1 Elliptic Curves

1.1 Geometry Intuition

Before we go on with more theoretical topics, let us examine another application of the quite abstract theory of algebra we have studied in the previous chapters, namely elliptic curve cryptography. It should be mentioned that linear algebra is by far not the only topic relevant in this context. In fact, important aspects of the theory of elliptic curves require the understanding of function analysis – where it actually comes from – but here we will focus on algebra and group theory.

Elliptic Curves (EC) provide the mathematical background for variants of public key cryptography. This kind of cryptography is being developed since the eighties, but it took a while until it was accepted by the industry. Today, however, it is the main public key cryptography scheme around. Its acceptance was accelerated by the smartphone boom. In smartphones and other devices with restricited resources, classic cryptographic schemes are not very practical. Their drawback is the computational overhead resulting from key size. Cryptoanalytic attacks forced classic schemes to be used with huge keys. To achieve 128-bit security with RSA, for instance, we need keys with at least 3072 bits. The same level of security can be reached with EC cryptography, according to known attacks today, with 256 bits. A huge improvement!

EC cryptography is different from classic cryptography in various respects. First, it includes much more math. That is to say, it does not include theory from only one or two branches of mathematics like number theory in classic cryptography, but from many different branches. This has huge impact on cryptoanalysis. Hidden attacks may lurk in apparently remote fields of mathematics that we did not account for. However, the theory surrounding EC is very well understood today and, as said, it is the mainline cryptography approach today.

Second, the basic means, especially the group we need for public key cryptography, are much more "engineered" than in classic cryptography. Classic schemes are based mainly on modular arithmetic, which was well known centuries before anyone thought of this use case. The groups found in modular arithmetic, in particular the multiplicative group, was then used to define cryptographic tools. In elliptic curves, there are no such groups "by nature". They are constructed on the curves with the purpose to use them in cryptography. Therefore, EC may sometimes feel a bit artificial. It is important to understand that the group we define on the curves is defined voluntarily according to our purpose. When we speak of *point addition* in this context, one must not confuse this operation with the arithmetic operation of addition. It is something totally different.

Anyway, what are elliptic curves in the first place? Elliptic curves are polynomials that were intensively studied in the late 19^{th} century, especially by German mathematician Karl Weierstrass (1815 – 1897), who was of huge importance in the sound fundamentation of analysis. We will meet him again in the third part. He studied polynomials of the form

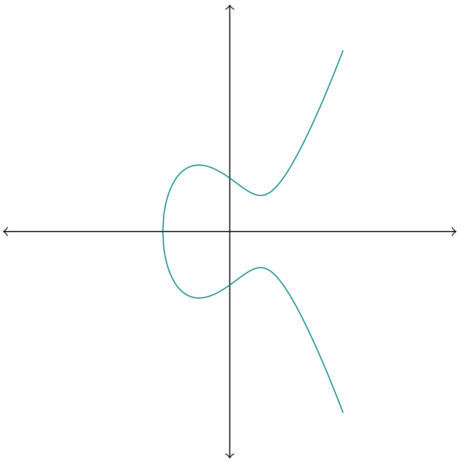
$$y^2 = x^3 + ax + b, \tag{1.1}$$

which is said to be in *Weierstrass form.* We can easily transform this equation into a form that looks more like something that can be computed, namely:

$$y = \sqrt{x^3 + ax + b}. \tag{1.2}$$

But be careful! Weierstrass polynomials are not functions, at least not in $\mathbb{R}$, since there is not exactly one y for each x. When the expression $x^3 + ax + b$ becomes negative, there is, in the world of real numbers, no solution for the right-hand side of the equation.

This is quite obvious, when we look at the geometric interpretation of that polynomial. It looks – more or less – like in the following sketch:

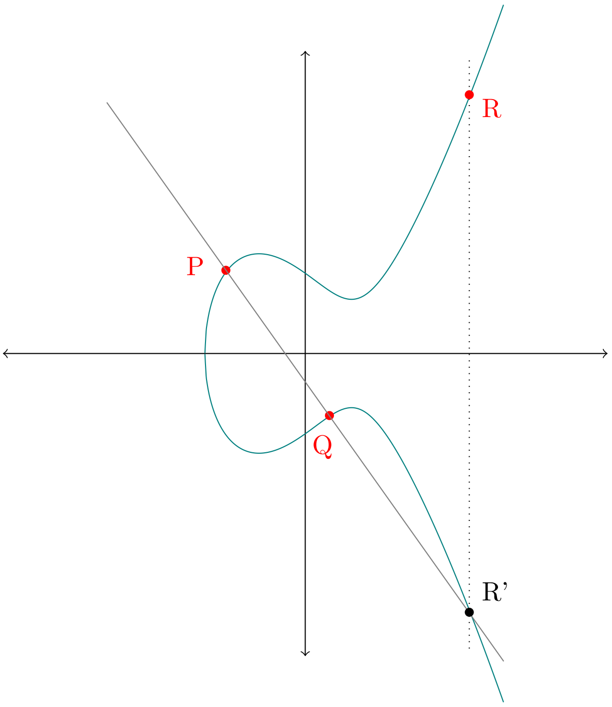


The exact shape depends on the coefficients a and b. The bubble on the left may sometimes be a circle or ellipse completely separated from the “tail” on the right; it may, in other cases, be less clearly distinguished from the tail on the right, forming just a tiny bulge in the tail.

In any case, the curve "ends" on the left-hand side for some $x < 0$. More precisely, it ends where the absolute value of x^3, for a negative value, becomes greater than $ax + b$. Then, the whole expression becomes negative and no real square root corresponds to it.

We will now start to construct a group on this kind of curves. We call it an *additional group*, but be aware that this is not addition in the sense of the arithmetic operation. It has nothing to do with that! It is a way to combine points with each other that can be captured in a – more or less – simple formula. We will start by giving a geometric interpretation of this operation. This will help getting an intuition. But, again, be aware that we are not dealing with geometry. We will soon deviate from geometry and talk about curves in a quite abstract way.

The following sketch shows an elliptic curve with three points P, Q and R, all coloured in red. These points are in the relation $P + Q = R$.



When adding two points P and Q on an elliptic curve, we draw a straight line through them (the grey one). From the nature of the elliptic curve, it is obvious that the straight line will meet the curve once again. At that intersection, we draw a helper point, R'. The we reflect this point across the x-axis, *i.e.*we draw another line (the dotted one) that goes straight up crossing R'. This line will meet the curve again, namely at a point with the same x coordinate, but with the inverse of the y coordinate $-y$. That point is R, the result of $P + Q$.

You see that this operation has in fact nothing to do with arithmetic addition. It is an arbitrary construction to relate three points. Nevertheless, it is carefully designed to give rise to a group based on this operation, as we will see later.

For the moment, our main question is how can we compute R from P and Q. We start by computing the straight line. A straight line is defined by a formula of the form

$$y = mx + c, \tag{1.3}$$

where m is the slope and c the y-intercept. What we need to do now is to find the third point, R', which, like P and Q, lies on both, the straight line and the elliptic curve. To find such a point, we set the two formulas equal. Since an elliptic curve is defined as

$$y^2 = x^3 + ax + b, \tag{1.4}$$

we can say

$$(mx + c)^2 = x^3 + ax + b. \tag{1.5}$$

By subtracting $(mx + c)^2$ from both sides, we get

$$x^3 + ax + b - (mx + c)^2 = 0. \tag{1.6}$$

Using the binomial theorem we can expand this to

$$x^3 - m^2x^2 - 2mxc - c^2 + ax + b = 0. \tag{1.7}$$

We already know two points, where this equation is fulfilled, namely x_P and x_Q. This means that these values are roots of the above equation. We can hence use them for factoring that equation into $(x - x_P)(x - x_Q)\Psi$, where Ψ is yet another factor. But we know even more. We just have to look at the sketch above to see that there are three roots and, hence, three factors. We, therefore, have $\Psi = x - x_{R'}$ and conclude that

$$x^3 - m^2x^2 - 2mxc - c^2 + ax + b = (x - x_P)(x - x_Q)(x - x_{R'}). \tag{1.8}$$

From here it is quite simple. We just apply the trick of the *opposite sum of the roots* and get

$$m^2 = x_P + x_Q + x_{R'}, \tag{1.9}$$

which we can easily transform to

$$x_{R'} = m^2 - x_P - x_Q. \tag{1.10}$$

Since R, the point we are finally looking for, is the reflection of R' across the x-axis, we have $x_R = x_{R'}$, *i.e.* the points have the same x-coordinate.

Computing $y_{R'}$ is again quite simple. The points P and R' are on the same straight line. The y-values on a straight line increase at a constant rate. So, we compute the value y should grow travelling on the segment between x_P and x_R, which is $m(x_R - x_P)$ and add this to the already known y-value at point P:

$$y_{R'} = y_P + m(x_R - x_P). \tag{1.11}$$

Now we compute y_R, the y-coordinate of the reflections of R' across the x-axis, which is simply $-y$. Alternatively, we can compute that value directly by rearranging the equation to

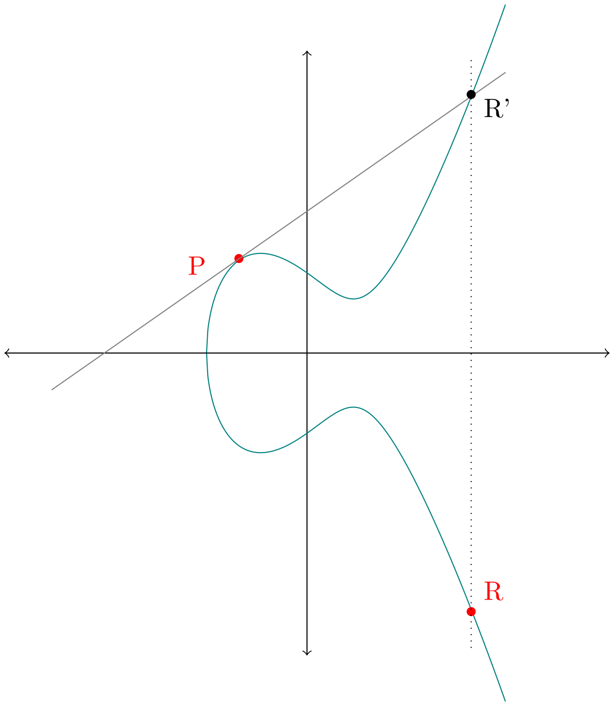
$$y_R = m(x_P - x_R) - y_P. \tag{1.12}$$

The final piece missing now is the slope, m, which can be expressed as a fraction:

$$m = \frac{y_Q - y_P}{x_Q - x_P}. \tag{1.13}$$

With this equation, however, we get into trouble. Everything is fine, when we assume that we add two distinct points P and Q. But if we have $P = Q$, *i.e.* if we want to add a point to itself, then the denominator of the above fraction becomes negative. That, clearly, is to be avoided.

To avoid that, we use, instead of a secant line that intersects the curve, the tangent line at point P, which, as we already know, measures the slope of the curve at P. Geometrically, this corresponds to the following sketch:



Here, we draw the tangent line at P. Where the tangent line intersects the curve again, we draw the helper point R'. We reflect it across the x-axis and obtain the point $R = P + P = 2P$.

As you hopefully remember, the slope of a curve at a given point can be calculated with the derivative of that curve. We will apply that derivative trick to get the tangent line at P. This task, however, is a bit more difficult than for the trivial cases we have seen so far. Until now, we have seen derivatives of simple functions like $f(x) = x^2$, whose derivative is $f'(x) = 2x$. Now, we have the equation

$$y^2 = x^3 + ax + b. \tag{1.14}$$

We can interpret this equation as an application of two different functions. The first function, say, g, is $g(x) = x^3 + ax + b$. The second function, f, is $f(x) = \sqrt{x} = x^{\frac{1}{2}}$.

For such cases, we have the *chain rule*, which we will discuss more thoroughly in part 3. The chain rule states that the derivative of the composition of two functions is

$$(f \circ g)' = (f' \circ g) \times g'. \tag{1.15}$$

That is, the derivative of the composition of two functions f and g is the derivative of f applied on g times the derivative of g. Let us figure out what the derivatives of our f and g are. The derivative of g is easy:

$$g'(x) = 3x^2 + a$$

A bit more difficult is f'. If $f(x) = x^{\frac{1}{2}}$, then

$$f'(x) = \frac{1}{2}x^{\frac{1}{2}-1} = \frac{1}{2}x^{-\frac{1}{2}} = \frac{1}{2x^{\frac{1}{2}}}.$$

Now, we apply this to the result of $g(x)$, which we can elegantly present as y^2. If we plug y^2 into the equation above, we get

$$\frac{1}{2y^{2\times\frac{1}{2}}} = \frac{1}{2y}.$$

We now multipy this by g' and get

$$\frac{3x^2 + a}{2y}.$$

When we use this formula for $x = x_P$, we get the formula to compute m:

$$m = \frac{3x_P^2 + a}{2y_P}. \tag{1.16}$$

So, we finally have an addition formula that covers both cases, $P \neq Q$ and $P = Q$:

$$x_R = \begin{cases} m^2 - x_P - x_Q & \text{if } x_P \neq x_Q \\ m^2 - 2x_P & \text{otherwise} \end{cases} \tag{1.17}$$

and

$$y_R = m(x_P - x_R) - y_P, \tag{1.18}$$

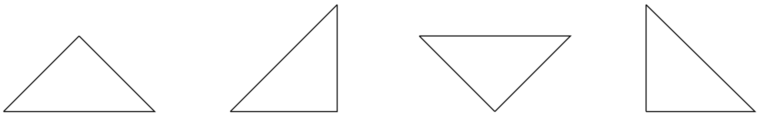
where

$$m = \begin{cases} \frac{y_Q - y_P}{x_Q - x_P} & \text{if } x_P \neq x_Q \\ \frac{3x_P^2 + a}{2y_P} & \text{otherwise.} \end{cases} \tag{1.19}$$

1.2 Projective Geometry

To complete the construction of the group of points on an elliptic curve, we still have to define the identity and the inverse. To do this we make a detour through the beautiful art of *projective geometry.* To be honest, we do not need too many concepts of projective geometry in practice. But with an intuitive understanding of those concepts the jargon common in EC cryptography becomes much clearer. Besides, projective geometry is really a beautiful part of mathematics worth studying whether we need it for EC or not.

Geometry is often concerned with difference and equality of quantities like length and angle. In this type of geometry, called *metric geometry*, one studies properties of objects under transformations that do not change the length and angle. A typical statement is, for instance, that two triangles are congruent (and hence equal), when one of them can be be as a roation or displacement of the other. The triangles below, for instance, are all congruent to each other:

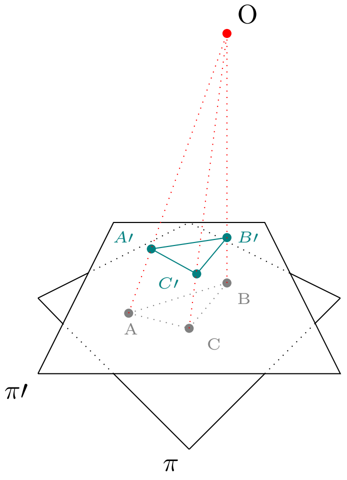


In fact, we could say it is four times the same triangle tumbling around. We are now looking at what remains from metric geometry, when we disregard length and angle as first-class properties of objects. One way to do so is by looking at logic configurations according to some basic notion, such as that of parallel lines. This gives rise to what is called *affine geometry.* Another way is to look at properties invariant under projective transformations and this is indeed what projective geometry does.

The triangles above are, as you can see, drawn on a plane. We could now take another plane, just as we would grab a piece of paper, and *project* the points on the first plane onto the second plane. The two planes do not need to fulfil any specific configuration. They may be parallel to each other or they may not. They may be arranged in any configuration relative to some orientation in the surrounding space. Indeed, we are now looking at transformations of two-dimensional figures on two-dimensional planes. But the transformations are created by projecting one figure through three-dimensional space onto another plane. Of course, we can generalise this to n-dimensional planes in an $n + 1$-dimensional space. But that would be far beyond our needs.

Projective transformations relate points on one plane, let us call it π, to points on the other plane, $\pi\prime$, by drawing a straight line that relates both points with yet another point, which, in *central projection*, is called the centre and is identical for all points we project from π to $\pi\prime$. We can also choose to use *parallel projection*, where points are projected by parallel lines, each one having its own projection "centre". As we will see later, in projective geometry, that is not a significant difference. The latter is just a special case of the first resulting from a very specific choice of the central point.

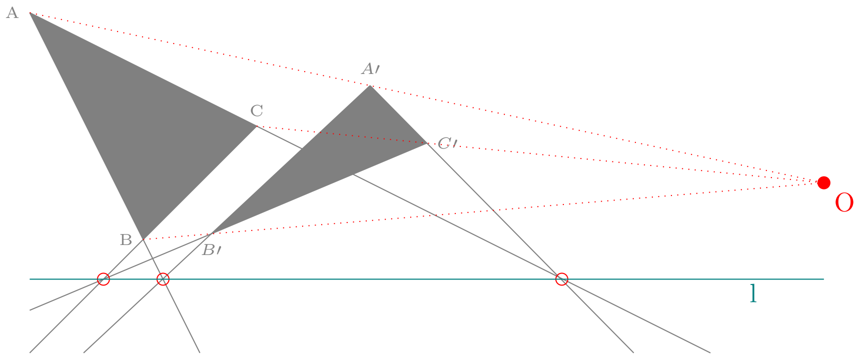
The following sketch shows a projection from π, the lower plane, to $\pi\prime$ using central projection with O being the central point:



There are, on π, three points, A, B and C, which form a triangle. The points are projected on $\pi\prime$ along the lines relating each of the points with O. We see that all points on π appear on $\pi\prime$ and we see that one of the properties that are preserved is that on both planes these points form a triangle pointing roughly in the same direction. The triangles, however, are of different size and, due to different arrangement of the planes in space, the shape of the original triangle is distorted on $\pi\prime$.

Projective geometry studies properties that remain unchanged under projection. Such properties are essential for us recognising projected shapes. It is therefore no surprise that projective geometry was originally introduced to mathematics by math-literate painters, in particular Renaissance artists like Leonardo da Vinci (1452 – 1519) and Albrecht Dürer (1471 – 2528). Today projective geometry is ubiquitous. It is used in all kinds of image processing and image recognition. It is widely used in digital cameras for instance, but also in many other kinds of applications.

The founding father of the mathematical discipline of projective geometry was the French engineer, architect and mathematician Girard Desargues (1591 – 1661). Desargues formulated and proved *Desargues' theorem*, one of the first triumphs of projective geometry. The theorem states that, if two triangles are situated such that the straight lines joining corresponding vertices of the triangles intersect in a point O, then the corresponding sides, when extended, will intersect in three points that are all on the same line. Here is a sketch to make that a bit clearer:



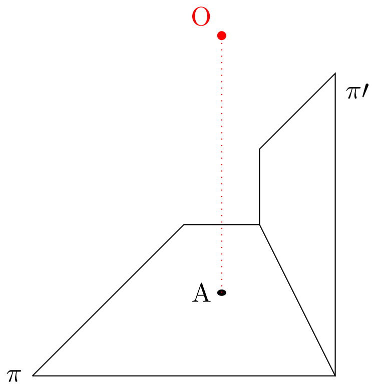
The dotted lines capture the theorem's precondition: corresponding vertices of the triangles lie on lines that intersect in one point O. The gray lines extend the sides of the triangles and the pairs of corresponding sides all intersect, each side with its corresponding side, on the same line l.

The theorem looks quite simple; after all, it contains only straight lines. It is nevertheless quite difficult to prove with means of metric geometry. If we consider the two triangles being on different planes, however, and one the projection of the other, the argument suddenly becomes very easy.

We first note that all points and lines making up one triangle are located on one plane. We then observe that each of the lines that relate one edge of one triangle with one edge of the other triangle, for instance $\overline{AA\prime}$ or $\overline{BB\prime}$, also lie in a plane, otherwise we could not draw these lines. But that means that A and $A\prime$, B and $B\prime$ and C and $C\prime$ as well as O all lie in the same plane. Therefore the lines $\overline{AB}$ and $\overline{A\prime B\prime}$ must meet somewhere. Since the triangles are in separate planes, the two planes must meet somewhere too and there, where the planes meet, there must be the intersection of all those lines. Two planes, however, meet in a line and, since they have exactly one line in common, it must be on that line where all the other lines itersect.

Note that this proof works with reasoning according to the logic of plane and space alone, which makes it concise and elegant, but also quite subtle. Indeed, I hesitate to put "□" to the end of the proof. In fact, there is a flaw in it. The proof only works when the planes are not parallel to each other! When we project the triangle onto a plane parallel to the first one, then these two planes will certainly never meet – and that crashes the proof.

That is a very typical situation in projective geometry. Theorems and proofs would look very nice and clean, had we not always those exceptions of parallel lines! In fact, there are even points on the original plane that will never appear in the projection, because their projective line is parallel to the second plane. Point A in the following configuration, for instance, with the projective centre at O will never show up on the target plane:



Projective geometry could be very clean and nice, was there not that issue of parallel lines. It comes into the way in every axiom and every theorem and every proof. Therefore, mathematicians tried to come around it. They did so by the following thought experiment. If we have two intersecting lines and now start to rotate them slowly so that they approximate the configuration where they are parallel to each other, the point of intersection moves farther away towards infinity. We could then assume that all lines intersect. There is then nothing special about parallel lines. They intersect too, but do so very far away, *viz.*at infinity. This way, we extend the concept of point and line by adding one point to each line, namely the point where this line and all lines parallel to it intersect. That point is then said *to be at infinity.*

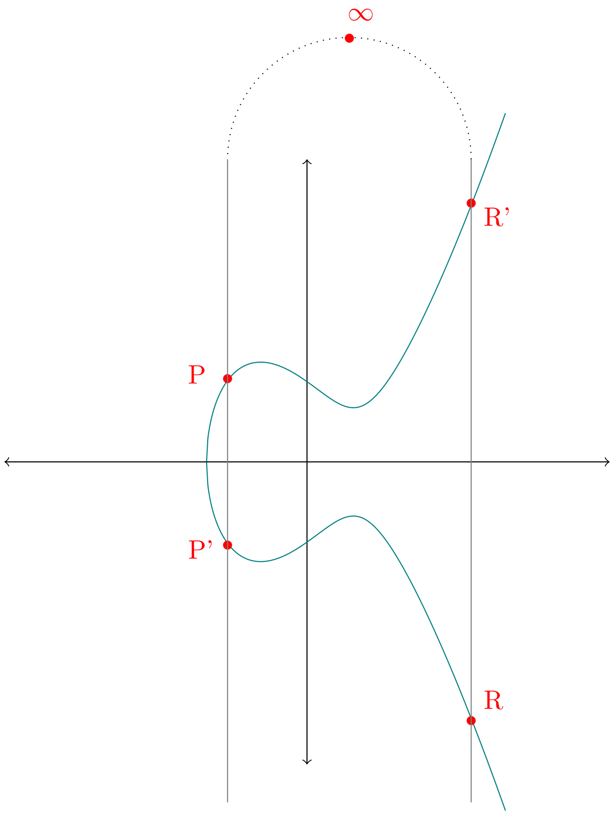
This trick to extend a concept is very similar to how we extended natural numbers to integers by adding a sign. Suddenly, we had a solution for problems that were unsolvable before, namely subracting a number from a smaller one.

Out there, at infinity, there are now many points, each one the intersection of an infinite number of parallel lines crossing it. Of course, we now can draw a line through all these points at infinity, the *line at infinity.* That is where the planes in the proof of Desargues' theorem intersect in the case where they are parallel to each other. We then only have to prove that, if one pair of lines intersect at infinity, the other two as well intersect at infinity.

It is perhaps worth to emphasis that this is not the result of observation of physical reality. Nobody has ever seen two lines intersecting at infinity. It is not a statement about physics at all. It is an axiom that we assume, because it makes reasoning in projective geometry much easier.

This handling of parallelism separates the two approaches to geometry, affine and projective geometry. Indeed, in affine geometry the notion of parallel lines is central. Many problems in that branch of mathematics are centred on the implications of one line being parallel to another or not. In projective geometry, by contrast, two lines being parallel to each other is nothing special. It just means that their point of intersection is very far away. We will discuss this again later in more detail.

Let us now come back to elliptic curves. Where do parallel lines play a role in our addition formula? Well, the line to reflect a point across the x-axis is parallel to the y-axis and all such lines are parallel to each other. In projective terminology, all these lines intersect at infinity. In other words, a point and its reflection define a line that intersects with the reflection lines of all other points at the point at infinity. We can sketch that like this:



Usually, when we add two points on an elliptic curve, we search for a third intersection of the straight line through the points with the curve and then reflect that point across the x-axis. What should happen, when we do this with a point and its reflection across the x-axis? It is indeed not quite clear, because we will not find any other intersection of line and curve. However, if we continue to travel along the line, we would at some point ("at infinity") reach the intersection of the line we are travelling with all other lines parallel to the y-axis. The idea now is to define addition of a point P with its reflection P' in such a way that $P + P'$ is precisely that point, which, in the context of elliptic curves, we call $\mathcal{O}$. So we add that point $\mathcal{O}$ to the curve and decide – deliberately – that this point is the additive identity. For any point P on the curve, it then holds that $P + \mathcal{O} = P$. The point P', for which $P + P' = \mathcal{O}$, *i.e.* the reflection of P across the x-axis, is in consequence the inverse of P.

Note that there is no deeper mathematics involved here that would directly lead to a formula that we could apply to "automatically" generate the result $P + P' = \mathcal{O}$. Instead, we have to consider this case as well as $P + \mathcal{O} = P$ explicitly in the addition formula.

But why do we reflect at all, when adding two points? That is because, otherwise, addition would be quite boring. Suppose we added without reflection. Then addition would go $P + Q = R'$ and the reverse additions $R' + Q = P$ and $R' + P = Q$ would just lead back to where we started. This would be true for any three points in such an additional constellation, because the three points are on the same straight line. If we go forward prolonging the line $\overline{PQ}$, we find R'. If we go backward prolonging the line $\overline{R'Q}$, we find P, or, if we draw the line $\overline{R'P}$, we find Q in the middle. Even if such a rule could ever lead to a group, there would be no generators of the group. A generator is a point that repeatedly added to itself creates the whole group. But leaving reflection out, the subsequent addition of a point P would give raise to a sequence like $P, 2P, P, 2P, P, \dots$, which, certainly, is not a group.

1.3 EC modulo a Prime

It was already indicated that the geometry exercises in the previous sections had the sole purpose of giving an intuition. EC Cryptography does not take place in the continuous universe. It does take place in modular arithmetic with integers and, hence, in a discrete world. This is a disruptive turning point, since we cannot plot a curve and search for a point in the Cartesian plane anymore. As we will see examining points of a curve modulo some number these points are not located on anything even close to the curves we saw in the previous sections. One could say that we adopt the algebra of elliptic curves, but drop the geometry.

Let us start with a data type. We define an elliptic curve as

```
data Curve = Curve {
  curA :: Natural,
  curB :: Natural,
  curM :: Natural}
  deriving (Show, Eq)
```

This type describes a curve in terms of its coefficient a and b and in terms of the modulus. When we consider only curves of the form

$$y^2 = x^3 + ax + b, \tag{1.20}$$

the definition given by the type is sufficient. There are other curves, though, for instance this one:

$$y^2 = x^3 + ax^2 + bx + c, \tag{1.21}$$

but we do not consider them in this humble introduction.

For the modulus, either a (huge) prime is used or a (huge) power of 2. Again, we do not consider powers of 2.

Now we define the notion of “point”:

```
data Point = O | P Natural Natural
  deriving (Eq)
```

and make it an instance of *Show* to get a more pleasant visualiation:

```
instance Show Point where
  show O = "O"
  show (P x y) = "(" ++ show x ++ "," ++ show y ++ ")"
```

Note that we explicitly define $\mathcal{O}$, the identity, to which we will have to refer explicitly in addition and other operations on points later.

We also define convenience getters for the point coordinates:

```
xco :: Point → Natural
xco O = error "O has no coordinates"
xco (P x _) = x

yco :: Point → Natural
yco O = error "O has no coordinates"
yco (P _ y) = y
```

To be sure that the points we create are modulo p, we define a convenient creator function:

```
point :: Curve → (Natural, Natural) → Point
point c (x, y) = P (x `mod` p) (y `mod` p)
  where p = curM c
```

Note that we use our *mod* function defined in section on modular arithmetic in the Prime chapter.

Now we would like to have a function that gives us the y-coordinate of the point with a given x-coordinate. In the continuous universe that would be quite easy. It is a bit complicated in modular arithmetic. We start with a function that gives us y^2:

```
curveY'2 :: Curve → Natural → Natural
curveY'2 c x = (x ↑ 3 + a * x + b) `mod` p
  where a = curA c
        b = curB c
        p = curM c
```

That is neat and simple. We just plug the given x-value into the right-hand side of the curve equation and get y^2 back. But, now, how to compute y? In the continuous universe, we would just call $\sqrt{y^2}$. But we are in modular arithmetic and y^2 is not necessarily a perfect square, but a quadratic residue, which may or may not be a perfect square. Here are as an example the residues of prime 17:

$$0, 1, 2, 4, 8, 9, 13, 15, 16.$$

Those are nine numbers, which was to be expected, since, for any prime modulus p, there are $\frac{p+1}{2}$ residues and $\frac{p-1}{2}$ nonresidues. Of these nine numbers, only five, namely 0, 1, 4, 9 and 16, are perfect squares. For those it is quite easy to compute the root. It is just the regular square root. For the others, however, it is quite hard. The problem is closely related to the Discrete Logarithm Problem (DLP), which is hard enough to provide the setting for most public key cryptographic schemes around today. Anyway, we have to live with it for the moment and implement a searching algorithm that is fine for small modulus, but infeasible in practice:

```
findRoot :: Natural → Natural → Natural
findRoot p q = go 0
  where go x | x > p       = error "not found!"
             | (x ↑ 2) ‘mod‘ p ≡ q = x
             | otherwise = go (x + 1)
```

Basically, we just go through all numbers from 0 to $p - 1$, until we find one that squared yields q, the residue in question. If we do not find such a number, we terminate with an error. If we map *findRoot* on the residues of 17, *map* (*findRoot* 17) (*residues* 17), we see:

$$0, 1, 6, 2, 5, 3, 8, 7, 4.$$

Some numbers are not surprising at all. 0 is of course the root of 0 and so is 1 of 1, 2 of 4, 3 of 9 and 4 of 16. But who had thought that 6 is the root of 2, 8 that of 13 or 7 that of 15?

With the help of this root finder, we can now implement a function that gives us y for x:

```
curveY :: Curve → Natural → Maybe Natural
curveY c x = let r = curveY′2 c x
             in if isSqrM r p then Just (findRoot p r)
                              else Nothing
  where p = curM c
```

We have inserted a safety belt in this function. Before we go into *findRoot*, which may cause an error when there is no root for the number in question, we check if it is a residue at all. If it is, we are confident to find a root and just return the result of *findRoot*. Otherwise, we return *Nothing*, meaning that the curve is not defined for this specific x. Here is the test for r being a residue using the Legendre symbol:

```
isSqrM :: Natural → Natural → Bool
isSqrM 0 _ = True
isSqrM n p = legendre n p ≡ 1
```

Based on these functions, we can define other useful tools. A function that verifies wether a given point is on the curve:

```
oncurve :: Curve → Point → Bool
oncurve _ O       = True
oncurve c (P x y) = case curveY c x of
                      Nothing → False
                      Just z  → y ≡ z ∨ y ≡ p − z
  where p = curM c
```

The function receives a curve and a point. It determines the y-coordinate for the x-coordinate of the point. If no y-coordinate is found, the point is certainly not on the curve. Otherwise, if the y we found is the same as the one of the point, then the point is on the curve. If the value we found is $-y$, that is to say, $p - y$, then the point is also on the curve, because $p - y$ is the additive inverse of y in the group and, if the point $(x, -y)$ is on the curve, then (x, y), the inverse of the point, is also in on the curve. Note that, when we say "a point is on the curve", we effectively say "the point is in the group". But be careful: we are here referring to two different groups. The group of integers modulo p and the group of points that "are on the curve".

The function *oncurve* is not very efficient, since it needs the root to calculate the result of *curveY*. A more efficient version is this one:

```
oncurve'2 :: Curve → Point → Bool
oncurve'2 _ O       = True
oncurve'2 c (P x y) = let z = curveY'2 c x
                      in  (y ↑ 2)     `mod` p ≡ z ∨
                          (p − y) ↑ 2 `mod` p ≡ z
  where p = curM c
```

As we are already dealing with inverses, here are two functions, one finding the inverse of a point and the other testing if a point is the inverse of the other:

```
pinverse :: Curve → Point → Point
pinverse _ O       = O
pinverse c (P x y) = point c (x, −y)

isInverse :: Curve → Point → Point → Bool
isInverse _ O O = True
isInverse c p q = q ≡ pinverse c p
```

Another useful tool would be one to find us a point on the curve. There are two ways to do it: deterministic and random. We start with the deterministic function that would basically go through all number from 0 to $p-1$ and stop, whenever there is a y for this x, such that (x, y) is on the curve:

```
findPoint :: Curve → Point
findPoint c = let (x, y′) = hf [(x, curveY′2 c x) | x ← [1 ..]]
              in  point c (x, findRoot p y′)
  where hf  = head ∘ filter (ism p ∘ snd)
        p   = curM c
        ism = flip isSqrM
```

The function generates tuples of the form (x, y^2) and filters those where y^2 is indeed a residue of p. The first of the resulting list is returned and laziness saves us from going through literally all possible x. This is a very useful tool to get started with a curve, but it is a bit boring, because it would always yield the same point. Randomness would make that more exciting giving us different points. Here is a function that yields a random point on a given curve:

```
randomPoint :: Curve → IO Point
randomPoint c = do
  x ← randomRIO (1, p − 1)
  let y′ = curveY′2 c x
  if isSqrM y′ p then return (point c (x, findRoot p y′))
                 else  randomPoint c
  where p = curM c
```

The code is straight forward. First we generate a random number x in the range $1 \dots p{-}1$. Then we determine y^2 and, if this is a residue, we return the point consisting of x and y. Otherwise, if it is not a residue, we start all over again.

Let us take a break here and look at some points in a real curve. We start by defining a curve for experiments:

```
c1 :: Curve
c1 = Curve 2 2 17
```

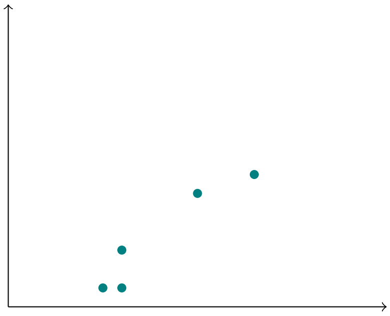
This corresponds to the curve

$$y^2 \equiv x^3 + 2x + 2 \pmod{17}.$$

We call *mapM* (\_ → *randomPoint c1*) [1 . . 5], generating five random points. We may see the points

$$(10, 6), (5, 1), (6, 3), (3, 1), (13, 7)$$

(or any other selection of points. It is a **random** list!) As expected, we see points with integer coordinates in the range $0 \ldots 16$. Let us look where those points are located in the Cartesian plane.



As already said: that does not look like an elliptic curve at all. It does not look completely random either. To have the complete picture, however, we need all points on that curve. How can we get them? Right! With a generator! Where do we get a generator? One way is trial and error. But for that we need the group operation. So let us get on with it. Here is addition:

```
add :: Curve → Point → Point → Point
add _ _ p O = p
add _ _ O p = p
add c p@(P x1 y1) q@(P x2 y2) | isInverse c p q = O
                              | otherwise       =
                                let xr = (l ↑ 2 − x1 − x2)     ‘mod‘ m
                                    yr = (l ∗ (xr − x1) + y1) ‘mod‘ m
                                in point c (xr, −yr)
  where a             = curA c
        m             = curM c
        l | x1 ≡ x2   =
           let t1     = (3 ∗ x1 ↑ 2 + a)   ‘mod‘ m
               t2     = inverse ((2 ∗ y1) ‘mod‘ m) m
           in  (t1 ∗ t2) ‘mod‘ m
          | otherwise =
           let t1     =           (y2 − y1) ‘mod‘ m
               t2     = inverse ((x2 − x1) ‘mod‘ m) m
           in  (t1 ∗ t2) ‘mod‘ m
```

We start with the base cases where one of the points is $\mathcal{O}$, the identity of the group of the curve. The result of addition in this case is just the other point. Then we handle two points none of which is the identity. If one is the inverse of the other, then the result is just $\mathcal{O}$. All these cases, as already mentioned in the previous section, must be explicitly handled in our implementation. There is no direct way that would produce the result. After all, this is a highly "engineered" group.

Now, we are finally in the “regular” case, where none of the points is the identity and the points are not the inverses of each other. In this case – we just apply the formula we have learnt before. However, it looks a bit different. This is because we are now in the discrete universe of modular arithmetic. The main difference is that, instead of dividing coordinates, we multiply them by the modular inverse of the denominator. We are here dealing with the group of integers modulo the prime we use for the curve.

It should be mentioned that to compute the slope of the line l, we distinguish the cases $p = q$ (point doubling) and $p \neq q$ by just comparing the x-coordinates ignoring the y-coordinates. We can do this, because we already have checked one point being the inverse of the other. Since the inverse of a point (x, y) is its reflection across the x-axis $(x, -y)$ and there, for sure, is no other point with that x-coordinate, it would be redundant to check the y-coordinate once again.

What do points look like, when we add them up? Let us take two points from the list above. What about the first two, $(10, 6)$ and $(5, 1)$? We add them calling
add c1 (*P* 10 6) (*P* 5 1) and get

$$(3, 1).$$

There is really nothing that would suggest any similarity to ordinary arithmetic addition.

How can we use addition to generate the whole group? Since we are dealing with an additive group (according to this strange definition of addition), we can pick a primitive element, a generator, and add it successivley to itself. But what is a primitive element of the group of our curve $c1$? Well, I happen to know that the order of that group is 19. Since 19 is a prime number and since we are talking about groups, Lagrange’s theorem applies, *i.e.* the order of subgroups must divide the order of the main group. Therefore, all members of the group are either member of a trivial subgroup (which contains only one element, namely the identity) or generators of the main group. Since the sole element in the trivial group must be the identity $\mathcal{O}$, all other members of the group must be generators. We, hence, can pick any point and generate the whole group from it. Here is a generator function:

```
gen :: Curve → Point → [Point]
gen c p = go p
  where go O = [O]
        go r = r : go (add c r p)
```

We call it like *gen c1* (*P* 10 6) and get

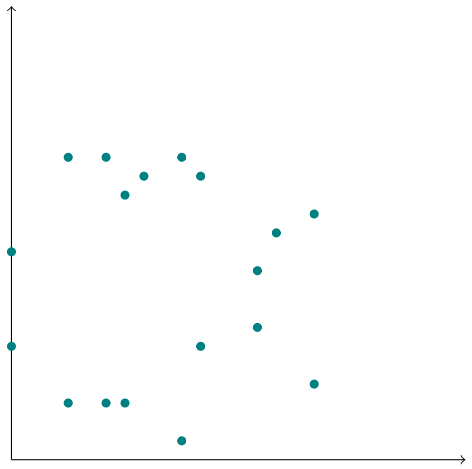
$$(10, 6), (16, 13), (7, 6), (0, 11), (3, 16), (5, 16), (6, 3),$$
$$(9, 16), (13, 7), (13, 10), (9, 1), (6, 14), (5, 1), (3, 1),$$
$$(0, 6), (7, 11), (16, 4), (10, 11), O,$$

which are 19 points and, hence, the entire group of the curve $c1$.

Note that the final point is the identity. This is exactly the same behaviour as we saw for multiplicative groups modulo a prime. For instance, 3 is a generator of the group modulo 7. We saw that $3^1 \equiv 3$, $3^2 \equiv 2$, $3^3 \equiv 6$, $3^4 \equiv 4$, $3^5 \equiv 5$ and $3^6 \equiv 1$ all $\pmod 7$.

The last but one point in the list is the inverse of the point we started with. In the integer case, there was nothing obvious that pointed to the fact that 5 is the inverse of 3 modulo 7. With the points above, however, it is immediately clear, since, as you can see, the penultimate point is $(10, 11)$. It has the same x-coordinate as $(10, 6)$ and the y-coordinate is $-y$ of the original point, because $17 - 6 = 11$. 11, hence, is -6 modulo 17.

Do we get a clearer picture when we put all these points on the Cartesian plane? Not really:

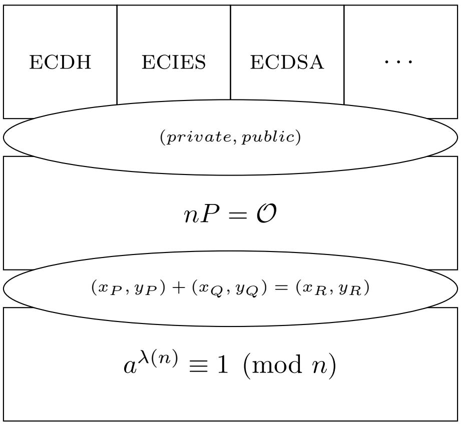


So, let us forget about geometry for a while. We are dealing with modular arithmetic related to a construction that we happen to call a curve. There is no more secret geometry behind it.

1.4 The EC Discrete Logarithm Problem

To summarise the results from the previous sections, we can describe EC Crypto as a system with three layers. The first layer consists in modular integer arithmetic and is used to do the math underlying point addition. Point addition itself belongs to the second layer, the additive group of points on the curve, which results from *using* point addition. Since all arithmetic modulo a prime is done within point addition, when we look at point addition itself, we do not “see” modular arithmetic anymore. We do not add points modulo something, we just add them using the addition formula.

The third layer consists of the cryptosystems built on top of the group of points. To actually build cryptosystems on top of elliptic curves, we need a secret that we can use as private key and an open parameter acting as public key. Perhaps the following schematic sketch helps:


ECDH
ECIES
ECDSA
⋯
$(private, public)$
$nP = \mathcal{O}$
$(x_P, y_P) + (x_Q, y_Q) = (x_R, y_R)$
$a^{\lambda(n)} \equiv 1 \pmod{n}$

Bottom up, we see first the arithmetic group modulo some prime, indicated by Carmichael's theorem. On top of it, we built point addition, which, in its turn, is the basis for the group of points on the curve, indicated by the fact that repeated addition (*i.e.* multiplication) of a point P yields the identity of that group $\mathcal{O}$. By means of this group, we build a key pair (*public, private*), which then enables us to build concrete cryptographic schemes, such as EC Diffie-Hellman (ECDH), EC Integrated Encryption Scheme (ECIES) and the EC Digital Signature Algorithm (ECDSA), which we will discuss in the next sections. As indicated by the right-most box in that layer, there are many more cryptographic schemes for elliptic curves. But we will not discuss all of them here.

When we look back at classic cryptography, we see that we typically used an invertible operation, namely *exponentiation.* In some cases, like in Diffie-Hellman, the private key was a combination of public keys of the form $a^{xy} = (a^x)^y = (a^y)^x$, in others, like RSA, public and private keys were inverses of each other, such that $a^{ed} = a^1 = a$. In either case, the security of the private key was based on the hardness of the discrete logarithm problem (DLP). The DLP aims to solve equations of the form

$$a^x = b$$

for x, *i.e.* it asks for the number x, to which we have to raise a to get to b. The classic crypto schemes are based on a multiplicative group. The DLP, hence, asks for the number of repetitions of successive multiplications of a to get to b. In EC, however, we have an additive group or, at least, we use "additive terminology" describing the group operation. When we ask for the number of repetitions of that operation on a to get b, we would hence ask for multiplication, not for exponentiation. There is some potential for confusion in this terminology switch from multiplicative to additive groups. The problem on which the security in EC relies is unfortunately also called DLP, discrete logarithm problem. It would be much more precise in this case to refer to the *discrete quotient problem*, since we talk about multiplication, not exponentiation. However, the terminology is like that. So, we stick to it and define the DLP for EC crypto.

The information of EC crypto systems that is publicly known consist of parameters describing the curve, the coefficients, the modulus, a starting point P and perhaps some other details specifying the exact curve. The public key is typically a point Q and the secret is a natural number n, such that

$$nP = Q. \tag{1.22}$$

The DLP, hence, consists in finding a *factor* n that determines how often we have to add P to itself to get to Q. This may sound weird, but, in fact, it is a hard problem equivalent in computational complexity to finding the discrete logarithm in classic cryptography.

Consider the curve we already used above with starting point $P = (5, 1)$. If you consider the public key to be $Q = (9, 16)$, can you tell, without looking at the whole group above, what n must be, such that $n \times (5, 1) = (9, 16)$? The point is that no algorithm is known that would do that in acceptable time for large groups. For this toy group, we obviously can try. We would see that:

$$\begin{aligned}
(5,1) + (5,1) &= (6,3) = 2P,\\
(6,3) + (5,1) &= (10,6) = 3P,\\
(10,6) + (5,1) &= (3,1) = 4P,\\
(3,1) + (5,1) &= (9,16) = 5P.
\end{aligned}$$

We, thus, find our point after four additions, which corresponds to multiplying the point by 5. With a group that contains 2^{256} elements, this approach would not be feasible.

But how can it then be feasible to compute nP for large n in the first place? We would need n steps to produce that result and n can be a really large number. Obviously, we need a way to perform multiplication in significantly less than n steps, in $\log n$ steps, for instance.

There is indeed a way to do this. The algorithm is quite similar to the algorithm we used to raise a number to a huge power, which was *multiply-and-square*. Here we use a variant of that algorithm called *double-and-add*, since we are dealing with an additive group. The algorithm may be implemented like this:

```
mul :: Curve → Natural → Point → Point
mul _ 0 _  = O
mul _ _ O  = O
mul c n p  = foldl da p (tail $ toBinary n)
  where da q 0 = add c q q
        da q 1 = add c p (add c q q)
```

The first base case states that 0 times anything is the identity. Then, the identity multiplied by any number is again the identity. The identity, indeed, never changes with multiplication. Just as zero would never change by multiplication in the realm of numbers. Any point multiplied by zero, on the other hand, is the identity of the additive group and that, again, is zero.

In all other cases, we convert n into a list of binary digits of which we process all but the head. For each digit, we double the intermediate result q, that is we compute *add c q q*. If the current digit is 0, we are done with this digit and continue with the next one. Otherwise, if it is 1, we additionally add p. When there are no more digits left, we have a result.

It is noteworthy that this implementation of multiplication does not use the means of the arithmetic group modulo a prime that underlies point addition. It is build on top of addition using only terms related to the group of points on the curve, the second layer in the sketch above.

Let us look at an example. Say, we want to compute $19P$. The binary representation of 19 is $[1, 0, 0, 1, 1]$. We, hence, would compute $P + P$ for the first digit 0 (which is the head of the tail of our number). This is $2P$. With this result we go into the next round. The next digit is 0 again and we get $4P$, which is the input for the next iteration. The next digit is 1, so we double and add. We, hence, compute $4P + 4P + P = 9P$. This is now the input to the final digit. Since it is 1 again, we again double and add and we have $9P + 9P + P = 19P$.

It works perfectly. But why does it work? Consider the representation of a number in terms of powers of 2 multiplied by a number k:

$$k(a_r 2^r + a_{r-1} 2^{r-1} + \cdots + a_0 2^0),$$

where, for $i \in \{0 \dots r\}$, $a_i \in \{0, 1\}$. Multiplying this out, we get

$$a_r 2^r k + a_{r-1} 2^{r-1} k + \cdots + a_0 2^0 k.$$

Obviously, from step to step, that is from plus sign to plus sign, right to left, k doubles. Ignoring the coefficients a_i for a moment, that would look for the concrete number $10011_2 = 19_{10}$ like

$$16k + 8k + 4k + 2k + k.$$

Doubling would in this case generate the number $16k$, which would indeed be the correct result if the binary number were 10000_2. Now, we eliminate all terms with coefficient $a_i = 0$, which are $8k$ and $4k$. We are left with

$$16k + 2k + k.$$

The value we add to $16k$ corresponds exactly to the value of k we would add with *mul*. For the example, we would add one k processing the last but one digit. This k is now part of the intermediate result, $2q + k$, that goes into the processing of the last digit. Processing the last digit, we double the previous result, obtaining $4q + 2k$ and, since the last digit is also 1, we add k again. We, hence, get three "extra" ks, which we add to the overall doubling result 16 and get $16 + 2 + 1 = 16 + 3 = 19$.

1.5 EC Diffie-Hellman

1.6 EC Integrated Encryption Scheme

1.7 EC Digital Signature Algorithm

1.8 Cryptoanalysis

1.9 Mr. Frobenius

1.10 Mr. Schoof

1.11 Mr. Elkies and Mr. Aktin

1.12 EC in Practice