# SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

## Cryptology & Elliptic curves

av

**Erik Liukko**

2023 - K9

# Cryptology & Elliptic curves

Erik Liukko

---

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Olof Sisask

2023

**Abstract:**

This thesis is about elliptic curve cryptography (ECC) which is a powerful form of public-key cryptography that utilizes the mathematical properties of elliptic curves over finite fields. Finite fields, play a crucial role in ECC by defining the arithmetic operations on curve points. The security of ECC lies in the difficulty of solving the elliptic curve discrete logarithm problem (ECDLP), which involves finding the scalar value that, when multiplied by a base point, equals a public point. ECC finds applications in key exchanges, digital signatures, and encryption, making it suitable for resource-constrained environments like mobile devices and the Internet of Things. Its adoption in industry standards and implementation in modern cryptographic systems is due to its strong security properties and efficient performance. In summary, ECC and finite fields provide a robust framework for secure communication and cryptographic protocols, enabling efficient and secure solutions for diverse applications in today's digital landscape.

# Contents

# 1 Introduction

This essay will focus on some of the fundamental structures in cryptology and we will go through some of the properties of elliptic curves and explain how they are implemented in today's field of cryptology. The first part of the paper will be a short preliminary part going through some important fundamental mathematical structures that will be used in the later chapters. After that, we will discuss some fundamentals of cryptology to get a better understanding of a few different methods that we then will use in the later chapters when we talk about elliptic curves and how they are used. We will also talk about a few things that we use cryptology for today such as the security of digital signatures and digital cash.

Cryptology is the study of encrypting and securing data. The practice of encrypting messages and hiding their meaning from unauthorized readers has been used for millennia by governments, military organizations, individuals, and other organizations to protect their secrets. Civilizations such as the Greeks and the Romans were known to use cryptology, however, early ciphers and codes were relatively simple compared to today's standards and were easily deciphered. The most famous example of this type of cipher is the Caesar cipher, named after the Roman emperor Julius Caesar. The Caesar cipher was a substitution cipher where each letter of the alphabet is replaced with another letter or symbol. Julius Caesar used this cipher to send messages to his officers during wartime, so even if the enemies got their hands on the messages they would not understand what was communicated[1]. Since then the evolution of mathematics allowed for the development of more sophisticated encryption methods, and the ones we use today are so complex that they would not be usable without the help of computers.

# 2 Preliminaries

This section introduces a few different mathematical terms so they easier can be understood in the later chapters of this essay.

## 2.1 Fields

A field is a set of elements that fulfills the axioms named in definition 1. A field can be both finite or infinite but the most used fields might be $\mathbb{R}$ or $\mathbb{Q}$, that is all real numbers and all rational numbers, but the ones we will mostly be talking about are finite fields.

[1] Simon Singh (2002). The Code Book. p. 14

**Definition 1.** A field is a set $F$ containing at least two elements that satisfy the following properties. The elements in $F$ need to work with two operations $+$ and $\cdot$ (called addition and multiplication) that are defined so that for each pair of elements $x, y \in F$ there are elements $x + y \in F$ and $x \cdot y \in F$. Also there needs to be elements $0, 1 \in F$ for which the following axioms are true for all elements $x, y, z, (-x) \in F$:

$$(i)\ x + y = y + x \text{ (commutativity of addition)}$$

$$(ii)\ (x + y) + z = x + (y + z) \text{ (associativity of addition)}$$

$$(iii)\ x + 0 = x \text{ (additive identity)}$$

$$(iv) \text{ there exists an element } (-x) \text{ such that } x + (-x) = 0 \text{ (additive inverses)}$$

$$(v)\ x \cdot y = y \cdot x \text{ (commutativity of multiplication)}$$

$$(vi)\ (x \cdot y) \cdot z = x \cdot (y \cdot z) \text{ (associativity of multiplication)}$$

$$(vii)\ (x + y) \cdot z = x \cdot z + y \cdot z \text{ (distributive)}$$

$$(viii)\ x \cdot 1 = x \text{ (multiplicative identity)}$$

$$(ix) \text{ if } x \neq 0 \text{ then there exists } x^{-1} \in F \text{ such that } x^{-1} \cdot x = 1 \text{ (multiplicative inverses)}$$

## 2.2 The modular inverse

As seen above in the definition of fields, an element in a field needs to have an inverse, except the element 0. Well since the fields that will be discussed in this essay will mostly be finite fields that use modular arithmetic to stay within the boundary of the finite field, we need to clarify what the modular inverse is.

**Definition 2.** Given two positive integers $a$ and $m$, the modular inverse of $a \ mod \ m$ is $b$ if $a \cdot b \equiv 1 \ mod \ m$. in other words, the modular inverse of $a \ mod(m)$ is the multiplicative inverse of $a \ modulo(m)$.

An example of how this is used is provided in example 1 in chapter 2.3 about finite fields below.

## 2.3 Finite fields

**Definition 3.** A finite field is a field that consists of a finite number of elements.

A finite field also known as a Galois field is a finite number of elements that satisfy the properties of being a field such as associativity, distributivity, and the existence of additive and multiplicative inverses. The number of elements in the field make up its size and is called the order of the field and the order of a finite field is always a prime number or a power of a prime number[2]. $GF(p^n)$ is the normal way to write the Galois field of order $p^n$, where p is a prime number and n is a positive integer. $GF(p)$ is also the field of residue classes modulo $p$ which means the possible elements in the field will be $0, 1, ...., (p-1)$. In Galois Field $(GF(p))$, elements are represented as integers modulo p. So when we say $a = b$ in $GF(p)$ it means the same as $a \equiv b \ mod(p)$.

**Theorem 1.** *If $GF$ is a finite field, then $GF$ most have $p^n$ number of elements, where $p$ is a prime number and $n \in \mathbb{Z}$.*

*Proof.* The proof of theorem 1 is provided in the attached source[3].

□

**Example 1.** An example of a small finite field is $\{0, 1, 2, 3, 4, 5, 6\}$ with addition and multiplication modulo 7. It is easy to see in this case that every operation regarding addition and multiplication involving the elements in the field will stay within the boundaries of the given finite field. Thus it is easy to see that the field fulfills the axioms $i - viii$. Then we also need to check if all of the elements have a multiplicative inverse except 0 which does not have a multiplicative inverse.

$$1 \cdot 1 = 1 \equiv 1 \ mod(7)$$

$$2 \cdot 4 = 8 \equiv 1 \ mod(7)$$

$$3 \cdot 5 = 15 \equiv 1 \ mod(7)$$

$$4 \cdot 2 = 8 \equiv 1 \ mod(7)$$

$$5 \cdot 3 = 15 \equiv 1 \ mod(7)$$

$$6 \cdot 6 = 36 \equiv 1 \ mod(7)$$

[2]Birkhoff, G. and Mac Lane(1996), S. A Survey of Modern Algebra, 5th ed. New York: Macmillan, p. 413

[3]John A. Beachy William D. Blair (2006), Third Edition Abstract Algebra, p.295

Because $\{0, 1, 2, 3, 4, 5, 6\}$ fulfills every axiom that is needed to be a field and have a finite element of numbers we can now surely state that this is a finite field.

So is every finite field that easy to "find"? We know that a finite field with 4 elements should be possible since $4 = 2^2$ so if we use the same example as above, adjusting for the different number of elements we get $\{0, 1, 2, 3\}$ with addition and multiplication modulo 4. When checking the inverses for the elements we find that 2 has no inverse, so this can not be a field. There is another way to build the fields if you have a number of elements that is a power of a prime.

**Example 2.** One field we can use that have 4 elements is $\{0, 1, \alpha, \alpha + 1\}$ where we add and multiply coefficients modulo 2 and $\alpha^2 = \alpha + 1$.

In this case, we can see that the addition of the objects will stay in the boundaries of the given field since the coefficients are regulated by modulo 2, ex $(\alpha + 1) + \alpha = 2\alpha + 1 = 1$.

When multiplication is used between the objects there will be times were you need to simplify by using $\alpha^2 = \alpha + 1$ to stay within the boundaries of the field, ex $\alpha \cdot (\alpha + 1) = \alpha^2 + \alpha = 2\alpha + 1 = 1$.

All the elements in the field also have inverses(except 0)

$$1 \cdot 1 = 1$$

$$\alpha \cdot (\alpha + 1) = \alpha^2 + \alpha = 2\alpha + 1 = 1$$

$$(\alpha + 1) \cdot \alpha = \alpha^2 + \alpha = 2\alpha + 1 = 1$$

therefore $\{0, 1, \alpha, \alpha + 1\}$ is a finite field under the given circumstances.

In cryptography, finite fields are used in the design of algorithms for secure communication between computers. This is because computers can not handle a nonfinite quantity very well. By using a finite field it is possible to ensure that information is handled with the preciseness that is needed while working with cryptology. This is also one of the reasons that makes society value big prime numbers as much as we do, since the bigger prime numbers the finite field uses the more elements it contains. We will talk more about why a lot of elements in a set is important in chapter 3.2.2 and 4.3.1.

## 2.4 The Fast Powering Algorithm

The fast powering algorithm is used to calculate a large powers of a number $g$ modulo $N$, where $N$ is a significantly large number. We will use this algorithm when we want to calculate the computation time to crack a cryptosystem in chapter 4.3.1.

**Theorem 2.** *Let $g, A$ and $N$ be constants, then the maximum amount of multiplications needed to calculate $g^A \equiv mod\ N$ will be $2\log_2 A$ when using the fast powering algorithm.*

*Proof.* So If we want to calculate a number $g^A \equiv mod\ N$ the conventional way would be to multiply $g$ by itself $A$ amount of times. If $A$ is a relatively small number $g^A$ is easy to calculate this way but if $A$ is a really big number the fast powering algorithm will compute $g^A$ way faster. the way the fast powering algorithm works is by splitting $A$ into a binary expansion, so

$$A = A_0 + A_1 \cdot 2^1 + A_2 \cdot 2^2 + A_3 \cdot 2^3 + ..... + A_r \cdot 2^r$$

Where $A_0, A_1, ...., A_{r-1} \in \{0, 1\}$ and $A_r = 1$. After this, the algorithm calculates all powers of $g^{2^i}\ mod\ N$ when $0 \leq i \leq r$, we will call each value of this $a_i$ as shown below.

$$a_0 \equiv g\ (mod\ N)$$

$$a_1 \equiv a_0^2 \equiv g^2\ (mod\ N)$$

$$a_2 \equiv a_1^2 \equiv g^{2^2}\ (mod\ N)$$

$$a_3 \equiv a_2^2 \equiv g^{2^3}\ (mod\ N)$$

$$.....$$

$$a_r \equiv a_{r-1}^2 \equiv g^{2^r}\ (mod\ N)$$

Since every term of $a$ is the square of the previous one we get that there is only needed one multiplication for each term, meaning a total of $r$ multiplications to calculate every term of $a$ from $a_0$ to $a_r$. The last step of the algorithm is to calculate $g^A$, and that is done the following way:

$$g^A = g^{A_0+A_1\cdot 2^1+A_2\cdot 2^2+A_3\cdot 2^3+.....+A_r\cdot 2^r} =$$

$$= g^{A_0}\cdot (g^2)^{A_1}\cdot (g^{2^2})^{A_2}\cdot (g^{2^3})^{A_3}\cdot .....\cdot (g^{2^r})^{A_r} =$$

$$= a_0^{A_0}\cdot a_1^{A_1}\cdot a_2^{A_2}\cdot a_3^{A_3}\cdot .....\cdot a_r^{A_r}\ (mod\ N)$$

And since every value of $a_i$ has been calculated and the values of $A_i$ can only be 0 or 1 the maximum amount of multiplications needed to calculate $g^A$ will be when every possible value of $A_i = 1$. This scenario will bring another total of $r$ multiplications, meaning the maximum amount of multiplications needed to calculate $g^A \equiv mod\ N$ to become $2r$. Another way to write how many multiplications at max is needed is $2\log_2 A$ since $A \geq 2^r$ □

# 3 Some fundamentals of cryptology

This chapter will discuss some of the more basic concepts that modern cryptology is built upon. This is to make sure that the understanding of the concepts that elliptic curve cryptology works with is in place before we start discussing them.

## 3.1 Hash functions

A hash function is a function that takes an input (ex a data file or password) and returns a fixed-size string of characters. For a hash function to be useful it should make sure that the same input will always produce the same output, but even a small change to the input will produce a very different output. Hash functions are used for many things such as data integrity and digital signatures, this will be discussed more in chapter 4.5.

**Definition 4.** Let $d$ be a arbitrarily long bit string so that $d = \{0,1\}^w$ and $w \in \mathbb{Z}$. A hash function takes $d$ as an input and transforms it to a hash value that has a length of $q$, $q$ being a positive integer. So we can write the transformation as follows

$$h : \{0,1\}^w \to \{0,1\}^q.$$

Here $\{0,1\}^w$ consists of all $0-1$ strings with only finitely many ones e.g. all the possible bit strings $d$ can have. While $\{0,1\}^q$ consists of all possible hash values the hash function can return. A specific hash function will always return hash values of the same length.

To better understand what makes a hash function good, here are some qualities that a good hash function should have.

1: If you hash a document the hash value of the document always has to be the same.

2: The calculation of the hash value needs to be fast.

3: The hash values that are possible for a hash function should be evenly distributed for all possible inputs, and even a small change in the input should change the hash value significantly.

4: The original document should not be possible to recreate from the hash value itself.

5: It should be very hard to find inputs to the hash function that gives the same hash value.

A hash function works by taking a binary string (that is the input/document) and transforming it into its hash value. The function can do this in many different ways but let us make a few examples of how it can be done.

**Example 3.** We shall describe two different hash functions called $Hash_1$ and $Hash_2$ that will hash a Document $D$. The two hash functions are not functions that are in use today. They are two that are made up just for us to get a better understanding of how a hash function could be able to work. Let's say that the document we want to hash is the following

$$D = 1001101100100111.$$

The way $Hash_1$ calculates its hash value is rather simple and very fast, $Hash_1$ will always take the two last numbers of the bit string of the document and that will be its hash value. So let's start by calculating the hash value of $D$ for the hash functions $Hash_1$. So, in this case, we get that

$$Hash_1(D) = 11$$

So $Hash_1$ transforms a bit string from its original length to a bit string of the length of two as shown bellow

$$Hash_1 : \{0,1\}^w \to \{0,1\}^2.$$

So let's now take a look at how $Hash_2$ calculates its hash value. The way $Hash_2$ works is that it first split the bit string of the document into sections of 4 bits so that

$$D = D_1|D_2|D_3|D_4$$

Then $Hash_2$ inverts every part of the document's bit string with an even index (in this case $D_2$ and $D_4$). Then it combines the parts of the documents in pairs, so $D_1$ and $D_2$ while $D_3$ and $D_4$ are combined and each pair will create a new bit string with a length of four. The way they are combined is by an operation called XOR, the way the XOR operation works is by taking two individual bits, and if the bits are the same the XOR operation makes the combined value a zero otherwise it will become one. When the XOR operation adds bit strings the bits will be added separately with the bit in the other bit string with the same index in the other bit string. So the way $Hash_2$ creates its hash value is as follows

$$D = 1001101100100111 \rightarrow 1001\ 1011\ 0010\ 0111 \rightarrow$$

$$\rightarrow 1001\ 0100\ 0010\ 1000 \rightarrow$$

$$\rightarrow 1101\ 1010.$$

So after this transformation, we also get the hash value from the second hash function $Hash_2$ for the document $D$ showing that

$$Hash_2(D) = 11011010$$

So $Hash_2$ transforms a bit string from its original length to a bit string of the length of eight as shown bellow

$$Hash_2 : \{0,1\}^w \rightarrow \{0,1\}^8.$$

But how does the $Hash_2$ function handle documents that are not of the same length as $D$ and still get a hash value that has the length of eight bits? The way $Hash_2$ solves this problem is to add zeros to the document if needed so that it has a length of $8 \cdot 2^c$ and $c \in \mathbb{N}$, it will always add as little amount of zeros as possible to meet the criteria. This means the amount of four-bit strings that the document is divided into will always be a power of two, the hash function $Hash_2$ can then add all pairs of the four-bit strings until there

are only two four-bit strings left. Bringing us a hash value that has a length of eight bits.

So let's take a look at $Hash_1$ and $Hash_2$ to see what good and not good qualities they have. We have already named 5 qualities that a good hash function should have. The first is that a hash function always creates the same hash value for a given document. This both $Hash_1$ and $Hash_2$ make sure of, and if this is not met the hash function can not be used for much. The second property that is valued is how quickly the function calculates its hash value. Since $Hash_1$ requires way fewer steps to calculate its hash value than $Hash_2$ it will be faster than $Hash_2$, but even if $Hash_1$ is faster, $Hash_2$ is still considered to be a fast function as well.

If we look at the third wanted quality for a hash function that is for their hash values to be evenly distributed and that small changes should drastically change the output. We can see that $Hash_1$ will have a perfectly even distribution of its hash values as long as the possible documents will be evenly distributed in their last two digits, and we assume that they are. $Hash_2$ should also have pretty evenly distributed hash values, but since the document will be given zeros in some cases it will affect its value so it is not as perfectly distributed but it will have a good spread of its hash values overall. The second part of criteria three is if the hash value changes a lot with even small changes in the input. $Hash_1$ is bad at this, since if not the last two numbers in the document change the hash value will not change, and since $Hash_1$ has a hash value that only is two bits long it can not change that much anyway. This $Hash_2$ does better than $Hash_1$ since every bit of the input will have an impact on the hash value of the function, although if only one bit is changed in the input a maximum of one bit is changed in the hash value for $Hash_2$, which is not a drastic change. The fourth quality is that one shall not be able to recreate the document from its hash value. This is true for both hash functions $Hash_1$ and $Hash_2$, as long as the document is longer than the hash value for the function, and most documents are way bigger than eight bits. The last and final quality we are looking for is that it shall be very hard to find inputs that create the same hash value. In $Hash_1$s case this is not hard, it only has four possible hash values so, therefore, it will not be long for someone to find two inputs that have the same hash value. $Hash_2$ does this way better than $Hash_1$. $Hash_2$ has 256 different hash values making it harder to find two inputs that have the same hash value than if it would only have four different hash values. Although in chapter 3.2 about collisions there will be made clear that even 256 different hash values are way less than is needed to make a hash function meet this criterion.

Now after comparing both $Hash_1$ and $Hash_2$ with the wanted qualities of a good hash function, we can say that both are considered too weak to use in today's society. But with that said $Hash_2$ has more of the wanted qualities than $Hash_1$ since it is more complex and has a way bigger set of possible hash values.

A way a hash function can be used is for password storage. The way, for example, a website can store the hash value of your password instead of storing the password itself. So when someone logs in the hash value of the entered password is compared to the stored hash value and if they match the user gets access to the information protected. This is used so if there is a data breach your password does not leak, only the hash value of it leaks. This is one of the reasons that a hash function is designed to be a one-way operational function meaning that one should not be able to recreate the original document from the hash value, the forth quality for a good hash function.

In some cases, a hash function can be "solved" meaning people learn how to generate a specific hash value at will. One example of a hash function with this problem is MD5. Because of this MD5 is not recommended to use as a hashing algorithm anymore since it is considered to be broken from a cryptographic point of view[4], which is why newer and more robust hash functions like SHA-256 are used in many applications and bitcoin[5].

## 3.2 Collisions

A collision algorithm is a method of finding matching objects in a set amount of objects. In other words, given a noninjective function $f$, a collision algorithm is designed to find two distinct inputs $x$ and $y$ such that $f(x) = f(y)$. In this chapter, the focus will be on the probability of a collision happening and not on the algorithms that are used to find collisions. An example of an algorithm is explained in theorem 8 in chapter 4.3.1 and how a collision can crack a cryptosystem. But in this chapter, we will focus on how the size of a set impacts the probability of collisions happening and how to calculate the probability of a collision happening.

The birthday paradox can show how easy it can be to find a collision if the set is not big enough. It also shows how much easier it is to find two persons with the same birthday than a person with a specific birthday.

### 3.2.1 Birthday paradox

The Birthday paradox is an example of how easy it is for collisions to happen in a data set if there are not enough given outcomes. The Paradox is based on the probability that in a group of $n$ people, there are at least two of them that

[4] Chad R Dougherty (2008). MD5 vulnerable to collision attacks. Carnegie Mellon University Software Engineering Institute.

[5] Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman (2014). An Introduction to Mathematical Cryptography Second Edition. p. 489

share their birthday. The number of people needed in the group so that there is a bigger chance that two people share a birthday than not is only 23 people. We begin by showing this.

**Example 4.** If we have a random group of 23 people what are the odds that two people share a birthday?

We start by naming the variables:

Let's call the event of at least two people sharing a birthday $A$

and

call the event of everyone having different birthdays $B$

The easiest way to solve this problem is to first calculate the probability that everyone has different birthdays in the group and then use that to calculate the probability that at least two people share their birthdays. Because there are only two possible outcomes in this scenario we get that

$$P(A) + P(B) = 1$$

must be true where

$P(A)$ = the probability that two people share their birthday

and

$P(B)$ = the probability that everyone has different birthdays.

The way we find $P(B)$ is that we take in one person at a time in the group until we have 23 people, We will assume that the people's birthdays are jointly independent random variables, meaning that we have no twins and so on in this example. So when taking in the first person in the group it does not matter which birthday the person has since there is no one that the first person can share a birthday with. The second person can now only have 364 of the possible birthdays since otherwise there would be two people sharing a birthday, the third person can then have 363 different possible birthdays, and so on. When using this way to solve the problem we get that the i:th person has a chance of sharing a birthday with someone else to be $\frac{365-(i-1)}{365}$, we do also assume that no person in this example is born on the 29:th of February(the extra day on a leap year). We can multiply all the probabilities of every person entering the group

sharing a birthday with someone already in the group to find the probability that everyone has different birthdays, we can do this since the people's birthdays are jointly independent random variables.

We use this to calculate $P(B)$:

$$P(B) = \prod_{i=1}^{23} \frac{365-(i-1)}{365} = \frac{365}{365} \cdot \frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{343}{365} = 0,4927 = 49,27\%$$

then we can use that to calculate P(A):

$$P(A) + P(B) = 1 \Leftrightarrow P(A) = 1 - P(B) \Leftrightarrow P(A) = 50,73\%$$

So as we can see there is a slightly higher chance that two people share a birthday than not if we have a group of 23 randomly selected people.

To get a better understanding of how often people share a birthday, figure 1 will show how likely it is that at least two persons share their birthday in a randomly selected group of a set amount of people. The graph in figure 1 uses the same principle as we use in example 1 ($\prod_{i=1}^{n} \frac{365-(i-1)}{365}$) showing the probability that two people have the same birthday in a group if the group is between 1 and 100 people. Figure 1 did take the possible groups containing 1 to 100 people then made a curve going through all the points that the different groups gave, this was done since a curve looks more esthetic than 100 different points.

As we can see in figure 1 it only shows the probabilities that someone shares a birthday of groups of people up to 100, but as we can see in the diagram, by then it is almost 100% chance that two people share a birthday and that will only continue after that.

### 3.2.2 A Collision Theorem

In this chapter, we will go through a more general way of collisions happening that instead of having a set of 365 elements as shown in the birthday paradox we will work with a set of $N$ numbers. From this set we will create two different sets, the sets will be called $n$ and $n^{-1}$, so $N = n + n^{-1}$.

If the numbers in $m$ are selected from the original set of numbers $N$, where every number is selected individually and the same number can be selected more than once. Then the following theorem will be true:

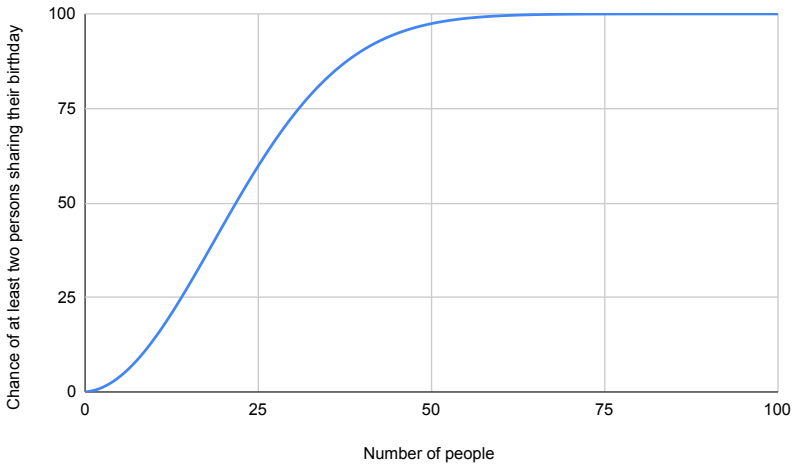


Figure 1: A diagram showing the chance that 2 persons share their birthday

**Theorem 3.** *If $m$ objects are selected individually from a set of $N$ elements, where $N$ can be divided into two subsets, $n$ and $n^{-1}$ so $N = n + n^{-1}$. Then the probability of selecting an element from $n$ will be:*

$$P(\text{at least one element from } n \text{ being picked}) = 1 - (1 - \tfrac{n}{N})^m$$

*Proof.* To prove this first we note that since there are two subsets of $N$ that:

$$\text{P(at least one element from } n \text{ being picked)} = 1 - P(\text{All } m \text{ elements picked being in subset } n^{-1})$$

Since the probability that one element from the set $n^{-1}$ is picked in one operation is $\frac{N-n}{N}$ and since the probability of an element being picked does not change by the earlier operations, meaning every element selected is an independent random operation. We get that the probability of picking elements from the subset $n^{-1}$ $m$-times in a row to be $\prod_{i=1}^{m} \frac{N-n}{N}$. We can substitute this into the earlier statement and get:

$$\text{P(one element from } n \text{ being picked)} = 1 - \prod_{i=1}^{m} \tfrac{N-n}{N} =$$

$$\text{We can rewrite } \prod_{i=1}^{m} \tfrac{N-n}{N} = \prod_{i=1}^{m} (\tfrac{N}{N} - \tfrac{n}{N}) = (1 - \tfrac{n}{N})^m$$

After knowing this we get that

$$\text{P(at least one element from } n \text{ being picked)} = 1 - (1 - \tfrac{n}{N})^m$$

□

This concept is one of the easiest ways of approximating how hard it can be to find collisions in a set of elements, and we can show how much the size of the set $N$ affects how hard it is to find a collision.

**Example 5.** Let's show how hard it is to find a collision when you have a set $N_1$ that includes 100 objects and $N_2$ that have 100 000 objects.

We note that since there are only two different outcomes in this example (either there is a collision or there is not). We can note that the probability of finding a collision is the same as the probability of choosing an element from $n$ in theorem 3.

We start by picking objects from the $N_1$ and then from $N_2$ and show how hard it is to find collisions depending on the size of the set.

$$N_1 \text{ picking 10 numbers:}$$
$$\text{P(a collision picking 10 objects from } N_1) = 1 - (1 - \tfrac{10}{N_1})^{10} \approx 65\%$$

$$N_1 \text{ picking 50 numbers:}$$
$$\text{P(a collision picking 50 objects from } N_1) = 1 - (1 - \tfrac{50}{N_1})^{50} \approx 100\%$$

Now we show the probability of a collision when the set is much bigger, as it is in $N_2$.

$$N_2 \text{ picking 10 numbers:}$$
$$\text{P(a collision picking 10 objects from } N_2) = 1 - (1 - \tfrac{10}{N_2})^{10} \approx 0,1\%$$

$$N_2 \text{ picking 50 numbers:}$$
$$\text{P(a collision picking 50 objects from } N_2) = 1 - (1 - \tfrac{50}{N_2})^{50} \approx 2,47\%$$

As we can see in example 2 the size of the set $N$ matters a lot if you want to make it hard to find collisions in a set. To make a cryptosystem secure there needs to be extremely hard to find collisions, otherwise, intruders can crack the system that is used, and that makes it insecure. More about how that works in chapter 4.3.1.

## 3.3 Ciphers

A cipher is a method of encrypting a message to make sure that only the ones who intended to get the message can read it. The most common way of encrypting a message like this is to switch around letters, numbers, and symbols with other characters until the message becomes unreadable to most people and machines. What a cipher uses to transform the messages is called a key, and a key is for both the decryption and encryption of the messages. Depending on how the cipher is structured, there can be either one key that both encrypts and decrypts the cipher or there can be two keys where one encrypts and the other decrypts, this is the difference between symmetric- and asymmetric ciphers. The security of a cipher depends on the secrecy of the encryption key, the decryption key, as well as the complexity of the cipher itself.

**Definition 5.** A Cipher is pair of functions, one encryption function($E$) and one decryption function ($D$). This function is used to transform an input ($m$), where $m \in M$ and $M$ is a set of all possible inputs, into a cipher text ($c$), where $c \in C$ and $C$ is a set of all possible cipher texts. Both $E$ and $D$ use a key ($k$) chosen from a set $K$ that is the set of all possible keys for the transformations, the key has two different parts one encryption part ($k_e$) and one decryption part ($k_d$). The way the transformations work is as follows:

$$(k_e, k_d) \in K$$

$$E : k_e \times M \to C$$

$$D : k_d \times C \to M$$

In definition 5 $k_e \times M \to C$ means that the encryption key is used to transform the input into the ciphertext and $k_d \times C \to M$ means that the decryption key is used to transform the ciphertext back to the original input. This means that the following needs to be true for a cipher to work as intended

$$D(k_d, E(k_e, m)) = m$$

meaning that someone can both transform the message to make it unreadable and then back to make it readable again.

### 3.3.1 Symmetric ciphers

**Definition 6.** A symmetric cipher is a cipher where $k_e = k_d$

Symmetric ciphers use the same key for both encryption and decryption of data, therefore making the keys symmetric hence the name symmetric ciphers. There are different sorts of symmetric ciphers which transforms the data in different ways. The first one is stream ciphers and the second is block ciphers[6]. Stream ciphers use an algorithm that encrypts data one byte(which is 8 bits) at a time to make the cipher text out of the original message. Block ciphers instead of encrypting the data byte by byte the plain text is divided into blocks of a fixed size and then transformed block by block to create the cipher text, it usually uses blocks with 64 or 128 bits in size.

One of the main advantages of block ciphers over stream ciphers is their security. They are generally considered to be more secure than stream ciphers as they are less vulnerable to certain attacks. But the things that make block ciphers more secure than stream ciphers are also what makes them slower to operate. Therefore stream ciphers are used more widely than block ciphers when data needs to be encrypted in real-time when the speed of the cipher is valued higher. Some applications that typically would fit this genre are wireless communication or video streaming. Block ciphers are instead used more in cases like digital signatures when the speed is not as important but security is a big focus and other applications where data is not transferred in real-time.

**Example 6.** Let's show what a simple symmetric cipher can look like. The operation we are gonna use to combine the input and the key will be the XOR operation explained in example 3, we will denote the XOR operation as $\otimes$ in this example. Of course, the key for encrypting and decrypting will be the same since we are working with a symmetric cipher, so we will just call the used key $k$. So let's name the input that we want to encrypt

$$m = 1011$$

and the key we will be using

$$k = 0010.$$

So if we first are going to encrypt the input that we call $m$ it would be done as followed

[6]Pelzl & Paar (2010). Understanding Cryptography. Berlin: Springer-Verlag. p. 30

$$E(k, m) = k \otimes m = 0010 \otimes 1011 = 1001$$

So the encrypted message is now 1001. If we now decrypt 1001 we should have the start input that is 1011, so let's see if that is the case.

$$D(k, E(k, m)) = k \otimes E(k, m) = 0010 \otimes 1001 = 1011$$

Now we have both encrypted and decrypted the original input and we can see that it is the same before and after the transformations e.g. the cipher works as intended and since the key is the same it is a symmetric cipher.

### 3.3.2 Asymmetric ciphers

**Definition 7.** An asymmetric cipher is a cipher where $k_e$ and $k_d$ may be different.

Asymmetric ciphers use different keys for encryption and decryption of data, therefore making the keys asymmetric hence the name asymmetric cipher. The two keys in an asymmetric cipher are called a private and a public key. The private key is used to decrypt data while the public key is used to encrypt the message. The different keys need to be handled differently, the public key is fine to share with anyone without compromising the security of the cipher while the private key needs to stay secret for the line of communication between the two parts to be secure[7].

As demonstrated in *Figure 2* we can see that no messages can be compromised by the public key since the public key just is used for sending messages that only Alice can decrypt, but if someone gets access to Alice's private key that person will have the possibility to get all of her private messages. Many everyday applications make use of asymmetric ciphers to secure their content such as email communication, online transactions, and securing digital signatures.

**Example 7.** Let's show what a simple asymmetric cipher can look like. We will show how an asymmetric cipher looks like using an RSA cryptosystem, we will have a private key and a public key that looks as followed,

the public key

[7]Stallings, William (1990). Cryptography and Network Security: Principles and Practice.Prentice Hall. p. 165

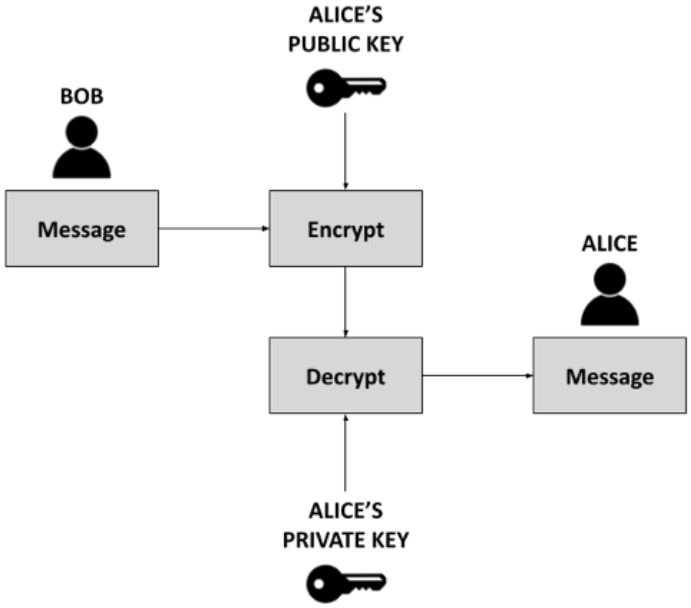


Figure 2: A typical Asymmetric cipher

$$k_{pub} = (p_{pub}, N)$$

and the private key

$$k_{priv} = (p_{priv}, N).$$

The keys will be used to encrypt $E$ a message $m$ into a cipher text $c$ and decrypt $D$ the cipher text back to the original message, as shown below

$$E : m^{p_{pub}} mod(N) \equiv c.$$

$$D : c^{p_{priv}} mod(N) \equiv m.$$

It is important to know that $k_{priv}$ and $k_{pub}$ have been made together to make sure that they work as intended and one can not just combine any numbers and expect them to work as intended. But since we are not going to use RSA in the later stages of this essay we will just use an example of this to show what an

asymmetric cipher can look like and not the process that is used to make the keys.

Let say that we have a message $m = 2$, a public key $k_{pub} = (5, 14)$ and a private key $k_{priv} = (11, 14)$. Then we start by encrypting the message

$$E : 2^5 mod(14) \equiv 4$$

so we get that the cipher text $c = 4$ in this case. Then if we want to decrypt it to get the original message we will need to use the private key. Then we get

$$D : 4^{11} mod(14) \equiv 2.$$

So as we can see we get the original message again after encrypting and decrypting it as intended, hence the procedure works as intended.

# 4 Elliptic curves

An elliptic curve is related to an equation of the form $Y^2 = X^3 + AX + B$ where $A$ and $B$ are constants. What makes elliptic curves so widely used in the field of cryptology is that they possess remarkable properties in that they allow for the combination of two points to produce a third point and that the curve is a reflection of itself in the x-axis. The procedure that these qualities can be used for is called "adding" or "points addition", although the procedure is not exactly like regular addition between constants it shares some vital properties that make it viable to use in cryptology. The values that point addition shares with regular addition are that they are both commutative and associative and there is an identity. Initially, we will focus on elliptic curves over the field $\mathbb{R}$ to establish the geometrical framework of point addition. Afterward, we will transition to elliptic curves over finite fields, which are used in modern cryptography.

**Definition 8.** An elliptic curve $E$ over $\mathbb{R}$ is a set of real solutions to the equation

$$E : Y^2 = X^3 + AX + B,$$

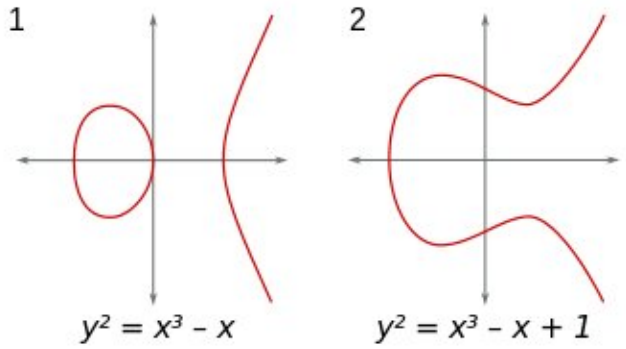


Figure 3: Example of two elliptic curves

where $A$ and $B$ are constants satisfying $4A^3 + 27B^2 \neq 0$, also a point at infinity $\mathcal{O}$ needs to be included in the set.

**Theorem 4.** *Let $l$ be a line that intersects an elliptic curve $E$ in two points. Then $l$ intersects $E$ exactly three times as long as the two points are not reflections of each other in the x-axis, counting multiplicity.*

*Proof.* To prove this first we take a look at a general line and elliptic curve, which brings us

$$E : y^2 = x^3 + ax + b$$

$$l : cy = dx + g$$

where $a, b, c, d, g$ are constants $\in \mathbb{R}$ while $x$ and $y$ are the variables in the functions. If we set $d \neq 0$ we get that

$$x = \tfrac{cy-g}{d}.$$

We can now use the expression for $x$ in the elliptic curve so that we get a one variable function of $y$ when $c \neq 0$. If we do this we get

$$y^2 = x^3 + ax + b \Leftrightarrow y^2 - x^3 - ax - b = 0 \Leftrightarrow$$

$$\Leftrightarrow y^2 - (\tfrac{cy-g}{d})^3 - a(\tfrac{cy-g}{d}) - b = 0.$$

This is a polynomial with a leading factor of $y^3$. We already have two roots of $y$ being $y_1$ and $y_2$ from the known points where the line $l$ intersects with $E$. Given that we have two existing roots, the equation will fully split, and since the leading coefficient is $y^3$, we will obtain a third root, denoted as $y_3$ that brings a third intersection point.

There are two cases where the calculation above does not apply, those are when $d = 0$ and when $c = 0$. So we will take a look at them.

When $c = 0$ we get that $x = \frac{-g}{d}$, since $\frac{-g}{d}$ is a constant we know that the two points have the same $x$ coordinate, in that case, the two points that intersect the line and the elliptic curve most be reflections of each other in the x-axis since an elliptic curve is a function that is a reflection of itself in the x-axis.

When $d = 0$ the function of the line will be $y = \frac{g}{c}$, if we then use the value of $y$ in the function for the elliptic curve we get

$$(\tfrac{g}{c})^2 - x^3 - ax - b = 0.$$

This is a polynomial with a leading factor of $x^3$, and this brings a third intersection point in the same way that the polynomial with a leading factor of $y^3$ did earlier in this proof.

□

Theorem 4 shows that point adding is possible but that does not necessarily mean that the same is true for point dubbing. We need another theorem for that.

**Theorem 5.** *A tangent line $l$ in a point on an elliptic curve will always have a second intersection point as long as the tangent line is not parallel to the y-axis.*

*Proof.* The proof is almost the same as the one in theorem 4, except instead of knowing two points on the curve we know one point on the curve that is a double root to the equation that the function of the line and the elliptic curve brings.

□

Now to define the procedure that is points addition.

**Definition 9.** Let $P$ and $Q$ be two points on an elliptic curve $E$ over $\mathbb{R}$. Then $P + Q$ on $E$ is as follows.

($a$) if $P \neq Q$

To add $P$ and $Q$ in this case we draw a line through the two points and find where it intersects the curve again as proven in theorem 4. The intersection point (the point $-R$ in Figure 4) is then reflected across the x-axis to find the point $R$ and $R = P + Q$.

($b$) if $P = Q$

This uses a method that looks similar to the one in ($a$) but instead of drawing a line between two points the line that is used will be the tangent line in the point $P$. This line will then have a second intersection point as theorem 5 suggests that is not equal to $P$ and $Q$, this second intersection point will then be reflected across the x-axis to find the point that is equal to $P + Q$ or $P + P$ in this case. This is called point doubling and an example of this is provided in figure 5.

**Example 8.** Let's calculate an example of points adding.

Let's work with the elliptic curve

$$E : y^2 = x^3 - x + 1$$

and we want to add the points

$$P = (1, 1) \text{ and } Q = (0, 1).$$

To calculate what $P + Q$ is we need to find the line that intersects both points, in this case, it is the line $y = 1$. Then we want to find the intersection points between $y = 1$ and $E$, therefore we substitute $y = 1$ into $E$ and get:

$$1 = x^3 - x + 1 \Leftrightarrow x^3 - x = 0 \Leftrightarrow x(x^2 - 1) = 0$$

we know that there are three solutions for $x$, and by using the points $P$ and $Q$ we can get that $x_1 = 1$ and $x_2 = 0$. We can then see that $x_3 = -1$

Since the line $y = 1$ has its third intersection point of the elliptic curve $E$ in $x = -1$, the intersection point $-R = (-1, 1)$. Then to get what $P+Q$ is we need to reflect the point $-R$ in the $x - axis$, which gives us the point $R = (-1, -1)$ where $P + Q = R$. This example is visualized in Figure 4.

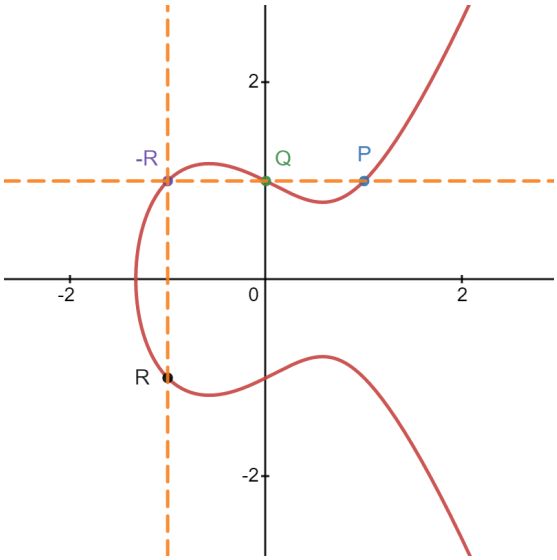


Figure 4: Visualising the addition between points P and Q

#### 4.0.1 The point at infinity

So as we can see looking at Figure 4 if we would like to add the points $R$ and $-R$ there will be no third intersection point. This is true since $R$ and $-R$ are reflections of each other in the x-axis, so the line that intersects both of them will be parallel to the $y-axis$ and all lines parallel to the $y-axis$ will never intersect with an elliptic curve more than twice. This is a problem since points adding is based on there being a third intersection point.

The problem is solved by introducing an extra point $\mathcal{O}$ that exists "at infinity" hence the name of it, the point at infinity. The point $\mathcal{O}$ therefore does not exist in the regular $xy-plane$ but we will use it like the point exist at the end of every vertical line while adding points, so $-R+R=\mathcal{O}$.

Adding a regular point with the point at infinity will always be equal to itself so $R+\mathcal{O}=R$, hence the line that intersects $R$ and $\mathcal{O}$ will be a vertical line and the third intersection point will be the point $-R$ (the first two points being $R$ and $\mathcal{O}$). When reflecting $-R$ in the $x-axis$ we get the point $R$ proving that $R+\mathcal{O}=R$ for all points $R$ that are on a given elliptic curve.

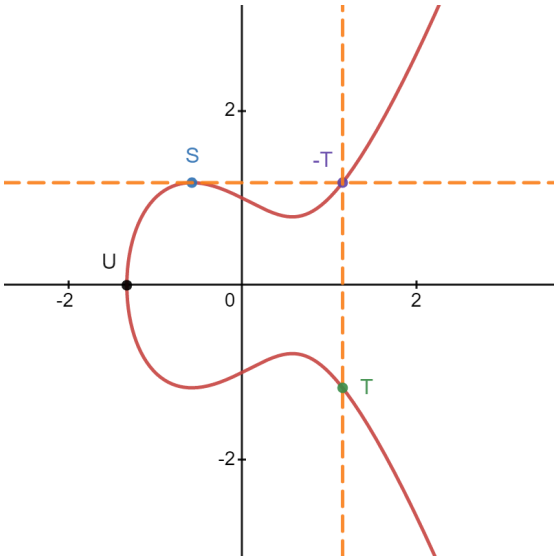


Figure 5: Visualising of doubling the point S

## 4.1 The properties of Elliptic curves

Now that the basic concepts of point addition have been explained we will dive into the more general qualities of the procedure that can make it useful.

**Theorem 6.** *Let $P$,$Q$,$R \in E(\mathbb{R})$ be points over an elliptic curve over $\mathbb{R}$, and let $\mathcal{O}$ be the point at infinity. Then adding points over an elliptic curve $E$ will have the following four qualities.*

$$(i)\ P + \mathcal{O} = \mathcal{O} + P = P$$

$$(ii)\ P + (-P) = \mathcal{O}$$

$$(iii)\ P + Q = Q + P$$

$$(iv)\ (P + Q) + R = P + (Q + R)$$

*Proof.* The laws $(i)$ and $(ii)$ have been clarified in chapter 4.0.1 and in $(iii)$ we can see is true since the line that someone draws thru $P$ and $Q$ will be the same line that is drawn through $Q$ and $P$. But the associative law $(iv)$ is more complex to prove and will not be provided in this essay [8]. □

[8] J.H. Silverman, J. Tate (1993). Rational Points on Elliptic Curves. Undergraduate Texts in Mathematics. p. 18-20

Now let's generalize how to calculate adding points algebraically. This is to make it very clear how it is calculated in every possible scenario for all the different points that can be on an elliptic curve that is used for adding points.

**Theorem 7.** *This is the Elliptic Curve Addition Algorithm:*

*First let $E$ be an elliptic curve where $E : y^2 = x^3 + Ax + B$ and the points we will be adding is $P$ and $Q$ where $P$, $Q \in E$.*

*(a) If $P = \mathcal{O}$ then $P + Q = Q$ and if $Q = \mathcal{O}$ then $P + Q = P$*

*(b) If $P = (x_1, y_1)$, $Q = (x_2, y_2)$ while $x_1 = x_2$ and $y_1 = -y_2$ then $P+Q = \mathcal{O}$*

*(c) in all other cases $P + Q = (x_3, y_3)$,*

*where $x_3 = \xi^2 - x_1 - x_2$ and $y_3 = \xi(x_1 - x_3) - y_1$*

*and $\xi$ is defined as followed*

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

*Proof.* $(a)$ and $(b)$ has been made clear in the chapter 4.0.1, while $(c)$ is yet to be proven. In $(c)$ $\xi$ is defined to be the slope of the line that intersects both point $P$ and $Q$ if $P \neq Q$, or if $P = Q$ the tangent line thru $P$ and $Q$. Therefore the line that goes thru both points will always be the line:

$$y = \xi x + \gamma \text{ where } \gamma = y_1 - \xi x_1.$$

If we substitute the equation of the line $y = \xi x + \gamma$ in the equation of the elliptic curve we get:

$$(\xi x + \gamma)^2 = x^3 + Ax + B \Leftrightarrow$$

$$\Leftrightarrow x^3 - \xi^2 x^2 + (A - 2\xi\gamma)x + B - \gamma^2 = 0$$

Since $x_1$ and $x_2$ are known it is easy to find the $x-$coordinate $x_3$ of the remaining point on the line satisfying the equation

$$x^3 - \xi^2 x^2 + (A - 2\xi\gamma)x + B - \gamma^2 = (x - x_1)(x - x_2)(x - x_3) = x^3 + x^2(-x_1 - x_2 - x_3) + x(x_1x_2 + x_1x_3 + x_2x_3) + x_1x_2x_3$$

If we look at the $x^2$ terms we see that $x_3$ satisfies

$$\Leftrightarrow -\xi^2 = -x_1 - x_2 - x_3 \Leftrightarrow$$

$$\Leftrightarrow x_3 = \xi^2 - x_1 - x_2$$

Now the $x$ coordinate ($x_3$) is proven to follow the rules of theorem 7. The $y$ coordinate ($y_3$) is given by reflecting the third intersection point of the line $y = \xi x + \gamma$ in $E$ across the $x - axis$. The $y-$coordinate $y_{int}$ of the third intersection point then satisfies

$$y_{int} = \xi x_3 + \gamma = \xi x_3 + y_1 - \xi x_1 = \xi(x_3 - x_1) + y_1$$

By negating $y_{int}$ we get that $y_3 = \xi(x_3 - x_1) - y_1$ and that makes all the properties of the elliptic curve addition algorithm proven.

□

## 4.2 Elliptic curves over finite fields

Elliptic curves over finite fields are used for a lot of different mathematical concepts but one way that they are used is to make computers able to handle elliptic curves and points adding so it can be used efficiently in today's society. This is because computers can not handle infinities very well, therefore we need a way to work with a set of data that has a finite number of elements.

**Definition 10.** An elliptic curve over a finite field is defined as follows,

we have an equation $E : y^2 = x^3 + Ax + B$ and a finite field $GF(p)$ where $p \geq 3$, also $A, B \in GF(p)$ and $4A^3 + 27B^2 \neq 0$. Then $E(GF(p)) = \{(x, y) : x, y \in GF(p)$ satisfy $y^2 = x^3 + Ax + B\} \cup \{\mathcal{O}\}$.

**Definition 11.** Points addition on an elliptic curve over a finite field is defined by the properties in theorem 7.

Points adding in elliptic curves over finite fields are calculated very similarly to the way of adding points in regular elliptic curves, although they are not as easy to follow visually. So let's make an example of how an elliptic curve over a finite field can look like:

**Example 9.** Let's take $E : y^2 = x^3 + 2x + 1$ over $GF(5)$ now we want to find all the points that are included $\in E(GF(5))$.

We know that the possible values of $x$ will be $0, 1, 2, 3$ and $4$ we can use that to find the points $\in E(GF(5))$.

When $x = 0$, $E$'s equation says $y^2 \equiv 1 \ mod(5)$ which has the solutions $y = 1, 4$. So we get the points

$$(0, 1) \text{ and } (0, 4) \text{ on } E.$$

When $x = 1$, $E$'s equation says $y^2 \equiv 4 \ mod(5)$ which has the solutions $y = 2, 3$. So we get the points

$$(1, 2) \text{ and } (1, 3) \text{ on } E.$$

When $x = 2$, $E$'s equation says $y^2 \equiv 3 \ mod(5)$ which has no solutions for $y$.

When $x = 3$, $E$'s equation says $y^2 \equiv 4 \ mod(5)$ which has the solutions $y = 2, 3$. So we get the points

$$(3, 2) \text{ and } (3, 3) \text{ on } E.$$

When $x = 4$, $E$'s equation says $y^2 \equiv 3 \ mod(5)$ which has no solutions for $y$.

So we get that $E(GF(5)) = \{(0, 1), (0, 4), (1, 2), (1, 3), (3, 2), (3, 3), \mathcal{O}\}$

So just as shown in example 9, we can take an elliptic curve with an infinite number of elements and use a finite field to create a set of only 7 points. If we were not able to do this the way we would encrypt and decrypt data today using elliptic curves would look very different. With that said, the fields used in modern cryptology are much larger than those demonstrated in example 9.

## 4.3 Elliptic Curve Discrete Logarithm Problem

The elliptic curve discrete logarithm problem is a way to explain how hard it is to decipher a message that is transformed using elliptic curve cryptology since one wants to make it hard to know how many times you add a point to itself.

**Definition 12.** Given an elliptic curve($E$) over a finite field($GF(p)$) and the points $P$ and $Q$ where $P$ and $Q \in E$ the elliptic curve discrete logarithm problem(ECDLP) is to find a number $n$ so that $P \cdot n = Q$. Where $n$ is the elliptic discrete logarithm of Q with respect to P or in technical terms $n = \log_P(Q)$.

In Definition 12 there is said that $n = \log_P(Q)$, for this to be correct $Q$ needs to be a multiple of $P$, otherwise $n = \log_P(Q)$ will not be defined. Therefore when someone is "building" the relation $P \cdot n = Q$ you start by selecting the point $P$ and the multiple $n$ to then calculate the point $Q$, in that way, it is sure that $Q$ is a multiple of $P$ and $n = \log_P(Q)$ will be defined.

The reason that one wants to make it hard to find the value of $n$ is because $n$ is the private key when using ECC. $Q$ is the public key and $P$ is a base point that is also known by the public. So the security of ECC relies on that the ECDLP is hard to solve.

**Example 10.** Let $E$ be the elliptic curve

$$y^2 = x^3 + 2x + 1 \text{ over } GF(23),$$

where

$$P = (4, 2) \text{ and } Q = (16, 14) \text{ are points on } E.$$

We know that there exists a constant $n$ so that $P \cdot n = Q$, our mission is to find the value for $n$.

So if there is known that $P \cdot n = Q$, because that $E$ works over the finite field there will be more numbers that solve $P \cdot n = Q$ such that $P \cdot n_1 = Q$, $P \cdot n_2 = Q$ and so on. This also means that a number $s$ exists so $P \cdot s = \mathcal{O}$ is true. Since $E(GF(p))$ has a finite number of points on it and $P \in E$, all multiples of the point $P$ can not be unique. Therefore the values $n_1$ and $n_2$ exists so that $P \cdot n_1 = P \cdot n_2$ where $n_2 - n_1 = s$ and $n_2 > n_1$. The value of $s$ is called the order of $P$ and can be used to find all the values that can solve $P \cdot n = Q$. Since $s$ is the number of operations between closely related solutions to $P \cdot n = Q$ we get the universal solution of the $n$ to be $n = n_1 + a \cdot s$ where $a \in \mathbb{Z}$.

Knowing this we need to find both a solution to $P \cdot n = Q$ and $P \cdot s = \mathcal{O}$. The way we will do this is by start adding the point $P$ to itself.

$$P = (4, 2)$$

$$2P = (16, 9)$$

$$3P = (15, 5)$$

$$4P = (17, 7)$$

$$5P = (8, 0)$$

$$6P = (17, 16)$$

$$7P = (15, 18)$$

$$8P = (16, 14) = Q \implies n_1 = 8$$

$$9P = (4, 21)$$

$$10P = \mathcal{O} \implies s = 10$$

After adding the point $P$ to itself ten times we get to the point at infinity, meaning the order that we call $s = 10$. While doing this we also find a solution to $P \cdot n = Q$ where $n = 8$, if you would not find it before the point at infinity there would not exist a solution to $P \cdot n = Q$. But now since we know the value of $n_1$ and the order we get that all possible solutions of $n$ to be $n = 8 + a \cdot 10$ where $a \in \mathbb{Z}$.

The way the ECDLP is solved in example 10 is not very effective. The average time to find a solution to the ECDLP using this method will be half of the order of $P$ ($\frac{s}{2}$ in example 10). In cases where the order will be much larger this method will be unfeasible to use. In the next chapter, we will go thru another way of solving the ECDLP that is more efficient than the one that is used in example 10.

### 4.3.1 Applying the collision algorithm

The point with the ECDLP is to make it hard to find the number $n$, where $n$ is the elliptic discrete logarithm of Q with respect to P. This is because when sending messages using ECC the number $n$ holds the "hidden" message that someone wants to keep safe, so if you solve $n$ in the problem $P \cdot n = Q$ you get access to the message that is being sent, but even if $n$ is hard to find it is not impossible. If we apply a collision algorithm to the ECDLP we can show an example of how a possible way of solving the ECDLP can look like. The way the algorithm works is that it uses randomly selected points in two different lists that look like the following.

$$\text{List 1:} \qquad a_1 \cdot P \ , \ a_2 \cdot P \ , \ a_3 \cdot P \ , \ ... \ , \ a_n \cdot P$$

$$\text{List 2:} \qquad b_1 \cdot P + Q \ , \ b_2 \cdot P + Q \ , \ b_3 \cdot P + Q \ , \ ... \ , \ b_n \cdot P + Q$$

The simulation of points in both lists will continue until a collision between List 1 and List 2 is found. When a collision is found then the user will get that $a_i \cdot P = b_j \cdot P + Q \Leftrightarrow Q = P(a_i - b_j)$ where $a_i - b_j$ will be a solution to $P \cdot n = Q$ which means that $a_i - b_j = n$. When a value for $n$ is found the message that the ECDLP holds will get in the hands of the "attacker" using the algorithm.

**Theorem 8.** *If we have a group $G$ and an element $g$ such that $g \in G$ and $g$ has the order $N$, meaning $g^N = e$ and there is no smaller number than $N$ making the statement true then, a solution to the discrete logarithm problem $g^x = h$ can with 98 % chance be found in $2\sqrt{N}$ operations by using a randomized algorithm, where every operation is an exponentiation in the group $G$.*

*Proof.* To prove this we first rewrite

$$x = y - z$$

after substituting this into $g^x = h$ we then get

$$g^x = h \Leftrightarrow g^y = h \cdot g^z.$$

Then to solve $g^y = h \cdot g^z$ we make two lists, one list with values from $g^y$ and one list of values from $h \cdot g^z$. The first list is made by choosing random values of $y$ between one and $N$ and creating a list of them. This will make the first list look like the following

$$\text{List 1: } g^{y_1}, g^{y_2}, g^{y_3}, ...., g^{y_n}$$

and then the second list is made by choosing random values of $z$ between one and $N$. That will look like

$$\text{List 2: } h \cdot g^{z_1}, h \cdot g^{z_2}, h \cdot g^{z_3}, ...., h \cdot g^{z_n}$$

all elements from both list 1 and list 2 will be in $G$. We can also note that the elements in List 1 lie in the set

$$S = \{e, g, g^1, g^2, ...., g^{N-1}\}$$

and since we assume that $g^x = h$ have a solution, $h$ needs to be a power of $g$ which means that $h \cdot g^{z_i}$ also need to be in the set $S$, where $z_i$ is one of the possible values of $z$. Because of this, we can use the fundamentals from theorem 3 to solve the probability of a collision between list 1 and list 2. This gives us the probability of a collision between the two lists after $n$ operations be

$$P(\text{a collision between list 1 and list 2}) = 1 - (1 - \tfrac{n}{N})^n.$$

After a collision is found between the two the discrete, meaning finding a solution to $g^y = h \cdot g^z$ the discrete logarithm problem is solved.

If we set the value of $n = 2\sqrt{N}$ we can get the accuracy that is searched for in theorem 8.

$$P(\text{a collision between list 1 and list 2 after } 2\sqrt{N} \text{ operations}) = \\ 1 - (1 - \tfrac{2\sqrt{N}}{N})^{2\sqrt{N}} \approx 98.17\%$$

Since $98.17\% > 98\%$ theorem 8 is correct.

□

But if we want to know how long it will take to compute a solution to the problem in theorem 8, we get the following. First, we know that both lists list 1 and list 2 are of the same length and the length of them we call $n$, thus it requires roughly $2n$ steps to construct each list. Every single step of constructing these lists needs us to compute $g^i$ when $i$ is a number between 1 and $N$. Then using theorem 2 we get that every one of these operations takes roughly $2\log_2(i)$ group multiplications to compute one value if $g^i$. So now knowing how many multiplications it takes to do one operation and how many operations that is needed to assemble the two lists we get that the total amount of multiplications needed is approximately $4n\log_2(N)$. Then it also takes about $n\log_2(n)$ multiplications to check for collisions between the two lists so the total number of multiplications be

$$4n\log_2(N) + n\log_2(n) = n\log_2(N^4) + n\log_2(n) =$$

$$= n\log_2(N^4 \cdot n)$$

So if we want to know how long the computation time in theorem 8 would be we can set $n = 2\sqrt{N}$. We call the computation time of theorem 8 $CTT8$ and this brings that

$$CTT8 = 2\sqrt{N}\log_2(N^4 \cdot 2\sqrt{N}) = 2\sqrt{N}\log_2(N^{4.5} \cdot 2) =$$
$$= 2 \cdot 4.5 \cdot \sqrt{N}\log_2(N \cdot 2^{1/4.5}) = 9\sqrt{N}\log_2(N \cdot 2^{2/9}).$$

So the computation time needed to do theorem 8 will be $9\sqrt{N}\log_2(N \cdot 2^{2/9})$.

## 4.4 Elliptic curve cryptology

Elliptic curve cryptology is a cryptographic system based on the specific qualities that elliptic curves over finite fields hold, and at the same time implements the features of an asymmetric cryptosystem.

### 4.4.1 Elliptic Diffie–Hellman Key Exchange

The elliptic Diffie–Hellman key exchange is an algorithm that two individuals can use to create a secure line of communication between one another. This chapter will discuss how an asymmetric cryptosystem can be used to set up a symmetric cryptosystem. Since a symmetric cryptosystem is way faster to operate than if you always use an asymmetric cryptosystem one rather wants to use a symmetric cryptosystem to communicate, but it is hard to set up a symmetric cryptosystem securely therefore one needs to use an asymmetric cryptosystem to set it up safely.

There is also a Diffie–Hellman key exchange that has the same purpose but we will focus on how the exchange works while using elliptic curves.

**Definition 13.** Let an elliptic curve $E(GF(p))$ exist and a point $P$ where $P \in E(GF(p))$. Then the elliptic Diffie–Hellman key exchange problem is to calculate $P \cdot k_a \cdot k_b$ from the values $P \cdot k_a$ and $P \cdot k_b$.

The elliptic Diffie–Hellman key exchange uses an asymmetric cryptosystem between two parts to create a secure symmetric cryptosystem between the same parties.

The way the key exchange works is, first there are two people that we will call $A$ and $B$. $A$ and $B$ agrees to use a elliptic curve $E(GF(p))$ and a point $P$ where $P \in E(GF(p))$. Then both parties choose a number, that will be their private key. Then they calculate the multiplies between their secret key ($k_a$ and $k_b$) and the shared point $P$.

$$\text{A: } P \cdot k_a = Q_a$$
$$\text{B: } P \cdot k_b = Q_b$$

Then when both $A$ and $B$ have their respective points $Q_a$ and $Q_b$. They will share their points so $A$ sends $Q_a$ to $B$ and $B$ sends $Q_b$ to $A$.

So how can the line stay secure even tho $A$ and $B$ send information over a not already secure line? After $A$ gets $Q_b$ and $B$ gets $Q_a$ both parts multiply their given point by their secret key, as follows.

$$\text{A: } Q_b \cdot k_a = Q_c$$
$$\text{B: } Q_a \cdot k_b = Q_c$$

So as we can see, both $A$ and $B$ get the same outcome after the procedure, $Q_c$. So even if someone else than $A$ or $B$ gets access to $Q_a$ or $Q_b$ they will not be able to find $Q_c$. $A$ and $B$ can then use $Q_c$ to set up a secure symmetric cryptosystem using $Q_c$ as the symmetric key.

*Proof.* The points $Q_c$ are always the same since $P \cdot k_a \cdot k_b = P \cdot k_b \cdot k_a$ according to the additive laws of elliptic curve addition.

□

With that said though the secure symmetric cryptosystem that uses $Q_c$ as the symmetric key, will only be as secure as the ECDLP is hard to solve for their chosen elliptic curve $E$, point $P$ and their respective private keys. This is because the system is set up using ECC where $P \cdot k_a \cdot k_b = Q_c$, this is the same form of $P \cdot n = Q$ that chapter 4.3 about the ECDLP discusses. In this case, $k_a \cdot k_b = n$ and that is the value the ones setting up the symmetric cryptosystem want to keep as secure as possible because if someone finds the value of $n$ the line that is set up will not longer be secure.

## 4.5 Digital signature

A digital signature is a way of using cryptology to make it possible to sign a document without being there in person to do a physical signing, but still let everyone know that you agree to what the document states. This has become a big part of our community in the last decades because most documents are not signed in person anymore.

Digital signatures use an asymmetric cryptosystem, this is so a private key and a public key to can be used to identify the person who wants to make

a signature, we will explain how this works by an example of how singing a document can look like using a digital signature.

**Example 11.** Let's call the signer Adam. If Adam wants to sign a document the first that is done is that the document $D$ that he wants to sign is hashed so that a fixed bit-string is created.

$$Hash(D) = D_{hash}$$

$D_{hash}$ is a bit-string where $D_{hash} \in \{0,1\}^q$ and $\{0,1\}^q$ is all possible hash values that the hash function used can create. Adam will then encrypt the bit-string $D_{hash}$ that is produced from the document $D$ by using his private key $k_{priv}$ creating a new privately encrypted bit-string that is the digital signature.

$$E(k_{priv}, D_{hash}) = D_{Adam}$$

Adam will then attach his digital signature $D_{Adam}$ to the document $D$ and in practice, it looks the same as physically signing a document.

After Adam has attached his signature to the document his part is done and the signed document is finished, although Bella who is Adams's business partner needs to make sure that the signature on the document is Adams's signature.

The way Bella checks if the signature is Adams's signature is by first hashing the document with the same hash function that Adam used to create the original bit-string $D_{hash}$ that Adam also used. After that, she will take Adams's digital signature and use the public key $k_{pub}$ to decrypt the bit string that Adam created.

$$D(k_{pub}, D_{Adam}) = D_{hash}$$

When Adams bit-string is decrypted, Bella will check if the decrypted bit-string is the same as the original bit-string from the document. If it is the same then she will know that Adam is the one who signed the document and if it's not then she will know that it is not his signature. One thing that differentiates a digital signature from a physical signature is that since a digital signature will look different every time since it is made by transforming the hash of the document, this also makes sure that the signature, in this case, $D_{Adam}$ can only be used for the document $D$. This is because otherwise when Bella is checking the signature, the signature will not match the hash value of the document it is signing making it invalid. The example of Adam and Bella's document being signed is visually shown in Figure 6.

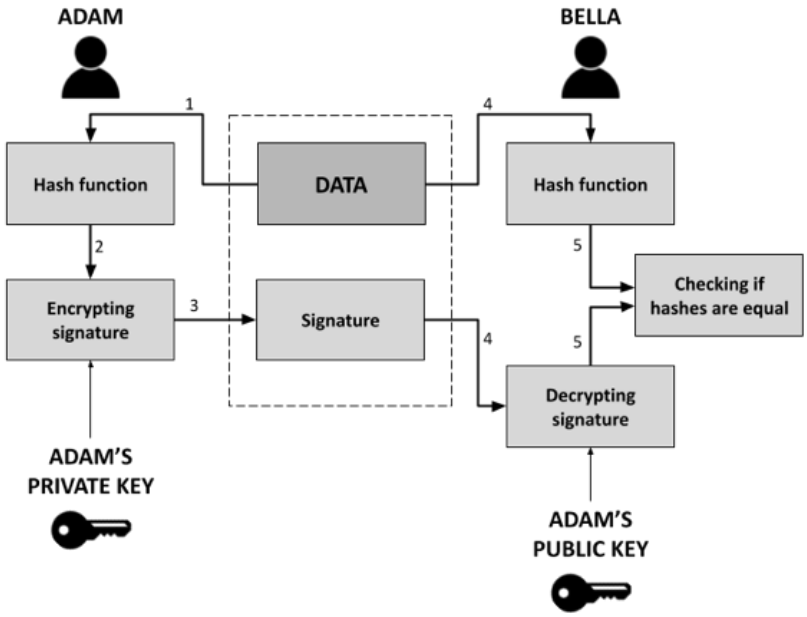


Figure 6: The process of Adam and Bellas's contract being signed

## 4.6 Blind digital signatures and digital cash

Blind digital signatures are a type of digital signature where the one that signs the document can not see the document that they are signing. This can be used for a variety of things like online voting systems or digital cash.

The reason blind signatures are used in digital cash is that the person using the digital cash still wants the anonymity that regular cash can give you. In other words, if you buy something with digital cash the organization that holds the digital cash you use (e.g. a bank) should not be able to know to who you are sending the money, but they still need to authorize the transaction. This is where the blind signature comes in. So the organization can approve what you want to do with your digital cash without ever being able to see what you want to do with it.

The way digital cash is explained in the paragraph above one still needs an organization that is trusted to control the digital cash. But a centralized organization does not control digital cash like Bitcoin or Ethereum. This is because these currencies instead use blind signatures to link every transaction to one another in a long chain, this is what is called blockchain. In these cases,

the transactions go thru when the transaction is attached to the chain, where every link in the chain is a previous transaction. So instead of an organization signing the transaction the blockchain approves the transaction by including it in the chain. To make the chain safe, much computing power is needed to make a transaction happen. It takes about 5-10 minutes to add a link to the chain and that is intentional so that one can not cancel a payment in a transaction or mess with the chain in another way[5].

If we take a closer look at the biggest blockchain based currency at the time of this essay, bitcoin. We can see that it uses an elliptic curve digital signature algorithm where the elliptic curve $E$, prime number $p$, and a specific point $P$ on the elliptic curve are known. They are:

$$E : y^2 = x^3 + 7$$

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

$$P = (x, y)$$

where $x$ and $y$ are numbers containing 77 digits[5].

$x$ and $y$ don't need to be 77 digits in general but it is just that the point in this case is chosen to be that.

The way the calculations to add a link to the chain works is that the computers try to find a specific input to the hash function that is used (for Bitcoin the one that is used is called SHA-256) that gives a hash value that begins with a specific amount of zeros. The computers will use brute force and try a very large amount of inputs in the hash function to try to find this specific output.

A way we get a better understanding of how hard it can be to find a hash value with this specific output is by looking at theorem 3. If we call the outputs that satisfy the specific conditions needed $n$ and all the other outputs $n^{-1}$ where $N$ are all possible outputs, and $N = n+n^{-1}$. We call the number of inputs tried in the hash function $m$, then theorem 3 states that the probability of finding the specific output in the number of tries to be

$$\text{P(at least one hash value from } n \text{ being found in } m \text{ tries)} =$$

$$= 1 - (1 - \tfrac{n}{N})^m.$$

So to make sure that it takes about 5-10 minutes to add a link to the chain, it is needed that the expected amount of hash values a computer can calculate in about 7,5 minutes is $m_a$. If $m_a$ gives

P(at least one hash value from $n$ being found in $m_a$ tries) = 50%.

To make sure that the time of finding a hash value in $n$ does not differ even when computers get faster, usually the hash functions that are used will be updated to have a more complex way to calculate the hash value so the time can stay consistent.

After finding the value that gives a wanted hash value the input will be linked to the transaction and add one more link to the chain. Once added to the chain the link will not change, therefore one can always follow the chain backward to see all the previous transactions that are connected in the chain[5].

# 5 References


1: Simon Singh (2002). The Code Book. p. 14

2: Pelzl & Paar (2010). Understanding Cryptography. Berlin: Springer-Verlag. p. 30

3: John A. Beachy William D. Blair (2006), Third Edition Abstract Algebra, p.295

4:Chad R Dougherty (2008). MD5 vulnerable to collision attacks. Carnegie Mellon University Software Engineering Institute.

5: Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman (2014). An Introduction to Mathematical Cryptography Second Edition. p. 489

6: Stallings, William (3 May 1990). Cryptography and Network Security: Principles and Practice.Prentice Hall. p. 165

7: Birkhoff, G. and Mac Lane, S. A Survey of Modern Algebra, 5th ed. New York: Macmillan, p. 413, 1996

8: J.H. Silverman, J. Tate (1993). Rational Points on Elliptic Curves. Undergraduate Texts in Mathematics. p. 18-20