Democratic and Popular Republic of Algeria
Ministry of Higher Education
and Scientific Research
Yahia Fares University of Medea
Faculty of Sciences
Department of Mathematics and Computer Science

MEMORY

presented for the purpose of obtaining the

MASTER'S DEGREE IN MATHEMATICS

Specialty: Combinatorial analysis, probabilities and applications

By

Reguieg El-bachir

Introduction To Elliptic Curve Cryptography

Publicly supported on ....., before the jury composed of

L.Bouzarra	*M.C.B.* **Univ. Medea**	**President**
M.Krimi	*M.A.A.* **Univ. Medea**	**Framer**
A.Bendadou	*M.C.B.* **Univ. Medea**	**Examiner**

Année universitaire 2024/2025

Acknowledgment

First and foremost, I would like to express my deepest gratitude to **Allah**, the Most Gracious and Most Merciful, for granting me the strength, patience, and guidance to complete this work. Without His blessings, this achievement would not have been possible.

I would also like to extend my sincere thanks to my supervisor, [**M.Krimi**], for their valuable guidance, continuous support, and encouragement throughout my research, in addition on the profissors [**A.Bendadou**] and [**L.Bouzarra**]. Their insights and expertise were crucial to the development of this thesis.

My appreciation goes to the faculty and staff of [**Yahia Farès University of Medea/Department of Mathematics and Computer Science**], whose efforts and dedication provided me with a solid foundation and a supportive environment.

To my beloved family — especially my parents and siblings — thank you for your unconditional love, prayers, and constant encouragement. Your belief in me has been a great source of motivation.

Finally, I am grateful to my friends and colleagues who shared this academic journey with me and supported me along the way. May Allah reward everyone who contributed to this achievement.

Dedications

"I dedicate this modest work."

Abstract

Elliptic Curve Cryptography (ECC) has emerged as a cornerstone of modern cryptographic systems, offering unparalleled efficiency and robust security with significantly smaller key sizes compared to traditional RSA-based schemes. This thesis provides a comprehensive introduction to ECC, beginning with its mathematical foundations—focusing on the structure of elliptic curves over finite fields and the hardness of the Elliptic Curve Discrete Logarithm Problem (ECDLP). We explore core cryptographic protocols, including Elliptic Curve Diffie-Hellman (ECDH) for secure key exchange and the Elliptic Curve Digital Signature Algorithm (ECDSA), underpinning blockchain technologies like Bitcoin.

A critical analysis of ECC's security is presented, examining known attacks (e.g., MOV, Pollard's Rho) and mitigation strategies, alongside performance benchmarks demonstrating ECC's superiority in resource-constrained environments (e.g., IoT devices). The discussion extends to standardized curves (NIST P-256, Curve25519) and optimizations such as projective coordinates and side-channel-resistant algorithms.

Finally, we address post-quantum challenges, evaluating isogeny-based and lattice-based alternatives, and highlight emerging applications in zero-knowledge proofs and scalable blockchain systems. This work bridges theoretical rigor with practical insights, serving as a primer for researchers and practitioners navigating the evolving landscape of asymmetric cryptography.

CONTENTS

Abstract iii

1 Generalities and basic definitions 1

1.1 An overview of the groups, rings and fields 1
1.2 Divisibility and greatest common divisors 4
1.3 Modular arithmetic 6
1.4 Prime numbers, unique factorization, and finite fields 10
1.5 Powers and primitive roots in finite fields 12

2 Introduction to Public Key Cryptography 16

2.1 Cryptography basics 16
2.2 Public-key cryptography 18
2.3 The discrete logarithm problem 21
2.4 Diffie–Hellman key exchange 24
2.5 The ElGamal public key cryptosystem 26

3 Elliptic Curves and Cryptography 29

3.1 Elliptic curves 29
3.2 Elliptic curves over finite fields 36
3.3 The elliptic curve discrete logarithm problem (ECDLP) 39
3.4 Elliptic curve cryptography 42
3.5 Elliptic Curve Digital Signature Algorithm (ECDSA) 45

4 Applications 47

4.1 ECC in Bitcoin 47

List of Figures

2.1 Basic communications model. 16
2.2 Symmetric-key versus public-key cryptography. 18
2.3 Public-key cryptography 19
2.4 Powers $627^i \bmod 941$ for $i = 1, 2, 3, \ldots$ 23

3.1 Two examples of elliptic curves 30
3.2 The addition law on an elliptic curve 31
3.3 Adding a point P to itself 32
3.4 The vertical line L through $P = (a, b)$ and $P' = (a, -b)$. 34

4.1 An elliptic curve 47
4.2 Elliptic curve cryptography: visualizing an elliptic curve over $F(p)$, with $p = 17$ 48
4.3 Elliptic curve cryptography: Visualizing the multiplication of a point G by an integer k on an elliptic curve 51

List of Tables

1.1 Addition and multiplication tables modulo 5 8
1.2 The unit groups $(\mathbb{Z}/24\mathbb{Z})^*$ and $(\mathbb{Z}/7\mathbb{Z})^*$ 9
1.3 Powers of numbers modulo 7 12

2.1 Powers and discrete logarithms for $g = 627$ modulo $p = 941$ 23
2.2 Diffie–Hellman key exchange 25
2.3 ElGamal key creation, encryption, and decryption 27

3.1 Addition table for $E : Y^2 = X^3 + 3X + 8$ over $\mathbb{F}_{13}$ 38
3.2 Number of points and trace of Frobenius for $E : Y^2 = X^3 + 4X + 6$ 39
3.3 Diffie–Hellman key exchange 42
3.4 ECC-based ElGamal key creation, encryption, and decryption 44

Chapter 1

Generalities and basic definitions

1.1 An overview of the groups, rings and fields

1.1.1 Groups

Definition 1.1: Group

A group $(G,*)$ consists of a set G with a binary operation $*$ on G satisfying the following three axioms:

1. **Associativity:** The group operation is associative. That is,
$$a*(b*c)=(a*b)*c \quad \text{for all } a,b,c\in G.$$
2. **Identity element:** There is an element $1\in G$, called the identity element, such that
$$a*1=1*a=a \quad \text{for all } a\in G.$$
3. **Inverses:** For each $a\in G$ there exists an element $a^{-1}\in G$, called the inverse of a, such that
$$a*a^{-1}=a^{-1}*a=1.$$

A group G is *abelian* (or *commutative*) if, furthermore,

$$a*b=b*a \quad \text{for all } a,b\in G.$$

[5]

Definition 1.2

A group G is *finite* if $|G|$ is finite. The number of elements in a finite group is called its *order*.
[5]

Example 1. (a) The set of integers $\mathbb{Z}$ with the operation of addition forms a group. The identity element is 0, and the inverse of an integer a is the integer $-a$.

(b) The set $\mathbb{Z}_n$, with the operation of addition modulo n, forms a group of order n.

The set $\mathbb{Z}_n$ with the operation of multiplication modulo n is not a group, since not all elements have multiplicative inverses. However, the set $\mathbb{Z}_n^*$ is a group of order $\varphi(n)$ under the operation of multiplication modulo n, with identity element 1.
[5]

Definition 1.3

A group G is *cyclic* if there exists an element $a \in G$ such that for each $b \in G$, there is an integer i with

$$b = a^i.$$

Such an element a is called a *generator* of G.
[5]

1.1.2 Rings

Definition 1.4: Ring

A *ring* $(R, +, \times)$ consists of a set R with two binary operations, arbitrarily denoted $+$ (addition) and $\times$ (multiplication), on R, satisfying the following axioms:

1. $(R, +)$ is an abelian group with identity denoted 0.
2. The operation $\times$ is associative. That is, $a \times (b \times c) = (a \times b) \times c$ for all $a, b, c \in R$.
3. There is a multiplicative identity denoted 1, with $1 \neq 0$, such that $1 \times a = a \times 1 = a$ for all $a \in R$.
4. The operation $\times$ is distributive over $+$. That is,

$$a \times (b + c) = (a \times b) + (a \times c) \quad \text{and} \quad (b + c) \times a = (b \times a) + (c \times a)$$

for all $a, b, c \in R$.

A ring is a *commutative ring* if $a \times b = b \times a$ for all $a, b \in R$.
[5]

Example 2. (a) The set of integers $\mathbb{Z}$ with the usual operations of addition and multiplication is a commutative ring.

(b) The set $\mathbb{Z}_n$ with addition and multiplication performed modulo n is a commutative ring.
[5]

Definition 1.5

An element a of a ring R is called a *unit* or an *invertible element* if there is an element $b \in R$ such that $ab = ba = 1$.
[5]

The set of units in a ring R forms a group under multiplication, called the *group of units of* R.

Example 3. The group of units of the ring $\mathbb{Z}_n$ is $\mathbb{Z}_n^*$.
[5]

1.1.3 Fields

Definition 1.6

A *field* is a commutative ring in which all non-zero elements have multiplicative inverses.
[5]

Definition 1.7

The *characteristic* of a field is 0 if

$$\underbrace{1+1+\cdots+1}_{m \text{ times}} \neq 0 \quad \text{for any } m > 1,$$

where the sum has m terms. Otherwise, the characteristic of the field is the least positive integer m such that

$$\underbrace{1+1+\cdots+1}_{m \text{ times}} = 0.$$

[5]

Finite field

Definition 1.8

A *finite field* is a field F which contains a finite number of elements. The *order* of F is the number of elements in F.
[5]

If $\mathbb{F}_q$ is a finite field of order $q = p^m$, where p is a prime, then the characteristic of $\mathbb{F}_q$ is p. Moreover, $\mathbb{F}_q$ contains a copy of $\mathbb{Z}_p$ as a subfield. Hence, $\mathbb{F}_q$ can be viewed as an extension field of $\mathbb{Z}_p$ of degree m.

Definition 1.9

The non-zero elements of $\mathbb{F}_q$ form a group under multiplication called the *multiplicative group* of $\mathbb{F}_q$, denoted by $\mathbb{F}_q^*$.
[5]

$\mathbb{F}_q^*$ is a cyclic group of order $q-1$. Hence $a^q = a$ for all $a \in \mathbb{F}_q$.

Definition 1.10

A generator of the cyclic group $\mathbb{F}_q^*$ is called a *primitive element* or *generator* of $\mathbb{F}_q$.
[5]

If $a, b \in \mathbb{F}_q$, a finite field of characteristic p, then

$$(a+b)^p = a^p + b^p \quad \text{for all } t \geq 0.$$

1.2 Divisibility and greatest common divisors

Definition 1.11

Let a and b be integers with $b \neq 0$. We say that b divides a, or that a is divisible by b, if there is an integer c such that $a = bc$. We write $b \mid a$ to indicate that b divides a. If b does not divide a, then we write $b \nmid a$.
[4]

Example 4. We have $847 \mid 485331$, since $485331 = 847 \cdot 573$. On the other hand, $355 \nmid 259943$, since when we try to divide 259943 by 355, we get a remainder of 83. More precisely,

$$259943 = 355 \cdot 732 + 83,$$

so 259943 is not an exact multiple of 355.
[4]

Remark 1.1. Notice that every integer is divisible by 1. The integers that are divisible by 2 are the even integers, and the integers that are not divisible by 2 are the odd integers.
[4]

Proposition 1.1. 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 $a \mid (b-c)$.

[4]

Proof. [4] *page 11* □

Definition 1.12

A common divisor of two integers a and b is a positive integer d that divides both of them. The greatest common divisor of a and b is, as its name suggests, the largest positive integer d such that $d \mid a$ and $d \mid b$. The greatest common divisor of a and b is denoted $\gcd(a, b)$.
[4]

Definition 1.13: Division Algorithm

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

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

The values of q and r are uniquely determined by a and b.
[4]

Theorem 1.1 – 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},
$$r_{i-1} = r_i \cdot q_i + r_{i+1} \quad \text{with} \quad 0 \leq r_{i+1} < r_i.$$
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 $2\log_2(b)+1$ times.
[4]

Proof. [4] *page 13-15* □

Remark 1.2. We proved that the Euclidean algorithm applied to a and b with $a \geq b$ requires no more than $2\log_2(b)+1$ iterations to compute $\gcd(a,b)$.
[4]

Theorem 1.2 – Extended Euclidean Algorithm. Let a and b be positive integers. Then the equation
$$a\,u + b\,v = \gcd(a,b)$$
, always has a solution in integers u and v.
[4]

Proof. [4] *page 16-17* □

Example 5. (a) The following algorithm computes the greatest common divisor g of the positive integers a and b, together with a solution (u,v) in integers to the equation $a\,u + b\,v = \gcd(a,b)$.

1. Set $u = 1$, $g = a$, $x = 0$, and $y = b$
2. If $y = 0$, set $v = \dfrac{g - a\,u}{b}$ and return the values (g,u,v)
3. Divide g by y with remainder, $g = q\,y + t$, with $0 \leq t < y$
4. Set $s = u - q\,x$
5. Set $u = x$ and $g = y$
6. Set $x = s$ and $y = t$
7. Go to Step (2)

(b) We use the above algorithm on a computer using the computer language of your choice.

(c) We use the program to compute $g = \gcd(a, b)$ and integer solutions to the equation $au + bv = g$ for the following pairs (a, b):

(i) $(527, 1258)$

(ii) $(228, 1056)$

(iii) $(163961, 167181)$

(iv) $(3892394, 239847)$

(d) What happens to program if $b = 0$? Fix the program so that it deals with this case correctly.

(e) It is often useful to have a solution with $u > 0$. Modify program so that it returns a solution with $u > 0$ and u as small as possible. *[Hint: If (u, v) is a solution, then so is $\left(u + \frac{b}{g},\ v - \frac{a}{g}\right)$.]* Redo part (c) we using modified program.
[4]

1.3 Modular arithmetic

Definition 1.14

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. We write

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

to indicate that a and b are congruent modulo m. The number m is called the *modulus*.
[4]

Example 6. We have

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

On the other hand,

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

Notice that the numbers satisfying

$$a \equiv 0 \pmod{m}$$

are the numbers that are divisible by m, i.e., the multiples of m.
[4]

The reason that congruence notation is so useful is that congruences behave much like equalities, as the following proposition indicates.

Proposition 1.2. Let $m \geq 1$ be an integer.

1. If $a_1 \equiv a_2 \pmod{m}$ and $b_1 \equiv b_2 \pmod{m}$, then

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

2. Let a be an integer. Then
$$a \cdot b \equiv 1 \pmod{m}$$
for some integer b if and only if **Greatest Common Divisor** $(\gcd)(a,m) = 1$.

   If such an integer b exists, then we say that b is the *(multiplicative) inverse* of a modulo m.

[4]

Proof. [4] *page 20* □

Proposition 1.1 (2) says that if $\gcd(a,m) = 1$, then there exists an inverse b of a modulo m. This has the curious consequence that the fraction
$$b^{-1} = \frac{1}{b}$$
then has a meaningful interpretation in the world of integers modulo m.

Example 7. We take $m = 5$ and $a = 2$. Clearly $\gcd(2,5) = 1$, so there exists an inverse to 2 modulo 5. The inverse of 2 modulo 5 is 3, since
$$2 \cdot 3 \equiv 1 \pmod{5}, \quad \text{so} \quad 2^{-1} \equiv 3 \pmod{5}.$$

Similarly, $\gcd(4,15) = 1$, so 4^{-1} exists modulo 15. In fact, $4 \cdot 4 \equiv 1 \pmod{15}$, so 4 is its own inverse modulo 15.

We can even work with fractions $\frac{a}{d}$ modulo m as long as the denominator is relatively prime to m. For example, we can compute $\frac{5}{7} \mod 11$ by first observing that $7 \cdot 8 \equiv 1 \pmod{11}$, so $7^{-1} \equiv 8 \pmod{11}$. Then
$$\frac{5}{7} = 5 \cdot 7^{-1} \equiv 5 \cdot 8 \equiv 40 \equiv 7 \pmod{11}.$$

[4]

Remark 1.3. In the preceding examples it was easy to find inverses modulo m by trial and error. However, when m is large, it is more challenging to compute $a^{-1} \mod m$. Note that we showed that inverses exist by using the extended Euclidean algorithm 5.

In order to actually compute the u and v that appear in the equation
$$a\,u + m\,v = \gcd(a,m),$$
we can apply the Euclidean algorithm directly.

In any case, since the Euclidean algorithm takes only $2\log_2(b)+3$ iterations to compute $\gcd(a,b)$, it takes only a small multiple of $\log_2(m)$ steps to compute $a^{-1} \mod m$.

We now continue our development of the theory of modular arithmetic. If a divided by m has quotient q and remainder r, it can be written as
$$a = m \cdot q + r \quad \text{with} \quad 0 \le r < m, \quad a, m \in \mathbb{Z}.$$
This shows that
$$a \equiv r \pmod{m}$$
for some integer r between 0 and $m-1$. So if we want to work with integers modulo m, it is enough to use the integers
$$0 \le r < m.$$
This prompts the following definition. [4]

Definition 1.15

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 an addition or multiplication in $\mathbb{Z}/m\mathbb{Z}$, we always divide the result by m and take the remainder in order to obtain an element in $\mathbb{Z}/m\mathbb{Z}$.
[4]

Table 1.1 illus trates the ring $\mathbb{Z}/5\mathbb{Z}$ by giving complete addition and multiplication tables modulo 5.

Remark 1.4. If you have studied ring theory, you will recognize that $\mathbb{Z}/m\mathbb{Z}$ is the *quotient ring* of $\mathbb{Z}$ by the principal ideal $m\mathbb{Z}$, and that the numbers $0, 1, \ldots, m-1$ are actually *coset representatives* for the congruence classes that comprise the elements of $\mathbb{Z}/m\mathbb{Z}$. For a discussion of congruence classes and general quotient rings.

Addition Table 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

Multiplication Table Modulo 5

·	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

Table 1.1: Addition and multiplication tables modulo 5

[4]

Definition 1.16

Proposition1.2(2) tells us 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

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

The set $(\mathbb{Z}/m\mathbb{Z})^*$ is called the *group of units modulo* m.
[4]

Example 8. he 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 illustrated in Table 1.2.
[4]

Example 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 illustrated in Table 1.2.

In many of the cryptosystems that we will study, it is important to know how many elements are in the unit group modulo m. This quantity is sufficiently ubiquitous that we give it a name.

[4]

Definition 1.17

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

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

[4]

For example, we see from Examples 7 and 8 that $\varphi(24) = 8$ and $\varphi(7) = 6$.

·	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 group 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 group modulo 7

Table 1.2: The unit groups $(\mathbb{Z}/24\mathbb{Z})^*$ and $(\mathbb{Z}/7\mathbb{Z})^*$
[4]

1.4 Prime numbers, unique factorization, and finite fields

In Section 1.3 we studied modular arithmetic and saw that it makes sense to add, subtract, and multiply integers modulo m. Division, however, can be problematic, since we can divide by a in $\mathbb{Z}/m\mathbb{Z}$ only if $\gcd(a,m)=1$. But notice that if the integer m is a prime, then we can divide by every nonzero element of $\mathbb{Z}/m\mathbb{Z}$. We start with a brief discussion of prime numbers before returning to the ring $\mathbb{Z}/p\mathbb{Z}$ with p prime.

Definition 1.18

An integer p is called a **prime** if $p \geq 2$ and if the only positive integers dividing p are 1 and p.
[4]

For example, the first ten primes are

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29,$$

while the hundred thousandth prime is 1299709 and the millionth is 15485863. There are infinitely many primes, a fact that was known in ancient Greece and appears as a theorem in *Euclid's Elements*.

A prime p is defined in terms of the numbers that divide p. So the following proposition, which describes a useful property of numbers that are divisible by p, is not obvious and needs to be carefully proved. Notice that the proposition is false for composite numbers. For example, 6 divides $3 \cdot 10$, but 6 divides neither 3 nor 10.

Proposition 1.3. 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 or b.

More generally, if

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

then p divides at least one of the individual a_i.
[4]

Proof. [4] *page 27* □

Theorem 1.3 – The Fundamental Theorem of Arithmetic. Let $a \geq 2$ be an integer. Then a can be factored as a product of prime numbers:

$$a = p_1^{e_1} \cdot p_2^{e_2} \cdot p_3^{e_3} \cdots p_r^{e_r},$$

Further, other than rearranging the order of the primes, this factorization into prime powers is unique.
[4]

Proof. [4] *page 27-28* □

Definition 1.19

The Fundamental Theorem of Arithmetic says that in the factorization of a 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.)
[4]

For example, the factorization of 1728 is $1728 = 2^6 \cdot 3^3$, so

$$\mathrm{ord}_2(1728)=6, \quad \mathrm{ord}_3(1728)=3, \quad \text{and} \quad \mathrm{ord}_p(1728)=0 \text{ for all primes } p \geq 5.$$

Using the ord_p notation, the factorization of a can be succinctly written as

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

Proposition 1.4. Let p be a prime. Then every nonzero element $a \in \mathbb{Z}/p\mathbb{Z}$ has a multiplicative inverse, that is, there exists a number b such that

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

We denote this value of b by $a^{-1} \bmod p$, or if p has already been specified, then simply by a^{-1}.
[4]

Proof. [4] *page 29* □

Remark 1.5. The extended Euclidean algorithm gives us an efficient computational method for computing $a^{-1} \bmod p$. We simply solve the equation

$$au + pv = 1$$

in integers u and v, and then $u = a^{-1} \bmod p$. For an alternative method of computing $a^{-1} \bmod p$, see Remark 1.27.

Proposition 1.4 can be restated by saying that if p is prime, then

$$(\mathbb{Z}/p\mathbb{Z})^* = \{1,2,3,4,\ldots,p-1\}.$$

In other words, when the 0 element is removed from $\mathbb{Z}/p\mathbb{Z}$, the remaining elements are units and closed under multiplication.
[4]

Definition 1.20

Let p be a prime number. Then the set of integers modulo p, denoted by

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

equipped with addition, subtraction, multiplication, and division (modulo p), forms a **field**.
You may already be familiar with other examples of fields, such as:

- The field of real numbers: $\mathbb{R}$,
- The field of rational numbers (fractions): $\mathbb{Q}$,
- The field of complex numbers: $\mathbb{C}$.

The field $\mathbb{Z}/p\mathbb{Z}$ has exactly p elements and is therefore a **finite field**. It is commonly denoted by $\mathbb{F}_p$, so:

$$\mathbb{F}_p \cong \mathbb{Z}/p\mathbb{Z}.$$

The set of nonzero elements of $\mathbb{F}_p$ forms a multiplicative group, called the **group of units**, and is denoted by:

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

Finite fields, especially the fields $\mathbb{F}_p$, play a central role in cryptography and are foundational throughout many areas of mathematics.
[4]

Remark 1.6. Although $\mathbb{Z}/p\mathbb{Z}$ and $\mathbb{F}_p$ are used to denote the same concept, equality of elements is expressed somewhat differently in the two settings. For $a, b \in \mathbb{F}_p$, the equality of a and b is denoted by $a = b$, while for $a, b \in \mathbb{Z}/p\mathbb{Z}$, the equality of a and b is denoted by equivalence modulo p, i.e., $a \equiv b \pmod{p}$.
[4]

1.5 Powers and primitive roots in finite fields

We begin with a simple example. Table 1.9 lists the powers of $1, 2, 3, \ldots, 6$ modulo the prime 7.

a^n	1	2	3	4	5	6
1	1	1	1	1	1	1
2	2	4	1	2	4	1
3	3	2	6	4	5	1
4	4	2	1	4	2	1
5	5	4	6	2	3	1
6	6	1	6	1	6	1

Table 1.3: Powers of numbers modulo 7
[4]

There are quite a few interesting patterns visible in Table 2.2, including in particular the fact that the right-hand column consists entirely of ones. We can restate this observation by saying that

$$a^6 \equiv 1 \pmod 7 \quad \text{for every} \quad a = 1,2,3,\ldots,6.$$

Of course, this cannot be true for all values of a, since if a is a multiple of 7, then so are all of its powers, so in that case

$$a^n \equiv 0 \pmod 7.$$

On the other hand, if a is not divisible by 7, then a is congruent to one of the values $1,2,3,\ldots,6$ modulo 7. Hence

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

Further experiments with other primes suggest that this example reflects a general fact.

Theorem 1.4 – Fermat's Little Theorem. Let p be a prime number and let a be any integer. Then

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

[4]

Proof. [4] *page 30-31* □

Example 10. The number $p = 15485863$ is prime, so Fermat's little theorem (Theorem 1.25) tells us that

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

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

[4]

Remark 1.7. Fermat's little theorem and the fast powering algorithm provide us with a reasonably efficient method of computing inverses modulo p, namely

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

This congruence is true because if we multiply a^{p-2} by a, then Fermat's theorem tells us that the product is equal to 1 modulo p. This gives an alternative to the extended Euclidean algorithm method described in Remark 1.3. In practice, the two algorithms tend to take about the same amount of time.

[4]

Example 11. 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

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

[4]

Example 12. Consider the number $m = 15485207$. Using the powering algorithm, it is not hard to compute (on a computer)

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

We did not get the value 1, so it seems that Fermat's little theorem is not true for m. What does that tell us? If m were prime, then Fermat's little theorem says that we would have obtained 1. Hence the fact that we did not get 1 proves that the number $m = 15485207$ is not prime.

Think about this for a minute, because it's actually a bit astonishing. By a simple computation, we have conclusively proven that m is not prime, yet we do not know any of its factors.

Fermat's little theorem tells us that if a is an integer not divisible by p, then

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

However, for any particular value of a, there may well be smaller powers of a that are congruent to 1. We define the order of a modulo p to be the smallest exponent $k \geq 1$ such that

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

[4]

Proposition 1.5. 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 $p-1$.

[4]

Proof. [4] *page 33* □

Theorem 1.5 – Primitive Root Theorem. Let p be a prime number. Then there exists an element $g \in \mathbf{F}_p^*$ whose powers give every element of $\mathbf{F}_p^*$, i.e.,

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

Elements with this property are called *primitive roots* of $\mathbf{F}_p$ or *generators* of $\mathbf{F}_p^*$. They are the elements of $\mathbf{F}_p^*$ having order $p-1$.

[4]

Proof. [4] *page 33* □

Example 13. The field $\mathbf{F}_{11}$ has 2 as a primitive root, since in $\mathbf{F}_{11}$,

$$2^0 \equiv 1, \quad 2^1 \equiv 2, \quad 2^2 \equiv 4, \quad 2^3 \equiv 8, \quad 2^4 \equiv 5,$$

$$2^5 \equiv 10, \quad 2^6 \equiv 9, \quad 2^7 \equiv 7, \quad 2^8 \equiv 3, \quad 2^9 \equiv 6 \pmod{11}.$$

Thus all 10 nonzero elements of $\mathbf{F}_{11}$ have been generated as powers of 2.

On the other hand, 2 is not a primitive root for $\mathbf{F}_{17}$, since in $\mathbf{F}_{17}$,

$$2^0 \equiv 1, \quad 2^1 \equiv 2, \quad 2^2 \equiv 4, \quad 2^3 \equiv 8, \quad 2^4 \equiv 16,$$

$$2^5 \equiv 15, \quad 2^6 \equiv 13, \quad 2^7 \equiv 9, \quad 2^8 \equiv 1 \pmod{17},$$

so we get back to 1 before obtaining all 16 nonzero values modulo 17.

However, it turns out that 3 is a primitive root for 17, since in $\mathbf{F}_{17}$,

$$3^0 \equiv 1, \quad 3^1 \equiv 3, \quad 3^2 \equiv 9, \quad 3^3 \equiv 10, \quad 3^4 \equiv 13, \quad 3^5 \equiv 5,$$

$$3^6 \equiv 15, \quad 3^7 \equiv 11, \quad 3^8 \equiv 16, \quad 3^9 \equiv 14, \quad 3^{10} \equiv 8, \quad 3^{11} \equiv 7,$$

$$3^{12} \equiv 4, \quad 3^{13} \equiv 12, \quad 3^{14} \equiv 2, \quad 3^{15} \equiv 6 \pmod{17}.$$

[4]

Remark 1.8. If p is large, then the finite field F_p has quite a few primitive roots. The precise formula says that F_p has exactly $\varphi(p-1)$ primitive roots, where φ is Euler's phi function. For example, you can check that the following is a complete list of the primitive roots for F_{29}:

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

This agrees with the value $\varphi(28) = 12$. More generally, if k divides $p-1$, then there are exactly $\varphi(k)$ elements of F_p^* having order k.

[4]

Chapter 2
Introduction to Public Key Cryptography

2.1 Cryptography basics

Cryptography is about the design and analysis of mathematical techniques that enable secure communications in the presence of malicious adversaries.

Basic communications model In Figure 2.1, entities A (Alice) and B (Bob) are communicating over an unsecured channel.We assume that all communications take place in the presence of an adversary E (Eve) whose objective is to defeat any security services being provided to A and B. [3]

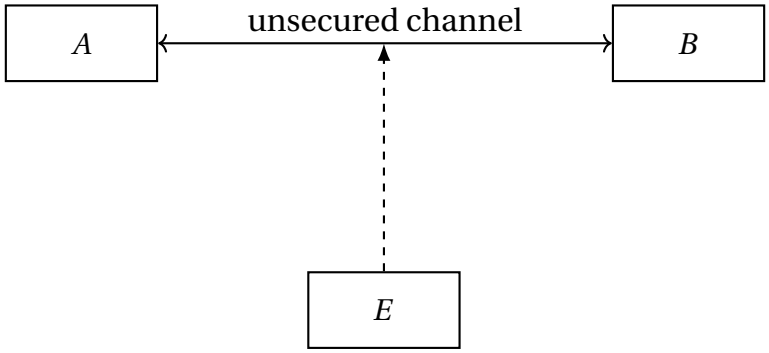


Figure 2.1: Basic communications model.
[3]

Security goals Careful examination of the scenarios outlined above reveals the following fundamental objectives of secure communications:

1. Confidentiality: keeping data secret from all but those authorized to see it—messages sent by A to B should not be readable by E.

2. Data integrity: ensuring that data has not been altered by unauthorized means—B should be able to detect when data sent by A has been modified by E.

3. Data origin authentication: corroborating the source of data—B should be able to verify that data purportedly sent by A indeed originated with A.

4. Entity authentication: corroborating the identity of an entity—B should be convinced of the identity of the other communicating entity.

5. Non-repudiation: preventing an entity from denying previous commitments or actions—when B receives a message purportedly from A, not only is B convinced that the message originated with A, but B can convince a neutral third party of this; thus A cannot deny having sent the message to B.

Some applications may have other security objectives such as anonymity of the communicating entities or access control (the restriction of access to resources). [3]

Symmetric-key cryptography Cryptographic systems can be broadly divided into two kinds. In *symmetric-key* schemes, depicted in Figure2.2, returning to Bob and Alice, we observe that they must share knowledge of the secret key k. Using that secret key, they can both encrypt and decrypt messages, so Bob and Alice have equal (or symmetric) knowledge and abilities. For this reason, ciphers of this sort are known as *symmetric ciphers.*
[3]

Mathematically, a symmetric cipher uses a key k chosen from a space (i.e., a set) of possible keys $\mathscr{K}$ to encrypt a plaintext message m chosen from a space of possible messages $\mathscr{M}$, and the result of the encryption process is a ciphertext c belonging to a space of possible ciphertexts $\mathscr{C}$. Thus, encryption may be viewed as a function

$$e : \mathscr{K} \times \mathscr{M} \to \mathscr{C}$$

whose domain $\mathscr{K} \times \mathscr{M}$ is the set of pairs (k, m) consisting of a key k and a plaintext m, and whose range is the space of ciphertexts $\mathscr{C}$.

Similarly, decryption is a function

$$d : \mathscr{K} \times \mathscr{C} \to \mathscr{M}.$$

Of course, we want the decryption function to "undo" the results of the encryption function. Mathematically, this is expressed by the formula

$$d(k, e(k, m)) = m \quad \text{for all } k \in \mathscr{K} \text{ and all } m \in \mathscr{M}.$$

It is sometimes convenient to write the dependence on k as a subscript. Then for each key k, we get a pair of functions

$$e_k : \mathscr{M} \to \mathscr{C} \quad \text{and} \quad d_k : \mathscr{C} \to \mathscr{M}$$

satisfying the decryption property

$$d_k(e_k(m)) = m \quad \text{for all } m \in \mathscr{M}.$$

In other words, for every key k, the function d_k is the inverse function of the function e_k. In particular, this means that e_k must be one-to-one, since if

$$e_k(m) = e_k(m'),$$

then

$$m = d_k(e_k(m)) = d_k(e_k(m')) = m'.$$

[4]

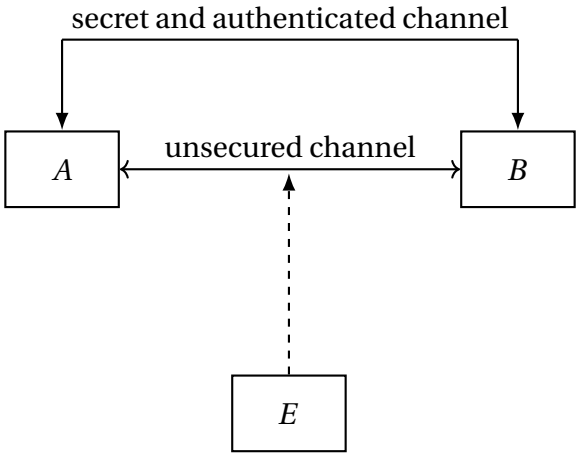


Figure 2.2: Symmetric-key versus public-key cryptography.
[3]

2.2 Public-key cryptography

The notion of public-key cryptography, depicted in Figure 2.3, was introduced in 1975 by Diffie, Hellman and Merkle to address the aforementioned shortcomings of symmetric-key cryptography. In contrast to symmetric-key schemes, public-key schemes require only that the communicating entities exchange keying material that is authentic (but not secret). Each entity selects a single key pair (e,d) consisting of a public key "e", and a related private key "d" (that the entity keeps secret). The keys have the property that it is computationally infeasible to determine the private key solely from knowledge of the public key.

In a public-key cryptographic scheme, a key pair is selected so that the problem of deriving the private key from the corresponding public key is equivalent to solving a computational problem that is believed to be intractable. Number-theoretic problems whose intractability form the basis for the security of commonly used public-key schemes are:

1. The integer factorization problem, whose hardness is essential for the security of RSA public-key encryption and signature schemes.

2. The discrete logarithm problem, whose hardness is essential for the security of the ElGamal public-key encryption and signature schemes and their variants such as the Digital Signature Algorithm (DSA).

3. The elliptic curve discrete logarithm problem, whose hardness is essential for the security of all elliptic curve cryptographic schemes.

[3]

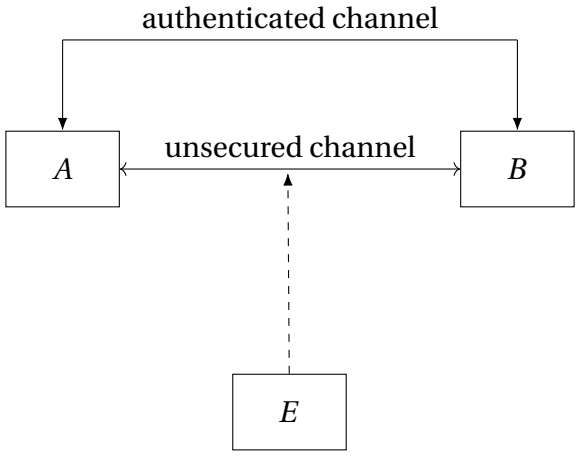


Figure 2.3: Public-key cryptography

2.2.1 RSA systems

RSA, named after its inventors Rivest, Shamir, and Adleman, was proposed in 1977 shortly after the discovery of public-key cryptography.

RSA key generation

An RSA key pair can be generated using Algorithm 1. The public key consists of a pair of integers (n, e) where the RSA modulus n is a product of two randomly generated (and secret) primes p and q of the same bitlength. The encryption exponent e is an integer satisfying $1 < e < \varphi$ and $\gcd(e, \varphi) = 1$ where $\varphi = (p-1)(q-1)$. The private key d, also called the *decryption exponent*, is the integer satisfying $1 < d < \varphi$ and $ed \equiv 1 \pmod{\varphi}$.

It has been proven that the problem of determining the private key d from the public key (n, e) is computationally equivalent to the problem of determining the factors p and q of n; the latter is the *integer factorization problem* (IFP). [3]

Algorithm 1 RSA Key Pair Generation

Require: Security parameter l
Ensure: RSA public key (n, e) and private key d
1: Randomly select two primes p and q of the same bitlength $l/2$
2: Compute $n \leftarrow pq$ and $\varphi \leftarrow (p-1)(q-1)$
3: Select an arbitrary integer e such that $1 < e < \varphi$ and $\gcd(e, \varphi) = 1$
4: Compute d such that $1 < d < \varphi$ and $ed \equiv 1 \pmod{\varphi}$
5: **return** (n, e, d)

[3]

RSA encryption scheme

RSA encryption and signature schemes use the fact that

$$m^{ed} \equiv m \pmod{n} \tag{2.1}$$

for all integers m. The encryption and decryption procedures for the (basic) RSA public-key encryption scheme are presented as Algorithms 2 and 3. Decryption works

because $c^d \equiv (m^e)^d \equiv m \pmod{n}$, as derived from expression (3.1). The security relies on the difficulty of computing the plaintext m from the ciphertext $c = m^e \bmod n$ and the public parameters n and e. This is the problem of finding eth roots modulo n and is assumed (but has not been proven) to be as difficult as the integer factorization problem. [3]

Algorithm 2 Basic RSA Encryption

Require: RSA public key (n, e), plaintext $m \in [0, n-1]$
Ensure: Ciphertext c
1: Compute $c \leftarrow m^e \bmod n$
2: **return** c
[3]

Algorithm 3 Basic RSA Decryption

Require: RSA public key (n, e), RSA private key d, ciphertext c
Ensure: Plaintext m
1: Compute $m \leftarrow c^d \bmod n$
2: **return** m
[3]

Hash function

Hash functions are used to create fixed-length digests of arbitrarily-long input strings. Hash functions are keyless, and they provide the data integrity service. They are usually built using iterated and dedicated hash function construction techniques.

Various families of hash functions are available, such as MD, SHA-1, SHA-2, SHA-3, RIPEMD, and Whirlpool. Hash functions are commonly used for digital signatures and Message Authentication Codes (MACs), such as HMACs. They have three security properties, namely: Preimage resistance, Second preimage resistance, and Collision resistance.

Hash functions are also typically used to provide data integrity services. These can be used both as one-way functions and to construct other cryptographic primitives, such as MACs and digital signatures. Some applications use hash functions as a means for generating Pseudo-random Number Generators (PRNGs).

There are two practical and three security properties of hash functions that must be met depending on the level of integrity required. [2]

RSA signature scheme

The RSA signing and verifying procedures are shown in Algorithms 4 and 5. The signer of a message m first computes its message digest $h = H(m)$ using a cryptographic hash function H, where h serves as a short fingerprint of m. Then, the signer uses their private key d to compute the eth root s of h modulo n: $s = h^d \bmod n$.

Note that $s^e \equiv h \pmod{n}$. The signer transmits the message m and its signature s to a verifying party. This party then recomputes the message digest $h = H(m)$, recovers a message digest $h' = s^e \bmod n$ from s, and accepts the signature as being valid for m provided that $h = h'$.

The security relies on the inability of a forger (who does not know the private key d) to compute eth roots modulo n. [3]

Algorithm 4 Basic RSA Signature Generation

Require: RSA public key (n, e), RSA private key d, message m
Ensure: Signature s
1: Compute $h \leftarrow H(m)$ where H is a hash function
2: Compute $s \leftarrow h^d \bmod n$
3: **return** s

[3]

Algorithm 5 Basic RSA Signature Verification

Require: RSA public key (n, e), message m, signature s
Ensure: Acceptance or rejection of the signature
1: Compute $h \leftarrow H(m)$
2: Compute $h' \leftarrow s^e \bmod n$
3: **if** $h = h'$ **then**
4: **return** "Accept the signature"
5: **else**
6: **return** "Reject the signature"
7: **end if**

[3]

The computationally expensive step in any RSA operation is the modular exponentiation, e.g., computing $m^e \bmod n$ in encryption and $c^d \bmod n$ in decryption. In order to increase the efficiency of encryption and signature verification, one can select a small encryption exponent e; in practice, $e = 3$ or $e = 2^{16} + 1$ is commonly chosen. The decryption exponent d is of the same bitlength as n. Thus, RSA encryption and signature verification with a small exponent e are significantly faster than RSA decryption and signature generation.

2.3 The discrete logarithm problem

The *discrete logarithm problem* is a mathematical problem that arises in many settings, including the mod p version described in this section and the elliptic curve version that will be studied later, in Chapter 3. The first published public key construction, due to Diffie and Hellman, is based on the discrete logarithm problem in a finite field $\mathbb{F}_p$, where recall that $\mathbb{F}_p$ is a field with a prime number of elements.

For convenience, we interchangeably use the notations $\mathbb{F}_p$ and $\mathbb{Z}/p\mathbb{Z}$ for this field, and we use equality notation for elements of $\mathbb{F}_p$ and congruence notation for elements of $\mathbb{Z}/p\mathbb{Z}$.

Let p be a (large) prime. In (primitive root theorem) tells us that there exists a *primitive element* g. This means that every nonzero element of $\mathbb{F}_p$ is equal to some power of g. In particular, $g^{p-1} = 1$ by Fermat's little theorem, and no smaller power of g is equal to 1. Equivalently, the list of elements

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

is a complete list of the elements in $\mathbb{F}_p^*$ in some order. [4]

Definition 2.1

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 $\log_g(h)$. [4]

Remark 2.1. An older term for the discrete logarithm is the *index*, denoted by $\text{ind}_g(h)$. The index terminology is still commonly used in number theory. It is also convenient if there is a danger of confusion between ordinary logarithms and discrete logarithms, since, for example, the quantity $\log 2$ frequently occurs in both contexts. [4]

Remark 2.2. The discrete logarithm problem is a well-posed problem, namely to find an integer exponent x such that $g^x = h$. However, if there is one solution, then there are infinitely many, because Fermat's little theorem tells us that $g^{p-1} \equiv 1 \pmod{p}$. Hence if x is a solution to $g^x = h$, then $x + k(p-1)$ is also a solution for every value of k, because

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

Thus $\log_g(h)$ is defined only up to adding or subtracting multiples of $p-1$. In other words, $\log_g(h)$ is really defined modulo $p-1$. It is not hard to verify that $\log_g$ gives a well-defined function

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

Sometimes, for concreteness, we refer to "the" discrete logarithm as the integer x lying between 0 and $p-2$ satisfying the congruence $g^x \equiv h \pmod{p}$. [4]

Remark 2.3. It is not hard to prove that

$$\log_g(ab) = \log_g(a) + log_g(b) \quad \text{for all } a, b \in \mathbb{F}_p^*.$$

[4]

Thus calling $\log_g$ a "logarithm" is reasonable, since it converts multiplication into addition in the same way as the usual logarithm function. In mathematical terminology, the discrete logarithm $\log_g$ is a group isomorphism from $\mathbb{F}_p^*$ to $\mathbb{Z}/(p-1)\mathbb{Z}$.

Example 14. The number $p = 56509$ is prime, and one can check that $g = 2$ is a primitive root modulo p. How would we go about calculating the discrete logarithm of $h = 38679$? The only method that is immediately obvious is to compute

$$2^2, 2^3, 2^4, 2^5, 2^6, 2^7, \ldots \pmod{56509}$$

until we find some power that equals 38679. It would be difficult to do this by hand, but using a computer, we find that $\log_p(h) = 11235$. You can verify this by calculating $2^{11235} \bmod 56509$ and checking that it is equal to 38679. [4]

n	$g^n \bmod p$
1	627
2	732
3	697
4	395
5	182
6	253
7	543
8	760
9	374
10	189

n	$g^n \bmod p$
11	878
12	21
13	934
14	316
15	522
16	767
17	58
18	608
19	111
20	904

h	$\log_g(h)$
1	0
2	183
3	469
4	366
5	356
6	652
7	483
8	549
9	938
10	539

h	$\log_g(h)$
11	429
12	835
13	279
14	666
15	825
16	732
17	337
18	181
19	43
20	722

Table 2.1: Powers and discrete logarithms for $g = 627$ modulo $p = 941$
[4]

Remark 2.4. It must be emphasized that the discrete logarithm bears little resemblance to the continuous logarithm defined on the real or complex numbers. The terminology is still reasonable, because in both instances the process of exponentiation is inverted—but exponentiation modulo p varies in a very irregular way with the exponent, contrary to the behavior of its continuous counterpart. The random-looking behavior of exponentiation modulo p is apparent from even a cursory glance at a table of values such as those in Table, where we list the first few powers and the first few discrete logarithms for the prime $p = 941$ and the base $g = 627$. The seeming randomness is also illustrated by the scatter graph of $627^i \bmod 941$ pictured in Figure 2.4. [4]

Remark 2.5. Our statement of the discrete logarithm problem includes the assumption that the base g is a primitive root modulo p, but this is not strictly necessary. In general, for any $g \in \mathbb{F}_p^*$ and any $h \in \mathbb{F}_p^*$, the discrete logarithm problem is the determination of an exponent x satisfying $g^x \equiv h \pmod{p}$, assuming that such an x exists. [4]

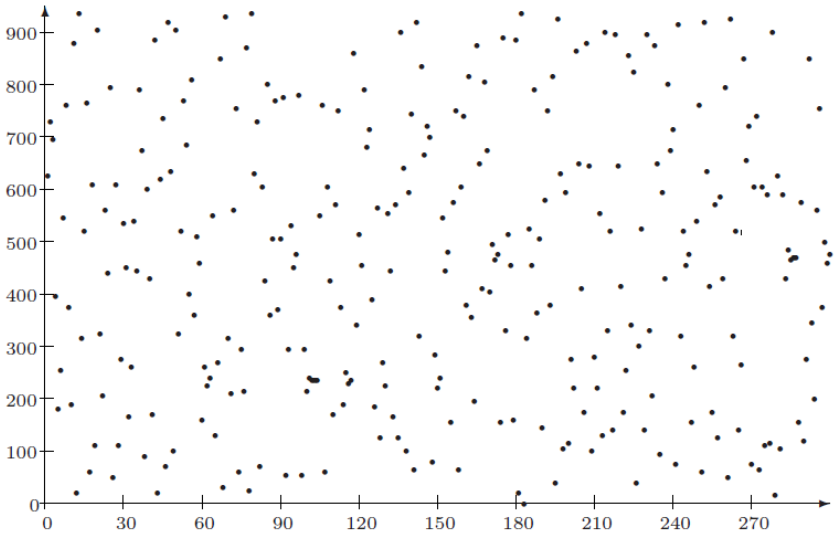


Figure 2.4: Powers $627^i \bmod 941$ for $i = 1, 2, 3, \ldots$
[4]

More generally, rather than taking nonzero elements of a finite field $\mathbb{F}_p$ and multiplying them together or raising them to powers, we can take elements of any group and use the group law instead of multiplication. This leads to the most general form of the discrete logarithm problem.

Definition 2.2

Let G be a group whose group law we denote by the symbol $\star$. The **Discrete Logarithm Problem** for G is to determine, for any two given elements g and h in G, an integer x satisfying

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

[4]

2.4 Diffie–Hellman key exchange

The **Diffie–Hellman key exchange** algorithm solves the following dilemma. Alice and Bob want to share a secret key for use in a symmetric cipher, but their only means of communication is insecure. Every piece of information that they exchange is observed by their adversary Eve. How is it possible for Alice and Bob to share a key without making it available to Eve? At first glance it appears that Alice and Bob face an impossible task. It was a brilliant insight of Diffie and Hellman that the difficulty of the discrete logarithm problem for $\mathbb{F}_p^*$ provides a possible solution.

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 knowledge; for example, they might post the values on their web sites, so Eve knows them, too. For various reasons to be discussed later, it is best if they choose g such that its order in $\mathbb{F}_p^*$ is a large prime.

The next step is for Alice to pick a secret integer a that she does not reveal to anyone, while at the same time Bob picks an integer b that he keeps secret. Bob and Alice use their secret integers to compute

$$A \equiv g^a \pmod{p} \quad \text{(Alice computes this)}$$

and

$$B \equiv g^b \pmod{p} \quad \text{(Bob computes this).}$$

They next exchange these computed values, 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 the insecure communication channel.

Finally, Bob and Alice again use their secret integers to compute

$$A' \equiv B^a \pmod{p} \quad \text{(Alice computes this)}$$

and

$$B' \equiv A^b \pmod{p} \quad \text{(Bob computes this).}$$

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

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

[4]

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

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

Table 2.2: Diffie–Hellman key exchange
[4]

Example 15. 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 channel, so both $A = 390$ and $B = 691$ should be considered 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

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

so 470 is their shared secret.

Suppose that Eve sees this entire exchange. She can reconstitute 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. As far as is known, this is the only way for Eve to find the secret shared value without Alice's or Bob's assistance.

[4]

2.5 The ElGamal public key cryptosystem

The ElGamal PKC is our first example of a public key cryptosystem, so we proceed slowly and provide all of the details. Alice begins by publishing information consisting of a public key and an algorithm. The public key is simply a number, and the algorithm is the method by which Bob encrypts his messages using Alice's public key. Alice does not disclose her private key, which is another number. The private key allows Alice, and only Alice, to decrypt messages that have been encrypted using her public key.

This is all somewhat vague and applies to any public key cryptosystem. For the ElGamal PKC, Alice needs a large prime number p for which the discrete logarithm problem in $\mathbb{F}_p^*$ is difficult, and she needs an element g modulo p of large (prime) order. She may choose p and g herself, or they may have been preselected by some trusted party such as an industry panel or government agency.

Alice chooses a secret number a to act as her private key, and she computes the quantity

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

Notice the resemblance to Diffie–Hellman key exchange. Alice publishes her public key A and she keeps her private key a secret.

Now suppose that 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. In order to encrypt m, Bob first randomly chooses another number k modulo p.

Bob uses k to encrypt one, and only one, message, and then he 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, his chosen random ephemeral key k, and Alice's public key A, and uses them to compute the two quantities:

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

(Remember that g and p are public parameters, so Bob also knows their values.)

Bob's ciphertext, i.e., his encryption of m, is the pair of numbers (c_1, c_2), which he sends to Alice.

How does Alice decrypt Bob's ciphertext (c_1, c_2)? Since Alice knows a, she can compute the quantity

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

and hence also $x^{-1} \pmod{p}$. Alice next multiplies c_2 by x^{-1}, and lo and behold, the resulting value is the plaintext m. To see why, we expand the value of $x^{-1} \cdot c_2$ and find that

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

The ElGamal public key cryptosystem is summarized in Table 2.3.

What is Eve's task in trying to decrypt the message? Eve knows the public parameters p and g, and she also knows the value of $A \equiv g^a \pmod{p}$, since Alice's public key A is public knowledge. If Eve can solve the discrete logarithm problem, she can find a and decrypt the message. Otherwise, it appears difficult for Eve to find the plaintext, although there are subtleties, some of which we'll discuss after doing an example with small numbers.

[4]

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	
Chooses private key $1 \le a \le p-1$. Computes $A = g^a \pmod{p}$. Publishes the public key A.	
Encryption	
	Chooses plaintext m. Chooses random ephemeral key k. Uses Alice's public key A to compute $c_1 = g^k \pmod{p}$ and $c_2 = mA^k \pmod{p}$. Sends 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 2.3: ElGamal key creation, encryption, and decryption

[4]

Example 16. 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 Alice the message $m = 331$. He chooses an ephemeral key at random, say $k = 197$, and computes the two quantities:

$$c_1 \equiv 2^{197} \equiv 87 \pmod{467}, \qquad c_2 \equiv 331 \cdot 224^{197} \equiv 57 \pmod{467}.$$

The pair $(c_1, c_2) = (87, 57)$ is the ciphertext that Bob sends to Alice.

Alice, knowing $a = 153$, first computes:

$$x \equiv c_1^a \equiv 87^{153} \equiv 367 \pmod{467},$$

and then the modular inverse:

$$x^{-1} \equiv 14 \pmod{467}.$$

She decrypts the message by computing:

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

which recovers Bob's original message. [4]

Finally, she computes

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

and recovers the plaintext message m.

Remark 2.6. 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 in the same range. Thus, in general, it takes *twice as many bits* to write down the ciphertext as it does to write down the plaintext. We say that ElGamal has a **2-to-1 message expansion**. [4]

Chapter 3
Elliptic Curves and Cryptography

3.1 Elliptic curves

Elliptic Curve Cryptography (ECC) is based on the discrete logarithm problem founded upon elliptic curves over finite fields (Galois fields). The main benefit of ECC over other types of public key algorithms is that it requires a smaller key size while providing the same level of security as, for example, RSA. Two notable schemes that originate from ECC are ECDH for key exchange and ECDSA for digital signatures. [2]

ECC can also be used for encryption, but it is not usually used for this purpose in practice. Instead, it is commonly used for key exchange and digital signatures. As ECC needs less space to operate, it is becoming very popular on embedded platforms and in systems where storage resources are limited. By comparison, the same level of security can be achieved with ECC using only 256-bit operands as compared to 3072-bit operands in RSA. [2]

To understand ECC, a basic introduction to the underlying mathematics is necessary. An elliptic curve is basically a type of polynomial equation known as the *Weierstrass equation*, which generates a curve over a finite field. [2]

The most commonly used field is where all the arithmetic operations are performed modulo a prime p. Elliptic curve groups consist of points on the curve over a finite field.

An elliptic curve is defined over finite field $\mathbb{F}_p$, so the coefficient A and B belong to $\mathbb{F}_p$. This should be in the Weierstrass equation:

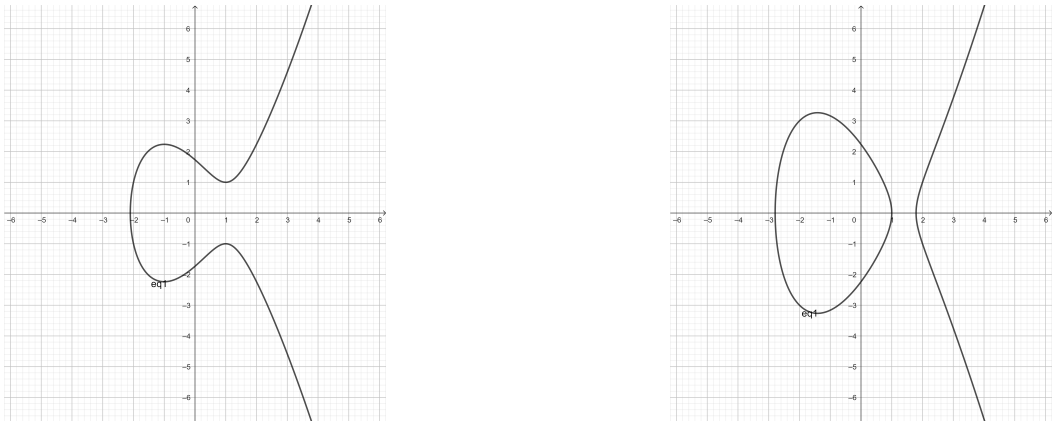
$$Y^2 = X^3 + AX + B. \quad A, B \in \mathbb{F}_p$$

Equations of this type are called *Weierstrass equations* after the mathematician who studied them extensively during the 19$^{\text{th}}$ century. Two examples of elliptic curves,

$$E_1 : Y^2 = X^3 - 3X + 3 \quad \text{and} \quad E_2 : Y^2 = X^3 - 6X + 5,$$

are illustrated in Figure 3.1.

[4]



(a) $E_1 : Y^2 = X^3 - 3X + 3$ (b) $E_2 : Y^2 = X^3 - 6X + 5$

Figure 3.1: Two examples of elliptic curves
[4]

An amazing feature of elliptic curves is that there is a natural way to take two points on an elliptic curve and "add" them to produce a third point. We put quotation marks around "add" because we are referring to an operation that combines two points in a manner analogous to addition in some respects (it is commutative and associative, and there is an identity), but very unlike addition in other ways. The most natural way to describe the "addition law" on elliptic curves is to use geometry.

Let P and Q be two points on an elliptic curve E, as illustrated in Figure 3.2. 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," although as you can see, this process is nothing like ordinary addition. For now, we denote this strange addition law by the symbol $\oplus$. Thus we write

$$P \oplus Q = R'.$$

[4]

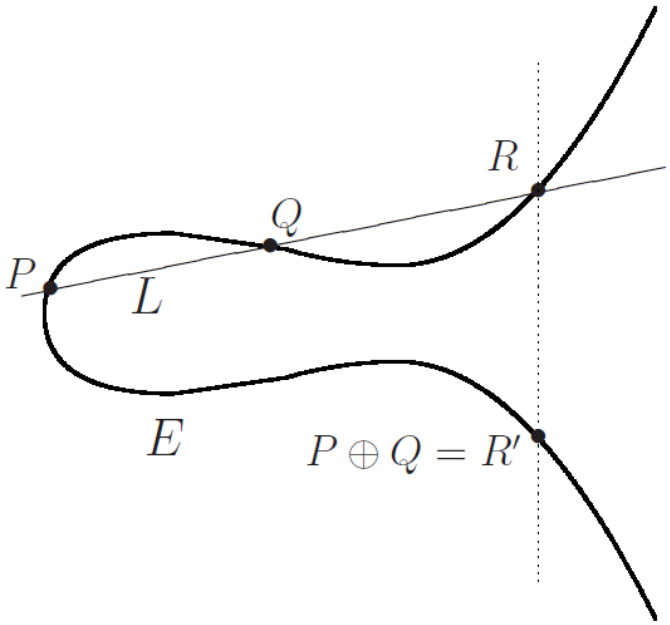


Figure 3.2: The addition law on an elliptic curve
[4]

Example 17. Let E be the elliptic curve

$$Y^2 = X^3 - 15X + 18. \tag{3.1}$$

The points $P = (7,16)$ and $Q = (1,2)$ are on the curve E. The line L connecting them is given by the equation

$$L: \quad Y = \frac{7}{3}X - \frac{1}{3}. \tag{3.2}$$

In order to find the points where E and L intersect, we substitute (3.1) into (3.2) and solve for X:

$$\left(\frac{7}{3}X - \frac{1}{3}\right)^2 = X^3 - 15X + 18,$$

$$\frac{49}{9}X^2 - \frac{14}{9}X + \frac{1}{9} = X^3 - 15X + 18,$$

$$0 = X^3 - \frac{49}{9}X^2 - \frac{121}{9}X + \frac{161}{9}.$$

We need to find the roots of this cubic polynomial. In general, finding the roots of a cubic is difficult. However, in this case we already know two of the roots, namely $X = 7$ and $X = 1$, since we know that P and Q are in the intersection $E \cap L$. It is then easy to find the other factor:

$$X^3 - \frac{49}{9}X^2 - \frac{121}{9}X + \frac{161}{9} = (X-7)(X-1)\left(X + \frac{23}{9}\right),$$

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 $X = -\frac{23}{9}$ into equation . This gives

$$Y = \frac{7}{3}\cdot\left(-\frac{23}{9}\right) - \frac{1}{3} = -\frac{161}{27} - \frac{1}{3} = -\frac{170}{27}.$$

Thus, the third point of intersection is $R = \left(-\frac{23}{9}, \frac{170}{27}\right)$. Finally, we reflect across the X-axis to obtain

$$P \oplus Q = \left(-\frac{23}{9}, -\frac{170}{27}\right).$$

[4]

There are a few subtleties to elliptic curve addition that need to be addressed. First, what happens if we want to add a point P to itself? Imagine what happens to the line L connecting P and Q if the point Q slides along the curve and gets closer and closer to P. In the limit, as Q approaches P, the line L becomes the tangent line to E at P. Thus, in order to add P to itself, we simply take L to be the tangent line to E at P, as illustrated in Figure 3.3. Then L intersects E at P and at one other point R, so we can proceed as before. In some sense, L still intersects E at three points, but P counts as two of them.

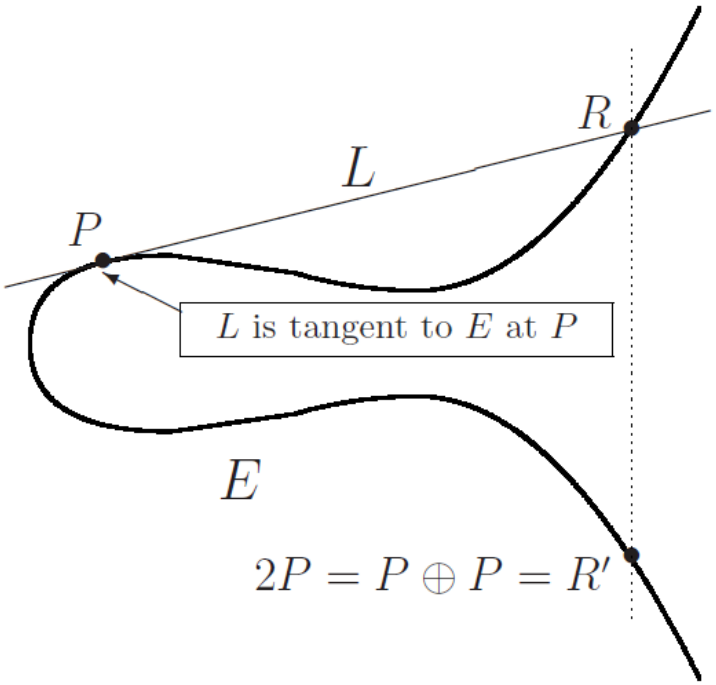


Figure 3.3: Adding a point P to itself
[4]

Example 18. Continuing with the curve E and point P from Example 17, we compute $P \oplus P$. The slope of E at P is computed by implicitly differentiating equation (3.1). Thus,

$$2Y\frac{dY}{dX} = 3X^2 - 15, \quad \text{so} \quad \frac{dY}{dX} = \frac{3X^2 - 15}{2Y}.$$

Substituting the coordinates of $P = (7, 16)$ gives slope $\lambda = \frac{33}{8}$, so the tangent line to E at P is given by the equation

$$L: \quad Y = \frac{33}{8}X - \frac{103}{8}. \tag{3.3}$$

Now we substitute (3.3) into the equation (3.1) for E, simplify, and factor:

$$\left(\frac{33}{8}X - \frac{103}{8}\right)^2 = X^3 - 15X + 18,$$

$$X^3 - \frac{1089}{64}X^2 + \frac{2919}{32}X - \frac{9457}{64} = 0,$$

$$(X-7)^2\left(X - \frac{193}{64}\right) = 0.$$

Notice that the X-coordinate of P, which is $X = 7$, appears as a double root of the cubic polynomial, so it was easy for us to factor the cubic. Finally, we substitute $X = \frac{193}{64}$ into the equation for L to get $Y = -\frac{223}{512}$, and then we switch the sign on Y to get

$$P \oplus P = \left(\frac{193}{64}, \frac{223}{512}\right).$$

[4]

A second potential problem with our "addition law" arises if we try to add a point $P = (a,b)$ to its reflection about the X-axis, denoted $\overline{P} = (a,-b)$. The line L through P and $\overline{P}$ is the vertical line $x = a$, and this line intersects E in only the two points P and $\overline{P}$. (See Figure 3.4.) There is no third point of intersection, so it appears that we are stuck! But there is a way out.

The solution is to create an extra point $\mathcal{O}$ that lives "at infinity." More precisely, the point $\mathcal{O}$ does not exist in the XY-plane, but we pretend that it lies on every vertical line. We then set

$$P \oplus \overline{P} = \mathcal{O}.$$

We also need to figure out how to add $\mathcal{O}$ to an ordinary point $P = (a,b)$ on E. The line L connecting P to $\mathcal{O}$ is the vertical line through P, since $\mathcal{O}$ lies on vertical lines, and that vertical line intersects E at the points P, $\mathcal{O}$, and $\overline{P} = (a,-b)$. To add P to $\mathcal{O}$, we reflect $\overline{P}$ across the X-axis, which gets us back to P. In other words,

$$P \oplus \mathcal{O} = P,$$

so $\mathcal{O}$ acts like zero for elliptic curve addition.

[4]

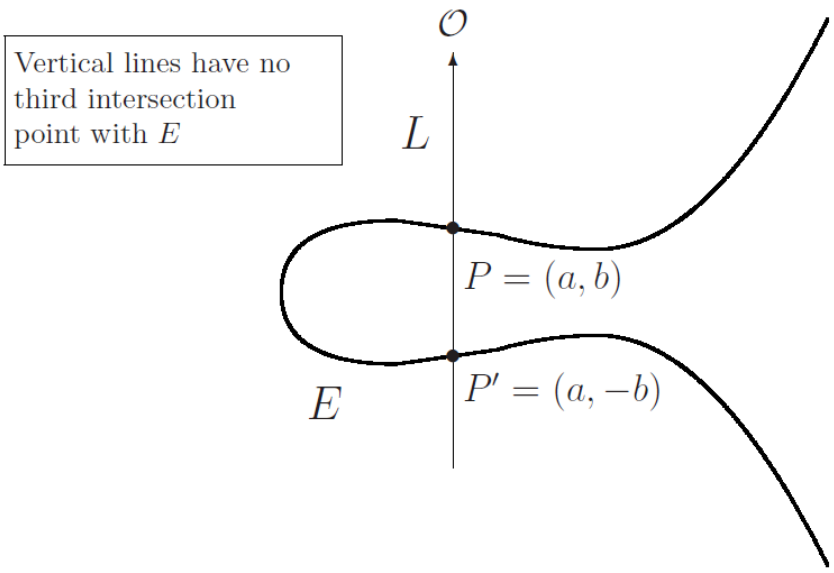


Figure 3.4: The vertical line L through $P = (a, b)$ and $P' = (a, -b)$.
[4]

Example 19. Continuing with the curve E from Example 17, notice that the point $T = (3, 0)$ is on the curve E and that the tangent line to E at T is the vertical line $X = 3$. Thus, if we add T to itself, we get

$$T \oplus T = \mathcal{O}.$$

[4]

Definition 3.1

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

$$4A^3 + 27B^2 \neq 0.$$

[4]

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 $\overline{R} = (a, -b)$ of R across the X-axis. This sum is denoted by

$$P \oplus Q,$$

or simply by $P + Q$.

Further, if $P = (a, b)$, we denote the reflected point by $\overline{P} = (a, -b)$, or simply by $-P$; and we define

$$P \ominus Q \quad (\text{or } P - Q) \quad \text{to be} \quad P \oplus (-Q).$$

Similarly, repeated addition is represented as multiplication of a point by an integer,

$$nP = \underbrace{P + P + \cdots + P}_{n \text{ copies}}.$$

[4]

Remark 3.1. What is this extra 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 $X^3 + AX + B$ has 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

$$4A^3 + 27B^2 \neq 0 \quad \text{if and only if} \quad e_1, e_2, e_3 \text{ are distinct.}$$

Curves with $\Delta_E = 0$ have *singular points*. The addition law does not work well on these curves. That is why we include the requirement that $\Delta_E \neq 0$ in our definition of an elliptic curve. [4]

Theorem 3.1. Let E be an elliptic curve. Then the addition law on E has the following properties:

(a) $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in E$. **[Identity]**

(b) $P + (-P) = \mathcal{O}$ for all $P \in E$. **[Inverse]**

(c) $(P + Q) + R = P + (Q + R)$ for all $P, Q, R \in E$. **[Associative]**

(d) $P + Q = Q + P$ for all $P, Q \in E$. **[Commutative]**

[4]

Proof. [4] *page 284-285* □

Theorem 3.2 – 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

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & \text{if } P_1 \neq P_2, \\ \dfrac{3x_1^2 + A}{2y_1} & \text{if } P_1 = P_2, \end{cases}$$

and let

$$x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda(x_1 - x_3) - y_1.$$

Then $P_1 + P_2 = (x_3, y_3)$.

[4]

Proof. [4] *page 285-286* □

3.2 Elliptic curves over finite fields

In the previous section we developed the theory of elliptic curves geometrically. For example, the sum of two distinct points P and Q on an elliptic curve E is defined by drawing the line L connecting P to Q and then finding the third point where L and E intersect, as illustrated in Figure 3.2. However, in order to apply the theory of elliptic curves to cryptography, we need to look at elliptic curves whose points have coordinates in a finite field $\mathbb{F}_p$. This is easy to do. We simply define an elliptic curve over $\mathbb{F}_p$ to be an equation of the form

$$E : Y^2 = X^3 + AX + B \quad \text{with } A, B \in \mathbb{F}_p \text{ satisfying } 4A^3 + 27B^2 \neq 0,$$

and then we look at the points on E with coordinates in $\mathbb{F}_p$, which we denote by

$$E(\mathbb{F}_p) = \{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p : y^2 = x^3 + Ax + B\} \cup \{\mathcal{O}\}.$$

[4]

Example 20. Consider the elliptic curve

$$E : Y^2 = X^3 + 3X + 8$$

over the field $\mathbb{F}_{13}$. We can find the points of $E(\mathbb{F}_{13})$ by substituting in all possible values $X = 0, 1, 2, \ldots, 12$ 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 $X = 1$, which gives $1 + 3 + 8 = 12$. 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 two points $(1, 5)$ and $(1, 8)$ in $E(\mathbb{F}_{13})$. Continuing in this fashion, we end up with 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. [4]

Suppose now that P and Q are two points in $E(\mathbb{F}_p)$ and that we want to "add" the points P and Q. One possibility is to develop a theory of geometry using the field $\mathbb{F}_p$ instead of $\mathbb{R}$. Then we could mimic our earlier constructions to define $P+Q$. This can be done, and it leads to a fascinating field of mathematics called algebraic geometry. However, in the interests of brevity of exposition, we instead use the explicit formulas given in Theorem 3.2 to add points in $E(\mathbb{F}_p)$. But we note that if one wants to gain a deeper understanding of the theory of elliptic curves, then it is necessary to use some of the machinery and some of the formalism of algebraic geometry.

Let $P=(x_1,y_1)$ and $Q=(x_2,y_2)$ be points in $E(\mathbb{F}_p)$. We define the sum P_1+P_2 to be the point (x_3,y_3) obtained by applying the elliptic curve addition algorithm (Theorem 3.2). Notice 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 $\mathbb{F}_p$. Of course, it is not completely clear that (x_3,y_3) is a point in $E(\mathbb{F}_p)$.

[4]

Theorem 3.3. 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 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 3.1. In other words, this addition law makes $E(\mathbb{F}_p)$ into a finite group.

[4]

Proof. [4] *page 287-288* □

Example 21. We continue with the elliptic curve

$$E: Y^2=X^3+3X+8 \quad \text{over } \mathbb{F}_{13}$$

from Example 5.8, and we use the addition algorithm (Theorem 3.2) 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 recall that all computations are being performed in the field $\mathbb{F}_{13}$, so $-8\equiv 5 \pmod{13}$ and $\frac{1}{5}=5^{-1}=8$. Next we compute

$$\nu=y_1-\lambda x_1=7-8\cdot 9=-65\equiv 0 \pmod{13}.$$

Finally, the addition algorithm tells us to compute

$$x_3=\lambda^2-x_1-x_2=64-9-1=54\equiv 2 \pmod{13},$$

$$y_3=-(\lambda x_3+\nu)=-8\cdot 2=-16\equiv 10 \pmod{13}.$$

This completes the computation of

$$P+Q=(1,8)+(9,7)=(2,10)\in E(\mathbb{F}_{13}).$$

Similarly, we can use the addition algorithm to add $P = (9,7)$ to itself. Keeping in mind that all calculations are in $\mathbb{F}_{13}$, we find that

$$\lambda = \frac{3x_1^2 + A}{2y_1} = \frac{3 \cdot 9^2 + 3}{2 \cdot 7} = \frac{246}{14} \equiv \frac{246 \mod 13}{14 \mod 13} = \frac{12}{1} = 12 \equiv 1 \pmod{13},$$

and

$$\nu = y_1 - \lambda x_1 = 7 - 1 \cdot 9 = -2 \equiv 11 \pmod{13}.$$

Then

$$x_3 = \lambda^2 - x_1 - x_2 = 1 - 9 - 9 = -17 \equiv 9 \pmod{13}, \quad y_3 = -(\lambda x_3 + \nu) = -1 \cdot 9 - 11 = -20 \equiv 6 \pmod{13},$$

so

$$P + P = (9,7) + (9,7) = (9,6) \in E(\mathbb{F}_{13}).$$

In a similar fashion, we can compute the sum of every pair of points in $E(\mathbb{F}_{13})$. The results are listed in Table 3.1.

[4]

	$\mathcal{O}$	(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 3.1: Addition table for $E : Y^2 = X^3 + 3X + 8$ over $\mathbb{F}_{13}$

[4]

It is clear that the set of points $E(\mathbb{F}_p)$ is a finite set, since there are only finitely many possibilities for the X- and Y-coordinates. More precisely, 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. 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 larger than the true size.

When we plug in a value for X, there are three possibilities for the value of the quantity

$$X^3 + AX + B.$$

First, it may be a quadratic residue modulo p, in which case it has two square roots and we get two points in $E(\mathbb{F}_p)$. This happens about 50% of the time. Second, it may be a nonresidue modulo p, in which case we discard X. This also happens about 50% of the time. Third, it might equal 0, in which case we get one point in $E(\mathbb{F}_p)$, but this case happens very rarely. Thus we might expect that the number of points in $E(\mathbb{F}_p)$ is approximately

$$\#E(\mathbb{F}_p) \approx 50\% \cdot 2 \cdot p + 1 = p + 1.$$

A famous theorem of Hasse, later vastly generalized by Weil and Deligne, says that this is true up to random fluctuations.

[4]

Theorem 3.4 – Hasse. Let E be an elliptic curve over $\mathbb{F}_p$. Then

$$\#E(\mathbb{F}_p) = p + 1 - t_p$$

with t_p satisfying

$$|t_p| \leq 2\sqrt{p}.$$

[4]

Definition 3.2

The quantity

$$t_p = p + 1 - \#E(\mathbb{F}_p)$$

appearing in Theorem 5.11 is called the **trace of Frobenius** for $E/\mathbb{F}_p$. We will not explain the somewhat technical reasons for this name, other than to say that t_p appears as the trace of a certain 2×2 matrix that acts as a linear transformation on a certain two-dimensional vector space associated to $E/\mathbb{F}_p$. [4]

Example 22. 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 3.2 lists the results for the first few primes, together with the value of t_p and, for comparison purposes, the value of $2\sqrt{p}$. [4]

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 3.2: Number of points and trace of Frobenius for $E : Y^2 = X^3 + 4X + 6$ [4]

Remark 3.2. Hasse's theorem gives a bound for $\#E(\mathbb{F}_p)$, but it does not provide a method for calculating this quantity. In principle, one can substitute in each value for X and check the value of $X^3 + AX + B$ against a table of squares modulo p, but this takes time $\mathcal{O}(p)$, so is very inefficient. [4]

3.3 The elliptic curve discrete logarithm problem (ECDLP)

In Chapter 2 we talked about 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 that solves the congruence

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

Let's consider how Alice can do something similar with an elliptic curve E over $\mathbb{F}_p$. If Alice views g and h as being elements of the group $\mathbb{F}_p^*$, then the discrete logarithm problem requires Alice's adversary Eve 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 by itself in order to get to 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 that makes

$$Q = \underbrace{P + P + P + \cdots + P}_{n \text{ additions on } E} = nP.$$

Then Eve needs to find out how many times P must be added to itself in order to get Q. Keep in mind that although the "addition law" on an elliptic curve is conventionally written with a plus sign, addition on E is actually a very complicated operation, so this elliptic analogue of the discrete logarithm problem may be quite difficult to solve. [4]

Definition 3.3

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. [4]

Remark 3.3. Our definition of $\log_P(Q)$ is not quite precise. The first difficulty is that there may be points $P, Q \in E(\mathbb{F}_p)$ such that Q is not a multiple of P. In this case, $\log_P(Q)$ is not defined. However, for cryptographic purposes, Alice starts out with a public point P and a private integer n and she computes and publishes the value of $Q = nP$. So in practical applications, $\log_P(Q)$ exists and its value is Alice's secret. [4]

The second difficulty is that if there is one value of n satisfying $Q = nP$, then there are many such values. To see this, we first note that there exists a positive integer s such that $sP = O$. We recall the easy proof of this fact. Since $E(\mathbb{F}_p)$ is finite, the points in the list $P, 2P, 3P, 4P, \ldots$ cannot all be distinct. Hence there are integers $k > j$ such that $kP = jP$, and we can take $s = k - j$. The smallest such $s \geq 1$ is called the order of P. Thus if s is the order of P and if n_0 is any integer such that $Q = n_0 P$, then the solutions to $Q = nP$ are the integers $n = n_0 + is$ with $i \in \mathbb{Z}$.

This means that the value of $\log_P(Q)$ 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. For concreteness we could set $\log_P(Q)$ equal to n_0. However the advantage of defining the values to be in $\mathbb{Z}/s\mathbb{Z}$ is that the elliptic discrete logarithm then satisfies

$$\log_P(Q_1 + Q_2) = \log_P(Q_1) + \log_P(Q_2)$$

for all $Q_1, Q_2 \in E(\mathbb{F}_p)$. Notice the analogy with the ordinary logarithm

$$\log(\alpha\beta) = \log(\alpha) + \log(\beta)$$

and the discrete logarithm for $\mathbb{F}_p^*$. The fact that the discrete logarithm for $E(\mathbb{F}_p)$ satisfies (5.4) means that it respects the addition law when the group $E(\mathbb{F}_p)$ is mapped to the group $\mathbb{Z}/s\mathbb{Z}$. We say that the map $\log_P$ defines a group homomorphism

$$\log_P : E(\mathbb{F}_p) \longrightarrow \mathbb{Z}/s\mathbb{Z}.$$

[4]

Example 23. 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 (by hand if you're patient and with a computer if not) that

$$Q = 11P, \quad \text{so } \log_P(Q) = 11.$$

Similarly, $R = (35, 47) \in E(\mathbb{F}_{73})$ and $S = (58, 4) \in E(\mathbb{F}_{73})$, and after some computation we find that they satisfy

$$R = 37P \quad \text{and} \quad S = 28P,$$

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

$$41P = O.$$

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 a multiple of P. [4]

3.3.1 How hard is the ECDLP?

The collision algorithms described in Section 4.4 are easily adapted to any group, for example to the group of points $E(\mathbb{F}_p)$ on an elliptic curve. In order to solve the equation

$$Q = nP,$$

Eve chooses random integers $j_1, \ldots, j_r$ and $k_1, \ldots, k_r$ between 1 and p, and makes two lists of points:

List #1. $j_1P, j_2P, j_3P, \ldots, j_rP$,

List #2. $k_1P + Q, k_2P + Q, k_3P + Q, \ldots, k_rP + Q$.

As soon as she finds a match (collision) between the two lists, she is done, since if she finds

$$j_uP = k_vP + Q,$$

then

$$Q = (j_u - k_v)P$$

provides the solution.

[4]

3.4 Elliptic curve cryptography

It is finally time to apply elliptic curves to cryptography. We start with the easiest application, **Diffie–Hellman key exchange**, which involves little more than replacing the discrete logarithm problem for the finite field $\mathbb{F}_p$ with the discrete logarithm problem for an elliptic curve $E(\mathbb{F}_p)$. We then describe an elliptic analogue of the **ElGamal public key cryptosystem**. [4]

3.4.1 Elliptic Diffie–Hellman key exchange

Alice and Bob agree to use a particular elliptic curve $E(\mathbb{F}_p)$ and a particular point $P \in E(\mathbb{F}_p)$.

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. Compute $Q_A \equiv n_A P$.	Choose a secret integer n_B. Compute $Q_B \equiv n_B P$.
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**
Compute the point $n_A Q_B$.	Compute 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 3.3: Diffie–Hellman key exchange
[4]

Alice chooses a secret integer n_A and Bob chooses a secret integer n_B. They compute the associated multiples:

$$Q_A = n_A P \quad \text{and} \quad Q_B = n_B P,$$

and exchange the values of Q_A and Q_B. Alice then uses her secret multiplier to compute $n_A Q_B$, and Bob similarly computes $n_B Q_A$. They now have the shared secret value

$$n_A Q_B = n_A(n_B P) = n_B(n_A P) = n_B Q_A,$$

which they can use as a key to communicate privately via a symmetric cipher.
[4]

Example 24. Alice and Bob decide to use elliptic Diffie–Hellman with the following parameters:

$$p = 3851, \quad E : Y^2 = X^3 + 324X + 1287, \quad P = (920, 303) \in E(\mathbb{F}_{3851}).$$

Alice and Bob choose respective secret values $n_A = 1194$ and $n_B = 1759$, and then compute:

$$Q_A = 1194P = (2067, 2178) \in E(\mathbb{F}_{3851}),$$

$$Q_B = 1759P = (3684, 3125) \in E(\mathbb{F}_{3851}).$$

Alice sends Q_A to Bob and Bob sends Q_B to Alice. Finally,

$$\text{Alice computes } n_A Q_B = 1194 \cdot (3684, 3125) = (3347, 1242) \in E(\mathbb{F}_{3851}),$$

$$\text{Bob computes } n_B Q_A = 1759 \cdot (2067, 2178) = (3347, 1242) \in E(\mathbb{F}_{3851}).$$

Bob and Alice have thus exchanged the secret point $(3347, 1242)$. They should discard the y-coordinate and use only the x-coordinate, $x = 3347$, as their shared secret value.

One way for Eve to discover Alice and Bob's secret is to solve the elliptic curve discrete logarithm problem (ECDLP)

$$n_A P = Q_A,$$

since if Eve can solve this problem, she obtains n_A and can use it to compute $n_A Q_B$. Of course, there might be other ways for Eve to compute the secret without explicitly solving the ECDLP. The exact problem that Eve needs to solve is the elliptic curve analogue of the Diffie–Hellman problem. [4]

3.4.2 Elliptic ElGamal public key cryptosystem

It is easy to create a direct analogue of the ElGamal public key cryptosystem. Briefly, Alice and Bob agree to use a particular prime p, elliptic curve E, and 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 ephemeral key and computes

$$C_1 = kP \quad \text{and} \quad C_2 = M + kQ_A.$$

He sends the two points (C_1, C_2) to Alice, who computes

$$C_2 - n_A C_1 = (M + kQ_A) - n_A(kP) = M + k(n_A P) - n_A(kP) = M$$

to recover the plaintext. The elliptic ElGamal public key cryptosystem is summarized in Table 3.4.

In principle, the elliptic ElGamal cryptosystem works fine, but there are some practical difficulties. [4]

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	
Chooses a private key n_A. Computes $Q_A = n_A P$ in $E(\mathbb{F}_p)$. Publishes the public key Q_A.	
Encryption	
	Chooses plaintext $M \in E(\mathbb{F}_p)$. Chooses an ephemeral key k. Uses Alice's public key Q_A to compute: $C_1 = kP \in E(\mathbb{F}_p)$, $C_2 = M + kQ_A \in E(\mathbb{F}_p)$. Sends ciphertext (C_1, C_2) to Alice.
Decryption	
Computes $C_2 - n_A C_1 \in E(\mathbb{F}_p)$. This quantity is equal to M.	

Table 3.4: ECC-based ElGamal key creation, encryption, and decryption [4]

1. There is no obvious way to attach plaintext messages 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 expansion ratio of ElGamal using $\mathbb{F}_p$. (See Remark 2.9.)

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, there are approximately p different points in $E(\mathbb{F}_p)$, hence only about 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 circumvented by choosing M randomly and using it as a mask for the actual plaintext. One such method, which also decreases message expansion. Another natural way to improve message expansion is to send only the x-coordinates of C_1 and C_2, as was suggested for 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

$$\text{Extra bit} = \begin{cases} 0 & \text{if } 0 \le y < \frac{p}{2}, \\ 1 & \text{if } \frac{p}{2} < y < p. \end{cases}$$

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*.

[4]

3.5 Elliptic Curve Digital Signature Algorithm (ECDSA)

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the Digital Signature Algorithm (DSA). It is the most widely standardized elliptic curve-based signature scheme, appearing in the ANSI X9.62, FIPS 186-2, IEEE 1363-2000, and ISO/IEC 15946-2 standards as well as several draft standards.

In the following, H denotes a cryptographic hash function whose outputs have bitlength no more than that of n. (If this condition is not satisfied, then the outputs of H can be truncated.) [3]

Algorithm 6 ECDSA Signature Generation

Require: Domain parameters $D = (q, \mathbb{F}_R, S, a, b, P, n, h)$, private key d, message m
Ensure: Signature (r, s)
1: Select $k \in_R [1, n-1]$
2: Compute $kP = (x_1, y_1)$ and convert x_1 to an integer
3: Compute $r = x_1 \mod n$; if $r = 0$, go to step 1
4: Compute $e = H(m)$
5: Compute $s = k^{-1}(e + dr) \mod n$; if $s = 0$, go to step 1
6: **return** (r, s)

[3]

Algorithm 7 ECDSA Signature Verification

Require: Domain parameters $D = (q, \mathbb{F}_R, S, a, b, P, n, h)$, public key Q, message m, signature (r, s)
Ensure: Acceptance or rejection of the signature
1: **if** $r \notin [1, n-1]$ or $s \notin [1, n-1]$ **then**
2: **return** "Reject the signature"
3: **end if**
4: Compute $e = H(m)$
5: Compute $w = s^{-1} \mod n$
6: Compute $u_1 = ew \mod n$ and $u_2 = rw \mod n$
7: Compute $X = u_1 P + u_2 Q$
8: **if** $X = \infty$ **then**
9: **return** "Reject the signature"
10: **end if**
11: Convert the x-coordinate x_1 of X to an integer and compute $v = x_1 \mod n$
12: **if** $v = r$ **then**
13: **return** "Accept the signature"
14: **else**
15: **return** "Reject the signature"
16: **end if**

[3]

Proof That Signature Verification Works. If a signature (r, s) on a message m was indeed generated by the legitimate signer, then by the ECDSA signature generation algorithm, we have:

$$s \equiv k^{-1}(e + d\,r) \pmod{n}$$

Rearranging the equation gives:

$$k \equiv s^{-1}(e + d\,r) \pmod{n}$$

We can split the terms as:

$$k \equiv s^{-1}e + s^{-1}d\,r \pmod{n}$$

Let $w = s^{-1} \mod n$, so we define:

$$u_1 = e\,w \mod n, \quad u_2 = r\,w \mod n$$

Then:

$$k \equiv u_1 + u_2 d \pmod{n}$$

Therefore, when computing the verification expression:

$$X = u_1 P + u_2 Q = (u_1 + u_2 d)P = kP$$

Since $X = kP$, and during signing we computed $r = x_1 \mod n$, where x_1 is the x-coordinate of kP, we conclude:

$$v = x_1 \mod n = r$$

Thus, the signature is verified successfully. [3] □

Chapter 4
Applications

4.1 ECC in Bitcoin

4.1.1 Elliptic Curve Cryptography Explained

Elliptic curve cryptography is a type of asymmetric or public-key cryptography based on the discrete logarithm problem as expressed by addition and multiplication on the points of an elliptic curve. Figure 4.1 is an example of an elliptic curve, similar to that used by bitcoin.

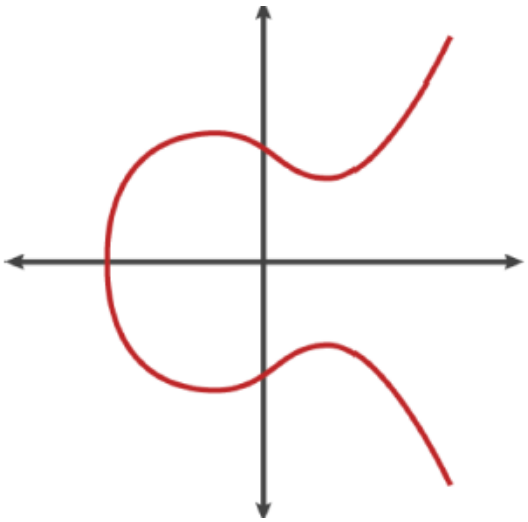


Figure 4.1: An elliptic curve
[1]

Bitcoin uses a specific elliptic curve and set of mathematical constants, as defined in a standard called `secp256k1`, established by the National Institute of Standards and Technology (NIST). The `secp256k1` curve is defined by the following function, which produces an elliptic curve:

$$y^2 = x^3 + 7 \quad \text{over } \mathbb{F}_p$$

or equivalently,

$$y^2 \mod p = (x^3 + 7) \mod p$$

The term $\mod p$ (modulo a prime number p) indicates that this curve is defined over a finite field of prime order p, also written as $\mathbb{F}_p$, where

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

This is a very large prime number.

Because the curve is defined over a finite field of prime order instead of over the real numbers, it does not form a smooth curve. Instead, it looks like a pattern of discrete points (or dots) scattered across a two-dimensional grid, which makes it difficult to visualize directly. However, the underlying mathematics is identical to that of an elliptic curve over the real numbers.

As an illustrative example, consider the same elliptic curve over a much smaller finite field of prime order 17. The resulting points form a visibly structured pattern of dots on a grid. The `secp256k1` Bitcoin elliptic curve can be thought of as a vastly more complex version of this pattern, spread over an unimaginably large grid.

[1]

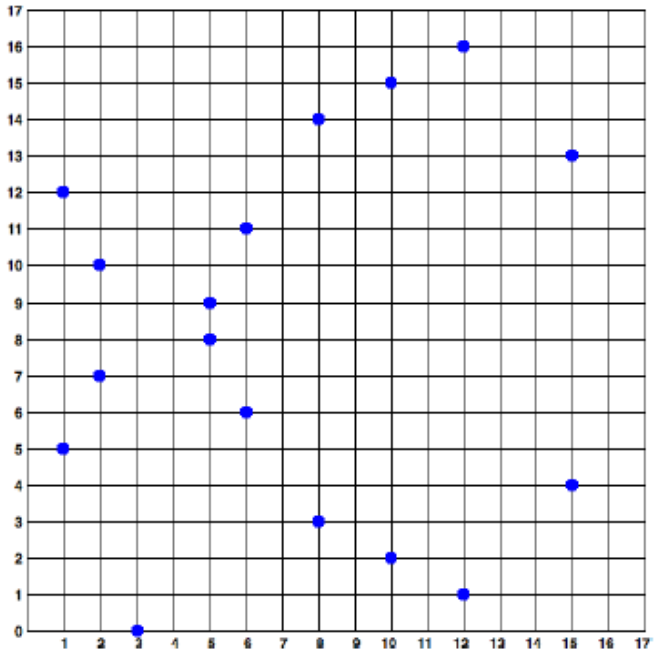


Figure 4.2: Elliptic curve cryptography: visualizing an elliptic curve over $F(p)$, with $p = 17$

[1]

An example point $P = (x, y)$ on the `secp256k1` curve is:

$x = 55066263022277343669578718895168534326250603453777594175500187360389116729240$

$y = 32670510020758816978083085130507043184471273380659243275938904335757337482424$

This point satisfies the elliptic curve equation. We can verify this using Python:

```
    P = (
    55066263022277343669578718895168534326250603453777594175500187360389116729240,
    32670510020758816978083085130507043184471273380659243275938904335757337482424
    )

Python 3.4.0 (default, Mar 30 2014, 19:23:13)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
        p = 115792089237316195423570985008687907853269984665640564039457584007908834671663
        x = 55066263022277343669578718895168534326250603453777594175500187360389116729240
        y = 32670510020758816978083085130507043184471273380659243275938904335757337482424
        (x ** 3 + 7 - y ** 2) % p
        # Output: 0
```

[1]

In elliptic curve math, there is a point called the **point at infinity**, which roughly corresponds to the role of 0 in addition. On computers, it is sometimes represented by $x = y = 0$, which does not satisfy the elliptic curve equation, but it provides an easy separate case that can be checked.

There is also a + operator, called **addition**, which has properties similar to the traditional addition of real numbers. Given two points P_1 and P_2 on the elliptic curve, there is a third point $P_3 = P_1 + P_2$, also on the elliptic curve.

Geometrically, this third point P_3 is calculated by drawing a line between P_1 and P_2. This line will intersect the elliptic curve in exactly one additional point, denoted $P_3' = (x, y)$. Reflect this point across the x-axis to get $P_3 = (x, -y)$.

There are a couple of special cases that explain the need for the point at infinity:

- If $P_1 = P_2$, the line between them becomes the tangent at that point on the curve. This tangent intersects the curve at exactly one new point. The slope of the tangent can be calculated using calculus, even though we are considering integer coordinates.

- If P_1 and P_2 have the same x-value but different y-values, the line is vertical. In this case, we define $P_3 = \mathcal{O}$, the point at infinity.

- If $P_1 = \mathcal{O}$, then $P_1 + P_2 = P_2$. Similarly, if $P_2 = \mathcal{O}$, then $P_1 + P_2 = P_1$. This shows that the point at infinity acts like the number zero.

The addition operation is **associative**, meaning:

$$(A + B) + C = A + (B + C)$$

This means we can write expressions like $A + B + C$ without parentheses and without ambiguity.

Now that we have defined addition, we can define multiplication in the standard way that extends addition. For a point P on the elliptic curve, and a whole number k, we define:

$$kP = \underbrace{P + P + \cdots + P}_{k \text{ times}}$$

Note that k is sometimes confusingly called an *exponent* in this case.

[1]

4.1.2 Generating a Public Key

Starting with a private key in the form of a randomly generated number k, we multiply it by a predetermined point on the curve, called the **generator point** G, to produce another point on the curve. This resulting point is the corresponding **public key** K.

The generator point G is specified as part of the `secp256k1` standard and is the same for all keys in Bitcoin. The public key is computed as:

$$K = k \cdot G$$

where:

- k is the private key (a randomly chosen large number),
- G is the generator point (a constant, defined by the elliptic curve standard),
- K is the resulting public key (a point on the elliptic curve).

Because the generator point G is always the same for all Bitcoin users, multiplying a private key k with G will always result in the same public key K.

The relationship between k and K is **deterministic** but **one-way**: it is computationally feasible to calculate K from k, but infeasible to derive k from K. This is the foundation of elliptic curve cryptography's security.

As a result, a Bitcoin address (which is derived from K) can be safely shared with others, without revealing the private key k.

A private key can be converted into a public key, but a public key cannot be converted back into a private key because the math only works one way.

Implementing elliptic curve multiplication, we take the private key k generated previously and multiply it with the generator point G to find the public key K:

$$K = \texttt{1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD} \cdot G$$

The public key K is defined as a point $K = (x, y)$, where:

$$x = \texttt{F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A}$$
$$y = \texttt{07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB}$$

To visualize the multiplication of a point with an integer, we consider a simpler elliptic curve over the real numbers — remember, the math is fundamentally the same. Our goal is to find the multiple kG of the generator point G, which is equivalent to adding G to itself k times.

In elliptic curves, adding a point to itself involves drawing a tangent line at the point, finding where it intersects the curve again, and then reflecting that point across the x-axis.

Figure 4.3 illustrates the process for deriving $G, 2G, 4G$ as geometric operations on the curve.

Most Bitcoin implementations use the `OpenSSL` cryptographic library to do the elliptic curve math. For example, to derive the public key, the function `EC_POINT_mul()` is used.

[1]

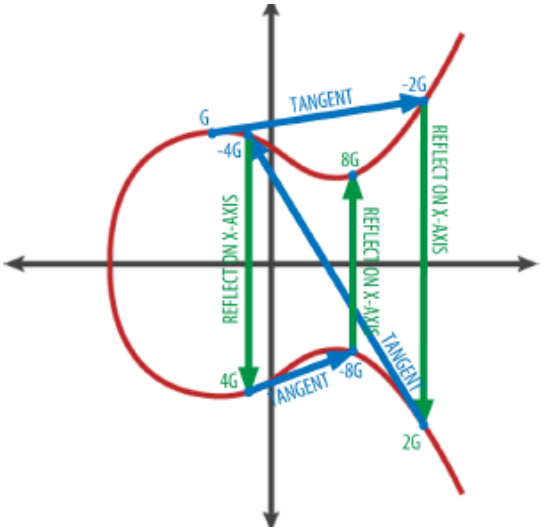


Figure 4.3: Elliptic curve cryptography: Visualizing the multiplication of a point G by an integer k on an elliptic curve

[1]

Conclusion

Elliptic Curve Cryptography (ECC) represents a powerful and efficient approach to modern public-key cryptography. By leveraging the mathematical structure of elliptic curves over finite fields, ECC achieves high levels of security with much smaller key sizes compared to traditional systems such as RSA.

Throughout this thesis, we explored the fundamental concepts of ECC, including the algebra of elliptic curves, finite field arithmetic, and essential cryptographic schemes built on ECC such as ECDH, ECDSA, and ElGamal. We also examined how ECC is applied in real-world systems such as blockchain technologies and cryptocurrencies like Bitcoin.

The main advantage of ECC lies in its ability to provide equivalent security to older systems using significantly less computational power and bandwidth, making it ideal for modern applications ranging from secure communications to digital currencies and constrained devices (e.g., IoT).

In conclusion, ECC continues to gain widespread adoption due to its robustness, efficiency, and scalability, and is expected to play a central role in the future of secure digital communication and cryptographic infrastructures.

Bibliography

[1] Antonopoulos, A. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. O'Reilly Media, Sebastopol, 2014.

[2] Bashir, I. *Mastering Blockchain: Distributed Ledger Technology, Decentralization and Smart Contracts Explained (2nd Edition)*. Packt Publishing, Birmingham, 2018.

[3] Hankerson, D., Vanstone, S., and Menezes, A. *Guide to Elliptic Curve Cryptography*. Springer, New York, 2004.

[4] Hoffstein, J., Pipher, J., and Silverman, J. *An Introduction to Mathematical Cryptography*. Springer, New York, 2008.

[5] Menezes, A., van Oorschot, P., and Vanstone, S. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, 1996.