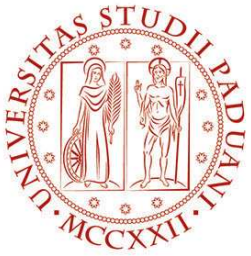


UNIVERSITÀ DEGLI STUDI DI PADOVA

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Corso di Laurea Magistrale in Matematica

Tesi di Laurea Magistrale

Cryptography and elliptic curves

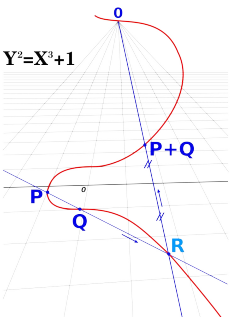
A path through modern mathematics and its worldwide daily hidden use

Relatrice:
Orsola Tommasi

Candidato:
Giulio Pozza
Matricola 2052534

Anno accademico 2024/2025
11 Aprile 2025

Contents



0 Introduction

This work starts with a naive question: how does modern mathematics influence nowadays life?
More precisely, I heard this question from a young friend, who has no background math studies after lower secondary school. He asked me what more there is to study in math beyond arithmetic. Moreover, assuming there is some further knowledge to deepen, does it have any sense for the daily life anyone lives?
This is not as naive as it may seem, because frequently math acts very silently and does not reveal itself. The following pages are devoted to answer this question by the example of cryptography, in particular via elliptic curves. We will reveal the hidden work of modern mathematics (in particular geometry, number theory and algebra), pointing out the role they play on protecting our sensitive data and privacy in every minute of our lives, ensuring secure channels of communication between people, and helping with the authentication of who we are communicating with.

We start in Chapter 1 with an introduction to cryptography, following a historical path, detailing its evolution and its relationship with mathematics. Chapter 2 is devoted to the theory of elliptic curves, where we limit ourselves to the results useful for the aim of cryptography. Chapter 3 contains the algorithms of elliptic curves cryptosystems as well as some results that restrict its use to special cases of elliptic curves. We conclude with Chapter 4, which deals with the (unaware) daily life use of elliptic curve cryptography.

I write these few lines for all the important people during my university path. First of all, I am grateful to my supervisor for the precious help in writing down these pages and making them ordered via the precision that characterized her professional attitude. Second, I am lucky on having my family and my friends as daily supporters of my studies, enhancing the quality of my life in general. A particular appreciation goes to my mom, who moreover has played the role of a print shop for me during these years. I thank also my job collagues, for bearing with me on all those days I was in a bad mood due to the challenge of working and studying at the same time. Finally, I am thankful to all the staff of University of Padova, from lecturers to every kind of employee, for keeping the University as a democratic place in which exchanging ideas, cultures and humanity is possible.

1 Historical background

Cryptography has accompanied humanity from the origin of civility, and has assumed many forms during the centuries, but its goal has ever been the same: to provide safe ways to communicate information. From a higher point of view, cryptography consists of a way of passing from a clear message (plaintext) to an unreadable one (ciphertext) with the aim of having a safe channel of communication; but due to its dual nature, at the same time it is concerned with the inverse technique, that is a way of interpreting a cover message, the so-called deciphering.
There are three dates which are central in our perspective. The first one is 1976, which represents (officially) the year in which public key cryptography made its appearance. The second one is less known: 1926. In this year matematicians started to give their contribution to cryptography. The last is 1984, the year in which elliptic curves were first introduced as a tool for cryptography.
In this chapter, we briefly start from the very beginning of the history of cryptography, passing through the dates above, reaching the present time with the crucial observation that compared to the past everyone does a daily use of cryptography. We refer the curious reader to [22] and [10] for this part.
From here on, we will use the classical convention that Alice and Bob represent the two end users (or entities, could be peoples, states, military commands, governments...) who try to establish a secure communication. Meanwhile, Eve is used as the name of the third person who wants to intercept their messages and make them readable (i.e. Eve tries to decipher).

1.1 Private Key

From the ancient times to nowadays there have been strong battles between two categories. On one hand we find the cryptographers: they start from a plaintext and, using a cryptosystem, they transform it in a ciphertext, whose feature is unreadability, in principle for all except for the designed recipient. On the other hand, there are the cryptanalists (or decryptors), who do the inverse: they catch encrypted messages and try to turn them back into plaintext. From the origin to 1976, the first group has obeyed to the famous Kerckhoffs' law:
"The security of a cryptosystem should not have to rely on hiding the algorithm used, it is based on keeping the key concealed."
A cryptosystem basically consists of an algorithm, that is, a series of instructions to transform a plaintext in a crypted text, and a key used for customizing the algorithm. Following the Kerckhoffs' law, if Alice and Bob want to communicate safely, it doesn't matter if Eve knows which algorithm they choose to encrypt their messages: the important point is they keep secret the key. This way of thinking is the paradigm of private key cryptography.
We give some historical examples, without any aim of completeness:

Example 1.1. (Steganography)
Steganography might be a story to tell at all levels of school due to its surprisingly facets. It is a predecessor of cryptography: it consists on physically concealing a message with any

possible stratagem. A famous example concerns the ancient Greek historian Herodotus who, recounting the battles between Greeks and Persians, reports the story of the Greek Histiaeus, who wanted to convince Aristagoras to rebel to the Persians. But Aristagoras was living in Persia, while Histiaeus in Greek. How could Histiaeus send a secure message to Aristagoras without any Persian finding? Histiaeus called a messenger, shaved his head and wrote the secret message on the skinhead. Then he waited that the hairs of the messenger had been regrown (of course, at that time there was not the hurry of present days). Once the hairs grew back, he sent the messenger to Aristagoras, instructing him to shave his head to show the message only once he would have reached Aristagoras. This is a funny example of how steganography works in principle!
There are plenty of examples of steganography (as the use of eggs and their shell porosity to write in there and transport secret messages, revealing them only with the correct boiling method...); [22] and [10] are exhaustive on this.
The point is, the algorithm here is simply physically concealing using some tool and the key is the tool itself (a head, an egg...).

Example 1.2. (Caesar's Cipher)
Caesar's Cipher is one of the simpler example of private key cryptosystem. Its algorithm is based on substituting any letter with a letter that lays k places ahead in the alphabet. Summarizing, if we call A the set of numerical alphabet (where 0 corresponds to a, 1 is b,..., 25 is z in our alphabet composed of 26 letters), then the algorithm is described by the function $c : A \longrightarrow A$, $x \mapsto x + k \mod \#(A)$, for some $k \in \mathbb{Z}$. The key of this cryptosystem is exactly k.
In the figure below we give the example with $k = 4$ and $\#(A) = 21$, plain characters are on the top, cipher ones are below.

plain	A	B	C	D	E	F	G	H	I	L	M	N	O	P	Q	R	S	T	U	V	Z
cipher	D	E	F	G	H	I	L	M	N	O	P	Q	R	S	T	U	V	Z	A	B	C

Suppose Alice would like to send Bob the message "Meet at the club". They first have to meet and agree about using the Caesar's cipher and exchange the key ($k = 4$). After that, Alice encrypts her message letter by letter, obtaining: "Phhz dz zmh foae". She then sends the encrypted message to Bob, who owns the private key and is able to clear the message doing the "same" operation Alice did. Roughly speaking, he just have to invert the two rows of the figure above and "encrypt again".
Notice that if Eve intercept the message and would like to decipher it, assuming furthermore she knows the algorithm used by Bob and Alice is the Caesar's one, it would be enough for her to apply the same key k to read the plaintext, the same work that Bob did to decipher the message with the key he possesses.
This cryptosystem is an example of a monoalphabetic cipher. The birth of cryptoanalysis in the Arabic world at the end of the first millennium, in particular the analysis of the frequencies of the single letters in any language, has weakened Caesar's cipher and made it obsolete for secure communications, already in the first millennum.

Example 1.3. (Vigenère's Cipher)
Blaise de Vigenère (1523-1596) built up a cryptosystem that was considered indecipherable for hundreds of years. The idea behind it is to use not just one ciphering alphabet, but 26 alphabets (at least in case the alphabet consists of 26 characters). The algorithm is composed by many steps, each of which is nothing but a Caesar cipher. The key is a word, who may be thought of as an n-tuple, where n is the length of the word, which indicates at any step which is the single key k of the Caesar cipher.
Concretely, let i the i-th step of the procedure, and let $k = (k_1, k_2, ..., k_n)$ the key (again, we have transformed letters of the alphabet in numbers), then the i-th enciphering function will be $c_i : A \longrightarrow A$, $x \mapsto (x + k_{(i) \mod n}) \mod \#A$.
The following Figure 1 reports all the 26 ciphering alphabets.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 1: Vigenère cipher table.

Suppose again Alice wants to send the message "Meet at the club" to Bob, this time using Vigenère's cipher. They first have to agree on the key, let us say they choose "sun". Alice starts ciphering. The first letter is M, and she uses the Caesar cipher given by the first letter of the key, S, that corresponds to the substituiton with $k = 18$. Using the table above, Alice finds E. She then passes to the second step, she enciphers letter E using the second letter of the key, U, i.e. $k = 20$, finding Y. And so on, observing that at step 4 she returns to the first letter of the key, S. Finally, she obtains the following ciphertext:

key	S	U	N	S	U	N	S	U	N	S	U	N	S
plaintext	M	E	E	T	A	T	T	H	E	C	L	U	B
ciphertext	E	Y	R	L	U	G	L	B	R	U	F	H	T

Once again, Bob is able to reconduct the ciphertext to the plaintext just by the use of the private key he owns. Again, ciphering and deciphering are equivalent tasks fully depending on the private key.
Vigenère's cipher has made many cryptoanalysts struggle a lot in several attempts to break it up, that's why it was considered undecipherable for years (it was even called *"le chiffre*

indèchiffrable"). Anyway, around the XIX century Charles Babbage and Friedrich Wilhelm Kasiski were able, independently, to break it.

Example 1.4. (Advanced Encryption Standard, AES)
AES is the most employed and secure symmetric private key cryptosystem. This cryptosystem officially substituted DES in 2002, after being announced from NIST in 2001: the inventors of AES are Joan Daemen and Vincent Rijmen. It is based on a cipher block, rather than ciphering one letter or character at a time, as seen in the previous examples. This way of ciphering is more secure than the monoalphabetic one.
Describing the math structure and how AES works would take some time, leading us out of our purposes. However we describe its main features since it is involved in the Https protocol and in modern cryptography, working together with elliptic curves cryptography and RSA, as we will show in Chapter 4.
First of all, AES needs a private key. The private keys accepted are the ones of size 128, 192 and 256 bit (corresponding to 16, 24 and 32 characters, respectively, in UTF-8/ASCII). The plaintext is block ciphered, and the dimension of any block is 128 bit. The whole algorithm is a so-called substitution-permutation network (SPN): for any block of 128 bit, there are 10 rounds (12 for 192 bit key and 14 for 256 bit key), and any round is composed by four steps, which correspond to nothing other than linear algebra operations (with the exception of one). The key is employed at any round. At the end of each round there is an output of 128 bit, which at end of the 10 rounds represents the ciphered block.
For further detail on the mathematical aspects of these rounds, we refer to [6].
We just want to stress that AES is a secure and fast algorithm, which becomes extremely secure using a public key algorithm such as elliptic curves cryptography to encrypt the private key. AES_128 is actually widely used for secret communications, such as bank payments and the https protocol. AES_192 and AES_256 are used for top secret communications, such as the intelligence or government channels.

There are two key points that arise at this point:

- *The key distribution issue*: if Alice and Bob want to communicate with private key cryptography, first they must exchange the private key in some way. This is a non-trivial vulnerability of the cryptosystem, since Eve can try to take possess of the key during the exchange or the transport of the key.

- *Symmetry*: the key of enciphering is the same of the key of deciphering: roughly speaking, if we are able to encipher, then so we are to decipher, and the inverse procedure also holds.

There are also other two features that characterized the history of private key cryptography. The first one is that for thousands of years it was a prerogative of governments, royal families and military commands. It is true that it has indirectly decided the lives of millions of people, especially during the First World War (Zimmerman's telegram) and the Second World War (the decryption of the Enigma machine as well of the Porpora machine). But it is only with the development of technology, such as the telegraph first and successively the radio, the cable telephone and internet, that cryptography started to be a matter who

involved directly the population. So, the problem of the key distribution was not as deeply felt as it nowadays is, due to the huge number of keys employed respect to the past.
Another feature of this first period of private key cryptography is that mathematicians and cryptography have started their relationships only "recently". Indeed, very few of the cryptologists around the centuries were mathematicians, and the first strict connection between the two disciplines is traceable to 1926, when Marian Rejewski (1905-1980) brought his crucial math mindset to the cryptographic world.
Rejewski was Polish, and at that ages Poland was probably the best organized state on enciphering administration. This was due to the potential risk of invasion by the German neighbour, which forced Poland to enhance the faculty of deciphering the German messages, which at the time mostly travelled via radio. In that period, Germans had already built the predecessor of Enigma, the famous machine used by German to encrypt all communication (except for high command ones). The first man able to break the Enigma code was Rejeswki, a mathematician; his job has been fundamental to the work made during the second world war by Alan Turing and his team on deciphering the ultimate version of Enigma machine.
The point we want to stress out is that from the 1926 on, math and cryptography have travelled together; in particular cryptography has been almost everywhere a math affair. This joint cooperation is one of the main factors that has led to the birth of public key cryptography, together with the two vulnerabilities of key distribution and symmetry of ciphering/deciphering. The other main factor, as already underlined, is the increased number of people directly involved in the usage of cryptography.

1.2 Public Key

Summer of 1976. Martin Hellman (1945) had one of the most brilliant idea of the 20th century. He caught a glimpse of a stratagem to break the Kerckoffs' law thanks to the idea of a trapdoor function. Despite having had just a theoretical idea, without a pratical result, this is commonly accepted as the moment in which public key cryptography was first invented (James Ellis, who worked at GCHQ in Great Britain, was apparently the first who created the public key idea around 1969, but due to the mandatory on keeping the secret by the English government, this result was published only in 1997, by Clifford Cocks, who collaborated with him in GCHQ).
We can describe the idea of Hellman with a metaphor as follows: Alice wants to send Bob a box and, of course, she wants to keep the content secret. In private key cryptography, Alice and Bob have to meet before exchanging the box, in order to choose a cryptosystem, composed by an algorithm and a key. Suppose that the algorithm is represented by a classical door lock, and the key by, of course, the key that allows to open and close the door lock. Alice gives the key to Bob, she then goes home, prepares the box and lockes it with the padlock. Then she sends the box to Bob, who is the only one that has the key (except Alice, who has a copy of the key) to open the door lock and access the content.
However, this private key cryptosystems has vulnerabilities, as pointed out above. This is why Hellman thought the following: Alice and Bob communicate over an unsecure line (Eve may be listening to them) just to agree on the kind of padlock to use. Alice prepares

the box putting the secret content inside and she locks it with the agreed padlock, keeping the key private. She sends it to Bob, who cannot open it since he does not have the key. He just puts another padlock to it, keeping its key secret too. He then sends back the box to Alice. Now Alice can open the only padlock of which she has the key, her padlock. After this operation, she can send back the box to Bob, who now can open the box closed by nothing but his padlock.
This is a theorical way to communicate safely, but there are still two problems. First, that it is theorical: the padlock system represents a kind of function that is easy to close (to cipher) but really hard to disclose without the key (i.e. really difficult to decipher). Does any such "trapdoor" function exist in practice? We will answer in the affirmative later.
Second, Eve can intercepts the box after the first sending from Alice, lock it with her own padlock and then act as Bob should. How can Alice make sure that she is communicating with Bob?
Again, a theorical answer to this second question was given around 1976 by Whitfield Diffie (1944), who had closely collaborated with Hellman, and gave rise to the idea of asymmetric key cryptography.
Using the metaphor again, Alice wants to send Bob the box. What they do is to communicate on an unsecure line in which Bob gives Alice the kind of padlock and a public key to close it (i.e. the algorithm and the key to encipher), concurrently Bob keeps secret the private key he chose. Alice then closes the box with the public key and sends the package to Bob, who is the only one who can open it. *Le jeux sont fait.*
All the process, and the impossibility for Eve to open the box (deciphering), is based on the possibility to create a cryptosystem that has an easy (and public) way to encipher, and a complicated way to decipher, which becomes simple for the only one who has the deciphering key. This is the so-called key separation. Thus, enciphering and deciphering are two different tasks (involving two different keys, one of them public and the other private) and there does not exist the problem of key distribution.
So, all the argument is interesting as long as we are able to find out a cryptosystem that acts as the padlocks and the couple of keys. Again, the way to achieve this is to look for functions that are easy to compute knowing a enciphering key, with an inverse which is hard to compute without knowing the deciphering key. Such functions are known as trapdoor functions.
In the following we give a more precise definition of what a cryptosystem is, in order to give some examples of public key cryptography.

Definition 1.1. A *cryptosystem* is a bijective enciphering transformation $f : P \longrightarrow C$, where P is the set of all possible plaintext message units and C is the set of all possible ciphertext message units.

Remark 1.1. In practice, one would work with block letters rather than with message units, as in Example 1.4, so in Definition 1.1 P might be of the type P^k for some k positive integer, representing the block of k letters in an N-alphabet (N is the cardinality of the alphabet as usual).

The enciphering transformation f is described, as already explained, by an algorithm and

an enciphering key, K_E. For instance, in Example 1.2 we considered the transformation

$$f : \mathbb{Z}/26\mathbb{Z} \longrightarrow \mathbb{Z}/26\mathbb{Z}, \qquad x \mapsto (x+4) \mod 26.$$

Here the algorithm is exactly f, with $K_E = 4$.
The deciphering transformation is given by the inverse of f, $f^{-1} : C \longrightarrow P$, and it is endowed with a deciphering key K_D.
The theoretical idea of padlocks was realized in practice not longer after Diffie and Hellman's ideas: with this, public key cryptography became concrete. One of the first public key cryptosystem published was RSA (1979) by Rivest, Shamir and Aldelman. Their idea comes from an ancient problem of number theory: finding the factorization of a large integer whose prime factors are not known in advance.

Example 1.5. (RSA)
This is how a communication between Alice and Bob proceeds using RSA.

- Alice randomly chooses two prime numbers, p and q. (This choice is the most time expensive part of the algorithm and requires a primality test: in current practice, we are thinking of numbers of order 10^{100}). These numbers represent the private key.
- Alice then multiplies the two primes obtaining the product $N = pq$. Moreover, she chooses at random another number e such that e and $\varphi(N) = (p-1)(q-1)$ are coprime.
- At this point, Alice publishes the two numbers e, N, which represent the public key ($K_E = (e, N)$) intended to cipher a plaintext.
- Bob takes his message M (of course, he will convert it in UTF-8/ASCII, for instance, to be able to work in digits rather than letters), and ciphers it using the formula: $C = M^e (\mathrm{mod} N)$ (so the enciphering tansformation is $f : \mathbb{Z}/N\mathbb{Z} \longrightarrow \mathbb{Z}/N\mathbb{Z}$, $f(M) = M^e (\mathrm{mod} N)$). He then sends it to Alice.
- Alice receives the ciphertext C, and she's able to decipher it using the formula $M = C^d (\mathrm{mod} N)$, where d is calculated[1] by the Euclidean algorithm (see Appendix B), knowing that $d = e^{-1} (\mathrm{mod}(\varphi(N))$ (so the deciphering key is $K_D = (d, N)$ and the deciphering tansformation is $f^{-1} : \mathbb{Z}/N\mathbb{Z} \longrightarrow \mathbb{Z}/N\mathbb{Z}$, $f^{-1}(C) = C^d (\mathrm{mod}(N)))$.
- Eve is able to cipher (she can intercept e, N), but she is in big trouble on deciphering: she does not know the value of either p or q and she can only try to calculate d by factorizing N, which is a complex operation to do. The exponential functions in modular arithmetics are a kind of trapdoor function, i.e. they are unidirectional. Easy to compute, but with an inverse hard to compute.

Example 1.6. (RSA for signature)
RSA may also be employed for authentication, a matter we discussed in Diffie's metaphor. Imagine, as in the Example 1.5, that Alice has generated her two keys, the public key

[1] Concretely, via the (Extended) Euclidean algorithm we get $ex + \varphi(N)y = \gcd(e, \varphi(N)) = 1$ for certain $x, y \in \mathbb{Z}$. This allow to compute the inverse d of e by computing $d := x = e^{-1} \mod \varphi(N)$.

$K_{E,A} = (e_A, N_A)$ and the private one $K_{D,A} = (d_A, N_A)$. Also Bob has built his keys: $K_{E,B} = (e_B, N_B)$, $K_{D,B} = (d_B, N_B)$. Alice wants to sign a message M allowing Bob to be sure that it comes from her. What Alice does is to compute the composition of her deciphering trasformation and the ciphering transformation of Bob. Using the notation of Example 1.5, this is equivalent to computing $f_B f_A^{-1}(M)$ (only if $N_A \leq N_B$, otherwise she computes $f_A^{-1} f_B(M)$), and sends it to Bob. Bob will just compose $f_A f_B^{-1}(C)$ to obtain the plainsignature of Alice.
We will see later how elliptic curves are widely used for signatures, when discussing ECDSA.

Nowadays RSA is still the most common cryptosystem used for communication. Rather than ciphering whole messages, it is commonly used together with a private key cryptosystem, of which nowadays AES is the best exponent (DES was in the past). The idea of mixing up the two systems takes life in crypting the private key of AES with RSA, and then crypting the whole message with AES. This a secure way of communication, under certain constraints (such as key length, as we will see).
However, since our target is elliptic curve cryptography (ECC), we anticipated that the same enciphering system with AES can be played with ECC in place of RSA. Moreover, we will underline the advantages that ECC has with respect to RSA in Chapter 4.
But first, the next chapters are devoted to showing how it is possible to build a cryptosystem using elliptic curves theory; furthermore, we will describe the algorithms of elliptic curve cryptography as well as the algorithms that warn on the possiblity to breaking an ECC cryptosystem under certain conditions.

2 Elliptic curves

The aim of this chapter is to introduce some of the theory of elliptic curves. Elliptic curves play a role in different areas of mathematics: for instance, are widely used in number theory, have a key role in the proof of Fermat's last theorem and are useful in algorithms of factorization and primality test.
Since our target is elliptic curves cryptography, we will limit ourself to a summary of the part of the theory that is helpful understanding it.

2.1 Definition and Addition Law

Let us start with the general definition of elliptic curves. We recall basic definitions and results for groups and fields in Appendix A.

Definition 2.1. Let K a field.
If the characteristic of the field K is $p \neq 2$, an *elliptic curve*[2] $E(K)$ over K is the set of points

$$E(K) = \{(x, y) \in K^2 : y^2 = x^3 + ax^2 + bx + c\} \cup \{\mathcal{O}\}$$

satisfying the condition

$$\Delta_E := -4a^3c + a^2b^2 + 18abc - 4b^3 - 27c^2 \neq 0,$$

with $a, b, c \in K$.
Generally, if the characteristic of the field K is $p = 2$, then an *elliptic curve* $E(K)$ over K is the set of points

$$E(K) = \{(x, y) \in K^2 : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\mathcal{O}\}$$

where $a_i \in K$ for $i \in \{1, 2, 3, 4, 6\}$ and satisfying a more complicated condition on Δ_E that will be introduced later.

In Definition 2.1 there are many points that need to be clarified.

Remark 2.1. Elliptic curves can be defined over different fields, such as $\mathbb{C}, \mathbb{R}, \mathbb{Q}$, but we will focus our attention on finite fields, since cryptographic algorithms are based on them. Thus our main interest is centered on the finite fields $K = \mathbb{F}_q$, where $q = p^k$ is a prime power, i.e. p is a prime and $k \geq 1$ (for further details on finite fields, look at Appendix A). Moreover, we can focus on the case $p \geq 3$, so that we can restrict our interest to elliptic curves described by the equation:

$$y^2 = x^3 + ax^2 + bx + c \tag{1}$$

We observe that (1) can also be rewritten as:

[2]Elliptic curves are not ellipses. This name comes from the research of a method to compute the arc length of an ellipse by integrals. In particular, to compute arc length one integrates a function of the form $y = \sqrt{f(x)}$, where f may be a cubic or quartic polynomial, so that the integrand satisfies an equation of the form $y^2 = f(x)$.

$$y^2 = x^3 + bx + c \tag{2}$$

The equation (2) can be obtained from (1) by the transformation $x \to (x - \frac{1}{3}a)$, with the only issue that there is a number 3 at denominator. That is, we are required to exclude also fields K with characteristic $p = 3$ for elliptic curves described as in (2).
Summarizing, we will use (1) for characteristic $p \neq 2$ rather than (2) for characteristic $p \neq 2, 3$.
We underline that we prefer to give the general definition, including the case of characteristic 2, not just for the sake of completeness but rather because characteristic 2 finite fields were suitable too for elliptic curves cryptography until recently. We will postpone a discussion of the special case $K = \mathbb{F}_{2^k}$ to a later section (3.4.1) , where we will also introduce the condition on Δ_E in the case $p = 2$.

Remark 2.2. What is the meaning of the condition
$\Delta_E := -4a^3c + a^2b^2 + 18abc - 4b^3 - 27c^2 \neq 0$ for characteristic $p \neq 2$?
The polynomial Δ_E is called the *discriminant* of E and we are requiring it to be nonzero. This is equivalent to require the roots of $f(x) = y^2$ to be distinct, where f is the cubic monic polynomial on the right hand side of (1) (we will prove it after these remarks).
The condition that the cubic f does not have multiple roots is equivalent to the fact that every point on the curve $E(K)$ is non-singular. So, by assuming $\Delta_E \neq 0$ we are avoiding all the degree three curves with singular points (which are the ones with $\Delta_E = 0$).
Geometrically, a curve being non-singular means that there exists a tangent line at every point of the curve.

Remark 2.3. The last point to clarify is the nature of the added point $\mathcal{O}$. Its presence is due to an issue: in general, given an elliptic curve over $\mathbb{K}$, there is no known method to establish whether the curve has any rational point (such a method does not exist for cubic in general, while it exists for conics: it is due to a Legendre's theorem, which was generalized by Hasse). However, assuming the existence of a rational point ($\mathcal{O}$), we can make the set of rational points into a group with the addition operation we'll soon introduce, in such a way that $\mathcal{O}$ becomes the identity element.
In practice, $\mathcal{O}$ is introduced by considering the projective plane $\mathbb{P}^2$, that is the set of all equivalence classes $[X, Y, Z]$[3]. We have a correspondence between the complement $\mathbb{P}^2 \setminus \{Z = 0\}$ and the affine space K^2 using the map $p : [X, Y, Z] \to (\frac{X}{Z}, \frac{Y}{Z})$. Thus the projective plane is identified with the affine plane (x, y) together with the projective points with $Z = 0$, which constitute so-called *line at infinity*. Now, writing the preimage of the elliptic curve (2) under the correspondence p, yields the equation:

$$\left(\frac{Y}{Z}\right)^2 = \left(\frac{X}{Z}\right)^3 + a\left(\frac{X}{Z}\right) + b,$$

[3]These are defined as

$$[X, Y, Z] = \{(X', Y', Z') \in K^3 \setminus (0, 0, 0) : \text{ there exists } \lambda \in K, \lambda \neq 0 \text{ such that } \lambda(X', Y', Z') = (X, Y, Z)\}.$$

and multiplying both sides by Z^3 we get:

$$E : Y^2Z = X^3 + aXZ^2 + bZ^3. \tag{3}$$

We wish to determine the intersection of the line at infinity with the projective curve (3). Hence, setting $Z = 0$, we get that $X^3 = 0$, so $X = 0$ with multiplicity three. We find the projective point [0,1,0], which corresponds to the point we called $\mathcal{O}$ in Definition 2.1. In practice, $\mathcal{O}$ is thought to be the third point of intersection of every vertical line with the elliptic curve; we will prove the existence of such a third point of intersection in Proposition 2.2, in which we will also explain why it is related with vertical lines. Notice moreover that the line at infinity $Z = 0$ meets the elliptic curve with multiplicity 3 at $\mathcal{O}$ due to the multiplicity three of the root $X = 0$. This is an important point in what will follow.

The previous remarks remind us that we are aiming to study elliptic curves defined over finite fields, which are non-singular and with a given rational point on the curve ($\mathcal{O}$). Before introducing a binary operation on elliptic curves, we prove the geometric interpretation of the discriminant condition as promised, as well as the existence of a third point of intersection between an elliptic curve and a line passing though two points of the curve.

Proposition 2.1. Let

$$f(x, y) = y^2 - x^3 - ax^2 - bx - c = 0$$

be the equation for a cubic curve $E(K)$ and set $g(x) = x^3 + ax^2 + bx + c$. Then:
(1) the roots of $g(x)$ are distinct if and only if $\Delta_E \neq 0$.
(2) E is non-singular if and only if $\Delta_E \neq 0$.

Proof. (1) g has three distinct roots $\alpha_1, \alpha_2, \alpha_3$ if and only if g can be written in the form:

$$g(x) = (x - \alpha_1)(x - \alpha_2)(x - \alpha_3).$$

Writing g as above, the discriminant takes the following form:

$$\Delta_E = (\alpha_1 - \alpha_2)^2(\alpha_1 - \alpha_3)^2(\alpha_2 - \alpha_3)^2$$

This form allows to immediately conclude the the discriminant is non-zero if and only if the roots of $g(x)$ are distinct.
(2) First we show that the point $\mathcal{O}$ is never singular. Rewriting the curve E in $\mathbb{P}^2$ in homogeneous coordinates

$$F(X, Y, Z) = Y^2Z - X^3 - aX^2Z - bXZ^2 - cZ^3 = 0,$$

we differentiate along the Z-coordinate:

$$\frac{\partial F}{\partial Z} = Y^2 - aX^2 - 2bXZ - 3cZ^2,$$

which takes the following value in $\mathcal{O} = [0, 1, 0]$:

$$\frac{\partial F}{\partial Z}(\mathcal{O}) = 1 \neq 0$$

proving that $\mathcal{O}$ is a non-singular point for an elliptic curve E.
Let's now prove (2) for all the other points.
Assume E is singular at the point $P = (x_P, y_P)$. The translation

$$x = x' + x_P, \qquad y = y' + y_P,$$

leaves Δ_E invariant. So, without loss of generality, we can consider our singular point to be $(0, 0)$. We get:

$$c = f(0, 0) = 0, \quad b = \frac{\partial f}{\partial x}(0, 0) = 0, \quad \frac{\partial f}{\partial y}(0, 0) = 0$$

so we conclude that (no matter what a is):

$$\Delta_E := -4a^3c + a^2b^2 + 18abc - 4b^3 - 27c^2 = 0$$

Conversely, assume now that E is non-singular; we want prove $\Delta_E \neq 0$. Given that

$$\frac{\partial f}{\partial y} = 2y, \qquad \frac{\partial f}{\partial x} = 3x^2 - 2ax + b,$$

we recall that the curve E is singular if and only if there exists a point $(x_0, y_0) \in E$ such that:

$$2y_0 = 3x_0^2 + 2ax_0 + b = 0.$$

This is saying that singular points are of the form $(x_0, 0)$, with x_0 a double root of $g(x)$. But in point (1) we have seen that if $g(x)$ has a double root then the discriminant Δ_E vanishes, so we are done.

□

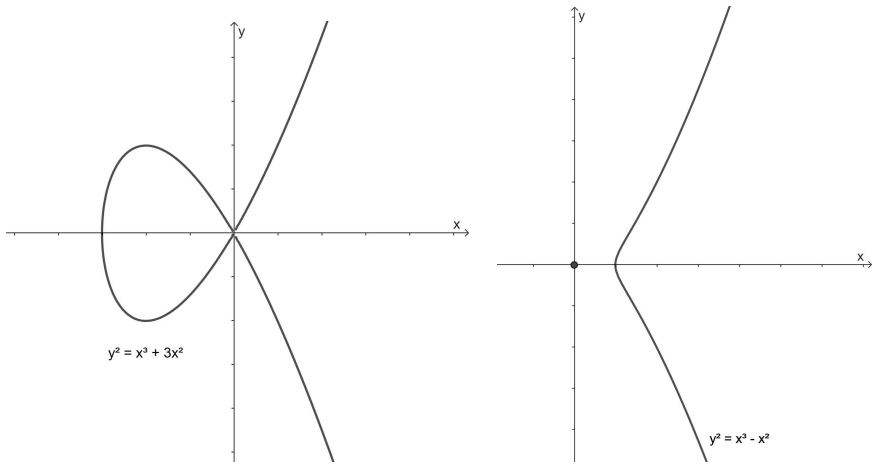


Figure 2: Two cases of singular cubic curves. These kind of curves are not used in elliptic curve cryptography.

Proposition 2.2. Let E be an elliptic curve over the field K given by the equation:

$$y^2 = x^3 + ax^2 + bx + c,$$

and let P and Q two arbitrary points of E. A line l through P and Q intersects E also in a third point $R \in E$.

Proof. We write $P = (x_1, y_1)$, $Q = (x_2, y_2)$ and $l : y = mx + q$.
Assume first that $P \neq Q$. Since l passes through P and Q, we immediately obtain the values of m and q in the equation of l:

$$m = \frac{y_2 - y_1}{x_2 - x_1}, \qquad q = y_1 - mx_1.$$

If $x_1 = x_2$ (l is a vertical line), we set $R = \mathcal{O}$ as the third point of intersection of l and E. Otherwise ($x_1 \neq x_2$) we observe that a point of l belongs to the curve E if and only if

$$(mx + q)^2 = x^3 + ax^2 + bx + c.$$

We rewrite the last equation, obtaining

$$x^3 + (a - m^2)x^2 + (b - 2mq)x + (c - q^2) = 0.$$

This is a polynomial of degree three and two of its roots are x_1, x_2 by hypothesis. The third root, x_3, is given by Viète's formula, which relates the sum of the roots of a polynomial with its coefficients:

$$x_1 + x_2 + x_3 = -\frac{(a - m^2)}{1} = m^2 - a.$$

Hence:

$$x_3 = m^2 - a - x_1 - x_2 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 - a,$$

and thus we obtained the x-coordinate of the point $R = (x_3, y_3)$, which belongs both to E and l: it is the third point of intersection of them. For the y-coordinate we have:

$$y_3 = mx_3 + q = mx_3 + y_1 - mx_1 = y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_3 - x_1).$$

It remains to prove the case $P = Q$. In this case we are considering l to be the line tangent E at P. This means that m coincides with the derivative dy/dx calculated on $P = (x_1, y_1)$. Hence:

$$2y\frac{dy}{dx} = 3x^2 + 2ax + b,$$

and thus

$$m = \left.\frac{dy}{dx}\right|_P = \frac{3x_1^2 + 2ax_1 + b}{2y_1}.$$

If $y_1 = 0$, we set $R = \mathcal{O}$ as the third point of intersection of l and E. Notice that this case corresponds again with l being vertical, as in the above case of distinct points having same

x-coordinate. Hence the point $\mathcal{O}$ is the third point of intersection of every vertical line. To conclude, if $y_1 \neq 0$, then we can proceed as above to obtain a third point (x_3, y_3) of intersection of l and E:

$$x_3 = \left(\frac{3x_1^2 + 2ax_1 + b}{2y_1}\right)^2 - 2x_1 - a,$$

$$y_3 = y_1 + \left(\frac{3x_1^2 + 2ax_1 + b}{2y_1}\right)(x_3 - x_1).$$

□

Now we are ready to introduce the operative heart of elliptic curves: the *Addition law*, which leads to a group structure for elliptic curves. Let us give the geometrical flavor of this operation by the so-called chord and tangent rule for group law.
Let P and Q be two points on an elliptic curve E: we consider the line passing through P and Q, which intersects E in a third point R (the existence of such a third point was proved in Proposition 2.2). We take the reflection of R across the x-axis. What we get is a point R' on E (since equation (1) is even in y) that represents exactly the sum of P and Q:

$$P + Q := R'.$$

Of course, it may happens that $P = Q$ (i.e. we would compute $P + P = 2P$), in which case we consider the line passing twice through P, which is nothing else than the tangent line to E passing through P: this line exists thanks to the non-singularity condition in the definition of elliptic curves. Moreover, if we try to add two points that have the same x-coordinate and opposite y's, then the issue of finding a third point is solved by the extra point $\mathcal{O}$, because two point of this kind would lie in the same vertical line, which intersects the curve at infinity. At the end, we just pointed out above the case of summing $\mathcal{O} + \mathcal{O}$, which gives again $\mathcal{O}$. In Figure 3 is sketched how the sum of two points behaves.
In this construction, we can define the point $-P$ to be the point with same x-coordinate of P and opposite y-coordinate (if $P = \mathcal{O}$, then $-P = \mathcal{O}$).

We put all the reasoning into a definition.

Definition 2.2. Let E be an elliptic curve over K, let $P, Q \in E$ and l the line through P and Q (if $P = Q$, l will be the line tangent to E at P). Let R the third point of intersection of l with E and l' the line through R and $\mathcal{O}$. The third point of intersection of l' with E is the point we denote by $P + Q$.

Remark 2.4. We use the symbol "+" for the addition law defined over points of an elliptic curve, which is the same symbol used for the usual additive operation on numbers. Anyway, the context will always make it clear in which environment we are adding.

We have an immediate consequence from Definition 2.2: if l intersects E at the points P, Q, R, then $(P + Q) + R = \mathcal{O}$. Indeed, $P + Q$ is the third point of intersection of the line l' passing through R and $\mathcal{O}$, so the sum of $P + Q$ and R is the third point of intersection of the line l'' passing through $\mathcal{O}$ and $\mathcal{O}$. This point is again $\mathcal{O}$.

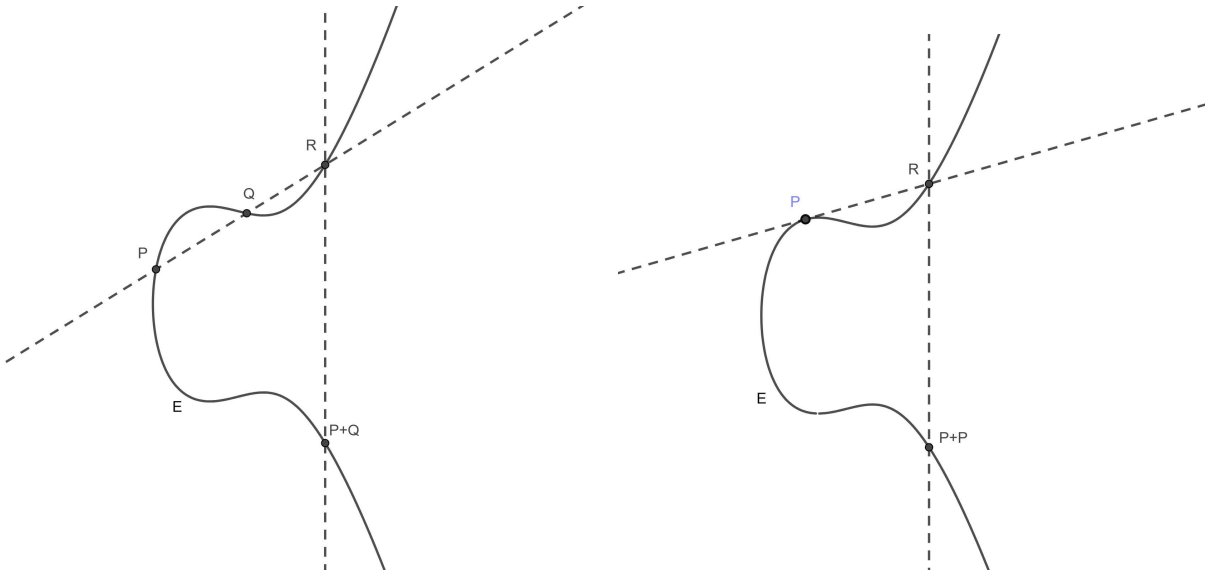


Figure 3: Sum of two points of an elliptic curve. The case of distinct points P, Q is on the left while on the right there is the sum $P + P = 2P$.

The following theorem shows that the elliptic curves endowed with "+" have an abelian group structure.

Theorem 2.1. Let E an elliptic curve over K. The following properties hold for all $P, Q, R \in E$.
(a) $P + \mathcal{O} = \mathcal{O} + P = P$.
(b) There exists a point $-P \in E$ such that $P + (-P) = \mathcal{O}$.
(c) $(P + Q) + R = P + (Q + R)$.
(d) $P + Q = Q + P$.

Proof. (a) We consider the line l through P and $\mathcal{O}$. If $P = \mathcal{O}$, then we have seen that $\mathcal{O} + \mathcal{O} = 2\mathcal{O} = \mathcal{O}$, so we are done. Otherwise, let P' be the third point of intersection of l and E. By Definition 2.2 of "+", the sum $P + \mathcal{O}$ is given by the third point of intersection of E with the line l' through P' and $\mathcal{O}$. So we have $l = l'$, meaning that this third point is exactly P. So $P + \mathcal{O} = P$.
(b) Consider the line l through P and an arbitrary point $Q \in E$. Then l intersects E also at a third point R'. Hence:

$$\mathcal{O} = (P + \mathcal{O}) + R' = P + R',$$

where we used an immediate consequence of Definition 2.2. The point R' satisfies the conditions to be the desired inverse of P (we denote it $-P$).
(c) We postpone the proof of the associativity property at the end of the present chapter (see 2.5), since it takes a lot of work.
(d) The property follows from the fact that the line through P and Q coincides with the line through Q and P.

□

We have proved that an elliptic curve, endowed with the addition law, is an abelian group,

with identity element $\mathcal{O}$. Next, we will write down explicit formulas for the addition law. These formulas give the most efficient way to compute the sum of two points and so are implemented in every algorithm involving elliptic curves.

Definition 2.3. Let E an elliptic curve over K given by the equation

$$y^2 = x^3 + ax^2 + bx + c, \tag{4}$$

with $a, b \in K$ and let $P = (x_1, y_1)$, $Q = (x_2, y_2)$ be two distinct points of E.
(a) If $Q = \mathcal{O}$ then $P + Q = P$; simmetrically, if $P = \mathcal{O}$ then $P + Q = Q$.
(b) If $x_1 = x_2$ and $y_1 = -y_2$, then $P + Q = \mathcal{O}$.
(c) Otherwise, we can define λ by

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1}, & \text{if } P \neq Q \\ \frac{3x_1^2 + 2ax_1 + b}{2y_1}, & \text{if } P = Q \end{cases}$$

getting the following addition formula:

$$P + Q = (\lambda^2 - a - x_1 - x_2, \lambda(-\lambda^2 + a + 2x_1 + x_2) - y_1).$$

Proof. Statements $(a), (b)$ follow directly from Theorem 2.1. For point (c) we give a sketch of a proof. Let $y = \lambda x + \nu$ be the equation for the line through $P = (x_1, y_1)$ and $Q = (x_2, y_2)$. First we consider the case of two distinct points $P \neq Q$.
We rewrite λ, ν thanks to the knowledge of P, Q:

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}, \qquad \nu = y_1 - \lambda x_1 = y_2 - \lambda x_2.$$

To calculate the third point of intersection of E and l we substitute the equation of l into the equation of E:

$$y^2 = (\lambda x + \nu)^2 = x^3 + ax^2 + bx + c,$$

that is,

$$x^3 + (a - \lambda^2)x^2 + (b - 2\lambda\nu)x + (c - \nu^2) = 0.$$

Let now $\alpha_1, \alpha_2, \alpha_3$ be the three roots of the latter cubic equation. These are exactly the x-coordinates of the three points $E \cap l$: so two of them are x_1, x_2. Let $\alpha_3 = x_3$ be the third one. We rewrite the last cubic as:

$$x^3 + (a - \lambda^2)x^2 + (b - 2\lambda\nu)x + (c - \nu^2) = (x - x_1)(x - x_2)(x - x_3),$$

hence

$$\begin{aligned} x^3 + (a - \lambda^2)x^2 + (b - 2\lambda\nu)x + (c - \nu^2) &= (x - x_1)(x - x_2)(x - x_3) \\ &= x^3 + x^2(-x_1 - x_2 - x_3) + \dots \\ &\dots + x(x_1x_2 + x_1x_3 + x_2x_3) - x_1x_2x_3. \end{aligned}$$

Thus we get x_3 by matching the second degree related coefficients:

$$x_3 = \lambda^2 - a - x_1 - x_2,$$

while y_3 is given by substituting x_3 in the equation of l:

$$y_3 = \lambda x_3 + \nu = \lambda(\lambda^2 - a - x_1 - x_2) + y_1 - \lambda x_1.$$

We conclude that the explicit formula of the sum of two distinct points on an elliptic curve is as in Definition 2.3, setting $-y_3$ for the y-coordinate. It remains to prove the case of adding up two equal points $P + P = 2P$, using the tangent line of E at P. Writing the equation (4) as $y^2 = g(x)$ we get λ by differentiating g:

$$\lambda = \left.\frac{dy}{dx}\right|_P = \frac{g'(x_1)}{2y_1} = \frac{3x_1^2 + 2ax_1 + b}{2y_1}$$

and then we proceed exactly as in the above case of distinct points. □

We conclude this section by recalling an important result, Mordell's theorem. It states that given an elliptic curve over K, then the abelian group $E(K)$ is finitely generated. For its proof, see [20].

Theorem 2.2. (Mordell)
Let K a number field and $E(K)$ an elliptic curve. Then the group $E(K)$ is finitely generated.

2.2 Finite fields

When considering an elliptic curve over the finite field K, we can no longer visualize the curve and apparently all the previous work of chord-and-tangent constructions does not make sense. For instance, while we previously drew some "continuous" curves, elliptic curves over finite fields look like in the Figure 4 below.

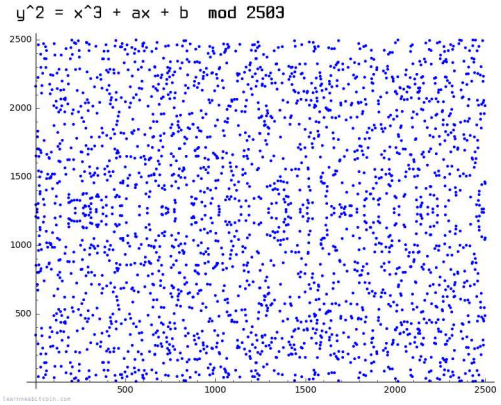


Figure 4: An elliptic curve over a finite field.

We overcome this doubt by noticing that we do not really need any pictures, due to the fact that the explicit formulas work well and make sense over any field, when taking coefficients and coordinates for calculating the sum of points over the finite field.

After this consideration, we give a useful definition: the order of a point of an elliptic curve.

Definition 2.4. Let E an elliptic curve over K. The *order* of a point $P \in E(K)$ is the smallest positive integer N such that $NP = \mathcal{O}$. If such integer does not exist, then we set $N = \infty$. Sometimes we will denote by $E[N] = \{P \in E : P \text{ has order } N\}$ the set of points of E having order N.

Now let E be an elliptic curve over $\mathbb{Q}$. There is a subgroup ϕ of $E(\mathbb{Q})$ called the *torsion subgroup of E*,

$$\phi = \{P \in \ E(\mathbb{Q}) : P \text{ has finite order } \} \cup \mathcal{O},$$

which is isomorphic to a subgroup of the same elliptic curve over $\mathbb{F}_p$ via the *reduction modulo p map*:

$$\phi \to \mathbb{F}_p, \qquad P \mapsto \tilde{P} = \begin{cases} (\tilde{x}, \tilde{y}) & \text{if } P = (x, y) \\ \tilde{\mathcal{O}} & \text{if } P = \mathcal{O} \end{cases}$$

where here the $\sim$ represents the restriction from $\mathbb{Z}$ to $\mathbb{Z}/p\mathbb{Z}$. We don't get into details, but what we are using behind the reduction modulo p is a famous theorem due to Nagell and Lutz. It states that, given an elliptic curve over $\mathbb{Q}$, every point P of finite order has integer coordinates. That is exactly why we can take the restriction $\sim$ from $\mathbb{Z}$ when considering the reduction modulo p map.
We conclude the section with two other results that will be useful on the algorithmic part of Chapter 3.
The first is Hasse's theorem, which suggests how to compute the cardinality of the set of rational points of an elliptic curve.

Theorem 2.3. [Hasse's Theorem] Let E an elliptic curve over $\mathbb{F}_q$. Then

$$\#E(\mathbb{F}_q) = q + 1 - t_q \qquad \text{where } t_{p_q} \text{ satisfies } \left|t_{p_q}\right| \leq 2\sqrt{q}.$$

The second useful tool is the Weil pairing. We shall introduce it below. First of all, recall that a *rational function* in one variable has the form:

$$f(x) = \frac{a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n}{b_0 + b_1 x + b_2 x^2 + \ldots + b_n x^n}.$$

We can rewrite the rational function $f(x)$ as:

$$f(x) = \frac{a(x - \alpha_1)^{e_1}(x - \alpha_2)^{e_2} \cdots (x - \alpha_r)^{e_r}}{b(x - \beta_1)^{d_1}(x - \beta_2)^{d_2} \cdots (x - \beta_s)^{d_s}},$$

where $\alpha_1, \ldots, \alpha_r$ are the distinct (complex) zeros of $f(x)$ and $\beta_1, \ldots, \beta_s$ are the distinct (complex) poles of $f(x)$. The integers $e_1, \ldots, e_r, d_1, \ldots, d_s$ represent the multiplicities of zeros and poles, respectively.
We introduce the *divisor* of a rational function $f(x)$ to be a "trace" of zeros and poles of $f(x)$, together with their multiplicities:

$$\text{div}(f(x)) = e_1\alpha_1 + e_2\alpha_2 + \ldots + e_r\alpha_r - d_1\beta_1 - d_2\beta_2 - \ldots - d_s\beta_s.$$

In our case of an elliptic curve E and a two variable rational function $f(x, y)$, we are interested in the points of E which vanish $f(x, y)$ (the zeros) and in the points which vanish the denominator of $f(x, y)$ (poles). Thus the divisor will take the form:

$$\mathrm{div}(f(x, y)) = \sum_{P \in E} n_P P,$$

where n_P are the integers representing the multiplicities of zeros and poles.

Definition 2.5. Let E an elliptic curve over $\mathbb{F}_q$ and let $P, Q \in E[N]$. Let f_P and f_Q be two rational functions[4] over E satisfying

$$div(f_P) = NP - N\mathcal{O}, \qquad div(f_Q) = NQ - N\mathcal{O}.$$

The *Weil pairing of P and Q* is the quantity:

$$e_N(P, Q) = \frac{f_P(Q + S)}{f_P(S)} / \frac{f_Q(P - S)}{f_Q(-S)},$$

where $S \in E$ is any point $S \notin \{\mathcal{O}, P, -Q, P - Q\}$.

In practice, it can be shown that the Weil pairing takes a pair of points $P, Q \in E[N]$ and gives an N^{th}-root of the unity, i.e. $e_N(P, Q)^N = 1$.
Moreover, the Weil pairing is bilinear, alternating ($e_N(P, P) = 1$) and nondegenerate ($e_N(P, Q) = 1$ for all Q if and only if $P = \mathcal{O}$). This suffices for our purposes in Chapter 3.

2.3 Double-and-Add algorithm

The following algorithm has a central role in cryptography: basically it permits Alice to encipher efficiently, allowing her to compute nP for a given point P in an elliptic curve E over K and a given $n \in \mathbb{N}$.
The idea behind the algorithm is the following. We write n in binary form:

$$n = n_0 + n_1 2 + n_2 2^2 + \ldots + n_r 2^r, \qquad n_0, \ldots, n_r \in \{0, 1\}, n_r = 1. \tag{5}$$

We then compute the quantities

$$Q_0 = P, \quad Q_1 = 2Q_0, \quad \ldots \quad Q_r = 2Q_{r-1}.$$

In this way at the i-th step we have $Q_i = 2^i P$.
Finally, $nP = n_0 Q_0 + n_1 Q_1 + \ldots + n_r Q_r$.
We write its operative form in the following algorithm.

Algorithm 2.1. [Double-and-Add] Let $P \in E(\mathbb{F}_q)$ and $n \geq 1$.
Set $Q = P$ and $R = \mathcal{O}$.
While $n > 0$ {
if $n \equiv 1 \mod 2$ {set $R = R + Q$}

[4] It is possible to prove that such two rational functions always exist for a divisor $\mathrm{div}(f) = NP - N\mathcal{O}$, with P a point of order N in E. We refer to [9] for the details.

set $Q = 2Q$ and $n = \lfloor n/2 \rfloor$ }
Return the point R which equals nP.

Remark 2.5. We write down the algorithm in the special case $K = \mathbb{F}_q$, $2 \nmid q$, because it makes operatively sense to work over a finite field for the purpose of elliptic curve cryptography: anyway, the algorithm can run also over a general field K.

We can now make a computational comparison. Imagine, as we will see, that Alice has to compute the point nP, given an elliptic curve $E(\mathbb{F}_q)$, a point $P \in E(\mathbb{F}_q)$ and an integer $n \in \mathbb{N}$. If she uses a naive approach she has to compute

$$2P = P + P, \quad 3P = 2P + P, \quad \dots \quad nP = (n-1)P + P,$$

so she would perform $n - 1$ steps, each of theme consisting of the addition of two points. This is infeasible if n is large, as for $n \sim 2^{2048}$.
Instead Algorithm 2.1 computes exactly r steps consisting of doublings (i.e. adding two same points) and (at most) r steps consisting of the addition of points. Since $n \geq 2^r$ (for large values of n this means $r \approx \log_2(n)$) and n_r is supposed equal to 1 (so $n < 2^{r+1}$), then we have at most $2\log_2(n)$ operations consisting in the addition of points.
So we pass from the linear complexity of the naive approach and we gain a logarithmic complexity of the Double-and-Add algorithm, allowing Alice to easily compute nP also for larger values of n.
We notice, moreover, that the addition part of the algorithm requires at most $\log_2(n)$ operations: on average, we have that half of the n_i's would be nonzero, so the expected addition time to spend is $\frac{1}{2}\log_2(n)$ operations, plus $\log_2(n)$ doublings.

Observation 2.1. It is possible to speed up the average time of the Double-and-Add algorithm, which we know to require $log_2(n) + \frac{1}{2}log_2(n) = \frac{3}{2}log_2(n)$ addition operations. The idea is to write n in a ternary expansion rather than binary:

$$n = n_0 + n_1 2 + n_2 2^2 + \ldots + n_r 2^r, \qquad n_0, \ldots, n_r \in \{-1, 0, 1\}, n_r = 1.$$

In this way the average time will be reduced to $\frac{4}{3}\log_2(n)$ steps.
Let us compute in general the number of operations that are saved.
We recall that addition and subtraction have the same complexity as operations on elliptic curves, since $-(x, y) = (x, -y)$. Given this, if we take n large then we write the binary expansion of n as in (5):

$$n = n_0 + n_1 2 + n_2 2^2 + \ldots + n_r 2^r, \qquad n_0, \ldots, n_r \in \{0, 1\}. \tag{6}$$

Then, we consider the first occurrence of two or more consecutive nonzero n_i's. Assuming this happens for a certain integer s, we have:

$$n_s = n_{s+1} = \ldots = n_{s+t-1} = 1, \qquad n_{s+t} = 0.$$

Rewriting equation (6):

$$2^s + 2^{s+1} + \ldots + 2^{s+t-1} + 0 \cdot 2^{s+t} = 2^s(1 + 2 + 4 + \ldots + 2^{t-1}) = 2^s(2^t - 1)$$

and so we are reducing the above equation to:

$$2^s + 2^{s+1} + ... + 2^{s+t-1} = -2^s + 2^{s+t}$$

and we can repeat again the procedure, moving on until equation (6) reaches an expansion with no consecutive nonzero n_i's. This implies that we can always write n in a ternary expansion with at most $\frac{1}{2}r = \frac{1}{2}\lfloor \log_2(n) \rfloor + 1$ of the n_i's nonzero. In complexity, this translate into having $r + 1$ doublings and at most $\frac{1}{2}r$ additions. This is the worst case ($\frac{3}{2}\log_2(n) + 1$ operations), but what happens on average?
If we allow sum and difference of powers of 2, then it is possible to show that most of integers n have an expansion with $\frac{2}{3}$ of the terms being zero. This means that the average cost of operations is of $r + 1$ doublings and $\frac{1}{3}r$ additions, which gives $\frac{4}{3}r + 1$ steps.

2.4 An example: Curve25519

To apply the previous theory, we give an example of a recommended elliptic curve for the aim of cryptography. Any of these curves can be find in [17], paying attention on its use, which is reserved only for digital signatures and key exchange systems (such as ECDSA and ECDH, of which we will talk in Chapter 3).
The elliptic curve is the *Curve25519*. It belongs to the so-called *Montgomery Curves*, a family of curves characterized by the equation

$$by^2 = x^3 + ax^2 + x$$

where $a, b \in \mathbb{F}_q$. A curve of this family is the set of solutions of the above equation plus a point at infinity, and the group law is the one described by the chord and tangent rule.
Curve25519 is defined on $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ with the following parameters:

$$p = 2^{255} - 19, \qquad a = 486662, \qquad b = 1,$$

and now the origin of its name has a reason that makes sense.
There are also other properties that are given for Curve25519 that allow to build over it a cryptographic project. One is a base point used for enciphering (Double-and-Add algorithm) and producing the discrete logarithm problem (which we will soon introduce in the next Chapter):

$P = (9, 43114425171068552920764898935933967039370386198203806730763910166200978582548)$.

Furthermore, there are two parameters related to the cardinality of the elliptic curve E:

$$\#(E) = h \cdot n,$$

where n is a large prime and h a small number. In practice, this corresponds to the existence of a large cyclic subgroup of prime order n, whose so-called cofactor is h.
For the Curve25519 these recommended values are:

$$h = 8,$$

$n = 7237005577332262213973186563042994240857116359379907606001950938285454250989.$

2.4.1 Twisted Edwards curves

In some elliptic curve cryptographic implementations may happens to find a slight different name for Curve25519. For instance, the names *Edwards25519* and *W-25519* are used. Are these the same curves or not? They are not, but they are birationally equivalent one to each other, i.e. they are isomorphic except for a (negligible) finite subset of points.
To illustrate this point, we gathered the classification of families of elliptic curves from [17] in the table below, adding between brackets the curves "25519" to underline to which family they belong to (we don't deal with binary curves here, postponing them to Chapter 3).

Curves Type	**Equation**	**Some features**
Short-Weierstrass form (W25519)	$W_{a,b} : y^2 = x^3 + ax + b$	$a, b \in \mathbb{F}_q$, q odd prime power $4a^3 + 27b^2 \neq 0$
Montgomery (Curve25519)	$M_{A,B} : By^2 = x(x^2 + Ax + 1)$	defined over $A, B \in \mathbb{F}_q$, $A \neq \pm 2$, $B \neq 0$, q odd prime power
Twisted Edwards (Edward25519)	$E_{a,d} : ax^2 + y^2 = 1 + dx^2y^2$	$a, d \in \mathbb{F}_q$, $a, d \neq 0$, a is a square while d is not, q odd prime power
Binary	$y^2 + xy = x^3 + ax^2 + b$	$a, b \in \mathbb{F}_{2^k}$, $b \neq 0$

Except for the case of Twisted Edwards curves, all the families of elliptic curves listed have the group structure induced by the chord and tanget method introduced in Definition 2.2, where the identity element is the point at infinity $\mathcal{O}$. On the contrary, this is not the case of Twisted Edwards curves. Notice that Twisted Edwards curves are singular curves of fourth degree with two singular points. Indeed, writing the equation of $E_{a,d}$ in homogeneous coordinates $[X, Y, Z]$:

$$F(X, Y, Z) := aX^2Z^2 + Y^2Z^2 - Z^2 - dX^2Y^2 = 0,$$

and differentiating F along the X, Y, Z-coordinates, we get:

$$\begin{cases} \frac{\partial F}{\partial X} = 2X(aZ^2 - dY^2), \\ \frac{\partial F}{\partial Y} = 2Y(Z^2 - dX^2), \\ \frac{\partial F}{\partial Z} = 2Z(aX^2 + 2Y^2 - 1). \end{cases}$$

Thus we have two singular points for F, the projective points $[1,0,0]$ and $[0,1,0]$, since both satisfy at the condition:

$$\left.\frac{\partial F}{\partial X}\right|_P = \left.\frac{\partial F}{\partial Y}\right|_P = \left.\frac{\partial F}{\partial Z}\right|_P = 0, \quad \text{for } P \in \mathbb{P}^2.$$

Now, about the group structure of a Twisted Edwards curve $E_{a,d}$, given two elements $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ over $E_{a,d}$, the addition of two points is defined by:

$$P + Q := \Big(\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2}\Big).$$

The particular case $P = Q$ is described by the formula:

$$2P := \Big(\frac{2x_1y_1}{1 + dx_1^2y_1^2}, \frac{y_1^2 - ax_1^2}{1 - dx_1^2y_1^2}\Big).$$

From the formulas above it is clear that the identity element for Twisted Edwards curves is the point (0,1).
Twisted Edward curves have been introduced only recently: they first be introduced in 2007, by Harold Edwards, with the subfamily of *Edwards curves* ($a = 1$ in the table above) and then extended to the Twisted Edwards curves in 2008 by Bernstein, Birkner, Joye, Lange and Peters (see [3]). Despite the fact that these curves appeared just some years ago, they find wide application in elliptic curve cryptography due to some advantages with respect to Weierstrass form curves: for instance, they are faster in both adding and doubling points, thus they are faster in running Algorithm 2.1, making Edwards curves attractive in ECC. The most important example of application is EdDSA, an ECDSA (see Algorithm 3.3) digital signature scheme based on Twisted Edwards curves.
In Definition 2.1 (and in Remark 2.1) we defined elliptic curves as Short-Weierstrass form, or simply in Weierstrass form. At this point one may point out that our Definition 2.1 is not complete, in the sense that there are more elliptic curves other than Short-Weierstrass form, for instance Twisted Edwards curves, having also different group structure. First of all, notice that for Twisted Edwards curves we can use the fact that every quartic plane curve in $\mathbb{P}^2$ with two singular points is birationally equivalent to a plane cubic curve. Hence, without loss of generality, we can concentrate on cubic curves. Then, as anticipated at the beginning of this subsection, we overcome the doubt on Definition 2.1 by a general result concerning cubic curves (we refer to [20] and [21] for the details) which shows that every cubic curve with a rational point can be transformed birationally in a cubic in Weierstrass form. That is, given the equation of a general cubic curve:

$$a_0x^3 + a_1x^2y + a_2xy^2 + a_3y^3 + a_4x^2 + a_5xy + a_6y^2 + a_7x + a_8y + a_9 = 0,$$

this cubic is birationally equivalent to a cubic of the form:

$$y^2 = x^3 + ax^2 + bx + c.$$

This allow us to focus on the curves defined in Definition 2.1 (Weierstrass form) when

talking of elliptic curve cryptography, however paying attention to the fact that the implementations of two birational equivalent curves may bring to different time of execution of a cryptographic task.
Thus, referring to the table above, Montgomery curves are isomorphic to Weierstrass curves and Montgomery curves are birationally equivalent to Edwards curves. For instance, the rational map:

$$p : M_{A,B} \to E_{a,d}, \qquad P = (x, y) \mapsto p(x, y) = \begin{cases} (\frac{x}{y}, \frac{x-1}{x+1}) & \text{if } P \neq \mathcal{O}, (0, 0) \\ (0, 1) & \text{if } P = \mathcal{O} \\ (0, -1) & \text{if } P = (0, 0), \end{cases}$$

is a birational map between two curves $M_{A,B}$, $E_{a,d}$ where $a = (A + 2)/2$ and $d = (A - 2)/B$, when considered together with the inverse:

$$p^{-1} : E_{a,d} \to M_{A,B}, \qquad P = (x, y) \mapsto p^{-1}(x, y) = \begin{cases} (\frac{1+y}{1-y}, \frac{1+y}{(1-y)x}) & \text{if } P \neq (0, \pm 1) \\ \mathcal{O} & \text{if } P = (0, 1) \\ (0, 0) & \text{if } P = (0, -1), \end{cases}$$

where $A = 2(a + d)(a - d), B = 4/(a - d)$.
A concrete example takes place by considering Curve25519 and Edwards25519 (M and E, respectively):

$$M : y^2 = x^3 + 486662x^2 + x, \qquad E : x^2 + y^2 = 1 + \frac{121665}{121666}x^2y^2.$$

We define a birational equivalence between the two curves by

$$p : M \to E, \qquad P = (x, y) \mapsto p(x, y) = \begin{cases} (\frac{\sqrt{486664}x}{y}, \frac{x-1}{x+1}) & \text{if } P \neq \mathcal{O}, (0, 0) \\ (0, 1) & \text{if } P = \mathcal{O} \\ (0, -1) & \text{if } P = (0, 0), \end{cases}$$

$$p^{-1} : E \to M, \qquad P = (x, y) \mapsto p^{-1}(x, y) = \begin{cases} (\frac{1+y}{1-y}, \frac{\sqrt{486664}(1+y)}{(1-y)x}) & \text{if } P \neq (0, \pm 1) \\ \mathcal{O} & \text{if } P = (0, 1) \\ (0, 0) & \text{if } P = (0, -1), \end{cases}$$

Thus Curve25519 is birationally equivalent to Edwards25519 (and hence to curve W-25519): thereby a discrete logarithm problem in either curve model is equally hard.

2.5 Associativity Proof

We sketch here some details of the proof of the associative property for the elliptic curve addition law from [18]. This proof has the advantage to be stated for an arbitrary field K, in comparison with other cases as [12] (requires K infinite: anyway it can be extended to K' finite field by considering a field extension $K' \subset K$). Moreover, this proof is based on linear algebra only (differently from [20], which gives a complete proof based on more

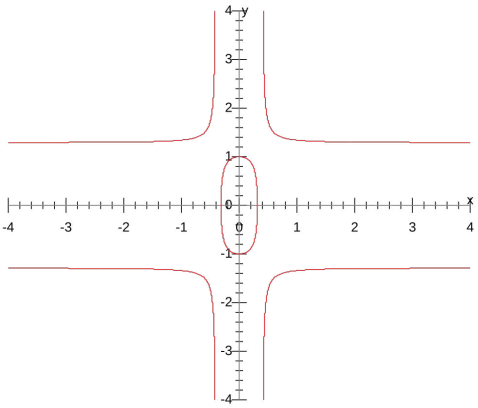


Figure 5: A twisted Edwards curve of equation $10x^2 + y^2 = 1 + 6x^2y^2$.

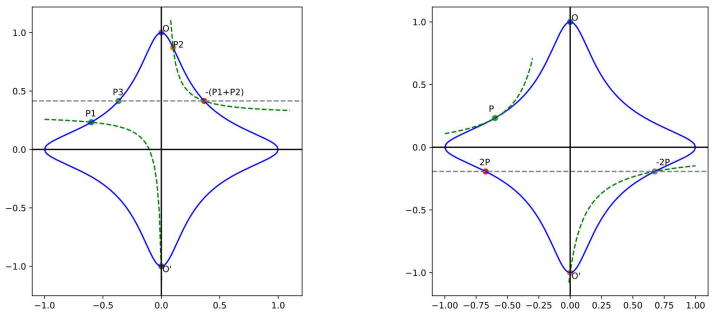


Figure 6: Sum of two points over the Edwards curve $x^2 + y^2 = 1 - 30x^2y^2$: two distinct points $P_1 + P_2 = P_3$ on the left and doubling $P + P = 2P$ on the right. Notice that the straight lines are transformed into conic sections via the birational equivalence p: the green dashed lines represent the transformation of the straight line passing through the two points $p^{-1}(P_1), p^{-1}(P_2)$ of the birational equivalent Weierstrass curve. Thus the points $P_1, P_2, -P_3$ lies on hyperbola.

advanced concepts from algebraic geometry such as Picard groups) and requires no computer-aided calculation (as it is the case of [7]).
We start with some notations used in the proof.
Let K an arbitrary field and E a (non-singular) elliptic curve over K defined in the projective plane $\mathbb{P}^2$ over K (see Remark 2.3) as:

$$E : X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 - Y^2Z - a_1XYZ - a_3YZ^2 = 0$$

with $a_1, a_2, a_3, a_4, a_6 \in K$. Given two points $P, Q \in \mathbb{P}^2$, we denote by $\overrightarrow{PQ}$ the line passing through P and Q (if $P = Q$, we will write $\overrightarrow{PP}$) and we denote $P * Q$ to be the third point of intersection of E and $\overrightarrow{PQ}$ (see Proposition 2.2 for the existence and uniqueness of such a point).
By Definition 2.2 it holds that:

$$-P = P * \mathcal{O}, \quad \text{for } P \in E(K),$$

thus

$$P * (-P) = \mathcal{O}, \quad -(-P) = P \quad \text{for } P \in E(K). \tag{7}$$

Moreover,

$$(P * Q) * P = Q, \quad -(P * Q) = (-P) * (-Q), \quad \text{for } P, Q \in E(K). \tag{8}$$

Therefore, since our commutative operator + is defined by

$$P + Q = -(P * Q), \quad \text{for } P, Q \in E(K)$$

and since it holds that

$$(P + Q) + R = -((-(P * Q)) * R) \overset{(8)}{=} (-(-(P * Q))) * (-R) \overset{(7)(8)}{=} (P * Q) * (-R),$$

$$P + (Q + R) = -((-(R * Q)) * P) \overset{(8)}{=} (-(-(R * Q))) * (-P) \overset{(7)(8)}{=} (R * Q) * (-P),$$

then it is enough to prove the following theorem to conclude that + is associative.

Theorem 2.4. We have

$$(P * Q) * (-R) = (R * Q) * (-P) \quad \text{for every } P, Q, R \in E(K). \tag{9}$$

The proof of Theorem 2.4 is mainly subdivided into three exhaustive steps, two of which are characterized by a case-by-case analysis. All the proof takes into account a list of ten points

$$\mathcal{O},\ P,\ -P,\ R,\ -R,\ Q,\ P * Q,\ R * Q,\ (P * Q) * (-R),\ (R * Q) * (-P), \tag{10}$$

with the aim of showing that the last two are equal, in agreement with the thesis of Theorem 2.4. We briefly summarize how these steps proceed.

Step 1. In the first step, equation (9) of Theorem 2.4 is immediately proved in some particular trivial cases of two points equality. For instance, let us prove Theorem 2.4 when $P = R$ (hence, $-P = -R$ too; see Figure 7). Equation (9) is immediately verified by the substitution $P = R$:

$$(P * Q) * (-P) = (P * Q) * (-P) \quad \text{for every } P, Q \in E(K).$$

Another of these obvious cases is $P = \mathcal{O}$ (see Figure 8). The equality in equation (9) holds for every $Q, R \in E(K)$ by expanding the left hand side and the right hand side of equation (9):

$$(P * Q) * (-R) = (\mathcal{O} * Q) * (-R) = (-Q) * (-R) = -(Q * R).$$

$$(R * Q) * (-P) = (R * Q) * (-\mathcal{O}) = (R * Q) * \mathcal{O} = -(R * Q) = -(Q * R).$$

We refer to [18] for the proof of the other (twentysix) obvious cases of two points coincidence related to the current Step 1. We just mention one of these cases, $R = \mathcal{O}$, useful in Step 3.

Step 2. This step proves Theorem 2.4 for all the remaining cases of at most two points equality (it suffices to consider the case of no three points coincidence of the first nine

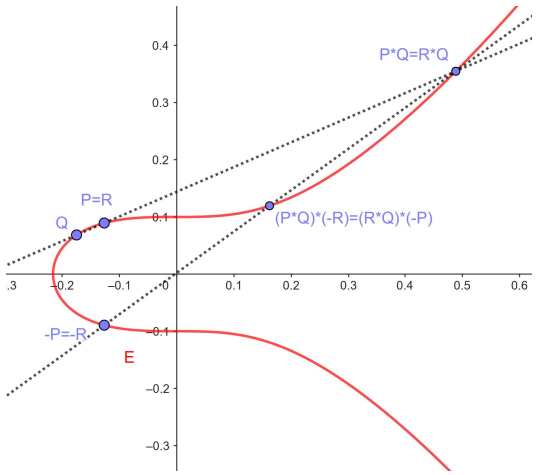


Figure 7: Theorem 2.4 proved in the trivial case $P = R$.

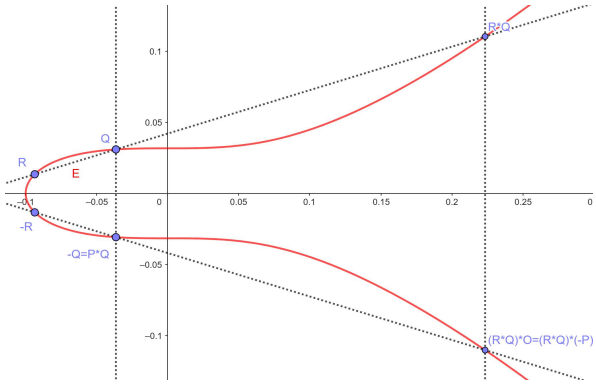


Figure 8: Theorem 2.4 proved in the trivial case $P = \mathcal{O}$. The point $P = \mathcal{O}$ is the third point of intersection of every vertical line with the elliptic curve E, accordingly with Remark 2.3.

points in equation (10), due to the symmetry of Theorem 2.4 for P and R). The idea relies on considering the following three degree homogeneous polynomials (for every P, Q points in $\mathbb{P}^2$, $\overrightarrow{PQ}$ can be write as one degree homogeneous polynomial $AX + BY + CZ = 0$, with $A, B, C \in K$):

$$E,$$
$$F_1 := (\overrightarrow{P - P}) \cdot (\overrightarrow{RQ}) \cdot (\overrightarrow{(P * Q) - R}),$$
$$F_2 := (\overrightarrow{R - R}) \cdot (\overrightarrow{PQ}) \cdot (\overrightarrow{(R * Q) - P}).$$

Thus by definition the first nine points of equation (10) belong to F_1,

$$S_1 := \{\mathcal{O}, P, -P, R, -R, Q, P * Q, R * Q, (P * Q) * (-R)\} \subset F_1, \tag{11}$$

while the first eight points and the last one listed in equation (10) belong to F_2:

$$S_2 := \{\mathcal{O}, P, -P, R, -R, Q, P * Q, R * Q, (R * Q) * (-P)\} \subset F_2 \tag{12}$$

By purely linear algebra arguments it is possible to show that the coefficients of the

equation of F_i is a linear combination of the coefficients of the equations of E, F_j (with $i, j \in \{1, 2\}$, $i \neq j$). This property is directly involved in the proof that, if no three points in the subset S_1 written in (11) (or, symmetrically, subset S_2 in (12)) are equal, than $S_1 = S_2$ and therefore equation (9) of Theorem 2.4 holds. We refer to [18] for the details.

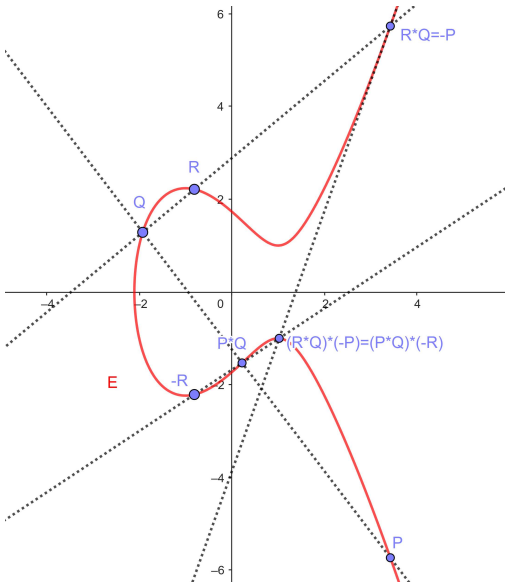


Figure 9: An example of non-trivial case $-P = (R * Q)$ treated in Step 2.

Step 3. The last step considers the remaining cases not treated in Step 1 and Step 2: of course, these are cases of three points coincidence between nine of the points listed in (11) (or, by symmetry, (12)). It turns out that it remains to verify six cases only. We give an example of such a case, referring to [18] the details and the proof of the others cases.

If $-P = (R * Q) = (P * Q) * (-R)$ (hp), then equation (9) is satisfied (see Figure 10). Since in Step 2 we have already treated the proof of (9) in case of no three points equality between the ones listed in the subset S_2 (12), we can assume without loss of generality that three points among the ones listed in (12) are equal. By Step 1 (many trivial cases of two points coincidence reduce the possibilities for three points coincidence) and the hypothesis $(-P = (R * Q) = (P * Q) * (-R))$, the possible coincidences remained to treat are three cases only:

$$P = Q = P * Q, \tag{13}$$

$$-P = R * Q = (R * Q) * (-P), \tag{14}$$

$$-R = P * Q = (R * Q) * (-P). \tag{15}$$

In the first case, we have $-P \overset{\text{hp}}{=} R * Q \overset{(13)}{=} R * P$, thus $-P = R * P$, i.e. $(-P) * P = R$, which implies that $R = \mathcal{O}$. This is one of the trivial cases of Step 1, hence the proof of (9) is reconducted to that case.

The second case is immediate since $(R * Q) * (-P) \overset{(14)}{=} R * Q \overset{\text{hp}}{=} (P * Q) * (-R)$, and

this is exactly equation (9).
The last case gives us $-P \overset{\text{hp}}{=} (P * Q) * (-R) \overset{(15)}{=} (-R) * (-R)$, while $-R \overset{(15)}{=} (R * Q) * (-P) \overset{\text{hp}}{=} (-P) * (-P)$. This leads to $-P = (-P) * (-R) = (-R) * (-R)$. Therefore, we can obtain: $(P * Q) * (-R) \overset{\text{hp}}{=} -P = (-P) * (-R) = -R \overset{(15)}{=} (R * Q) * (-P)$. This concludes the proof of Theorem 2.4.

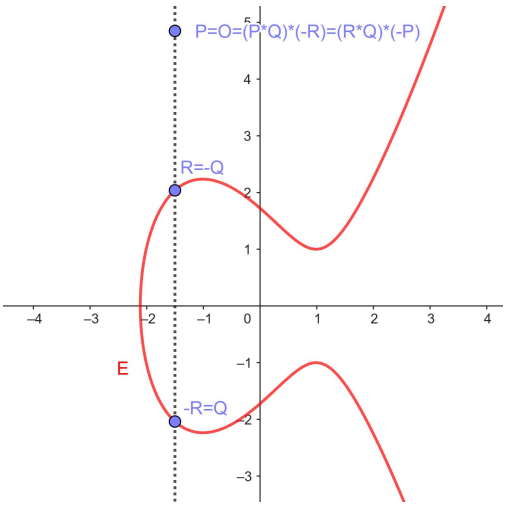


Figure 10: An example of the case $-P = (R * Q) = (P * Q) * (-R)$, one of the remaining cases treated in Step 3 (here P is forced to coincide with $\mathcal{O}$).

3 Elliptic curve Cryptography

We are ready to introduce the use of elliptic curves in cryptography. The aim of elliptic curve cryptography is to encode plaintexts as points on some elliptic curve E defined over a finite field $\mathbb{F}_q$ and make these points indecipherable by taking advantage of the discrete logarithm problem.

3.1 DLP and ECDLP

In Chapter 1 we dwelt on the concept of trapdoor functions and their role in public key cryptography. Here we define one of these functions.

Definition 3.1. Let G be a finite group and $x, y \in G$ be such that y is a power of x. The *discrete logarithm problem* (DLP) is the issue of finding an integer $n \geq 1$ such that

$$x^n = y.$$

We observe that the name "logarithm" comes from the fact that $n = \log_x(y)$.

Example 3.1. The DLP in the additive group $\mathbb{F}_q$ consists of finding an integer n such that $xn = y$, for two given $x, y \in \mathbb{F}_q$. To solve it, we need to find a multiplicative inverse $x^{-1} \in \mathbb{F}_q$ of x, so that $n = x^{-1}x = x^{-1}y$. This can be done for instance by the (Extended) Euclidean algorithm.

Many public key cryptosystems are based on the discrete logarithm problem in $\mathbb{F}_q$. In 1984, Koblitz and Miller, independently, suggested to substitute the group $\mathbb{F}_q$ with an elliptic curve E and the related group $E(\mathbb{F}_q)$.

Definition 3.2. Let E an elliptic curve over $\mathbb{F}_q$ and $P, Q \in E(\mathbb{F}_q)$. The *elliptic curve discrete logarithm problem* (ECDLP) is the issue of finding an integer $n \geq 1$ such that

$$nP = Q.$$

In the next section we will describe some algorithms related with this problem and later we will see how ECDLP is harder to solve compared to the general DLP. This makes ECDLP an interesting working ground for cryptography.

3.2 Cryptographic algorithms via elliptic curves

The following algorithm is intended to be used together with a private key cryptosystem. Indeed, its only aim is exchanging a key, which for instance might represent the private key of a system such as AES (but in practice is not, see Remark 3.3). All of this translate in practice into an exchange of points of an elliptic curve without knowing the base integers of the original addition.

Algorithm 3.1. (Elliptic-curve Diffie-Hellman key exchange, ECDH).
(1) Alice and Bob agree on a finite field $\mathbb{F}_q$, an elliptic curve $E(\mathbb{F}_q)$ and a point $P \in E(\mathbb{F}_q)$ (all these values are public).

(2) Alice chooses a secret integer n_a and computes $Q_a = n_a P \in E(\mathbb{F}_q)$; in the meanwhile, Bob chooses too a secret integer n_b, and he also does a computation: $Q_b = n_b P \in E(\mathbb{F}_q)$.
(3) Alice and Bob send each other the values Q_a and Q_b (this may be done over an insecure channel).
(4) Alice computes $n_a Q_b$, whereas Bob calculates $n_b Q_a$. Now, both Alice and Bob have the point $n_a n_b P$.
This is the key they use to communicate privately via a symmetric cipher. We observe that in this algorithm n_a, n_b represent the private keys, while Q_a, Q_b are the public ones.

What about Eve? She finds herself in need of solving an ECDLP (more precisely, she chooses arbitrarily between two ECDLPs). Indeed, assuming she has been able to intercept Bob and Alice's communications, she ownes the values P, Q_a, Q_b but neither n_a nor n_b. This means that she have to solve an ECDLP in order to find n_a ($n_a P = Q_a$) or to obtain n_b ($n_b P = Q_b$).

Remark 3.1. Notice that Alice has to compute $n_a P$ in order to create her public key (as Bob has to do). To do this, Alice can use the Algorithm 2.1, i.e. via doubling and adding points. As we noticed at the time, the algorithm has logarithmic complexity (we will give a definition of computational complexity in the next section 3.2.1), so it is feasible also for large values of n.

Remark 3.2. What is essential here (and in all of the following algorithms...) is to pick P with an order divisible by a large prime.
Otherwise, there is an algorithm due to Pohlig and Hellman (see Theorem 3.1) which tells that the time of solving ECDLP depends only on the largest prime dividing the order of P. For this argument, it is recommended to use only a point P with prime order in cryptographic applications.

Remark 3.3. As we will see in Chapter 4, in cryptographic applications the recommended elliptic curve key exchange method is the so-called "ECDHE". The last letter "E" stands for *ephemeral*, and it is an attribute for the key. A key is ephemeral if it is generated for each execution of a key-establishment process (for instance, establishing an AES symmetric key for encrypt email messages); also, the ephemeral key must be unique to each message or session. This translate into the fact that an ephemeral key is only used once and then discarded.
The use of ECDHE provides *forward secrecy* (or *perfect forward secrecy*). This property means that if an attacker compromise the private key of one party at a later time, he would not be able to use it to decrypt past communications. Hence, a key exchange via ECDHE generates a temporary ECDH key for every connection, and thus the same key is never used twice.

The following algorithm consists of a elliptic curve public key cryptosystem, who was first proposed for the DLP in $\mathbb{F}_q$ by the Egyptian cryptographer Taher ElGamal in 1985.

Algorithm 3.2. (ElGamal)
(1) Alice and Bob agree on a finite field $\mathbb{F}_q$, an elliptic curve $E(\mathbb{F}_q)$ and a point $P \in E(\mathbb{F}_q)$ (all these values are public).

(2) Alice chooses a secret integer n_a and computes $Q_a = n_a P \in E(\mathbb{F}_q)$.
(3) At this point Alice publishes the point Q_a, which represents her public key, and she keeps secret the value of n_a, her private key.
(4) Bob takes the plaintext $M \in E(\mathbb{F}_q)$ and chooses a random integer n_b. He then computes the two points:

$$C_1 = n_b P, \qquad C_2 = M + n_b Q_a.$$

(5) Bob sends the ciphertext (C_1, C_2) to Alice (Eve may intercepts here).
(6) Alice is now able to rebuild M thanks to her private key: she just computes the point $C_2 - n_a C_1$. Indeed:

$$C_2 - n_a C_1 = (M + n_b Q_a) - n_a n_b P = M + n_b n_a P - n_a n_b P = M.$$

Remark 3.4. In both algorithms above, and in general in any use of elliptic curve cryptography, there is always a trusted third part that has the duty of making available a list of elliptic curves, finite fields and points which are good for cryptographic purposes. We will see in the next sections some special cases of finite fields, curves and points that must be avoided due the existence of algorithms that solve the ECDLP in a feasibile time. Anyway, if one is curious, one of the most advanced complete list is made up by NIST (National Institute for Standards and Technology). This list can be found at [17], and it is last updated in February, 2023.

What about Eve for ElGamal? She might possess the values $P, n_a P, n_b P, M + n_b Q_a$. Again, she needs to solve an ECDLP to achieve the plaintext M.
Now we describe one of the most widely used elliptic curve algorithm. The idea behind it is the same as for the digital signature scheme: it allows Alice to sign a document and Bob to validate the signature (roughly speaking, Bob recognizes the document comes from Alice without doubts).

Algorithm 3.3. (Elliptic Curve Digital Signature Algorithm).
(1) Alice and Bob agree on a finite field $\mathbb{F}_p$, an elliptic curve $E(\mathbb{F}_p)$ and a point $P \in E(\mathbb{F}_p)$ of prime order N.
(2) Alice chooses a secret integer n_a and computes $Q_a = n_a P \in E(\mathbb{F}_p)$.
(3) Alice publishes the point Q_a (the public key).
(4) Alice chooses a digital document $d (\mathrm{mod} N)$ to sign[5], and an integer $k (\mathrm{mod} N)$. Then she computes kP and sets

$$s_1 \equiv x(kP) \ (\mathrm{mod} N), \qquad s_2 \equiv (d + n_a s_1) k^{-1} \ (\mathrm{mod} N).$$

At this point Alice publishes the signature (s_1, s_2) for the document d.
(5) Bob computes $v_1 = d s_2^{-1} \ (\mathrm{mod} N)$ and $v_2 = s_1 s_2^{-1} \ (\mathrm{mod} N)$.
Then he computes $v_1 P + v_2 Q_a$, verifying that $x(v_1 P + v_2 Q_a) \equiv s_1 \ (\mathrm{mod} N)$.

[5] In practice Alice uses a Hash function applied to the document. We will mention Hash functions later in Section 3.3.

The verification for Bob works, indeed:

$$v_1P + v_2Q_a = ds_2^{-1}P + s_1s_2^{-1}n_aP = \underbrace{s_2^{-1}(d + n_as_1)}_{=k} P = kP,$$

hence:

$$x(v_1P + v_2Q_a) = x(kP) \equiv s_1(\mathrm{mod}N)$$

Remark 3.5. Between the lines we didn't mention that, in practice, Alice and Bob do not exchange the whole points of an elliptic curve, but just the x-coordinate of the points. Indeed, if the x-coordinate is known, the other one is obtained by the equation describing E: there are two possible values for the y-coordinate, one the negative of the other. In both cases, the x-coordinate is the same, and exchanging just this value speeds up the process.

3.2.1 Computational complexity of the algorithms

An algorithm is an explicit step by step procedure for doing calculations and solve problems. The efficiency or computational complexity of an algorithm is given by two main factors: the time cost and the space cost. Time cost refers to the speed of execution of an algorithm (it is measured in seconds and its multiples). Space cost concerns the quantity of storage needed to perform the algorithm (measured in byte or bit). There might be various algorithms to solve a given problem; in order to compare the algorithms, we need the following definition of the big O notation.

Definition 3.3. (big O notation)
Let f, g be two functions defined over a subset of $\mathbb{N}^r$. We say that f *is bounded by* g and write $f = O(g)$ if there exist constants B, C such that $f(n) < C \cdot g(n)$ for every $n > \underbrace{(B, \ldots, B)}_{r-times}$. (For these values we require also f, g to be well-defined and positive).

Example 3.2. (1) Let $f(n)$ be a polynomial of degree d, with positive leading coefficient. Then $f(n) = O(n^d)$.
(2) Let $\varepsilon > 0$ (no matter how small it is), then $\log(n) = O(n^\varepsilon)$. Indeed:

$$\lim_{n\to\infty} \frac{\log n}{n^\varepsilon} = 0.$$

(3) Referring to Algorithm 2.1 of double-and-add to compute nP for a point $P \in E(\mathbb{F}_q)$, we have that the algorithm cost is bounded by $O(\log\, n)$ (see Algorithm 2.1).
(4) Let G be a group, $x, y \in G$ and n the order of x. A naive algorithm to solve the DLP "$x^m = y$" is to compute $x, x^2, x^3, ...$ until we get y. This algorithm takes $O(n)$ steps, but has the good feature to use no storage (it is $O(1)$).

In most of the sequel we will focus on the time cost only, thus we will consider the computational complexity of an algorithm to coincide with the time cost. In order to solve a problem, the unit measure to estimate the time of execution of an algorithm is the number of *bit operations* or "steps", which consist of elementary operations at computer/processor level: each bit operation corresponds to the procedure of addition of

two binary digits (bits). The amount of time taken (in seconds) and the number of bit operations performed are related by a constant factor, since every elementary operation requires a small well defined quantity of time to be performed.
Numbers can be written in different bases. Given a base b, any integer n can be thought as a k-digit number respect to the base b: we write $n = (d_0, ..., d_{k-1})$, with $0 \leq d_j < b$ meaning that:

$$n = d_{k-1}b^{k-1} + \ldots + d_1 b + d_0.$$

For our purposes we work with binary digits, therefore $b = 2$. We notice that a k-digit number n satisfies:

$$b^{k-1} \leq n < b^k,$$

hence k can be written in logarithmic form:

$$k = \lfloor \log_b n \rfloor + 1 = \left\lfloor \frac{\log n}{\log b} \right\rfloor + 1.$$

Example 3.3. We can estimate the time employed for multiplying a k-digit integer n and an l-digit integer m (in base 2). Using the well-known classical method, we have to sum at most l rows (the case of exactly l rows happens if and only if m is equal to $\underbrace{111\ldots1}_{l-times}$), and every partial sum of two of these rows requires k bit operations. Hence the time for compute the multiplication is bounded by the following number of bit operations:

$$\text{Time}(n \cdot m) < k \cdot l,$$

thus writing k, l with logarithmic notation we get:

$$\text{Time}(n \cdot m) = O(\log n \cdot \log m).$$

Sometimes the complexity is written referring to the number of digits of a given integer, while in most cases (including ours) the complexity is written explicitely on the value of the integer. Given a k-digit number n as input of a problem, we can talk of an efficient algorithm if it has polynomial complexity: this complexity is $O(k^l) = O(\log n^l)$ for a positive constant l. Instead, a non-efficient algorithm is one that has an exponential complexity $O(2^{k^l}) = O(2^{\log n^l})$, for a positive constant l. An intermediate level of efficiency between the polynomial and exponential complexities is the subexponential complexity: $O(2^{k^\varepsilon}) = O(2^{\log n^\varepsilon})$ for every positive constant ε.

3.3 Algorithms for solving ECDLP

As a spoiler, in general the fastest known algorithm to solve ECDLP in $E(\mathbb{F}_q)$ takes approximately $\sqrt{q}$ steps.
This fastest known method is called *collision algorithm* due to its nature. We give first some useful results, and then we will state the collision algorithm for a general group, keeping in mind that we are going to apply it to $E(\mathbb{F}_q)$.

3.3.1 Index calculus for DLP

Now that we have a way to estimate the efficiency of an algorithm, it's time to give details of the best known algorithm for solving DLP in $\mathbb{F}_q$ in order to operatively compare it with ECDLP. We present it for the case $q = p$ a prime.

Algorithm 3.4. [Index Calculus] We want to solve the problem

$$g^x \equiv h \pmod{p} \tag{16}$$

where p is a prime and $g, h \in \mathbb{F}_p$ are given. For simplicity we assume that the powers of g give all $\mathbb{F}_p^*$, i.e g is a primitive root modulo p.
The idea is, rather than try to solve the DLP (16), to choose a value B and solve instead

$$g^x \equiv l \pmod{p}, \qquad \text{for all primes } l \leq B.$$

After this, we look at the quantities

$$h \cdot g^{-k}, \qquad \text{for } k = 1, 2, \ldots$$

until we find a value of k such that $h \cdot g^{-k} \pmod{p}$ is B-smooth (i.e. all the prime factors of $h \cdot g^{-k} \pmod{p}$ are less or equal B).
So for this value of k we have

$$h \cdot g^{-k} \equiv \prod_{l \leq B} l^{e_l} \pmod{p}.$$

for certain exponents e_l. Rewriting the last equation yields

$$\log_g(h) \equiv k + \sum_{l \leq B} e_l \cdot \log_g(l) \bmod (p-1),$$

where discrete logarithms are defined only modulo $p - 1$.
Assuming we have already computed $\log_g(l)$ for all primes $l \leq B$, we are done since we have found $\log_g(h)$.
To find $\log_g(l)$ we compute

$$g_i \equiv g^i \pmod{p}, \qquad \text{with } 0 < g_i < p.$$

If g_i is not B-smooth, then we discard it, otherwise we can factor it as

$$g_i = \prod_{l \leq B} l^{u_l(i)},$$

which yields to the relation

$$i \equiv \log_g(g_i) \equiv \sum_{l \leq B} u_l(i) \cdot \log_g(l) \bmod (p-1).$$

At this point it is enough to find more equations as the last one to obtain $\log_g(l)$.

We don't get further into details (for more, see [19]). We simply recall that index calculus for solving the discrete logarithm problem in $\mathbb{F}_p$ has a computational complexity that is $\exp(c\sqrt[3]{(\log q)(\log\log q)^2})$, where c is a small constant. This means that index calculus has subexponential running time (we recall from the beginning of the chapter that the amount of time taken and the number of elementary operations performed are related by a constant factor). This stands in marked contrast to the discrete logarithm problem in elliptic curve groups (ECDLP): we anticipated and we will see that the best known algorithms (collision algorithms) to solves ECDLP have computational complexity of $O(\sqrt{q})$ steps, and this coincide with an exponential time of execution. Thus, ECDLP is much harder than general DLP, because the efficiency of the best known algorithm to solve ECDLP is less efficient than the best known method to solve the general DLP.

3.3.2 Pohlig-Hellman

For this result, which we anticipated in Remark 3.2, we assume that the DLP for an element of order a prime power (p^k) can be solved in $O(\sqrt{p^k})$. We will prove this in general in the next section: it is the complexity of the collision algorithm (Algorithm 3.5).

Theorem 3.1. (Pohlig-Hellman for prime-power order groups)
Let G a group and $g \in G$ an element of order N, and suppose N factors into a product of prime powers:

$$N = p_1^{k_1} \cdot p_2^{k_2} \cdot \ldots \cdot p_t^{k_t}.$$

Let $h \in G$. Then the discrete logarithm problem (DLP) $g^x = h$ can be solved in

$$O\Big(\sum_{i=1}^{t} \sqrt{p_i^{k_i}} + \log N\Big)$$

using the following algorithm:
(1) for each $1 \leq i \leq t$ let

$$g_i = g^{N/p_i^{k_i}}, \qquad h_i = h^{N/p_i^{k_i}}.$$

Since g_i has order $p_i^{k_i}$, we can use the collision algorithm (Algorithm 3.5) to solve every DLP for $g_i^{y_i} = h_i$.
So let $y_1, ..., y_t$ be solutions of the corresponding t DLP.
(2) We solve

$$\begin{cases} x \equiv y_1 \pmod{p_1^{k_1}} \\ x \equiv y_2 \pmod{p_2^{k_2}} \\ \ldots \\ x \equiv y_t \pmod{p_t^{k_t}} \end{cases}$$

Proof. The running time comes from step (1), which is made by t substeps, any of these having complexity $O(\sqrt{p_i^{k_i}})$ by the collision algorithm, and step (2), whose complexity is given by the Chinese Remainder Theorem: $O(\log N)$.
Let us show that steps (1) and (2) give a solution to $g^x = h$. Let x be a solution to the

congruences of step (2). Then for each i we can write:

$$x = y_i + p_i^{k_i} q_i, \qquad \text{for some } q_i.$$

So we can compute:

$$\begin{aligned}(g^x)^{N/p_i^{k_i}} &= (g^{y_i + p_i^{k_i} q_i})^{N/p_i^{k_i}} \\ &= (g^{N/p_i^{k_i}})^{y_i} \cdot g^{N q_i} \\ &= (g^{N/p_i^{k_i}})^{y_i} \\ &= g^{y_i} = h_i = h^{N/p_i^{k_i}}.\end{aligned}$$

We can now rewrite this on terms of discrete logarithms to the base g:

$$\frac{N}{p_i^{k_i}} \cdot x \equiv \frac{N}{p_i^{k_i}} \cdot \log_g(h) \pmod{N} \tag{17}$$

where the discrete logarithm to the base g is defined modulo N only since $g^N = 1$.
We observe that the numbers:

$$\frac{N}{p_1^{k_1}}, \frac{N}{p_2^{k_2}}, \ldots, \frac{N}{p_t^{k_t}}$$

have greatest common divisor equal to 1. So, applying repeatedly the Extended Euclidean algorithm (Theorem B.2) we can find integers $c_1, ..., c_t$ such that:

$$c_1 \cdot \frac{N}{p_1^{k_1}} + \ldots + c_t \cdot \frac{N}{p_t^{k_t}} = 1.$$

We multiply both sides of (17) by c_i and sum over i:

$$\underbrace{\Big(\sum_{i=1}^{t} c_i \cdot \frac{N}{p_i^{k_i}}\Big)}_{=1} \cdot x \equiv \underbrace{\Big(\sum_{i=1}^{t} c_i \cdot \frac{N}{p_i^{k_i}}\Big)}_{=1} \cdot \log_g(h) \pmod{N}$$

and this allows to conclude

$$x = \log_g(h).$$

□

3.3.3 Collision algorithms

It's time to state and prove the collision algorithm used in Pohlig-Hellman.

Algorithm 3.5. Let G a group and $x, y \in G$, with n the order of x.
The following algorithm solves DLP in $O(\sqrt{n})$ steps with the use of $O(\sqrt{n})$ storage.
(1) Let N the smallest integer that is greater or equal to $\sqrt{n}$ (i.e. $N - 1 < \sqrt{n} \leq N$).
(2) Make a list of the elements

$$x, x^2, x^3, \ldots, x^N. \tag{18}$$

(3) Let $z = (x^N)^{-1}$ and make another list of elements:

$$yz, yz^2, yz^3, \ldots, yz^N. \tag{19}$$

(4) Look for a match between the two lists (18) and (19). If this happens we have $x^i = yz^j$ for some i, j, and so $y = x^{i+jN}$. If it does not happen, then y is not a power of x.

Proof. Suppose $y = x^m$ with $0 \leq m < n$ and set $m = jN + i$ with $0 \leq i < N$. What about j?
Then $m = jN + i \geq i$, so $m - i \geq 0$. Moreover, $(m - i)/N \leq m/N \leq n/\sqrt{n} = \sqrt{n} \leq N$. Therefore:

$$0 \leq j = (m - i)/N \leq N$$

Hence we have that x^i lies in the first list (18) while $yz^j = yx^{-jN}$ is in the second one (19). So there is a match:

$$y = x^m = x^{i+jN} = x^i(x^{-N})^{-j} = x^i z^{-j},$$

that yields:

$$x^i = yz^j.$$

To find out matches in step (4) requires a sort of the elements $x, x^2, \ldots, x^N$ which takes $O(N \log N)$ steps[6], and then other $O(\log N)$ steps to check whether elements yz^j are in the sorted list. So, the collision algorithm asks $O(\sqrt{n}(\log \sqrt{n})^2)$ steps (recall that $N \approx \sqrt{n}$). Since log is negligible compared to square we conclude it takes $O(\sqrt{n})$ steps. □

We can interpret the theorem in our case of elliptic curves and the related ECDLP, playing ourselves the role Eve plays. Given two points $P, Q \in E(\mathbb{F}_q)$ with N the order of P, we have to make two lists of points:

$$j_1P, j_2P, \ldots, j_NP,$$

$$k_1P + Q, k_2P + Q, \ldots, k_NP + Q.$$

We sort the elements of the first list and then try to have a match with elements of the second list. We are done if we can match $j_uP = k_vP + Q$ for some u, v. Then we have the immediate solution to ECDLP: $Q = (j_u - k_v)P$. The issue for Eve is when the order of the point P is a large prime, which makes such a computation infeasible for her ($O(\sqrt{N})$ steps). There is an issue with Algorithm 3.5: it requires a lot of storage for the two lists. To solve this problem, there is an alternative algorithm with the same number of steps but without almost any need of physical space to storage datas. This algorithm is due to Pollard. We first need a lemma.

Lemma 3.1. Let S a finite set of N elements and $f : S \to S$ a function. We define recursively the following sequence $\{x_i\}_{i \in \mathbb{N}} \subseteq S$:
(1) set x_0 to be any element of S (the initial value of the sequence);

[6]There are several sorting algorithms for ordering an N-array with complexity $O(N \log N)$. For instance, the most well-known algorithm is the *Merge sort*.

(2) for $i \geq 1$, set $x_i = f(x_{i-1}) = \underbrace{f \circ f \circ \ldots \circ f}_{i-times}(x_0)$.

Set furthermore:
$T :=$ largest integer such that x_{T-1} appears only once in $\{x_i\}_{i\in\mathbb{N}}$.
$M :=$ smallest integer such that $x_{T+M} = x_T$.
Then the following properties hold:
(a) There exists an index $1 \leq i < T + M$ such that $x_{2i} = x_i$.
(b) If $f : S \to S$ and its iterates have a sufficiently random behaviour at mixing the elements of S, then the expected value of $T + M$ is $\sqrt{\pi N/2}$.

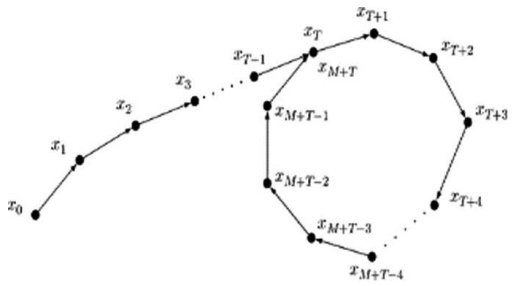


Figure 11: A draw of the sequence $\{x_i\}_{i\in\mathbb{N}}$ explaining the origin of "ρ" in Pollard's algorithm.

Proof. (a) For $j > i$ we have that $x_j = x_i$ if and only if $i \geq T$ and $j \equiv i (\text{mod } M)$.
From this we obtain that $x_{2i} = x_i$ if and only if $i \geq T$ and $i \equiv 0 (\text{mod } M)$, i.e. $M \mid i$. This implies that the first such i satisfies $T \leq i < T + M$.
(b) The proof requires arguments of discrete probability theory. We refer to [20] (section XI.5, theorem 5.3) for any detail. □

Algorithm 3.6. (Pollard's rho algorithm)
Let G a group, $x, y \in G$ and n the order of x in G. We want to solve DLP, i.e. to find an integer m for which $x^m = y$.
Pollard's method uses Lemma 3.1 to look for integers i, j, k, l such that:

$$x^i y^j = x^k y^l$$

so that $x^{i-k} = y^{j-l}$. Assuming $\gcd(j - l, n) = 1$, we obtain $x^{\frac{i-k}{j-l}} = y$ and we are done.
Pollard's idea consists of the choice of the following $f : G \to G$.
Let $G = A \cup B \cup C$, where A, B, C are three disjoint sets with approximately the same size. Then f is defined by:

$$f(z) = \begin{cases} xz & \text{if } z \in A \\ z^2 & \text{if } z \in B \\ yz & \text{if } z \in C \end{cases}$$

We set the initial value of the sequence described in Lemma 3.1 as $z_0 = 1$. Recursively, we obtain at any $i \geq 1$:

$$z_i = \underbrace{f \circ f \circ \ldots \circ f}_{i-times}(z_0) = x^{a_i} y^{b_i}$$

where a_i and b_i are integers satisfying the following iterative formulas (after having set $a_0 = b_0 = 0$):

$$a_{i+1} = \begin{cases} a_i + 1 & \text{if } z_i \in A \\ 2a_i & \text{if } z_i \in B \\ a_i & \text{if } z_i \in C \end{cases}$$

$$b_{i+1} = \begin{cases} b_i & \text{if } z_i \in A \\ 2b_i & \text{if } z_i \in B \\ b_i + 1 & \text{if } z_i \in C \end{cases}$$

We recall that n is the order of x in G, so all the a_i, b_i are considered modulo n.
Similarly, we can repeat the above procedure with another sequence of points:

$$w_0 = 1, \qquad w_{i+1} = f(f(w_i))$$

and hence

$$w_i = z_{2i} = x^{c_i} y^{d_i}$$

where again c_i and d_i are integers satisfying the following iterative formulas (once have set $c_0 = d_0 = 0$, we just write down the case of the c_i's, for the d_i's proceed in the same way by symmetry):

$$c_{i+1} = \begin{cases} c_i + 2 & \text{if } w_i \in A, f(w_i) \in A \\ 2(c_i + 1) & \text{if } w_i \in A, f(w_i) \in B \\ c_i + 1 & \text{if } w_i \in A, f(w_i) \in C \text{ or } w_i \in C, f(w_i) \in A \\ 4c_i & \text{if } w_i \in B, f(w_i) \in B \\ 2c_i + 1 & \text{if } w_i \in B, f(w_i) \in A \\ 2c_i & \text{if } w_i \in B, f(w_i) \in C \text{ or } w_i \in C, f(w_i) \in B \\ c_i & \text{if } w_i \in C, f(w_i) \in C \end{cases}$$

Now we compute $(z_1, w_1), (z_2, w_2), (z_3, w_3), \ldots$ until we find a pair with same coordinates. The key point here is that every couple (z_{i+1}, w_{i+1}) is computed just using the previous couple (z_i, w_i), as seen in the above procedure, so the need of storage is approximately null: compared to Algorithm 3.5, it is reduced from $O(\sqrt{n})$ to $O(1)$.
We observe that point (b) of Lemma 3.1 tells us that, assuming that A, B, C are good enough at mixing elements of G (and so f is sufficiently random too), we will match $z_i = w_i = z_{2i}$ in $O(\sqrt{n})$ steps. Going on from the (eventual) equality $z_i = w_i$ we get the announced result:

$$x^{a_i - c_i} = y^{d_i - b_i}$$

from which we can find the integer m that solves the DLP:

$$m \equiv (a_i - c_i)(d_i - b_i)^{-1} \pmod{n}.$$

We underline again that to find m requires $\gcd(d_i - b_i, n) = 1$. But, as explained in Remark 3.2 and confirmed by Theorem 3.1, this is always the case in cryptographic applications, in which the order of the point is taken prime.

We have just seen the reason why elliptic curves are used in cryptography. Indeed, at present, and despite since the mid 1980s there have been various attempts to improve the extimates of collision algorithms, there are no index calculus algorithms known for ECDLP or other faster methods, that is, there are no known algorithms to solve ECDLP in fewer than $O(\sqrt{q})$ steps.
The next section is devoted to show some fastest algorithms that exist in some particular subcases (subcases one needs to avoid for cryptography).

3.4 Solving ECDLP in special cases

Although the collision algorithm is the fastest known method to solve ECDLP, there are particular elliptic curves that are studied as special cases. In some of these special cases there are algorithms that reduce the difficulties of ECDLP making the problem easier, rather than other "positive" cases that allow to increase the powerness of ECDLP (here positive is meant for cryptographers). This section is devoted to some of these special cases, in order to avoid or take advantage of them.
The first case is called MOV algorithm (Menezes, Okamoto, Vanstone): using the Weil pairing we reduce ECDLP to DLP in $\mathbb{F}_q$.

Definition 3.4. Let $\mathbb{F}_q$ be a finite field and $N \geq 1$ an integer. The *embedding degree of* N in $\mathbb{F}_q$ is the smallest integer $d \geq 1$ such that the group of N^{th} roots of unity is contained in $\mathbb{F}_{q^d}^*$.
The order of the cyclic group $\mathbb{F}_{q^d}^*$ is $q^d - 1$, so the definition is equivalent to $N \mid q^d - 1$.

Proposition 3.1. (MOV algorithm)
Let E be an elliptic curve over $\mathbb{F}_q$, $P, Q \in E(\mathbb{F}_q)$ be two points having prime order N and d be the embedding degree of N in $\mathbb{F}_q$. Assume that $\gcd(q-1, N) = 1$. Then there is an algorithm that reduces ECDLP for P, Q to DLP in $\mathbb{F}_{q^d}^*$.

Proof. We are looking for an integer m such that $mP = Q$. We take a point $T \in E$ with order N such that P, T generate $E[N] = \{$ points in E of order $N\}$. Then, by definition of the Weil pairing, we have $e_N(P,T)^N = 1$. By definition of embedding degree, this means that $e_N(P,T) \in \mathbb{F}_{q^d}^*$ (since the N^{th}-roots of unity are contained in $\mathbb{F}_{q^d}^*$). By linearity of the Weil pairing we have:

$$e_N(Q,T) = e_N(mP,T) = e_N(P,T)^m.$$

We know the values of P, Q, T so if we can solve the DLP

$$e_N(Q,T) = e_N(P,T)^m$$

in $\mathbb{F}_{q^d}^*$, we can recover m and we have done.

□

Although for most of elliptic curves Koblitz and Balasubramanian [2] have shown that the MOV algorithm transforms ECDLP in $E(\mathbb{F}_q)$ into a much harder DLP in $\mathbb{F}_{q^d}$, there are special cases of curves that are banned for cryptography due to the MOV algorithm.

Example 3.4. (Supersingular elliptic curves)
The MOV algorithm is useful when the embedding degree is low, and this is the case for supersingular elliptic curves.
We are in $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ with $p \geq 5$ and this class of curves satisfies $\#E(\mathbb{F}_p) = p + 1$.
If $P \in E(\mathbb{F}_p)$ has order N, then $N \mid \#E(\mathbb{F}_p)$, so that $p \equiv -1 \pmod{N}$. Hence $p^2 \equiv 1 \pmod{N}$, i.e. N has embedding degree 2 in $\mathbb{F}_p$.
This translates into the fact that ECDLP on a supersingular elliptic curve can be reduced via MOV algorithm to solving DLP in $\mathbb{F}_{p^2}$. That's why the use of them in cryptography is deprecated, as it is confirmed by NIST in [17].

There are also worse cases than supersingular elliptic curves. This happens with a class of curves called *anomalous curves*, the ones that satisfy $\#E(\mathbb{F}_p) = p$. Simultaneously and independently Semaev, Satoh-Araki and Smart have shown the existence of a very fast algorithm to solve ECDLP on anomalous elliptic curves. Since the proof use concepts as the one of formal group, we do not treat it, limiting ourselves to stating the algorithm (for the proof we suggest [20]).

Proposition 3.2. (Satoh)
Let $p \geq 3$ and let $E(\mathbb{F}_p)$ be an elliptic curve such that $\#E(\mathbb{F}_p) = p$ (E is an anomalous curve).
Then the following algorithm solves ECDLP in $E(\mathbb{F}_p)$.
(1) Let $P, Q \in E(\mathbb{F}_p)$ nonzero points satisfying $Q = mP$ for an unknown integer m.
(2) Choose an elliptic curve $E'(\mathbb{Q}_p)$ whose reduction modulo p is $E(\mathbb{F}_p)$.
Here $\mathbb{Q}_p$ denotes the field of p-adic numbers (see [13]).
(3) Using Hensel's lemma (see [20]) lift P, Q to points $P', Q' \in E'(\mathbb{Q}_p)$.
(4) The points pP' and pQ' are in the formal group $E'_1(\mathbb{Q}_p)$. Let

$$\log_E : E'_1(\mathbb{Q}_p) \to p\mathbb{Z}_p^+$$

the formal logarithm map, and compute

$$pa = \log_E(pP') \in p\mathbb{Z}_p, \qquad pb = \log_E(pQ') \in p\mathbb{Z}_p.$$

For formal groups and logarithm, see [20].
(5) Then $m = a^{-1}b \pmod{p}$

We just stress out that anomalous curves are avoided in cryptography.
We have seen that the Weil pairing brings to a negative application for cryptography, as MOV is. But the Weil pairing is also involved in special cases that give positive applications in cryptography, in the sense that they increase the possibilities of different cryptosystems. We now see two of these applications, by using a modified Weil pairing.

Definition 3.5. Let $N \geq 3$ be a prime, E an elliptic curve, $P \in E[N]$ be a point of order N and $\phi : E \to E$ a map from E to itself.
We call ϕ an *N-distortion map for P* if it has the following properties:
(1) $\phi(nP) = n\phi(P)$ for all $n \geq 1$.
(2) The number $e_N(P, \phi(P))$ is a primitive N^{th} root of unity. This means that

$$e_N(P, \phi(P))^r = 1 \quad \text{if and only if} \quad N \mid r.$$

Definition 3.6. Let E an elliptic curve, $P \in E[N]$ be a point of prime order $N \geq 3$ and ϕ be an N-distortion map for P.
The *modified Weil pairing* $\hat{e}_N$ on $E[N]$ (relative to ϕ) is defined by

$$\hat{e}_N(Q, Q') := e_N(Q, \phi(Q')), \qquad \text{for every } Q, Q' \in E[N].$$

We start with a version of Diffie-Hellman key exchange involving three people instead of two.

Algorithm 3.7. (Tripartite Diffie-Hellman key exchange)
(1) A trusted authority publishes a finite field $\mathbb{F}_q$, an elliptic curve $E(\mathbb{F}_q)$, a point $P \in E(\mathbb{F}_q)$ of prime order N and an N-distortion map ϕ for P.
(2) Each of Alice, Bob and Carl chooses, respectively, a secret integer n_A, n_B, n_C. Then everyone computes the quantity $Q_i = n_i P$, for $i \in \{A, B, C\}$.
(3) Alice, Bob and Carl publish their points Q_A, Q_B, Q_C.
(4) Then everyone does the following computation, which we write only for the case of Alice:

$$\hat{e}_N(Q_B, Q_C)^{n_A} = \hat{e}_N(n_B P, n_C P)^{n_A} = \hat{e}_N(P, P)^{n_A n_B n_C}$$

and the same procedure is done by Bob and Carl.
(5) Finally, they share the secret value

$$\hat{e}_N(P, P)^{n_A n_B n_C}.$$

The secret value $\hat{e}_N(P, P)^{n_A n_B n_C}$ represents a secret key to use, for instance, with AES. We notice that if Eve can solve ECDLP (for one of the n_i is enough), then she can break the tripartite Diffie-Helmann key exchange. Furthermore, the security of the system can also be endangered if Eve can solve the classical DLP for a subgroup of $\mathbb{F}_q^*$ of order N. Indeed, Eve knows Q_A, P and so can compute both

$$\hat{e}_N(P, P) \qquad \text{and} \qquad \hat{e}_N(Q_A, P) = \hat{e}_N(n_A P, P) = \hat{e}_N(P, P)^{n_A}$$

reducing the problem to solving $a^n = b$ in $\mathbb{F}_q$. Since we know that DLP in $\mathbb{F}_q^*$ can be solved in subexponential time, this requires to choose a large q for tripartite Diffie-Hellman.
Let us now consider another positive application of the modified Weil pairing, the so-called *ID-based public key cryptosystem.* This cryptosystem allows the users to choose directly their public keys in a manner that the public key identifies the user. For instance, Alice may want to choose the email address *alice@idbased.com* as her identity-based public key,

so everyone who wishes to text her automatically knows her public key.
Before the algorithm, let us give the definition of a hash function and a remark of the XOR operation.

Definition 3.7. A *hash function* is a function that takes as input an arbitrary long document D and returns a short string H.

A hash function should be at the same time fast and easy to compute and really hard to invert.
Nowadays, the most common used Hash algorithm is SHA-256, which is recommended by NIST. The previously used SHA-1 has been deprecated.

Remark 3.6. The XOR operation is a bit to bit operation that acts in the following manner:

$$\begin{cases} 0 \text{ XOR } 0 = 1 \text{ XOR } 1 = 0 \\ 0 \text{ XOR } 1 = 1 \text{ XOR } 0 = 1 \end{cases}$$

Let see the algorithm.

Algorithm 3.8. (ID-based public key cryptosystem)
(1) A trusted authority (Tom) publishes a finite field $\mathbb{F}_q$, an elliptic curve $E(\mathbb{F}_q)$, a point $P \in E(\mathbb{F}_q)$ of prime order N and an N-distortion map ϕ for P. Tom also chooses two hash functions:

$$H_1 : \{\text{IDs}\} \to E(\mathbb{F}_q)$$

that allows him to assign a point in $E(\mathbb{F}_q)$ to each possible user ID, and

$$H_2 : \mathbb{F}_q^* \to \{\text{bit strings of length } B\} = \{0,1\}^B$$

that assigns to each element in $\mathbb{F}_q^*$ a binary string of length B.
(2) Tom chooses a secret integer s modulo N, and publishes the public point $P^{Tom} = sP \in E(\mathbb{F}_q)$.
(3) Alice chooses an ID-based public key $Alice^{Pub}$.
Tom computes the point $P^{Alice} = H_1(Alice^{Pub}) \in E(\mathbb{F}_q)$.
Tom sends to Alice the point $Q^{Alice} = sP^{Alice} \in E(\mathbb{F}_q)$.
(4) Bob chooses a plaintext M and a random number r modulo $q-1$.
Bob computes $P^{Alice} = H_1(Alice^{Pub}) \in E(\mathbb{F}_q)$.
Bob's ciphertext is the pair:

$$C = (rP, M \text{ XOR } H_2(\hat{e}_N(P^{Alice}, P^{Tom})^r)) = (C_1, C_2).$$

(5) Alice decrypts the ciphertexts (C_1, C_2) by computing

$$C_2 \text{ XOR } \hat{e}_N(Q^{Alice}, C_1).$$

We prove that Alice is really able to obtain the plaintext of Bob in step (5). Indeed:

$$\hat{e}_N(Q^{Alice}, C_1) = \hat{e}_N(sP^{Alice}, rP) = \hat{e}_N(P^{Alice}, P)^{sr} = \hat{e}_N(P^{Alice}, sP)^r = \hat{e}_N(P^{Alice}, P^{Tom})^r$$

and so

$$C_2 \text{ XOR } \hat{e}_N(Q^{Alice}, C_1) = M \text{ XOR } H_2(\hat{e}_N(P^{Alice}, P^{Tom})^r) \text{ XOR } \hat{e}_N(P^{Alice}, P^{Tom})^r = M.$$

3.4.1 About ECDLP over $\mathbb{F}_{2^k}$

Since computers are developed in a binary language, elliptic curves over binary fields seem to be really suitable for calculations modulo 2, and they are. These curves were initially accepted for cryptography due to the advantages we describe below, but as can be seen in [17], they are actually deprecated.
We rewrite Definition 2.1 for binary fields $\mathbb{F}_{2^k}$, with the additional information about the discriminant we have omitted at the time.

Definition 3.8. An *elliptic curve* E over $\mathbb{F}_{2^k}$ is the set of points

$$\{(x, y) \in \mathbb{F}_{2^k}^2 : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\mathcal{O}\}$$

with $a_i \in \mathbb{F}_{2^k}$ for $i \in \{1, 2, 3, 4, 6\}$ and such that the following condition on the discriminant holds:

$$b_2 = a_1^2 + 4a_2, \quad b_4 = 2a_4 + a_1a_3, \quad b_6 = a_3^2 + 4a_6,$$
$$b_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2,$$
$$\Delta_E = -b_2^2b_8 - 8b_4^3 - 27b_6^2 + 9b_2b_4b_6 \neq 0.$$

We can repeat all the arguments seen in Chapter 2 to prove that $E(\mathbb{F}_{2^k})$ is an abelian group, with a slight modification of the reflection step:

$$(x, y) \mapsto (x, -y - a_1x - a_3)$$

that has a subsequent change on the explicit formulas for the addition law.
Anyway, the reason for which elliptic curves were interesting from a cryptographic point of view, beyond the binary nature that matches with computers, is due to an idea of Koblitz. The idea is using an elliptic curve E over $\mathbb{F}_2$ while taking the points on E with coordinates in $\mathbb{F}_{2^k}$, and play in there with a particular map that allows to gain efficiency in the computation of the sum nP, better than Double-and-Add algorithm does. This means, a faster way to do the encryption part of the algorithm.
Let's briefly describe the idea.

Definition 3.9. The *2-power Frobenius map* τ is the map

$$\tau : \mathbb{F}_{2^k} \to \mathbb{F}_{2^k}, \qquad x \mapsto x^2.$$

The 2-power Frobenius map preserves addition and multiplication. Moreover, if E is an elliptic curve over $\mathbb{F}_2$ and $P \in E(\mathbb{F}_{2^k})$, τ is a group homomorphism on $E(\mathbb{F}_{2^k})$ to itself by defining it on the coordinates of P:

$$\tau : E(\mathbb{F}_{2^k}) \to E(\mathbb{F}_{2^k}), \qquad \tau(P) = (\tau(x), \tau(y)) = (x^2, y^2).$$

Koblitz suggests the use of Frobenius map to a particular class of binary curves which takes his name.

Definition 3.10. A *Koblitz curve* is an elliptic curve E over $\mathbb{F}_2$ that satisfies the equation

$$y^2 + xy = x^3 + ax^2 + 1,$$

with $a \in \{0, 1\}$ and $\Delta_E = 1$.

There is a theorem (see [9]) that tells that given an elliptic curve E over $\mathbb{F}_2$, for every point $P \in E(\mathbb{F}_{2^k})$ the 2-power Frobenius map satisfies

$$\tau^2(P) - t\tau(P) + 2P = 0$$

where $t = p + 1 - \#E(\mathbb{F}_p) = 3 - \#E(\mathbb{F}_2)$ (recall from Theorem 2.2, Hasse's Theorem, that $t \leq \sqrt{2}$). This can be applied to the following result.

Proposition 3.3. Let n a positive integer. The n can be written in the form

$$n = v_0 + v_1\tau + v_2\tau^2 + \ldots + v_l\tau^l, \qquad \text{with } v_i \in \{-1, 0, 1\}$$

under the assumption that τ satisfies $\tau^2 - t\tau + 2 = 0$.
Further, this can always be done with $l \approx 2 \log n$ and with at most $1/3$ of the v_i's nonzero.

Proof. Same proof as for Double-and-Add, with the observation of starting with $n = 2a + b$ and then we proceed substituting $2 = -\tau^2 - \tau$. □

Hence, given an integer n and a point $P \in E(\mathbb{F}_{2^k})$, we can compute nP in the following way,

$$nP = v_0 + v_1\tau(P) + v_2\tau^2(P) + \ldots + v_l\tau^l(P)$$

and this is faster since the Frobenius map on E is easier to compute than the duplication map (so, also faster than the ternary expansion method).
We recall, anyway, that for cryptography elliptic curves over binary fields have been deprecated due to their limited adoption, so despite the advantages we have seen above, we should not be tempted anymore to apply these curves in a secure cryptosystem.

4 Uses of ECC

The aim of this chapter is to demonstrate the importance of mathematics in everybody's daily life: to achieve this goal we highlight the central role of elliptic curve cryptography in guaranteeing a protection to our privacy, our communications and our sensitive data in various settings. To be more specific, what is required in any kind of online transaction (also for the offline ones, as is the case for some smart cards) are the following features.

- *Authentication*: people want to know with whom they are communicating, as well as being able to provide a proof of their identity when required (for mutual authentication).
- *Data integrity*: people need the information they communicate to remain the same from sender to receiver, without any modification or manipulation, both from a third party (Eve) or by accident.
- *Confidentiality*: people also desire the message not to be disclosed along the transit.

Before going on with the ECC applications, we point out how ECC speed up processes of data transmission requiring less storage respect to RSA. This will be helpful for the sequel.

4.0.1 RSA and ECC: Comparison

We want to compare the RSA algorithm (Example 1.5) and ECC. As seen in Chapter 3, the collision algorithm is actually the fastest known general method to solve ECDLP, taking $O(\sqrt{q})$ operations (for the problem in $E(\mathbb{F}_q)$...). To confront ECC with RSA we need to compare the collision algorithm with the best known method to factorize, which turns out to be the Number Field Sieve algorithm. This sieve applies to factorize an integer n larger than 10^{100}, and has the same subexponential time operative cost of the index calculus (Algorithm 3.4):

$$\exp(\sqrt[3]{(\log n)(\log\log n)^2}).$$

We leave out all the details about this method, referring to [9] for them. We just point out that the comparison of the complexity of the two algorithms yields that ECC is harder to solve than RSA for values of q, n having the same length. That is, ECC provides an equivalent level of encryption strength as RSA when using a shorter key length. We report the empirical comparison of the strength of the two cryptosystems based on the key length in the table below. For instance, an RSA cryptosystem using key length of 2048 bit is as secure as an ECC cryptosystem with a 224 bit key length. The security level is a measure of the strength that a cryptographic algorithm reaches in bit: an n bit security means Eve would have to perform 2^n operations to break it. Actually a key length of at least 2048 bit is required for implementing an RSA secure algorithm.
This key length argument translates into the fact that ECC requires less storage to preserve keys and to transmit them (a crucial factor for smart cards as we will see). Furthermore this increases the speed and the security offered by an ECC certificate (the key exchange and signatures of a transmission protocol) compared to an RSA certificate.

So ECC has more efficient implementation at the same security level when referring to key exchange and digital signature systems.

Security (bit)	RSA key length	ECC key length
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

We underline once more the advantages of ECC with respect to RSA by reporting a recent experiment, published in 2023: we refer all the details to [11]. In this article [11], a research group compared ECC and RSA in terms of key creation, message encryption and message decryption. They apply both encryption and decryption part to a short plaintext: "Our research focuses on comparative study of RSA and ECC." For the purpose of comparison they used the Maple software and a "normal" computer system: 64-bit OS, Windows version 10, Intel Core i7, CPU with 1.80GHz, 8GB RAM. The results are listed in the tables below.
In the first table ECC and RSA are compared for the task of key generation. For security level greater than 128 bits, the difference of average time required to accomplish this task is not negligible. In particular, when increasing the security, the experiment shows an exponential increasement of the time difference between ECC and RSA in the key creation process.

Key generation time requirement				
Security Bits	**RSA key length**	**ECC key length**	**RSA key time generation**	**ECC key time generation**
80	1536	192	1.3188	0.0904
112	2048	224	0.2752	0.1028
128	3072	256	6.3312	0.103
192	7680	384	29.5404	0.1188
256	15360	512	58.1994	0.1436

*The last two columns on the right represent the average time required to generate a key (in seconds).

The other two tables below show the time required for the process of encryption and decryption, respectively. For the encryption process, we can notice that RSA is faster than ECC, despite the difference of time of execution is not too much (anyway, the small size of the plaintext must be considered there: ECC requires on average about two and half the time of RSA in the case of 128 bits security, and for large size plaintexts this difference is not negligible). Instead, for the decryption process we find again that ECC is faster than RSA algorithm.

Performance for encryption				
Security Bits	**RSA key length**	**ECC key length**	**RSA Time (seconds)**	**ECC Time (seconds)**
80	1536	192	0.078	0.2062
112	2048	224	0.072	0.2096
128	3072	256	0.0812	0.2158
192	7680	384	0.0844	0.2282
256	15360	512	0.1	0.251

Performance for decryption				
Security Bits	**RSA key length**	**ECC key length**	**RSA Time (seconds)**	**ECC Time (seconds)**
80	1536	192	0.0625	0.0525
112	2048	224	0.0782	0.582
128	3072	256	0.0812	0.059
192	7680	384	0.1096	0.07
256	15360	512	0.2658	0.0811

This research shows what is frequently employed in many cryptographic applications: the most used cipher suites employ ECC as key exchange system, since ECC is faster than RSA for key generation, at the same level of security. For the encryption part, RSA is preferred to ECC, but it is not used as we told in Chapter 1.

4.1 Https and TLS

We are not telling lies if we claim that a part of our life is actually in binary digits. Sometimes we hear of people having troubles after the loss of all authentication passwords to their personal email account. Our mail services are accessible almost everywhere via the internet, we have enhanced our purchases on e-commerce, our messages and calls take place via the web, as well as for any kind of job career internet is a mandatory tool (Internet of Things, smart working, storekeepers too, ecc...). The world wide web is soaked with our information from sensitive to trivial ones. And all this information must travel over a secure web.
The main tool for exchanging informations via web is HTTPS (HyperTextTransferProtocol over TLS), a secure communication protocol used by internet. HTTPS includes the cryptographic protocol TLS (*Transfer Layer Security*), which allows secure communication. This protocol substituted the previous SSL (*Secure Sockets Layer*, deprecated in 2015), and is actually active with the latest two releases: TLS 1.2 and TLS 1.3. Let see how the TLS protocol works.
There are two peers who want to communicate, a client (a browser over a device, for instance) and a server (such as a web server). To establish a secure communication channel

between them they use the TLS protocol, which consists of two sub-protocols: the *handshake* protocol and the *record* protocol.
The handshake is mainly used to establish a common cipher suite (a set of algorithms that globally allows a secure network communication), and the choice is made between the available cipher suites of both client and server. We notice in advance that the existence of a certificate authority (CA, a trusted third party) is assumed, which creates and distributes personal certificates containing cipher suites, both to servers and clients at their request. These certificates contain also a temporary identity assigned by the CA for the requesting party, the public key of the requesting party and an expiry date of the certificate (this collection form then a binary string signed by the private key of the CA to obtain the certificate).
The following is the handshake procedure.

- The client sends to the server a "ClientHello", which is nothing but a list of cryptographic algorithms and parameters supported by the client. In particular, there might be various cipher suites written in a code point. For instance,

  TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

  is a cipher suite for TLS that uses ECDSA for signatures, ECDHE for key exchange, AES_128_GCM as symmetric cipher and SHA256 as hash function. Moreover in the certificate sended by the client there are other parameters, such as the supported Diffie-Hellman groups (in particular the elliptic curves and their base fields) and a public key pair for each Diffie-Hellman group.

- The server receives the ClientHello and computes the (unauthenticated) Diffie-Hellman shared secret key, that will be used to derive symmetric keys for subsequent record protocol. Next, the server responds with a "ServerHello". This message contains the server choice for the Diffie-Hellman group as well as the choice for the cipher suite and the public key (notice that in this way the server does not show its cipher suites, but only one among them; notice also the server may close the connection if it does not support any common cipher suites with the client). Moreover, the server sends also a certificate that contains a public key of the chosen signature algorithm (as a special case for signatures, in ClientHello the client has the possibility to choose in advance which signature method must be used. This does not hold for other cryptographic schemes).

- In case the server requires the client to provide a certificate of authentication (a common case in any log in platform for the client), the latter must answer with its own verified certificate.

- The client receives the ServerHello and compute the Diffie-Hellman shared secret key. Then he sends the message "Finished" to the server.

If all the steps above are satisfied, the handshake is complete and so the connection is established; it is possible to proceed with the record protocol, which consists of data

transmission between the two peers, encrypting the information via the chosen symmetric cryptosystem. We point out that hash functions also play an important role in the handshake as well as in the certificates release by CA: we refer to [4] and [1] for the details. Elliptic curve cipher suites offering forward secrecy (establishing a session key using elliptic curve Diffie-Hellman key exchange) were introduced in 2006; in the same year ECDSA was introduced for signatures from servers.
For what concerns us, elliptic curves arise in several stages of the TLS protocol, both for the elliptic curve Diffie-Hellman key exchange and for the transmission of the public key that the server uses to authenticate itself (ECDSA).
The following key agreement and authentication suites are active in TLS 1.3:

ECDHE_RSA, DHE_ECDSA, ECDHE_ECDSA, ECDHE_EdDSA, ECDHE_PSK.

We recall (see Remark 3.3) that there is a slight difference between ECDH and ECDHE. The latter added E stands for *ephemeral* and is an attribute of the keys. In ECDHE the keys are ephemeral (short-lived) while in ECDH they are static; ephemeral keys are temporary and not authenticated, so in ECDHE (uncertified) key pairs are generated anew at the moment of the handshake, instead of using long-lived certified keypairs. Thus, when client and server complete their handshake with ECDHE, they have a secure private communication channel. Meanwhile, their identities are not verified, because ephemeral keys aren't certified. That's why a signature scheme as ECDSA is needed.
We put the handshake in practice by giving some examples of secure connections established with our device (a smartphone) via the browser Google Chrome (version 130) to some well-known web servers.

- We accessed "en.m.wikipedia.org", the most used encyclopedia nowadays; the handshake took place over a TLS 1.3 protocol and the connection was established. The key exchange algorithm chosen was ECDHE, using the curve 25519 (Section 2.4). The public key of the server was (in hexadecimal):
  00 04 AC 5C 9F E4 4E 15 1E 22 D0 9C 3C F7 01 97 23 B5 6A 31 D7 6C 61 57 11 13 08 4C 4F 2B 3A D9 50 66 01 11 30 21 36 D3 85 BB 43 EB D5 A9 10 20 3A 36 67 86 32 A7 01 4D 3C A6 D9 0F 09 CD 72 8A 5E 32.
  Notice that accessing with another browser or device (i.e. a possibly different certificate client side) may bring to a different choice of key exchange.
  We observe moreover that the choice for the signature scheme was by RSA method (similar to Example 1.6). Referring to the above section 4.0, we report the public key of the signature, just to underline the length difference with the elliptic curve key above:
  D8 B3 F1 F6 91 64 E5 A5 EA 66 04 53 5E 0B 6B EB 21 7C BE F1 28 8E C6 5A 77 4D 33 84 51 39 AE 6C 0D 1F EE 3C C1 B6 00 D6 38 DA 45 ED 50 90 58 35 4B AD 49 C7 21 90 12 95 D4 42 4A AE F4 BF E3 75 FB 3D 95 10 40 C2 30 84 61 1C FF 44 D2 12 BA 12 EC 9A 40 D9 59 41 B7 13 14 0C 21 9B 3A B2 FE BD E2 C6 76 B0 D1 05 22 20 E6 E1 68 DB 01 93 25 9D 8D 79 17 59 A6 C4 31 45 39 E5 33 13 65 5B 7D D4 25 C3 7F CA 73 A2 50 45 59 3B DD 7B 77 51 DF 57 A0 DC 0C 31 69 36 8D EC

10 9D 7D 01 37 15 71 33 5F 86 97 01 0B FD EA A2 29 E8 C1 DC A7 39 32 94 02 1B 2D BD 35 60 FD 86 EF 16 47 16 DD A9 DC 1B 7B A1 E7 DF A4 BF 0F E1 71 44 E9 A9 3A 0D BF 63 3F 11 AB C0 A8 FD 50 FA FA 49 65 7D 37 4C B1 C5 02 30 A7 F3 43 91 2C 9B AB EB C6 D7 53 E3 CD 48 3D 0C 60 D0 60 40 82 C4 95 15 53 D2 B6 2D F0 B5.

- We were curious about our University of Padova website. The protocol used for the connection was TLS 1.2, with ECDHE_RSA as key exchange algorithm (AES_256_GCM for the record protocol). Conversely, the access to uniweb requires a TLS 1.3 protocol, using again curve 25519 on the key exchange ECDHE. The reason of the common use of the curve 25519 (also Whatsapp and Facebook Messenger use it) is because it is open source (no license required to implement the curve) and the curve guarantees high speed of execution. Also Amazon, the most used platform for e-commerce, "prefers" an agreement of key exchange ECDHE via this curve.

- We accessed the Whatsapp website without authentication and as claimed it uses ECC for key exchange. The public key used by the server is:
0004A25C5C1B343C9D2C0D89820BE15E813C608FD0268B45EFEB8B9DC18F08A92
34D1529207073C44204EA74D0CD8E0B516655DB45FE0E19CF866AE493A38806CF71

- The most visited web server in Italy is a search engine; no need to say, the web address is "www.google.com". We accessed it both with a smartphone and a personal computer using different Google Chrome versions; in both cases the handshake choice for the key exchange was elliptic curve Diffie-Hellman ephemeral key exchange.

4.2 Smart Cards

The use of smart cards has increased for the last 20 years. They are almost everywhere, from payments circuits and identity cards to transport subscription cards, passports etc... Smart cards are involved in a wide variety of applications, such as electronic commerce, identification, health care. For all these applications, cryptographic services are required, in particular digital signatures. The key point is that smart cards are constrained by some features: first of all their storage capability is little. Moreover, smart cards have limited computing power and are required to be cheap for a wide distribution and use. To give some data, smart cards on today's market commonly have a 8 bit CPU clocked at 4MHz, 4Kb of EEPROM (Electrically Erasable Progammable Read Only Memory, to store small datas without electrical support) and 8.2Kb of RAM (this is really little: a medium use of 10 opened windows of Google Chrome in a smartphone 1000Mb of RAM usage).
As we pointed out at the beginning of chapter, ECC offers small key size and high security at the same time. Therefore it represents a good cryptographic environment for smart cards. In particular the benefits of ECC are:
(1) Smaller key size means less EEPROM required to store keys and certificates; moreover this translates also in less data transmission between the card and the application/terminal, resulting in shorter time transmission.
(2) ECC smart cards provide higher levels of security with lower cost, since they can

continue to provide the security with proportionately fewer additional system resources (for instance, requiring less RAM yields to no need of additional coprocessor, saving further cost).
We give now an example of smart card protocol. For any details for this part, we refer to [8].
In the following, the server can be thought of as a terminal, while the client is the smart card. The algorithm used for the protocol is ECDSA (Algorithm 3.3). Both terminal and card have to be initialized first, i.e. they have to install certificates released from a CA. The initialization requires the choice of a finite field $\mathbb{F}_p$, an elliptic curve $E(\mathbb{F}_p)$ and a point $P \in E(\mathbb{F}_p)$ of prime order N. In the following, $t_{\mathbf{s}}$ and $t_{\mathbf{u}}$ are the certificate expiry dates, $I_{\mathbf{s}}$ and $I_{\mathbf{u}}$ are temporary identities, H is a hash function, f and f^{-1} represent the symmetric encryption and decryption functions via the shared secret key (we recall the notation of Chapter 1: for the secret key k we have $C = f(k, M)$ and $M = f^{-1}(k, C)$, where M is the plaintext and C the ciphertext).
The initialization is written in the table below. Basically, this process corresponds to the first part of ECDSA (points (1)-(4) of Algorithm 3.3, i.e. signature and key generation).

Terminal/Server		**Certification Authority**
Choose secret integer $n_{\mathbf{s}}$		Choose secret integer $k_{\mathbf{s}}$
Compute $Q_{\mathbf{s}} = n_{\mathbf{s}}P$		Compute $R_{\mathbf{s}} = k_{\mathbf{s}}P$
Send	$\rightarrow Q_{\mathbf{s}}$	Receive
		Choose unique $I_{\mathbf{s}}$
		$r_{\mathbf{s}} = R_{\mathbf{s}}.x$ (x-compression)
		$s_{\mathbf{s}} = k_{\mathbf{s}}^{-1}(H(Q_{\mathbf{s}}.x, I_{\mathbf{s}}, t_{\mathbf{s}}) + d_{CA}r_{\mathbf{s}})$
Receive	$Q_{CA}, I_{\mathbf{s}}, (r_{\mathbf{s}}, s_{\mathbf{s}}), t_{\mathbf{s}} \leftarrow$	Send
$e_{\mathbf{s}} = H(Q_{\mathbf{s}}.x, I_{\mathbf{s}}, t_{\mathbf{s}})$		
Store $Q_{\mathbf{s}}, Q_{CA}, I_{\mathbf{s}}, (r_{\mathbf{s}}, s_{\mathbf{s}}), e_{\mathbf{s}}, t_{\mathbf{s}}$		

This is how the server has been initialized. At the end of the initialization the terminal stores the final information:

$$Q_{\mathbf{s}}, Q_{CA}, I_{\mathbf{s}}, (r_{\mathbf{s}}, s_{\mathbf{s}}), e_{\mathbf{s}}, t_{\mathbf{s}}.$$

The certificate itself consists in the pair of integers $(r_{\mathbf{s}}, s_{\mathbf{s}})$ for the server. For the user/client/card the procedure is analogous, leading to certificate $(r_{\mathbf{u}}, s_{\mathbf{u}})$ and final stored information:

$$Q_{\mathbf{u}}, Q_{CA}, I_{\mathbf{u}}, (r_{\mathbf{u}}, s_{\mathbf{u}}), e_{\mathbf{u}}, t_{\mathbf{u}}.$$

Notice that $r_{\mathbf{s}}, r_{\mathbf{u}}$ are simply the x-coordinate of the elliptic curve points $R_{\mathbf{s}}$ and $R_{\mathbf{u}}$, respectively.
We describe in the table below the protocol of communication between smart card and terminal. A mutual authentication (via ECDSA) is needed for both terminal and card, in order to protect their communications against eavesdropping (especially for the case of contactless cards, in which datas travel everywhere over the air, rather than inserting card

in a suitable reader), to ensure authenticity of each party and to avoid cases of fake cards or fraudulent terminals.
The protocol starts with a public keys exchange and proceeds with the simultaneous computation of a secret key (ECDHE). This key serves to encrypt data required for a mutual authentication. At this point the server encrypts its certificate together with the expiry date and a random number g and sends it to the user, which can decrypt and read it. The user repeats the same procedure and sends its encrypted certificate to the server, which decrypts it too.
The next step is a check of the regularity of the certificate for both user and server (otherwise the temptative connection can be aborted by whichever side is not satisfied). This step corresponds exactly to the point (5) of Algorithm 3.3. Notice that before this step the server can already close the service if the expiry date of user's certificate is not valid or if the random number g is not "random enough" (the latter check prevent spoofing attacks by the user side, we mention this issue in Remark 4.1 below).
Last step involves both card and terminal and is the computation of a new secret key to be used for encrypting all the communication in the channel (must be different of the mutual agreed key).

USER/CARD		**TERMINAL/SERVER**
Receive/Send	$\leftarrow Q_{\mathbf{s}} \quad \rightarrow Q_{\mathbf{u}}$	Send/Receive
Compute $Q_k = n_{\mathbf{u}} Q_{\mathbf{s}} = n_{\mathbf{u}} n_{\mathbf{s}} P$		Compute $Q_k = n_{\mathbf{s}} Q_{\mathbf{u}} = n_{\mathbf{s}} n_{\mathbf{u}} P$
	$Q_k.x$ mutually agreed key	
		Generate random number g
		$C_0 = f(Q_k.x, (e_{\mathbf{s}}, (r_{\mathbf{s}}, s_{\mathbf{s}}), t_{\mathbf{s}}, g))$
Receive	$C_0 \leftarrow$	Send
$f^{-1}(Q_k.x, C_0)$		
$C_1 = f(Q_k.x, e_{\mathbf{u}}(r_{\mathbf{u}}, s_{\mathbf{u}}), t_{\mathbf{u}}, g)$		
Send	$\rightarrow C_1$	Receive
		$f^{-1}(Q_k.x, C_1)$
		If g and $t_{\mathbf{u}}$ are valid then
$c = s_s^{-1}$		$c = s_{\mathbf{u}}^{-1}$
$u_1 = ce_{\mathbf{s}}$		$u_1 = ce_{\mathbf{u}}$
$u_2 = cr_{\mathbf{s}}$		$u_2 = cr_{\mathbf{u}}$
$R = u_1 P + u_2 Q_{CA}$		$R = u_1 P + u_2 Q_{CA}$
$v = R.x$		$v = R.x$
If $v \neq r_{\mathbf{s}}$ then abort		If $v \neq r_{\mathbf{u}}$ then abort
$k_m = Q_k.x + g$		$k_m = Q_k.x + g$
$Q_f = H(k_m) Q_k$		$Q_f = H(k_m) Q_k$
$k_f = Q_f.x$		$k_f = Q_f.x$

We want to underline that this is only one kind of possible mutual authentication protocol for server and user in the world of smart cards. For instance, there are a lot of cases in

which certificates travel uncrypted. The procedure above enhances the security with a (little) loss of time of execution.

Remark 4.1. (Poor randomness risk)
In Algorithm 3.3 we have described ECDSA and used it in the protocol. What is essential with ECDSA is the choice of the random number for the signer. We recall that Alice in Algorithm 3.3 generates her signature (s_1, s_2) starting (directly) from the choice of the random number k (corresponding to the choice g for the Certificate Authority above). This is a sensitive step! This value must be unpredictable and distinct at every signature. Otherwise, Eve might be able to recover the long-term private key from one or two signature values. Indeed:

- *Predictability*: if k is predictable Eve can recover it and then the secret signing key can be computed:
$$n_a = s_1^{-1}(s_2 k - d) \mod N, \tag{20}$$
since s_1, s_2 are given and the document d can be easily computed from the signed message (the hash function is known).

- *Constant value*: assume that the random secret integer k is used more than once to sign different documents. Let us say k is used to sign documents d_1, d_2, producing two signatures (r, s_1) and (r, s_2) (the secret key k remains the same so the first coordinate must coincide). Eve can immediately obtain the value of k:
$$k \equiv (s_2 - s_1)^{-1}(d_1 - d_2) \mod N.$$
Moreover, Eve can now recover the secret key n_a using (20).

About the constant value case, a well known event happened some years ago. In April 2011 there was an external intrusion on Sony Playstation Network. The Sony Playstation 3 product, a well known video game console, used a constant value for signatures generated via ECDSA. This allowed hackers to compute the secret code signing key. This made 77 million accounts personal details compromised and unabled users of Playstation 3 to access the service. All this occurrence has caused a blackout of the Network services lasting 23 days, with a damage of 171$ millions for Sony.

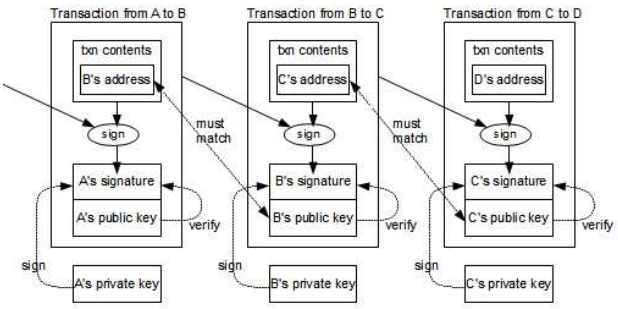
Another good example of the ECC implementation in smart cards can be found in [4]. It concerns Austria national e-ID cards, which contain both RSA and ECDSA public keys, used to provide legally binding digital signatures. A great amount of Citizen Cards certificates were gathered in 2013, showing that about 60% of these certificates contained an elliptic curve public key, showing which curves were employed and proving the absence of issues related to ECDSA keys.
We conclude the section with the fact that in order to guarantee security for limited devices having low computing power, storage and battery (for instance, most of IoT), one of the best recommended choice for a cipher suite is:

TLS_ECDHE_ECDSA_WITH_AES_128_GCM_8.

In particular, this is a good choice also for smart cards; contactless payment cards are the best example to allow us underline once again the central role of elliptic curve cryptography, and mathematics, in our daily life.

4.3 Bitcoin and Blockchain

One of the most actual topic nowadays in cryptography are cryptocurrencies and the technology behind them: the blockchain. We briefly introduce the case of Bitcoin since the mathematics used on blockchain is simply elliptic curve cryptography. Of course there are tens of other cryptocurrencies, as Ethereum; but we choose Bitcoin since it is the first cryptocurrency created, it is still the most used all over the world and has an attractive economic market. For more details on Bitcoin and Blockchain we refer to [15] and [4].
The origin of Bitcoin is a document published in 2009 in a cryptography forum under the pseudonym of Satoshi Nakamoto [15]. Its aim was to create a virtual currency which allows to send online payments directly from one party to another without need of intermediaries (as financial institutions) when carrying out these transactions. This is phrased by stating that Bitcoin is a peer-to-peer digital currency.
To obtain this peer-to-peer system, Nakamoto proposed a public blockchain, which consists of a computer protocol similar to a database: the blockchain is a journal of all the transactions ever executed, and the information is collected using different computer equipments of users throughout the world. Each block in the blockchain contains the hash (using SHA-256 hash function) of the previous block, and thus the blocks are chained together starting from the starting block.
In practice, as shown in the figure below, the transfer ownership of bitcoins from user A to user B is realized by attaching a digital signature (using A's private key) to the hash of the previous transaction and the information of the public key of user B. This concatenation, together with the value of bitcoins transferred, is hashed and constitutes the transaction from A to B. The fundamental point here is the signature, which can be verified by everyone on the network using user A's public key (this is part of the *mining* process).
At this point everyone can buy bitcoins, once an electronic wallet is installed on their computer in order to deposit bitcoins. At any given instant, the user has a Bitcoin address (generated by hashing in a particular manner a given public key) coming from previous transactions, and a new Bitcoin address is generated for each new operation.
We omit further details from the references, pointing out that the whole system is based on mining by a network of computers: this process allows the extraction of new bitcoins, helps with indexing the blocks (thanks to the timestamp) and prevents the double spending issue.



All the above Blockchain system is obtained with ECC. In comparison with the other uses of ECC previously seen, here the exclusivity of the use of ECC stands out. Indeed the whole procedure of chaining blocks, guaranteeing the inviolability of records, is obtained by signing the transactions with ECDSA.
The elliptic curve used in Bitcoin and Blockchain is

$$y^2 = x^3 + 7,$$

defined over the field $\mathbb{F}_p$, where p is

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 2^0.$$

The base point P, written in (hexadecimal) x-compressed form, is :
0279BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798.
We conclude the section emphasising the increasing importance of cryptocurrencies, together with the controversies they produce. The value of the Bitcoin is determined by supply and demand. If in 2009 the value of 1 Bitcoin was close to zero, around December 2017 it reached the value of US $10000 and doubled this value in a few weeks. The controversies concern the wide fluctuations of the Bitcoin value during the successive period, and this is the reason why some economic experts claimed that Bitcoin is not a currency, but only a speculative instrument.
Anyway, we cannot ignore the current value of 1 Bitcoin (US $95817 at February 2025), with the various predictions that talk of a fast growth of the Bitcoin value in the next future. Despite its fluctuations, Bitcoin marketplace is an great gain opportunity all over the world and plays a non-trivial active economic role.

4.4 ECC and future security, post-quantum cryptography

We end the chapter with another crucial modern topic: quantum computing, with references in [1], [22] and [5].
Quantum computing has its origins in 1984. This is the year in which David Deutsch (1953) first doubted the tacit assumption that computers must obey the laws of classical physics. Since their operations take place in the microscopic world, and this world is regulated by quantum mechanics, why not think of quantum computers? He then reformulated computing science under the hypothesis of quantum mechanics, indeed noticing that computers work in a microscopic world in which phenomenas are regulated by quantum laws.
If it is true that quantum computers are not actually present in our daily life, it is also true that their power of computation compared to traditional computers would be a threat for our lives. Indeed, as reported in the table below, not only ECC but also the most common public key cryptography, key exchange methods and signatures would be obsolete with respect to a quantum computer.

Primitive	**Algorithm**	**Quantum Resistance**
Symmetric cipher	AES	Requires larger keys
Hash function	SHA-2	Requires larger output
	SHA-3	Requires larger output
Key exchange	(EC)DH	Not secure
Public key encryption	RSA	Not secure
	ElGamal	Not secure
Digital signature	RSA	Not secure
	(EC)DSA	Not secure

For instance, referring to the TLS protocol for the security via Internet, an adversary (Eve) who would own a quantum computer would be able to break the integer factorization and discrete logarithm problems, assumed at the start of the cryptographic protocol. There is no need to say how this would determine a security and privacy disaster.
To prevent this situation, in the last years attention has been focused on PQC (*Post Quantum Cryptography*). PQC basically consists of building blocks whose security is based on certain computational problems that are believed to be practically unsolvable for quantum computers. Despite that we are just at the beginning of this new path, NIST launched a standardization process for PQC in 2016 and in August 2024 it published the three post quantum standard cryptosystem. They consist of signature schemes and are available on [16].
To conclude, despite the actual great employment of elliptic curve cryptography, the next future may be dominated by post quantum cryptography, confirming the transitory nature of cryptosystems through human history.

A Groups and Fields

Definition A.1. A *group* $(G, *)$ is a set G endowed with a binary operation $* : G \times G \to G$ wich satisfies the following properties:
(1) [associativity] $(a * b) * c = a * (b * c)$, for all $a, b, c \in G$.
(2) [existence of the identity] There exists an $e \in G$ such that $a * e = e * a = a$, for all $a \in G$.
(3) [existence of the inverse] For all $a \in G$ there exists $a^{-1} \in G$ such that $a * a^{-1} = a^{-1} * a = e$.
If it also satisfies the commutative property ($a * b = b * a$ for all $a, b \in G$), then the group is said to be *abelian*.
A subset H of G is said to be a *subgroup* of G if it is a group respect to the restriction of $*$ to H.

Definition A.2. A group G is *cyclic* if there exists $g \in G$ such that

$$G = \{g^n : n \in \mathbb{Z}\}.$$

In such a case g is called a *generator* of the group G.

Definition A.3. A *field* $(K, +, \cdot)$ is a set K endowed with two binary operations $+, \cdot : K \times K \to K$ that satisfy the following properties:
(i) $(K, +, 0)$ is an abelian group, where 0 is the identity element for $+$.
(ii) $(K \setminus \{0\}, \cdot, 1)$ is an abelian group, where 1 is the identity element for $\cdot$.
(iii) $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$, for all $a, b, c \in G$. [distributivity]

Definition A.4. Let K a field. The *characteristic* of a field K is the integer k so defined:

$$k = \begin{cases} min\{n \in \mathbb{N}^* : \underbrace{1 + 1 + \ldots + 1}_{n-times} = 0\} & \text{if } \{n \in \mathbb{N}^* : \underbrace{1 + 1 + \ldots + 1}_{n-times} = 0\} \neq \emptyset \\ 0 & \text{if } \{n \in \mathbb{N}^* : \underbrace{1 + 1 + \ldots + 1}_{n-times} = 0\} = \emptyset \end{cases}$$

The characteristic of a field is always 0 or a prime number p.

Example A.1. Examples of fields are $\mathbb{Q}, \mathbb{R}, \mathbb{C}$, which are well known.
The finite fields we most widely use in these pages is $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$, the integers modulo a prime number p. This is nothing else than the field of modular arithmetic (remainder classes) modulo p.

Example A.2. (Field $\mathbb{F}_q$)
For cryptography, the natural environment is the finite field $\mathbb{F}_q$ of q elements: we give here a brief proof that for any such a field q is a power of its characteristic p, namely $q = p^k$. We also prove that for every $q = p^k$ such a field exists, is unique up to isomorphism and coincides with the splitting field of the polynomial

$$x^q - x = 0.$$

Theorem A.1. Let $\mathbb{F}_q$ a finite field of q elements. Then:

(1) $q = p^k$, where p is the characteristic of the field and $k \geq 1$.
(2) Every element of $\mathbb{F}_q$ satisfies the equation $x^q - x = 0$: equivalently, $\mathbb{F}_q$ is precisely the set of roots of that equation.
(3) Conversely, for every prime power $q = p^k$ the splitting field over $\mathbb{F}_p$ of the polynomial $x^q - x = 0$ is a field of q elements.

Remark A.1. Let K a field and $f(x) \in K(x)$ a polynomial. The *splitting field* of $f(x)$ over K is the smallest extension field $L \supset K$ such that $f(x)$ splits into a product of linear factors

$$f(x) = c \prod_{i=1}^{\deg f} (x - \alpha_i), \qquad c \in K, (x - \alpha_i) \in L(x)$$

where the roots α_i are in L.

Example A.3. Let $K = \mathbb{Q}$ and $f(x) = x^2 + 1 \in \mathbb{Q}(x)$. Thus $f(x) = (x + i)(x - i)$ splits in $\mathbb{Q}(i) \subset \mathbb{C}$, hence $\mathbb{Q}(i)$ is the splitting field of $f(x)$ over $\mathbb{Q}$ (by the lines we are using the fact that there is no field L satisfying $\mathbb{Q} \subset L \subset \mathbb{Q}(i)$).

Proof. (1) A finite field cannot have characteristic zero , so the characteristic of $\mathbb{F}_q$ must be a prime number p. Hence $\mathbb{F}_q$ contains the prime field $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ and so is a finite-dimensional vector space over it. Let k be its dimension as a $\mathbb{F}_p$-vector space. After the choice of a basis $(b_1, ..., b_k)$, we have a biijective correspondence between the elements of this k-dimensional vector space and the set of all k-tuples of elements of $\mathbb{F}_p$ (indeed $x = x_1 b_1 + ... + x_k b_k$, for $x_i \in \mathbb{F}_p$). This implies that the cardinality of $\mathbb{F}_q$ is p^k. So q is always a power of the characteristic p.
(2) We use a known result: for a finite field $\mathbb{F}_q$ of q elements the order of a nonzero element divides $q - 1$ (see, for instance, [13]). It follows that any non-zero element satisfies the equation

$$x^{q-1} = 1.$$

We get the desired equation after multiplying both sides by x:

$$x^q - x = 0.$$

Of course, also zero satisfies the equation, thus any element of $\mathbb{F}_q$ is a root of the degree q equation. Moreover, the polynomial has at most q roots; this yields that the roots of the polynomial are exactly the q elements of $\mathbb{F}_q$.
We remark that this means that $\mathbb{F}_q$ is the splitting field of the polynomial $x^q - x = 0$, i.e. the smallest field extension of $\mathbb{F}_p$ that contains all of its roots.
(3) Let now $q = p^k$ and $\mathbb{F}$ be the splitting field over $\mathbb{F}_p$ of the polynomial $x^q - x = 0$.
The derivative of this polynomial is $qx^{q-1} - 1 = -1$ (because $p \mid q$, so $q = 0$ in $\mathbb{F}_p$): hence the polynomial has no roots in common with its derivative, so there are no multiple roots. Thus, $\mathbb{F}$ must contain at least the q distinct roots of $x^q - x = 0$.
We claim that the set of q roots is already a field. The key point is that a sum or product of two roots is again a root. Indeed, if a, b are roots then $a^q = a$ and $b^q = b$, hence:

$$(ab)^q = a^q b^q = ab,$$

$$(a+b)^q = \sum_{j=0}^{q} \binom{q}{j} a^{q-j} b^j = a + b,$$

where the second equation comes from the fact that $\binom{q}{j} = 0$ in $\mathbb{F}_p$ for $j \neq 0, q$.
Therefore, if a, b are roots, also ab and $a+b$ are. Thanks to this, we conclude that the set of q roots is the smallest field containing the roots of $x^q - x = 0$, i.e. the splitting field of this polynomial is a field with q elements. □

Definition A.5. Let $\mathbb{F}_q$ finite field. The *order* of a nonzero element $x \in \mathbb{F}_q$ is the smallest positive integer n such that $x^n = 1$ (its order as an element of the multiplicative group $\mathbb{F}_q^*$).

Definition A.6. Let F be a field and x be an element of F such that $x^N = 1$, where N is a positive integer. Then x is called a N^{th} *root of unity*. Moreover, x is called a *primitive* N^{th} root of unity if x is a N^{th} root of unity and it satisfies $x^k \neq 1$ for every $1 \leq k < N$.

B Some results in number theory

Theorem B.1. (Euclidean Algorithm)
Let a, b 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 a remainder r_{i+1}:

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

(4) If the remainder $r_{i+1} = 0$, then $r_i = gcd(a, b)$.
Otherwise ($r_{i+1} > 0$) set $i = i + 1$ and repeat step (3).

Notice that the division in step (3) is executed at most $2 \log_2 a$ times. We prove this assertion on the operative cost.

Proof. We claim that $r_{i+1} < r_{i-1}$, indeed:
- if $r_i \leq \frac{1}{2} r_{i-1}$ we are done, since $r_{i+1} < r_i$.
- otherwise, if $r_i > \frac{1}{2} r_{i-1}$, since we know that

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

we immediately get

$$r_{i+1} = r_{i-1} - r_i q_i < r_{i-1} - \frac{1}{2} r_{i-1} q_i = r_{i-1}(1 - \frac{1}{2} q_i).$$

But now we observe that the r_i's are decreasing, so must be $q_i \geq 1$. On the other hand, r_i's are nonnegative, hence $q_i \leq 1$. This yields to $q_i = 1$, and in particular to the claim $r_{i+1} < r_{i-1}$.
So at every two steps the remainder is cut in a half:

$$r_2 < \frac{1}{2} a, \quad r_4 < \frac{1}{2} r_2 < \frac{1}{2^2} a, \quad \ldots, \quad r_{2i} < \frac{1}{2^i} a.$$

Now, $r_{2i} \in \mathbb{N}$, so as soon as $2^i \geq a$ we get $r_{2i} < 1$, therefore $r_{2i} = 0$.
We can conclude that this happens when

$$i \geq 1 + \log_2 a = \log_2 2a$$

and so the division step is executed at most $2i = 2 \log_2 a$ times.
Logarithm grows up really slow, and this feature makes Euclidean algorithm works well in prectice, also for large values of a, b. □

Theorem B.2. (Extended Euclidean Algorithm)
Let a, b positive integers. Then the equation

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

has always a solution u, v.
If (u_0, v_0) is one such a solution, then every solution has the form

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

Theorem B.3. (Chinese Remainder Theorem)
Let $m_1, \ldots, m_k$ be a collection of pairwise relatively prime numbers (i.e. for $i \neq j$ $\gcd(m_i, m_j) = 1$).
Let $a_1, ..., a_k$ be any list of k integers. Then the system of simultaneous congruences

$$\begin{cases} x \equiv a_1 (\bmod\ m_1) \\ \ldots \\ x \equiv a_k (\bmod\ m_k) \end{cases}$$

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

$$c \equiv c' (\bmod\ m_1 \cdot m_2 \cdot \ldots \cdot m_k)$$

References

[1] Alnahawi Nouri, Müller Jonathan, Oupický Jan, Wiesmaier Alexander, *A comprehensive Survey on Post-Quantum TLS*, IACR Communications in Cryptology, 2024.

[2] Balasubramanian Ramachandran, Koblitz Neal, *The improbability that an elliptic curve has subexponential discrete log problem under Menezes-Okamoto-Vanstone algorithm*, J. Cryptology, 1998.

[3] Bernstein Daniel J., Birkner Peter, Joye Marc, Lange Tanja, Peters Christiane, *Twisted Edwards Curves*, https://eprint.iacr.org/2008/013.pdf ,2008.

[4] Bos Joppe W., Halderman Alex J., Heninger Nadia, Moore Jonathan, Naehrig Michael, Wustrow Eric, *Ellyptic Curve Cryptography in Practice*, Conference Financial cryptography in practice, 2014.

[5] Castryck Wouter, Decru Thomas, *An efficient key recovery attack on SIDH*, eprint.iacr.org, Paper 2022/975.

[6] Federal Information Processing Standards Publication 197, *Announcing the Advanced Encryption Standard (AES)*, November 26, 2001.

[7] Friedl S., *An Elementary Proof of the Group Law for Elliptic Curves*, Groups Complex. Cryptol, 2017.

[8] Ghonaimy Adeeb M., El-Hadidi Mahmoud T., Asian Heba K., *Security in the information society: Visions and Perspectives*, Springer-Verlag , 2002.

[9] Hoffstein Jeffrey, Pipher Jill, Silverman Joseph H., *An introduction to mathematical cryptography*, Springer, 2008.

[10] Kahn David, *The Codebreakers*, Scribner, 1996.

[11] Khan Mohammad R., Upreti Kamal, Alam Mohammad I., Khan Haneef, Siddiqui Shams T., Haque Mustafizul, Parashar Jyoti, *Analysis of Elliptic Curve Cryptography & RSA*, Journal of ICT Standardization, Vol. 11_4, 355-378, River Publishers, November 2023.

[12] Knapp Anthony W., *Elliptic Curves*, Princeton University Press, 1992.

[13] Koblitz Neal, *A Course in Number Theory and Cryptography*, 2nd edition, Springer-Verlag, 1994.

[14] Milne James S., *Elliptic Curves*, BookSurge publisher, https://www.jmilne.org/math/Books/ectext6.pdf, 2006.

[15] Nakamoto S., *Bitcoin: A peer-to-peer electronic cash system*, https://bitcoin.org/bitcoin.pdf, 2009.

[16] NIST, *FIPS 203, 204, 205*, https://csrc.nist.gov/projects/post-quantum-cryptography, 2024.

[17] NIST, *Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters*, https://doi.org/10.6028/NIST.SP.800-186, 2023.

[18] Nuida Koji, *An Elementary Linear-Algebraic Proof without Computer-Aided Arguments for the Group Law on Elliptic Curves*, International Journal of Mathematics, vol. 13, no.1, 2150001, 2021.

[19] Odlyzko Andrew M., *Discrete Logarithms in finite fields and their cryptographic significance*, Advances in Cryptology. EUROCRYPT 1984. Lecture Notes in Computer Science, vol 209. Springer, Berlin, Heidelberg, https://doi.org/10.1007/3-540-39757-4_20, 1985.

[20] Silverman Joseph H., *The Arithmetic of Elliptic Curves*, 2nd edition, Springer, 2009.

[21] Silverman Joseph H., Tate John, *Rational Points on Elliptic Curves*, 2nd edition, Springer, 2015.

[22] Singh Simon, *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*, Fourth Estate(UK), 1999.