{F}_p$ to high powers. We have seen how to do this using the powering algorithm described in an earlier section. In this section, we investigate powers of $\mathbb{F}_p$ from a purely mathematical point of view. We prove a fundamental result due to Fermat and state and important property of the group of units in $\mathbb{F}_p^*$. We begin with an example. The following table lists the powers of $1,2,3,\ldots,6$ modulo 7.

$1^1 \equiv 1$	$1^2 \equiv 1$	$1^3 \equiv 1$	$1^4 \equiv 1$	$1^5 \equiv 1$	$1^6 \equiv 1$
$2^1 \equiv 2$	$2^2 \equiv 4$	$2^3 \equiv 1$	$2^4 \equiv 2$	$2^5 \equiv 4$	$2^6 \equiv 1$
$3^1 \equiv 3$	$3^2 \equiv 2$	$3^3 \equiv 6$	$3^4 \equiv 4$	$3^5 \equiv 5$	$3^6 \equiv 1$
$4^1 \equiv 4$	$4^2 \equiv 2$	$4^3 \equiv 1$	$4^4 \equiv 4$	$4^5 \equiv 2$	$4^6 \equiv 1$
$5^1 \equiv 5$	$5^2 \equiv 4$	$5^3 \equiv 6$	$5^4 \equiv 2$	$5^5 \equiv 3$	$5^6 \equiv 1$
$6^1 \equiv 6$	$6^2 \equiv 1$	$6^3 \equiv 6$	$6^4 \equiv 1$	$6^5 \equiv 6$	$6^6 \equiv 1$

Table 1.5: Powers of numbers modulo 7

From the above table, we can observe an interesting pattern in its last column. We have

$$a^6 \equiv 1 \pmod 7 \quad \text{for every } a = 1,2,\ldots,6.$$

However, this is true only if a is not a multiple of 7 because if 7 divides a, then $a \equiv 0 \pmod 7$. Then for all the powers n of a, we have $a^n \equiv 0 \pmod 7$. Hence, we have

$$a^6 \equiv \begin{cases} 1 \pmod 7 & \text{if } 7 \nmid a, \\ 0 \pmod 7 & \text{if } 7 \mid a. \end{cases}$$

Theorem 1.5.1: Fermat's Little Theorem

Let p be a prime number and a be any integer. Then

$$a^{p-1} \equiv \begin{cases} 1 \pmod p & \text{if } p \nmid a, \\ 0 \pmod p & \text{if } p \mid a. \end{cases}$$

Proof. If p divides a, then $a \equiv 0 \pmod p$. Then for all the powers n of a, we have $a^n \equiv 0 \pmod p$. So let us assume that $p \nmid a$. Consider the following list of numbers.

$$a,\ 2a,\ 3a,\ \ldots,\ (p-1)a \quad \text{reduced modulo } p. \tag{1.8}$$

Claim. All the $p-1$ numbers in the above list are distinct.
Suppose if possible $ja \equiv ka \pmod p$ for some j and k with $1 \le j,k \le p-1$. Without the loss of generality, we may assume that $j \ge k$. Then

$$(j-k)a \equiv 0 \pmod p.$$

Thus, $p \mid (j-k)a$. Since p is prime, $p \mid j-k$ or $p \mid a$. But by our assumption $p \nmid a$. Therefore, $p \mid j-k$. Since $1 \le j,k \le p-1$ and $j \ge k$, we have $0 \le j-k \le p-2$. Since $p \mid j-k$, the only possibility is $j-k=0$. This means $j=k$ and that all the numbers in the list (1.8) are distinct. Further, they are all nonzero as p does not divide $1,2,\ldots,p-1$ and a.
Since there are $p-1$ distinct nonzero numbers modulo p in the list 1.8, we have

$$\{a,2a,\ldots,(p-1)a\} = \{1,2,\ldots,p-1\}.$$

Thus, multiplying all the numbers in (1.8), we get

$$a \cdot 2a \cdot 3a \cdots (p-1)a \equiv 1 \cdot 2 \cdot 3 \cdots (p-1) \pmod p.$$

This gives $a^{p-1}(p-1)! \equiv (p-1)! \pmod p$. Since $p \nmid (p-1)!$, it has an inverse in $\mathbb{F}_p$. Hence, cancelling $(p-1)!$ from both the sides, we get

$$a^{p-1} \equiv 1 \pmod p.$$

This completes the proof of the Fermat's little theorem. □

There are many proofs of the Fermat's little theorem. One using group theory is the quickest by observing that the set of nonzero elements of $\mathbb{F}_p$ forms a group $\mathbb{F}_p^*$ of order $p-1$. So by Lagrange's theorem, order of every element of $\mathbb{F}_p^*$ divides $p-1$.

Example 1.5.2. The number $p = 15485863$ is prime, so by Fermat's little theorem

$$2^{15485862} \equiv 1 \pmod{15485863}.$$

Thus without doing any computing, we know that the number $2^{15485862} - 1$ is a multiple of 15485863 even though it might have more than two million digits. ∎

Remark 1.5.3. If $p \nmid a$, then by the Fermat's little theorem,

$$a^{p-1} \equiv 1 \pmod{p}.$$

Since a is nonzero modulo p, its inverse a^{-1} exists modulo p. Multiplying by a^{-1} on both the sides, we get

$$a^{-1} \equiv a^{p-2} \pmod{p}.$$

By the fast factoring algorithm seen earlier we can compute a^{p-2}. Thus we have a reasonably efficient method to compute inverses modulo p.

Example 1.5.4. We compute the inverse of 7814 modulo 17449 in two ways. First,

$$7814^{-1} \equiv 7814^{17447} \equiv 1284 \pmod{17449}.$$

Second, we use the extended Euclidean algorithm to solve

$$7814u + 17449v = 1.$$

The solution is $(u,v) = (1284, -575)$. So $7814^{-1} \equiv 1284 \pmod{17449}$. ∎

Example 1.5.5. Consider the number $m = 15485207$. Using the powering algorithm and **computer**, we get

$$2^{m-1} = 2^{15485206} \equiv 4136685 \pmod{15485207}.$$

If m were prime, then by Fermat's little theorem, we must have $2^{m-1} \equiv 1 \pmod{m}$. But instead of 1 we got 4136685. This tells us that $m = 15485207$ is not a prime (even without knowing its factors). ∎

By Fermat's little theorem, if a is an integer and $p \nmid a$, then $a^{p-1} \equiv 1 \pmod{p}$. However, for some a, there may be smaller powers (smaller than $p-1$) that are congruent to 1. We define this as the order of a modulo p.

Definition 1.5.6

Let a be an integer and p be a prime such that $p \nmid a$. The *order of a modulo p* is the smallest exponent (or smallest power) $k \geq 1$ such that

$$a^k \equiv 1 \pmod{p}.$$

Proposition 1.5.7

Let p be a prime and let a be an integer not divisible by p. Suppose that $a^n \equiv 1 \pmod{p}$. Then the order of a modulo p divides n. In particular, the order of a divides $p-1$.

Proof. Let k be the order of a modulo p. Then by the definition, k is the smallest exponent such that $a^k \equiv 1 \pmod{p}$. We are given $a^n \equiv 1 \pmod{p}$. Dividing n by k, we get

$$n = kq + r \quad \text{with } 0 \leq r < k.$$

Then

$$1 \equiv a^n \equiv a^{kq+r} \equiv \left(a^k\right)^q \cdot a^r \equiv 1^q \cdot a^r \equiv a^r \pmod{p}.$$

But $r < k$ and k is the smallest positive power such that $a^k \equiv 1 \pmod{p}$. Thus, $r = 0$ and so $n = kq$. Hence, $k \mid n$.
Finally, by the Fermat's little theorem, $a^{p-1} \equiv 1 \pmod{p}$. Then k divides $p-1$. □

Theorem 1.5.8: Primitive Root Theorem

Let p be a prime number. Then there exists an element $g \in \mathbb{F}_p^*$ whose powers give every element of $\mathbb{F}_p^*$, i.e.,

$$\mathbb{F}_p^* = \{1, g, g^2, g^3, \ldots, g^{p-2}\}.$$

Proof. Exercise. □

Definition 1.5.9

Let p be a prime. An element $g \in \mathbb{F}_p^*$ is called a *primitive root modulo* p or a *primitive root* of $\mathbb{F}_p$ or *generator of* $\mathbb{F}_p^*$ if

$$\{\underset{\substack{\| \\ g^{p-1}}}{\overset{\substack{g^0 \\ \| \\ 1}}{}}, g, g^2, g^3, \ldots, g^{p-2}\} = \mathbb{F}_p^*.$$

Primitive roots are the elements of $\mathbb{F}_p^*$ having order $p-1$.

Example 1.5.10. 2 is a primitive root in the field $\mathbb{F}_{11}$ as

$$\begin{array}{lllll} 2^0 = 1 & 2^1 = 2 & 2^2 = 4 & 2^3 = 8 & 2^4 = 5 \\ 2^5 = 10 & 2^6 = 9 & 2^7 = 7 & 2^8 = 3 & 2^9 = 6. \end{array}$$

Thus, all the 10 nonzero elements of $\mathbb{F}_{11}$ are generated by the power of 2. However, 2 is not a primitive root in $\mathbb{F}_{17}$ as

$$\begin{array}{lllll} 2^0 = 1 & 2^1 = 2 & 2^2 = 4 & 2^3 = 8 & 2^4 = 16 \\ 2^5 = 15 & 2^6 = 13 & 2^7 = 9 & 2^8 = 1. & \end{array}$$

Note that we got back to 1 before obtaining all the 16 nonzero values modulo 17. It turns out that 3 is a primitive root for 17 since

$$\begin{array}{llllll} 3^0 = 1 & 3^1 = 3 & 3^2 = 9 & 3^3 = 10 & 3^4 = 13 & 3^5 = 5 \\ 3^6 = 15 & 3^7 = 11 & 3^8 = 16 & 3^9 = 14 & 3^{10} = 8 & 3^{11} = 7 \\ 3^{12} = 4 & 3^{13} = 12 & 3^{14} = 2 & 3^{15} = 6. & & \end{array}$$

■

Remark 1.5.11. If p is large, then the finite field $\mathbb{F}_p$ has many primitive roots. The precise number of primitive roots in $\mathbb{F}_p$ is $\phi(p-1)$, where ϕ is the Euler's phi function. For example, one can check that

$$\{2,3,8,10,11,14,15,18,19,21,26,27\}$$

is the complete list of primitive roots in $\mathbb{F}_{29}$ and it agrees with the formula $\phi(28) = 12$.
More generally, if $k \mid p-1$, then there are exactly $\phi(k)$ elements of $\mathbb{F}_p^*$ having order k.

1.6 The Discrete Logarithm Problem

The discrete logarithm problem is a mathematical problem which is very important in cryptography. It arises in many settings, for example in modulo p setting which is described in this section and there is an elliptic curve version also.
Let p be a large prime. By the primitive root theorem (Theorem 1.5.8), there is a primitive root g in the field $\mathbb{F}_p$. Hence, every nonzero element of $\mathbb{F}_p$ is equal to some power of g. In particular, by Fermat's little theorem, $g^{p-1} = 1$ and no smaller positive power of g is equal to 1. Thus,

$$1, g, g^2, g^3, \ldots, g^{p-2} \in \mathbb{F}_p^*$$

is a complete list of elements in $\mathbb{F}_p^*$.

Definition 1.6.1: The Discrete Logarithm Problem (DLP)

Let g be a primitive root for $\mathbb{F}_p$ and let h be a nonzero element of $\mathbb{F}_p$. The Discrete Logarithm Problem (DLP) is the problem of finding an exponent x such that

$$g^x \equiv h \pmod{p}.$$

The number x is called the *discrete logarithm of h to the base g* and is denoted by $\log_g(h)$.

Remark 1.6.2. The discrete logarithm problem is a well-posed problem. It is the problem of finding a power x of g such that $g^x = h$. However, if there is one solution, then there are infinitely many solutions. By Fermat's little theorem, $g^{p-1} \equiv 1 \pmod{p}$. Hence, if x is a solution of $g^x = h$, then $x + k(p-1)$ is also a solution for every k, because

$$g^{x+k(p-1)} = g^x \cdot \left(g^{p-1}\right)^k \equiv h \cdot 1^k \equiv h \pmod{p}.$$

Thus, $\log_g(h)$ is defined only up to adding or subtracting multiples of $p-1$, i.e., modulo $p-1$. In other words, $\log_g(h)$ is defined modulo $p-1$ and therefore sometimes we refer to it as "the" discrete logarithm for the integer x between 0 and $p-2$ satisfying $g^x \equiv h \pmod{p}$.
It is not difficult to verify that $\log_g$ gives a well-defined function

$$\log_g : \mathbb{F}_p^* \to \frac{\mathbb{Z}}{(p-1)\mathbb{Z}}. \tag{1.9}$$

Remark 1.6.3. Note that in the above definition of the discrete logarithm problem, we used a primitive root g in $\mathbb{F}_p$. This is not strictly necessary. One can define the discrete logarithm problem for any $g \in \mathbb{F}_p^*$ and for any $h \in \mathbb{F}_p^*$.
More generally, instead of taking nonzero elements of a finite field $\mathbb{F}_p$ and raising them to powers or multiplying them, we can take elements of any group and use the group law instead of multiplication. This leads to a more general form of the discrete logarithm problem.

Definition 1.6.4: DLP for group

Let G be a group with binary operation (group law) $*$. The *Discrete Logarithm Problem for G* is to determine, for any two given elements g and h in G, an integer x satisfying

$$\underbrace{g * g * g * \cdots * g}_{x \text{ times}} = h.$$

CHAPTER 2

Discrete Logarithm Problem based Cryptosystems

2.1 Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange algorithm solves the following problem.
Alice and Bob want to share a secret key for use in a symmetric cipher, but their only means of communication is insecure. Every information they exchange is observed by their adversary, an eavesdropper, Eve. How is it possible for Alice to share a key with Bob without making it available to Eve?

At first look, this seems to be an impossible task but the solution to this problem is provided by Diffie and Hellman using the difficulty of the discrete logarithm problem for $\mathbb{F}_p^*$.

The first step is for Alice and Bob to agree on a large prime p and a nonzero integer g modulo p. Alice and Bob make the values of p and g public, for example, they might post these values on their websites. So it is accessible and known to everyone including Eve. For security and other reasons, it is best if the value of g is chosen such that its order in $\mathbb{F}_p^*$ is a large prime.

The next step for Alice is to pick a secret integer a that she does not reveal to anyone. At the same time Bob picks his secret integer b. Alice and Bob use their secret integers to compute

$$\underbrace{A \equiv g^a \pmod{p}}_{\text{Alice computes this}} \quad \text{and} \quad \underbrace{B \equiv g^b \pmod{p}}_{\text{Bob computes this}}.$$

Next, they exchange these computed values, i.e., Alice sends A to Bob and Bob sends B to Alice. Note that Eve gets to see the values of A and B, since they are sent over an insecure channel of communication.
Finally, Alice and Bob again use their secret integers to compute

$$\underbrace{A' \equiv B^a \pmod{p}}_{\text{Alice computes this}} \quad \text{and} \quad \underbrace{B' \equiv A^b \pmod{p}}_{\text{Bob computes this}}.$$

The values A' and B' that they compute are actually the same since

$$A' \equiv B^a \equiv \left(g^b\right)^a \equiv g^{ab} \equiv \left(g^a\right)^b \equiv A^b \equiv B' \pmod{p}.$$

This common value is their exchanged key. The Diffie-Hellman key exchange algorithm is summarized in Table 2.1

Public parameter creation	
A trusted party chooses and publishes a (large) prime p and an integer g having large prime order in $\mathbb{F}_p^*$	
Private computations	
Alice	**Bob**
Choose a secret integer a.	Choose a secret integer b.
Compute $A \equiv g^a \pmod{p}$.	Compute $B \equiv g^b \pmod{p}$.
Public exchange of values	
Alice sends A to Bob $\longrightarrow$	A
B $\longleftarrow$	Bob sends B to Alice
Further private computations	
Alice	**Bob**
Compute the number $B^a \pmod{p}$.	Compute the number $A^b \pmod{p}$.
The shared secret value is $B^a \equiv \left(g^b\right)^a \equiv g^{ab} \equiv \left(g^a\right)^b \equiv A^b \pmod{p}$.	

Table 2.1: Diffie-Hellman key exchange

Example 2.1.1. Alice and Bob agree to use the prime $p = 941$ and the primitive root $g = 627$. Alice chooses the secret key $a = 347$ and computes $A = 390 \equiv 627^{347} \pmod{941}$. Similarly, Bob chooses the secret key $b = 781$ and computes $B = 691 \equiv 627^{781} \pmod{941}$.
Alice sends Bob the number 390 and Bob sends Alice the number 691. Both of these transmissions are done over an insecure or open channel, so both $A = 390$ and $B = 691$ should be considered as public knowledge. The numbers $a = 347$ and $b = 781$ are not transmitted and remain secret. Then Alice and Bob are both able to compute the number

$$470 \equiv 627^{347 \cdot 781} \equiv A^b \equiv B^a \pmod{941},$$

and so 470 is their shared secret.
Suppose that Eve sees this entire exchange. She can obtain Alice's and Bob's shared secret if she can solve either of the congruences

$$627^a \equiv 390 \pmod{941} \quad \text{or} \quad 627^b \equiv 691 \pmod{941},$$

since then she will know one of their secret exponents. So far it is known that this is the only way for Eve to find the secret shared value without the assistance of Alice or Bob. ∎

In this example, the numbers are too small and they do not offer Alice and Bob any real security as Eve's computer will take a very little time to check all the possible powers of 627 modulo 941. Current guidelines suggest that Alice and Bob choose a prime p having approximately 1000 bits (that is $p \approx 2^{1000}$) and an element g whose order is prime and approximately $p/2$. Then Eve will face a truly difficult task.

In general Eve's dilemma is that she knows the values of A and B and so she knows the values of g^a and g^b. She also knows the values of g and p. So if she can solve DLP, then she can find a and b, after which it is easy for her to compute the shared secret key g^{ab} or Alice and Bob. Thus, it seems that Alice and Bob are safe provided that Eve is unable to solve DLP. However, this is not the case. The security of Alice's and Bob's shared secret key rests on the difficulty of the following problem which is potentially easier than DLP.

Definition 2.1.2: The Diffie-Hellman Problem (DHP)

Let p be a prime number and g be an integer. The *Diffie-Hellman Problem* (DHP) is the problem of computing the value of $g^{ab} \pmod{p}$ from the known values of $g^a \pmod{p}$ and $g^b \pmod{p}$.

If is clear that DHP is no harder than the DLP. If Eve can solve the DLP, then she can compute Alice and Bob's secret exponents a and b from the intercepted values $A = g^a$ and $B = g^b$, and then it is easy for her to compute their shared secret key g^{ab}. (In fact, Eve needs to compute only one of a and b.) However, the converse is not known. That is, suppose Eve has an algorithm that efficiently solves the DHP, then whether she can efficiently solve DLP or not is not known.

2.2 The Elgamal Public Key Cryptosystem

In the previous section, we saw the Diffie-Hellman key exchange algorithm which provides a method of sharing a secret key publicly. However, it does not achieve the full goal of being a public key cryptosystem, since a cryptosystem permits exchange of specific information, not just a random string of bits. The first natural development of a public key cryptosystem after Diffie-Hellman is a system developed by Taher Elgamal in 1985.

For the Elgamal PKC, Alice needs a large prime number p for which the discrete logarithm problem in $\mathbb{F}_p^*$ is hard. She also needs an element g modulo p. Alice can choose p and g herself or they may be preselected or supplied by some trusted party like an industry or government agency.
Alice chooses a secret number a as her private key. She then computes

$$A \equiv g^a \pmod{p}.$$

Alice publishes her public key A and keeps her private key a secret to herself.
Now suppose Bob wants to encrypt a message using Alice's public key A. We will assume that Bob's message m is an integer between 2 and p. To encrypt m, Bob randomly chooses another number k modulo p. Bob chooses this k to encrypt one, and only one, message, and then Bob discards it. The number k is called an *ephemeral key*, since it exists only for the purposes of encrypting a single message.
Bob takes his plaintext message m, ephemeral key k, and Alice's public key A and uses them to compute the two quantities

$$c_1 \equiv g^k \pmod{p} \quad \text{and} \quad c_2 \equiv mA^k \pmod{p}.$$

Note that g and p are public parameters and so Bob also knows their values. Bob's ciphertext, i.e., encryption of m, is the pair of numbers (c_1, c_2) which he sends to Alice.

Now, Alice decrypts the message. Since Alice knows a, she can compute the quantity

$$x \equiv c_1^a \pmod{p}.$$

Then she computes $x^{-1} \pmod{p}$ and multiplies it by c_2 to get the plaintext m as follows:

$$\begin{aligned}
x^{-1} \cdot c_2 &\equiv (c_1^a)^{-1} \cdot c_2 && \pmod{p}, && \text{since } x \equiv c_1^a \pmod{p},\\
&\equiv (g^{ak})^{-1} \cdot (mA^k) && \pmod{p}, && \text{since } c_1 \equiv g^k, c_2 \equiv mA^k \pmod{p},\\
&\equiv (g^{ak})^{-1} \cdot (m(g^a)^k) && \pmod{p}, && \text{since } A \equiv g^a \pmod{p},\\
&\equiv m && \pmod{p}, && \text{since the } g^{ak} \text{ terms cancel out.}
\end{aligned}$$

The Elgamal public key cryptosystem is summarized in Table 2.2.
Eve will try to decrypt the message in the following way.
Eve knows the public parameters p and g, and she also knows the value of $A \equiv g^a \pmod{p}$ because Alice's public key A is public knowledge. If Eve can solve the discrete logarithm problem, then she can find a and decrypt the message. Otherwise it seems to be difficult for Eve to find the plaintext.

Public parameter creation	
A trusted party chooses and publishes a large prime p and an element g modulo p of (large) prime order.	
Alice	**Bob**
Key creation	
Choose private key $1 \leq a \leq p-1$. Compute $A = g^a \pmod{p}$. Publish the public key A.	
Encryption	
	Choose plaintext m. Choose random element k. Use Alice's public key A to compute $c_1 = g^k \pmod{p}$ and $c_2 = mA^k \pmod{p}$. Send ciphertext (c_1, c_2) to Alice.
Decryption	
Compute $(c_1^a)^{-1} \cdot c_2 \pmod{p}$. This quantity is equal to m.	

Table 2.2: Elgamal key creation, encryption, and decryption

Example 2.2.1. Alice uses the prime $p = 467$ and the primitive root $g = 2$. She chooses $a = 153$ to be her private key and computes her public key

$$A \equiv g^a \equiv 2^{153} \equiv 224 \pmod{467}.$$

Bob decides to send the message $m = 331$ to Alice. He chooses a random ephemeral key $k = 197$, and he computes two numbers

$$c_1 \equiv g^k \equiv 2^{197} \equiv 87 \pmod{467} \quad \text{and} \quad c_2 \equiv mA^k \equiv 331 \cdot 224^{197} \equiv 57 \pmod{467}.$$

Bob sends the ciphertext pair $(c_1, c_2) = (87, 57)$ to Alice. Using her secret key $a = 153$, Alice computes

$$x \equiv c_1^a \equiv 87^{153} \equiv 367 \pmod{467}, \quad \text{and then } x^{-1} \equiv 14 \pmod{467}.$$

Finally, she computes

$$c_2 x^{-1} \equiv 57 \cdot 14 \equiv 331 \pmod{467}$$

and recovers the plaintext message m. ∎

Remark 2.2.2. In the Elgamal cryptosystem, the plaintext is an integer m between 2 and $p-1$, while the ciphertext consists of two integers c_1 and c_2 between 2 and $p-1$. Thus, in general, it takes twice as many bits to write down the ciphertext as it does to write down plaintext. Therefore, we say that Elgamal has a 2-*to*-1 *message expansion*.

The important question here is the following. Is Elgamal system as difficult as the Diffie-Hellman? In other words, is Elgamal system as hard for Eve to attack as the Diffie-Hellman? Or otherwise, unknowingly while giving a method of encrypting the messages, has the Elgamal system created a back door that makes it easy to decrypt messages without solving the Diffie-Hellman?
In modern cryptography, for any cryptosystem, one of the important goal is to identify the underlying hard problem (for example Diffie-Hellman) and to prove that the cryptosystem is at least as hard to attack as the underlying problem. In this case, we would like to prove that anyone who can decrypt arbitrary ciphertexts created by Elgamal encryption can also solve the Diffie-Hellman problem. More precisely, we have the following result.

Proposition 2.2.3

Fix a prime p and base g to use for the Elgamal encryption. Suppose that Eve has access to an oracle that decrypts arbitrary Elgamal ciphertexts using arbitrary Elgamal public keys. Then she can use the oracle to solve the Diffie-Hellman problem.

Proof. In the Diffie-Hellman problem, Eve is given the two values

$$A \equiv g^a \pmod{p} \quad \text{and} \quad B \equiv g^b \pmod{p},$$

and she is required to compute the value of $g^{ab} \pmod{p}$. Note that Eve knows the values of both A and B although she knows neither a nor b.
Now suppose Eve can consult an Elgamal oracle. This means that Eve can send the oracle a prime p, a base g, a purported (not necessarily true) public key A, and a purported ciphertext (c_1, c_2). From the Table 2.2, the oracle returns the quantity

$$(c_1^a)^{-1} \cdot c_2 \pmod{p}$$

to Eve. If Eve wants to solve the Diffie-Hellman problem, then she would choose $c_1 = B = g^b$ and $c_2 = 1$ and send this input to the oracle. The oracle would return $(g^{ab})^{-1} \pmod{p}$, and then Eve can take the inverse modulo p to get $g^{ab} \pmod{p}$, thereby solving the Diffie-Hellman problem.
Even if the oracle is smart and does not decrypt ciphertexts having $c_2 = 1$, Eve can still fool the oracle by sending it random-looking ciphertexts as follows.

She chooses an arbitrary c_2 and tells the oracle that the public key is A and that the cipher text is (B, c_2). The oracle returns to her the supposed plaintext m that satisfies

$$m \equiv (c_1^a)^{-1} \cdot c_2 \equiv (B^a)^{-1} \cdot c_2 \equiv (g^{ab})^{-1} \cdot c_2 \pmod{p}.$$

After the oracle returns the value of m to Eve, she finds the value of g^{ab} by simply computing

$$m^{-1} \cdot c_2 \equiv g^{ab} \pmod{p}.$$

Note that, with the help of oracle, although Eve has computed $g^{ab} \pmod{p}$, she has done it without knowing the value of a or b. Thus, she has solved only the Diffie-Hellman problem and not the discrete logarithm problem. □

Conversely, show that access to a Diffie-Hellman oracle breaks the Elgamal PKC (see Exercise 2.1).

Definition 2.2.4

An attack in which Eve has access to an oracle that decrypts arbitrary ciphertexts is known as a *chosen ciphertext attack*.

Exercise 2.1

Suppose that Eve is able to solve the Diffie-Hellman problem, i.e., if Eve is given two powers g^u and $g^v \mod p$, then she is able to compute $g^{uv} \mod p$. Show that Eve can break the Elgamal PKC.

The above proposition shows that the Elgamal system is secure against the chosen ciphertext attack. More precisely, it says that the Elgamal system is secure if one assumes that the Diffie-Hellman problem is hard.

2.3 How hard is the discrete logarithm problem?

Let G be a group and $g, h \in G$. The discrete logarithm problem is the problem to determine the exponent x such that $g^x = h$. What do we mean by the difficulty or the hardness of this problem? When do we say that this problem is "hard"? A natural measure of hardness of the problem is the approximate number of operations necessary for a person or a computer to solve the problem using the most efficient and currently known method.

Definition 2.3.1: Order Notation

Let $f(x)$ and $g(x)$ be functions of x taking values that are positive. We say that "f is big-$\mathcal{O}$ of g" and write

$$f(x) = \mathcal{O}(g(x))$$

if there are positive constants c and C such that

$$f(x) \leq cg(x) \qquad \text{for all } x \geq C.$$

In particular, we write $f(x) = \mathcal{O}(1)$ if $f(x)$ is bounded for all $x \geq C$.

The following proposition gives a method that can be used to prove that $f(x) = \mathcal{O}(g(x))$.

Proposition 2.3.2

If the limit

$$\lim_{x\to\infty} \frac{f(x)}{g(x)}$$

exits (and is finite), then $f(x) = \mathcal{O}(g(x))$.

Proof. Let L be the limit. By definition of limit, for any $\varepsilon > 0$ there is a constant C_ε such that

$$\left|\frac{f(x)}{g(x)} - L\right| < \varepsilon \qquad \text{for all } x > C_\varepsilon.$$

Taking $\varepsilon = 1$, we get

$$\frac{f(x)}{g(x)} < L+1 \qquad \text{for all } x > C_1.$$

Therefore, $f(x) < (L+1)g(x)$ for all $x > C_1$. Hence by definition, $f(x) = \mathcal{O}(g(x))$ with $c = L+1$ and $C = C_1$. □

Example 2.3.3. We have $2x^3 - 3x^2 + 7 = \mathcal{O}(x^3)$ since

$$\lim_{x\to\infty} \frac{2x^3 - 3x^2 + 7}{x^3} = 2.$$

Similarly, we have $x^2 = \mathcal{O}(2^x)$ since (by applying L'Hôpital's rule twice)

$$\lim_{x\to\infty} \frac{x^2}{2^x} = 0.$$

Note that we can have $f(x) = \mathcal{O}(g(x))$ even if the limit $\frac{f(x)}{g(x)}$ does not exist. For example, the limit

$$\lim_{x\to\infty} \frac{(x+2)\cos^2(x)}{x}$$

does not exist, but

$$(x+2)\cos^2(x) = \mathcal{O}(x)$$

since $(x+2)\cos^2(x) \le x+2 \le 2x$ for all $x \ge 2$. ∎

Exercise 2.2

Verify the following examples of the big-$\mathcal{O}$ notation.

(a) $x^2 + \sqrt{x} = \mathcal{O}(x^2)$.
(b) $5 + 6x^2 - 37x^5 = \mathcal{O}(x^5)$.
(c) $k^{300} = \mathcal{O}(2^k)$.
(d) $(\ln k)^{375} = \mathcal{O}(k^{0.001})$.
(e) $k^2 2^k = \mathcal{O}(e^{2k})$.
(f) $N^{10} 2^N = \mathcal{O}(e^N)$.

Suppose that we are trying to solve a certain type of mathematical problem, where the input to the problem is a number whose size may vary. For example, consider the *Integer Factorization Problem* whose input is a number N and whose output is a prime factor of N. We are interested in knowing how long it takes to solve the problem in terms of the size of the input. Usually, the size of the input is measured by the number of bits, since that is how much storage it will take to record the input.

Definition 2.3.4

Suppose that there is a constant $A \geq 0$, independent of the size of the input, such that if the input is $\mathcal{O}(k)$ bits long, then it takes $\mathcal{O}(k^A)$ steps to solve the problem. Then the problem is said to be solvable in *polynomial time*. If we take $A = 1$, then the problem is solvable in *linear time*, and if we take $A = 2$, then the problem is solvable in *quadratic time*. Polynomial times algorithms are considered to be fast algorithms.

If there is a constant $c > 0$ such that for inputs of size $\mathcal{O}(k)$ bits, there is an algorithm to solve the problem in $\mathcal{O}(e^{ck})$ steps, then the problem is solvable in *exponential time*. Exponential-time algorithms are considered to be slow algorithms.

Intermediate between polynomial-time algorithms and exponential-time algorithms are *subexponential-time algorithms*. They have the property that for every $\varepsilon > 0$, they are solvable in $\mathcal{O}_\varepsilon(e^{\varepsilon k})$ steps. This notation means that the constants c and C appearing in the definition of the order notation are allowed to depend on ε.

As a general rule in cryptography, problems that are solvable in polynomial time are considered to be "easy" while problems that are solvable in exponential-time are considered to be "hard", and problems with subexponential-time are considered lying in between. However, this is true only when the variables are very large. Depending on the big-$\mathcal{O}$ constants and on the size of input, an exponential problem may be easier than a polynomial problem. We understand these concepts by considering the discrete logarithm problem in various groups.

Example 2.3.5. Consider the discrete logarithm problem $g^x = h$ in $\mathbb{F}_p^*$. If the prime p is chosen between 2^k and 2^{k+1}, then g, h, and p all require at most k bits (binary digits), so the problem can be stated in $\mathcal{O}(k)$-bits. (Note that $\mathcal{O}(k)$ is same as $\mathcal{O}(\log_2 p)$.)
If we solve the DLP using trial-and-error method, then it takes $\mathcal{O}(p)$ steps to solve the problem. Since $\mathcal{O}(p) = \mathcal{O}(2^k)$, this algorithm takes exponential time. ∎

Example 2.3.6. Consider the discrete logarithm problem in the group $\mathbb{F}_p$, where the group operation is addition. The DLP in this context is finding a solution x to the congruence

$$x \cdot g \equiv h \pmod{p},$$

where g and h are given elements of $\mathbb{Z}/p\mathbb{Z}$. We can solve this congruence using the extended Euclidean algorithm to find $g^{-1} \pmod{p}$ and then $x \equiv g^{-1} \cdot h \pmod{p}$. This takes $\mathcal{O}(p)$ steps. So there is a linear-time algorithm to solve DLP in the additive group $\mathbb{F}_p$. This is a very fast algorithm. So the DLP in $\mathbb{F}_p$ with addition is not a good choice for one-way function in cryptography.
The discrete logarithm problems in different groups may have different levels of difficulty for finding their solution. In the last unit, we are going to see elliptic curves. If the elliptic

curve group has N elements, then the best known algorithm to solve the elliptic curve discrete logarithm problem (ECDLP) takes $\mathcal{O}(\sqrt{N})$ steps. Thus, it currently takes exponential time to solve ECDLP. ∎

2.4 A Collision Algorithm for the DLP

In this section, we see a discrete logarithm algorithm due to Shanks which is an example of a collision algorithm or meet-in-the-middle algorithm.

Proposition 2.4.1: Trivial Bound for DLP

Let G be a group and let $g \in G$ be an element of order N. Then the discrete logarithm problem

$$g^x = h$$

can be solved in $\mathcal{O}(N)$ steps and $\mathcal{O}(1)$ storage, where each step consists of multiplication by g.

Proof. We simply compute $g, g^2, g^3, \ldots$, where each successive value is obtained by multiplying the previous value by g. So we need to store only two values at a time. If a solution to $g^x = h$ exists, then h will appear as a power of g before we reach g^N.
Thus, it takes $\mathcal{O}(N)$ steps and $\mathcal{O}(1)$ storage to solve the DLP. □

The idea behind a collision algorithm is to make two lists and look for an element that appears in both the lists. For DLP, the *running time* (number of computer steps) of a collision algorithm is a little more than $\mathcal{O}(\sqrt{N})$ steps, which is a huge savings over $\mathcal{O}(N)$ if N is large.

Proposition 2.4.2: Shanks's Babystep-Giantstep Algorithm

Let G be a group and let $g \in G$ be an element of order $N \geq 2$. The following algorithm solves the discrete logarithm problem $g^x = h$ in $\mathcal{O}(\sqrt{N} \cdot \log N)$ steps using $\mathcal{O}(\sqrt{N})$ storage.

(1) Let $n = 1 + \lfloor \sqrt{N} \rfloor$, so in particular, $n > \sqrt{N}$.

(2) Create two lists,

List 1: $e, g, g^2, g^3, \ldots, g^n$,

List 2: $h, h \cdot g^{-n}, h \cdot g^{-2n}, h \cdot g^{-3n}, \ldots, h \cdot g^{-n^2}$.

(3) Find a match between the two lists, say $g^i = hg^{-jn}$.

(4) Then $x = i + jn$ is a solution to $g^x = h$.

Proof. We begin with some observations. First, when creating the List 2, we start by computing the quantity $u = g^{-n}$ and then compile List 2 by computing h, $h \cdot u$, $h \cdot u^2, \ldots, h \cdot u^n$. Thus, creating the two lists takes approximately $2n$ multiplications.
Second, assuming that a match exists, we can find a match in a small multiple of $n \log(n)$ steps using the standard sorting and searching algorithm. So Step (3) takes $\mathcal{O}(n \log n)$ steps. Hence

the total running time for the algorithm is $\mathcal{O}(n\log n) = \mathcal{O}(\sqrt{N}\log N)$. For the last step we have used the fact that $n \approx \sqrt{N}$ and so

$$n\log n \approx \sqrt{N}\log\sqrt{N} = \frac{1}{2}\sqrt{N}\log N.$$

Third, the lists in Step (2) have length n, so they require $\mathcal{O}(\sqrt{N})$ storage.
To prove that the algorithm works, we have to show that Lists 1 and 2 always have a match. To see this, let x be the unknown solution to $g^x = h$ and write x as

$$x = nq + r \quad \text{with } 0 \le r < n.$$

Since $1 \le x < N$

$$q = \frac{x-r}{n} < \frac{N}{n} < n \quad \text{since } n > \sqrt{N}.$$

Hence, we can rewrite the equation $g^x = h$ as

$$g^r = h \cdot g^{-qn} \qquad \text{with } 0 \le r < n \text{ and } 0 \le q < n.$$

Thus, g^r is in List 1 and $h \cdot g^{-qn}$ is in List 2, which shows that Lists 1 and 2 have a common element. □

Example 2.4.3. We illustrate Shanks's babystep-giantstep method by using it to solve the discrete logarithm problem

$$g^x = h \quad \text{in} \quad \mathbb{F}_p^* \quad \text{with} \quad g = 3, \quad h = 2, \quad p = 17.$$

We know that the number 3 has order $N = 16$ (since 3 is a primitive root for $\mathbb{F}_{17}$) and so $n = \lfloor\sqrt{N}\rfloor + 1 = \lfloor\sqrt{16}\rfloor + 1 = 5$. Then we compute $u = g^{-n} = 3^{-5}$. Note that $3^{16} \equiv 1 \pmod{17}$ and so $3^{11} = 3^{16-5} \equiv 3^{-5} \pmod{17}$. Therefore, $u = 3^{-5} \equiv 3^{11} \equiv 7 \pmod{17}$. Then the two lists are

List 1:

i	0	1	2	3	4	5
$g^i = 3^i$	1	3	9	27	**13**	5

List 2:

j	0	1	2
$h\cdot(g^{-n})^j = 2\cdot 7^j$	7	14	**13**

From the two lists we find the collision

$$3^4 \equiv 13 \equiv 2\cdot(3^{-5})^2 \equiv 2\cdot 3^{-10} \quad \text{in } \mathbb{F}_{17}.$$

This gives us $3^4 \cdot 3^{10} = 3^{14} \equiv 2 \pmod{17}$. ∎

2.5 The Chinese Remainder Theorem

The Chinese remainder theorem gives the solution of a system of simultaneous linear congruences. The simplest situation is a system of two linear congruences,

$$x \equiv a \pmod{m} \quad \text{and} \quad x \equiv b \pmod{n},$$

with $\gcd(m,n) = 1$. The Chinese remainder theorem says that this system has a unique solution modulo mn.
It was first recorded in a Chinese mathematical work in the late third or early fourth century describing a problem of three simultaneous congruences as follows:

> We have a number of things, but we do not know exactly how many. If we count them by threes, we have two left over. If we count them by fives, we have three left over. If we count them by sevens, we have two left over. How many things are there? [*Sun Tzu Suan Ching* (Master Sun's Mathematical Manual) circa 300 AD, volume 3, problem 26.]

The Chinese remainder theorem and its generalizations have many applications in number theory and other areas of mathematics. In the next section, we will see how it is used to solve some instances of the discrete logarithm problem. Before we state and prove the result, let us try to understand it by an example of two simultaneous linear congruences.

Example 2.5.1. Find an integer x that simultaneously solves both of the following linear congruences.

$$x \equiv 1 \pmod 5 \quad \text{and} \quad x \equiv 9 \pmod{11}.$$

The set of solutions of the first congruence $x \equiv 1 \pmod 5$ are integers of the form

$$x = 1 + 5y, \qquad y \in \mathbb{Z}. \tag{2.1}$$

Substituting this value in the second congruence $x \equiv 9 \pmod{11}$, we get

$$5y \equiv 8 \pmod{11}. \tag{2.2}$$

Multiplying both the sides of (2.2) by the inverse of 5 modulo 11. Note that the inverse exists since $\gcd(5,11) = 1$ (which can be computed using the extended Euclidean algorithm or the box method). Since $5 \cdot 9 = 45 \equiv 1 \pmod{11}$, the inverse of 5 modulo 11 is 9. So multiplying both sides of (2.2) by 9 we get

$$y \equiv 9 \cdot 8 \equiv 72 \equiv 6 \pmod{11}.$$

Substituting this value of y in equation (2.1), we get

$$x = 1 + 5 \cdot 6 = 31$$

which is the solution to the given simultaneous system of two linear congruences. ∎

The method described in the above example can be used to derive a general formula.

Theorem 2.5.2: Chinese Remainder Theorem

Let $m_1, m_2, \ldots, m_k$ be a collection of pairwise relatively prime integers. That is

$$\gcd(m_i, m_j) = 1 \quad \text{for all } i \neq j.$$

Let $a_1, a_2, \ldots, a_k$ be arbitrary integers. Then the system of simultaneous congruences

$$x \equiv a_1 \pmod{m_1}, \quad x \equiv a_2 \pmod{m_2}, \quad \ldots, \quad x \equiv a_k \pmod{m_k} \tag{2.3}$$

has a solution $x = c$. Further, if $x = c$ and $x = c'$ are both solutions, then

$$c \equiv c' \pmod{m_1 m_2 \cdots m_k}. \tag{2.4}$$

Proof. Consider the first i simultaneous congruences. For $i = 1$, $c_1 = a_1$ is the solution of the first congruence. Suppose that we have found a solution $x = c_i$ for the first i simultaneous congruences

$$x \equiv a_1 \pmod{m_1}, \quad x \equiv a_2 \pmod{m_2}, \quad \ldots, \quad x \equiv a_i \pmod{m_i}. \tag{2.5}$$

Then we find a solution to more more congruence, i.e., the simultaneous solution of the following system

$$x \equiv a_1 \pmod{m_1}, \quad x \equiv a_2 \pmod{m_2}, \quad \ldots, \quad x \equiv a_{i+1} \pmod{m_{i+1}}. \tag{2.6}$$

Let x be of the form

$$x = c_i + m_1 m_2 \cdots m_i y.$$

Then clearly x satisfies all the i congruences in (2.5). So we need to find y such that it also satisfies $x \equiv a_{i+1} \pmod{m_{i+1}}$. In other words, we need to find a value of y satisfying

$$c_i + m_1 m_2 \cdot m_i y \equiv a_{i+1} \pmod{m_{i+1}}.$$

Since $\gcd(m_{i+1}, m_1 m_2 \cdot m_i) = 1$, there exist integers u and v such that

$$m_{i+1} u + m_1 m_2 \cdots m_i v = 1.$$

Therefore, $m_1 m_2 \cdots m_i v \equiv 1 \pmod{m_{i+1}}$. Multiplying both sides by $a_{i+1} - c_i$, we get

$$m_1 m_2 \cdots m_i v(a_{i+1} - c_i) \equiv a_{i+1} - c_i \pmod{m_{i+1}}.$$

Therefore,

$$c_i + m_1 m_2 \cdots m_i (v(a_{i+1} - c_i)) \equiv a_{i+1} \pmod{m_{i+1}}.$$

Taking $y = v(a_{i+1} - c_i)$, we get the solution of the system of $i+1$ simultaneous linear congruences in (2.6). This completes the proof of the existence of solution.
Now we show that if $x = c$ and $x = c'$ are solutions to the system of simultaneous congruences in (2.3), then $c \equiv c' \pmod{m_1 m_2 \cdots m_k}$ (**left as a seminar exercise**). □

Exercise 2.3

(a) Let a, b, c be positive integers and suppose that

$$a \mid c, \quad b \mid c, \quad \text{and} \quad \gcd(a, b) = 1.$$

Prove that $ab \mid c$.

(b) Let $x = c$ and $x = c'$ be two solutions to the system of simultaneous congruences

$$x \equiv a_1 \pmod{m_1}, \quad x \equiv a_2 \pmod{m_2}, \quad \ldots, \quad x \equiv a_k \pmod{m_k}$$

(in the Chinese remainder theorem). Prove that

$$c \equiv c' \pmod{m_1 m_2 \cdots m_k}.$$

The proof of the Chinese remainder theorem can be converted into an algorithm for finding the solution to a system of simultaneous congruences. We illustrate by the following example.

Example 2.5.3. Solve the three simultaneous congruences

$$x \equiv 2 \pmod 3, \quad x \equiv 3 \pmod 7, \quad x \equiv 4 \pmod{16}. \tag{2.7}$$

■

Solution. The Chinese remainder theorem says that there is a unique solution modulo $336 = 3 \cdot 7 \cdot 16$. Clearly, $x = 2$ is the solution to the first congruence $x \equiv 2 \pmod 3$. Then its general solution $x = 2 + 3y$. Substituting it into the second congruence, we get

$$2 + 3y \equiv 3 \pmod 7.$$

This gives

$$3y \equiv 1 \pmod 7.$$

Since $3 \cdot 5 = 15 \equiv 1 \pmod 7$, 5 is the inverse of 3. So multiplying both sides by 5 in the above congruence, we get

$$y \equiv 5 \pmod 7.$$

This gives the value of x as

$$x = 2 + 3y = 2 + 3 \cdot 5 = 17$$

as a solution to the first two congruences in (2.7) which is modulo $21 = 3 \cdot 7$. Thus, the general form of the solution to the first two congruences is $x = 17 + 21z$. Substituting this value in the third congruence $x \equiv 4 \pmod{16}$, we get $17 + 21z \equiv 4 \pmod{16}$ which on simplifying gives

$$5z \equiv 3 \pmod{16}.$$

Multiplying both sides by 13, which is the inverse of 5 modulo 16, we get

$$z \equiv 3 \cdot 13 \equiv 39 \equiv 7 \pmod{16}.$$

Finally substituting this value of z in $x = 17 + 21z$, we get the solution

$$x = 17 + 21 \cdot 7 = 164.$$

All other solutions are obtained by adding and subtracting multiples of 336 to this particular solution. □

2.5.1 Solving Congruences with Composite Moduli

To solve a congruence with a composite modulus, we first solve several congruences modulo primes (or prime powers) and then fit the solutions together using the Chinese remainder theorem.

We discuss the problem of finding square roots modulo m. It is relatively easy to compute square roots modulo a prime and for primes congruent to 3 modulo 4, it is extremely easy. This is illustrated in the following proposition.

Proposition 2.5.4

Let p be a prime satisfying $p \equiv 3 \pmod 4$. Let a be an integer such that the congruence $x^2 \equiv a \pmod p$ has a solution, i.e., such that a has a square root modulo p. Then

$$b \equiv a^{(p+1)/4} \pmod p$$

is a solution, i.e., it satisfies $b^2 \equiv a \pmod p$.

Proof. Let g be a primitive root modulo p. Then a is equal to some power of g. Since a has a square root modulo p, it follows that a is equal to some even power of g, say $a = g^{2k} \pmod{p}$ (see Exercise 2.4). Now we compute

$$\begin{aligned}
b^2 &\equiv a^{\frac{p+1}{2}} && \pmod{p} && \text{definition of } b,\\
&\equiv \left(g^{2k}\right)^{\frac{p+1}{2}} && \pmod{p} && \text{since } a \equiv g^{2k} \pmod{p},\\
&\equiv g^{(p+1)k} && \pmod{p} &&\\
&\equiv g^{2k+(p-1)k} && \pmod{p} &&\\
&\equiv a \cdot (g^{p-1})^k && \pmod{p} && \text{since } a \equiv g^{2k} \pmod{p},\\
&\equiv a && \pmod{p} && \text{since } g^{p-1} \equiv 1 \pmod{p}.
\end{aligned}$$

Hence b is a square root of a modulo p. □

Note that the formula in the above proposition is valid only if a has a square root modulo p.

Exercise 2.4

Let p be an odd prime and let g be a primitive root modulo p. Prove that a has a square root modulo p if and only if its discrete logarithm $\log_g(a)$ modulo $p-1$ is even.

Example 2.5.5. A square root of $a = 2201$ modulo the prime $p = 4127$ is

$$b \equiv a^{(p+1)/4} = 2201^{4128/4} \equiv 2201^{1032} \equiv 3718 \pmod{4127}.$$

To check that a has a square root modulo 4127, we compute b^2 and check that

$$3718^2 = 13823524 \equiv 2201 \pmod{4127}.$$

▪

Suppose that we want to compute a square root modulo m, where m is not necessarily a prime. An efficient method is to factor m and compute square root modulo each of the prime (or prime powers) factors, and then combine the solutions using the Chinese remainder theorem. See the following example.

Example 2.5.6. We solve the congruence $x^2 \equiv 197 \pmod{437}$.
The modulus 437 is factored as $437 = 19 \cdot 23$. So we first solve the two congruences

$$y^2 \equiv 197 \equiv 7 \pmod{19} \quad \text{and} \quad z^2 \equiv 197 \equiv 13 \pmod{23}.$$

Since both 19 and 23 are congruent to 3 modulo 4, by above proposition (or by trial and error method), we can find the square roots. We have

$$y \equiv \pm 8 \pmod{19} \quad \text{and} \quad z \equiv \pm 6 \pmod{23}.$$

We can choose either 8 or -8 for y and either 6 or -6 for z. Choosing the two positive solutions and using the Chinese remainder theorem to solve the simultaneous congruences

$$x \equiv 8 \pmod{19} \quad \text{and} \quad x \equiv 6 \pmod{23} \tag{2.8}$$

we find that $x \equiv 236 \pmod{437}$ is the desired solution. ▪

Remark 2.5.7. Note that the solution of above example is not unique. We can always take negative solution

$$-236 \equiv 201 \pmod{437}$$

to get a second square root of 197 modulo 437.
If the modulus is prime, then there are only two square roots (see Exercise 2.5). However, if the modulus is composite, like $437 = 19 \cdot 23$, then there are two others. To find the other two solutions, we replace 8 and 6 by their negatives in (2.8) in the above example. This gives us $x = 144$ and $x = 293$. So 197 has four square roots modulo 437.

Exercise 2.5

(a) Let p be an odd prime and b be an integer with $p \nmid b$. Prove that either b has two square roots modulo p or else b has no square roots modulo p. In other words, prove that the congruence

$$X^2 \equiv b \pmod{p}$$

has either two solutions or no solutions in $\mathbb{Z}/p\mathbb{Z}$. (What happens if $p = 2$? What happens if $p \mid b$?)

(b) For each of the following values of p and b, find all the square roots of b modulo p.

(i) $(p,b) = (7,2)$

(ii) $(p,b) = (11,5)$

(iii) $(p,b) = (11,7)$

(iv) $(p,b) = (37,3)$

(c) How many square roots does 29 have modulo 35? Does this contradict the assertion in (a)? Justify.

(d) Let p be an odd prime and let g be a primitive root modulo p. Then any number a is equal to some power of g modulo p, say $a \equiv g^k \pmod{p}$. Prove that a has a square root modulo p if and only if k is even.

Remark 2.5.8. From the above example it is clear that it is relatively easy to compute square roots modulo m if one knows how to factor m in to product of prime powers. However, if m is so large that we are not able to factor it, then the problem to find square roots modulo m is very difficult.
In fact, if m is a large composite number whose factorization is unknown, then it is a difficult problem to determine whether a given number a has a square root modulo m, even without requiring to compute the square roots. The Goldwasser-Micali public key cryptosystem is based on the difficulty of identifying which numbers have square roots modulo m. The trapdoor information is knowledge of the factors of m.

2.6 The Pohlig-Hellman Algorithm

In the discrete logarithm problem, we need to solve the equation

$$g^x \equiv h \pmod{p}.$$

Recall that, as remarked earlier, if x is one solution, then $x+k(p-1)$ is also a solution for all k. So the solution x is determined only modulo $p-1$, i.e., we can think of the solution in $\mathbb{Z}/(p-1)\mathbb{Z}$. This indicates that the factorization of $p-1$ into primes may play an important role in the difficulty of the DLP in $\mathbb{F}_p^*$. In general, if G is a group and $g \in G$ is an element of order N, then solutions to $g^x = h$ in G are determined only modulo N. So the factorization of N would appear to be relevant. This idea is the core of the Pohlig-Hellman algorithm.

Theorem 2.6.1: Pohlig-Hellman Algorithm

Let G be a group, and suppose that we have an algorithm to solve the discrete logarithm problem in G for any element whose order is a power of a prime. To be concrete, if $g \in G$ has order q^e, suppose that we can solve $g^x = h$ in $\mathcal{O}(S_{q^e})$ steps.
Now let $g \in G$ be an element of order N, and suppose that N factors into a product of prime powers as

$$N = q_1^{e_1} \cdot q_2^{e_2} \cdots q_t^{e_t}.$$

Then the discrete logarithm problem $g^x = h$ can be solved in

$$\mathcal{O}\left(\sum_{i=1}^{t} S_{q_i^{e_i}} + \log N\right) \text{ steps} \tag{2.9}$$

using the following procedure:

(1) For each $1 \le i \le t$, let

$$g_i = g^{N/q_i^{e_i}} \qquad \text{and} \qquad h_i = h^{N/q_i^{e_i}}.$$

Notice that g_i has prime power order $q_i^{e_i}$, so use the given algorithm to solve the discrete logarithm problem

$$g_i^y = h_i. \tag{2.10}$$

Let $y = y_i$ be a solution to (2.10).

(2) Use the Chinese remainder theorem to solve

$$x \equiv y_1 \pmod{q_1^{e_1}}, \quad x \equiv y_2 \pmod{q_2^{e_2}}, \quad \ldots, \quad x \equiv y_t \pmod{q_t^{e_t}}. \tag{2.11}$$

Proof. The running time is clear, since for each i, $1 \le i \le t$ in Step (1) it takes $\mathcal{O}(S_{q_i^{e_i}})$ steps and Step (2), by the Chinese remainder theorem, takes $\mathcal{O}(\log N)$ steps. Hence, the total running time is

$$\mathcal{O}(\sum S_{q_i^{e_i}} + \log N).$$

Now we show that Step (1) and Step (2) gives a solution to $g^x = h$. Let x be a solution to the system of congruences (2.11). Then for each i we have

$$x = y_i + q_i^{e_i} z_i \quad \text{for some } z_i. \tag{2.12}$$

Then we can compute

$$\begin{aligned}
(g^x)^{N/q_i^{e_i}} &= (g^{y_i+q_i^{e_i}z_i})^{N/q_i^{e_i}} && \text{from (2.12)} \\
&= (g^{N/q_i^{e_i}})^{y_i} \cdot g^{Nz_i} && \\
&= (g^{N/q_i^{e_i}})^{y_i} && \text{since } g^N \text{ is the identity element,} \\
&= g_i^{y_i} && \text{by the definition of } g_i, \\
&= h_i && \text{from (2.10)} \\
&= h^{N/q_i^{e_i}} && \text{by the definition of } h_i.
\end{aligned}$$

In terms of the discrete logarithms to the base g, we can rewrite this as

$$\frac{N}{q_i^{e_i}} \cdot x \equiv \frac{N}{q_i^{e_i}} \cdot \log_g(h) \pmod{N}. \tag{2.13}$$

Now, observe that the numbers

$$\frac{N}{q_1^{e_1}}, \frac{N}{q_2^{e_2}}, \ldots, \frac{N}{q_t^{e_t}}$$

have no nontrivial common factor, i.e., their greatest common divisor is 1. By a repeated application of the extended Euclidean algorithm (see Exercise 2.6), we can find integers $c_1, c_2, \ldots, c_t$ such that

$$\frac{N}{q_1^{e_1}} \cdot c_1 + \frac{N}{q_2^{e_2}} \cdot c_2 + \cdots + \frac{N}{q_t^{e_t}} \cdot c_t = 1. \tag{2.14}$$

Multiplying both sides of equation (2.13) by c_i and taking the sum over $i = 1, 2, \ldots, t$, we get

$$\sum_{i=1}^{t} \frac{N}{q_i^{e_i}} \cdot c_i \cdot x \equiv \sum_{i=1}^{t} \frac{N}{q_i^{e_i}} \cdot c_i \cdot \log_g(h) \pmod{N}.$$

By (2.14), we have

$$x = \log_g(h) \pmod{N}.$$

This completes the proof that x satisfies $g^x = h$. □

Exercise 2.6

Let $a_1, a_2, \ldots, a_k$ be integers with $\gcd(a_1, a_2, \ldots, a_k) = 1$, i.e., the largest positive integer dividing all of $a_1, a_2, \ldots, a_k$ is 1. Prove that the equation

$$a_1u_1 + a_2u_2 + \cdots + a_ku_k = 1$$

has a solution in integers $u_1, u_2, \ldots, u_k$.
(Hint: Repeatedly apply the extended Euclidean algorithm.)

Remark 2.6.2. The Pohlig-Hellman algorithm has reduced the discrete logarithm problem for elements of arbitrary order to the discrete logarithm problem for elements of prime power order. A further refinement is given by the following proposition, which essentially reduce the problem to elements of prime order.

Proposition 2.6.3

Let G be a group. Suppose that q is a prime, and suppose that we know an algorithm that takes S_q steps to solve the discrete logarithm problem $g^x = h$ in G whenever g has order q. Now let $g \in G$ be an element of order q^e with $e \geq 1$. Then we can solve the discrete logarithm problem

$$g^x = h \quad \text{in } \mathcal{O}(eS_q) \text{ steps.} \tag{2.15}$$

Proof. We write the unknown exponent x in the following form.

$$x = x_0 + x_1 q + x_2 q^2 + \cdots + x_{e-1} q^{e-1} \quad \text{with } 0 \leq x_i < q. \tag{2.16}$$

Since g is of order q^e, the element $g^{q^{e-1}}$ is of order q. Raising both sides of (2.15) to the power q^{e-1}, we get

$$\begin{aligned} h^{q^{e-1}} &= (g^x)^{q^{e-1}} \\ &= \left(g^{x_0+x_1q+x_2q^2+\cdots+x_{e-1}q^{e-1}}\right)^{q^{e-1}} && \text{from (2.16)} \\ &= g^{x_0 q^{e-1}} \cdot \left(g^{q^e}\right)^{x_1+x_2q+\cdots+x_{e-1}q^{e-2}} \\ &= \left(g^{q^{e-1}}\right)^{x_0} && \text{since } g^{q^e} = 1. \end{aligned}$$

Since $g^{q^{e-1}}$ is an element of order q in G, the equation

$$\left(g^{q^{e-1}}\right)^{x_0} = h^{q^{e-1}}$$

is a discrete logarithm problem whose base is an element of order q. By our assumption, we can solve this problem in S_q steps. Once this is done, we know an exponent x_0 with the property that

$$g^{x_0 q^{e-1}} = h^{q^{e-1}} \text{ in } G.$$

Next, raising both sides of (2.15) to the power q^{e-2}, we get

$$\begin{aligned} h^{q^{e-2}} &= (g^x)^{q^{e-2}} \\ &= \left(g^{x_0+x_1q+x_2q^2+\cdots+x_{e-1}q^{e-1}}\right)^{q^{e-2}} && \text{from (2.16)} \\ &= g^{x_0 q^{e-2}} \cdot g^{x_1 q^{e-1}} \cdot \left(g^{q^e}\right)^{x_2+x_3q+\cdots+x_{e-1}q^{e-3}} \\ &= g^{x_0 q^{e-2}} \cdot g^{x_1 q^{e-1}}. \end{aligned}$$

Since we have already determined the value of x_0 and the element $g^{q^{e-1}}$ has order q in G, to find x_1, we must solve the discrete logarithm problem

$$\left(g^{q^{e-1}}\right)^{x_1} = \left(h \cdot g^{-x_0}\right)^{q^{e-2}}.$$

Applying the given algorithm again, we can solve this in S_q steps. Hence, in $\mathcal{O}(2S_q)$ steps we have determined the values for x_0 and x_1 satisfying

$$g^{(x_0+x_1q)q^{e-2}} = h^{q^{e-2}} \text{ in } G.$$

Similarly, we can find x_2 by solving the discrete logarithm problem

$$\left(g^{q^{e-1}}\right)^{x_2} = \left(h \cdot g^{-x_0 - x_1 q}\right)^{q^{e-3}}.$$

In general, after we have determined $x_0, \ldots, x_{i-1}$, then the value of x_o is obtained by solving

$$\left(g^{q^{e-1}}\right)^{x_i} = \left(h \cdot g^{-x_0 - x_1 q - \cdots - x_{i-1} q^{i-1}}\right)^{q^{e-i-1}} \quad \text{in } G.$$

Each of these is a discrete logarithm problem whose base is of order q. So each of them can be solved in S_q steps. Hence, after $\mathcal{O}(eS_q)$ steps, we obtain an exponent $x = x_0 + x_1 q + \cdots + x_{e-1} q^{e-1}$ satisfying $g^x = h$, thus solving the original discrete logarithm problem. □

CHAPTER 3

The RSA Algorithm

3.1 Euler's Formula and Roots Modulo pq

In the previous unit, we studies the Diffie-Hellman key exchange and the Elgamal public key cryptosystem. They were based on the fact that it is easy to compute $a^n \mod p$ but difficult to recover the exponent n provided that we know a and $a^n \mod p$, i.e., the discrete logarithm problem. Recall that, Fermat's little theorem, which was used to analyse the security of Diffie-Hellman and Elgamal, states that

$$a^{p-1} \equiv 1 \pmod{p} \quad \text{for all } a \not\equiv 0 \pmod{p},$$

where p is a prime. Natural question one would like to ask here is that can we replace prime p with any positive integer m? That is, is it true $a^{m-1} \equiv 1 \pmod{m}$? The answer to this is No. However, in this section, we shall see a generalization of Fermat's little theorem for $m = pq$, product of primes. We have the following result.

Theorem 3.1.1: Euler's Formula for pq

Let p and q be distinct primes and let

$$g = \gcd(p-1, q-1).$$

Then

$$a^{(p-1)(q-1)/g} \equiv 1 \pmod{pq} \quad \text{for all } a \text{ satisfying } \gcd(a, pq) = 1.$$

In particular, if p and q are odd primes, then

$$a^{(p-1)(q-1)/2} \equiv 1 \pmod{pq} \quad \text{for all } a \text{ satisfying } \gcd(a, pq) = 1.$$

Proof. Let a be an integer satisfying $\gcd(a, pq) = 1$. Then $p \nmid a$. Since $g = \gcd(p-1, q-1)$, we have $g \mid q-1$. Then

$$a^{(p-1)(q-1)/g} = \left(a^{(p-1)}\right)^{(q-1)/g} \qquad \text{since } (q-1)/g \text{ is an integer,}$$

$$\equiv 1^{(q-1)/g} \pmod{p} \qquad \text{from Fermat's little theorem,}$$
$$\equiv 1 \pmod{p}.$$

Thus, $a^{(p-1)(q-1)/g} - 1$ is divisible by p. Similarly, interchanging the roles of p and q, we get $a^{(q-1)(p-1)/g} - 1$ is divisible by q. Therefore, $a^{(p-1)(q-1)/g} - 1$ is divisible by, the lcm of p and q, that is pq. Hence,

$$a^{(p-1)(q-1)/g} \equiv 1 \pmod{pq} \quad \text{for all } a \text{ satisfying } \gcd(a, pq) = 1.$$

In particular, if p and q are odd primes, then $g = \gcd(p-1, q-1) = 2^k m$ for some integers k and m. Then we have

$$a^{(p-1)(q-1)/2^k m} \equiv 1 \pmod{pq} \quad \text{for all } a \text{ satisfying } \gcd(a, pq) = 1.$$

Raising to $2^{k-1}m$ on both sides we get

$$a^{(p-1)(q-1)/2} \equiv 1^{2^{k-1}m} \equiv 1 \pmod{pq} \quad \text{for all } a \text{ satisfying } \gcd(a, pq) = 1.$$

□

As discussed earlier, Diffie-Hellman key exchange and the Elgamal public key cryptosystem, for their security, depend on the difficulty of solving equations of the form

$$a^x \equiv b \pmod{p},$$

where a, b, and p are known quantities, p is prime, and x is the unknown exponent. The RSA public key cryptosystem, which we are going to discuss in the next section depend on the difficulty of solving equations of the form

$$x^e \equiv c \pmod{N},$$

where e, c, and N are known and x is the unknown. Thus, the security of RSA depends on the assumption that it is difficult to take eth roots modulo N.
Note that if the modulus N is a prime, then it turns out that it is relatively easy to compute eth roots modulo N. This is given by the next proposition.

Proposition 3.1.2

Let p be a prime and let $e \geq 1$ be an integer satisfying $\gcd(e, p-1) = 1$. We know that e has an inverse modulo $p-1$, say

$$de \equiv 1 \pmod{p-1}.$$

Then the congruence

$$x^e \equiv c \pmod{p} \tag{3.1}$$

has the unique solution $x \equiv c^d \pmod{p}$.

Proof. If $c \equiv 0 \pmod{p}$, then $x \equiv 0 \pmod{p}$ is the unique solutions and we are done.

So assume that $c \not\equiv 0 \pmod{p}$. The congruence $de \equiv 1 \pmod{p-1}$ implies $p-1 \mid de-1$. Therefore, there is an integer k such that

$$de - 1 = k(p-1) \quad \text{or} \quad de = 1 + k(p-1).$$

Now we check that c^d is a solution to $x^e \equiv c \pmod{p}$:

$$\begin{aligned}
(c^d)^e &\equiv c^{de} \pmod{p} && \text{law of exponents,}\\
&\equiv c^{1+k(p-1)} \pmod{p} && \text{since } de = 1+k(p-1),\\
&\equiv c \cdot (c^{p-1})^k \pmod{p} && \text{law of exponents,}\\
&\equiv c \cdot 1^k \pmod{p} && \text{by Fermat's little theorem,}\\
&\equiv c \pmod{p}.
\end{aligned}$$

This completes the proof that the congruence $x^e \equiv c \pmod{p}$ has a solution $x = c^d$. Now, we prove the uniqueness of the solution. Suppose x_1 and x_2 are both solutions of the congruence (3.1). We just prove that $z^{de} \equiv z \pmod{p}$ for any nonzero z. So we have

$$x_1 \equiv x_1^{de} \equiv (x_1^e)^d \equiv c^d \equiv (x_2^e)^d \equiv x_2^{de} \equiv x_2 \pmod{p}.$$

Thus, x_1 and x_2 are congruent modulo p and hence the congruence (3.1) has a unique solution. □

Example 3.1.3. We solve the congruence

$$x^{1583} \equiv 4714 \pmod{7919},$$

where the modulus $p = 7919$ is prime.
Here $e = 1583$ and $c = 4714$. By above the proposition, we first need to solve the congruence

$$de \equiv 1 \pmod{p-1} \qquad \text{i.e., } 1583d \equiv 1 \pmod{7918}.$$

Using the extended Euclidean algorithm, we find that the solution is $d = 5277$. Then by the above proposition $x = c^d$ is the solution, i.e.,

$$x \equiv 4714^{5277} \equiv 6059 \pmod{7919}.$$

■

Remark 3.1.4. In Proposition 3.1.2, we assume that $\gcd(e, p-1) = 1$. If this assumption is removed, then the congruence $x^e \equiv c \pmod{p}$ will have a solution for some, but not for all, values of c. Also, if it has a solution then it will have more than one solutions (see Exercise 3.1).

The above proposition also shows that it is easy to take eth roots if the modulus is a prime p. If the modulus is a composite number N and if we know how to factor N, then again it is easy to compute eth roots. The following proposition shows how to do this if $N = pq$ is a product of primes. The general case is left as an exercise (see Exercise 3.2).

Exercise 3.1

This exercise investigates what happens if we drop the assumtpion that $\gcd(g, p-1) = 1$ in Proposition 3.1.2. So let p be a prime, let $c \not\equiv 0 \pmod{p}$, let $e \geq 1$, and consider the congruence

$$x^e \equiv c \pmod{p}.$$

(a) Prove that if the above congruence has one solution, then it has exactly $\gcd(e, p-1)$ distinct solutions.
(Hint: Use primitive root theorem combined with the extended Euclidean algorithm.)

(b) For how many non-zero values of $c \pmod{p}$ does the above congruence have a solution?

Exercise 3.2

Let N, c, and e be positive integers satisfying the conditions $\gcd(N,c) = 1$ and $\gcd(e, \phi(N)) = 1$.

(a) Explain how to solve the congruence

$$x^e \equiv c \pmod{N},$$

assuming that we know the value of $\phi(N)$.
(Hint: Use the formula $a^{\phi(N)} \equiv 1 \pmod{N}$ for all integers a satisfying $\gcd(a,N) = 1$. This result is called Euler's theorem or Euler's formula.)

(b) Solve the following congruences. (Use the formula $\phi(N) = N \prod_{i=1}^{r} \left(1 - \frac{1}{p_i}\right)$ for computing the value of $\phi(N)$.)

(i) $x^{577} \equiv 60 \pmod{1463}$.

(ii) $x^{959} \equiv 1583 \pmod{1625}$.

(iii) $x^{133957} \equiv 224689 \pmod{2134440}$.

Proposition 3.1.5

Let p and q be distinct primes and let $e \geq 1$ satisfy

$$\gcd(e, (p-1)(q-1)) = 1.$$

We know that e has an inverse modulo $(p-1)(q-1)$, say

$$de \equiv 1 \pmod{(p-1)(q-1)}.$$

Then the congruence

$$x^e \equiv c \pmod{pq} \tag{3.2}$$

has the unique solution $x \equiv c^d \pmod{pq}$.

Proof. Assume that $\gcd(c, pq) = 1$. (See Exercise 3.3 for other cases). The congruence $de \equiv 1 \pmod{(p-1)(q-1)}$ means that there is an integer k such that

$$de = 1 + k(p-1)(q-1).$$

Now we check that c^d is a solution to $x^e \equiv c \pmod{pq}$:

$$(c^d)^e \equiv c^{de} \pmod{pq} \qquad \text{law of exponents,}$$

$$\begin{aligned}
&\equiv c^{1+k(p-1)(q-1)} \pmod{pq} && \text{since } de = 1+k(p-1)(q-1),\\
&\equiv c\cdot (c^{(p-1)(q-1)})^k \pmod{pq} && \text{law of exponents,}\\
&\equiv c\cdot 1^k \pmod{pq} && \text{by Euler's formula,}\\
&\equiv c \pmod{pq}.
\end{aligned}$$

This completes the proof that the congruence $x^e \equiv c \pmod{pq}$ has a solution $x = c^d$. Now, we prove the uniqueness of the solution. Suppose $x = u$ is a solution to (3.2). Then

$$\begin{aligned}
u &\equiv u^{de-k(p-1)(q-1)} \pmod{pq} && \text{since } de = 1+k(p-1)(q-1),\\
&\equiv (u^e)^d \cdot (u^{(p-1)(q-1)})^{-k} \pmod{pq}\\
&\equiv (u^e)^d \cdot 1^{-k} \pmod{pq} && \text{using Euler's formula,}\\
&\equiv c^d \pmod{pq} && \text{since } u \text{ is a solution to (3.2).}
\end{aligned}$$

Thus, every solution to (3.2) is equal to $c^d \pmod{pq}$ and hence the solution is unique. □

Exercise 3.3

Let p and q be distinct primes and let e and d be positive integers satisfying

$$de \equiv 1 \pmod{(p-1)(q-1)}.$$

Suppose further that c is an integer with $\gcd(c, pq) > 1$. Prove that $x \equiv c^d \pmod{pq}$ is a solution to the congruence $x^e \equiv \pmod{pq}$, thereby completing the proof of Proposition 3.1.5.

Remark 3.1.6. Proposition 3.1.5 gives an algorithm for solving $x^e \equiv c \pmod{pq}$ that involves first solving $de \equiv 1 \pmod{(p-1)(q-1)}$ and then computing $c^d \pmod{pq}$. We can make the computations faster by using a smaller value of g. Let $g = \gcd(p-1, q-1)$ ans suppose that we solve the congruence

$$de \equiv 1 \pmod{\frac{(p-1)(q-1)}{g}}$$

for d. Then Euler's formula (Theorem 3.1.1) says that $a^{(p-1)(q-1)/g} \equiv 1 \pmod{pq}$. Hence, as in the proof of above proposition, writing $de = 1+k(p-1)(q-1)/g$, we get

$$(c^d)^e = c^{dc} = c^{1+k(p-1)(q-1)/g} = c\cdot (c^{(p-1)(q-1)/g})^k \equiv c \pmod{pq}.$$

Thus, using a smaller value of d, we can still find that $c^d \mod pq$ is a solution to $x^e \equiv c \pmod{pq}$.

3.2 The RSA Public Key Cryptosystem

Alice and Bob have the usual problem of exchanging secret information over an insecure channel of communication. In the previous unit, we have seen that they carried out this task using systems based on the difficulty of solving the discrete logarithm problem.
In this section, we describe the RSA public key cryptosystem which is the first invented and best known such system. RSA is named after its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman.
The security of RSA depends on the following dichotomy:

Setup. Let p and q be large primes, let $N = pq$, and let e and c be integers.

Problem. Solve the congruence $x^e \equiv c \pmod{N}$ for the variable x.

Easy. Bob, who knows the values of p and q, can easily solve for x as described in Proposition 3.1.5.

Hard. Eve, who does not know the values of p and q, cannot easily find x.

Dichotomy. Solving $x^e \equiv c \pmod{N}$ is easy for a person who possesses certain extra information, but it is apparently hard for all other people.

The RSA public key cryptosystem is summarized in Table 3.1.

Bob	**Alice**
Key creation	
Choose secret prime p and q. Choose encryption exponent e with $\gcd(e,(p-1)(q-1)) = 1$ Publish $N = pq$ and e.	
Encryption	
	Choose plaintext m. Use Bob's public key (N,e) to compute $c = m^e \pmod{N}$. Send ciphertext c to Bob.
Decryption	
Compute d satisfying $ed \equiv 1 \pmod{(p-1)(q-1)}$. Compute $m' \equiv c^d \pmod{N}$. Then m' equals the plaintext m.	

Table 3.1: RSA key creation, encryption, and decryption

Bob's secret key is a pair of large primes p and q. His public key is the pair (N,e), where N is the product $N = pq$ and e is the encryption exponent that is relatively prime to $(p-1)(q-1)$. Alice takes her plaintext and converts it into an integer m between 1 and N. She encrypts m by computing

$$c \equiv m^e \pmod{N}.$$

The integer c is her ciphertext which she sends to Bob. It is then easy for Bob to solve the congruence $x^e \equiv c \pmod{N}$ to get Alice's message m because Bob knows the factorization $N = pq$. Eve, on the other hand, may intercept the ciphertext c, but unless she knows how to factor N, it is difficult for her to solve $x^e \equiv c \pmod{N}$.

Example 3.2.1. Following is an illustration of the RSA public key cryptosystem with a small numerical example. This is not secure as the numbers are small and Eve would easily factor the modulus N. Practical secure implementations of RSA uses moduli N with hundreds of digits.
RSA Key Creation

- Bob chooses two secret primes $p = 1223$ and $q = 1987$. Bob computes his public modulus

$$N = p \cdot q = 1223 \cdot 1987 = 2430101.$$

- Bob chooses a public encryption exponent $e = 948047$ with the property that

$$\gcd(e, (p-1)(q-1)) = \gcd(948047, 2426892) = 1.$$

RSA Encryption

- Alice converts her plaintext into an integer

$$m = 1070777 \quad \text{satisfying } 1 \leq m < N.$$

- Alice uses Bob's public key $(N, e) = (2430101, 948047)$ to compute

$$c \equiv m^e \pmod{N}, \quad c \equiv 1070777^{948047} \equiv 1473513 \pmod{2430101}.$$

- Alice sends the ciphertext $c = 1473513$ to Bob.

RSA Decryption

- Bob knows $(p-1)(q-1) = 1222 \cdot 1986 = 2426892$, and so he can solve

$$ed \equiv \pmod{(p-1)(q-1)}, \qquad 948047 \cdot d \equiv 1 \pmod{2426892},$$

  for d and find that $d = 1051235$.

- Bob takes the ciphertext $c = 1473513$ and computes

$$c^d \pmod{N}, \qquad 1473513^{1051235} \equiv 1070777 \pmod{2430101}.$$

  The value that he computes is Alice's message $m = 1070777$.

■

Remark 3.2.2. The quantities N and e that form Bob's public key are called the *modulus* and the *encryption exponent*. The number d that Bob uses to decrypt Alice's message, that is, the number d satisfying

$$ed \equiv 1 \pmod{(p-1)(q-1)} \tag{3.3}$$

is called the *decryption exponent*.
It is clear that encryption can be done efficiently and easily if the encryption exponent e is a small number and decryption can be done efficiently and easily if the decryption exponent is a small number. Bob cannot choose both of them to be small, since once one of them is selected, the other is determined by the congruence (3.3).
If $e = 1$, then $d = 1$. In this case, the plaintext and the ciphertext will be same. So Bob cannot take e to be 1. He cannot take e to be 2 because of the condition that $\gcd(e, (p-1)(q-1)) = 1$. So the smallest possible value for e is $e = 3$.
Alternately, Bob can choose d to be small and use the congruence (3.3) to determine e which would be large. However, if d is smaller than $N^{1/4}$, then the theory of continued fractions will enable Eve to break RSA.

Remark 3.2.3. Bob's public key includes the number $N = pq$, which is a product of two secret primes p and q. Proposition 3.1.5 says that if Eve knows the value of $(p-1)(q-1)$, then she can solve $x^e \equiv c \pmod{N}$, and thus decrypt the message sent to Bob. Expanding $(p-1)(q-1)$ gives

$$(p-1)(q-1) = pq - p - q + 1 = N - (p+q) + 1. \tag{3.4}$$

Since the value of N is public, Eve knows N. Thus if Eve can find the sum $p+q$, then equation (3.4) gives her the value of $(p-1)(q-1)$. Then she can decrypt the messages.
If Eve knowns $p+q$ and pq, then it is easy for her to compute the values of p and q. She simply uses the quadratic formula to find the roots of the polynomial

$$X^2 - (p+q)X + pq,$$

since its factors are $(X-p)(X-q)$. Thus, once Bob publishes the value of $N = pq$, it is no easier for Eve to find $(p-1)(q-1)$ than to find p and q themselves.

Remark 3.2.4. We have shown that it is no easier for Eve to determine $(p-1)(q-1)$ than it is for her to factor N, i.e., to determine p and q themselves. However, this does not prove that Eve must factor N to in order to decrypt Bob's message. This is just one way to solve the problem. The point is that what Eve really needs to do is to solve the congruence

$$x^e \equiv c \pmod{N}$$

if there is a method or an efficient algorithm to solve such congruences without the knowledge of $(p-1)(q-1)$. No one knows whether such a method exists or not.

3.3 Implementation and Security Issues

In this section, we discuss some of the security issues related to the implementation of the cryptosystem.

Example 3.3.1 (Woman-in-the-Middle Attack)**.** Suppose Eve is not just simply an eavesdropper but she has full control over Alice and Bob's communication network. In this case, she can institute what is known as a *man-in-the-middle* attack. This attack is described below for the Diffie-Hellman key exchange but it exists for most of the public key constructions (see Exercise 3.4).
Recall that in the Diffie-Hellman key exchange, Alice sends $A = g^a$ to Bob and Bob sends $B = g^b$ to Alice, where a and b are secret key of Alice and Bob respectively and the computations takes place in the finite field $\mathbb{F}_p$. Now, Eve chooses her secret exponent e and computes $E = g^e$. She then intercepts Alice and Bob's communications, and instead of sending A to Bob and B to Alice, she sends both of them the number E. Notice that Eve has exchanged the value A^e with Alice and the value B^e with Bob, while Alice and Bob believe that they have exchanged values with each other. The man-in-the-middle attack is illustrated in Figure 3.1.

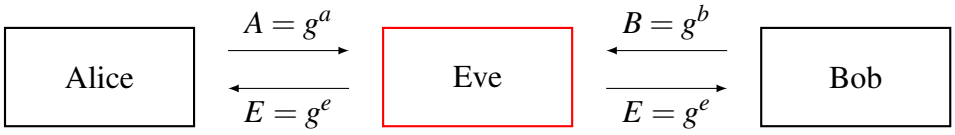


Figure 3.1: "Man-in-the-middle" attack on Diffie-Hellman key exchange

Suppose that Alice and Bob use their supposed shared secret value as the key for a symmetric cipher and send each other messages. For example, Alice encrypts a message m using E^a as the symmetric cipher key. Eve intercepts this message and is able to decrypt it using A^e as the symmetric cipher key, so she can read Alice's message. She then encrypts it using B^e as the symmetric cipher key for sending it to Bob. Since Bob is then able to decrypt it using E^b as the symmetric cipher key, he is unaware that there is a breach in security.
Note that in this attack even though Eve does not solve the underlying hard problem, i.e., Eve does not solve the discrete logarithm problem or the Diffie-Hellman problem, she is able to read Alice and Bob's communications and they are not aware of her success. ∎

Exercise 3.4

Formulate a man-in-the-middle attack, similar to the attack described in Example 3.3.1 for the following public key cryptosystems:

(a) The Elgamal public key cryptosystem.

(b) The RSA public key cryptosystem.

Example 3.3.2. Suppose that Eve is able to convince Alice to decrypt "random" RSA messages using her (Alice's) private key. This is a plausible scenario, since one way for Alice to authenticate her identity as the owner of the public key (N,e) is to show that she knows how to decrypt messages. One says that Eve has access to an *RSA oracle*.
Eve can exploit Alice's generosity as follows. Suppose that Eve has intercepted a ciphertext c that Bob has sent to Alice. Eve chooses a random value k and sends Alice the "message"

$$c' \equiv k^e \cdot c \pmod{N}.$$

Alice decrypts c' and returns the resulting message m' to Eve, where

$$m' \equiv (c')^d \equiv (k^e \cdot c)^d \equiv (k^e \cdot m^e)^d \equiv k \cdot m \pmod{N}.$$

Thus, Eve knows the quantity $k \cdot m \pmod{N}$ and since she knows k, she can immediately recover Bob's plaintext m.
There are following two important observations.

1. Eve has decrypted Bob's message without knowing or gaining knowledge of how to factor N. Thus, the difficulty of the underlying mathematical problem is irrelevant.

2. Since Eve has used k to mask Bob's ciphertext, Alice has no way to tell that Eve's message is in any way related to Bob's message. Thus, Alice sees only the values $k^e \cdot c \pmod{N}$ and $k \cdot m \pmod{N}$, which looks random to her as compared to c and m.

∎

Example 3.3.3. Suppose that Alice publishes two different exponents e_1 and e_2 for use with her public modulus N and that Bob encrypts a single message m using both of Alice's exponents. If Eve intercepts the ciphertexts

$$c_1 \equiv m^{e_1} \pmod{N} \quad \text{and} \quad c_2 \equiv m^{e_2} \pmod{N},$$

then she can take a solution to the equation

$$e_1 \cdot u + e_2 \cdot v = \gcd(e_1, e_2)$$

and use it to compute

$$c_1^u \cdot c_2^v \equiv (m^{e_1})^u \cdot (m^{e_2})^v \equiv m^{e_1 \cdot u + e_2 \cdot v} \equiv m^{\gcd(e_1, e_2)} \pmod{N}.$$

If it happens that $\gcd(e_1, e_2) = 1$, Eve has recovered the plaintext m (see Exercise 3.5 for a particular numerical example). More generally, if Bob encrypts a single message using several exponents $e_1, e_2, \ldots, e_r$, then Eve can recover the plaintext if $\gcd(e_1, e_2, \ldots, e_r) = 1$.
Hence, Alice should use only one encryption exponent e for a given modulus N. ∎

Exercise 3.5

Alice decides to use RSA with the public key $N = 1889570071$. In order to guard against transmission errors, Alice has Bob encrypt his message twice, once using the encryption exponent $e_1 = 1021763679$ and once using the encryption exponent $e_2 = 519424709$. Eve intercepts the two encrypted messages

$$c_1 = 1244183534 \text{ and } c_2 = 732959706.$$

Assuming that Eve also knows N and the two encryption exponents e_1 and e_2, use the method described in Example 3.3.3 to help Eve recover Bob's plaintext without finding a factorization of N.

3.4 Primality Testing

3.4.1 Bob's quest for large prime

As we have seen, in order for Bob to implement RSA, he needs to choose two large prime numbers p and q. He cannot choose composite numbers because he needs to factor them to decrypt Alice's message and also if the composite numbers have small prime factors, then it would be easy for Eve to find the factors of pq and break Bob's system.
Thus in order to find large primes numbers, Bob needs a way to distinguish between prime and composite numbers because if he knows such a way, then he can randomly choose large numbers until he hits a prime. For example, Bob chooses the following large number

$$n = 319879377374793553326200686437131014909523353 01$$

and he wants to know whether it is prime or not. First Bob tries to find the small factors and he observes that n is not divisible by primes smaller than 1000000. So he thinks that n maybe a prime. Next, he computes $2^{n-1} \mod n$ and finds that

$$2^{n-1} \equiv 128126595355135906413360121624715183605316 0074 \pmod{n}.$$

The above congruence tells Bob that the number n is not prime because by Fermat's little theorem, if p is a prime and $p \nmid a$, then $a^{p-1} \equiv 1 \pmod{p}$. Since the right hand side of the above congruence is not 1, n is not a prime number. In this context, we have another version of the Fermat's little theorem given below.

Theorem 3.4.1: Fermat's Little Theorem, Version 2

Let p be a prime number. Then

$$a^p \equiv a \pmod{p} \quad \text{for every integer } a. \tag{3.5}$$

Proof. If $p \nmid a$, then the first version of Fermat's little theorem implies that $a^{p-1} \equiv 1 \pmod{p}$. Multiplying both sides by a proves (3.5). On the other hand, if $p \mid a$, then both the sides of (3.5) are 0 modulo p. □

Now again suppose Bob randomly chooses another large number

$$n = 296795298595169276282041874013832900431516513 1.$$

After checking for smaller prime factors, Bob computes $2^n \mod n$ to find that

$$2^n \equiv 2 \pmod{n}.$$

Does this mean, by Fermat's little theorem version 2 (Theorem 3.4.1), that n is prime? The answer is **No!** Fermat's little theorem only works one way and the converse is not true. That is if p is a prime then

$$a^p \equiv a \pmod{p}.$$

However, note that $341 = 11 \cdot 31$ but still the congruence (3.5) holds as we have

$$2^{341} \equiv 2 \pmod{341}.$$

The fact that the congruence $2^n \equiv 2 \pmod{n}$ holds makes it more likely that n could be a prime number for if n was composite, then there are high changes for the value of $2^n \mod n$ to turn out different from 2. That is if $2^n \not\equiv 2 \pmod{n}$, then n is definitely composite. This leads us to the following definition.

Definition 3.4.2

Fix an integer n. We say that an integer a is a *witness for (the compositeness of) n* if

$$a^n \not\equiv a \pmod{n}.$$

As we observed before, by Fermat's little theorem (version 2), a single witness for n is enough to prove that n is composite. Thus, to check whether n is prime of not Bob tries a lot of number $a_1, a_2, a_3, \ldots$. If any one of them is a witness for n, then Bob knows that n is composite, and if none of them is a witness for n, then Bob suspects, still not certain, that n is prime.
Observe that the number 561 is composite as $561 = 3 \cdot 11 \cdot 17$, yet it has no witnesses. That is

$$a^{561} \equiv a \pmod{561} \quad \text{for every integer } a.$$

One can check that for all $a = 0, 1, 2, \ldots, 560$, one has $a^{561} \equiv a \pmod{561}$.
Composite numbers having no witnesses are called *Carmichael numbers*. They are named after R. D. Carmichael who listed 15 such numbers in his paper published in 1910. Although

Carmichael numbers are rare, in 1994, Alford, Granville, and Pomerance proved that there are infinitely many Carmichael numbers.
Evidently, Bob needs a better test for compositness of a number, something stronger than the Fermat's little theorem. The following property of prime numbers is used to formulate the *Miller-Rabin test* which says that every composite number has a large number of (Miller-Rabin) witnesses.

Proposition 3.4.3

Let p be an odd prime and write

$$p-1 = 2^k q \quad \text{with } q \text{ odd.}$$

Let a be any number not divisible by p. Then one of the following two conditions is true:

(i) a^q is congruent to 1 modulo p.

(ii) One of $a^q, a^{2q}, a^{4q}, \ldots, a^{2^{k-1}q}$ is congruent to -1 modulo p.

Proof. By Fermat's little theorem, $a^{p-1} \equiv 1 \pmod{p}$. Thus, we know that the last number in the following list of numbers is $a^{p-1} = a^{2^k q} \equiv \mod p$.

$$a^q, a^{2q}, a^{4q}, \ldots, a^{2^{k-1}q}, a^{2^k q}.$$

Further, each number in the list is the square of the previous one. Therefore, one of the two possibilities must occur:

(i) The first number in the list is congruent to 1 modulo p, i.e., $a^q \equiv 1 \mod p$.

(ii) Some number in the list is not congruent to 1 modulo p, but when it is squared, it becomes congruent to 1 modulo p. But the only number satisfying both

$$b \not\equiv 1 \pmod{p} \quad \text{and} \quad b^2 \equiv 1 \pmod{p}$$

is -1. So one of the numbers in the list is congruent to -1 modulo p.

This completes the proof of the proposition. □

Definition 3.4.4

Let n be an odd number and write $n-1 = 2^k q$ with q odd. An integer a satisfying $\gcd(a,n) = 1$ is called a *Miller-Rabin witness for (the compositeness of) n* if both of the following conditions are true:

(a) $a^q \not\equiv 1 \pmod{n}$.

(b) $a^{2^i q} \not\equiv -1 \pmod{n}$ for all $i = 1, 2, \ldots, k-1$.

It follows from Proposition 3.4.3 that if a is a Miller-Rabin witness for n, then n is definitely a composite number. This leads to the Miller-Rabin test for composite numbers described in Table 3.2 below.

	Input. Integer n to be tested, integer a as potential witness.
1.	If n is even or $1 < \gcd(a,n) < n$, return **Composite**.
2.	Write $n-1 = 2^k q$ with q odd.
3.	Set $a = a^q \pmod{n}$.
4.	If $a \equiv 1 \pmod{n}$, return **Test Fails**.
5.	Loop $i = 0, 1, 2, \ldots, k-1$
6.	If $a \equiv -1 \pmod{n}$, return **Test Fails**.
7.	Set $a = a^2 \mod n$.
8.	End i loop.
9.	Return **Composite**.

Table 3.2: Miller-Rabin test for composite numbers

Now suppose Bob wants to check whether a large number n is prime or not. To do this, he runs the Miller-Rabin test using a bunch of randomly selected values of a. Note that this is better than using the Fermat's little theorem test because there are no Carmichael-like numbers for the Miller-Rabin test. In fact, every composite number has a lot of Miller-Rabin witnesses which is described in the following proposition.

Proposition 3.4.5

Let n be an odd composite number. Then at least 75% of the numbers a between 1 and $n-1$ are Miller-Rabin witness for n.

Now suppose Bob chooses a large number n and runs the Miller-Rabin test for n for say 10 different values of a. If any a value if a Miller-Rabin witness for n, then clearly n is composite. But suppose none of the a values chosen by Bob is a Miller-Rabin witness for n. Proposition 3.4.5 tells that if n is composite, then there are at least 75% chances of getting a witness. Since Bob has found no witness in 10 tries, the probability of n being composite is at most $(25\%)^{10}$ which is approximately 10^{-6}. If Bob uses 100 different values of a, and if none of them is a witness for n, then the probability for n to be composite is less than $(25\%)^{100} \approx 10^{-6}$.

Example 3.4.6. We take the Miller-Rabin test for $n = 561$ and $a = 2$. Recall that $n = 651$ is a Carmichael number. We factor

$$n - 1 = 560 = 2^4 \cdot 35$$

and then we compute

$$\begin{aligned} 2^{35} &\equiv 263 && \pmod{561}, \\ 2^{2\cdot 35} &\equiv 263^2 \equiv 166 && \pmod{561}, \\ 2^{4\cdot 35} &\equiv 166^2 \equiv 67 && \pmod{561}, \\ 2^{8\cdot 35} &\equiv 67^2 \equiv 1 && \pmod{561}. \end{aligned}$$

The first number $2^{35} \mod 561$ is neither 1 nor -1, and the other number in the lists are not equal to -1. So 2 is a Miller-Rabin witness which shows that 561 is composite. ∎

Example 3.4.7. Take $n = 172947529$ and factor $n-1$ to get

$$n-1 = 172947528 = 2^3 \cdot 21618441.$$

We apply the Miller-Rabin test with $a = 17$ and find that

$$17^{21618441} \equiv 1 \pmod{172947529}.$$

Thus, 17 is not a Miller-Rabin witness for n. Next, we try $a = 3$ and find that

$$2^{21618441} \equiv -1 \pmod{172947529}.$$

So $a = 3$ also fails to be a Miller-Rabin witness for n. At this point, we might think that n is prime. But if we take $a = 23$, then we find that

$$\begin{aligned} 23^{21618441} &\equiv 40063806 \pmod{172947529}, \\ 23^{2\cdot 21618441} &\equiv 2257065 \pmod{172947529}, \\ 23^{4\cdot 21618441} &\equiv 1 \pmod{172947529}. \end{aligned}$$

Thus, 23 is a Miller-Rabin witness for n and $n = 172947529$ turns out to be a composite number. In fact, note that, n is a Carmichael number but it is not so easy to factor n by hand. ∎

3.4.2 The Distribution of the Set of Primes

If Bob chooses a large number at random, what is the likelihood that it is prime? The answer to this question is given by one of the most famous and important result of Number Theory which is called the prime number theorem. First, we have the following definition.

Definition 3.4.8

For any number X, let

$$\pi(X) = (\#\text{ of primes } p \text{ satisfying } 2 \le p \le X).$$

For example, $\pi(10) = 4$, since the primes between 2 and 10 are 2, 3, 5, and 7.

Theorem 3.4.9: The Prime Number Theorem

$$\lim_{X\to\infty} \frac{\pi(X)}{X/\ln(X)} = 1.$$

Proof. Beyond the scope of our syllabus (and our text). □

Example 3.4.10. How many primes are there between 900000 and 1000000? The prime number theorem says that

$$\begin{aligned} &(\text{Number of primes between } 900000 \text{ and } 1000000) \\ &\quad = \pi(1000000) - \pi(900000) \approx \frac{1000000}{\ln 1000000} - \frac{900000}{\ln 900000} = 6737.62\ldots. \end{aligned}$$

It turns out that there are exactly 7224 primes between 900000 and 1000000. ∎

For cryptographic purposes, we need even larger primes. For example, we need primes having approximately 300 decimal digits or almost equivalently, primes that are 1024 bits in length. Note that $2^{1024} \approx 10^{308.25}$.
How many primes p satisfy $2^{1023} < p < 2^{1024}$. The prime number theorem gives

$$\#\text{ of 1024 bit primes } = \pi(2^{1024}) - \pi(2^{1023}) \approx \frac{2^{1024}}{\ln 1024} - \frac{2^{1023}}{\ln 1023} \approx 2^{1013.53}.$$

So there should be a lot of primes in this interval.
Intuitively, the prime number theorem says that if we look at all the numbers between 1 and X, then the proportion of them that are primes is approximately $1/\ln(X)$. In other words, the prime number theorem says:

$$\boxed{\text{A randomly chosen number } N \text{ has probability } 1/\ln(N) \text{ of being prime.}} \tag{3.6}$$

Clearly a number is either prime or composite. What (3.6) describes is that how many primes one expects to find in an interval around N.

Remark 3.4.11. There are many deep open questions concerning the distribution of prime numbers. One of the most important and famous such result is the *Riemann hypothesis*. The Riemann zeta function $\zeta(s)$ is defined by the series

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s},$$

which converges when s is a complex number with real part greater than 1. IT has an analytic continuation to the entire complex plane with a simple pole at $s = 1$ and no other poles. The Riemann hypothesis says that if $\zeta(\sigma + it) = 0$ with σ and t real and $0 \leq \sigma \leq 1$, then in fact $\sigma = \frac{1}{2}$.
Note that $\zeta(s)$ is also equal to the product

$$\zeta(s) = \prod_{p \text{ prime}} \left(1 - \frac{1}{p^s}\right)^{-1}.$$

So $\zeta(s)$ incorporates information about the set of prime numbers.

3.4.3 Primality Proofs Verses Probabilistic Tests

The Miller-Rabin test is a powerful and practical method for finding large numbers that are "probably prime". Proposition 3.4.5 says that every composite number has many Miller-Rabin witnesses. So after 50 or 100 repetitions of Miller-Rabin test, if one doesn't find a witness then the number n is highly likely to be prime. However, there is no certainty in this.
Suppose Bob is not satisfied with the evidence provided by the Miller-Rabin test and he wants to make sure that the number n is prime. For this he checks whether n is divisible by any of the numbers 2, 3, 4... upto $\sqrt{n}$. If none of them divides n, then n is definitely prime. Unfortunately, if n is large, say $n \approx 2^{1000}$, then it is almost impossible task as the running time of this naive method is $\mathcal{O}(\sqrt{n})$ which means that it is an exponential-time algorithm, since $\sqrt{n}$ is exponential in the number of bits required to write down the number n.
It would be better if we have a polynomial-time algorithm that proves primality. If a generalized version of the Riemann hypothesis is true, then the following proposition says that this can be done (we see this without proof).

Proposition 3.4.12

If a generalized version of the Riemann hypothesis is true, then every composite number n has a Miller-Rabin witness a for its compositeness satisfying

$$a \leq 2(\ln n)^2.$$

Thus, we need to apply the Miller-Rabin test using every a smaller than $2(\ln n)^2$. If some a proves n is composite, then it is composite, otherwise by the above proposition, n is prime. Unfortunately, the proof of the above proposition assumes that the generalized Riemann hypothesis is true, and no one has yet been able to prove the original Riemann hypothesis. After many years of research, in 2002, M. Agarwal, N. Kayal, and N. Saxena found a polynomial-time algorithm for primality test. We have the following result which we see without proof.

Theorem 3.4.13: AKS Primality Test

For every $\varepsilon > 0$, there is an algorithm that conclusively determines whether a given number n is prime in no more than $\mathcal{O}((\ln n)^{6+\varepsilon})$ steps.

Since AKS algorithm is much slower than the Miller-Rabin test, in practice, most people willingly accept that a number is prime if it passes the Miller-Rabin test for say $50-100$ randomly chosen values of a.

3.5 Pollard's $p-1$ Factorization Algorithm

In the previous section, we saw that it is a bit easy to check whether a large number is (probably) prime or not. This is beneficial for the RSA cryptosystem since it requires large primes in order to operate.
Conversely, the security of RSA relies on the difficulty of factoring large numbers. The paradox of RSA is that in order to make the RSA more efficient, we want to use the modulus $N = pq$ as small as possible. On the other hand, if an opponent can factor N, then messages can be decrypted easily and our system is not secure. It is thus important to understand how hard it is to factor large numbers, and in particular, to understand the capabilities of different algorithms that are currently used for factorization.
In this section, we study an algorithm called *Pollard's $p-1$ method* which is very efficient for certain types of numbers. Pollard's method shows that there are insecure RSA moduli which appear to be secure at the first glance. That is why the study of Pollard's method is important. In addition, this method provides the inspiration for Lenstra's elliptic curve factorization method.
We are given a number $N = pq$ and our task is to determine the prime factors p and q. Suppose that we manage to find an integer L with the property that

$$p-1 \text{ divides } L \quad \text{and} \quad q-1 \text{ does not divide } L.$$

This means that there are integers i, j, and k with $k \neq 0$ satisfying

$$L = i(p-1) \quad \text{and} \quad L = j(q-1)+k.$$

Let us consider what happens if we take a randomly chosen integer a and compute a^L. By Fermat's little theorem, we have

$$a^L = a^{i(p-1)} = (a^{p-1})^i \equiv 1^i \equiv 1 \pmod{p},$$
$$a^L = a^{j(q-1)+k} = a^k (a^{q-1})^j \equiv a^k \cdot 1^j \equiv a^k \pmod{q}.$$

Since the exponent k is not equal to 0, it is unlikely that a^k will be congruent to 1 modulo q. Thus, with very high probability, i.e., for most choices of a, we find that

$$p \text{ divides } a^L - 1 \quad \text{and} \quad q \text{ does not divide } a^L - 1.$$

This means that we can recover p by the simple gcd computation

$$p = \gcd(a^L - 1, N),$$

where $N = pq$.
The question is can we find such an exponent L which is divisible by $p-1$ but not by $q-1$? Pollard's observation is that if $p-1$ happens to be a product of many small primes, then it will divide $n!$ for some value of n which is not too large. Thus, for each number $n = 2, 3, 4, \ldots$ we choose a value of a and compute

$$\gcd(a^{n!} - 1, N).$$

In practice, we simply take $a = 2$. If the gcd is equal to 1, then we go on to the next value of n. If the gcd equals N anytime, then we are unfortunate. But then a different value of a will probably work. If we get a number strictly between 1 and N, then we have a nontrivial factor of N and we are done.

Remark 3.5.1. Before coming to the Pollard's idea, we make the following two important remarks.
The first point is the quantity $a^{n!} - 1$ will be too large. Even for $a = 2$ and for quite moderate values of n, say $n = 100$, it is not feasible to compute $a^{n!} - 1$ exactly. Note that the number 2^{100} has more than 10^{157} digits. But fortunately, we need not compute this number exactly. We are only interested in the greatest common divisor of $a^{n!} - 1$ and N. So it suffices to compute

$$a^{n!} - 1 \pmod{N}$$

and then take the gcd with N. Thus, we never need to work with numbers larger than N.
Second, we do not even need to compute the exponent $n!$ every time. Assuming that we have already computed $a^{n!} \mod N$ in the previous step, we can compute the next value as

$$a^{(n+1)!} \equiv \left(a^{n!}\right)^{n+1} \pmod{N}.$$

This leads to the algorithm describe in Table 3.3 below.

	Input. Integer N to be factored.
1.	Set $a = 2$ (or some other convenient value).
2.	Loop $j = 2, 3, 4, \ldots$ upto a specified bound.
3.	Set $a = a^j \mod N$.
4.	Compute $d = \gcd(a-1, N)^\dagger$.
5.	If $1 < d < N$ then **success**, return d.
6.	Increment j and loop again at Step **2**.

† For added efficiency, choose an appropriate k and compute the gcd in Step **4** only every kth iteration.

Table 3.3: Pollard's $p-1$ factorization algorithm

Remark 3.5.2. The fast powering algorithm studied in Unit 1 gives a method for computing $a^k \mod N$ in at most $2\log_2 k$ steps, where each step is a multiplication modulo N. Stirling's formula says that if n is large, then $n!$ is approximately equal to $(n/e)^n$. So we can compute $a^{n!} \mod N$ in $2n\log_2(n)$ steps. Thus, it is feasible to compute $a^{n!} \mod N$ for reasonably large values on n.

Example 3.5.3. Use Pollard's $p-1$ method to factor $N = 13927189$.

Starting with $\gcd(2^{9!}-1, N)$ and taking successively larger factorials in the exponent we find that

$$\begin{aligned}
2^{9!}-1 &\equiv 13867883 \pmod{13927189}, & \gcd(2^{9!}-1, 13927189) &= 1,\\
2^{10!}-1 &\equiv 5129508 \pmod{13927189}, & \gcd(2^{10!}-1, 13927189) &= 1,\\
2^{11!}-1 &\equiv 4405233 \pmod{13927189}, & \gcd(2^{11!}-1, 13927189) &= 1,\\
2^{12!}-1 &\equiv 6680550 \pmod{13927189}, & \gcd(2^{12!}-1, 13927189) &= 1,\\
2^{13!}-1 &\equiv 6161077 \pmod{13927189}, & \gcd(2^{13!}-1, 13927189) &= 1,\\
2^{14!}-1 &\equiv 879290 \pmod{13927189}, & \gcd(2^{14!}-1, 13927189) &= 3823.
\end{aligned}$$

The last line gives a nontrivial factor $p = 3823$ of N. This factor is prime and the other factor $q = N/p = 13927189/3823 = 3643$ is also prime.
The reason that an exponent of 14! worked in this case is that $p-1$ factors into a product of small primes,

$$p-1 = 3822 = 2 \cdot 3 \cdot 7^2 \cdot 13.$$

The other factor $q-1 = 3642 = 2 \cdot 3 \cdot 607$, which is not a product of small primes. ∎

Example 3.5.4. Let us consider one more example using larger numbers. Let $N = 168441398857$. Then

$$\begin{aligned}
2^{50!}-1 &\equiv 114787431143 \pmod{N}, & \gcd(2^{50!}-1, N) &= 1,\\
2^{51!}-1 &\equiv 36475745067 \pmod{N}, & \gcd(2^{51!}-1, N) &= 1,\\
2^{52!}-1 &\equiv 67210629098 \pmod{N}, & \gcd(2^{52!}-1, N) &= 1,\\
2^{53!}-1 &\equiv 8182353513 \pmod{N}, & \gcd(2^{53!}-1, N) &= 350437.
\end{aligned}$$

So using $2^{53!}-1$ gives the prime factor $p = 350437$ of N. The other prime factor of N is $q = N/p = 480661$. Note that $p-1$ is a product of small prime factors,

$$p-1 = 350436 = 2^2 \cdot 3 \cdot 19 \cdot 29 \cdot 53.$$

■

Remark 3.5.5. Note that it is easy for Bob and Alice to avoid dangers of Pollard's $p-1$ method when creating RSA keys. They simply check that their chosen secret primes p and q have the property that neither $p-1$ nor $q-1$ can be factored into small primes. From cryptographic point of view it is an important lesson. Most of the times people would not expect, at first glance, that factorization properties of $p-1$ and $q-1$ should have to do anything with the difficulty of factoring pq.
Thus, the moral is that even if we build a cryptosystem based on a seemingly hard problem such as integer factorization, we must be wary of special cases of the problem like this that are easier to solve than the general case. We have already seen an example of this in the Pohlig-Hellman algorithm for the discrete logarithm problem.

Remark 3.5.6. Suppose that p and q are randomly chosen primes of about the same size. Pollard's method works if at least one of $p-1$ and $q-1$ factors entirely into a product of small prime powers. Clearly $p-1$ is even and so we can take out one factor of 2. But then the quantity $\frac{1}{2}(p-1)$ should behave more or less like a random number of size approximately $\frac{1}{2}p$. This leads to the following question:

> What is the probability that a randomly chosen integer of size approximately n divides $B!$?

Note that, in particular, if n divides $B!$, then every prime ℓ dividing n must satisfy $\ell \leq B$. A number whose prime factors are all less than or equal to B is called a *B-smooth number*. It is thus natural to ask for the probability that a randomly chosen integer of size approximately n is a B-smooth number or not. We can also ask the following:

> Given n, how large should we choose B so that a randomly chosen integer of size approximately n has a reasonably good probability of being a B-smooth number?

The efficiency (or lack) of all modern methods of integer factorization is largely determined by the answer to this question.

CHAPTER 4

Elliptic Curve Cryptography

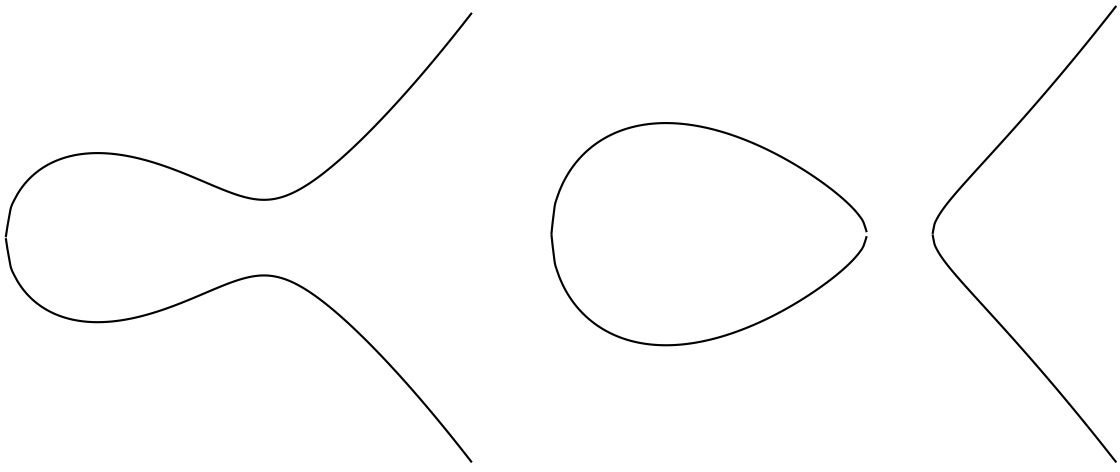
4.1 Elliptic Curves

An *elliptic curve* is the set of solutions to an equation of the form

$$Y^2 = X^3 + AX + B.$$

Equations of this form are called *Weierstrass equations*. Two examples of elliptic curves are given below and shown in Figure 4.1.

$$E_1 : Y^2 = X^3 - 3X + 3 \quad \text{and} \quad E_2 : Y^2 = X^3 - 6X + 5.$$



$E_1 : Y^2 = X^3 - 3X + 3$ $E_2 : Y^2 = X^3 - 6X + 5$

Figure 4.1: Two examples of elliptic curves

An amazing property of elliptic curves is that there is a natural way to add two points on the curve to get a third point. Let P and Q be two points on an elliptic curve E as shown in Figure 4.2. We start by drawing the line L through P and Q. This line L intersects E at three

points, namely P, Q, and one other point R. We take that point R and reflect it across the X-axis (i.e., we multiply its Y-coordinate by -1) to get a new point R'. The point R' is called the "sum of P and Q" and we denote this addition law by $\oplus$. Thus, we write

$$P \oplus Q = R'.$$

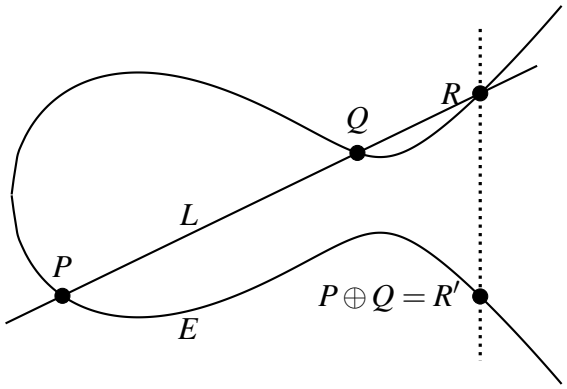


Figure 4.2: The addition law on an elliptic curve

Example 4.1.1. Let E be the elliptic curve

$$Y^2 = X^3 - 15X + 18. \tag{4.1}$$

The points $P = (7,16)$ and $Q = (1,2)$ are on the curve E. The line L connecting them is given by the equation (using $\frac{y-y_1}{y_2-y_1} = \frac{x-x_1}{x_2-x_1}$),

$$L : Y = \frac{7}{3}X - \frac{1}{3}. \tag{4.2}$$

In order to find the points where E and L intersect, we substitute (4.2) in (2.7) and solve for X. Thus, we have

$$\begin{aligned} \left(\frac{7}{3}X - \frac{1}{3}\right)^2 &= X^3 - 15X + 18, \\ \frac{49}{9}X^2 - \frac{14}{9}X + \frac{1}{9} &= X^3 - 15X + 18, \\ 0 &= X^3 - \frac{49}{9}X^2 - \frac{121}{9}X + \frac{161}{9}. \end{aligned}$$

We need to find the roots of this cubic polynomial. In general, this may be a difficult task but since we already know that P and Q are in the intersection $E \cap L$, we already know two roots, which are $X = 7$ and $X = 1$. Then it is easy to find the other factor and we find that

$$X^3 - \frac{49}{9}X^2 - \frac{121}{9}X + \frac{161}{9} = (X-7)\cdot(X-1)\cdot\left(X + \frac{23}{9}\right),$$

so the third point of intersection of L and E has X-coordinate equal to $-\frac{23}{9}$. Next we find the Y-coordinate by substituting $X = -\frac{23}{9}$ in (4.2). This gives $R = \left(-\frac{23}{9}, -\frac{170}{27}\right)$. Finally, we reflect this point across X-axis to get

$$P \oplus Q = \left(-\frac{23}{9}, \frac{170}{27}\right).$$

■

What happens if we want to add the point to itself? Visualize the line L connecting P and Q and the point Q sliding along to curve, getting closer and closer to P and finally coinciding with P. What happens to the line L in that case? In the limit as Q approaches P, the line L becomes the tangent line to E at P. Thus in order to add a point P to itself, we take L to be the tangent to E at the point P as shown in Figure 4.3. Then the line L intersects E at P and at one other point R. Then as before, we take the reflection of R across X-axis to get the point $2P = P \oplus P = R'$. In some sense, the line L intersects E at three points, where we count it intersecting E at the point P twice.

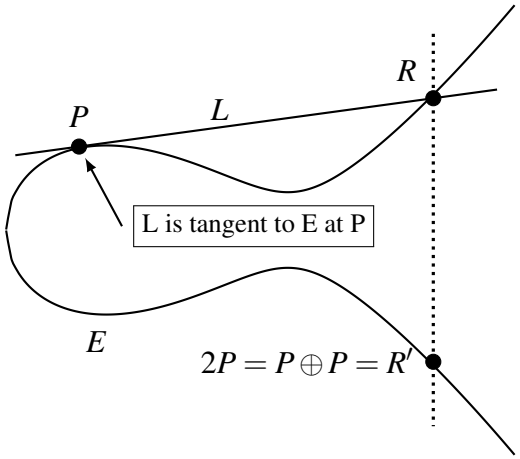


Figure 4.3: Adding a point to itself

Example 4.1.2. Let E be the elliptic curve $Y^2 = X^3 - 15X + 18$ as in equation (4.1) and P be the point $P = (7, 16)$ as in the above example. We compute $P \oplus P$. The slope of E at P is computed by differentiating equation (4.1),

$$2Y\frac{dY}{dX} = 3X^2 - 15, \quad \text{so} \quad \frac{dY}{dX} = \frac{3X^2 - 15}{2Y}.$$

Substituting the coordinates of $P = (7, 16)$ gives the slope $\lambda = \frac{33}{8}$. So the tangent line to E at the point P is given by the equation

$$L : Y = \frac{33}{8}X - \frac{103}{8} \tag{4.3}$$

Now substituting (4.3) in (4.1) for E, we get

$$\left(\frac{33}{8}X - \frac{103}{8}\right)^2 = X^3 - 15X + 18,$$
$$X^3 - \frac{1089}{64}X^2 + \frac{2919}{32}X - \frac{9457}{64} = 0,$$
$$(X-7)^2 \cdot \left(X - \frac{193}{64}\right) = 0.$$

Since the X-coordinate of P, $X = 7$, appears as a double root of the cubic polynomial, it becomes easy to factor the cubic polynomial. Finally, substituting $X = \frac{193}{64}$ into equation (4.3) for L, we get $Y = -\frac{223}{512}$. Then changing the sign of the Y-coordinate, we get

$$P \oplus P = \left(\frac{193}{64}, \frac{223}{512}\right).$$

∎

Now, what if we want to add a point $P = (a,b)$ to its reflection about the X-axis $P' = (a,-b)$? The line L through P and P' is the vertical line $x = a$ and it intersects E in only two points P and P' as shown in Figure 4.4. There is no third point of intersection. The way out is to create an extra point $\mathcal{O}$ which is called the "*point at infinity*". This point $\mathcal{O}$ does not exist in the XY-plane, but we consider that it lies on the vertical line. We then set

$$P \oplus P = \mathcal{O}.$$

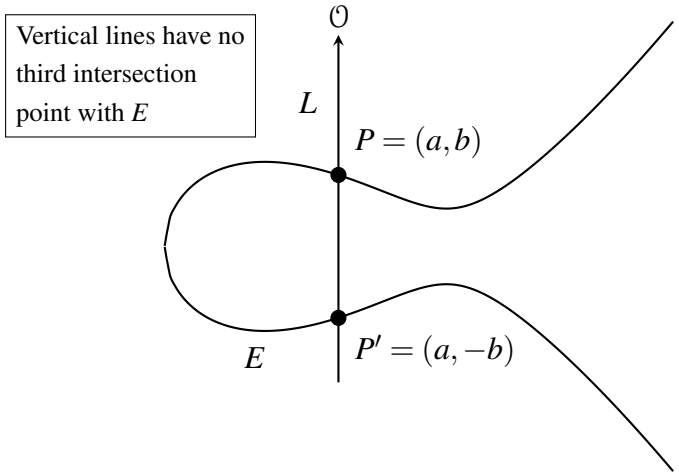


Figure 4.4: Adding a point to itself

We also need to state how to add $\mathcal{O}$ to an ordinary point $P = (a,b)$ on E. The line L joining P and $\mathcal{O}$ is the vertical line through P, since $\mathcal{O}$ lies on the vertical lines, and that vertical line intersects E at the points P, $\mathcal{O}$, and $P' = (a,-b)$. To add P to $\mathcal{O}$, we reflect P' across the X-axis, which gives us back P. In other words, $P \oplus \mathcal{O} = P$. Thus, the point $\mathcal{O}$ acts like zero (i.e., identity element) for the elliptic curve addition.

Example 4.1.3. Let E be the elliptic curve $Y^2 = X^3 - 15X + 18$ as in the above example and consider the point $T = (3,0)$ on E. Note that the tangent line to E at the point T is the vertical line $X = 3$. Hence, if we add T to itself, we get $T \oplus T = \mathcal{O}$. ∎

Definition 4.1.4

An *elliptic curve* E is the set of solutions to a Weierstrass equation

$$E : Y^2 = X^3 + AX + B,$$

together with an extra point $\mathcal{O}$, where the constants A and B must satisfy

$$4A^3 + 27B^2 \neq 0.$$

The *addition law* on E is defined as follows:
Let P and Q be two points on E. Let L be the line connecting P and Q, or the tangent line to E at P if $P = Q$. Then the intersection of E and L consists of three points P, Q, and R, counted with appropriate multiplicities and with the understanding that $\mathcal{O}$ lies on every vertical line.

Writing $R = (a,b)$, the sum of P and Q is defined to be the reflection $R' = (a,-b)$ of R across the X-axis. This sum is denoted by $P \oplus Q$, or sometimes simply by $P+Q$.
Further if $P = (a,b)$, we denote the reflected point by $\ominus P = (a,-b)$, or simply by $-P$l and we define $P \ominus Q$ (or $P-Q$) to be $P \oplus (\ominus Q)$. Similarly, repeated addition is represented as multiplication of a point by an integer,

$$nP = \underbrace{P+P+P+\cdots+P}_{n \text{ copies}}.$$

Remark 4.1.5. Note that in the definition of elliptic curves, we have a condition $4A^3+27B^2 \neq 0$. The quantity $\Delta_E = 4A^3+27B^2$ is called the *discriminant* of E. The condition $\Delta_E \neq 0$ is equivalent to the condition that the cubic polynomial X^3+AX+B have no repeated roots, i.e., if we factor X^3+AX+B completely as

$$X^3+AX+B = (X-e_1)(X-e_2)(X-e_3),$$

where e_1, e_2, e_3 are allowed to be complex numbers, then

$$4A^3+27B^2 \neq 0 \quad \text{if and only if} \quad e_1, e_2, e_3 \text{ are distinct.}$$

(See Exercise 4.1). The curves with $\Delta_E = 0$ have singular points and the addition law does not work well on such curves. That is why the condition $\Delta_E \neq 0$ is included in the definition of an elliptic curve.

Exercise 4.1

Suppose that the cubic polynomial X^3+AX+B factors as

$$X^3+AX+B = (X-e_1)(X-e_2)(X-e_3).$$

Prove that $4A^3+27B^2 = 0$ if and only if two (or more) of e_1, e_2, and e_3 are the same.
(Hint: Multiply out the right hand side and compare coefficients to relate A and B to e_1, e_2, and e_3).

Theorem 4.1.6

Let E be an elliptic curve. Then the addition law on E has the following properties:

(a)	$P+\mathcal{O} = P = \mathcal{O}+P$	for all $P \in E$.	[Identity]
(b)	$P+(-P) = \mathcal{O}$	for all $P \in E$.	[Inverse]
(b)	$(P+Q)+R = P+(Q+R)$	for all $P,Q,R \in E$.	[Associative]
(b)	$P+Q = Q+P$	for all $P,Q \in E$.	[Commutative]

In other words, the addition law makes the points of E into an abelian group.

Proof. As discussed earlier, the identity law (a) and the inverse law (b) are true because $\mathcal{O}$ lies on all vertical lines. The commutative law (d) is easy to verify, since the line that goes through P and Q is same as the line that goes through Q and P. So the order of the points does not matter.

The associative law (c) can be verified geometrically by putting all the lines needed and check that it is true. There are many ways to prove the associative law, but none of the proofs are easy. From the following theorem, the associative law can be verified by a direct calculations from the explicit formulas derived for the addition law. □

Next, we find the explicit formulas to enable us to easily add and subtract points on an elliptic curve. The derivation of these formulas uses elementary analytic geometry, some differential calculus to find a tangent line, and a certain amount of algebraic manipulation. We state the following theorem in the form of an algorithm and give a brief proof of the same.

Theorem 4.1.7: Elliptic Curve Addition Algorithm

Let

$$E : Y^2 = X^3 + AX + B$$

be an elliptic curve and let P_1 and P_2 be points on E.

(a) If $P_1 = \mathcal{O}$, then $P_1 + P_2 = P_2$.

(b) Otherwise, if $P_2 = \mathcal{O}$, then $P_1 + P_2 = P_1$.

(c) Otherwise, write $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$.

(d) If $x_1 = x_2$ and $y_1 = -y_2$, then $P_1 + P_2 = \mathcal{O}$.

(e) Otherwise, define λ by

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{if } P_1 \neq P_2, \\ \frac{3x_1^2 + A}{2y_1} & \text{if } P_1 = P_2, \end{cases}$$

and let

$$x_3 = \lambda^2 - x_1 - x_2 \quad \text{and} \quad y_3 = \lambda(x_1 - x_3) - y_1.$$

Then $P_1 + P_2 = (x_3, y_3)$.

Proof. Parts (a) and (b) are clear. Part (d) is the case where the line passing through P_1 and P_2 is vertical. So $P_1 + P_2 = \mathcal{O}$. Note that if $y_1 = y_2 = 0$, then the tangent line is vertical, so this case works too.
For part (e), we note that if $P_1 \neq P_2$, then λ is the slope of the line passing through P_1 and P_2 and $P_1 = P_2$, then λ is the slope of the tangent line at $P_1 = P_2$. In either case, the line L is given by the equation $Y = \lambda X + \nu$, where $\nu = y_1 - \lambda x_1$. Substituting the equation for L into the equation for E, we get

$$(\lambda X + \nu)^2 = X^3 + AX + B,$$

so

$$X^3 - \lambda^2 X^2 + (A - 2\lambda\nu)X + (B - \nu^2) = 0.$$

We know that this cubic has x_1 and x_2 as two of its roots. If we denote the third root by x_3, then it factors as

$$X^3 - \lambda^2 X^2 + (A - 2\lambda\nu)X + (B - \nu^2) = (X - x_1)(X - x_2)(X - x_3).$$

Now multiplying out the right hand side and comparing the coefficient of X^2 on both sides, we get

$$-x_1 - x_2 - x_3 = -\lambda^2.$$

Therefore, we get $x_3 = \lambda^2 - x_1 - x_2$, and then the Y-coordinate of the third intersection point of E and L is given by $\lambda x_3 + \nu$. Finally, in order to find $P_1 + P_2$, we reflect across the X-axis, which means that the Y-coordinate has to be replaced by its negative. This completes the proof. □

In the next section, we shall study elliptic curve over finite fields.

4.2 Elliptic Curves over Finite Fields

In order to apply elliptic curves to cryptography, we need to study elliptic curves whose points have coordinates in a finite field $\mathbb{F}_p$.

Definition 4.2.1

Let $p \geq 3$ be a prime. An *elliptic curve over* $\mathbb{F}_p$ is an equation of the form

$$E : Y^2 = X^3 + AX + B \quad \text{with } A, B \in \mathbb{F}_p \text{ satisfying } 4A^3 + 27B^2 \neq 0.$$

The set of points on E with coordinates in $\mathbb{F}_p$ is the set

$$E(\mathbb{F}_p) = \{(x,y) : x, y \in \mathbb{F}_p \text{ satisfy } y^2 = x^3 + Ax + B\} \cup \{\mathcal{O}\}.$$

Example 4.2.2. Consider the elliptic curve

$$E : Y^2 = X^3 + 3X + 8 \quad \text{over the field } \mathbb{F}_{13}.$$

We can find the points of $E(\mathbb{F}_{13}$ by substituting in all the possible values of $X = 0, 1, 2, \ldots, 12$ and checking for which X values the quantity $X^3 + 3X + 8$ is a square modulo 13. For example, putting $X = 0$ gives 8, and 8 is not a square modulo 13. Next, we try $X = 1$. This gives $1 + 3 + 8 = 12$ and it turns out that 12 is a square modulo 13. In fact, it has two square roots,

$$5^2 \equiv 12 \pmod{13} \quad \text{and} \quad 8^2 \equiv 12 \pmod{13}.$$

This gives us two points $(1,5)$ and $(1,8)$ in $E(\mathbb{F}_{13})$. Continuing this way, we can find a complete list.

$$E(\mathbb{F}_{13}) = \{\mathcal{O}, (1,5), (1,8), (2,3), (2,10), (9,6), (9,7), (12,2), (12,11)\}.$$

Thus, $E(\mathbb{F}_{13})$ consists of nine points. ∎

Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be points on $E(\mathbb{F}_p)$. We define the sum $P + Q$ to be the point (x_3, y_3) obtained by applying the elliptic curve addition algorithm (Theorem 4.1.7). Note that in this algorithm, the only operations used are addition, subtraction, multiplication, and division involving the coefficients of E and the coordinates of P and Q. Since those coefficients and coordinates are in the field $\mathbb{F}_p$, we end up with a point (x_3, y_3) whose coordinates are in the field $\mathbb{F}_p$. However, it is not completely clear that (x_3, y_3) is a point in $E(\mathbb{F}_p)$.

Theorem 4.2.3

Let E be an elliptic curve over $\mathbb{F}_p$ and let P and Q be points in $E(\mathbb{F}_p)$.

(a) The elliptic curve addition algorithm (Theorem 4.1.7) applied to P and Q yields a point in $E(\mathbb{F}_p)$. We denote this point by $P+Q$.

(b) This addition law on $E(\mathbb{F}_p)$ satisfies all of the properties listed in Theorem 4.1.6 (i.e., properties of the addition law on elliptic curves). In other words, this addition law makes $E(\mathbb{F}_p)$ into a finite group.

Proof. The formulas in Theorem 4.1.7 (e) are derived by substituting the equation of a line into the equation for E and solving for X. So the resulting point is automatically a point on E, i.e., it is a solution to the equation defining E. This shows why (a) is true, although when $P = Q$ (Justify!).
For (b), the identity law follows from the addition algorithm steps (a) and (b), the inverse law is clear from the addition algorithm Step (d), and the commutative law is easy, since a brief examination of the addition algorithm shows that switching the two points leads to the same result.
Unfortunately, the associative law is not so clear. It is possible to verify the associative law directly using the addition algorithm formulas, although there are many special cases to consider. □

Example 4.2.4. Let $E : Y^2 = X^3 + 3X + 8$ be the elliptic curve over $\mathbb{F}_{13}$. We use the addition algorithm (Theorem 4.1.7) to add the points $P = (9,7)$ and $Q = (1,8)$ in $E(\mathbb{F}_{13})$. Step (e) of that algorithm tells us to first compute

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} = \frac{8-7}{1-9} = \frac{1}{-8} = \frac{1}{5} = 8,$$

where the computations are being performed in the field $\mathbb{F}_{13}$. So $-8 = 5$ and $\frac{1}{5} = 5^{-1} = 8$. Next we compute

$$\nu = y_1 - \lambda x_1 = 7 - 8 \cdot 9 = -65 = 0.$$

Finally, the addition algorithm tells us to compute

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 = 64 - 9 - 1 = 54 = 2, \\ y_3 &= -(\lambda x_3 + \nu) = -8 \cdot 2 = -16 = 10. \end{aligned}$$

This completes the computation of

$$P + Q = (1,8) + (9,7) = (2,10) \quad \text{in } E(\mathbb{F}_{13}).$$

Similarly, we can use the addition algorithm to add $P = (9,7)$ to itself. We have

$$\lambda = \frac{3x_1^2 + A}{2y_1} = \frac{3 \cdot 9^2 + 3}{2 \cdot 7} = \frac{246}{14} = 12 \quad \text{and} \quad \nu = y + 1 - \lambda x_1 = 7 - 12 \cdot 9 = 3.$$

Then

$$x_3 = \lambda^2 - x_1 - x_2 = (12)^2 - 9 - 9 - 9 \quad \text{and} \quad y_3 = -(\lambda x_3 + \nu) = -(12 \cdot 9 + 3) = 6.$$

So $P + P = (9,7) + (9,7) = (9,6)$ in $E(\mathbb{F}_{13})$. Similarly, we can add any pair of points in $E(\mathbb{F}_{13})$. The results of the addition law are listed in Table 4.1. ∎

	$\mathcal{O}$	$(1,5)$	$(1,8)$	$(2,3)$	$(2,10)$	$(9,6)$	$(9,7)$	$(12,2)$	$(12,11)$
$\mathcal{O}$	$\mathcal{O}$	$(1,5)$	$(1,8)$	$(2,3)$	$(2,10)$	$(9,6)$	$(9,7)$	$(12,2)$	$(12,11)$
$(1,5)$	$(1,5)$	$(2,10)$	$\mathcal{O}$	$(1,8)$	$(9,7)$	$(2,3)$	$(12,2)$	$(12,11)$	$(9,6)$
$(1,8)$	$(1,8)$	$\mathcal{O}$	$(2,3)$	$(9,6)$	$(1,5)$	$(12,11)$	$(2,10)$	$(9,7)$	$(12,2)$
$(2,3)$	$(2,3)$	$(1,8)$	$(9,6)$	$(12,11)$	$\mathcal{O}$	$(12,2)$	$(1,5)$	$(2,10)$	$(9,7)$
$(2,10)$	$(2,10)$	$(9,7)$	$(1,5)$	$\mathcal{O}$	$(12,2)$	$(1,8)$	$(12,11)$	$(9,6)$	$(2,3)$
$(9,6)$	$(9,6)$	$(2,3)$	$(12,11)$	$(12,2)$	$(1,8)$	$(9,7)$	$\mathcal{O}$	$(1,5)$	$(2,10)$
$(9,7)$	$(9,7)$	$(12,2)$	$(2,10)$	$(1,5)$	$(12,11)$	$\mathcal{O}$	$(9,6)$	$(2,3)$	$(1,8)$
$(12,2)$	$(12,2)$	$(12,11)$	$(9,7)$	$(2,10)$	$(9,6)$	$(1,5)$	$(2,3)$	$(1,8)$	$\mathcal{O}$
$(12,11)$	$(12,11)$	$(9,6)$	$(12,2)$	$(9,7)$	$(2,3)$	$(2,10)$	$(1,8)$	$\mathcal{O}$	$(1,5)$

Table 4.1: Addition table for $E : Y^2 = X^3 + 3X + 8$ over $\mathbb{F}_{13}$

It is clear that the set of points $E(\mathbb{F}_p)$ is a finite set since there are finitely many possibilities for the X and the Y-coordinate. There are p possibilities for X, and then for each X, the equation

$$Y^2 = X^3 + AX + B$$

shows that there are at most two possibilities for Y (see Exercise 2.5). Adding in the extra point $\mathcal{O}$, this shows that $\#E(\mathbb{F}_p)$ has at most $2p+1$ points. However, this estimate is considerably large than the actual size.
A famous result of Hasse, which was later generalized by Weil and Deligne, gives an estimate for the number of points in $E(\mathbb{F}_p)$.

Theorem 4.2.5: Hasse

Let E be an elliptic curve over $\mathbb{F}_p$. Then

$$\#E(\mathbb{F}_p) = p + 1 - t_p \qquad \text{with } t_p \text{ satisfying } |t_p| \le 2\sqrt{p}.$$

Definition 4.2.6

The quantity

$$t_p = p + 1 - \#E(\mathbb{F}_p)$$

appearing in the above theorem is called the *trace of Frobenius for* $E/\mathbb{F}_p$.

Example 4.2.7. Let E be given by the equation

$$E : Y^2 = X^3 + 4X + 6.$$

We can think of E as an elliptic curve over $\mathbb{F}_p$ for different finite fields $\mathbb{F}_p$ and count the number of points in $E(\mathbb{F}_p)$. Table 4.2 lists the results for the first few primes, together with the value of t_p and the value of $2\sqrt{p}$. ∎

p	$\#E(\mathbb{F}_p)$	t_p	$2\sqrt{p}$
3	4	0	3.46
5	8	-2	4.47
7	11	-3	5.29
11	16	-4	6.63
13	14	0	7.21
17	15	3	8.25

Table 4.2: Number of points and trace of Frobenius for $E : Y^2 = X^3 + 4X + 6$

4.3 The Elliptic Curve Discrete Logarithm Problem (ECDLP)

In Unit 2, we studied the discrete logarithm problem (DLP) in the finite field $\mathbb{F}_p^*$. In order to create a cryptosystem based on the DLP for $\mathbb{F}_p^*$, Alice publishes two numbers g and h, and her secret is the exponent x which is the solution of the congruence

$$h \equiv g^x \pmod{p}.$$

Viewing g and h as elements of the group $\mathbb{F}_p^*$, Eve needs to find an x such that

$$h \equiv \underbrace{g \cdot g \cdot g \cdots g}_{x \text{ multiplications}} \pmod{p}.$$

In other words, Eve needs to determine how many times g must be multiplied to itself in order to get h.
With this formulation, it is clear that Alice can do the same thing with the group of points $E(\mathbb{F}_p)$ of an elliptic curve E over a finite field $\mathbb{F}_p$. She chooses and publishes two points P and Q in $E(\mathbb{F}_p)$, and her secret is an integer n such that

$$Q = \underbrace{P + P + P + \cdots + P}_{n \text{ additions on } E} = nP.$$

Then Eve needs to find out how many times P must be added to itself in order to get Q. Note that although the "addition law" on an elliptic curve is denote with a plus sign, the actual addition on E is a complicated operation. So this elliptic curve analogue of the discrete logarithm problem maybe quite difficult to solve.

Definition 4.3.1: ECDLP

Let E be an elliptic curve over the finite field $\mathbb{F}_p$ and let P and Q be points in $E(\mathbb{F}_p)$. The *Elliptic Curve Discrete Logarithm Problem* (ECDLP) is the problem of finding an integer n such that $Q = nP$. By analogy with the discrete logarithm problem for $\mathbb{F}_p^*$, we denote this integer n by

$$n = \log_P(Q)$$

and we call n the *elliptic discrete logarithm of Q with respect to P*.

Remark 4.3.2. Note that the definition of $\log_P(Q)$ is not precise. First difficulty is that there may be points $P, Q \in E(\mathbb{F}_p)$ such that A is not a multiple of P. In that case, $\log_P(Q)$ is not defined. However, for cryptographic purposes, Alice chooses a public point P and her private integer n and computes $Q = nP$ for publishing. So in practical applications, $\log_P(Q)$ exists and its Alice's secret.

The second difficulty is that if there is one value of n satisfying $Q = nP$, then there are many such values. To see this, first note that there exists a positive integer s such that $sP = \mathbb{O}$. This is because the points $P, 2P, 3P, \ldots$ cannot be all distinct as $E(\mathbb{F}_p)$ is finite. Hence, there are integer $k > j$ such that $kP = jP$, and so we have $s = k - j$. The smallest such $s \geq 1$ is called the *order of* P. Thus, if s is the order of P and n_0 is an integer such that $Q = n_0P$, then the solutions to $Q = nP$ are the integers $n = n_0 + is$ with $i \in \mathbb{Z}$ (see Exercise 4.2).
This means that $\log_P(Q)$ is really an element of $\mathbb{Z}/s\mathbb{Z}$, i.e., $\log_P(Q)$ is an integer modulo s, where s is the order of P. An advantage of defining these values to be in $\mathbb{Z}/s\mathbb{Z}$ is that the elliptic discrete logarithm then satisfies

$$\log_P(Q_1 + Q_2) = \log_P(Q_1) + \log_P(Q_2) \quad \text{for all } Q_1, Q_2 \in E(\mathbb{F}_p). \tag{4.4}$$

The fact that the discrete logarithm for $E(\mathbb{F}_p)$ satisfies (4.4) means that it respects the addition law when the group $E(\mathbb{F}_p)$ is mapped to the group $\mathbb{Z}/s\mathbb{Z}$. In other words, we say that the map $\log_P$ defines a *group hoomorphism*

$$\log_P : E(\mathbb{F}_p) \to \mathbb{Z}/s\mathbb{Z}.$$

Exercise 4.2

Let E be an elliptic curve over $\mathbb{F}_p$ and let P and Q be points in $E(\mathbb{F}_p)$. Assume that Q is a multiple of P and let $n_0 > 0$ be the smallest solution to $Q = nP$. Also let $s > 0$ be the smallest solution to $sP = \mathcal{O}$. Prove that every solution to $Q = nP$ looks like $n_0 + is$ for some $i \in \mathbb{Z}$.
(Hint: Write n as $n = is + r$ for some $0 \leq r < s$ and determine the value of r.)

Example 4.3.3. Consider the elliptic curve

$$E : Y^2 = X^3 + 8X + 7 \quad \text{over } \mathbb{F}_{73}.$$

The points $P = (32, 53)$ and $Q = (39, 17)$ are both in $E(\mathbb{F}_{73})$, and it is easy to verify that

$$Q = 11P, \quad \text{so} \quad \log_P(Q) = 11.$$

Similarly, $R = (35, 47) \in E(\mathbb{F}_{73})$ and $S = (58, 4) \in E(\mathbb{F}_{73})$, and after some computations we find that they satisfy $R = 37P$ and $S = 28P$. So

$$\log_P(R) = 37 \quad \text{and} \quad \log_P(S) = 28.$$

Finally, we mention that $\#E(\mathbb{F}_{73}) = 82$, but P satisfies $41P = \mathcal{O}$. Thus, P has order $41 = \frac{82}{2}$. So only half of the points in $E(\mathbb{F}_{73})$ are multiples of P. For example, $(20, 65)$ is in $E(\mathbb{F}_{73})$ but it does not equal to any multiple of P. ∎

4.3.1 The Double-and-Add Algorithm

In cryptography, in order to use the function

$$\mathbb{Z} \to E(\mathbb{F}_p) \quad \text{given by} \quad n \mapsto nP$$

we need to efficiently compute nP from the known values n and P. If n is large, then we do not want to compute nP by computing $P, 2P, 3P, \ldots$.
The most efficient way to compute nP is very similar to the square-and-multiply algorithm (fast factoring algorithm) which was used for Diffie-Hellman and Elgamal public key cryptosystems. However, since the operation on an elliptic curve is addition instead of multiplication, we call it "double-and-add" algorithm instead of "square-and-multiply".
The underlying idea is same as before. We first write n in binary form as

$$n = n_0 + n_1 \cdot 2 + n_2 \cdot 4 + n_3 \cdot 8 + \cdots + n_r \cdot 2^r \quad \text{with } n_0, n_1, \ldots, n_r \in \{0,1\}.$$

We assume that $n_r = 1$. Next we compute the following quantities:

$$Q_0 = P,\ Q_1 = 2Q_0,\ Q_2 = 2Q_1,\ \ldots,\ Q_r = 2Q_{r-1}.$$

Notice that Q_i is simply twice the previous Q_{i-1}. So

$$Q_i = 2^i P.$$

These points are referred to as 2-power multiples of P, and computing them requires r doublings. Finally, we compute nP using at most r additional additions,

$$nP = n_0 Q_0 + n_1 Q_1 + n_2 Q_2 + \cdots + n_r Q_r.$$

This addition of two points in $E(\mathbb{F}_p)$ is referred as a *point operation*. Thus, the total time to compute nP is at most $2r$ point operations in $E(\mathbb{F}_p)$. Note that $n \geq 2^r$ and so it takes no more than $2\log_2(n)$ point operations to compute nP. This makes it feasible to compute nP even for very large values of n. The double-and-add algorithm is summarized in Table 4.3 below.

	Input. Point $P \in E(\mathbb{F}_p)$ and integer $n \geq 1$.
1.	Set $Q = P$ and $R = \mathcal{O}$.
2.	Loop while $n > 0$.
	3. If $n \equiv 1 \pmod 2$, set $R = R + Q$.
	4. Set $Q = 2Q$ and $n = \lfloor n/2 \rfloor$.
	5. If $n > 0$, continue with the loop at Step **2**.
6.	Return the point R, which equals nP.

Table 4.3: The double-and-add algorithm for elliptic curves

Example 4.3.4. Use the Double-and-Add algorithm to compute nP in $E(\mathbb{F}_p)$ for

$$n = 947, \quad E : Y^2 = X^3 + 14X + 19, \quad p = 3623, \quad P = (6, 730).$$

The binary expansion of n is

$$n = 947 = 1 + 2 + 2^4 + 2^5 + 2^7 + 2^8 + 2^9.$$

The step by step calculation which requires 9 doublings and 6 additions is given in Table 4.4 below.

Step i	n	$Q = 2^i P$	R
0	947	$(6,730)$	$\mathcal{O}$
1	473	$(2521,3601)$	$(6,730)$
2	236	$(2277,502)$	$(2149,196)$
3	118	$(3375,535)$	$(2149,196)$
4	59	$(1610,1851)$	$(2149,196)$
5	29	$(1753,2436)$	$(2838,2175)$
6	14	$(2005,1764)$	$(600,2449)$
7	7	$(2425,1791)$	$(600,2449)$
8	3	$(3529,2158)$	$(3247,2849)$
9	1	$(2742,3254)$	$(932,1204)$
10	0	$(1814,3480)$	$(3492,60)$

Table 4.4: Computing $947 \cdot (6,730)$ on $Y^2 = X^3 + 14X + 19$ modulo 3623

■

Remark 4.3.5. There is an addition technique that can be used to further reduce the time required to compute nP. The idea is to write n using sums and differences of powers of 2. This technique is advantageous because there are fewer terms and so fewer point additions are required to compute nP. Also note that subtracting two points on an elliptic curve is as easy as adding them because $-(x,y) = (x,-y)$. This is different from $\mathbb{F}_p^*$, where computing a^{-1} takes more time than multiplying two elements.
Consider the following example which illustrates the above idea. In Example 4.3.4, we saw that

$$947 = 1+2+2^4+2^5+2^7+2^8+2^9.$$

So it takes 15 operations (9 doublings and 6 additions) to compute $947P$. But instead if we write

$$947 = 1+2-2^4-2^6+2^{10},$$

then we can compute

$$947P = P+2P-2^4P-2^6P+2^{10}P$$

using 10 doublings and 4 additions, a total of 14 point operations.
Writing a number n as a sum of positive and negative powers of 2 is called a *ternary expansion of* n.
How much savings is expected? Suppose that n is a large number and let $k = \lfloor \log n \rfloor + 1$. In the worst case, if n has the form $2^k - 1$, then computing nP using binary expansion of n requires $2k$ point operations (k doublings and k additions), since

$$2^k - 1 = 1+2+2^2+\cdots+2^{k-1}.$$

But if we allow ternary expansions, then the following propositions proves that computing nP does not require more than $\frac{3}{2}k+1$ point operations ($k+1$ doublings and $\frac{1}{2}k$ additions).
This was the worst case scenario. On an average, for most n, we can compute nP in about $\frac{4}{3}k+1$ steps ($k+1$ doublings and $\frac{1}{3}k$ additions).

Proposition 4.3.6

Let n be a positive integer and let $k = \lfloor \log n \rfloor + 1$, which means that $2^k > n$. Then we can always write

$$n = u_0 + u_1 \cdot 2 + u_2 \cdot 4 + u_3 \cdot 8 + \cdots + u_k \cdot 2^k \tag{4.5}$$

with $u_0, u_1, \ldots, u_k \in \{-1, 0, 1\}$ and at most $\frac{1}{2}k$ of the u_i nonzero.

Proof. Writing n in binary,

$$n = n_0 + n_1 \cdot 2 + n_2 \cdot 4 + \cdots + n_{k-1} \cdot 2^{k-1} \quad \text{with } n_0, \ldots, n_{k-1} \in \{0, 1\}.$$

Working from left to right, look for the first occurrence of two or more consecutive nonzero n_i coefficients. For example, suppose that

$$n_s = n_{s+1} = \cdots = n_{s+t-1} = 1 \quad \text{and} \quad n_{s+t} = 0$$

for some $t \geq 2$. In other words, the quantity

$$2^s + 2^{s+1} + \cdots + 2^{s+t-1} + 0 \cdot 2^{s+t} \tag{4.6}$$

appears in the binary expansion of n. Observe that

$$2^s + 2^{s+1} + \cdots + 2^{s+t-1} + 0 \cdot 2^{s+t} = 2^s(1 + 2 + 4 + \cdots + 2^{t-1}) = 2^s(2^t - 1).$$

So we can replace (4.6) with

$$-2^s + 2^{s+t}.$$

Repeating this procedure, we end up with an expansion of n of the form (4.5) in which no two consecutive u_i are nonzero. (Note that although the original binary expansion went up to only 2^{k-1}, the new expansion might go up to 2^k.) □

4.3.2 How Hard is the ECDLP?

The collision algorithms can be easily adapted to any group. In order to solve $Q = nP$, Eve chooses random integers $j_1, \ldots, j_r$ and $k_1, \ldots, k_r$ between 1 and p and makes two list of points:

$$\begin{aligned} &\text{List \#1.} \quad j_1P, j_2P, j_3P, \ldots, j_rP, \\ &\text{List \#2.} \quad k_1P + Q, k_2P + Q, k_3P + Q, \ldots, k_rP + Q. \end{aligned}$$

As soon as she finds a match (i.e., a collision) between the two lists, she is done. This is because if $j_uP = k_vP + Q$, then $Q = (j_u - k_v)P$ provides the solution. The naive collision algorithm requires quite a lot of storage for the two lists. However, there are algorithms that solve the ECDLP for $E(\mathbb{F}_p)$ in $\mathcal{O}(\sqrt{p})$ steps.

The main and important reason that elliptic curves are used in cryptography is the fact that there are no general algorithms known that solve the ECDLP in fewer than $\mathcal{O}(\sqrt{p})$ steps. In other words, the fastest known algorithms to solve the ECDLP are no better than the generic usual algorithms that works equally well to solve the discrete logarithm problem in any group. That is,

The fastest known algorithm to solve ECDLP in $E(\mathbb{F}_p)$ takes approximately $\sqrt{p}$ steps.

Thus, the ECDLP appears to be much more difficult than the DLP. However, we know that, there are some primes p for which the DLP in $\mathbb{F}_p^*$ is comparatively easy. Recall, for example, if $p-1$ is a product of small primes, then the Pohlig-Hellman algorithm gives a quick solution to the DLP in $\mathbb{F}_p^*$. In a similar way, there are some elliptic curves and some primes for which the ECDLP in $E(\mathbb{F}_p)$ is comparatively easy.

4.4 Elliptic Curve Cryptography (ECC)

In this section, we see two applications of elliptic curves to cryptography which are the Diffie-Hellman key exchange and the Elgamal public key cryptosystem.

4.4.1 Elliptic Diffie-Hellman Key Exchange

Alice and Bob agree to use a particular elliptic curve $E(\mathbb{F}_p)$ and a particular point $P \in E(\mathbb{F}_p)$. Alice chooses a secret integer n_A and Bob chooses a secret integer n_B. They compute the associated multiples

$$\overbrace{Q_A = n_A P}^{\text{Alice computes this}} \quad \text{and} \quad \overbrace{Q_B = n_B P}^{\text{Bob computes this}} .$$

Then they exchange the values Q_A and Q_B. Alice then uses her secret multiplier to compute $n_A Q_B$, and Bob similarly uses his secret integer n_B to compute $n_B Q_A$. They now have shared the secret value

$$n_A Q_B = (n_A n_B)P = n_B Q_A,$$

which they can use as a key to communicate privately via a symmetric cipher. Table 4.5 summarizes elliptic Diffie-Hellman key exchange.

Public parameter creation	
A trusted party chooses and publishes a (large) prime p, an elliptic curve E over $\mathbb{F}_p$, and a point P in $E(\mathbb{F}_p)$	
Private computations	
Alice	**Bob**
Choose a secret integer n_A. Computes the point $Q_A = n_A P$.	Choose a secret integer n_B. Computes the point $Q_B = n_B P$.
Public exchange of values	
Alice sends Q_A to Bob $\longrightarrow$	Q_A
Q_B $\longleftarrow$	Bob sends Q_B to Alice
Further private computations	
Alice	**Bob**
Computes the point $n_A Q_B$.	Computes the point $n_B Q_A$.
The shared secret value is $n_A Q_B = n_A(n_B P) = n_B(n_A P) = n_B Q_A$.	

Table 4.5: Diffie-Hellman key exchange using elliptic curves

Example 4.4.1. Alice and Bob decide to use elliptic Diffie-Hellman with prime the $p = 3851$, the elliptic curve $E : Y^2 = X^3 + 324X + 1287$, and the point $P = (920, 303) \in E(\mathbb{F}_{3851})$. Alice and Bob choose respective secret values $n_A = 1194$ and $n_B = 1759$, and then

$$\begin{aligned} \text{Alice computes} \quad & Q_A = 1194P = (2067, 2178) \in E(\mathbb{F}_{3851}), \\ \text{Bob computes} \quad & Q_B = 1759P = (3684, 3125) \in E(\mathbb{F}_{3851}). \end{aligned}$$

Alice sends Q_A to Bob and Bob sends Q_B to Alice. Finally, they compute

$$\begin{aligned} \text{Alice computes} \quad & n_A Q_B = 1194(3684, 3125) = (3347, 1242) \in E(\mathbb{F}_{3851}), \\ \text{Bob computes} \quad & n_B Q_A = 1759(2067, 2178) = (3347, 1242) \in E(\mathbb{F}_{3851}). \end{aligned}$$

Bob and Alice have exchanged the secret point $(3347, 1242)$. As explained in Remark 4.4.3 they should discard the y-coordinate and take only the value $x = 3347$ as a shared secret value. For Eve to discover Alice and Bob's shared secret key, one ways is to solve the ECDLP

$$nP = Q_A.$$

If Eve can solve this problem, then she can find n_A and can use it to compute $n_A Q_B$. There might be some other way for Eve to compute their shared secret without actually solving the ECDLP. The precise problem Eve needs to solve is the elliptic analogue of the classical Diffie-Hellman problem described in Unit 2. ∎

Definition 4.4.2: ECDHP

Let $E(\mathbb{F}_p)$ be an elliptic curve over a finite field $\mathbb{F}_p$ and let $P \in E(\mathbb{F}_p)$. The *Elliptic Curve Diffie-Hellman Problem* is the problem of computing the value of $n_1 n_2 P$ from the known values of $n_1 P$ and $n_2 P$.

Remark 4.4.3. Elliptic Diffie-Hellman key exchange requires Alice and Bob to exchange points on an elliptic curve. A point Q in $E(\mathbb{F}_p)$ consists of two coordinates $Q = (x_Q, y_Q)$, where x_Q and y_Q are elements of the finite field $\mathbb{F}_p$. So it seems that Alice must send Bob two numbers in $\mathbb{F}_p$. However, those two numbers modulo p do not contain as much information as two asbitrary numbers, since they are related by the formula

$$y_Q^2 = x_Q^3 + Ax_Q + B \quad \text{in } \mathbb{F}_p.$$

Note that Eve knows A and B, so if she can guess the correct value of x_Q, then there are only two possible values of y_Q. In practice, it is not too difficult for Eve to actually compute the two values of y_Q.
Thus, there is no point for Alice to send both the coordinates of Q_A to Bob since the y-coordinate does not contain much addition information. Thus, Alice sends Bob only the x-coordinate of Q_A. Bob then computes and uses one of the two possible y-coordinates. If he chooses the correct y-coordinate, then he is using Q_A. If he chooses the incorrect y-coordinate (which is nothing but the negative of the correct y-coordinate), then he is using $-Q_A$. Thus, in any case, Bob ends up computing one of

$$\pm n_B Q_A = \pm(n_A n_B)P.$$

Similarly, Alice ends up computing one of $\pm(n_A n_B)P$. Then Alice and Bob use the x-coordinate as their shared secret value, since the x-coordinate is the same regardless of which y they use.

Example 4.4.4. Alice and Bob decide to exchange another shared secret value using the following public parameters (same as in the previous example).

$$p = 3851, \quad E : Y^2 = X^3 + 324X + 1287, \quad P = (920, 303) \in E(\mathbb{F}_{3851}).$$

However, this time they want to send fewer bits to one another. Alice and Bob choose their new secret integers $n_A = 2489$ and $n_B = 2286$ respectively. As before, they compute

$$\begin{aligned} \text{Alice computes} \quad & Q_A = n_A P = 2489(920, 303) = (593, 719) \in E(\mathbb{F}_{3851}), \\ \text{Bob computes} \quad & Q_B = n_B P = 2286(920, 303) = (3681, 612) \in E(\mathbb{F}_{3851}). \end{aligned}$$

However, rather than sending both the coordinates, Alice sends only $x_A = 593$ to Bob and Bob sends only $x_B = 3681$ to Alice.
Alice substitutes $x_B = 3681$ into the equation for E and finds that

$$y_B^2 = x_B^3 + 324x_B + 1287 = 3681^3 + 324 \cdot 3681 + 1287 = 997.$$

Note that the calculations are performed in $\mathbb{F}_{3851}$. Alice needs to compute a square root of 997 modulo 3851. This is not hard to do, especially for prime $p \equiv 3 \pmod 4$. By Proposition 2.5.4, she knows that $b^{(p+1)/4}$ is a square root of b modulo p. So Alice sets

$$y_B = 997^{(3851+1)/4} = 997^{963} \equiv 612 \pmod{3851}.$$

It happens that she gets the same point $Q_B = (x_B, y_B) = (3681, 612)$ that Bob used, and she computes $n_A Q_B = 2489(3681, 612) = (509, 1108)$.
Similarly, Bob substitutes $x_A = 593$ into the equation for E and takes a square root,

$$y_A^2 = x_A^3 + 324x_A + 1287 = 593^3 + 324 \cdot 593 + 1287 = 927,$$

$$y_A = 927^{(3851+1)/4} = 927^{963} \equiv 3132 \pmod{3851}.$$

Bob then uses the point $Q'_A = (593, 3132)$, which is not Alice's point Q_A. Bob computes $n_B Q'_A = 2286(593, 3132) = (509, 2743)$. Thus, Bob and Alice end up with points that are negatives of one another in $E(\mathbb{F}_p)$ but this is all right as their shared secret value is the x-coordinate $x = 509$ which is same for both points. ∎

4.4.2 Elliptic Elgamal Public Key Cryptosystem

In this section, we see the elliptic curve analogue of the classical Elgamal public key cryptosystem.
Alice and Bob agree to use a particular prime p, an elliptic curve E, and a point $P \in E(\mathbb{F}_p)$. Alice chooses a secret multiplier n_A and publishes the point $Q_A - n_A P$ as her public key. Bob's plaintext is a point $M \in E(\mathbb{F}_p)$. He chooses an integer k to be his random element and computes

$$C_1 = kP \qquad \text{and} \qquad C_2 = M + kQ_A.$$

He sends the two points (C_1, C_2) to Alice. Then Alice computes

$$C_2 - n_A C_1 = (M + kQ_A) - n_A(kP) = M + k(n_A P) - n_A(kP) = M.$$

Thus, Alice recovers the plaintext M The elliptic Elgamal public key cryptosystem is summarized in Table 4.6 below.

Public parameter creation	
A trusted party chooses and publishes a (large) prime p, an elliptic curve E over $\mathbb{F}_p$, and a point P in $E(\mathbb{F}_p)$	
Alice	**Bob**
Key creation	
Choose private key n_A. Compute $Q_A = n_A P$ in $E(\mathbb{F}_p)$. Publish the public key Q_A.	
Encryption	
	Choose plaintext $M \in E(\mathbb{F}_p)$. Choose random element k. Use Alice's public key Q_A to compute $C_1 = kP \in E(\mathbb{F}_p)$ and $C_2 = MkQ_A \in E(\mathbb{F}_p)$. Send ciphertext (C_1, C_2) to Alice.
Decryption	
Compute $C_2 - n_A C_1 \in E(\mathbb{F}_p)$. This quantity is equal to M.	

Table 4.6: Elliptic Elgamal key creation, encryption, and decryption

In principle, the elliptic Elgamal cryptosystem works fine, but there are some practical difficulties.

1. There is no obvious way to attach a plaintext message to points in $E(\mathbb{F}_p)$.

2. The elliptic Elgamal cryptosystem has 4-to-1 message expansion, as compared to the 2-to-1 message expansion ratio of the classical Elgamal cryptosystem using $\mathbb{F}_p$.

The reason that elliptic Elgamal has a 4-to-1 message expansion lies in the fact that the plaintext M is a single point in $E(\mathbb{F}_p)$. By Hasse's theorem (Theorem 4.2.5) there are approximately p different points in $E(\mathbb{F}_p)$. Hence, there are only p different plaintexts. However, the ciphertext (C_1, C_2) consists of four numbers modulo p, since each point in $E(\mathbb{F}_p)$ has two coordinates.
Various methods have been proposed to solve these problems. The difficulty of associating plaintexts to points can be overcome by choosing M randomly and using it as a mask for the actual plaintext. One such method, which also decreases message expansion, is the Menezes-Vanstone variant of the elliptic Elgamal public key cryptosystem, known as the MV-Elgamal cryptosystem.
Another natural way to improve message expansion is to send only the x-coordinate of C_1 and C_2, as done in the Diffie-Hellman key exchange. Unfortunately, since Alice must compute the difference $C_2 - n_A C_1$, she needs the correct values of both the x-and y-coordinates of C_1 and C_2. Note that the points $C_2 - n_A C_1$ and $C_2 + n_A C_1$ are quite different. However, the x-coordinate of a point determines the y-coordinate up to change of sign, so Bob can send one extra bit, for example

$$\text{Extra bit } = \begin{cases} 0 & \text{if } 0 \le y < \frac{1}{2}p, \\ 1 & \text{if } \frac{1}{2}p < y < p \end{cases}$$

(see Exercise 4.3). In this way, Bob needs to send only the x-coordinates of C_1 and C_2, plus two extra bits. This idea is sometimes referred to as *point compression*.

Exercise 4.3

A shortcoming of using an elliptic curve $E(\mathbb{F}_p)$ for cryptography is the fact that it takes two coordinates to specify a point in $E(\mathbb{F}_p)$. However, as discussed briefly earlier, the second coordinate actually conveys very little additional information.

(a) Suppose that Bob wants to send Alice the value of a point $R \in E(\mathbb{F}_p)$. Explain why it suffices for Bob to send Alice the x-coordinate of $R = (x_R, y_R)$ together with the single bit

$$\beta_R = \begin{cases} 0 & \text{if } 0 \le y_R < \frac{1}{2}p, \\ 1 & \text{if } \frac{1}{2}p < y_R < p \end{cases}$$

(You may assume that Alice is able to efficiently compute square roots modulo p. This is certainly true, for example, if $p \equiv 3 \pmod 4$, then it is given by Proposition 2.5.4).

(b) Alice and Bob decide to use the prime $p = 1123$ and the elliptic curve

$$E : Y^2 = X^3 + 54X + 87.$$

Bob sends Alice the x-coordinate $x = 278$ and the bit $\beta = 0$. What point is Bob trying to convey to Alice? What about if instead Bob sent $\beta = 1$?

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

O

P

Q

R

S

T

U

W