Operational notes

Document updated on **March 26, 2022**.

The following colors are **not** part of the final product, but serve as highlights in the editing/review process:

- text that needs attention from the Subject Matter Experts: Mirco, Anna,& Jan
- terms that have not yet been defined in the book
- things that need to be checked only at the very final typesetting stage
  (and it doesn't make sense to do them before)
- text that needs advice from the communications/marketing team: Aaron & Shane
- text that needs to be completed or otherwise edited (by Sylvia)

NB: This PDF only includes the Elliptic Curves chapter

Todo list

MoonMath manual

TechnoBob and the Least Scruples crew

March 26, 2022

Contents

 CONTENTS

Chapter 5

Elliptic Curves

TODO: Elliptic Curve asymmetric cryptography examples. Private key, generator, public key.

Generally speaking, elliptic curves are "curves" defined in geometric planes like the Euclidean or the projective plane over some given field. One of the key features of elliptic curves over finite fields from the point of view of cryptography is that their set of points has a group law such that the resulting group is finite and cyclic, and it is believed that the discrete logarithm problem on these groups is hard.

A special class of elliptic curves are so-called **pairing-friendly curves**, which have a notation of a group pairing as defined in XXX. This pairing has cryptographically advantageous properties. Those curve are useful in the development of SNARKs, since they allow to compute so-called R1CS-satisfiability "in the exponent" MIRCO: (THIS HAS TO BE REWRITTEN WITH WAY MORE DETAIL)

add reference

In this chapter, we introduce epileptic curves as they are used in pairing-based approaches to the construction of SNARKs. The elliptic curves we consider are all defined over prime fields or prime field extensions and the reader should be familiar with the contend of the previous section on those fields.

In its most generality elliptic curves are defined as a smooth projective curve of genus 1 defined over some field $\mathbb{F}$ with a distinguished $\mathbb{F}$-rational point, but this definition is not very useful for the introductory character of this book. We will therefore look at 3 more practical definitions in the following sections, by introducing Weierstraß, Montgomery and Edwards curves. All of them are widely used in cryptography, and understanding them is crucial to being able to follow the rest of this book.

maybe remove this sentence?

5.1 Elliptic Curve Arithmetics

5.1.1 Short Weierstraß Curves

In this section, we introduce **short Weierstraß** curves, which are the most general types of curves over finite fields of characteristic greater than 3.

We start with their representation in affine space. This representation has the advantage that affine points correspond to pairs of numbers, which makes it more accessible for beginners. However, it has the disadvantage that a special "point at infinity", that is not a point on the curve, is necessary to describe the group structure. We introduce the elliptic curve group law and describe elliptic curve scalar multiplication, which is an instantiation of the exponential map from general cyclic groups.

affine space

Then we look at the projective representation of short Weierstraß curves. This has the advantage that no special symbol is necessary to represent the point at infinity but comes with the drawback that projective points are classes of numbers, which might be a bit unusual for a beginner.

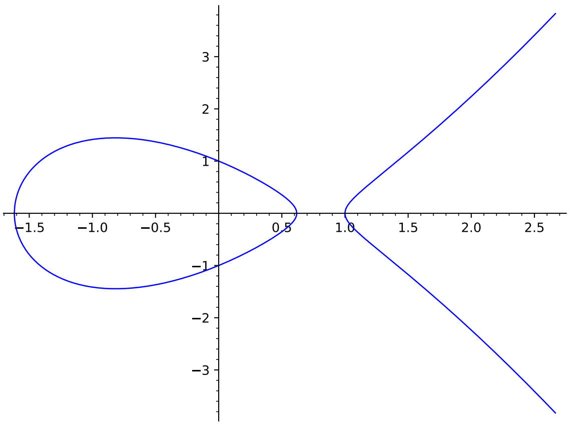
We finish this section with an explicit equivalence that transforms affine representations into projective ones and vice versa.

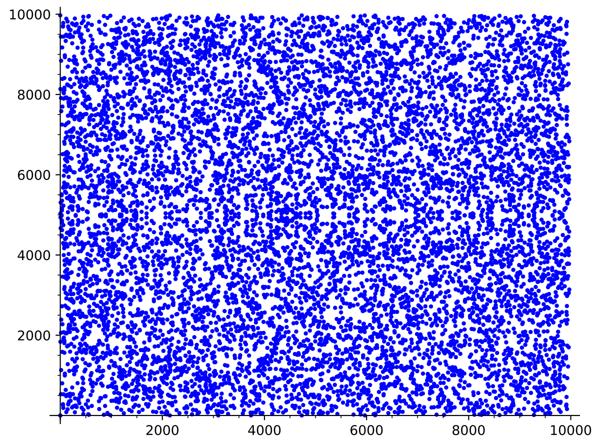
Affine short Weierstraß form Probably the least abstract and most straight-forward way to introduce elliptic curves for non-mathematicians and beginners is the so-called affine representation of a short Weierstraß curve. To see what this is, let $\mathbb{F}$ be a finite field of order q and $a,b \in \mathbb{F}$ two field elements such that $4a^3 + 27b^2 \bmod q \neq 0$. Then a **short Weierstraß elliptic curve** $E(\mathbb{F})$ over $\mathbb{F}$ in its affine representation is the set of all pairs of field elements $(x,y) \in \mathbb{F}\times\mathbb{F}$ that satisfy the short Weierstraß cubic equation $y^2 = x^3 + a\cdot x + b$, together with a distinguished symbol $\mathcal{O}$, called the **point at infinity**:

$$E(\mathbb{F}) = \{(x,y) \in \mathbb{F}\times\mathbb{F} \mid y^2 = x^3 + a\cdot x + b\} \bigcup \{\mathcal{O}\} \tag{5.1}$$

Notation and Symbols 7. In the literature, the set $E(\mathbb{F})$, which includes the symbol $\mathcal{O}$, is often called the set of **rational points** of the elliptic curve, in which case the curve itself is usually written as $E/\mathbb{F}$. However, in what follows, we will frequently identify an elliptic curve with its set of rational points and therefore use the notation $E(\mathbb{F})$ instead. This is possible in our case, since we only the group structure of the curve in consideration is relevant for us.

The term "curve" is used here because, in the ordinary 2 dimensional plane $\mathbb{R}^2$, the set of all points (x,y) that satisfy $y^2 = x^3 + a\cdot x + b$ looks like a curve. We should note however that visualizing elliptic curves over finite fields as "curves" has its limitations, and we will therefore not stress the geometric picture too much, but focus on the computational properties instead. To understand the visual difference, consider the following two elliptic curves:





Both elliptic curves are defined by the same short Weierstraß equation $y^2 = x^3 - 2x + 1$, but the first curve is defined in the real affine plane $\mathbb{R}^2$, that is, the pair (x,y) contains real numbers, while the second one is defined in the affine plane $\mathbb{F}^2_{9973}$, which means that both x and y are from the prime field $\mathbb{F}_{9973}$. Every blue dot represents a pair (x,y), that is a solution to $y^2 = x^3 - 2x + 1$. As we can see, the second curve hardly looks like a geometric structure one would naturally call a curve. This shows that our geometric intuitions from $\mathbb{R}^2$ are obfuscated in curves over finite fields.

The identity $6\cdot(4a^3 + 27b^2) \bmod q \neq 0$ ensures that the curve is non-singular, which basically means that the curve has no cusps or self-intersections.

cusps

self-intersections

Throughout this book, the reader is advised to do as many computations in a pen-and-paper fashion as possible, as this is helps getting a deeper understanding of the details. However, when dealing with elliptic curves, computations can quickly become cumbersome and tedious, and one might get lost in the details. Fortunately, Sage is very helpful in dealing with elliptic curves. This book to introduces the reader to the great elliptic curve capabilities of Sage. The following snipped shows a way to define elliptic curves and work with them in Sage:

```
sage: F5 = GF(5) # define the base field
sage: a = F5(2) # parameter a
sage: b = F5(4) # parameter b
sage: # check non-sigularity
sage: F5(6)*(F5(4)*a^3+F5(27)*b^2) != F5(0)
True
sage: # short Weierstrass curve
sage: E = EllipticCurve(F5,[a,b]) # y^2 == x^3 + ax +b
sage: P = E(0,2) # 2^2 == 0^3 + 2*0 + 4
sage: P.xy() # affine coordinates
(0, 2)
sage: INF = E(0) # point at infinity
sage: try:  # point at infinity has no affine coordinates
....:     INF.xy()
....: except ZeroDivisionError:
....:     pass
sage: P = E.plot() # create a plotted version
```

The following three examples give a more practical understanding of what an elliptic curve is and how we can compute it. The reader is advised to read them carefully, and ideally, to also carry out the computation themselves. We will repeatedly build on these examples in this chapter, and use the second example throughout this book.

Example 65. To provide the reader with an example of a small elliptic curve where all computation can be done with pen and paper, consider the prime field $\mathbb{F}_5$ from example 59 (page 60). quite familiar to readers who had worked through the examples and exercises in the previous chapter.

check reference

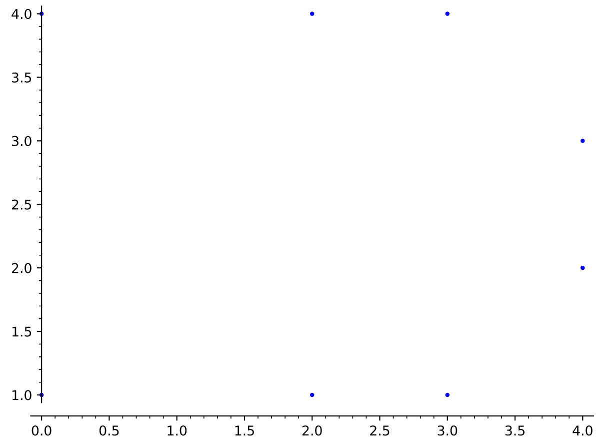
To define an elliptic curve over $\mathbb{F}_5$, we have to choose to numbers a and b from that field. Assuming we choose $a = 1$ and $b = 1$ then $4a^3 + 27b^2 \equiv 1 \pmod 5$ from which follows that the corresponding elliptic curve $E_1(\mathbb{F}_5)$ is given by the set of all pairs (x,y) from $\mathbb{F}_5$ that satisfy the equation $y^2 = x^3 + x + 1$, together with the special symbol $\mathcal{O}$, which represents the "point at infinity".

To get a better understand of that curve, observer that if we choose arbitrarily the pair $(x,y) = (1,1)$, we see that $1^2 \neq 1^3 + 1 + 1$ and hence $(1,1)$ is not an element of the curve $E_1(\mathbb{F}_5)$. On the other hand choosing for example $(x,y) = (2,1)$ gives $1^2 = 2^3 + 2 + 1$ and hence the pair $(2,1)$ is an element of $E_1(\mathbb{F}_5)$ (Remember that all computations are done in modulo 5 arithmetics).

Now since the set $\mathbb{F}_5 \times \mathbb{F}_5$ of all pairs (x,y) from $\mathbb{F}_5$ contains only $5 \cdot 5 = 25$ pairs, we can compute the curve, by just inserting every possible pair (x,y) into the short Weierstraß equation $y^2 = x^3 + x + 1$. If the equation holds, the pair is a curve point, if not that means that the point is not on the curve. Combining the result of this computation with the point at infinity gives the curve as follows:

$$E_1(\mathbb{F}_5) = \{\mathcal{O}, (0,1), (2,1), (3,1), (4,2), (4,3), (0,4), (2,4), (3,4)\}$$

This means that our elliptic curve is a set of 9 elements, 8 of which are pairs of numbers and one special symbol $\mathcal{O}$. Visualizing $E1$ gives the following plot:



In the development of SNARKs, it is sometimes necessary to do elliptic curve cryptography “in a circuit", which basically means that the elliptic curves need to be implemented in a certain SNARK-friendly way. We will look at what this means in chapter 7. To be able to do this efficiently, it is desirable to have curves with special properties. The following example is a pen-and-paper version of such a curve, called **Baby-jubjub**, which resembles cryptographically secure curves extensively used in real-world SNARKs. The interested reader is advised to study this example carefully, as we will use it and build on it in various places throughout the book. I feel like a lot of people won’t get the Lewis Carroll reference unless we make it more explicit

check reference

jubjub

Example 66 (Pen-JubJub). Consider the prime field $\mathbb{F}_{13}$ from exercise 4.3 (page 62. If we choose $a = 8$ and $b = 8$, then $4a^3 + 27b^2 \equiv 6 \pmod{13}$ and the corresponding elliptic curve is given by all pairs (x,y) from $\mathbb{F}_{13}$ such that $y^2 = x^3 + 8x + 8$ holds. We call this curve the **Pen-JubJub** curve, or *PJJ_13* for short.

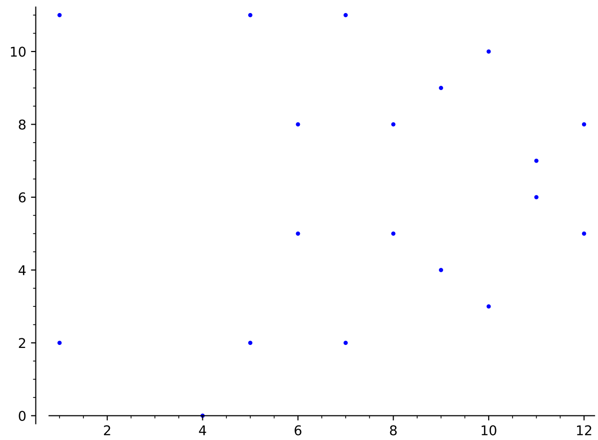
check reference

Now, since the set $\mathbb{F}_{13} \times \mathbb{F}_{13}$ of all pairs (x,y) from $\mathbb{F}_{13}$ contains only $13 \cdot 13 = 169$ pairs, we can compute the curve by just inserting every possible pair (x,y) into the short Weierstraß equation $y^2 = x^3 + 8x + 8$. We get the following result:

$$\begin{aligned} PJJ\_13 = \{&\mathcal{O},(1,2),(1,11),(4,0),(5,2),(5,11),(6,5),(6,8),(7,2),(7,11),(8,5),(8,8),\\ &(9,4),(9,9),(10,3),(10,10),(11,6),(11,7),(12,5),(12,8)\} \end{aligned} \tag{5.2}$$

As we can see, the curve consists of 20 points; 19 points from the affine plane and the point at infinity. To get a visual impression of the *PJJ_13* curve, we might plot all of its points (except the point at infinity) in the $\mathbb{F}_{13} \times \mathbb{F}_{13}$ affine plane. We get the following plot:

affine plane



As we will see in what follows, this curve is rather special, as it is possible to represent it in two alternative forms, called the **Montgomery** and the **twisted Edwards form** (See XXX and XXX).

add reference

add reference

Now that we have seen two pen-and-paper friendly elliptic curves, let us look at a curve, that is used in actual cryptography. Cryptographically secure elliptic curves are not **qualitatively** different from the curves we looked at so far, but the prime number modulus of their prime field is much larger. Typical examples use prime numbers that have binary representations in the magnitude of more than double the size of the desired security level. If, for example, a security of 128 bits is desired, a prime modulus of binary size ≥ 256 is chosen. The following example provides such a curve.

Example 67 (Bitcoin's Secp256k1 curve). To give an example of a real-world, cryptographically secure curve, let us look at curve Secp256k1, which is famous for being used in the public key cryptography of Bitcoin. The prime field $\mathbb{F}_p$ of Secp256k1 is defined by the following prime number:

$$p = 115792089237316195423570985008687907853269984665640564039457584007908834671663$$

The binary representation of this number needs 256 bits, which implies that the prime field $\mathbb{F}_p$ contains approximately 2^{256} many elements, which is considered quite large. To get a better impression of how large the base field is, consider that the number 2^{256} is approximately in the same order of magnitude as the estimated number of atoms in the observable universe.

The curve Secp256k1 is defined by the parameters $a, b \in \mathbb{F}_p$ with $a = 0$ and $b = 7$. Since $4 \cdot 0^3 + 27 \cdot 7^2 \bmod p = 1323$, those parameters indeed define an elliptic curve given as follows:

$$Secp256k1 = \{(x,y) \in \mathbb{F}_p \times \mathbb{F}_p \mid y^2 = x^3 + 7\}$$

Clearly, the Secp256k1 curve is too large to do computations by hand, since it can be shown that the number of its elements is a prime number r that also has a binary representation of 256 bits:

$$r = 115792089237316195423570985008687907852837564279074904382605163141518161494337$$

Cryptographically secure elliptic curves are therefore not useful in pen-and-paper computations. Fortunately, Sage handles large curves efficiently:

```
sage: p = 115792089237316195423570985008687907853269984665640564039457584007908834671663564039457584007908834671663
```

```
sage: # Hexadecimal representation
sage: p.str(16)
fffffffffffffffffffffffffffffffffffffffffffffffffffffffeffffc
    2f
sage: p.is_prime()
True
sage: p.nbits()
256
sage: Fp = GF(p)
sage: Secp256k1 = EllipticCurve(Fp,[0,7])
sage: r = Secp256k1.order() # number of elements
sage: r.str(16)
fffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd03641
    41
sage: r.is_prime()
True
sage: r.nbits()
256
```

Exercise 35. Look up the definition of curve BLS12-381, implement it in Sage and compute its order.

Affine compressed representation As we have seen in example 67, cryptographically secure elliptic curves are defined over large prime fields, where elements of those fields typically need more than 255 bits of storage on a computer. Since elliptic curve points consist of pairs of those field elements, they need double that amount of storage. check reference

However, we can reduce the amount of space needed to represent a curve point by using a technique called **point compression**. Note that, up to a sign, the y coordinate of a curve point can be computed from the x coordinate by simply inserting x into the Weierstraß equation and then computing the roots of the result. This gives two results, and it means that we can represent a curve point in **compressed form** by simply storing the x coordinate together with a single sign bit only, the latter of which deterministically decides which of the two roots to choose. One convention could be to always choose the root closer to 0 when the sign bit is 0, and the root closer to the order of $\mathbb{F}$ when the sign bit is 1. In case the y coordinate is zero, both sign bits give the same result. sign

more explanation of what the sign is

Example 68 (Pen-jubjub). To understand the concept of compressed curve points a bit better, consider the *PJJ_13* curve from example 66 again. Since this curve is defined over the prime field $\mathbb{F}_{13}$, and numbers between 0 and 13 need approximately 4 bits to be represented, each *PJJ_13* point on this curve needs 8 bits of storage in uncompressed form. The following set represents the uncompressed form of the points on this curve: check reference

$$
\begin{aligned}
PJJ\_13 = \{&\mathcal{O},(1,2),(1,11),(4,0),(5,2),(5,11),(6,5),(6,8),(7,2),(7,11),\\
&(8,5),(8,8),(9,4),(9,9),(10,3),(10,10),(11,6),(11,7),(12,5),(12,8)\}
\end{aligned}
$$

Using the technique of point compression, we can reduce the bits needed to represent the points on this curve to 5 per point. To achieve this, we can replace the y coordinate in each (x,y) pair by a sign bit indicating whether or not y is closer to 0 or to 13. As a result y values in the range $[0,\dots,6]$ will have the sign bit 0, while y-values in the range $[7,\dots,12]$ will have the sign bit 1.

Applying this to the points in *PJJ_13* gives the compressed representation as follows:

$$PJJ\_13 = \{\mathcal{O}, (1,0), (1,1), (4,0), (5,0), (5,1), (6,0), (6,1), (7,0), (7,1), (8,0), (8,1), (9,0), (9,1), (10,0), (10,1), (11,0), (11,1), (12,0), (12,1)\}$$

Note that the numbers $7,\ldots,12$ are the negatives (additive inverses) of the numbers $1,\ldots,6$ in modular 13 arithmetics and that $-0 = 0$. Calling the compression bit a "sign bit" therefore makes sense.

S: I don't follow this at all

To recover the uncompressed counterpart of, say, the compressed point $(5,1)$, we insert the x coordinate 5 into the Weierstraß equation and get $y^2 = 5^3 + 8\cdot 5 + 8 = 4$. As expected, 4 is a quadratic residue in $\mathbb{F}_{13}$ with roots $\sqrt{4} = \{2, 11\}$. Since the sign bit of the point is 1, we have to choose the root closer to the modulus 13, which is 11. The uncompressed point is therefore $(5,11)$.

Looking at the previous examples, the compression rate does not look very impressive. However, looking at the real-life example of the Secp256k1 curve shows that compression is has significant practical advantages.

Example 69. Consider the Secp256k1 curve from example 67 again. The following code invokes Sage to generate a random affine curve point, then applies our compression method to it:

check reference

```
sage: P = Secp256k1.random_point().xy()
sage: P
(573274555909292870027549532819570308193155586251244694583622856308870288524
36, 24242609999426606897142811967939071817174686615886596221090801834998454950146)
sage: # uncompressed affine point size
sage: ZZ(P[0]).nbits()+ZZ(P[1]).nbits()
509
sage: # compute the compression
sage: if P[1] > Fp(-1)/Fp(2):
....:     PARITY = 1
....: else:
....:     PARITY = 0
sage: PCOMPRESSED = [P[0],PARITY]
sage: PCOMPRESSED
[57327455590929287002754953281957030819315558625124469458362285630887028852436, 0]
sage: # compressed affine point size
sage: ZZ(PCOMPRESSED[0]).nbits()+ZZ(PCOMPRESSED[1]).nbits()
255
```

add explanation of how this shows what we claim

Affine group law One of the key properties of an elliptic curve is that it is possible to define a group law on the set of its rational points such that the point at infinity serves as the neutral element and inverses are reflections on the x-axis.

The origin of this law can be understood in a geometric picture and is known as the **chord-and-tangent rule**. In the affine representation of a short Weierstraß curve, the rule can be described in the following way:

Definition 5.1.1.1. **Chord-and-tangent rule**

- (Point at infinity) We define the point at infinity $\mathcal{O}$ as the neutral element of addition, that is, we define $P+\mathcal{O}=P$ for all points $P\in E(\mathbb{F})$. should this def. be moved even earlier?
- (Point addition) Let $P,Q\in E(\mathbb{F})\backslash\{\mathcal{O}\}$ with $P\neq Q$ be two distinct points on an elliptic curve, neither of them the point at infinity. The sum of P and Q is defined as follows:
  Consider the line l which intersects the curve in P and Q. If l intersects the elliptic curve at a third point R', define the sum $R=P\oplus Q$ of P and Q as the reflection of R' at the x-axis. If the line l does not intersect the curve at a third point, define the sum to be the point at infinity $\mathcal{O}$. It can be shown that no such chord line will intersect the curve in more than three points, so addition is not ambiguous. chord line
- (Point doubling) Let $P\in E(\mathbb{F})\backslash\{\mathcal{O}\}$ be a point on an elliptic curve, that is not the point at infinity. The sum of P with itself (the doubling of P) is defined as follows:
  Consider the line which is tangential to the elliptic curve at P. If this line intersects the elliptic curve at a second point R', the sum $2P=P+P$ is the reflection of R' at the x-axis. If it does not intersect the curve at a third, point define the sum to be the point at infinity $\mathcal{O}$. It can be shown that no such tangent line will intersect the curve in more than two points, so addition is not ambiguous. tangential tangent line

It can be shown that the points of an elliptic curve form a commutative group with respect to the tangent-and-chord rule such that $\mathcal{O}$ acts the neutral element, and the inverse of any element $P\in E(\mathbb{F})$ is the reflection of P on the x-axis.

To translate the geometric description into algebraic equations, first observe that, for any two given curve points $(x_1,y_1),(x_2,y_2)\in E(\mathbb{F})$, it can be shown that the identity $x_1=x_2$ implies $y_2=\pm y_1$, which shows that the following rules are a complete description of the affine addition law.

Definition 5.1.1.2. **Chord-and-tangent rule: algebraic equations**

- (Neutral element) The point at infinity $\mathcal{O}$ is the neutral element.
- (Additive inverse) The additive inverse of $\mathcal{O}$ is $\mathcal{O}$. For any other curve point $(x,y)\in E(\mathbb{F})\backslash\{\mathcal{O}\}$, the additive inverse is given by $(x,-y)$.
- (Addition rule) For any two curve points $P,Q\in E(\mathbb{F})$, addition is defined by one of the following three cases:
  1. (Adding the neutral element) If $Q=\mathcal{O}$, then the sum is defined as $P\oplus Q=P$.
  2. (Adding inverse elements) If $P=(x,y)$ and $Q=(x,-y)$, the sum is defined as $P\oplus Q=\mathcal{O}$.
  3. (Adding non-self-inverse equal points) If $P=(x,y)$ and $Q=(x,y)$ with $y\neq 0$, the sum $2P=(x',y')$ is defined as follows:We only referred to P in the definition of point doubling above so Q seems a bit confusing here even though it's defined as equal to P remove Q?
  $$x'=\left(\frac{3x^2+a}{2y}\right)^2-2x \quad , \quad y'=\left(\frac{3x^2+a}{2y}\right)^2(x-x')-y$$
  4. (Adding non-inverse different points) If $P=(x_1,y_1)$ and $Q=(x_2,y_2)$ such that $x_1\neq x_2$, the sum $R=P+Q$ with $R=(x_3,y_3)$ is defined as follows:
  $$x_3=\left(\frac{y_2-y_1}{x_2-x_1}\right)^2-x_1-x_2 \quad , \quad y_3=\left(\frac{y_2-y_1}{x_2-x_1}\right)(x_1-x_3)-y_1$$

Note that short Weierstraß curve points P with $P = (x, 0)$ are inverses of themselves, which implies $2P = \mathcal{O}$ in this case.

Notation and Symbols 8. Let $\mathbb{F}$ be a field and $E(\mathbb{F})$ be an elliptic curve over $\mathbb{F}$. We write $\oplus$ for the group law on $E(\mathbb{F})$ and $(E(\mathbb{F}), \oplus)$ for the group of rational points.

As we can see, it is very efficient to compute inverses on elliptic curves. However, computing the addition of elliptic curve points in the affine representation needs to consider many cases and involves extensive finite field divisions. As we will see in the next paragraph, this can be simplified in projective coordinates. where?

To get some practical impression of how the group law on an elliptic curve is computed, let's look at some actual cases:

Example 70. Consider the elliptic curve $E_1(\mathbb{F}_5)$ from example 65 again. As we have seen, the set of rational points contains 9 elements: check reference

$$E_1(\mathbb{F}_5) = \{\mathcal{O}, (0,1), (2,1), (3,1), (4,2), (4,3), (0,4), (2,4), (3,4)\}$$

We know that this set defines a group, so we can add any two elements from $E_1(\mathbb{F}_5)$ to get a third element.

To give an example, consider the elements $(0,1)$ and $(4,2)$. Neither of these elements is the neutral element $\mathcal{O}$, and since, the x coordinate of $(0,1)$ is different from the x coordinate of $(4,2)$, we know that we have to use the chord rule, that is, rule number 4 from definition 5.1.1.2 to compute the sum $(0,1) \oplus (4,2)$: check reference

$$\begin{aligned}
x_3 &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 && \text{\# insert points} \\
&= \left(\frac{2-1}{4-0}\right)^2 - 0 - 4 && \text{\# simplify in } \mathbb{F}_5 \\
&= \left(\frac{1}{4}\right)^2 + 1 = 4^2 + 1 = 1 + 1 = 2
\end{aligned}$$

$$\begin{aligned}
y_3 &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1 && \text{\# insert points} \\
&= \left(\frac{2-1}{4-1}\right)(0-2) - 1 && \text{\# simplify in } \mathbb{F}_5 \\
&= \left(\frac{1}{4}\right) \cdot 3 + 4 = 4 \cdot 3 + 4 = 2 + 4 = 1
\end{aligned}$$

So, in our elliptic curve $E_1(\mathbb{F}_5)$ we get $(0,1) \oplus (4,2) = (2,1)$, and, indeed, the pair $(2,1)$ is an element of $E_1(\mathbb{F}_5)$ as expected. On the other hand, $(0,1) \oplus (0,4) = \mathcal{O}$, since both points have equal x coordinates and inverse y coordinates, rendering them inverses of each other. Adding the point $(4,2)$ to itself, we have to use the tangent rule, that is, rule 3 from definition 5.1.1.2: check reference

$$
\begin{aligned}
x' &= \left(\frac{3x^2+a}{2y}\right)^2 - 2x && \text{\# insert points}\\
&= \left(\frac{3\cdot 4^2+1}{2\cdot 2}\right)^2 - 2\cdot 4 && \text{\# simplify in } \mathbb{F}_5\\
&= \left(\frac{3\cdot 1+1}{4}\right)^2 + 3\cdot 4 = \left(\frac{4}{4}\right)^2 + 2 = 1+2 = 3
\end{aligned}
$$

$$
\begin{aligned}
y' &= \left(\frac{3x^2+a}{2y}\right)^2 (x-x') - y && \text{\# insert points}\\
&= \left(\frac{3\cdot 4^2+1}{2\cdot 2}\right)^2 (4-3) - 2 && \text{\# simplify in } \mathbb{F}_5\\
&= 1\cdot 1 + 3 = 4
\end{aligned}
$$

So, in our elliptic curve $E_1(\mathbb{F}_5)$, we get the doubling of $(4,2)$, that is, $(4,2)\oplus(4,2) = (3,4)$, and, indeed the pair $(3,4)$ is an element of $E_1(\mathbb{F}_5)$ as expected. The group $E_1(\mathbb{F}_5)$ has no self-inverse points other than the neutral element $\mathcal{O}$, since no point has 0 as its y coordinate. We can invoke Sage to double-check the computations.

```
sage: F5 = GF(5)
sage: E1 = EllipticCurve(F5,[1,1])
sage: INF = E1(0) # point at infinity
sage: P1 = E1(0,1)
sage: P2 = E1(4,2)
sage: P3 = E1(0,4)
sage: R1 = E1(2,1)
sage: R2 = E1(3,4)
sage: R1 == P1+P2
True
sage: INF == P1+P3
True
sage: R2 == P2+P2
True
sage: R2 == 2*P2
True
sage: P3 == P3 + INF
True
```

Example 71 (Pen-jubjub). Consider the *PJJ_13*-curve from example 66 again and recall that its group of rational points is given as follows: check reference

$$
\begin{aligned}
PJJ\_13 = \{&\mathcal{O},(1,2),(1,11),(4,0),(5,2),(5,11),(6,5),(6,8),(7,2),(7,11),\\
&(8,5),(8,8),(9,4),(9,9),(10,3),(10,10),(11,6),(11,7),(12,5),(12,8)\}
\end{aligned}
$$

In contrast to the group from the previous example, this group contains a self-inverse point, which is different from the neutral element, defined by $(4,0)$. To see what this means, observe that we cannot add $(4,0)$ to itself using the tangent rule 3 from definition 5.1.1.2, as the y coordinate is zero. Instead, we have to use rule 2, since $0 = -0$. We therefore get $(4,0)\oplus$ check reference

$(4,0) = \mathcal{O}$ in *PJJ_13*. The point $(4,0)$ is therefore the inverse of itself, as adding it to itself results in the neutral element.

```
sage: F13 = GF(13)
sage: MJJ = EllipticCurve(F13,[8,8])
sage: P = MJJ(4,0)
sage: INF = MJJ(0) # Point at infinity
sage: INF == P+P
True
sage: INF == 2*P
True
```

Example 72. Consider the Secp256k1 curve from example 67 again. The following code invokes Sage to generate a random affine curve point, then applies our compression method:

check reference

```
sage: P = Secp256k1.random_point()
sage: Q = Secp256k1.random_point()
sage: INF = Secp256k1(0)
sage: R1 = -P
sage: R2 = P + Q
sage: R3 = Secp256k1.order()*P
sage: P.xy()
(2437965124411773648884901383952245798298026200193112014924104
   5920541255603582, 38155318538062562663408568861188374070643
   301057931057692802349663368915027747)
sage: Q.xy()
(6273267811834346524071370277009541823203325405903695727983144
   7554159754801518, 81206263702504109131546480004400274036228
   732572045186080577817223096074627142)
sage: (ZZ(R1[0]).str(16), ZZ(R1[1]).str(16))
('35e664c3768462813f30192e327e60c61508d279931cdbc639f3cb11c5b3
   157e', 'aba4dae1f8c83f0ac955259cd78622327b9f107d82937463dd8
   cded0c012750c')
sage: R2.xy()
(8315162076242884051827668971975027473477042355284820491860209
   945466147353499, 128083043736478847072934266448265932843478
   45733596286872839204967881615931190)
sage: R3 == INF
True
sage: P[1]+R1[1] == Fp(0) # -(x,y) = (x,-y)
True
```

Exercise 36. Consider the *PJJ_13*-curve from example 66.

check reference

1. Compute the inverse of $(10,10)$, $\mathcal{O}$, $(4,0)$ and $(1,2)$.
2. Compute the expression $3*(1,11)-(9,9)$.
3. Solve the equation $x+2(9,4) = (5,2)$ for some $x \in PJJ\_13$
4. Solve the equation $x \cdot (7,11) = (8,5)$ for $x \in \mathbb{Z}$

Scalar multiplication As we have seen in the previous section, elliptic curves $E(\mathbb{F})$ have the structure of a commutative group associated to them. Moreover, It can moreover be shown that this group is finite and cyclic whenever the field is finite.

To understand elliptic curve scalar multiplication, recall from page 43 that every finite cyclic group of order q has a generator g and an associated exponential map $g^{(\cdot)} : \mathbb{Z}_q \to \mathbb{G}$, where g^n is the n-fold product of g with itself.

check reference

Elliptic curve scalar multiplication is the exponential map written in additive notation. To be more precise, let $\mathbb{F}$ be a finite field, $E(\mathbb{F})$ an elliptic curve of order r, and P a generator of $E(\mathbb{F})$. Then the **elliptic curve scalar multiplication** with base P is defined as follows (where $[0]P = \mathcal{O}$ and $[m]P = P + P + \ldots + P$ is the m-fold sum of P with itself):

$$[\cdot]P : \mathbb{Z}_r \to E(\mathbb{F}); m \mapsto [m]P$$

therefore, elliptic curve scalar multiplication is an instantiation of the general exponential map using additive instead of multiplicative notation. This map is a homomorph of groups, which means that $[n+m]P = [n]P \oplus [m]P$.

As with all finite, cyclic groups, the inverse of the exponential map exists and is usually called the **elliptic curve discrete logarithm map**. However, elliptic curves are believed to be XXX-groups, which means that we don't know of any efficient way to actually compute this map.

add term

Scalar multiplication and its inverse, the elliptic curve discrete logarithm, define **the elliptic curve discrete logarithm problem**, which consists of finding solutions $m \in \mathbb{Z}_r$ such that the following equation holds:

$$P = [m]Q \tag{5.3}$$

Any solution m is usually called a **discrete logarithm relation** between P and Q. If Q is a generator of the curve, then there is a discrete logarithm relation between Q and any other point, since Q generates the group by repeatedly adding Q to itself. Therefore, we know that some discrete logarithm relation exists for generator Q and point P. However, since elliptic curves are believed to be XXX-groups, finding actual relations m is computationally hard, with runtimes being approximately the size of the order of the group. In practice, we often need the assumption that a discrete logarithm relation exists, while the relation itself is not known.

add term

One useful property of the exponential map in regard to the examples in this book is that it can be used to greatly simplify pen-and-paper computations. As we have seen in example XXX, computing the elliptic curve addition law takes quite a bit of effort when done without a computer. However, when g is a generator of a small pen-and-paper elliptic curve group of order r, we can use the exponential map to write the group using cofactor clearing, which implies that $[r]g = \mathcal{O}$:

add reference

cofactor clearing

$$\mathbb{G} = \{[1]g \to [2]g \to [3]g \to \cdots \to [r-1]g \to \mathcal{O}\} \tag{5.4}$$

"Logarithmic ordering" like this greatly simplifies complicated elliptic curve addition to the much simpler case of modular r addition. In order to add two curve points P and Q, we only have to look up their discrete log relations with the generator, say $P = [n]g$ and $Q = [m]g$, and compute the sum as $P \oplus Q = [n+m]g$. This is, of course, only possible for small groups where we can keep a clear overview, such as XXX.

add reference

In the following example, we will look at some implications of the fact that elliptic curves are finite cyclic groups. We will apply the fundamental theorem of finite cyclic groups and look how it reflects on the curves in consideration.

Example 73. Consider the elliptic curve group $E_1(\mathbb{F}_5)$ from example 65. Since it is a finite cyclic group of order 9, and the prime factorization of 9 is $3 \cdot 3$, we can use the fundamental

check reference

theorem of finite cyclic groups to reason about all its subgroups. In fact, since the only prime factor of 9 is 3, we know that $E_1(\mathbb{F}_5)$ has the following subgroups:

- $\mathbb{G}_1 = E_1(\mathbb{F}_5)$ is a subgroup of order 9. By definition, any group is a subgroup of itself.
- $\mathbb{G}_2 = \{(2,1),(2,4),\mathcal{O}\}$ is a subgroup of order 3. This is the subgroup associated to the prime factor 3.
- $\mathbb{G}_3 = \{\mathcal{O}\}$ is a subgroup of order 1. This is the trivial subgroup.

Moreover, since $E_1(\mathbb{F}_5)$ and all its subgroups are cyclic, we know from page 43 that they must have generators. For example, the curve point $(2,1)$ is a generator of the order 3 subgroup $\mathbb{G}_2$, since every element of $\mathbb{G}_2$ can be generated by repeatedly adding $(2,1)$ to itself:

check reference

$$
\begin{aligned}
[1](2,1) &= (2,1)\\
[2](2,1) &= (2,4)\\
[3](2,1) &= \mathcal{O}
\end{aligned}
$$

Since $(2,1)$ is a generator, we know from XXX that it gives rise to an exponential map from the finite field $\mathbb{F}_3$ onto $\mathbb{G}_2$ defined by scalar multiplication:

add reference

$$[\cdot](2,1) : \mathbb{F}_3 \to \mathbb{G}_2 \;:\; x \mapsto [x](2,1)$$

To give an example of a generator that generates the entire group $E_1(\mathbb{F}_5)$, consider the point $(0,1)$. Applying the tangent rule repeatedly, we compute as follows:

$$
\begin{array}{lcllcl}
[0](0,1) & = & \mathcal{O} & [1](0,1) & = & (0,1)\\
[2](0,1) & = & (4,2) & [3](0,1) & = & (2,1)\\
[4](0,1) & = & (3,4) & [5](0,1) & = & (3,1)\\
[6](0,1) & = & (2,4) & [7](0,1) & = & (4,3)\\
[8](0,1) & = & (0,4) & [9](0,1) & = & \mathcal{O}
\end{array}
$$

Again, since $(2,1)$ is a generator, we know from XXX that it gives rise to an exponential map. However, since the group order is not a prime number, the exponential map does not map from any field, but from the residue class ring $\mathbb{Z}_9$ only:

add reference

$$[\cdot](0,1) : \mathbb{Z}_9 \to \mathbb{G}_1 \;:\; x \mapsto [x](0,1)$$

Using the generator $(0,1)$ and its associated exponential map, we can write $E(\mathbb{F}_1)$ i logarithmic order with respect to $(0,1)$ as explained in equation 5.4. We get the following:

check reference

$$E_1(\mathbb{F}_5) = \{(0,1) \to (4,2) \to (2,1) \to (3,4) \to (3,1) \to (2,4) \to (4,3) \to (0,4) \to \mathcal{O}\}$$

This indicates that the first element is a generator, and the n-th element is the scalar product of n and the generator. To see how logarithmic orders like this simplify the computations in small elliptic curve groups, consider example 70 again. In that example, we use the chord-and-tangent rule to compute $(0,1) \oplus (4,2)$. Now, in the logarithmic order of $E_1(\mathbb{F})$, we can compute that sum much easier, since we can directly see that $(0,1) = [1](0,1)$ and $(4,2) = [2](0,1)$. We can then deduce $(0,1) \oplus (4,2) = (2,1)$ immediately, since $[1](0,1) \oplus [2](0,1) = [3](0,1) = (2,1)$.

check reference

To give another example, we can immediately see that $(3,4) \oplus (4,3) = (4,2)$, without doing any expensive elliptic curve addition, since we know $(3,4) = [4](0,1)$ as well as $(4,3) =$

$[7](0,1)$ from the logarithmic representation of $E_1(\mathbb{F}_5)$. Since $4+7=2$ in $\mathbb{Z}_9$, the result must be $[2](0,1)=(4,2)$.

Finally we can use $E_1(\mathbb{F}_5)$ as an example to understand the concept of cofactor clearing from 5.4. Since the order of $E_1(\mathbb{F}_5)$ is 9, we only have a single factor, which happen to be the cofactor as well. Cofactor clearing then implies that we can map any element from $E_1(\mathbb{F}_5)$ onto its prime factor group $\mathbb{G}_2$ by scalar multiplication with 3. For example, taking the element $(3,4)$, which is not in $\mathbb{G}_2$, and multiplying it with 3, we get $[3](3,4)=(2,1)$, which is an element of $\mathbb{G}_2$ as expected.

check reference

In the following example, we will look at the subgroups of our pen-jubjub curve, define generators, and compute the logarithmic order for pen-and-paper computations. Then we take another look at the principle of cofactor clearing.

Example 74. Consider the pen-jubjub curve *PJJ_13* from example 66 again. Since the order of *PJJ_13* is 20, and the prime factorization of 20 is $2^2 \cdot 5$, we know that the *PJJ_13* contains a "large" prime-order subgroup of size 5 and a small prime oder subgroup of size 2.

check reference

To compute those groups, we can apply the technique of cofactor clearing in a try-and-repeat loop. We start the loop by arbitrarily choosing an element $P \in PJJ\_13$, then multiplying that element with the cofactor of the group that we want to compute. If the result is $\mathcal{O}$, we try a different element and repeat the process until the result is different from the point at infinity $\mathcal{O}$.

To compute a generator for the small prime-order subgroup $(PJJ\_13)_2$, first observe that the cofactor is 10, since $20 = 2\cdot 10$. We then arbitrarily choose the curve point $(5,11) \in PJJ\_13$ and compute $[10](5,11)=\mathcal{O}$. Since the result is the point at infinity, we have to try another curve point, say $(9,4)$. We get $[10](9,4)=(4,0)$ and we can deduce that $(4,0)$ is a generator of $(PJJ\_13)_2$. Logarithmic order then gives $(PJJ\_13)_2 = \{(4,0)\to\mathcal{O}\}$ as expected, since we know from example 71 that $(4,0)$ is self-inverse, with $(4,0)\oplus(4,0)=\mathcal{O}$. We double check the computations using Sage:

check reference

```
sage: F13 = GF(13)
sage: PJJ = EllipticCurve(F13,[8,8])
sage: P = PJJ(5,11)
sage: INF = PJJ(0)
sage: 10*P == INF
True
sage: Q = PJJ(9,4)
sage: R = PJJ(4,0)
sage: 10*Q == R
True
```

We can apply the same reasoning to the "large" prime-order subgroup $(PJJ\_13)_5$, which contains 5 elements. To compute a generator for this group, first observe that the associated cofactor is 4, since $20 = 5\cdot 4$. We choose the curve point $(9,4)\in PJJ\_13$ again, and compute $[4](9,4)=(7,11)$. We can deduce that $(7,11)$ is a generator of $(PJJ\_13)_5$. Using the generator $(7,11)$, we compute the exponential map $[\cdot](7,11): \mathbb{F}_5 \to PJJ\_13$ and get the following:

Explain how

$$
\begin{aligned}
[0](7,11) &= \mathcal{O}\\
[1](7,11) &= (7,11)\\
[2](7,11) &= (8,5)\\
[3](7,11) &= (8,8)\\
[4](7,11) &= (7,2)
\end{aligned}
$$

We can use this computation to write the large-order prime group $(PJJ\_13)_5$ of the pen-jubjub curve in logarithmic order, which we will use quite frequently in what follows. We get the following:

$$(PJJ\_13)_5 = \{(7,11) \to (8,5) \to (8,8) \to (7,2) \to \mathcal{O}\} \tag{5.5}$$

From this, we can immediately see, for example that $(8,8) \oplus (7,2) = (8,5)$, since $3+4=2$ in $\mathbb{F}_5$.

From the previous two examples, the reader might get the impression that elliptic curve computation can be largely replaced by modular arithmetics. This however, is not true in general, but only an artifact of small groups, where it is possible to write the entire group in a logarithmic order. The following example gives some understanding of why this is not possible in cryptographically secure groups.

Example 75. SEKTP BICOIN. DISCRETE LOG HARDNESS PROHIBITS ADDITION IN THE FIELD... write example

Projective short Weierstraß form As we have seen in the previous section, describing elliptic curves as pairs of points that satisfy a certain equation is relatively straight-forward. However, in order to define a group structure on the set of points, we had to add a special point at infinity to act as the neutral element.

Recalling from the definition of projective planes (section 4.4), we know that points at infinity are handled as ordinary points in projective geometry. Therefore, it makes sense to look at the definition of a short Weierstraß curve in projective geometry. check reference

To see what a short Weierstraß curve in projective coordinates is, let $\mathbb{F}$ be a finite field of order q and characteristic > 3, let $a, b \in \mathbb{F}$ be two field elements such that $4a^3 + 27b^2 \bmod q \neq 0$ and let $\mathbb{F}\mathrm{P}^2$ be the projective plane over $\mathbb{F}$. Then a **short Weierstraß elliptic curve** over $\mathbb{F}$ in its projective representation is the set of all points $[X:Y:Z] \in \mathbb{F}\mathrm{P}^2$ from the projective plane that satisfy the **homogenous** cubic equation $Y^2 \cdot Z = X^3 + a \cdot X \cdot Z^2 + b \cdot Z^3$:

$$E(\mathbb{F}\mathrm{P}^2) = \{[X:Y:Z] \in \mathbb{F}\mathrm{P}^2 \mid Y^2 \cdot Z = X^3 + a \cdot X \cdot Z^2 + b \cdot Z^3\} \tag{5.6}$$

To understand how the point at infinity is unified in this definition, recall from XXX that, in projective geometry, points at infinity are given by homogeneous coordinates $[X:Y:0]$. Inserting representatives $(x_1, y_1, 0) \in [X:Y:0]$ from those classes into the defining homogenous cubic equations gives the following: add reference

$$\begin{aligned} y_1^2 \cdot 0 &= x_1^3 + a \cdot x_1 \cdot 0^2 + b \cdot 0^3 && \Leftrightarrow \\ 0 &= x_1^3 \end{aligned}$$

This shows that the only point at infinity, that is also a point on a projective short Weierstraß curve is the class $[0,1,0] = \{(0,y,0) \mid y \in \mathbb{F}\}$.

This point is the projective representation of $\mathcal{O}$. The projective representation of a short Weierstraß curve, therefore, has the advantage that it does not need a special symbol to represent the point at infinity $\mathcal{O}$ from the affine definition.

Example 76. To get an intuition of how an elliptic curve in projective geometry looks, consider curve $E_1(\mathbb{F}_5)$ from example (65). We know that, in its affine representation, the set of rational points is given as follows: check reference

$$E_1(\mathbb{F}_5) = \{\mathcal{O}, (0,1), (2,1), (3,1), (4,2), (4,3), (0,4), (2,4), (3,4)\} \tag{5.7}$$

This is defined as the set of all pairs $(x,y) \in \mathbb{F}_5 \times \mathbb{F}_5$ such that the affine short Weierstraß equation $y^2 = x^3 + ax + b$ with $a = 1$ and $b = 1$ is satisfied.

To find the projective representation of a short Weierstraß curve with the same parameters $a = 1$ and $b = 1$, we have to compute the set of projective points $[X : Y : Z]$ from the projective plane $\mathbb{F}_5\mathrm{P}^2$ that satisfy the following homogenous cubic equation for any representative $(x_1, y_1, z_1) \in [X : Y : Z]$:

$$y_1^2 z_1 = x_1^3 + 1 \cdot x_1 z_1^2 + 1 \cdot z_1^3 \tag{5.8}$$

We know from XXX that the projective plane $\mathbb{F}_5\mathrm{P}^2$ contains $5^2 + 5 + 1 = 31$ elements, so we can take the effort and insert all elements into equation 5.8 and see if both sides match.

add reference

For example, consider the projective point $[0 : 4 : 1]$. We know from XXX that this point in the projective plane represents the following line in the three-dimensional space $\mathbb{F}^3$:

check reference

add reference

$$[0 : 4 : 1] = \{(0,0,0), (0,4,1), (0,3,2), (0,2,3), (0,1,4)\}$$

To check whether or not $[0 : 4 : 1]$ satisfies 5.8, we can insert any representative, in other words, any element from XXX. Each element satisfies the equation if and only if all other elements satisfy the equation. So, we insert $(0,4,1)$ and get the following result:

check reference

add reference

$$1^2 \cdot 1 = 0^3 + 1 \cdot 0 \cdot 1^2 + 1 \cdot 1^3$$

This tells us that the affine point $[0 : 4 : 1]$ is indeed a solution to the equation 5.8, but we could just as well have inserted any other representative. For example, inserting $(0,3,2)$ also satisfies 5.8:

check reference

$$3^2 \cdot 2 = 0^3 + 1 \cdot 0 \cdot 2^2 + 1 \cdot 2^3$$

To find the projective representation of E_1, we first observe that the projective line at infinity $[1 : 0 : 0]$ is not a curve point on any projective short Weierstraß curve, since it cannot satisfy XXX for any parameter a and b. Therefore, we can exclude it from our consideration.

add reference

Moreover, a point at infinity $[X : Y : 0]$ can only satisfy equation XXX for any a and b, if $X = 0$, which implies that the only point at infinity relevant for short Weierstraß elliptic curves is $[0 : 1 : 0]$, since $[0 : k : 0] = [0 : 1 : 0]$ for all k from the finite field. Therefore, we can exclude all points at infinity except the point $[0 : 1 : 0]$.

add reference

All points that remain are the affine points $[X : Y : 1]$. Inserting all of them into XXX, we get the set of all projective curve points as follows:

add reference

$$\begin{aligned} E_1(\mathbb{F}_5\mathrm{P}^2) = \{&[0:1:0], [0:1:1], [2:1:1], [3:1:1], \\ &[4:2:1], [4:3:1], [0:4:1], [2:4:1], [3:4:1]\} \end{aligned}$$

If we compare this with the affine representation, we see that there is a 1:1 correspondence between the points in the affine representation in 5.7 and the affine points in projective geometry, and that the point $[0 : 1 : 0]$ represents the additional point $\mathcal{O}$ in the projective representation.

check reference

Exercise 37. Compute the projective representation of the pen-jubjub curve and the logarithmic order of its large prime-order subgroup with respect to the generator $(7, 11)$.

Projective Group law As we have seen on page 69, one of the key properties of an elliptic curve is that it comes with a definition of a group law on the set of its rational points, described geometrically by the chord-and-tangent rule (definition 5.1.1.1). This rule was kind of intuitive,

check reference

with the exception of the distinguished point at infinity, which appeared whenever the chord or the tangent did not have a third intersection point with the curve.

One of the key features of projective coordinates is that, in projective space, it is guaranteed that any chord will always intersect the curve in three points, and any tangent will intersect it in two points including the tangent point. So, the geometric picture simplifies, as we don't need to consider external symbols and associated cases.

Again, it can be shown that the points of an elliptic curve in projective space form a commutative group with respect to the tangent-and-chord rule such that the projective point $[0:1:0]$ is the neutral element, and the additive inverse of a point $[X:Y:Z]$ is given by $[X:-Y:Z]$. The addition law is usually described by the following algorithm, minimizing the number of necessary additions and multiplications in the base field.

Check if following Alg is floated too far

Exercise 38. Compare the affine addition law for short Weierstraß curves with the projective addition rule. Which branch in the projective rule corresponds to which case in the affine law?

Coordinate Transformations As we have seen in example XXX, there was a close relation between the affine and the projective representation of a short Weierstraß curve. This was not a coincidence. In fact, from a mathematical point of view, projective and affine short Weierstraß curves describe the same thing, as there is a one-to-one correspondence (an isomorphism) between both representations for any arbitrary parameters a and b.

add reference

To specify the isomorphism, let $E(\mathbb{F})$ and $E(\mathbb{F}\mathrm{P}^2)$ be an affine and a projective short Weierstraß curve defined for the same parameters a and b. Then the map in 5.9 maps points from the affine representation to points from the projective representation of a short Weierstraß curve. In other words, if the pair of points (x,y) satisfies the affine equation $y^2 = x^3 + ax + b$, then all homogeneous coordinates $(x_1, y_1, z_1) \in [x:y:1]$ satisfy the projective equation $y_1^2 \cdot z_1 = x_1^3 + ay_1 \cdot z_1^2 + b \cdot z_1^3$.

$$\Phi : E(\mathbb{F}) \to E(\mathbb{F}\mathrm{P}^2) \;:\; \begin{array}{lll} (x,y) & \mapsto & [x:y:1] \\ \mathcal{O} & \mapsto & [0:1:0] \end{array} \tag{5.9}$$

The inverse is given by the following map:

$$\Phi^{-1} : E(\mathbb{F}\mathbb{P}^2) \to E(\mathbb{F}) \;:\; [X:Y:Z] \mapsto \begin{cases} (\frac{X}{Z}, \frac{Y}{Z}) & \text{if } Z \neq 0 \\ \mathcal{O} & \text{if } Z = 0 \end{cases} \tag{5.10}$$

Note that the only projective point $[X:Y:Z]$ with $Z \neq 0$ that satisfies XXX is given by the class $[0:1:0]$.

add reference

One key feature of Φ and its inverse is that it respects the group structure, which means that $\Phi((x_1,y_1) \oplus (x_2,y_2))$ is equal to $\Phi(x_1,y_1) \oplus \Phi(x_2,y_2)$. The same holds true for the inverse map Φ^{-1}.

Maps with these properties are called **group isomorphisms**, and, from a mathematical point of view, the existence of Φ implies that these two definitions are equivalent, and implementations can choose freely between these representations.

5.1.2 Montgomery Curves

History and use of them (optimized scalar multiplication)

write up this part

Algorithm 6 Projective Weierstraß Addition Law

```
Require: [X_1 : Y_1 : Z_1], [X_2 : Y_2 : Z_2] ∈ E(FP^2)
  procedure ADD-RULE([X_1 : Y_1 : Z_1], [X_2 : Y_2 : Z_2])
    if [X_1 : Y_1 : Z_1] == [0 : 1 : 0] then
      [X_3 : Y_3 : Z_3] ← [X_2 : Y_2 : Z_2]
    else if [X_2 : Y_2 : Z_2] == [0 : 1 : 0] then
      [X_3 : Y_3 : Z_3] ← [X_1 : Y_1 : Z_1]
    else
      U_1 ← Y_2 · Z_1
      U_2 ← Y_1 · Z_2
      V_1 ← X_2 · Z_1
      V_2 ← X_1 · Z_2
      if V_1 == V_2 then
        if U_1 ≠ U_2 then [X_3 : Y_3 : Z_3] ← [0 : 1 : 0]
        else
          if Y_1 == 0 then [X_3 : Y_3 : Z_3] ← [0 : 1 : 0]
          else
            W ← a · Z_1^2 + 3 · X_1^2
            S ← Y_1 · Z_1
            B ← X_1 · Y_1 · S
            H ← W^2 − 8 · B
            X' ← 2 · H · S
            Y' ← W · (4 · B − H) − 8 · Y_1^2 · S^2
            Z' ← 8 · S^3
            [X_3 : Y_3 : Z_3] ← [X' : Y' : Z']
          end if
        end if
      else
        U = U_1 − U_2
        V = V_1 − V_2
        W = Z_1 · Z_2
        A = U^2 · W − V^3 − 2 · V^2 · V_2
        X' = V · A
        Y' = U · (V^2 · V_2 − A) − V^3 · U_2
        Z' = V^3 · W
        [X_3 : Y_3 : Z_3] ← [X' : Y' : Z']
      end if
    end if
    return [X_3 : Y_3 : Z_3]
  end procedure
Ensure: [X_3 : Y_3 : Z_3] == [X_1 : Y_1 : Z_1] ⊕ [X_2 : Y_2 : Z_2]
```

Affine Montgomery Form To see what a Montgomery curve in affine coordinates is, let $\mathbb{F}$ be a finite field of characteristic > 2, and let $A, B \in \mathbb{F}$ be two field elements such that $B \neq 0$ and $A^2 \neq 4$. A **Montgomery elliptic curve** $M(\mathbb{F})$ over $\mathbb{F}$ in its affine representation is the set of all pairs of field elements $(x, y) \in \mathbb{F} \times \mathbb{F}$ that satisfy the Montgomery cubic equation $B \cdot y^2 = x^3 + A \cdot x^2 + x$, together with a distinguished symbol $\mathcal{O}$, called the **point at infinity**.

$$M(\mathbb{F}) = \{(x,y) \in \mathbb{F} \times \mathbb{F} \mid B \cdot y^2 = x^3 + A \cdot x^2 + x\} \bigcup \{\mathcal{O}\} \tag{5.11}$$

is the label in LaTeX correct here?

Despite the fact that Montgomery curves look different from short Weierstraß curves, they are just a special way to describe certain short Weierstraß curves. In fact, every curve in affine Montgomery form can be transformed into an elliptic curve in Weierstraß form. To see that, assume that a curve is given in Montgomery form $By^2 = x^3 + Ax^2 + x$. The associated Weierstraß form is then as follows:

$$y^2 = x^3 + \frac{3 - A^2}{3B^2} \cdot x + \frac{2A^3 - 9A}{27B^3} \tag{5.12}$$

On the other hand, an elliptic curve $E(\mathbb{F})$ over base field $\mathbb{F}$ in Weierstraß form $y^2 = x^3 + ax + b$ can be converted to Montgomery form if and only if the following conditions hold:

Definition 5.1.2.1. **Requirements for Montgomery curves**

- The number of points on $E(F)$ is divisible by 4
- The polynomial $z^3 + az + b \in \mathbb{F}[z]$ has at least one root $z_0 \in \mathbb{F}$
- $3z_0^2 + a$ is a quadratic residue in $\mathbb{F}$.

When these conditions are satisfied, then for $s = (\sqrt{3z_0^2 + a})^{-1}$, the equivalent Montgomery curve is defined by the following equation:

$$sy^2 = x^3 + (3z_0 s)x^2 + x \tag{5.13}$$

In the following example we will look at our pen-jubjub curve again, and show that it is actually a Montgomery curve.

Example 77. Consider the prime field $\mathbb{F}_{13}$ and the pen-jubjub curve *PJJ_13* from example 66. To see that it is a Montgomery curve, we have to check the requirements from 5.1.2.1: check reference check reference

Since the order of *PJJ_13* is 20, which is divisible by 4, the first requirement is met.

Next, since $a = 8$ and $b = 8$, we have to check if the polynomial $P(z) = z^3 + 8z + 8$ has a root in $\mathbb{F}_{13}$. We simply evaluate P at all numbers $z \in \mathbb{F}_{13}$, and find that $P(4) = 0$, so a root is given by $z_0 = 4$.

In the last step, we have to check that $3 \cdot z_0^2 + a$ has a root in $\mathbb{F}_{13}$. We compute as follows:

$$\begin{aligned} 3z_0^2 + a &= 3 \cdot 4^2 + 8 \\ &= 3 \cdot 3 + 8 \\ &= 9 + 8 \\ &= 4 \end{aligned}$$

To see if 4 is a quadratic residue, we can use Euler's criterion (4.16) to compute the Legendre symbol of 4. We get the following: check reference

$$\left(\frac{4}{13}\right) = 4^{\frac{13-1}{2}} = 4^6 = 1$$

This means that 4 does have a root in $\mathbb{F}_{13}$. In fact, computing a root of 4 in $\mathbb{F}_{13}$ is easy, since the integer root 2 of 4 is also one of its roots in $\mathbb{F}_{13}$. The other root is given by $13-4=9$.

Since all requirements are meet, we have now shown that *PJJ_13* is indeed a Montgomery curve, and we can use 5.13 to compute its associated Montgomery form. We compute as follows:

check reference

$$\begin{aligned}
s &= \left(\sqrt{3\cdot z_0^2+8}\right)^{-1} \\
&= 2^{-1} && \text{\# Fermat's little theorem} \\
&= 2^{13-2} && \text{\# } 2048 \bmod 13 = 7 \\
&= 7
\end{aligned}$$

The defining equation for the Montgomery form of our pen-jubjub curve is then given by the following equation:

$$\begin{aligned}
sy^2 &= x^3+(3z_0s)x^2+x && \Rightarrow \\
7\cdot y^2 &= x^3+(3\cdot 4\cdot 7)x^2+x && \Leftrightarrow \\
7\cdot y^2 &= x^3+6x^2+x
\end{aligned}$$

So, we get the defining parameters as $B=7$ and $A=6$, and we can write the pen-jubjub curve in its affine Montgomery representation as follows:

$$PJJ\_13 = \{(x,y)\in\mathbb{F}_{13}\times\mathbb{F}_{13} \mid 7\cdot y^2 = x^3+6x^2+x\}\bigcup\{\mathcal{O}\} \tag{5.14}$$

Now that we have the abstract definition of our pen-jubjub curve in Montgomery form, we can compute the set of points by inserting all pairs $(x,y)\in\mathbb{F}_{13}\times\mathbb{F}_{13}$ similarly to how we computed the curve points in its Weierstraß representation. We get the following:

$$\begin{aligned}
PJJ_13 = \{&\mathcal{O},(0,0),(1,4),(1,9),(2,4),(2,9),(3,5),(3,8),(4,4),(4,9),\\
&(5,1),(5,12),(7,1),(7,12),(8,1),(8,12),(9,2),(9,11),(10,3),(10,10)\}
\end{aligned}$$

```
sage: F13 = GF(13)
sage: L_MPJJ = []
....: for x in F13:
....:     for y in F13:
....:         if F13(7)*y^2 == x^3 + F13(6)*x^2 +x:
....:             L_MPJJ.append((x,y))
sage: MPJJ = Set(L_MPJJ)
sage: # does not compute the point at infinity
```

Affine Montgomery coordinate transformation Comparing the Montgomery representation of the previous example (equation 5.14) with the Weierstraß representation of the same curve (equation 5.2), we see that there is a 1:1 correspondence between the curve points in both

check reference

examples. This is no accident. In fact, if $M_{A,B}$ is a Montgomery curve, and $E_{a,b}$ a Weierstraß curve with $a = \frac{3-A^2}{3B^2}$ and $b = \frac{2A^2-9A}{27B^3}$ then the following function maps all points in Montgomery representation onto the points in Weierstraß representation:

$$\Phi : M_{A,B} \to E_{a,b} \ : \ (x,y) \mapsto \left(\frac{3x+A}{3B}, \frac{y}{B}\right) \tag{5.15}$$

This map is a 1:1 correspondence (am isomorphism), and its inverse map is given by the following equation (where z_0 is a root of the polynomial $z^3+az+b \in \mathbb{F}[z]$ and $s = (\sqrt{3z_0^2+a})^{-1}$).

$$\Phi^{-1} : E_{a,b} \to M_{A,B} \ : \ (x,y) \mapsto (s\cdot(x-z_0), s\cdot y) \tag{5.16}$$

Using this map, it is therefore possible for implementations of Montgomery curves to freely transit between the Weierstraß and the Montgomery representation. However, as we saw in definition 5.1.2.1, not every Weierstraß curve is a Montgomery curve, as all criteria in 5.1.2.1 have to be satisfied. This means that the map Φ^{-1} does not always exist.

check reference

Example 78. Consider our pen-jubjub curve again. In equation 5.2 we derived its Weierstraß representation and in example 5.14, we derived its Montgomery representation.

check reference

check reference

To see how coordinate transformation Φ works in this example, let's map points from the Montgomery representation onto points from the Weierstraß representation. Inserting, for example, the point $(0,0)$ from the Montgomery representation 5.14 into Φ gives the following:

check reference

check reference

$$\begin{aligned}\Phi(0,0) &= \left(\frac{3\cdot 0+A}{3B}, \frac{0}{B}\right)\\ &= \left(\frac{3\cdot 0+6}{3\cdot 7}, \frac{0}{7}\right)\\ &= \left(\frac{6}{8}, 0\right)\\ &= (4,0)\end{aligned}$$

As we can see, the Montgomery point $(0,0)$ maps to the self-inverse point $(4,0)$ of the Weierstraß representation. On the other hand, we can use our computations of $s=7$ and $z_0=4$ from XXX to compute the inverse map Φ^{-1}, which maps points on the Weiertraß representation to points on the Mongomery form. Inserting, for example, $(4,0)$ we get the following:

add reference

$$\begin{aligned}\Phi^{-1}(4,0) &= (s\cdot(4-z_0), s\cdot 0)\\ &= (7\cdot(4-4), 0)\\ &= (0,0)\end{aligned}$$

As expected, the inverse map maps the Weierstraß point back to where it originated in the Montgomery form. We can invoke Sage to check that our computation of Φ is correct:

```
sage: # Compute PHI of Montgomery form:
sage: L_PHI_MPJJ = []
sage: for (x,y) in L_MPJJ: # LMJJ as defined previously
....:     v = (F13(3)*x + F13(6))/(F13(3)*F13(7))
....:     w = y/F13(7)
....:     L_PHI_MPJJ.append((v,w))
```

```
sage: PHI_MPJJ = Set(L_PHI_MPJJ)
sage: # Computation Weierstrass form
sage: C_WPJJ = EllipticCurve(F13,[8,8])
sage: L_WPJJ = [P.xy() for P in C_WPJJ.points() if P.order() >
    1]
sage: WPJJ = Set(L_WPJJ)
sage: # check PHI(Montgomery) == Weierstrass
sage: WPJJ == PHI_MPJJ
True
sage: # check the inverse map PHI^(-1)
sage: L_PHIINV_WPJJ = []
sage: for (v,w) in L_WPJJ:
....:     x = F13(7)*(v-F13(4))
....:     y = F13(7)*w
....:     L_PHIINV_WPJJ.append((x,y))
sage: PHIINV_WPJJ = Set(L_PHIINV_WPJJ)
sage: MPJJ == PHIINV_WPJJ
True
```

Montgomery group law We have seen that Montgomery curves special cases of short Weierstraß curves. As such, they have a group structure defined on the set of their points, which can also be derived from the chord-and-tangent rule. In accordance with short Weierstraß curves, it can be shown that the identity $x_1 = x_2$ implies $y_2 = \pm y_1$, meaning that the following rules are a complete description of the affine addition law.

Definition 5.1.2.2. **Montgomery group law**

- (Neutral element) Point at infinity $\mathcal{O}$ is the neutral element.
- (Additive inverse) The additive inverse of $\mathcal{O}$ is $\mathcal{O}$. For any other curve point $(x,y) \in M(\mathbb{F}_q)\backslash\{\mathcal{O}\}$, the additive inverse is given by $(x,-y)$.
- (Addition rule) For any two curve points $P,Q \in M(\mathbb{F}_q)$, addition is defined by one of the following cases:
    1. (Adding the neutral element) If $Q = \mathcal{O}$, then the sum is defined as $P+Q = P$.
    2. (Adding inverse elements) If $P = (x,y)$ and $Q = (x,-y)$, the sum is defined as $P+Q = \mathcal{O}$.
    3. (Adding non-self-inverse equal points) If $P = (x,y)$ and $Q = (x,y)$ with $y \neq 0$, the sum $2P = (x',y')$ is defined as follows:

    $$x' = \left(\frac{3x_1^2+2Ax_1+1}{2By_1}\right)^2 \cdot B - (x_1+x_2) - A \quad , \quad y' = \frac{3x_1^2+2Ax_1+1}{2By_1}(x_1-x') - y_1$$

    4. (Adding non-inverse different points) If $P = (x_1,y_1)$ and $Q = (x_2,y_2)$ such that $x_1 \neq x_2$, the sum $R = P+Q$ with $R = (x_3,y_3)$ is defined as follows:

    $$x' = \left(\frac{y_2-y_1}{x_2-x_1}\right)^2 B - (x_1+x_2) - A \quad , \quad y' = \frac{y_2-y_1}{x_2-x_1}(x_1-x') - y_1$$

5.1.3 Twisted Edwards Curves

As we have seen in 5.1.2.2 both Weierstraß and Montgomery curves have somewhat complicated addition and doubling laws, as many cases have to be distinguished. Those various cases translate to branches in computer programs.

check reference

In the context of SNARK development, two computational models for bounded computations are used, called **circuits** and **rank-1 constraint systems**. Program branches are undesirably costly when implemented in those models. It is therefore advantageous to look for curves with an addition/doubling rule that requires no branches and as few field operations as possible.

Twisted Edwards curves are particularly useful here, as a subclass of these curves has a compact and easily implementable addition law that works for all points including the point at infinity. Implementing this law needs no branching.

Twisted Edwards Form To see what an affine **twisted Edwards curve** looks like, let $\mathbb{F}$ be a finite field of characteristic > 2, and let $a,d \in \mathbb{F}\backslash\{0\}$ be two non-zero field elements with $a \neq d$. A **twisted Edwards elliptic curve** in its affine representation is the set of all pairs (x,y) from $\mathbb{F}\times\mathbb{F}$ that satisfy the twisted Edwards equation $a\cdot x^2 + y^2 = 1 + d\cdot x^2y^2$, given below:

$$E(\mathbb{F}) = \{(x,y) \in \mathbb{F}\times\mathbb{F} \mid a\cdot x^2 + y^2 = 1 + d\cdot x^2y^2\} \tag{5.17}$$

A twisted Edwards curve is called an **Edwards curve (non-twisted)**, if the parameter a is equal to 1, and it is called a **SNARK-friendly twisted Edwards curve** if the parameter a is a quadratic residue and the parameter d is a quadratic non-residue.

As we can see from the definition, affine twisted Edwards curves look somewhat different from Weierstraß curves, as their affine representation does not need a special symbol to represent the point at infinity. In fact, we we will see that the pair $(0,1)$ is always a point on any twisted Edwards curve, and that it takes the role of the point at infinity.

Despite their different appearances however, twisted Edwards curves are equivalent to Montgomery curves in the sense that, for every twisted Edwards curve, there is a Montgomery curve, and a way to map the points of one curve in a 1:1 correspondence onto the other and vice versa. To see that, assume that a curve in twisted Edwards form $a\cdot x^2 + y^2 = 1 + d\cdot x^2y^2$ is given. The associated Montgomery curve is then defined by the Montgomery equation:

$$\frac{4}{a-d}y^2 = x^3 + \frac{2(a+d)}{a-d}\cdot x^2 + x \tag{5.18}$$

On the other hand, a Montgomery curve $By^2 = x^3 + Ax^2 + x$ with $B \neq 0$ and $A^2 \neq 4$ can give rise to a twisted Edwards curve defined by the following equation:

$$(\frac{A+2}{B})x^2 + y^2 = 1 + (\frac{A-2}{B})x^2y^2 \tag{5.19}$$

As we have seen in equation 5.12 and the following discussion, Montgomery curves are just a special class of Weierstraß curves. Furthermore we now know that twisted Edwards curves are special Weierstraß curves too. This means that the more general way to describe elliptic curves is as Weierstraß curves.

check reference

Example 79. Consider the pen-jubjub curve from example 66 again. We know from example 77 that it is a Montgomery curve, and, since Montgomery curves are equivalent to twisted Edwards curves, we want to write this curve in twisted Edwards form. We use equation 5.19,

check reference

check reference

check reference

and compute the parameters a and d as follows:

$$\begin{aligned} a &= \frac{A+2}{B} && \text{\# insert A=6 and B=7} \\ &= \frac{8}{7} = 3 && \# \, 7^{-1} = 2 \end{aligned}$$

$$\begin{aligned} d &= \frac{A-2}{B} \\ &= \frac{4}{7} = 8 \end{aligned}$$

Thus, we get the defining parameters as $a = 3$ and $d = 8$. Since our goal is to use this curve later on in implementations of pen-and-paper SNARKs, let us show that tiny-jubjub is also a **SNARK-friendly** twisted Edwards curve. To see that, we have to show that a is a quadratic residue and d is a quadratic non-residue. We therefore compute the Legendre symbols of a and d using Euler's criterion. We get the following:

change "tiny-jubjub" to "pen-jubjub" throughout?

$$\begin{aligned} \left(\frac{3}{13}\right) &= 3^{\frac{13-1}{2}} \\ &= 3^6 = 1 \end{aligned}$$

$$\begin{aligned} \left(\frac{8}{13}\right) &= 8^{\frac{13-1}{2}} \\ &= 8^6 = 12 = -1 \end{aligned}$$

This proves that tiny-jubjub is SNARK-friendly. We can write the tiny-jubjub curve in its affine twisted Edwards representation as follows:

$$TJJ\_13 = \{(x,y) \in \mathbb{F}_{13} \times \mathbb{F}_{13} \mid 3 \cdot x^2 + y^2 = 1 + 8 \cdot x^2 \cdot y^2\} \tag{5.20}$$

Now that we have the abstract definition of our pen-jubjub curve in twisted Edwards form, we can compute the set of points by inserting all pairs $(x,y) \in \mathbb{F}_{13} \times \mathbb{F}_{13}$, similarly to how we computed the curve points in its Weierstraß or Edwards representation. We get the following:

$$\begin{aligned} PJJ\_13 = \{&(0,1),(0,12),(1,2),(1,11),(2,6),(2,7),(3,0),(5,5),(5,8),(6,4), \\ &(6,9),(7,4),(7,9),(8,5),(8,8),(10,0),(11,6),(11,7),(12,2),(12,11)\} \end{aligned} \tag{5.21}$$

```
sage: F13 = GF(13)
sage: L_EPJJ = []
....: for x in F13:
....:     for y in F13:
....:         if F13(3)*x^2 + y^2 == 1+ F13(8)*x^2*y^2:
....:             L_EPJJ.append((x,y))
sage: EPJJ = Set(L_EPJJ)
```

Twisted Edwards group law As we have seen, twisted Edwards curves are equivalent to Montgomery curves, and, as such, also have a group law. However, in contrast to Montgomery and Weierstraß curves, the group law of SNARK-friendly twisted Edwards curves can be described by a single computation that works in all cases, no matter if we add the neutral element, the inverse, or if we have to double a point. To see what the group law looks like, first observe that the point $(0,1)$ is a solution to $a\cdot x^2 + y^2 = 1 + d\cdot x^2\cdot y^2$ for any curve. The sum of any two points (x_1,y_1), (x_2,y_2) on an Edwards curve $E(\mathbb{F})$ is then given by the following equation:

$$(x_1,y_1)\oplus(x_2,y_2) = \left(\frac{x_1y_2+y_1x_2}{1+dx_1x_2y_1y_2}, \frac{y_1y_2-ax_1x_2}{1-dx_1x_2y_1y_2}\right) \tag{5.22}$$

and it can be shown that the point $(0,1)$ serves as the neutral element and the inverse of a point (x_1,y_1) is given by $(-x_1,y1)$.

Example 80. Lets look at the tiny-jubjub curve in Edwards form from example 5.20 again. As we have seen, this curve is given by check reference

$$\begin{aligned}PJJ\_13 = \{&(0,1),(0,12),(1,2),(1,11),(2,6),(2,7),(3,0),(5,5),(5,8),(6,4),\\ &(6,9),(7,4),(7,9),(8,5),(8,8),(10,0),(11,6),(11,7),(12,2),(12,11)\}\end{aligned}$$

To get an understanding of the twisted Edwards addition law, let's first add the neutral element $(0,1)$ to itself. We apply the group law 5.22 and get the following: check reference

$$\begin{aligned}(0,1)\oplus(0,1) &= \left(\frac{0\cdot1+1\cdot0}{1+8\cdot0\cdot0\cdot1\cdot1}, \frac{1\cdot1-3\cdot0\cdot0}{1-8\cdot0\cdot0\cdot1\cdot1}\right)\\ &= (0,1)\end{aligned}$$

So, as expected, adding the neutral element added to itself gives the neutral element again. Now let's add the neutral element to some other curve point. We get the following:

$$\begin{aligned}(0,1)\oplus(8,5) &= \left(\frac{0\cdot5+1\cdot8}{1+8\cdot0\cdot8\cdot1\cdot5}, \frac{1\cdot5-3\cdot0\cdot8}{1-8\cdot0\cdot8\cdot1\cdot5}\right)\\ &= (8,5)\end{aligned}$$

Again, as expected, adding the neutral element to any element will result in that element again. Given any curve point (x,y), we know that its inverse is given by $(-x,y)$. To see how the addition of a point to its inverse works, we compute as follows:

$$\begin{aligned}(5,5)\oplus(8,5) &= \left(\frac{5\cdot5+5\cdot8}{1+8\cdot5\cdot8\cdot5\cdot5}, \frac{5\cdot5-3\cdot5\cdot8}{1-8\cdot5\cdot8\cdot5\cdot5}\right)\\ &= \left(\frac{12+1}{1+5}, \frac{12-3}{1-5}\right)\\ &= \left(\frac{0}{6}, \frac{12+10}{1+8}\right)\\ &= \left(0, \frac{9}{9}\right)\\ &= (0,1)\end{aligned}$$

Adding a curve point to its inverse gives the neutral element, as expected. As we have seen from these examples, the twisted Edwards addition law handles edge cases particularly well and in a unified way.

5.2 Elliptic Curve Pairings

As we have seen in equation 4.1, some groups come with the notation of a so-called pairing map, which is a non-degenerate bilinear map from two groups into another group.

check reference

In this section, we discuss **pairings on elliptic curves**, which form the basis of several zk-SNARKs and other zero-knowledge proof schemes. The SNARKs derived from pairings have the advantage of constant proof sizes, which is crucial to blockchains.

We start out by defining elliptic curve pairings and discussing a simple application which bears some resemblance to more advanced SNARKs. We then introduce the pairings arising from elliptic curves and describe Miller's algorithm, which makes these pairings practical rather than just theoretically interesting.

Elliptic curves have a few structures, like the Weil or the Tate map that qualifies as pairing.

either expand on this or delete it

Embedding Degrees As we will see in what follows, every elliptic curves gives rise to a pairing map. However, we will also see in example XXX that not every such pairing can be efficiently computed. In order to distinguish curves with efficiently computable pairings from the rest, we need to start with an introduction to the so-called **embedding degree** of a curve.

add reference

Definition 5.2.0.1. **Embedding degree**
Let $\mathbb{F}$ be a finite field, let $E(\mathbb{F})$ be an elliptic curve over $\mathbb{F}$, and let n be a prime number that divides the order of $E(\mathbb{F})$. The embedding degree of $E(\mathbb{F})$ with respect to n is then the smallest integer k such that n divides $q^k - 1$.

Fermat's little theorem (page 21 ff.) implies that every curve has at least **some** embedding degree k, since at least $k = n - 1$ is always a solution to the congruency $q^k \equiv 1 \pmod{n}$. This implies that the remainder of the integer division of $q^k - 1$ by n is 0.

check reference

Example 81. To get a better intuition of the embedding degree, let's consider the elliptic curve $E_1(\mathbb{F}_5)$ from example 65. We know from 65 that the order of $E_1(\mathbb{F}_5)$ is 9, and, since the only prime factor of 9 is 3, we compute the embedding degree of $E_1(\mathbb{F}_5)$ with respect to 3.

check reference

To find the embedding degree, we have to find the smallest integer k such that 3 divides $q^k - 1 = 5^k - 1$. We try and increment until we find a proper k.

check reference

$$
\begin{aligned}
&k = 1\colon\ 5^1 - 1 = 4 && \text{not divisible by 3}\\
&k = 2\colon\ 5^2 - 1 = 24 && \text{divisible by 3}
\end{aligned}
$$

Now we know that the embedding degree of $E_1(\mathbb{F}_5)$ is 2 relative to the the prime factor 3.

Example 82. Let us consider the tiny jubjub curve *TJJ_13* from example 66. We know from 66 that the order of *TJJ_13* is 20, and that the order therefore has two prime factors. A "large" prime factor 5 and a small prime factor 2.

check reference

We start by computing the embedding degree of *TJJ_13* with respect to the large prime factor 5. To find that embedding degree, we have to find the smallest integer k such that 5 divides $q^k - 1 = 13^k - 1$. We try and increment until we find a proper k.

check reference

$$
\begin{aligned}
&k = 1\colon\ 13^1 - 1 = 12 && \text{not divisible by 5}\\
&k = 2\colon\ 13^2 - 1 = 168 && \text{not divisible by 5}\\
&k = 3\colon\ 13^3 - 1 = 2196 && \text{not divisible by 5}\\
&k = 4\colon\ 13^4 - 1 = 28560 && \text{divisible by 5}
\end{aligned}
$$

Now we know that the embedding degree of *TJJ_13* is 4 relative to the the prime factor 5.

In real-world applications, like on pairing-friendly elliptic curves such as BLS_12-381, usually only the embedding degree of the large prime factor is relevant, which in the case of our tiny-jubjub curve is represented by 5. It should be noted, however that every prime factor of a curve's order has its own notation of embedding degree despite the fact that this is mostly irrelevant in applications.

To find the embedding degree of the small prime factor 2, we have to find the smallest integer k such that 2 divides $q^k - 1 = 13^k - 1$. We try and increment until we find a proper k.

$$k = 1\colon\ 13^1 - 1 = 12 \qquad\qquad \text{divisible by } 2$$

Now we know that the embedding degree of *TJJ_13* is 1 relative to the the prime factor 2. As we have seen, different prime factors can have different embedding degrees in general.

```
sage: p = 13
sage: # large prime factor
sage: n = 5
sage: for k in range(1,5): # Fermat's little theorem
....:     if (p^k-1)%n == 0:
....:         break
sage: k
4
sage: # small prime factor
sage: n = 2
sage: for k in range(1,2): # Fermat's little theorem
....:     if (p^k-1)%n == 0:
....:         break
sage: k
1
```

Example 83. To give an example of a cryptographically secure real-world elliptic curve that does not have a small embedding degree, let's look at curve Secp256k1 again. We know from 67 that the order of this curve is a prime number, so we only have a single embedding degree.

check reference

To test potential embedding degrees k, say, in the range $1 \ldots 1000$, we can invoke Sage and compute as follows:

```
sage: p = 115792089237316195423570985008687907853269984665640564039457584007908834671663
sage: n = 115792089237316195423570985008687907852837564279074904382605163141518161494337
sage: for k in range(1,1000):
....:     if (p^k-1)%n == 0:
....:         break
sage: k
999
```

We see that Secp256k1 has at least no embedding degree $k < 1000$, which renders Secp256k1 a curve that has no small embedding degree. This property will be of importance later on.

Elliptic Curves over extension fields Suppose that p is a prime number, and $\mathbb{F}_p$ its associated prime field. We know from equation 4.17 that the fields $\mathbb{F}_{p^m}$ are extensions of $\mathbb{F}_p$ in the sense

check reference

that $\mathbb{F}_p$ is a subfield of $\mathbb{F}_{p^m}$. This implies that we can extend the affine plane that an elliptic curve is defined on by changing the base field to any extension field. To be more precise, let $E(\mathbb{F}) = \{(x,y) \in \mathbb{F}\times\mathbb{F} \mid y^2 = x^3 + a\cdot x + b\}$ be an affine short Weierstraß curve, with parameters a and b taken from $\mathbb{F}$. If $\mathbb{F}'$ is an extension field of $\mathbb{F}$, then we extend the domain of the curve by defining $E(\mathbb{F}')$ as follows:

$$E(\mathbb{F}') = \{(x,y) \in \mathbb{F}'\times\mathbb{F}' \mid y^2 = x^3 + a\cdot x + b\} \tag{5.23}$$

While we did not change the defining parameters, we consider curve points from the affine plane over the extension field now. Since $\mathbb{F} \subset \mathbb{F}'$, it can be shown that the original elliptic curve $E(\mathbb{F})$ is a sub-curve of the extension curve $E(\mathbb{F}')$.

Example 84. Consider the prime field $\mathbb{F}_5$ from example 59 and the elliptic curve $E_1(\mathbb{F}_5)$ from example 65. Since we know from XXX that $\mathbb{F}_{5^2}$ is an extension field of $\mathbb{F}_5$, we can extend the definition of $E_1(\mathbb{F}_5)$ to define a curve over $\mathbb{F}_{5^2}$:

check reference

check reference

$$E_1(\mathbb{F}_{5^2}) = \{(x,y) \in \mathbb{F}\times\mathbb{F} \mid y^2 = x^3 + x + 1\}$$

Since $\mathbb{F}_{5^2}$ contains 25 points, in order to compute the set $E_1(\mathbb{F}_{5^2})$, we have to try $25\cdot 25 = 625$ pairs, which is probably a bit too much for the average motivated reader. Instead, we invoke Sage to compute the curve for us. To do, we so choose the representation of $\mathbb{F}_{5^2}$ from XXX. We get:

add reference

add reference

```
sage: F5= GF(5)
sage: F5t.<t> = F5[]
sage: P = F5t(t^2+2)
sage: P.is_irreducible()
True
sage: F5_2.<t> = GF(5^2, name='t', modulus=P)
sage: E1F5_2 = EllipticCurve(F5_2,[1,1])
sage: E1F5_2.order()
27
```

The curve $E_1(\mathbb{F}_{5^2})$ consist of 27 points, in contrast to curve $E_1(\mathbb{F}_5)$, which consists of 9 points. Printing the points gives the following:

$$\begin{aligned}E_1(\mathbb{F}_{5^2}) = \{&\mathcal{O},(0,4),(0,1),(3,4),(3,1),(4,3),(4,2),(2,4),(2,1),\\ &(4t+3,3t+4),(4t+3,2t+1),(3t+2,t),(3t+2,4t),\\ &(2t+2,t),(2t+2,4t),(2t+1,4t+4),(2t+1,t+1),\\ &(2t+3,3),(2t+3,2),(t+3,2t+4),(t+3,3t+1),\\ &(3t+1,t+4),(3t+1,4t+1),(3t+3,3),(3t+3,2),(1,4t)\}\end{aligned}$$

As we can see, curve $E_1(\mathbb{F}_5)$ sits inside curve $E(\mathbb{F}_{5^2})$, which is implied from $\mathbb{F}_5$ being a subfield of $\mathbb{F}_{5^2}$.

Full torsion groups The fundamental theorem of finite cyclic groups XXX implies that every prime factor n of a cyclic group's order defines a subgroup of the size of the prime factor. Such a subgroup is called an n-torsion group. We have seen many of those subgroups in the examples XXX and XXX.

This needs to be written (in Algebra)

When we consider elliptic curve extensions as defined in 5.23, we could ask what happens to the n-torsion groups in the extension. One might intuitively think that their extension just

add reference

add reference

parallels the extension of the curve. For example, when $E(\mathbb{F}_p)$ is a curve over prime field $\mathbb{F}_p$, with some n-torsion group $\mathbb{G}$ and when we extend the curve to $E(\mathbb{F}_{p^m})$, then there is a bigger n-torsion group such that $\mathbb{G}$ is a subgroup. This might make intuitive sense, as $E(\mathbb{F}_p)$ is a sub-curve of $E(\mathbb{F}_{p^m})$.

However, the actual situation is a bit more surprising than that. To see that, let $\mathbb{F}_p$ be a prime field and let $E(\mathbb{F}_p)$ be an elliptic curve of order r, with embedding degree k and n-torsion group $E(\mathbb{F}_p)[n]$ for the same prime factor n of r. Then it can be shown that the n-torsion group $E(\mathbb{F}_{p^m})[n]$ of a curve extension is equal to $E(\mathbb{F}_p)[n]$, as long as the power m is less than the embedding degree k of $E(\mathbb{F}_p)$.

However, for the prime power p^m, for any $m \geq k$, $E(\mathbb{F}_{p^m})[n]$ is strictly larger than $E(\mathbb{F}_p)[n]$ and contains $E(\mathbb{F}_p)[n]$ as a subgroup. We call the n-torsion group $E(\mathbb{F}_{p^k})[n]$ of the extension of E over $\mathbb{F}_{p^k}$ the **full n-torsion group** of that elliptic curve. It can be shown that it contains n^2 many elements and consists of $n+1$ subgroups, one of which is $E(\mathbb{F}_p)[n]$.

So, roughly speaking, when we consider towers of curve extensions $E(\mathbb{F}_{p^m})$ ordered by the prime power m, then the n-torsion group stays constant for every level m, that is smaller than the embedding degree, while it suddenly blossoms into a larger group on level k with $n+1$ subgroups, and then stays like that for any level m larger than k. In other words, once the extension field is big enough to find one more point of order n (that is not defined over the base field), then we actually find all of the points in the full torsion group.

towers of curve extensions

Example 85. Consider curve $E_1(\mathbb{F}_5)$ again. We know that it contains a 3-torsion group and that the embedding degree of 3 is 2. From this we can deduce that we can find the full 3-torsion group $E_1[3]$ in the curve extension $E_1(\mathbb{F}_{5^2})$, the latter of which we computed in example 84 .

check reference

Since that curve is small, in order to find the full 3-torsion, we can loop through all elements of $E_1(\mathbb{F}_{5^2})$ and check check the defining equation $[3]P = \mathcal{O}$. Invoking Sage, we compute as follows:

```
sage: INF = E1F5_2(0) # Point at infinity
sage: L_E1_3 = []
sage: for p in E1F5_2:
....:     if 3*p == INF:
....:         L_E1_3.append(p)
sage: E1_3 = Set(L_E1_3) # Full 3-torsion set
```

We get the following result:

$$E_1[3] = \{\mathcal{O}, (1,t), (1,4t), (2,1), (2,4), (2t+1,t+1), (2t+1,4t+4), (3t+1,t+4), (3t+1,4t+1)\}$$

Example 86. Consider the tiny jubjub curve from example 66. We know from example 82 that it contains a 5-torsion group and that the embedding degree of 5 is 4. This implies that we can find the full 5-torsion group *TJJ_13*[5] in the curve extension *TJJ_13*$(\mathbb{F}_{13^4})$.

check reference

check reference

To compute the full torsion, first observe that, since $\mathbb{F}_{13^4}$ contains 28561 elements, computing *TJJ_13*$(\mathbb{F}_{13^4})$ means checking $28561^2 = 815730721$ elements. From each of these curve points P, we then have to check the equation $[5]P = \mathcal{O}$. Doing this for 815730721 is a bit too slow even on a computer.

Fortunately, Sage has a way to loop through points of a given order efficiently. The following Sage code provides a way to compute the full torsion group:

```
sage: # define the extension field
sage: F13= GF(13) # prime field
sage: F13t.<t> = F13[] # polynomials over t
```

```
sage: P = F13t(t^4+2) # irreducible polynomial of degree 4
sage: P.is_irreducible()
True
sage: F13_4.<t> = GF(13^4, name='t', modulus=P) # F_{13^4}
sage: TJJF13_4 = EllipticCurve(F13_4,[8,8]) # tiny jubjub
    extension
sage: # compute the full 5-torsion
sage: L_TJJF13_4_5 = []
sage: INF = TJJF13_4(0)
sage: for P in INF.division_points(5): # [5]P == INF
....:     L_TJJF13_4_5.append(P)
sage: len(L_TJJF13_4_5)
25
sage: TJJF13_4_5 = Set(L_TJJF13_4_5)
```

As expected, we get a group that contains $5^2 = 25$ elements. As it's rather tedious to write this group down, and as we don't need it in what follows, we forgo doing this. To see that the embedding degree 4 is actually the smallest prime power to find the full 5-torsion group, let's compute the 5-torsion group over of the tiny-jubjub curve of the extension field $\mathbb{F}_{13^3}$. We get the following:

```
sage: # define the extension field
sage: P = F13t(t^3+2) # irreducible polynomial of degree 3
sage: P.is_irreducible()
True
sage: F13_3.<t> = GF(13^3, name='t', modulus=P) # F_{13^3}
sage: TJJF13_3 = EllipticCurve(F13_3,[8,8]) # tiny jubjub
    extension
sage: # compute the 5-torsion
sage: L_TJJF13_3_5 = []
sage: INF = TJJF13_3(0)
sage: for P in INF.division_points(5): # [5]P == INF
....:     L_TJJF13_3_5.append(P)
sage: len(L_TJJF13_3_5)
5
sage: TJJF13_3_5 = Set(L_TJJF13_3_5) # full $5$-torsion
```

As we can see, the 5-torsion group of tiny-jubjub over $\mathbb{F}_{13^3}$ is equal to the 5-torsion group of tiny-jubjub over $\mathbb{F}_{13}$ itself.

Example 87. Let's look at the curve Secp256k1. We know from example 67 that the curve is of some prime order r. Because of this, the only n-torsion group to consider is the curve itself, so the curve group is the r-torsion.

check reference

However, in order to find the full r-torsion of Secp256k1, we need to compute the embedding degree k. And as we have seen in XXX it is at least not small. However, we know from Fermat's little theorem (page 21 ff.) that a finite embedding degree must exist. It can be shown that it is given by the following 256-bit number:

add reference

$$k = 192986815395526992372618308347813175472927379845817397100860523586360249056$$

This means that the embedding degree is huge, which implies that the field extension $\mathbb{F}_{p^k}$ is huge too. To understand how big $\mathbb{F}_{p^k}$ is, recall that an element of $\mathbb{F}_{p^m}$ can be represented as a

is "huge" a technical term?

string $[x_0,\ldots,x_m]$ of m elements, each containing a number from the prime field $\mathbb{F}_p$. Now, in the case of Secp256k1, such a representation has k-many entries, each of them 256 bits in size. So, without any optimizations, representing such an element would need $k \cdot 256$ bits, which is too much to be represented in the observable universe.

Torsion subgroups As we have stated above, any full n-torsion group contains $n+1$ cyclic subgroups, two of which are of particular interest in pairing-based elliptic curve cryptography. To characterize these groups, we need to consider the so-called **Frobenius endomorphism** of an elliptic curve $E(\mathbb{F})$ over some finite field $\mathbb{F}$ of characteristic p:

$$\pi : E(\mathbb{F}) \to E(\mathbb{F}) : \begin{array}{ccc} (x,y) & \mapsto & (x^p,y^p) \\ \mathcal{O} & \mapsto & \mathcal{O} \end{array} \tag{5.24}$$

It can be shown that π maps curve points to curve points. The first thing to note is that, in case $\mathbb{F}$ is a prime field, the Frobenius endomorphism acts trivially, since $(x^p,y^p) = (x,y)$ on prime fields due to Fermat's little theorem (page 21 ff.). This means that the Frobenius map is more interesting over prime field extensions.

check reference

With the Frobenius map at hand, we can characterize two important subgroups of the full n-torsion. The first subgroup is the n-torsion group that already exists in the curve over the base field. In pairing-based cryptography, this group is usually written as $\mathbb{G}_1$, assuming that the prime factor n in the definition is implicitly given. Since we know that the Frobenius map acts trivially on curves over the prime field, we can define $\mathbb{G}_1$ as follows:

$$\mathbb{G}_1[n] := \{(x,y) \in E[n] \mid \pi(x,y) = (x,y)\ \} \tag{5.25}$$

In more mathematical terms, this definition means that $\mathbb{G}_1$ is the **Eigenspace** of the Frobenius map with respect to the **Eigenvalue** 1.

S: either add more explanation or move to a footnote

It can be shown that there is another subgroup of the full n-torsion group that can be characterized by the Frobenius map. In the context of so-called type 3 pairing-based cryptography, this subgroup is usually called $\mathbb{G}_2$ and it is defined as follows:

type 3 pairing-based cryptography

$$\mathbb{G}_2[n] := \{(x,y) \in E[n] \mid \pi(x,y) = [p](x,y)\ \} \tag{5.26}$$

In mathematical terms, $\mathbb{G}_2$ is the **Eigenspace** of the Frobenius map with respect to the **Eigenvalue** p.

Notation and Symbols 9. If the prime factor n of a curve's order is clear from the context, we sometimes simply write $\mathbb{G}_1$ and $\mathbb{G}_2$ to mean $\mathbb{G}_1[n]$ and $\mathbb{G}_2[n]$, respectively.

It should be noted, however that other definitions of $\mathbb{G}_2$ also exists in the literature. However, in the context of pairing-based cryptography, this is the most common one. It is particularly useful because we can define hash functions that map into $\mathbb{G}_2$, which is not possible for all subgroups of the full n-torsion.

add references?

Example 88. Consider the curve $E_1(\mathbb{F}_5)$ from example 65 again. As we have seen, this curve has the embedding degree $k=2$, and a full 3-torsion group is given as follows:

$$\begin{aligned} E_1[3] = \{&\mathcal{O},(2,1),(2,4),(1,t),(1,4t),(2t+1,t+1), \\ &(2t+1,4t+4),(3t+1,t+4),(3t+1,4t+1)\} \end{aligned} \tag{5.27}$$

According to the general theory, $E_1[3]$ contains 4 subgroups, and we can characterize the subgroups $\mathbb{G}_1$ and $\mathbb{G}_2$ using the Frobenius endomorphism. Unfortunately, at the time of writing, Sage does not have a predefined Frobenius endomorphism for elliptic curves, so we have to use the Frobenius endomorphism of the underlying field as a temporary workaround. We compute as follows:

```
sage: L_G1 = []
sage: for P in E1_3:
....:     PiP = E1F5_2([a.frobenius() for a in P]) # pi(P)
....:     if P == PiP:
....:         L_G1.append(P)
sage: G1 = Set(L_G1)
```

As expected, the group $\mathbb{G}_1 = \{\mathcal{O}, (2,4), (2,1)\}$ is identical to the 3-torsion group of the (unextended) curve over the prime field $E_1(\mathbb{F}_5)$. We can use almost the same algorithm to compute the group $\mathbb{G}_2$ and get the following:

```
sage: L_G2 = []
sage: for P in E1_3:
....:     PiP = E1F5_2([a.frobenius() for a in P]) # pi(P)
....:     pP = 5*P # [5]P
....:     if pP == PiP:
....:         L_G2.append(P)
sage: G2 = Set(L_G2)
```

Thus, we have computed the the second subgroup of the full 3-torsion group of curve E_1 as the set $\mathbb{G}_2 = \{\mathcal{O}, (1,t), (1,4t)\}$.

Example 89. Consider the tiny-jubjub curve *TJJ_13* from example 66. In example 86, we computed its full 5 torsion, which is a group that has 6 subgroups. We compute $G1$ using Sage as follows:

check reference

check reference

```
sage: L_TJJ_G1 = []
sage: for P in TJJF13_4_5:
....:     PiP = TJJF13_4([a.frobenius() for a in P]) # pi(P)
....:     if P == PiP:
....:         L_TJJ_G1.append(P)
sage: TJJ_G1 = Set(L_TJJ_G1)
```

We get $\mathbb{G}1 = \{\mathcal{O}, (7,2), (8,8), (8,5), (7,11)\}$

```
sage: L_TJJ_G1 = []
sage: for P in TJJF13_4_5:
....:     PiP = TJJF13_4([a.frobenius() for a in P]) # pi(P)
....:     pP = 13*P # [5]P
....:     if pP == PiP:
....:         L_TJJ_G1.append(P)
sage: TJJ_G1 = Set(L_TJJ_G1)
```

$\mathbb{G}_2 = \{\mathcal{O}, (9t^2+7, t^3+11t,), (9t^2+7, 12t^3+2t), (4t^2+7, 5t^3+10t), (4t^2+7, 8t^3+3t)\}$

Example 90. Consider Bitcoin's curve Secp256k1 again. Since the group $\mathbb{G}_1$ is identical to the torsion group of the unextended curve, and since Secp256k1 has prime order, we know that, in this case, $\mathbb{G}_1$ is identical to Secp256k1. It is however, infeasible not to compute not only $\mathbb{G}_2$ itself, but to even compute an average element of $\mathbb{G}_2$, as elements need too much storage to be representable in this universe.

The Weil pairing In this part, we consider a pairing function defined on the subgroups $\mathbb{G}_1[r]$ and $\mathbb{G}_2[r]$ of the full r-torsion $E[r]$ of a short Weierstraß elliptic curve. To be more precise, let $E(\mathbb{F}_p)$ be an elliptic curve of embedding degree k such that r is a prime factor of its order. Then the **Weil pairing** is a bilinear, non-degenerate map:

$$e(\cdot,\cdot):\mathbb{G}_1[r]\times\mathbb{G}_2[r]\to\mathbb{F}_{p^k}\,;\,(P,Q)\mapsto(-1)^r\cdot\frac{f_{r,P}(Q)}{f_{r,Q}(P)} \tag{5.28}$$

The extension field elements $f_{r,P}(Q), f_{r,Q}(P)\in\mathbb{F}_{p^k}$ are computed by **Miller's algorithm**: check floating of algorithm

Algorithm 7 Miller's algorithm for short Weierstraß curves $y^2=x^3+ax+b$

Require: $r>3$, $P\in E[r]$, $Q\in E[r]$ and
 $b_0,\ldots,b_t\in\{0,1\}$ with $r=b_0\cdot 2^0+b_1\cdot 2^1+\ldots+b_t\cdot 2^t$ and $b_t=1$
 procedure MILLER'S ALGORITHM(P,Q)
  if $P=\mathcal{O}$ or $Q=\mathcal{O}$ or $P=Q$ **then**
   return $f_{r,P}(Q)\leftarrow(-1)^r$
  end if
  $(x_T,y_T)\leftarrow(x_P,y_P)$
  $f_1\leftarrow 1$
  $f_2\leftarrow 1$
  for $j\leftarrow t-1,\ldots,0$ **do**
   $m\leftarrow\frac{3\cdot x_T^2+a}{2\cdot y_T}$
   $f_1\leftarrow f_1^2\cdot(y_Q-y_T-m\cdot(x_Q-x_T))$
   $f_2\leftarrow f_2^2\cdot(x_Q+2x_T-m^2)$
   $x_{2T}\leftarrow m^2-2x_T$
   $y_{2T}\leftarrow -y_T-m\cdot(x_{2T}-x_T)$
   $(x_T,y_T)\leftarrow(x_{2T},y_{2T})$
   if $b_j=1$ **then**
    $m\leftarrow\frac{y_T-y_P}{x_T-x_P}$
    $f_1\leftarrow f_1\cdot(y_Q-y_T-m\cdot(x_Q-x_T))$
    $f_2\leftarrow f_2\cdot(x_Q+(x_P+x_T)-m^2)$
    $x_{T+P}\leftarrow m^2-x_T-x_P$
    $y_{T+P}\leftarrow -y_T-m\cdot(x_{T+P}-x_T)$
    $(x_T,y_T)\leftarrow(x_{T+P},y_{T+P})$
   end if
  end for
  $f_1\leftarrow f_1\cdot(x_Q-x_T)$
  return $f_{r,P}(Q)\leftarrow\frac{f_1}{f_2}$
 end procedure

Understanding how the algorithm works in detail requires the concept of **divisors**, which is outside of the scope this book. The interested reader might look at XXX. add references

In real-world applications of pairing-friendly elliptic curves, the embedding degree is usually a small number like 2, 4, 6 or 12, and the number r is the largest prime factor of the curve's order.

Example 91. Consider curve $E_1(\mathbb{F}_5)$ from example 65. Since the only prime factor of the group's order is 3, we cannot compute the Weil pairing on this group using our definition of Miller's algorithm. In fact, since $\mathbb{G}_1$ is of order 3, executing the if statement on line XXX will lead to a "division by zero" error in the computation of the slope m.

check reference

add reference

Example 92. Consider the tiny-jubjub curve *TJJ_13*$(\mathbb{F}_{13})$ from example 66 again. We want to instantiate the general definition of the Weil pairing for this example. To do so, recall that, as we have see in example 82, its embedding degree is 4, and that we have the following type-3 pairing groups (where $\mathbb{G}_1$ and $\mathbb{G}_2$ are subgroups of the full 5-torsion found in the curve *TJJ_13*$(\mathbb{F}_{13^4})$):

check reference

check reference

$$\mathbb{G}_1 = \{\mathcal{O}, (7,2), (8,8), (8,5), (7,11)\}$$
$$\mathbb{G}_2 = \{\mathcal{O}, (9t^2+7, t^3+11t), (9t^2+7, 12t^3+2t), (4t^2+7, 5t^3+10t), (4t^2+7, 8t^33+3t)\}$$

The type-3 Weil pairing is a map $e(\cdot,\cdot) : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{F}_{13^4}$. From the first if-statement in Miller's algorithm, we can deduce that $e(\mathcal{O}, Q) = 1$ as well as $e(P, \mathcal{O}) = 1$ for all arguments $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$. In order to compute a non-trivial Weil pairing, we choose the arguments $P = (7,2)$ and $Q = (9t^2+7, 12t^3+2t)$.

To compute the pairing $e((7,2),(9t^2+7, 12t^3+2t))$, we have to compute the extension field elements $f_{5,P}(Q)$ and $f_{5,Q}(P)$ by applying Miller's algorithm. Do do so, observe that we have $5 = 1\cdot 2^0 + 0\cdot 2^1 + 1\cdot 2^2$, so we get $t = 2$ as well as $b_0 = 1$, $b_1 = 0$ and $b_2 = 1$. The loop therefore needs to be executed two times.

Computing $f_{5,P}(Q)$, we initiate $(x_T, y_T) = (7,2)$ as well as $f_1 = 1$ and $f_2 = 1$. Then we proceed as follows:

j	b_j	m	f_1	f_2	x_{2T}	y_{2T}	x_{T+P}	y_{T+P}
1	$\cdot$							

$$
\begin{aligned}
m &= \frac{3\cdot x_T^2 + a}{2\cdot y_T} \\
&= \frac{3\cdot 2^2+1}{2\cdot 4} = \frac{3}{3} \\
&= 1
\end{aligned}
$$

$$
\begin{aligned}
f_1 &= f_1^2\cdot(y_Q - y_T - m\cdot(x_Q - x_T)) \\
&= 1^2\cdot(t-4-1\cdot(1-2)) = t-4+1 \\
&= t+2
\end{aligned}
$$

$$
\begin{aligned}
f_2 &= f_2^2\cdot(x_Q + 2x_T - m^2) \\
&= 1^2\cdot(1+2\cdot 2-1^2) = (1+4-1) \\
&= 4
\end{aligned}
$$

$$
\begin{aligned}
x_{2T} &= m^2 - 2x_T \\
&= 1^2 - 2\cdot 2 = -3 \\
&= 2
\end{aligned}
$$

$$
\begin{aligned}
y_{2T} &= -y_T - m\cdot(x_{2T} - x_T) \\
&= -4-1\cdot(2-2) = -4 \\
&= 1
\end{aligned}
$$

We update $(x_T, y_T) = (2,1)$ and, since $b_0 = 1$, we have to execute the if statement on line XXX in the **for** loop. However, since we only loop a single time, we don't need to compute y_{T+P}, since we only need the updated x_T in the final step. We get:

add reference

should all lines of all algorithms be numbered?

$$
\begin{aligned}
m &= \frac{y_T - y_P}{x_T - x_P} \\
&= \frac{1-4}{2-x_P}
\end{aligned}
$$

$$
f_1 = f_1\cdot(y_Q - y_T - m\cdot(x_Q - x_T))
$$

$$
f_2 = f_2\cdot(x_Q + (x_P + x_T) - m^2)
$$

$$
x_{T+P} = m^2 - x_T - x_P
$$

5.3 Hashing to Curves

Elliptic curve cryptography frequently requires the ability to hash data onto elliptic curves. If the order of the curve is not a prime number, hashing to prime number subgroups is also of importance. In the context of pairing-friendly curves, it is also sometimes necessary to hash specifically onto the group $\mathbb{G}_1$ or $\mathbb{G}_2$.

As we have seen in section 4.1.2, many general methods are known for hashing into groups in general, and finite cyclic groups in particular. As elliptic groups are cyclic, those methods can be utilized in this case, too. However, in what follows we want to describe some methods specific to elliptic curves that are frequently used in real-world applications.

check reference

Try-and-increment hash functions One of the most straight-forward ways of hashing a bit-string onto an elliptic curve point in a secure way is to use a cryptographic hash function together with one of the methods we described in section 4.1.2 to hash to the modular arithmetics base field of the curve. Ideally, the hash function generates an image that is at least one bit longer than the bit representation of the base field modulus.

check reference

The image in the base field can then be interpreted as the x coordinate of the curve point, and the two possible y coordinates are derived from the curve equation, while one of the bits that exceeded the modulus determines which of the two y coordinates to choose.

Such an approach would be deterministic and easy to implement, and it would conserve the cryptographic properties of the original hash function. However, not all x coordinates generated in such a way will result in quadratic residues when inserted into the defining equation. It follows that not all field elements give rise to actual curve points. In fact, on a prime field, only half of the field elements are quadratic residues. Hence, assuming an even distribution of the hash values in the field, this method would fail to generate a curve point in about half of the attempts.

One way to account for this problem is the so-called **try-and-increment** method. Its basic assumption is that, when hashing different values, the result will eventually lead to a valid curve point.

Therefore, instead of simply hashing a string s to the field, we has the concatenation of s with additional bytes to the field instead. In other words, we use a try-and-increment hash as described in 5 is used. If the first try of hashing to the field does not result in a valid curve point, the counter is incremented, and the hashing is repeated again. This is done until a valid curve point is found.

check reference

check if the algorithm is floated properly

This method has a number of advantages: It is relatively easy to implement in code, and it maintains the cryptographic properties of the original hash function. However, it is not guaranteed to find a valid curve point, as there is a chance that all possible values in the chosen size of the counter will fail to generate a quadratic residue. Fortunately, it is possible to make the probability for this arbitrarily small by choosing large enough counters and relying on the (approximate) uniformity of the hash-to-field function.

If the curve is not of prime order, the result will be a general curve point that might not be in the "large" prime-order subgroup. In this case, a **cofactor clearing** step is then necessary to project the curve point onto the subgroup. This is done by scalar multiplication with the cofactor of prime order with respect to the curves order.

Example 93. Consider the tiny jubjub curve from example 66. We want to construct a try-and-increment hash function that hashes a binary string s of arbitrary length onto the large prime-order subgroup of size 5.

check reference

Since the curve, as well as our targeted subgroup, is defined over the field $\mathbb{F}_{13}$, and the binary representation of 13 is $13.bits() = 1101$, we apply SHA256 from Sage's hashlib library on the concatenation $s||c$ for some binary counter string, and use the first 4 bits of the image to try to hash into $\mathbb{F}_{13}$. In case we are able to hash to a value z such that $z^3 + 8 \cdot z + 8$ is a quadratic residue in $\mathbb{F}_{13}$, we use the 5-th bit to decide which of the two possible roots of $z^3 + 8 \cdot z + 8$ we

Algorithm 8 Hash-to-$E(\mathbb{F}_r)$

Require: $r \in \mathbb{Z}$ with $r.nbits() = k$ and $s \in \{0,1\}^*$
Require: Curve equation $y^2 = x^3 + ax + b$ over $\mathbb{F}_r$
  procedure TRY-AND-INCREMENT(r,k,s)
    $c \leftarrow 0$
    repeat
      $s' \leftarrow s||c\_bits()$
      $z \leftarrow H(s')_0 \cdot 2^0 + H(s')_1 \cdot 2^1 + \ldots + H(s')_k \cdot 2^k$
      $x \leftarrow z^3 + a \cdot z + b$
      $c \leftarrow c + 1$
    until $z < r$ and $x^{\frac{r-1}{2}} \bmod r = 1$
    if $H(s')_{k+1} == 0$ **then**
      $y \leftarrow \sqrt{x}$ #(root in $\mathbb{F}_r$)
    else
      $y \leftarrow r - \sqrt{x}$ #(root in $\mathbb{F}_r$)
    end if
    return (x,y)
  end procedure
Ensure: $(x,y) \in E(\mathbb{F}_r)$

will choose as the y coordinate. The result is a curve point different from the point at infinity. To project it to a point of $\mathbb{G}_1$, we multiply it with the cofactor 4. If the result is still not the point at infinity, it is the result of the hash.

To make this concrete, let $s =' 1001100111101011010000111'$ be our binary string that we want to hash onto $\mathbb{G}_1$. We use a 4-bit binary counter starting at zero, that is, we choose $c = 0000$. Invoking Sage, we define the try-hash function as follows:

```
sage: import hashlib
sage: def try_hash(s,c):
....:     s_1 = s+c
....:     hasher = hashlib.sha256(s_1.encode('utf-8'))
....:     digest = hasher.hexdigest()
....:     d = Integer(digest,base=16)
....:     sign = d.str(2)[-5:-4]
....:     d = d.str(2)[-4:]
....:     z = Integer(d,base=2)
....:     return (z,sign)
sage: try_hash('1001100111101011010000111','0000')
(15, '1')
```

As we can see, our first attempt to hash into $\mathbb{F}_{13}$ was not successful, as 15 is not a number in $\mathbb{F}_{13}$, so we increment the binary counter by 1 and try again:

```
sage: try_hash('1001100111101011010000111','0001')
(3, '0')
```

With this try, we found a hash into $\mathbb{F}_{13}$. However, this point is not guaranteed to define a curve point. To see that, we insert $z = 3$ into the right side of the Weierstraß equation of the tiny.jubjub curve, and compute $3^3 + 8 * 3 + 8 = 7$. However, 7 is not a quadratic residue in

$\mathbb{F}_{13}$, since $7^{\frac{13-1}{2}} = 7^6 = 12 = -1$. This means that 3 is a not a suitable point, and we have to increment the counter two more times:

```
sage: try_hash('100110011110101101000001111','0010')
(3, '0')
sage: try_hash('100110011110101101000001111','0011')
(6, '1')
```

Since $6^3 + 8 \cdot 6 + 8 = 12$, and we have $\sqrt{12} \in \{5, 8\}$, we finally found the valid x coordinate $x = 6$ for the curve point hash. Now, since the sign bit of this hash is 1, we choose the larger root $y = 8$ as the y coordinate and get the following hash which is a valid curve point point on the tiny jubjub curve:

$$H('100110011110101101000001111') = (6, 8)$$

In order to project this onto the "large" prime-order subgroup, we have to do cofactor clearing, that is, we have to multiply the point with the cofactor 4. We get the following:

$$[4](6, 8) = \mathcal{O}$$

This means that the hash value is still not right. We therefore have to increment the counter two more times again, until we finally find a correct hash to $\mathbb{G}_1$:

```
sage: try_hash('100110011110101101000001111','0100')
(0, '1')
sage: try_hash('100110011110101101000001111','0101')
(12, '0')
```

Since $12^3 + 8 \cdot 12 + 8 = 12$, and we have $\sqrt{12} \in \{5, 8\}$, we found another valid x coordinate $x = 12$ for the curve point hash. Since the sign bit of this hash is 0, we choose the smaller root $y = 5$ as the y coordinate, and get the following hash, which is a valid curve point point on the tiny jubjub curve:

$$H('100110011110101101000001111') = (12, 5)$$

In order to project this onto the "large" prime-order subgroup we have to do cofactor clearing, that is, we have to multiply the point with the cofactor 4. We get the following: again?

$$[4](12, 5) = (8, 5)$$

So, hashing the binary string $'100110011110101101000001111'$ onto $\mathbb{G}_1$ gives the hash value $(8, 5)$ as a result.

5.4 Constructing elliptic curves

Cryptographically secure elliptic curves like Secp256k1 from example 67 have been known for quite some time. Given the latest advancements of cryptography, however, it is often necessary to design and instantiate elliptic curves from scratch that satisfy certain very specific properties. check reference

For example, in the context of SNARK development, it was necessary to design a curve that can be efficiently implemented inside of a so-called circuit in order to enable primitives like elliptic curve signature schemes in a zero-knowledge proof. Such a curve is given by the Baby-jubjub curve (66 and we have paralleled its definition by introducing the tiny-jubjub curve from circuit signature schemes this was called "pen-jubjub" example XX. Clarify difference between baby- pen- and tiny-jubjub. As we have seen, those curves are instances of so-called twisted Edwards curves, and as such have easy to implement addition laws that work without branching. However, we introduced the tiny-jubjub curve out of thin air, as we just gave the curve parameters without explaining how we came up with them.

Another requirement in the context of many so-called **pairing-based zero-knowledge proofing systems** is the existence of a suitable, pairing-friendly curve with a specified security level and a low embedding degree as defined in 5.2.0.1. Famous examples are the BLS_12 and the NMT curves.

check reference

add references

The major goal of this section is to explain the most important method of designing elliptic curves with predefined properties from scratch, called the **complex multiplication method**. We will apply this method in section XXX to synthesize a particular BLS_6 curve, which is one of the most insecure curves, but it will serve as the main curve to build our pen-and-paper SNARKs on. As we will see, this curve has a "large" prime factor subgroup of order 13, which implies that we can use our tiny-jubjub curve to implement certain elliptic curve cryptographic primitives in circuits over that BLS_6 curve.

add reference

Before we introduce the complex multiplication method, we have to explain a few properties of elliptic curves that are of key importance in understanding the complex multiplication method.

The Trace of Frobenius To understand the complex multiplication method of elliptic curves, we have to define the so-called **trace** of an elliptic curve first.

We know from XXX that elliptic curves over finite fields are cyclic groups of finite order. Therefore, an interesting question is whether it is possible to estimate the number of elements that this curve contains. Since an affine short Weierstraß curve consists of pairs (x,y) of elements from a finite field $\mathbb{F}_q$ plus the point at infinity, and the field $\mathbb{F}_q$ contains q elements, the number of curve points cannot be arbitrarily large, since it can contain at most q^2+1 many elements.

reference text to be written in Algebra

There is however, a more precise estimation, usually called the **Hasse bound**. To understand it, let $E(\mathbb{F}_q)$ be an affine short Weierstraß curve over a finite field $\mathbb{F}_w$ of order q, and let $|E(\mathbb{F}_q)|$ be the order of the curve. Then there is an integer $t \in \mathbb{Z}$, called the **trace of Frobenius** of the curve, such that $|t| \leq 2\sqrt{q}$ and the following equation holds:

$$|E(\mathbb{F})| = q + 1 - t \tag{5.29}$$

A positive trace, therefore, implies that the curve contains less points than the underlying field, whereas a negative trace means that the curve contains more points. However, the estimation $|t| \leq 2\sqrt{q}$ implies that the difference is not very large in either direction, and the number of elements in an elliptic curve is always approximately in the same order of magnitude as the size of the curve's base field.

Example 94. Consider the elliptic curve $E_1(\mathbb{F}_5)$ from example 65. We know that it contains 9 curve points. Since the order of $\mathbb{F}_5$ is 5, we compute the trace of $E_1(\mathbb{F})$ to be $t=-3$, since the Hasse bound is given by the following equation:

check reference

$$9 = 5 + 1 - (-3)$$

Indeed, we have $|t| \leq 2\sqrt{q}$, since $\sqrt{5} > 2.23$ and $|-3| = 3 \leq 4.46 = 2 \cdot 2.23 < 2 \cdot \sqrt{5}$.

Example 95. To compute the trace of the tiny-jubjub curve, recall from example 74 that the order of *PJJ_13* is 20. Since the order of $\mathbb{F}_{13}$ is 13, we can therefore use the Hasse bound and compute the trace as $t = -6$:

check reference

$$20 = 13 + 1 - (-6) \tag{5.30}$$

Again, we have $|t| \leq 2\sqrt{q}$, since $\sqrt{13} > 3.60$ and $|-6| = 6 \leq 7.20 = 2 \cdot 3.60 < 2 \cdot \sqrt{13}$.

Example 96. To compute the trace of Secp256k1, recall from example 67 that this curve is defined over a prime field with p elements, and that the order of that group is given by r: check reference

$$p = 115792089237316195423570985008687907853269984665640564039457584007908834671663$$
$$r = 115792089237316195423570985008687907852837564279074904382605163141518161494337$$

Using the Hesse bound $r = p+1-t$, we therefore compute $t = p+1-r$, which gives the trace of curve Secp256k1 as follows:

$$t = 432420386565659656852420866390673177327$$

As we can see, Secp256k1 contains less elements than its underlying field. However, the difference is tiny, since the order of Secp256k1 is in the same order of magnitude as the order of the underlying field. Compared to p and r, t is tiny.

```
sage: p = 1157920892373161954235709850086879078532699846656405
   64039457584007908834671663
sage: r = 1157920892373161954235709850086879078528375642790749
   04382605163141518161494337
sage: t = p + 1 -r
sage: t.nbits()
129
sage: abs(RR(t)) <= 2*sqrt(RR(p))
True
```

The j-invariant As we have seen in XXX, two elliptic curves $E_1(\mathbb{F})$ defined by $y^2 = x^3 + ax + b$ and $E_2(\mathbb{F})$ defined by $y^2 + a'x + b'$ are strictly isomorphic if and only if there is a quadratic residue $d \in \mathbb{F}$ such that $a' = ad^2$ and $b' = bd^3$. add reference

There is, however, a more general way to classify elliptic curves over finite fields $\mathbb{F}_q$, based on the so-called ***j*-invariant** of an elliptic curve with $j(E(\mathbb{F}_q)) \in \mathbb{F}_q$, as defined below:

$$j(E(\mathbb{F}_q)) = (1728 \bmod q)\frac{4 \cdot a^3}{4 \cdot a^3 + (27 \bmod q) \cdot b^2} \tag{5.31}$$

A detailed description of the j-invariant is beyond the scope of this book. For our present purposes, it is sufficient to note that two elliptic curves $E_1(\mathbb{F})$ and $E_2(\mathbb{F}')$ are isomorphic over the algebraic closures of $\mathbb{F}$ and $\mathbb{F}'$, if and only if $\overline{\mathbb{F}} = \overline{\mathbb{F}'}$ and $j(E_1) = j(E_2)$. algebraic closures

So, the j-invariant is an important tool to classify elliptic curves and it is needed in the complex multiplication method to decide on an actual curve instantiation that implements abstractly chosen properties.

Example 97. Consider the elliptic curve $E_1(\mathbb{F}_5)$ from example 65. We compute its j-invariant as follows: check reference

$$\begin{aligned} j(E_1(\mathbb{F}_5)) &= (1728 \bmod 5)\frac{4 \cdot 1^3}{4 \cdot 1^3 + (27 \bmod 5) \cdot 1^2} \\ &= 3\frac{4}{4+2} \\ &= 3 \cdot 4 && = 2 \end{aligned}$$

Example 98. Consider the elliptic curve *PJJ_13* from example 66. We compute its j-invariant as follows: check reference

$$
\begin{aligned}
j(E_1(\mathbb{F}_5)) &= (1728 \bmod 13)\,\frac{4\cdot 8^3}{4\cdot 8^3+(27 \bmod 13)\cdot 8^2} \\
&= 12\cdot\frac{4\cdot 5}{4\cdot 5+1\cdot 12} \\
&= 12\cdot\frac{7}{7+12} \\
&= 12\cdot 7\cdot 6^{-1} \\
&= 12\cdot 7\cdot 11 \\
01
\end{aligned}
$$

Example 99. Consider Secp256k1 from example Secp256k1. We compute its j-invariant using Sage: check reference

```
sage: p = 115792089237316195423570985008687907853269984665640
    5640394575840079088346716 63
sage: F = GF(p)
sage: j = F(1728)*((F(4)*F(0)^3)/(F(4)*F(0)^3+F(27)*F(7)^2))
sage: j == F(0)
True
```

The Complex Multiplication Method As we have seen in the previous sections, elliptic curves have various defining properties, like their order, their prime factors, the embedding degree, or the cardinality (number of elements) of the base field. The **complex multiplication** (CM) method provides a practical way of constructing elliptic curves with pre-defined restrictions on the order and the base field.

The method usually starts by choosing a base field $\mathbb{F}_q$ of the curve $E(\mathbb{F}_q)$ we want to construct such that $q = p^m$ for some prime number p, and " $m \in \mathbb{N}$ with $m \geq 1$. We assume $p > 3$ to simplify things in what follows.

Next, the trace of Frobenius $t \in \mathbb{Z}$ of the curve is chosen such that p and t are coprime, that is, $gcd(p,t) = 0$ holds true. The choice of t also defines the curve's order r, since $r = p+1-t$ by the Hasse bound (equation 5.29), so choosing t will define the large order subgroup as well as all small cofactors. r has to be defined in such a way that the elliptic curve meets the security requirements of the application it is designed for. check reference

Note that the choice of p and t also determines the embedding degree k of any prime-order subgroup of the curve, since k is defined as the smallest number such that the prime order n divides the number $q^k - 1$.

$$
\begin{aligned}
D &< 0 \\
D \bmod 4 = 0 \text{ or } D &\bmod 4 = 1 \\
4q &= t^2 + |D|v^2
\end{aligned}
\tag{5.32}
$$

In order for the complex multiplication method to work, neither q nor t can be arbitrary, but must be chosen in such a way that two additional integers $D \in \mathbb{Z}$ and $v \in \mathbb{Z}$ exist and the following conditions hold:

If such numbers exist, we call D the **CM-discriminant**, and we know that we can construct a curve $E(\mathbb{F}_q)$ over a finite field $\mathbb{F}_q$ such that the order of the curve is $|E(\mathbb{F}_q)| = q+1-t$.

It is the content of the complex multiplication method to actually construct such a curve, that is finding the parameters a and b from $\mathbb{F}_q$ in the defining Weiertraß equation such that the curve has the desired order r.

Finding solutions to equation 5.29,= can be achieved in different ways, but we will forego the fine detail here. In general, it can be said that there are well-known constraints for elliptic curve families (e.g. the BLS (ECT) families) that provides families of solutions. In what follows, we will look at one type curve in the BLS-family, which gives an entire range of solutions.Are we looking at a subtype of BLS or is BLS the specific type we're referring to?

check reference

disambiguate

Assuming that the proper parameters q, t, D and v are found, we have to compute the so-called **Hilbert class polynomial** $H_D \in \mathbb{Z}[x]$ of the CM-discriminant D, which is a polynomial with integer coefficients. To do so, we first have to compute the following set:

$$ICG(D) = \{(A,B,C) \mid A,B,C \in \mathbb{Z}, D = B^2 - 4AC, gcd(A,B,C) = 1,$$
$$|B| \leq A \leq \sqrt{\frac{|D|}{3}}, A \leq C, \text{ if } B < 0 \text{ then } |B| < A < C\}$$

One way to compute this set is to first compute the integer $A_{max} = Floor(\sqrt{\frac{|D|}{3}})$, then loop through all the integers A in the range $[0,\ldots,A_{max}]$, as well as through all the integers B in the range $[-A_{max},\ldots,A_{max}]$, then see if there is an integer C that satisfies $D = B^2 - 4AC$ and the rest of the requirements in XXX.

add reference

To compute the Hilbert class polynomial, the so-called **j-function** (or j-invariant) is needed, which is a complex function defined on the upper half $\mathbb{H}$ of the complex plane $\mathbb{C}$, usually written as follows:is this the same as equation eq:j-invariant1?

unify terminology

$$j : \mathbb{H} \to \mathbb{C} \tag{5.33}$$

Roughly speaking, what this means is that the j-functions takes complex numbers $(x + i \cdot y)$ with a positive imaginary part $y > 0$ as inputs and returns a complex number $j(x + i \cdot y)$ as a result.

For the purposes of this book, it is not important to understand the j-function in detail, and we can use Sage to compute it in a similar way that we would use Sage to compute any other well-known function. It should be noted, however, that the computation of the j-function in Sage is sometimes prone to precision errors. For example, the j-function has a root in $\frac{-1+i\sqrt{3}}{2}$, which Sage only approximates. Therefore, when using Sage to compute the j-function, we need to take precision loss into account and possibly round to the nearest integer.

```
sage: z = ComplexField(100)(0,1)
sage: z # (0+1i)
1.0000000000000000000000000000*I
sage: elliptic_j(z)
1728.0000000000000000000000000
sage: # j-function only defined for positive imaginary
   arguments
sage: z = ComplexField(100)(1,-1)
sage: try:
....:     elliptic_j(z)
....: except PariError:
....:     pass
```

```
sage: # root at (-1+i sqrt(3))/2
sage: z = ComplexField(100)(-1,sqrt(3))/2
sage: elliptic_j(z)
-2.6445453750358706361219364880e-88
sage: elliptic_j(z).imag().round()
0
sage: elliptic_j(z).real().round()
0
```

With a way to compute the j-function and the precomputed set $ICG(D)$ at hand, we can now compute the Hilbert class polynomial as follows:

$$H_D(x) = \Pi_{(A,B,C)\in ICG(D)}\left(x - j\left(\frac{-B+\sqrt{D}}{2A}\right)\right) \tag{5.34}$$

In other words, we loop over all elements (A,B,C) from the set $ICG(D)$ and compute the j-function at the point $\frac{-B+\sqrt{D}}{2A}$, where D is the CM-discriminant that we chose in a previous step. The result defines a factor of the Hilbert class polynomial and all factors are multiplied together.

It can be shown that the Hilbert class polynomial is an integer polynomial, but actual computations need high-precision arithmetics to avoid approximation errors that usually occur in computer approximations of the j-function (as shown above). So, in case the calculated Hilbert class polynomial does not have integer coefficients, we need to round the result to the nearest integer. Given that the precision we used was high enough, the result will be correct.

In the next step, we use the Hilbert class polynomial $H_D \in \mathbb{Z}[x]$, and project it to a polynomial $H_{D,q} \in \mathbb{F}_q[x]$ with coefficients in the base field $\mathbb{F}_q$ as chosen in the first step. We do this by simply computing the new coefficients as the old coefficients modulus p, that is, if $H_D(x) = a_m x^m + a_{m-1}x^{m-1} + \ldots + a_1 x + a_0$, we compute the q-modulus of each coefficient $\tilde{a}_j = a_j \bmod p$, which defines the **projected Hilbert class polynomial** as follows:

$$H_{D,p}(x) = \tilde{a}_m x^m + \tilde{a}_{m-1}x^{m-1} + \ldots + \tilde{a}_1 x + \tilde{a}_0$$

We then search for roots of $H_{D,p}$, since every root j_0 of $H_{D,p}$ defines a family of elliptic curves over $\mathbb{F}_q$, which all have a j-invariant 5.31 or 5.33 equal to j_0. We can pick any root, since all of them will lead to proper curves eventually. check reference

However, some of the curves with the correct j-invariant might have an order different from the one we initially decided on. Therefore, we need a way to decide on a curve with the correct order.

To compute such a curve, we have to distinguish a few different cases based on our choice of the root j_0 and of the CM-discriminant D. If $j_0 \neq 0$ or $j_0 \neq 1728 \bmod q$, we compute $c_1 = \frac{j_0}{(1728 \bmod q) - j_0}$, then we chose some arbitrary quadratic non-residue $c_2 \in \mathbb{F}_q$, and some arbitrary cubic non-residue $c_3 \in \mathbb{F}_q$.

The following table is guaranteed to define a curve with the correct order $r = q + 1 - t$ for the trace of Frobenius t we initially decided on: actually make this a table?

Definition 5.4.0.1.

- Case $j_0 \neq 0$ and $j_0 \neq 1728 \bmod q$. A curve with the correct order is defined by one of the following equations:

$$y^2 = x^3 + 3c_1 x + 2c_1 \quad \text{or} \quad y^2 = x^3 + 3c_1 c_2^2 x + 2c_1 c_2^3 \tag{5.35}$$

- Case $j_0 = 0$ and $D \neq -3$. A curve with the correct order is defined by one of the following equations:
$$y^2 = x^3 + 1 \quad \text{or} \quad y^2 = x^3 + c_2^3 \tag{5.36}$$

- Case $j_0 = 0$ and $D = -3$. A curve with the correct order is defined by one of the following equations:
$$\begin{aligned} y^2 = x^3 + 1 \quad &\text{or} \quad y^2 = x^3 + c_2^3 \quad \text{or} \\ y^2 = x^3 + c_3^2 \quad &\text{or} \quad y^2 = c_3^2 c_2^3 \quad \text{or} \\ y^2 = x^3 + c_3^{-2} \quad &\text{or} \quad y^2 = x^3 + c_3^{-2} c_2^3 \end{aligned}$$

- Case $j_0 = 1728 \bmod q$ and $D \neq -4$. A curve with the correct order is defined by one of the following equations:
$$y^2 = x^3 + x \quad \text{or} \quad y^2 = x^3 + c_2^2 x \tag{5.37}$$

- Case $j_0 = 1728 \bmod q$ and $D = -4$. A curve with the correct order is defined by one of the following equations:
$$\begin{aligned} y^2 = x^3 + x \quad &\text{or} \quad y^2 = x^3 + c_2 x \quad \text{or} \\ y^2 = x^3 + c_2^2 x \quad &\text{or} \quad y^2 = x^3 + c_2^3 x \end{aligned}$$

To decide the proper defining Weierstraß equation, we therefore have to compute the order of any of the potential curves above, and then choose the one that fits our initial requirements. Since it can be shown that the Hilbert class polynomials for the CM-discriminants $D = -3$ and $D = -4$ are given by $H_{-3,q}(x) = x$ and $H_{-4,q} = x - (1728 \bmod q)$ (EXERCISE), the previous cases are exhaustive.

exercise still to be written?

To summarize, using the complex multiplication method, it is possible to synthesize elliptic curves with predefined order over predefined base fields from scratch. However, the curves that are constructed this way are just some representatives of a larger class of curves, all of which have the same order. Therefore, in real-world applications, it is sometimes more advantageous to choose a different representative from that class. To do so recall from XXX that any curve defined by the Weierstraß equation $y^2 = x^3 + axb$ is isomorphic to a curve of the form $y^2 = x^3 + ad^2x + bd^3$ for some quadratic residue $d \in \mathbb{F}_q$.

add reference

In order to find a suitable representative (e.g. with small parameters a and b) in the last step, the curve designer might choose a quadratic residue d such that the transformed curve has the properties they wanted.

Example 100. Consider curve $E_1(\mathbb{F}_5)$ from example 65. We want to use the complex multiplication method to derive that curve from scratch. Since $E_1(\mathbb{F}_5)$ is a curve of order $r = 9$ over the prime field of order $q = 5$, we know from example 94 that its trace of Frobenius is $t = -3$, which also implies that q and $|t|$ are coprime.

check reference

check reference

We then have to find parameters $D, v \in \mathbb{Z}$ such that the criteria in 5.32 hold. We get the following:

$$\begin{aligned} 4q &= t^2 + |D|v^2 && \Rightarrow \\ 20 &= (-3)^2 + |D|v^2 && \Leftrightarrow \\ 11 &= |D|v^2 \end{aligned}$$

Now, since 11 is a prime number, the only solution is $|D| = 11$ and $v = 1$ here. With $D = -11$ and the Euclidean division of -11 by 4 being $-11 = -3 \cdot 4 + 1$, we have $-11 \bmod 4 = 1$, which shows that $D = -11$ is a proper choice.

In the next step, we have to compute the Hilbert class polynomial H_{-11}. To do so, we first have to find the set $ICG(D)$. To compute that set, observe that, since $\sqrt{\frac{|D|}{3}} \approx 1.915 < 2$, we know from $A \leq \sqrt{\frac{|D|}{3}}$ and $A \in \mathbb{Z}$ that A must be either 0 or 1.

For $A = 0$, we know $B = 0$ from the constraint $|B| \leq A$. However, in this case, there could be no C satisfying $-11 = B^2 - 4AC$. So we try $A = 1$ and deduce $B \in \{-1, 0, 1\}$ from the constraint $|B| \leq A$. The case $B = -1$ can be excluded, since then $B < 0$ has to imply $|B| < A$. The case $B = 0$ can also be excluded, as there cannot be an integer C with $-11 = -4C$, since 11 is a prime number.

This leaves the case $B = 1$, and we compute $C = 3$ from the equation $-11 = 1^2 - 4C$, which gives the solution $(A, B, C) = (1, 1, 3)$:

$$ICG(D) = \{(1, 1, 3)\}$$

With the set $ICG(D)$ at hand, we can compute the Hilbert class polynomial of $D = -11$. To do so, we have to insert the term $\frac{-1+\sqrt{-11}}{2 \cdot 1}$ into the j-function. To do so, first observe that $\sqrt{-11} = i\sqrt{11}$, where i is the imaginary unit, defined by $i^2 = -1$. Using this, we can invoke Sage to compute the j-invariant and get the following:

$$H_{-11}(x) = x - j\left(\frac{-1 + i\sqrt{11}}{2}\right) = x + 32768$$

As we can see, in this particular case, the Hilbert class polynomial is a linear function with a single integer coefficient. In the next step, we have to project it onto a polynomial from $\mathbb{F}_5[x]$ by computing the modular 5 remainder of the coefficients 1 and 32768. We get $32768 \bmod 5 = 3$, from which it follows that the projected Hilbert class polynomial is considered a polynomial from $\mathbb{F}_5[x]$:

$$H_{-11,5}(x) = x + 3$$

As we can see, the only root of this polynomial is $j = 2$, since $H_{-11,5}(2) = 2 + 3 = 0$. We therefore have a situation with $j \neq 0$ and $j \neq 1728$, which tells us that we have to compute the parameter c_1 in modular 5 arithmetics:

$$c_1 = \frac{2}{1728 - 2}$$

Since $1728 \bmod 5 = 3$, we get $c_1 = 2$.

Next, we have to check if the curve $E(\mathbb{F}_5)$ defined by the Weierstraß equation $y^2 = x^3 + 3 \cdot 2x + 2 \cdot 2$ has the correct order. We invoke Sage, and find that the order is indeed 9, so it is a curve with the required parameters. Thus, we have successfully constructed the curve with the desired properties.

Note, however, that in real-world applications, it might be useful to choose parameters a and b that have certain properties, e.g. to be a small as possible. As we know from XXX, choosing any quadratic residue $d \in \mathbb{F}_5$ gives a curve of the same order defined by $y^2 = x^2 + ak^2x + bk^3$. Since 4 is a quadratic residue in $\mathbb{F}_4$, we can transform the curve defined by $y^2 = x^3 + x + 4$ into the curve $y^2 = x^3 + 4^2 + 4 \cdot 4^3$ which gives the following: add reference

$$y^2 = x^3 + x + 1$$

This is the curve $E_1(\mathbb{F}_5)$ that we used extensively throughout this book. Thus, using the complex multiplication method, we were able to derive a curve with specific properties from scratch.

Example 101. Consider the tiny jubjub curve *TJJ_13* from example 66. We want to use the complex multiplication method to derive that curve from scratch. Since *TJJ_13* is a curve of order $r = 20$ over the prime field of order $q = 13$, we know from example 95 that its trace of Frobenius is $t = -6$, which also implies that q and $|t|$ are coprime.

check reference

check reference

We then have to find parameters $D, v \in \mathbb{Z}$ such that 5.32 holds. We get the following:

$$
\begin{aligned}
4q &= t^2 + |D|v^2 && \Rightarrow \\
4 \cdot 13 &= (-6)^2 + |D|v^2 && \Rightarrow \\
52 &= 36 + |D|v^2 && \Leftrightarrow \\
16 &= |D|v^2
\end{aligned}
$$

This equation has two solutions for (D, v), namely $(-4, \pm 2)$ and $(-16, \pm 1)$. Looking at the first solution, we know that $D = -4$ implies $j = 1728$, and the constructed curve is defined by a Weierstraß equation 5.1 that has a vanishing parameter $b = 0$. We can therefore conclude that choosing $D = -4$ will not help us reconstructing *TJJ_13*. It will produce curves with order 20, just not the one we are looking for.

check reference

So we choose the second solution $D = -16$. In the next step, we have to compute the Hilbert class polynomial H_{-16}. To do so, we first have to find the set $ICG(D)$. To compute that set, observe that since $\sqrt{\frac{|-16|}{3}} \approx 2.31 < 3$, we know from $A \leq \sqrt{\frac{|-16|}{3}}$ and $A \in \mathbb{Z}$ that A must be in the range 0..2. So we loop through all possible values of A and through all possible values of B under the constraints $|B| \leq A$, and if $B < 0$ then $|B| < A$. Then we compute potential C's from $-16 = B^2 - 4AC$. We get the following two solutions for $ICG(D)$: we get

$$ICG(D) = \{(1,0,4), (2,0,2)\}$$

With the set $ICG(D)$ at hand, we can compute the Hilbert class polynomial of $D = -16$. We can invoke Sage to compute the j-invariant and get the following:

$$
\begin{aligned}
H_{-16}(x) &= \left(x - j\left(\frac{i\sqrt{16}}{2}\right)\right)\left(x - j\left(\frac{i\sqrt{16}}{4}\right)\right) \\
&= (x - 287496)(x - 1728)
\end{aligned}
$$

As we can see, in this particular case, the Hilbert class polynomial is a quadratic function with two integer coefficients. In the next step, we have to project it onto a polynomial from $\mathbb{F}_5[x]$ by computing the modular 5 remainder of the coefficients 1, 287496 and 1728. We get $287496 \bmod 13 = 1$ and $1728 \bmod 13 = 2$, which means that the projected Hilbert class polynomial is as follows:

$$H_{-11,5}(x) = (x-1)(x-12) = (x+12)(x+1)$$

This is considered a polynomial from $\mathbb{F}_5[x]$. Thus, we have two roots, namely $j = 1$ and $j = 12$. We already know that $j = 12$ is the wrong root to construct the tiny jubjub curve, since $1728 \bmod 13 = 2$, and that case is not compatible with a curve with $b \neq 0$. So we choose $j = 1$.

Another way to decide the proper root is to compute the j-invariant of the tiny-jubjub curve. We get the following:

$$\begin{aligned}
j(TJJ_13) &= 12\frac{4\cdot 8^3}{4\cdot 8^3+1\cdot 8^2}\\
&= 12\frac{4\cdot 5}{4\cdot 5+12}\\
&= 12\frac{7}{7+12}\\
&= 12\frac{7}{7+12}\\
&= 1
\end{aligned}$$

This is equal to the root $j=1$ of the Hilbert class polynomial $H_{-16,13}$ as expected. We therefore have a situation with $j\neq 0$ and $j\neq 1728$, which tells us that we have to compute the parameter c_1 in modular 5 arithmetics:

$$c_1 = \frac{1}{12-1} = 6$$

Since 1728 mod 13 = 12, we get $c_1 = 6$. Then we have to check if the curve $E(\mathbb{F}_5)$ defined by the Weierstraß equation $y^2 = x^3+3\cdot 6x+2\cdot 6$, which is equivalent to $y^2 = x^3+5x+12$, has the correct order. We invoke Sage and find that the order is 8, which implies that the trace of this curve is 6, not -6 as required. So we have to consider the second possibility, and choose some quadratic non-residue $c_2\in\mathbb{F}_{13}$. We choose $c_2 = 5$ and compute the Weierstraß equation $y^2 = x^3+5c_2^2+12c_2^3$ as follows:

$$y^2 = x^3+8x+5$$

We invoke Sage and find that the order is 20, which is indeed the correct one. As we know from XXX, choosing any quadratic residue $d\in\mathbb{F}_5$ gives a curve of the same order defined by $y^2 = x^2+ad^2x+bd^3$. Since 12 is a quadratic residue in $\mathbb{F}_{13}$, we can transform the curve defined by $y^2 = x^3+8x+5$ into the curve $y^2 = x^3+12^2\cdot 8+5\cdot 12^3$ which gives the following: add reference

$$y^2 = x^3+8x+8$$

This is the tiny jubjub curve that we used extensively throughout this book. So using the complex multiplication method, we were able to derive a curve with specific properties from scratch.

Example 102. To consider a real-world example, we want to use the complex multiplication method in combination with Sage to compute Secp256k1 from scratch. So based on example 67, we decided to compute an elliptic curve over a prime field $\mathbb{F}_p$ of order r for the following security parameters: check reference

$$p = 115792089237316195423570985008687907853269984665640564039457584007908834671663$$

$$r = 115792089237316195423570985008687907852837564279074904382605163141518161494337$$

According to example 96, this gives the following trace of Frobenius: check reference

$$t = 432420386565659656852420866390673177327$$

We also decided that we want a curve of the form $y^2 = x^3+b$, that is, we want the parameter a to be zero. This implies that the j-invariant of our curve must be zero.

In a first step, we have to find a CM-discriminant D and some integer v such that the equation $4p = t^2 + |D|v^2$ is satisfied. Since we aim for a vanishing j-invariant, the first thing to try is $D = -3$. In this case, we can compute $v^2 = (4p - t^2)$, and if v^2 happens to be an integer that has a square root v, we are done. Invoking Sage we compute as follows:

```
sage: D = -3
sage: p = 1157920892373161954235709850086879078532699846656405
    64039457584007908834671663
sage: r = 1157920892373161954235709850086879078528375642790749
    04382605163141518161494337
sage: t = p+1-r
sage: v_sqr = (4*p - t^2)/abs(D)
sage: v_sqr.is_integer()
True
sage: v = sqrt(v_sqr)
sage: v.is_integer()
True
sage: 4*p == t^2 + abs(D)*v^2
True
sage: v
303414439467246543595250775667605759171
```

The pair $(D, v) = (-3, 303414439467246543595250775667605759171)$ does indeed solve the equation, which tells us that there is a curve of order r over a prime field of order p, defined by a Weierstraß equation $y^2 = x^3 + b$ for some $b \in \mathbb{F}_p$. Now we need to compute b.

For $D = -3$, we already know that the associated Hilbert class polynomial is given by $H_{-3}(x) = x$, which gives the projected Hilbert class polynomial as $H_{-3,p} = x$ and the j-invariant of our curve is guaranteed to be $j = 0$. Now, looking at 5.4.0.1, we see that there are 6 possible cases to construct a curve with the correct order r. In order to construct the curves in question, we have to choose some arbitrary quadratic and cubic non-residue. So we loop through $\mathbb{F}_p$ to find them, invoking Sage:

check reference

```
sage: F = GF(p)
sage: for c2 in F:
....:     try: # quadratic residue
....:         _ = c2.nth_root(2)
....:     except ValueError: # quadratic non residue
....:         break
sage: c2
3
sage: for c3 in F:
....:     try:
....:         _ = c3.nth_root(3)
....:     except ValueError:
....:         break
sage: c3
2
```

We found the quadratic non-residue $c_2 = 3$ and the cubic non-residue $c_3 = 2$. Using those numbers, we check the six cases against the the expected order r of the curve we want to synthesize:

```
sage: C1 = EllipticCurve(F,[0,1])
sage: C1.order() == r
False
sage: C2 = EllipticCurve(F,[0,c2^3])
sage: C2.order() == r
False
sage: C3 = EllipticCurve(F,[0,c3^2])
sage: C3.order() == r
False
sage: C4 = EllipticCurve(F,[0,c3^2*c2^3])
sage: C4.order() == r
False
sage: C5 = EllipticCurve(F,[0,c3^(-2)])
sage: C5.order() == r
False
sage: C6 = EllipticCurve(F,[0,c3^(-2)*c2^3])
sage: C6.order() == r
True
```

As expected, we found an elliptic curve of the correct order r over a prime field of size p. In principle. we are done, as we have found a curve with the same basic properties as Secp256k1. However, the curve is defined by the following equation, which uses a very large parameter b_1, and so it might perform too slowly in certain algorithms.

$$y^2 = x^3 + 86844066927987146567678238756515930889952488499230423029593188005931626003754$$

It is also not very elegant to be written down by hand. It might therefore be advantageous to find an isomorphic curve with the smallest possible parameter b_2. In order to find such a b_2, we have to choose a quadratic residue d such that $b_2 = b_1 \cdot d^3$ is as small as possible. To do so, we rewrite the last equation into the following form:

what does this mean?

$$d = \sqrt[3]{\frac{b_2}{b_1}}$$

Then we invoke Sage to loop through values $b_2 \in \mathbb{F}_p$ until it finds some number such that the quotient $\frac{b_2}{b_1}$ has a cube root d and this cube root itself is a quadratic residue.

```
sage: b1=86844066927987146567678238756515930889952488499230423
    029593188005931626003754
sage: for b2 in F:
....:     try:
....:         d = (b2/b1).nth_root(3)
....:         try:
....:             _ = d.nth_root(2)
....:             if d != 0:
....:                 break
....:         except ValueError:
....:             pass
....:     except ValueError:
....:         pass
```

```
sage: b2
7
```

Indeed, the smallest possible value is $b_2 = 7$ and the defining Weierstraß equation of a curve over $\mathbb{F}_p$ with prime order r is $y^2 = x^3 + 7$, which we might call Secp256k1. As we have just seen, the complex multiplication method is powerful enough to derive cryptographically secure curves like Secp256k1 from scratch.

The ***BLS*6_6 pen-and-paper curve** In this paragraph, we summarize our understanding of elliptic curves to derive our main pen-and-paper example for the rest of the book. To do so, we want to use the complex multiplication method to derive a pairing-friendly elliptic curve that has similar properties to curves that are used in actual cryptographic protocols. However, we design the curve specifically to be useful in pen-and-paper examples, which mostly means that the curve should contain only a few points so that we are able to derive exhaustive addition and pairing tables.

A well-understood family of pairing-friendly curves is the the group of BLS curves (STUFF ABOUT THE HISTORY AND THE NAMING CONVENTION), which are derived in [XXX]. BLS curves are particularly useful in our case if the embedding degree k satisfies $k \equiv 6 \pmod 0$. Of course, the smallest embedding degree k that satisfies this congruency is $k = 6$ and we therefore aim for a BLS6 curve as our main pen-and-paper example.

write up this part

add reference

To apply the complex multiplication method from page 109 ff., recall that this method starts with a definition of the base field $\mathbb{F}_{p^m}$, as well as the trace of Frobenius t and the order of the curve. If the order $p^m + 1 - t$ is not a prime number, then the order r of the largest prime factor group needs to be controlled.

check reference

In the case of BLS_6 curves, the parameter m is chosen to be 1, which means that the curves are defined over prime fields. All relevant parameters p, t and r are then themselves parameterized by the following three polynomials:

$$
\begin{aligned}
r(x) &= \Phi_6(x) \\
t(x) &= x + 1 \\
q(x) &= \frac{1}{3}(x-1)^2(x^2 - x + 1) + x
\end{aligned}
\tag{5.38}
$$

In the equations above, Φ_6 is the 6-th cyclotomic polynomial and $x \in \mathbb{N}$ is a parameter that the designer has to choose in such a way that the evaluation of p, t and r at the point x gives integers that have the proper size to meet the security requirements of the curve that they want to design. It is then guaranteed that the complex multiplication method can be used in combination with those parameters to define an elliptic curve with CM-discriminant $D = -3$, embedding degree $k = 6$, and curve equation $y^2 = x^3 + b$ for some $b \in \mathbb{F}_p$.

cyclotomic polynomial

For example, if the curve should target the 128-bit security level, due to the Pholaard-rho attack (TODO) the parameter r should be prime number of at least 256 bits.

Pholaard-rho attack

todo

In order to design the smallest BLS_6 curve, we therefore have to find a parameter x such that $r(x)$, $t(x)$ and $q(x)$ are the smallest natural numbers that satisfy $q(x) > 3$ and $r(x) > 3$.[1]

We therefore initiate the design process of our *BLS*6 curve by looking up the 6-th cyclotomic polynomial, which is $\Phi_6 = x^2 - x + 1$, and then insert small values for x into the defining

[1] The smallest BLS curve will also be the most insecure BLS curve. However, since our goal with this curve is ease of pen-and-paper computation rather than security, it fits the purposes of this book.

polynomials r, t, q. We get the following results:

$$
\begin{array}{lll}
x=1 & (r(x),t(x),q(x)) & (1,2,1) \\
x=2 & (r(x),t(x),q(x)) & (3,3,3) \\
x=3 & (r(x),t(x),q(x)) & (7,4,\frac{37}{3}) \\
x=4 & (r(x),t(x),q(x)) & (13,5,43)
\end{array}
$$

Since $q(1) = 1$ is not a prime number, the first x that gives a proper curve is $x = 2$. However, such a curve would be defined over a base field of characteristic 3, and we would rather like to avoid that. We therefore find $x = 4$, which defines a curve over the prime field of characteristic 43 that has a trace of Frobenius $t = 5$ and a larger order prime group of size $r = 13$.

why?

Since the prime field $\mathbb{F}_{43}$ has 43 elements and 43's binary representation is $43_2 = 101011$, which consists of 6 digits, the name of our pen-and-paper curve should be $BLS6\_6$, since its is common to name a BLS curve by its embedding degree and the bit-length of the modulus in the base field. We call $BLS6\_6$ the **moon-math-curve**.

Based on 5.29, we know that the Hasse bound implies that $BLS6\_6$ will contain exactly 39 elements. Since the prime factorization of 39 is $39 = 3 \cdot 13$, we have a "large" prime factor group of size 13, as expected, and a small cofactor group of size 3. Fortunately, a subgroup of order 13 is well suited for our purposes, as 13 elements can be easily handled in the associated addition, scalar multiplication and pairing tables in a pen-and-paper style.

check reference

We can check that the embedding degree is indeed 6 as expected, since $k = 6$ is the smallest number k such that $r = 13$ divides $43^k - 1$.

```
sage: for k in range(1,42): # Fermat's little theorem
....:     if (43^k-1)%13 == 0:
....:         break
sage: k
6
```

In order to compute the defining equation $y^2 = x^3 + ax + b$ of BLS6-6, we use the complex multiplication method as described in 5.4. The goal is to find $a, b \in \mathbb{F}_{43}$ representations that are particularly nice to work with. The authors of XXX showed that the CM-discriminant of every BLS curve is $D = -3$ and, indeed, the following equation has the four solutions $(D, v) \in \{(-3,-7), (-3,7), (-49,-1), (-49,1)\}$ if D is required to be negative, as expected:

check reference

what does this mean?

add reference

$$
\begin{aligned}
4p &= t^2 + |D|v^2 && \Rightarrow \\
4 \cdot 43 &= 5^2 + |D|v^2 && \Rightarrow \\
172 &= 25 + |D|v^2 && \Leftrightarrow \\
49 &= |D|v^2
\end{aligned}
$$

This means that $D = -3$ is indeed a proper CM-discriminant, and we can deduce that the parameter a has to be 0, and that the Hilbert class polynomial is given by $H_{-3,43}(x) = x$.

This implies that the j-invariant of $BLS6\_6$ is given by $j(BLS6\_6) = 0$. We therefore have to look at case XXX in table 5.4.0.1 to derive a parameter b. To decide the proper case for $j_0 = 0$ and $D = -3$, we therefore have to choose some arbitrary quadratic non-residue c_2 and cubic non-residue c_3 in $\mathbb{F}_{43}$. We choose $c_2 = 5$ and $c_3 = 36$. We check these with Sage:

add reference

check reference

```
sage: F43 = GF(43)
```

```
sage: c2 = F43(5)
....: try: # quadratic residue
....:     c2.nth_root(2)
....: except ValueError: # quadratic non residue
....:     c2
sage: c3 =F43(36)
....: try:
....:     c3.nth_root(3)
....: except ValueError:
....:     c3
```

Using those numbers we check the six possible cases from 5.4.0.1 against the the expected order 39 of the curve we want to synthesize:

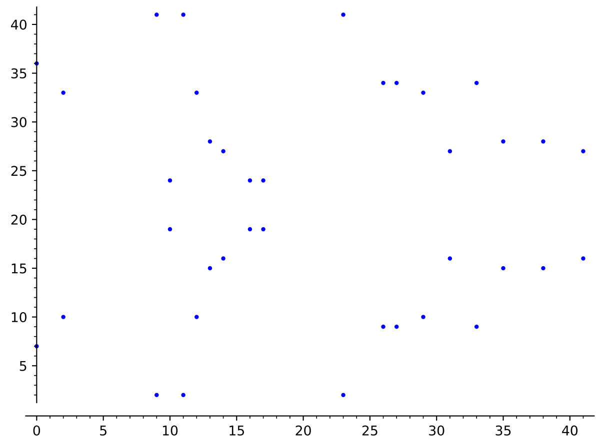
check reference

```
sage: BLS61 = EllipticCurve(F43,[0,1])
sage: BLS61.order() == 39
False
sage: BLS62 = EllipticCurve(F43,[0,c2^3])
sage: BLS62.order() == 39
False
sage: BLS63 = EllipticCurve(F43,[0,c3^2])
sage: BLS63.order() == 39
True
sage: BLS64 = EllipticCurve(F43,[0,c3^2*c2^3])
sage: BLS64.order() == 39
False
sage: BLS65 = EllipticCurve(F43,[0,c3^(-2)])
sage: BLS65.order() == 39
False
sage: BLS66 = EllipticCurve(F43,[0,c3^(-2)*c2^3])
sage: BLS66.order() == 39
False
sage: BLS6 = BLS63 # our BLS6 curve in the book
```

As expected, we found an elliptic curve of the correct order 39 over a prime field of size 43, defined by the following equation:

$$BLS6\_6 := \{(x,y) \mid y^2 = x^3 + 6 \text{ for all } x,y \in \mathbb{F}_{43}\} \tag{5.39}$$

There are other choices for b, such as $b = 10$ or $b = 23$, but all these curves are isomorphic, and hence represent the same curve in a different way. Since BLS6-6 only contains 39 points ,it is possible to give a visual impression of the curve:



As we can see, our curve has some desirable properties: it does not contain self-inverse points, that is, points with $y = 0$. It follows that the addition law can be optimized, since the branch for those cases can be eliminated.

Summarizing the previous procedure, we have used the method of Barreto, Lynn and Scott to construct a pairing-friendly elliptic curve of embedding degree 6. However, in order to do elliptic curve cryptography on this curve, note that, since the order of $BLS6\_6$ is 39, its group of rational points is not a finite cyclic group of prime order. We therefore have to find a suitable subgroup as our main target. Since $39 = 13 \cdot 3$, we know that the curve must contain a "large" prime-order group of size 13 and a small cofactor group of order 3.

add reference

The following step is to construct this group. One way to do so is to find a generator. We can achieve this by choosing an arbitrary element of the group that is not the point at infinity, and then multiply that point with the cofactor of the group's order. If the result is not the point at infinity, the result will be a generator. If it is the point at infinity we have to choose a different element.

In order to find a generator for the large order subgroup of size 13, we first notice that the cofactor of 13 is 3, since $39 = 3 \cdot 13$. We then need to construct an arbitrary element from $BLS6\_6$. To do so in a pen-and-paper style, we can choose some $arbitrary x \in \mathbb{F}_{43}$ and see if there is some solution $y \in \mathbb{F}_{43}$ that satisfies the defining Weierstraß equation $y^2 = x^3 + 6$. We choose $x = 9$, and check that $y = 2$ is a proper solution:

$$\begin{aligned}
y^2 &= x^3 + 6 && \Rightarrow \\
2^2 &= 9^3 + 6 && \Leftrightarrow \\
4 &= 4
\end{aligned}$$

This implies that $P = (9,2)$ is therefore a point on $BLS6\_6$. To see if we can project this point onto a generator of the large order prime group $BLS6\_6[13]$, we have to multiply P with the cofactor, that is, we have to compute $[3](9,2)$. After some computation (EXERCISE) we get $[3](9,2) = (13,15)$. Since this is not the point at infinity, we know that $(13,15)$ must be a generator of $BLS6\_6[13]$. The generator $g_{BLS6\_6[13]}$, which we will use in pairing computations in the remainder of this book, is given as follows:

add exercise

$$g_{BLS6\_6[13]} = (13,15) \tag{5.40}$$

Since $g_{BLS6\_6[13]}$ is a generator, we can use it to construct the subgoup $BLS6\_6[13]$ by repeatedly adding the generator to itself. Using Sage, we get the following:

```
sage: P = BLS6(9,2)
```
594

```
sage: Q = 3*P
sage: Q.xy()
(13, 15)
sage: BLS6_13 = []
sage: for x in range(0,13): # cyclic of order 13
....:     P = x*Q
....:     BLS6_13.append(P)
```

Repeatedly adding a generator to itself, as we just did, will generate small groups in logarithmic order with respect to the generator as, explained on page 43 ff. We therefore get the following description of the large prime-order subgroup of $BLS6\_6$:

check reference

$$
\begin{aligned}
BLS6\_6[13] = \\
&\{(13,15) \to (33,34) \to (38,15) \to (35,28) \to (26,34) \to (27,34) \to \\
&(27,9) \to (26,9) \to (35,15) \to (38,28) \to (33,9) \to (13,28) \to \mathcal{O}\}
\end{aligned} \tag{5.41}
$$

Having a logarithmic description of this group is tremendously helpful in pen-and-paper computations. To see that, observe that we know fromXXX that there is an exponential map from the scalar field $\mathbb{F}_{13}$ to $BLS6\_6[13]$ with respect to our generator, which generates the group in logarithmic order:

add reference

$$[\cdot]_{(13,15)} : \mathbb{F}_{13} \to BLS6\_6[13] \; ; \; x \mapsto [x](13,15)$$

So, for example, we have $[1]_{(13,15)} = (13,15)$, $[7]_{(13,15)} = (27,9)$ and $[0]_{(13,15)} = \mathcal{O}$ and so on. The relevant point here is that we can use this representation to do computations in $BLS6\_6[13]$ efficiently in our head using XXX, as in the following example:

add reference

$$
\begin{aligned}
(27,34) \oplus (33,9) &= [6](13,15) \oplus [11](13,15) \\
&= [6+11](13,15) \\
&= [4](13,15) \\
&= (35,28)
\end{aligned}
$$

So XXX is really all we need to do computations in $BLS6\_6[13]$ in this book efficiently. However, out of convenience, the following picture lists the entire addition table of that group, as it might be useful in pen-and-paper computations:

add reference

$\oplus$	$\mathcal{O}$	$(13,15)$	$(33,34)$	$(38,15)$	$(35,28)$	$(26,34)$	$(27,34)$	$(27,9)$	$(26,9)$	$(35,15)$	$(38,28)$	$(33,9)$	$(13,28)$
$\mathcal{O}$	$\mathcal{O}$	$(13,15)$	$(33,34)$	$(38,15)$	$(35,28)$	$(26,34)$	$(27,34)$	$(27,9)$	$(26,9)$	$(35,15)$	$(38,28)$	$(33,9)$	$(13,28)$
$(13,15)$	$(13,15)$	$(33,34)$	$(38,15)$	$(35,28)$	$(26,34)$	$(27,34)$	$(27,9)$	$(26,9)$	$(35,15)$	$(38,28)$	$(33,9)$	$(13,28)$	$\mathcal{O}$
$(33,34)$	$(33,34)$	$(38,15)$	$(35,28)$	$(26,34)$	$(27,34)$	$(27,9)$	$(26,9)$	$(35,15)$	$(38,28)$	$(33,9)$	$(13,28)$	$\mathcal{O}$	$(13,15)$
$(38,15)$	$(38,15)$	$(35,28)$	$(26,34)$	$(27,34)$	$(27,9)$	$(26,9)$	$(35,15)$	$(38,28)$	$(33,9)$	$(13,28)$	$\mathcal{O}$	$(13,15)$	$(33,34)$
$(35,28)$	$(35,28)$	$(26,34)$	$(27,34)$	$(27,9)$	$(26,9)$	$(35,15)$	$(38,28)$	$(33,9)$	$(13,28)$	$\mathcal{O}$	$(13,15)$	$(33,34)$	$(38,15)$
$(26,34)$	$(26,34)$	$(27,34)$	$(27,9)$	$(26,9)$	$(35,15)$	$(38,28)$	$(33,9)$	$(13,28)$	$\mathcal{O}$	$(13,15)$	$(33,34)$	$(38,15)$	$(35,28)$
$(27,34)$	$(27,34)$	$(27,9)$	$(26,9)$	$(35,15)$	$(38,28)$	$(33,9)$	$(13,28)$	$\mathcal{O}$	$(13,15)$	$(33,34)$	$(38,15)$	$(35,28)$	$(26,34)$
$(27,9)$	$(27,9)$	$(26,9)$	$(35,15)$	$(38,28)$	$(33,9)$	$(13,28)$	$\mathcal{O}$	$(13,15)$	$(33,34)$	$(38,15)$	$(35,28)$	$(26,34)$	$(27,34)$
$(26,9)$	$(26,9)$	$(35,15)$	$(38,28)$	$(33,9)$	$(13,28)$	$\mathcal{O}$	$(13,15)$	$(33,34)$	$(38,15)$	$(35,28)$	$(26,34)$	$(27,34)$	$(27,9)$
$(35,15)$	$(35,15)$	$(38,28)$	$(33,9)$	$(13,28)$	$\mathcal{O}$	$(13,15)$	$(33,34)$	$(38,15)$	$(35,28)$	$(26,34)$	$(27,34)$	$(27,9)$	$(26,9)$
$(38,28)$	$(38,28)$	$(33,9)$	$(13,28)$	$\mathcal{O}$	$(13,15)$	$(33,34)$	$(38,15)$	$(35,28)$	$(26,34)$	$(27,34)$	$(27,9)$	$(26,9)$	$(35,15)$
$(33,9)$	$(33,9)$	$(13,28)$	$\mathcal{O}$	$(13,15)$	$(33,34)$	$(38,15)$	$(35,28)$	$(26,34)$	$(27,34)$	$(27,9)$	$(26,9)$	$(35,15)$	$(38,28)$
$(13,28)$	$(13,28)$	$\mathcal{O}$	$(13,15)$	$(33,34)$	$(38,15)$	$(35,28)$	$(26,34)$	$(27,34)$	$(27,9)$	$(26,9)$	$(35,15)$	$(38,28)$	$(33,9)$

Now that we have constructed a "large" cyclic prime-order subgroup of $BLS6\_6$ suitable for many pen-and-paper computations in elliptic curve cryptography, we have to look at how to do pairings in this context. We know that $BLS6\_6$ is a pairing-friendly curve by design, since it has a small embedding degree $k = 6$. It is therefore possible to compute Weil pairings efficiently. However, in order to do so, we have to decide the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ as explained in exercise 73.

check reference

Since $BLS6\_6$ has two non-trivial subgroups, it would be possible to use any of them as the n-torsion group. However, in cryptography, the only secure choice is to use the large prime-order subgroup, which in our case is $BLS6\_6[13]$. We therefore decide to consider the 13-torsion and define $G_1[13]$ as the first argument for the Weil pairing function:

$$\mathbb{G}_1[13] = \{(13,15) \to (33,34) \to (38,15) \to (35,28) \to (26,34) \to (27,34) \to \\ (27,9) \to (26,9) \to (35,15) \to (38,28) \to (33,9) \to (13,28) \to \mathcal{O}\}$$

In order to construct the domain for the second argument, we need to construct $\mathbb{G}_2[13]$, which, according to the general theory, should be defined by those elements P of the full 13-torsion group $BLS6\_6[13]$ that are mapped to $43 \cdot P$ under the Frobenius endomorphism (equation 5.24).

check reference

To compute $\mathbb{G}_2[13]$, we therefore have to find the full 13-torsion group first. To do so, we use the technique from XXX, which tells us that the full 13-torsion can be found in the curve extension over the extension field $\mathbb{F}_{43^6}$, since the embedding degree of $BLS6\_6$ is 6:

$$BLS6\_6 := \{(x,y) \mid y^2 = x^3 + 6 \text{ for all } x,y \in \mathbb{F}_{43^6}\} \tag{5.42}$$

Thus, we have to construct $\mathbb{F}_{43^6}$, a field that contains 6321363049 elements. In order to do so, we use the procedure of XXX and start by choosing a non-reducible polynomial of degree 6 from the ring of polynomials $\mathbb{F}_{43}[t]$. We choose $p(t) = t^6 + 6$. Using Sage, we get the following:

add reference

```
sage: F43 = GF(43)
sage: F43t.<t> = F43[]
sage: p = F43t(t^6+6)
sage: p.is_irreducible()
True
sage: F43_6.<v> = GF(43^6, name='v', modulus=p)
```

Recall from XXX that elements $x \in \mathbb{F}_{43^6}$ can be seen as polynomials $a_0 + a_1 v + a_2 v^2 + \ldots + a_5 v^5$ with the usual addition and multiplication modulo $t^6 + 6$.

add reference

In order to compute $\mathbb{G}_2[13]$, we first have to extend $BLS6\_6$ to $\mathbb{F}_{43^6}$, that is, we keep the defining equation, but expand the domain from $\mathbb{F}_{43}$ to $\mathbb{F}_{43^6}$. After that, we have to find at least one element P from that curve that is not the point at infinity, is in the full 13-torsion and satisfies the identity $\pi(P) = [43]P$. We can then use this element as our generator of $\mathbb{G}_2[13]$ and construct all other elements by repeatedly adding the generator to itself.

Since $BLS6(\mathbb{F}_{43^6})$ contains 6321251664 elements, it's not a good strategy to simply loop through all elements. Fortunately, Sage has a way to loop through elements from the torsion group directly:

```
sage: BLS6 = EllipticCurve (F43_6,[0 ,6]) # curve extension
sage: INF = BLS6(0) # point at infinity
```

```
sage: for P in INF.division_points(13): # full 13-torsion
....: # PI(P) == [q]P
....:     if P.order() == 13: # exclude point at infinity
....:         PiP = BLS6([a.frobenius() for a in P])
....:         qP = 43*P
....:         if PiP == qP:
....:             break
sage: P.xy()
(7*v^2, 16*v^3)
```

We found an element from the full 13-torsion that is in the Eigenspace of the Eigenvalue 43, which implies that it is an element of $\mathbb{G}_2[13]$. As $\mathbb{G}_2[13]$ is cyclic of prime order, this element must be a generator:

$$g_{\mathbb{G}_2[13]} = (7v^2, 16v^3) \tag{5.43}$$

We can use this generator to compute $\mathbb{G}_2$ in logarithmic order with respect to $g_{\mathbb{G}_{[}13]}$. Using Sage we get the following:

```
sage: Q = BLS6(7*v^2,16*v^3)
sage: BLS6_13_2 = []
sage: for x in range(0,13):
....:     P = x*Q
....:     BLS6_13_2.append(P)
```

$$\begin{aligned}\mathbb{G}_2 = \{&(7v^2,16v^3) \to (10v^2,28v^3) \to (42v^2,16v^3) \to (37v^2,27v^3) \to \\ &(16v^2,28v^3) \to (17v^2,28v^3) \to (17v^2,15v^3) \to (16v^2,15v^3) \to \\ &(37v^2,16v^3) \to (42v^2,27v^3) \to (10v^2,15v^3) \to (7v^2,27v^3) \to \mathcal{O}\}\end{aligned}$$

Again, having a logarithmic description of $\mathbb{G}_2[13]$ is tremendously helpful in pen-and-paper computations, as it reduces complicated computation in the extended curves to modular 13 arithmetics, as in the following example:

$$\begin{aligned}(17v^2,28v^3) \oplus (10v^2,15v^2) &= [6](7v^2,16v^3) \oplus [11](7v^2,16v^3) \\ &= [6+11](7v^2,16v^3) \\ &= [4](7v^2,16v^3) \\ &= (37v^2,27v^3)\end{aligned}$$

So XXX is really all we need to do computations in $\mathbb{G}_2[13]$ in this book efficiently. add reference

To summarize the previous steps, we have found two subgroups, $\mathbb{G}_1[13]$ and $\mathbb{G}_2[13]$ suitable to do Weil pairings on *BLS*6_6 as explained in 5.28. Using the logarithmic order XXX of $\mathbb{G}_1[13]$, the logarithmic order XXX of $\mathbb{G}_2[13]$ and the bilinearity in 5.44, we can do Weil pairings on *BLS*6_6 in a pen-and-paper style: check reference add reference add reference

$$e([k_1]g_{BLS6\_6[13]},[k_2]g_{\mathbb{G}_2[13]}) = e(g_{BLS6\_6[13]},g_{\mathbb{G}_2[13]})^{k_1\cdot k_2} \tag{5.44}$$

Observe that the Weil pairing between our two generators is given by the following identity: add reference

$$e(g_{BLS6\_6[13]},g_{\mathbb{G}_2[13]}) = 5v^5+16v^4+16v^3+15v^2+3v+41$$

```
sage: g1 = BLS6([13,15])
sage: g2 = BLS6([7*v^2, 16*v^3])
sage: g1.weil_pairing(g2,13)
5*v^5 + 16*v^4 + 16*v^3 + 15*v^2 + 3*v + 41
```

Hashing to pairing groups We give various constructions to hash into $\mathbb{G}_1$ and $\mathbb{G}_2$.

We start with hashing to the scalar field... TO APPEAR

finish writing this up

None of these techniques work for hashing into $\mathbb{G}_2$. We therefore implement Pederson's Hash for BLS6.

We start with $\mathbb{G}_1$. Our goal is to define an 12-bit bounded hash function:

$$H_1 : \{0,1\}^{12} \to \mathbb{G}_1$$

Since $12 = 3 \cdot 4$ we "randomly" select 4 uniformly distributed generators $\{(38,15),(35,28),(27,34),(38,28)\}$ from $\mathbb{G}_1$ and use the pseudo-random function from XXX. Therefore, we have to choose a set of 4 randomly generated invertible elements from $\mathbb{F}_{13}$ for every generator. We choose the following:

add reference

$$\begin{array}{lcl} (38,15) & : & \{2,7,5,9\} \\ (35,28) & : & \{11,4,7,7\} \\ (27,34) & : & \{5,3,7,12\} \\ (38,28) & : & \{6,5,1,8\} \end{array}$$

Our hash function is then computed as follows:

$$\begin{aligned} H_1(x_{11},x_1,\ldots,x_0) = [2\cdot 7^{x_{11}}\cdot 5^{x_{10}}\cdot 9^{x_9}](38,15) + [11\cdot 4^{x_8}\cdot 7^{x_7}\cdot 7^{x_6}](35,28) + \\ [5\cdot 3^{x_5}\cdot 7^{x_4}\cdot 12^{x_3}](27,34) + [6\cdot 5^{x_2}\cdot 1^{x_1}\cdot 8^{x_0}](38,28) \end{aligned}$$

Note that $a^x = 1$ when $x = 0$. Hence, those terms can be omitted in the computation. In particular, the hash of the 12-bit zero string is given as follows:

correct computations

$$WRONG-ORDERING-REDO$$

$$\begin{aligned} H_1(0) = [2](38,15)+[11](35,28)+[5](27,34)+[6](38,28) = \\ (27,34)+(26,34)+(35,28)+(26,9) = (33,9)+(13,28) = (38,28) \end{aligned}$$

The hash of 011010101100 is given as follows:

fill in missing parts

$$\begin{aligned} H_1(011010101100) = WRONG-ORDERING-REDO \\ [2\cdot 7^0\cdot 5^1\cdot 9^1](38,15)+[11\cdot 4^0\cdot 7^1\cdot 7^0](35,28)+[5\cdot 3^1\cdot 7^0\cdot 12^1](27,34)+[6\cdot 5^1\cdot 1^0\cdot 8^0](38,28) = \\ [2\cdot 5\cdot 9](38,15)+[11\cdot 7](35,28)+[5\cdot 3\cdot 12](27,34)+[6\cdot 5](38,28) = \\ [12](38,15)+[12](35,28)+[11](27,34)+[4](38,28) = \\ TOAPPEAR \end{aligned}$$

We can use the same technique to define a 12-bit bounded hash function in $\mathbb{G}_2$:

$$H_2 : \{0,1\}^{12} \to \mathbb{G}_2$$

Again, we "randomly" select 4 uniformly distributed generators $\{(7v^2,16v^3),(42v^2,16v^3),(17v^2,15v^3),(10v^2,15v^3)\}$ from $\mathbb{G}_2$, and use the pseudo-random function from XXX. Therefore, we have to choose a set of 4 randomly generated invertible elements from $\mathbb{F}_{13}$ for every generator:

add reference

$$\begin{array}{lcl}
(7v^2,16v^3) & : & \{8,4,5,7\} \\
(42v^2,16v^3) & : & \{12,1,3,8\} \\
(17v^2,15v^3) & : & \{2,3,9,11\} \\
(10v^2,15v^3) & : & \{3,6,9,10\}
\end{array}$$

Our hash function is then computed like this:

$$\begin{aligned}
H_1(x_{11},x_{10},\ldots,x_0) = [8\cdot 4^{x_{11}}\cdot 5^{x_{10}}\cdot 7^{x_9}](7v^2,16v^3)+[12\cdot 1^{x_8}\cdot 3^{x_7}\cdot 8^{x_6}](42v^2,16v^3)+ \\
[2\cdot 3^{x_5}\cdot 9^{x_4}\cdot 11^{x_3}](17v^2,15v^3)+[3\cdot 6^{x_2}\cdot 9^{x_1}\cdot 10^{x_0}](10v^2,15v^3)
\end{aligned}$$

We extend this to a hash function that maps unbounded bitstrings to $\mathbb{G}_2$ by precomposing with an actual hash function like $MD5$, and feed the first 12 bits of its outcome into our previously defined hash function, with $TinyMD5_{\mathbb{G}_2}(s) = H_2(MD5(s)_0,\ldots MD5(s)_{11})$:

$$TinyMD5_{\mathbb{G}_2} : \{0,1\}^* \to \mathbb{G}_2$$

For example, since $MD5(\text{""}) =$
$0xd41d8cd98f00b204e9800998ecf8427e$, and the binary representation of the hexadecimal number $0x27e$ is 001001111110, we compute $TinyMD5_{\mathbb{G}_2}$ of the empty string as follows:

$$TinyMD5_{\mathbb{G}_2}(\text{""}) = H_2(MD5(s)_{11},\ldots MD5(s)_0) = H_2(001001111110) =$$

check equation

Bibliography

Jens Groth. On the size of pairing-based non-interactive arguments. *IACR Cryptol. ePrint Arch.*, 2016:260, 2016. URL http://eprint.iacr.org/2016/260.

P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. doi: 10.1109/SFCS.1994.365700.

David Fifield. The equivalence of the computational diffie–hellman and discrete logarithm problems in certain groups, 2012. URL https://web.stanford.edu/class/cs259c/finalpapers/dlp-cdh.pdf.

Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg. ISBN 978-3-540-46766-3. URL https://fmouhart.epheme.re/Crypto-1617/TD08.pdf.

Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. Cryptology ePrint Archive, Report 2016/492, 2016. https://ia.cr/2016/492.