NTNU – Trondheim
Norwegian University of
Science and Technology

Bilinear pairings of elliptic curves

Tjerand Aga Silde

May 2015

Bachelor's Project in Mathematical Studies
Department of Mathematical Sciences
Norwegian University of Science and Technology

Supervisor: Professor Kristian Gjøsteen

Acknowledgements

This project was written in the autumn term 2014 and spring term in 2015. It has been a very interesting subject of study, but as always, it was both a joy and a relief to complete the project. This is without doubt the most challenging material I've worked with during my undergraduate degree, but this project has also been my greatest achievement.

First of all I would like to thank Professor Kristian Gjøsteen for all the help during my work, your explanations gave me a great insight in things I would use hours to understand on my own. I would further thank Vektorprogrammet, Kodeklubben Trondheim and Spanskrøret Linjeforening for making my last three years in Trondheim amazing. In particular, I would like to thank Stein Olav Romslo, Øistein Søvik, Lars Klingenberg, Håkon Kaurel and Reidun Persdatter Ødegaard for all the time we have spent together in various settings during these years. Also, I would like to thank Dr. Geir Arne Hjelle for proofreading my project. At last, I would like to thank my girlfriend Hedda and my familiy for all the support in these busy hours combining studies and my work at Lær Kidsa Koding.

Abstract

English:
In this project we study bilinear pairings of elliptic curves. The motivation for the project is the Diffie-Hellmann Protocol for three people to communicate secret messages over an insecure channel. We look at the properties of elliptic curves in simplified Weierstrass form, how to calculate the pairings on these curves and finally we present Miller's algorithm which performs the particular calculations.

Norsk:
I dette prosjektet studerer vi bilineære paringer av elliptiske kurver. Motivasjonen for prosjektet er å studere Diffie-Hellmann Protokollen for tre personer som vil sende hemmelige bekjeder over en usikker kanal. Vi ser på egenskapene til elliptiske kurver på en forenklet Weierstrass form, hvordan vi kan kalkulere paringer på disse kurvene og til slutt presenterer vi Millers algoritme som utfører de nevnte beregningene.

Contents

1. Introduction

Bilinear pairings have been used to design ingenious protocols for sending secret messages between more than two parties, identity-based encryption and signature schemes. The goal of this project is to study bilinear pairings of elliptic curves, which is used in a three-party one-round key agreement. Also, we will implement Miller's algorithm to calculate the parings and to do an example.

The reader is supposed to have some knowledge about abstract algebra and elementary number theory, in addition to a little insight in cryptography. In particular, the reader should at least be comfortable dealing with basic group-theory and modular arithmetic.

This project is organized as follows:

Chapter 2 introduces public-key cryptography, discrete logarithms and the three-party one-round key agreement which is deduced from the Diffie-Hellman protocol. This will give a greater motivation for our further work.

Chapter 3 introduces elliptic curves, discover their properties and show that the points of elliptic curves indeed form an Abelian group.

Chapter 4 introduces pairings, in particular Weil- and Tate-pairings, and connect them to elliptic curves. We also give an alternative definition of the Weil-pairing, for which we use to calculate the pairings, and prove that they are the same. This is the main result in this text.

Chapter 5 introduces Miller's algorithm to calculate the pairings and give some elementary examples.

In the appendix the code for calculating the pairings is attached, written in Python.

2. Cryptography

This chapter gives a short introduction to public-key cryptography and motivates the study of the main subject of this paper, namely, bilinear pairing of elliptic curves.

2.1. Public-key cryptography

Public-key cryptography is an asymmetric system that requires two different keys, mathematically linked, one to encrypt and one to decrypt a secret message. The encryption-key is public knowledge and anyone can use it to encrypt a message. The decryption-key is private, that is, known to the receiver of the encrypted text and secret to everyone else. This ensures that only the receiver can read the message.

The protocols that are used in public-key cryptography are based on mathematical problems that have no efficient solutions, usually integer factorization, discrete logarithms and elliptic curve relations. The main point is that it is easy to generate a key-pair for a user, but infeasible to obtain the private key from the public key. The two main protocols used are the RSA protocol and the Diffie-Hellman protocol, for which the last one is of high importance for this text.

2.2. Discrete Logarithms

In mathematics, a discrete logarithm is an integer $k \in \{0, ..., n-1\}$ solving the equation $g^k = b$, where g and b are known elements of a finite, multiplicative group G of size n. Discrete logarithms are thus the finite-group-theoretic analogue of ordinary logarithms, which solve the same equation for real numbers g and b, where g is the base of the logarithm and b is the value whose logarithm is being taken.

There are some main differences worth noticing between discrete and ordinary logarithms. The first is that for ordinary logarithms, b grows as k grows, i.e. if you choose some k_1 and k_2 and calculate b_1 and b_2 such that $b_1 < b < b_2$, then $k_1 < k < k_2$. This is not necessary true for discrete logarithms. The second is that ordinary logarithms are injective, that is if $g^{k_1} = b$ and $g^{k_2} = b$, then $k_1 = k_2$. Discrete logarithms are injective if and only if the order n of the group G is prime.

2.3. The Discrete Logarithm Problem

The Discrete Logarithm Problem (DLP) is easy to understand, but may be hard to solve. Given a finite, multiplicative group G of prime order n generated by g, and given $b \in G$, the DLP is about to find k such that $g^k = b$. How hard this problem is depends on the choice of group. In this text we will take a closer look at additive groups of points of elliptic curves, where DLP is considered hard to solve.

No efficient classical algorithm for computing general discrete logarithms is known, however, there exist algorithms that are linear in the square root of the size of the group G. Examples are Shank's algorithm, the Index calculus algorithm and the Pohlig–Hellman algorithm - but these are not of further interest in this text.

2.4. The Diffie-Hellman Protocol

The Diffie-Hellman Protocol (DHP) is a method for two users to generate a shared secret key for which they can then exchange secret information across an insecure channel, for example the Internet. To get a better intuition for DHP, we present an analogy for mixing paint before we will take a closer look at the details.

> Suppose Alice and Bob want to agree on a secret color. Unfortunately, they don't have time to meet and discuss, and have to decide through sending paint-samples to each other by an insecure channel. First, they decide on a public color, let's say yellow, and mix one liter of that color. Secondly, they each choose a random secret color and mix two liters of their secret color. Then, they keep one liter of their secret color and mix the other with the public color. They exchange the mixtures over the public channel and when they receive the other person's mixture, they combine it with their retained secret color. The secret is the resulting color: Public + Alice's + Bob's.

It is quite obvious that this system will be safe, given that we can, in a mathematical sense, mix colors without the possibility to recognize the colors mixed. This is where discrete logarithms are highly useful. As mentioned, points of elliptic curves, usually denoted P or Q etc., construct a group under a certain operation. This can be used in the Diffie-Hellman Protocol as following:

1. We choose an additive group E of prime size n which is generated by P. Both E, n and P are public information.

2. Alice and Bob choose their own secret, random numbers $a, b \in \{2, ..., n-1\}$ and calculate, respectively, $Q = aP$ and $R = bP$ on their own.

3. Alice sends Q to Bob and Bob sends R to Alice, over the insecure channel.

4. Alice calculates $S = aR$ and Bob calculates $S = bQ$. They now share a common secret S. The only way for other people to discover S is to calculate the discrete logarithms of $Q = aP$ or $R = bP$ to find either a or b.

DHP can be illustrated as in Figure 2.1. This is a so-called two-party one-round key agreement protocol.

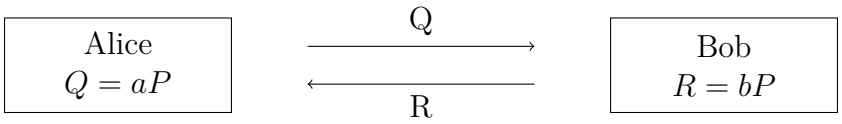

Figure 2.1.: The Diffie-Hellmann Protocol for two users.

2.5. Pairing based cryptography

It is possible to construct a three-party two-round key agreement protocol that is the natural extension of the DHP presented in the previous section. In general, it is easy to construct a n-party $(n-1)$-round key agreement protocol from DHP. But it is a lot more interesting to construct protocols that allow n people to communicate simultaneously, that is, a n-party one-round key agreement protocol. The motivation for the study of pairings in this text is the following three-party one-round key agreement protocol:

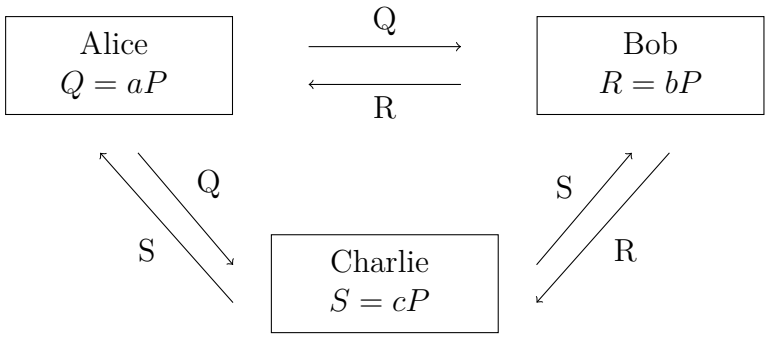

Figure 2.2.: The Diffie-Hellmann Protocol for three users.

The natural question that arises is whether it is possible to agree on a common secret from this protocol? The answer is yes, indeed it is, by the use of bilinear pairings. We need a function, say $\hat{e}$, that gives the same output K for each of the pairs of information shared in the protocol, depending on your own secret. That is,

$$K = \hat{e}(R, S)^a = \hat{e}(Q, S)^b = \hat{e}(Q, R)^c = \hat{e}(P, P)^{abc}$$

Such a function is constructed and studied in more detail in chapter 4. The bilinear Diffie-Hellmann Problem is the following: Given $\hat{e}, P, aP, bP$ and cP, compute $\hat{e}(P, P)^{abc}$.

3. Elliptic Curves

This chapter is about elliptic curves, in particular how they are defined, what properties they have and how to construct an Abelian group of the points of the elliptic curves.

3.1. Algebraic curves

We start this chapter with a crucial definition of algebraic curves.

Definition 1. *Let $\mathbb{F}$ be a field. An algebraic curve C is a polynomial equation in one or several variables, over $\mathbb{F}$, which is equal to zero. That is,*

$$C(X_1, X_2, ..., X_n) = 0.$$

There are a lot of different algebraic curves. We usually name the curves by their properties and structures, for example the degree. Some curves have their own special names, like the circle, ellipse and hyperbola, all of degree two. For our further work on more complicated curves, we need to define several important concepts.

Definition 2. *A field $\mathbb{F}$ is an algebraically closed field if it contains a root for every non-constant polynomial in $\mathbb{F}[x]$, where $\mathbb{F}[x]$ is the ring of polynomials in the variable x with coefficients in $\mathbb{F}$.*

Definition 3. *A curve $C(x_1, x_2, ..., x_n)$ in n variables is singular if there exists a point $P = (\alpha_1, ..., \alpha_n)$ on the curve such that the curve and all the partial derivatives are zero at P, that is, $C(P) = C_{x_1}(P) = ... = C_{x_n}(P) = 0$.*

Definition 4. *Let $\mathbb{F}$ be a field and let $\sim$ be an equivalence relation. The projective plane $\mathbb{P}^2$ is the set of all points, or triples, $(a, b, c) \in \mathbb{F}^3$, with $a, b, c \in \mathbb{F}$ not all zero. Define $(a, b, c) \sim (\lambda a, \lambda b, \lambda c)$ for all $\lambda \neq 0$. We write $[a, b, c]$ for the residue class of (a, b, c). Then,*

$$\mathbb{P}^2 = \frac{[a, b, c] \setminus \{(0, 0, 0)\}}{\sim}.$$

Remark 1. *In this project we study elliptic curves in two variables, hence we only care about the projective space $\mathbb{P}^2$ in the three variables X, Y and Z. Equations in a projective space are represented with capital letters.*

Let A^2 be all the points (x, y) in the plane $\mathbb{R}^2$, but think of them as a subset of $\mathbb{P}^2$ where $z = 1$, i.e. $A^2 = \{[X, Y, 1]\}$. Let a line in $\mathbb{P}^2$ be an equation $\alpha X + \beta Y + \gamma Z = 0$, for α, β, γ not all zero. Recall that two points defines a unique line, and that two lines which are not parallel intersect in a unique point. Wouldn't it be great if all lines had a point in common? We can try to construct that. The idea is to let L be a line from $[0, 0, 1]$ to $[m, 0, 1]$ and L' be a line from $[0, 0, 0]$ to $[m, 0, 1]$ for a given m. In a way, L and L' is the same line in A^2 and as $m \to \infty$ the lines L and L' are parallel in $\mathbb{P}^2$ which "meets at infinity".

Definition 5. *The points at infinity in $\mathbb{P}^2$ are defined as the points of the form $[a, b, 0]$, with a, b not both zero, which represent the intersections between parallel lines of A^2 in $\mathbb{P}^2$. Hence, the points at infinity are the points of the form $[1, m, 0]$, $\forall m$ together with the point $[0, 1, 0]$.*

Remark 2. *$\mathbb{P}^2$ is the union of the points at infinity and A^2.*

Definition 6. *A curve in a projective space is homogeneous of degree d if the powers of all variables in each term of the polynomial sums to d.*

3.2. Weierstrass equations of eliptic curves

The most interesting examples of algebraic curves, from our point of view, are the elliptic curves. These special kind of curves are very useful in modern developments of mathematics, and are defined as follows:

Definition 7. *An elliptic curve E over an algebraically closed field $\mathbb{F}$ is a projective, non-singular algebraic curve of degree three, having the homogeneous equation*

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

where the coefficients a_i are non-zero elements in $\mathbb{F}$. This equation is called the long Weierstass equation of an elliptic curve.

Remark 3. *An elliptic curve has only one point at infinity, namely $O = [0, 1, 0]$. This because no points of the form $[1, m, 0]$ solves the long Weierstrass equation.*

The long Weierstass equation is an equation with homogeneous coordinates in the projective plane $\mathbb{P}^2$. This equation is quite complicated, but by a simple substitution we can make it easier to understand. By setting $x = X/Z$ and $y = Y/Z$, we obtain the non-homogeneous equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6.$$

Further, we may assume that $\text{char}(\mathbb{F}) \neq 2, 3$. We then obtain the following simplified Weierstrass equation for elliptic curves, which is the equation we are going to study in detail. The remaining substitutions are not conducted here.

$$E : y^2 = x^3 + Ax + B, \;\; A, B \in \mathbb{F}.$$

But not all cubic curves on this form are smooth, i.e. non-singular, for arbitrary $A, B \in \mathbb{F}$. This is the topic of the next section.

3.3. Discriminant

When substituting the homogeneous equation into the simplified Weierstass equation we obtained an equation depending on A and B. Further, we have to determine which of these curves that are smooth, i.e. an elliptic curve, and later on can be used to calculate pairings. The discriminant is very useful for classifying these cubic curves.

Definition 8. *Let E be an algebraic curve represented by the simplified Weierstrass equation. Then the discriminant,denoted by Δ, is defined as $\Delta = -16(4A^3 + 27B^2)$.*

The discriminant tell us a lot about the curve, and is essential for our purpose of studying elliptic curves due to the following important result, which we will prove in detail.

Theorem 1. *A cubic curve given by the simplified Weierstrass equation is non- singular if and only if $\Delta \neq 0$.*

Proof. Let E be an elliptic curve given by the simplified Weierstrass equation. In addition to show that a cubic curve is singular if and only if the discriminant $\Delta = 0$, we also have to show that the point at infinity never is a singular point on an elliptic curve.

1. E is singular if and only if there exists a point $P = (\alpha, \beta)$ on the curve such that $E(P) = 0$ and the partial derivatives

$$E_x : 3x^2 + A = 0$$
$$E_y : 2y = 0.$$

This implies that a singular point must be on the form $P = (\alpha, 0)$, where α is both a root of $x^3 + Ax + B$ and $3x^2 + A$. Hence, α is double root of $x^3 + Ax + B$. We claim that $\Delta = 0$ if and only if this polynomial has a double root, and by proving that we are done.

Suppose α is a double root of the cubic equation $x^3 + Ax + B$, and that γ is the second root. Then,

$$\begin{aligned} x^3 + Ax + B &= (x-\alpha)^2(x-\gamma) \\ &= (x^2 - 2\alpha + \alpha^2)(x-\gamma) \\ &= x^3 - (2\alpha+\gamma)x^2 + (\alpha^2 + 2\alpha\gamma)x - \alpha^2\gamma, \end{aligned}$$

but the simplified Weierstrass equation does not contain a term of degree two. Hence, $\gamma = -2\alpha$. By substitution we get

$$\begin{aligned} x^3 + (\alpha^2 + 2\alpha\gamma)x - \alpha^2\gamma &= x^3 + (\alpha^2 + 2\alpha(-2\alpha))x - \alpha^2(-2\alpha) \\ &= x^3 - 3\alpha^2 x + 2\alpha^3. \end{aligned}$$

Inserting this into Δ, where $A = -3\alpha^2$ and $B = 2\alpha^3$, we obtain

$$\begin{aligned} \Delta &= -16(4(-3\alpha^2)^3 + 27(2\alpha^3)^2) \\ &= -16(-4 \cdot 27\alpha^6 + 27 \cdot 4\alpha^6) \\ &= 0. \end{aligned}$$

Hence, E is singular if and only if $\Delta = 0$.

2. Suppose E is an elliptic curve represented by homogeneus coordinates as the long Weierstrass equation in the projective plane $\mathbb{P}^2$, and let $O = [0, 1, 0]$ be the point at infinity. Then the partial derivative $E_Z(O) = 1 \neq 0$. Hence, E is non-singular in O.

□

3.4. Arithmetic of elliptic curves

As mentioned a couple of times earlier, the points on elliptic curves construct an Abelian group. In this section we will develop explicit algebraic formulas to calculate the sum of two points of an elliptic curve, and argue that this is indeed a binary operation for an Abelian group. Remember that E is an elliptic curve given by a Weierstrass-equation in the projective plane $\mathbb{P}^2$, together with the point of infinity, namely $O = [0, 1, 0]$. Before we can do any calculations, we define the point-addition from a geometric perspective of view in the following way.

Let P and Q be two, not necessarily distinct, points on an elliptic curve E, and let $L \in \mathbb{P}^2$ be the line connecting the two points. If $P = Q$, then L is the tangent line at P. We claim, and will later prove, that as E is a smooth cubic equation, the intersection

between L and E will always be three points, where P and Q are two of them. Further, let the third point be named R, and define the sum of the three points to be the point at infinity, namely $P + Q + R = O$. This can be visualised as in the figures, for $P \neq Q$ in Figure 3.1 and $P = Q$ in Figure 3.2.

But R is not the sum of P and Q, actually, it is the negation of the sum. The next thing we do is to draw a line $L' \in \mathbb{P}^2$ from R to O. If $R = O$, then L' is the tangent line at O. As before, the line L' intersect E at a third point, and this point is the sum of P and Q. This adding is called the composition law.

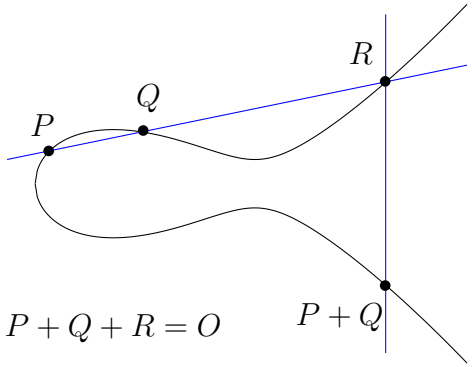

Figure 3.1.: Addition of distinct points

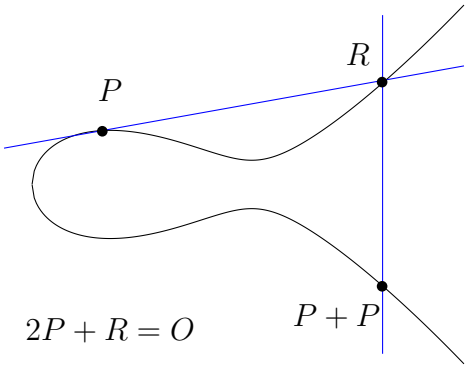

Figure 3.2.: Adding a point to itself

From the addition algorithm described, we can make the following observations:

1. (*identity*) $P + O = O + P = P$ for all $P \in E$.
2. (*commutativity*) $P + Q = Q + P$ for all $P, Q \in E$.
3. (*inverses*) Let $P \in E$, then $\exists -P \in E$ such that $P + (-P) = O$.

From this all the axioms for an Abelian group are satisfied, except associativity. For the law of associativity, we will give a geometric proof.

Theorem 2. *The law of associativity holds for points on elliptic curves.*

Proof. Let E be an elliptic curve, and let $P, Q, R \in E(\mathbb{F})$. Let $P + Q = S$ (blue) and $Q + R = T$ (red), as shown in Figure 3.3. Further, we see that both $S + R = U$ (blue) and $P + T = U$ (red). Hence, $(P + Q) + R = P + (Q + R)$ the law of associativity for addition of points on elliptic curves is true.

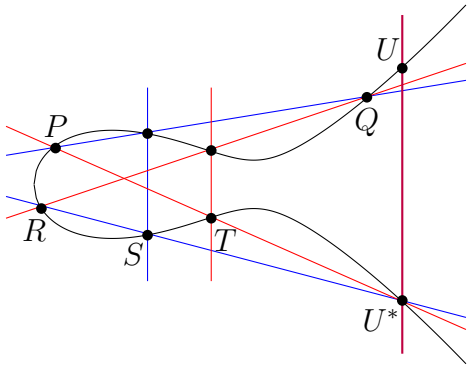

Figure 3.3.: Associativity of points on elliptic curves

□

But to calculate these additions, we need to derive explicit equations. For that we start with an elliptic curve $E : y^2 = x^3 + Ax + B$ over a finite or infinite field $\mathbb{F}$. Let $P, Q \in E(\mathbb{F})$, where $P = (x_1, y_1)$ and $Q = (x_2, y_2)$. From the definition, we already know that $P + O = O + P = P$, so we can assume that $P, Q \neq O$. We also know that if $R = (x, y)$, then $-R = (x, -y)$, as it is the reflection-point across the x-axis. We are then left with two cases:

1. $x_1 \neq x_2$ and $y_1 \neq y_2$
2. $(x_1, y_1) = (x_2, y_2)$.

In the first case, we let the line $L : y = \lambda x + \nu$ go through P and Q. In the second case we let L be the tangent to P. Inserting this into the equation of E, we get:

$$\begin{gathered} y^2 = (\lambda x + \nu)^2 = x^3 + Ax + B \\ \Updownarrow \\ x^3 + \lambda x^2 + (A - 2\lambda\nu)x + B - \nu^2 = 0. \end{gathered}$$

This equation have three solutions, namely x_1, x_2 and x_3, where it is obvious that x_3 also has to be real, since both x_1 and x_2 are real. It then follows from the coefficients in the cubic polynomial that $\lambda^2 = x_1 + x_2 + x_3$, and hence $x_3 = \lambda^2 - x_1 - x_2$. Also, it follows directly that $y_3 = \lambda(x_1 - x_3) - y_1$, remembering that y_3 is the negation of the solution lying on L. This is valid for both cases, but we need different procedures to find λ. It is easy to also find ν, but we don't need it.

1. Let $L : y = \lambda x + \nu$ be the line going through P and Q. Then, as usual, we get

$$\lambda = (y_2 - y_1) \cdot (x_2 - x_1)^{-1}.$$

2. Let $L : y = \lambda x + \nu$ be the tangent line to E going through P. Then, by implicit differentiation of E, we get

$$\begin{gathered} 2y\frac{dy}{dx} = 3x^2 + A, \\ \Updownarrow \\ \lambda = (3x^2 + A) \cdot (2y)^{-1}. \end{gathered}$$

All summed up, we have

$$\begin{aligned} \lambda &= \begin{cases} (y_2 - y_1) \cdot (x_2 - x_1)^{-1} & \text{if } P \neq Q \\ (3x^2 + A) \cdot (2y)^{-1} & \text{if } P = Q, \end{cases} \\ x_3 &= \lambda^2 - x_1 - x_2, \\ y_3 &= \lambda(x_1 - x_3) - y_1. \end{aligned}$$

3.5. Points on elliptic curves

As briefly mentioned in chapter 2, there exist no efficient algorithms to calculate discrete logarithms, and the problem gets infeasible when the groups are very large. As proved in last section, the points on elliptic curves construct an Abelian group over addition. It is naturally of great interest to know how many elements there are in a certain group, depending on A, B and the size of the finite field, say q. This is also a great effort to calculate, but a theorem by Helmut Hasse give a great estimate, determining both upper and lower bounds.

Theorem 3. *Hasse's Theorem. Let $q = p^n$ be a power of a prime and let E be an elliptic curve defined over a finite field $\mathbb{F}_q$. Then*

$$|\#E(\mathbb{F}_q) - q - 1| \leq 2\sqrt{q}.$$

Before we can prove this theorem, we need to recall some important definitions and results from abstract algebra.

Definition 9. *Let E be an elliptic curve. Then an endomorphism ϕ of E is a homomorphism $\phi : E \to E$ such that $\phi(O) = O$.*

Remark 4. *Let $q = p^n$ be a power of a prime and let E be an elliptic curve defined over a finite field $\mathbb{F}_q$. Then the set of all endomorphisms of E form a ring under function addition and composition, that is, for $P \in E(\mathbb{F}_q)$ and endomorphisms ϕ and φ of E,*

$$(\phi + \varphi)(P) = \phi(P) + \varphi(P)$$
$$(\phi \circ \varphi)(P) = \phi(\varphi(P)).$$

The trivial map is the multiplicative identity of the endomorphism ring.

Definition 10. *Assume that an endomorphism is given by an irreducible polynomial, then we define the degree of an endomorphism as the degree of the polynomial. Since the degree of a polynomial is multiplicative, so is the degree of an endomorphism. That is, for endomorphisms ϕ and φ,*

$$\deg(\phi \circ \varphi) = \deg(\phi)\deg(\varphi).$$

Result 1. *Let E be an elliptic curve over $\mathbb{F}$ and let $P \in E$. The map $\phi_m : E \to E$ given by $\phi_m(P) = mP$, denoted $[m]$, is an endomorphism of E with degree m^2.*

Definition 11. *Let $q = p^n$ be a power of a prime and let E be an elliptic curve defined over a finite field $\mathbb{F}_q$. Then the map $\phi : E \to E$, given by raising each coefficient of E to the q^{th} power, is called the q^{th}-power Frobenius endomorphism and $\deg(\phi) = q$.*

Result 2. *Let ϕ and φ be endomorphisms of an elliptic curve E. Then,*

$$|\deg(\phi - \varphi) - \deg(\phi) - \deg(\varphi)| \leq 2\sqrt{\deg(\phi)\deg(\varphi)}.$$

We are now ready to prove Hasse's theorem.

Proof. Hasse's theorem. Let $q = p^n$ be a power of a prime and let E be an elliptic curve defined over a finite field $\mathbb{F}_q$. Consider the q^{th}-power Frobenius endomorphism $\phi : E \to E$. For each point $P \in E(\mathbb{F}_q)$, we have by Fermat's little theorem $x^q \equiv x \mod q$ that $\phi(P) = P$. Thus,$\phi(P) - P = 0$ and P is in the kernel of $(\phi - 1)$. We can then conclude that E is isomorphic to the kernel of the map $(\phi - 1)$, hence,

$$\#E(\mathbb{F}_q) = \#\ker(\phi - 1) = \deg(\phi - 1).$$

By the previous result, we note that

$$|\deg(\phi - 1) - \deg(\phi) - \deg(1)| \leq 2\sqrt{\deg(\phi)\deg(1)}$$

for $\deg(\phi - 1) = \#E(\mathbb{F}_q)$, $\deg(\phi) = q$ and $\deg(1) = 1$. Hence,

$$|\#E(\mathbb{F}_q) - q - 1| \leq 2\sqrt{q}.$$

□

It is also of great importance to know the structure of a group for which you will compute discrete logarithms, especially if the group is cyclic or not. The following theorem gives a considerable amount of information about the group structure of an elliptic curve E.

Result 3. *Let E be an elliptic curve defined over $\mathbb{F}_p$, where $p > 3$ is prime. Then there exist positive integers n_1 and n_2 such that $E(\mathbb{F}_p)$ is isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$. Further, $n_2 | n_1$ and $n_2 | (p-1)$.*

One of the most efficient algorithms to calculate the points of an elliptic curve is Schoof's algorithm, that computes $\#E(\mathbb{F}_q)$ in $\Theta((\log q)^8)$ steps.

3.6. Supersingular curves

Hasse's theorem tell us that $\#E(\mathbb{F}_q) = q + 1 - t$, where $|t| \leq 2\sqrt{q}$. For calculating pairings on elliptic curves we want curves with a nice structure, that makes it easy to compute the pairings. On the other hand, we must use curves that still makes it hard to calculate discrete logarithms. That ensures that the cryptographic protocol still is secure. The curves of choice are supersingular elliptic curves, which must not be confused with the singular curves. Supersingular elliptic curves are smooth curves, and are defined in the following way.

Definition 12. *Let $q = p^n$, where $p > 3$ is prime and let E be an elliptic curve over a field $\mathbb{F}_q$ with characteristic p. Then E is a supersingular elliptic curve if $\#E(\mathbb{F}_q) \equiv 1 \mod p$.*

An equivalent way to define supersingular elliptic curves is to say that p divides t. In fact, if p divides t, then t is usually, but not necessary, equal to zero. Note that if $q = p$, then E is supersingular if and only if E has exactly $p + 1$ points. This follows from the boundaries given by Hasse's theorem. A more general result, given without proof, about the number of points of supersingular elliptic curves is:

Result 4. *Let E be a supersingular elliptic curve over the field $\mathbb{F}_p$ where $p > 3$ is prime and $n \geq 1$, then*

$$\#E(\mathbb{F}_{p^n}) = \begin{cases} p^n + 1 & \text{if } x \text{ is odd} \\ (p^{n/2} - (-1)^{n/2})^2 & \text{if } x \text{ is even.} \end{cases}$$

Counting points unfortunately take a great effort, but the following result gives two alternative ways to determine if an elliptic curve is supersingular or not.

Result 5. *Let $\mathbb{F}_q$ be a finite field of characteristic $p \geq 3$.*

1. *Let E be an elliptic curve given by a Weierstrass equation*

$$E : y^2 = f(x),$$

where $f(x) \in \mathbb{F}_q[x]$ is a cubic polynomial with distinct roots in the algebraic closure of $\mathbb{F}_q$. Then E is supersingular if and only if the coefficient of x^{p-1} in $(f(x))^{p-1}$ is zero.

2. *Let $m = (p-1)/2$ and define the polynomial*

$$H_p(t) = \sum_{i=0}^{m} \binom{m}{i}^2 t^i.$$

Let $\lambda \in \overline{\mathbb{F}_q}$ with $\lambda \neq 0, 1$. Then the elliptic curve

$$E : y^2 = x(x-1)(x-\lambda)$$

is supersingular if and only if $H_p(\lambda) = 0$.

Remark 5. *Every elliptic curve is isomorphic to a curve on the form $E : y^2 = x(x-1)(x-\lambda)$ for $\lambda \neq 0, 1$. Hence, 2. holds for every elliptic curve.*

Example 1. *The following are examples of supersingular elliptic curves with $\#E(\mathbb{F}_p) = p+1$ and $0 \neq A, B \in \mathbb{F}_p$:*

1. *$E : y^2 = x^3 - Ax$, for $p \equiv 3 \mod 4$*
2. *$E : y^2 = x^3 + B$, for $p \equiv 2 \mod 3$*

4. Bilinear pairings

This chapter is about bilinear pairings, how they are defined and what properties they have. In particular, we will take a look at the Weil Pairing, give two definitions, and prove that it is the same pairing.

4.1. Pairings

Definition 13. *Let n be a prime number. Let G_1 be an additive group of points of an elliptic curve E over a field $\mathbb{F}$ of order n and with identity O. Let G_2 be a multiplicatively-written group of order n with identity 1. A bilinear pairing $\hat{e}$ on (G_1, G_2) is a map*

$$\hat{e} : G_1 \times G_1 \to G_2,$$

that satisfies the following conditions.

1. *(bilinear) For all $R, S, T \in G_1$,*

$$\hat{e}(R+S,T) = \hat{e}(R,T)\hat{e}(S,T)$$
$$\hat{e}(R,S+T) = \hat{e}(R,S)\hat{e}(R,T).$$

2. *(non-degenerate) $\hat{e}(R,S) \neq 1$ for some $R, S \in G_1$.*
3. *(alternating) For all $R \in G_1$, $\hat{e}(R,R) = 1$, and further $\hat{e}(S,T) = \hat{e}(T,S)^{-1}$.*
4. *(compatible) For all $S \in E[nn']$, $T \in E[n]$ and $n, n' \in \mathbb{Z}$, we have $e_{nn'}(S,T) = e_n([n']S,T)$.*

From this definition of a bilinear pairing, there is a lot of properties which can be easily verified and will give us more insight in what kind of map this is.

Result 6. *For all $S, T \in G_1$.*

1. *$\hat{e}(aS,bT) = \hat{e}(S,T)^{ab}, \forall a, b \in \mathbb{Z}$.*
2. *$\hat{e}(S,O) = \hat{e}(O,S) = 1$.*
3. *$\hat{e}(S,-T) = \hat{e}(-S,T) = \hat{e}(S,T)^{-1}$.*
4. *If $\hat{e}(S,R) = 1, \forall R \in G_1$, then $S = O$.*

4.2. Divisors

To define the Weil Pairing it is necessary to take a closer look at divisors and divisor groups. This is also of great importance for the next chapter, where we have the intention to calculate the pairings with Miller's algorithm, using divisors.

Definition 14. *Let E be an elliptic curve defined over a field $\mathbb{F}$. Then the divisor group of E, denoted* $\operatorname{div}(E)$*, is an Abelian group generated by the points P of E. Thus, a divisor $D \in \operatorname{div}(E)$ is a formal sum*

$$D = \sum_{P \in E(\mathbb{F})} n_P(P),$$

where $n_P \in \mathbb{Z}$ and $n_P = 0$ for all but finitely many $P \in E(\mathbb{F})$.

Definition 15. *The degree* $\deg D$ *of a divisor D is the coefficients n_P of $D = \sum_{P \in E(\mathbb{F})} n_P(P)$.*

Result 7. *A divisor $D = \sum_P n_P(P)$ is principal if $\deg D = 0$ and $\sum_P n_P(P) = O$.*

Result 8. *Let E be an elliptic curve over a field $\mathbb{F}$ and let $P, Q \in E(\mathbb{F})$. Then $(P) \sim (Q)$ if and only if $P = Q$.*

Result 9. *Let E be an elliptic curve over a field $\mathbb{F}$ and let D be a divisor in* $\operatorname{div}(E)$*. Then there exist a unique point $P \in E(\mathbb{F})$ satisfying $D \sim (P) - (O)$. Define $\tau : D_P \to P$, that is, τ send each divisor D_P in* $\operatorname{div}(E)$ *to its associated point P.*

Result 10. *Let P be a point of an elliptic curve, f_P a function, and $D_P = (P) - (O)$ a divisor such that* $\operatorname{div}(f_P) = D_P$*. Then $f_P(D_P) = f_P(P)/f_P(O)$.*

4.3. The Weil Pairing

Finally, we are going to define the Weil Pairing. It was introduced by André Weil in 1940, who gave an abstract definition. Let E be an elliptic curve over $\mathbb{F}$ and let $E[n] = \{P \in E(\mathbb{F}) : nP = O\}$, that is, the set of n-torsion points of $E(\mathbb{F})$. The Weil Pairing is a map $\hat{e}_n : E[n] \times E[n] \to \mu$, where μ is the set of n-th roots of unity. The Weil Pairing is a bilinear pairing satisfying the usual properties, that is, the pairing is bilinear, non-degenerate, alternating and compatible.

Remark 6. *Let $n \geq 2$ be an integer. Let $p = char(\mathbb{F})$. Then the group of n-torsion points $E[n]$ have the form $E[n] \simeq \mathbb{Z}/n\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$.*

In the literature, there are in fact at least three equivalent definitions of the Weil Pairing. In the following, we will give two definitions, like they do in Silverman[1]. Further, we will prove the four properties of the Weil Pairing for the first definition and show equivalence of the two definitions.

4.4. First definition of the Weil Pairing

The map described in the Weil Pairing is determined by some special divisors, so we need to construct a function g that we use in the definition of the pairing.

Let $E[n]$ be the group of n-torsion points of an elliptic curve $E(\mathbb{F})$ and let $P \in E[n]$. Define $\text{div}(f) = \sum n_i(P_i)$ to be a divisor of a function f if and only if both $\sum n_i = 0$ and $\sum [n_i](P_i) = O$. Let $T \in E[n]$, then

$$\text{div}(f) = (T) - (O).$$

Further, take $T' \in E$ such that $[n]T' = T$. Then there exists a function g such that

$$\text{div}(g) = [n](T) - [n](O).$$

Then, the functions f and g must have the same divisor, so we can assume that $f \circ [n] = g^n$. Finally, let $S \in E[n]$ and $X \in E$, then we have

$$g(X+S)^n = f([n]X + [n]S) = f([n]X) = g(X)^n.$$

Remark 7. *Note that the function $g(X+S)/g(X)$ only takes on only finitely many values. That is, for every $X \in E(\mathbb{F})$, there is a n^{th}-root of unity. In particular, the map $S \to g(X+S)/g(X)$ is constant.*

Definition 16. *Let $E(\mathbb{F})$ be an elliptic curve over the field $\mathbb{F}$ with $p = char(E)$. Let n be an integer such that $gcd(p, n) = 1$ and let $E[n]$ be the group of n-torsion points on E. Let μ_n denote the group of n^{th} roots of unity. Then the first definition of the Weil-pairing $\hat{e}_n$ is a map*

$$\hat{e}_n : E[n] \times E[n] \to \mu_n$$

given by

$$\hat{e}_n(S, T) = \frac{g(X+S)}{g(X)}$$

for all $X \in E$, for all $S, T \in E[n]$ and g defined as above.

Proof. 1. (*bilinear*) First, we check for linearity.

$$\begin{aligned}\hat{e}_n(S_1+S_2,T) &= \frac{g(X+S_1+S_2}{g(X))}\\ &= \frac{g(X+S_1+S_2)}{g(X+S_1)}\frac{g(X+S_1)}{g(X)}\\ &= \hat{e}_n(S_2,T)\hat{e}_n(S_1,T).\end{aligned}$$

Further, bilinearity follows if we let $f_1, f_2, f_3, g_1, g_2, g_3$ be functions and let $T_1, T_2, T_3 = T_1 + T_2$ be points in $E(\mathbb{F})$ such that

$$\begin{aligned}\operatorname{div}(h) &= (T_1+T_2)-(T_1)-(T_2)+(O)\\ \operatorname{div}(\frac{f_3}{f_1f_2}) &= n\cdot\operatorname{div}(h)\\ f_3 &= c\cdot f_1f_2h^n\\ g_3 &= c'\cdot g_1g_2(h\circ[n]).\end{aligned}$$

It then follows that

$$\begin{aligned}\hat{e}_n(S,T_1+T_2) &= \frac{g_3(X+S)}{g_3(X)}\\ &= \frac{g_1(X+S)g_2(X+S)h([n]X+[n]S)}{g_1(X)g_2(X)h([n]X)}\\ &= \hat{e}_n(S,T_1)\hat{e}_n(S,T_2).\end{aligned}$$

2. (*non-degenerate*) If $\hat{e}_n(S,T)=1$ for all $S\in E[n]$, then $g(X+S)=g(X)$ for all $S\in E[n]$. This tell us that $g=h\circ[n]$ for some function h. But then

$$\begin{gathered}g^n=(h\circ[n])=f\circ[n]\\ \Downarrow\\ n\cdot div(h)=div(f)=n(T)-n(O)\\ \Downarrow\\ div(h)=(T)-(O)\\ \Downarrow\\ T=O\end{gathered}$$

3. (*alternating*) We have from bilinearity that

$$\hat{e}_n(S+T,S+T) = \hat{e}_n(S,S)\hat{e}_n(S,T)\hat{e}_n(T,S)\hat{e}_n(T,T).$$

Hence, it is enough to show that $\hat{e}_n(T,T) = 1$ for all $T \in E[n]$. Recall that

$$\mathrm{div}(\prod_{i=0}^{n-1} f \circ \tau_{[i]T}) = n\sum_{i=0}^{n-1}([1-i]T) - ([-i]T) = 0$$

and

$$\begin{aligned}\prod_{i=0}^{n-1} f \circ \tau_{[i]T} &= k_1\\ \prod_{i=0}^{n-1} g \circ \tau_{[i]T'} &= k_2.\end{aligned}$$

Then,

$$\begin{aligned}\prod_{i=0}^{n-1} g(X+[i]T') = \prod_{i=0}^{n-1} g(X+[i+1]T)\\ \Downarrow\\ g(X) = g(X+[n]T') = g(X+T).\end{aligned}$$

It follows that

$$\hat{e}_n(T,T) = \frac{g(X+T)}{g(X)} = 1.$$

4. (*compatible*) Given f and g such that $\mathrm{div}(f) = n(T) - n(O)$ and $\mathrm{div}(g) = [n](T) - [n](O)$, we have the two following equations

$$\begin{aligned}\mathrm{div}(f^{n'}) &= nn'(T) - nn'(O)\\ (g\circ[n'])^{nn'} &= (f\circ[nn'])^{n'}\end{aligned}$$

Then, directly from the definition of $\hat{e}_{nn'}$ and $\hat{e}_n$, we get

$$\hat{e}_{nn'}(S,T) = \frac{g\circ[n'](X+S)}{g\circ[n'](X)} = \frac{g(Y+[n']S)}{g(Y)} = \hat{e}_n([n']S,T)$$

□

4.5. Second definition of the Weil Pairing

Definition 17. *Let $E(\mathbb{F})$ be an elliptic curve over the field $\mathbb{F}$ with $p = char(E)$. Let n be an integer such that $gcd(p, n) = 1$ and let $E[n]$ be the group of n-torsion points on E. Let μ_n denote the group of n^{th} roots of unity. For $P, Q \in E[n], P \neq Q$, let f_P and f_Q be functions such that $\mathrm{div}(f_P) = nD_P = n(P) - n(O)$ and $\mathrm{div}(f_Q) = nD_Q = n(Q) - n(O)$, both D_P and D_P divisors of degree zero. Then the second definition of the Weil-pairing $\hat{e}_n$ is defined as a map*

$$\hat{e}_n : E[n] \times E[n] \to \mu_n$$

given by

$$\hat{e}_n(P, Q) = \frac{f_P(D_Q)}{f_Q(D_P)} = \frac{f_P(Q)}{f_Q(P)}.$$

If $P = Q$ or one or both of P and Q being O, the definition is completed by noticing that $\hat{e}_n(P, Q) = 1$ in all cases.

Theorem 4. *The two definitions of the Weil Paring is equivalent.*

Proof. This proof follows the steps from Silverman[1] from problem 16(c) at page 462.

Let the pairing be defined as above. Choose points $P', Q', R \in E$ satisfying

$$[n]P' = P, [n]Q' = Q, P \neq \pm Q, [2]R = P' - Q'.$$

Define the divisors D_P and D_Q as above and choose functions f_P, f_Q, g_P, g_Q satisfying

$$\begin{aligned} D_P &= (P) - (O), & D_Q &= (Q + [n]R) - ([n]R), \\ \mathrm{div}(f_P) &= nD_P, & \mathrm{div}(f_Q) &= nD_Q, \\ f_P \circ [n] &= g_P^n, & f_Q \circ [n] &= g_Q^n. \end{aligned}$$

As $\mathrm{div}(g_P) = \sum_{T \in E[n]} (T + P') - (T)$ and $\mathrm{div}(g_Q) = \sum_{T \in E[n]} (T + Q + R) - (T + R)$, we see that both

$$\frac{g_P(X + Q' + R) g_Q(X)}{g_P(X + R) g_Q(X + P')}$$

and

$$\prod_{i=0}^{n-1} g_Q(X + [i]Q')$$

has divisors equal to zero when viewed as functions of X. Let the second definition of the Weil Pairing be denoted as just e. Then, the following calculations show that the two definitions are equal.

$$\begin{aligned}
e_n(P,Q) &= \frac{f_P(D_Q)}{f_Q(D_P)} \\
&= \frac{f_P(Q+[n]R)f_Q(O)}{f_P([n]R)f_Q(P)} \\
&= \frac{f_P([n]Q'+[n]R)f_Q([n]O)}{f_P([n]R)f_Q([n]P')} \\
&= \left(\frac{g_P(Q+R)g_Q(O)}{g_P(R)g_Q(P')}\right)^m \\
&= \prod_{i=0}^{n-1} \frac{g_P(R+[i+1]Q')g_Q([i]Q')}{g_P(R+[i]Q')g_Q(P'+[i]Q')} \\
&= \frac{g_P(R+[n]Q')}{g_P(R)} \prod_{i=0}^{n-1} \frac{g_Q([i]Q')}{g_Q(P'+[i]Q')} \\
&= \frac{g_P(R+Q)}{g_P(R)} \\
&= \hat{e}_n(P,Q)
\end{aligned}$$

□

5. Calculating Pairings

The previous chapter presented the theory behind pairings. The Weil-pairing was defined, and its properties were proved. But the pairing is very abstract and difficult to actually calculate. In this chapter we present Miller's algorithm, which is a double-and-add-algorithm that constructs the divisor-polynomials for elliptic curves that we use to calculate the Weil-pairing.

5.1. Miller's algorithm

In this chapter we let E be an elliptic curve given by the simplified Weierstrass equation

$$E : y^2 = x^3 + Ax + B$$

where $A, B \in \mathbb{F}_p$. Also, let $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ be two non-zero points on E. We then have the following theorems describing how to calculate the Weil-pairing.

Theorem 5. *Let λ be the slope of the line connecting P and Q, or the slope of the tangent line to E at P if $P = Q$. If the line is vertical, set $\lambda = \infty$. Define the function $h_{P,Q}$ on E as follows:*

$$h_{P,Q} = \begin{cases} \frac{y - y_P - \lambda(x - x_P)}{x + x_P + x_Q - \lambda^2} & \text{if } \lambda \neq \infty \\ x - x_P \text{ if } \lambda = \infty. \end{cases}$$

This polynomial has divisor

$$\operatorname{div}(h_{P,Q}) = (P) + (Q) - (P + Q) - (O).$$

Proof. Suppose first that $\lambda \neq \infty$, and let $y = \lambda x + v$ be the line through P and Q, or the tangent line at P if $P = Q$. This line intersects E at the three points P, Q and $S = -P - Q$, so

$$\operatorname{div}(y - \lambda x - v) = (P) + (Q) + (S) - 3(O).$$

Vertical lines intersect E at points and their negatives, hence

$$\mathrm{div}(x - x_{P+Q}) = (P+Q) + (S) - 2(O).$$

It follows that

$$h_{P,Q} = \frac{y - \lambda x - v}{x + x_{P+Q}}$$

has the stated divisor. Finally, the addition formula tell us that $x_{P+Q} = \lambda^2 - x_P - x_Q$, and we can eliminate v from the numerator of $h_{P,Q}$ using $y_P = \lambda x_P + v$.

If $\lambda = \infty$, then $P + Q = O$, so we need $\mathrm{div}(h_{P,Q}) = (P) + (-P) - 2(O)$. The function $x - x_P$ has this divisor. □

Theorem 6. ***Miller's algorithm.*** *Let $N \geq 1$ be an integer, with $N = \sum_{i=0}^{t} \epsilon_i \cdot 2^i$ as the binary representation of N. That is, $\epsilon_i \in \{0, 1\}$ and $\epsilon_t \neq 0$. Then Millers algorithm, described below, calculates the function f_P, whose divisor satisfies*

$$\mathrm{div}(f_P) = N(P) - ([N]P) - (N-1)(O).$$

Proof. By analysing the divisors of $h_{T,T}$ and $h_{T,P}$, we get

$$\begin{aligned}\mathrm{div}(h_{T,T}) &= 2(T) - (2T) - (O),\\ \mathrm{div}(h_{T,P}) &= (T) + (P) - (T+P) - (O).\end{aligned}$$

Now, consider the effect of executing the $t-1$ steps in the loop, see Algorithm 1. Let T_i^{start} and f_i^{start} be the initial values and let T_i^{end} and f_i^{end} be the final values for the i-th iteration. Let us, for simplicity, just consider one iteration. Then,

$$\begin{aligned}T_i^{\mathrm{end}} &= 2T_i^{\mathrm{start}} + \epsilon_i P,\\ f_i^{\mathrm{end}} &= (f_i^{\mathrm{start}})^2 \cdot h_{T_i^{\mathrm{start}}, T_i^{\mathrm{start}}} \cdot h_{2T_i^{\mathrm{start}}, P}^{\epsilon_i}.\end{aligned}$$

Hence,

$$\begin{aligned}\mathrm{div}(f_i^{\mathrm{end}}) &= 2 \cdot \mathrm{div}(f_i^{\mathrm{start}}) + \mathrm{div}(h_{T_i^{\mathrm{start}}, T_i^{\mathrm{start}}}) + \epsilon_i \cdot \mathrm{div}(h_{2T_i^{\mathrm{start}}, P})\\ &= 2 \cdot \mathrm{div}(f_i^{\mathrm{start}}) + (2(T_i^{\mathrm{start}}) - (2T_i^{\mathrm{start}}) - (O)) + \epsilon_i((2T_i^{\mathrm{start}})\\ &\quad + (P) - (2T_i^{\mathrm{start}} + P) - (O))\\ &= 2 \cdot \mathrm{div}(f_i^{\mathrm{start}}) + 2(T_i^{\mathrm{start}}) - (2T_i^{\mathrm{start}} + \epsilon_i P) + \epsilon_i(P) - (1+\epsilon_i)(O)\\ &= 2 \cdot \mathrm{div}(f_i^{\mathrm{start}}) + 2(T_i^{\mathrm{start}}) - (T_i^{\mathrm{end}}) + \epsilon_i(P) - (1+\epsilon_i)(O).\end{aligned}$$

Notice that $T_i^{\text{end}} = T_{i-1}^{\text{start}}$ and $f_i^{\text{end}} = f_{i-1}^{start}$ as i goes from $t-1$ to 0. This gives us the following recurrence relations

$$\begin{aligned} T_{i-1}^{\text{start}} - 2T_i^{\text{start}} &= \epsilon_i P \\ \operatorname{div}(f_{i-1}^{\text{start}}) - 2 \cdot \operatorname{div}(f_i^{\text{start}}) &= 2(T_i^{\text{start}}) - (T_{i-1}^{\text{start}}) + \epsilon_i(P) - 1 + \epsilon_i)(O). \end{aligned}$$

These formulas are designed to telescope when they are summed. When the algorithm terminates we get

$$\begin{aligned} T_0^{\text{end}} &= \epsilon_0 P + 2T_0^{\text{start}} \\ &= \epsilon_0 P + \Big[\sum_{i=1}^{t-1} 2^i (T_{i-1}^{\text{start}} - 2T_i^{\text{start}})\Big] + 2^t T_{t-1}^{\text{start}} \\ &= \epsilon_0 P + \sum_{i=1}^{t-1} 2^i \epsilon_i P + 2^t T_{t-1}^{\text{start}} \\ &= \sum_{i=0}^{t} 2^i \epsilon_i P = N \cdot P \end{aligned}$$

Finally, the function f_P, computed by Miller's algorithm, has the divisor

$$\begin{aligned} \operatorname{div}(f_0^{\text{end}}) &= 2 \cdot \operatorname{div}(f_0^{\text{start}}) + 2(T_0^{\text{start}}) - (T_0^{\text{end}}) + \epsilon_0(P) - (1+\epsilon_0)(O) \\ &= \Big[\sum_{i=1}^{t-1} 2^i (div(f_{i-1}^{\text{start}}) - 2 \cdot \operatorname{div}(f_i^{\text{start}}))\Big] + 2(T_0^{\text{start}}) - (N \cdot P) \\ &\quad + \epsilon_0(P) - (1+\epsilon_0)(O) \\ &= \Big[\sum_{i=1}^{t-1} 2^i (2(T_i^{\text{start}}) - (T_{i-1}^{\text{start}}) + \epsilon_i(P) - (1+\epsilon_0)(O))\Big] + 2(T_0^{\text{start}}) \\ &\quad - (N \cdot P) + \epsilon_0(P) - (1+\epsilon_0)(O) \\ &= 2^t (T_{i-1}^{\text{start}}) + \sum_{i=0}^{t-1} 2^i \epsilon_i(P) - \sum_{i=0}^{t-1} 2^i (1+\epsilon_0)(O) - (N \cdot P) \\ &= N(P) - (N-1)(O) - (N \cdot P) \end{aligned}$$

□

Remark 8. *Let $P \in E[N]$ and let S be a point which is not generated by P and Q. Then Miller's algorithm computes a function f_P such that*

$$\operatorname{div}(f_P) = N(P) - (O).$$

By the alternative definition of the Weil-pairing, we can easily calculate the pairing by evaluating the function f_P in the point S. This is done by the following equation

$$\hat{e}_N(P,Q) = \frac{f_P(Q+S)}{f_P(S)} \Big/ \frac{f_Q(P-S)}{f_Q(-S)},$$

which is simply four applications of Millers algorithm.

Algorithm 1 Miller's algorithm

```
1: procedure MILLERS(P)
2:     T ← P
3:     f ← 1
4:     for i = t − 1 down to 0 do
5:         f ← f^2 · h_{T,T}
6:         T ← 2T
7:         if ε_i = 1 then
8:             f ← f · h_{T,P}
9:             T ← T + P
10:        end if
11:    end for
12:    return f
13: end procedure
```

5.2. Example

Let $E : y^2 = x^3 + 30x + 34$ over the field $\mathbb{F}_{631}$. We have that $E(\mathbb{F}_{631}) \simeq \mathbb{Z}/5\mathbb{Z} \times \mathbb{Z}/130\mathbb{Z}$. The points $P = (617, 5)$ and $Q = (121, 244)$ are in $E[5]$ and the point $S = (0, 36)$ are in $E(\mathbb{F}_{631})$, but not in the subgroup spanned by P and Q. Miller's Algorithm gives

$$\begin{aligned} f_P(Q+S) &= 326 \\ f_P(S) &= 523 \\ f_Q(P-S) &= 483 \\ f_Q(-S) &= 576, \end{aligned}$$

so the pairing $\hat{e}_5(P,Q)$ is

$$\hat{e}_5 = \frac{326}{523} \Big/ \frac{483}{576} = 512.$$

As $512^5 \mod 631 = 1$, 512 is indeed a 5-th root in $\mathbb{F}_{631}$.

Bibliography

[1] Joseph H. Silverman; The Arithmetic of Elliptic Curves.
Springer, 2nd Edition, 2009.

[2] Joseph H. Silverman and John Tate; Rational Points on Elliptic Curves.
Springer, 2nd Edition, 1994.

[3] Douglas R. Stinson; Cryptography, Theory and Practice.
Chapman & Hall/CRC, 3rd Edition, 2006.

[4] Alfred Menezes; An introduction to Pairing-Based Cryptography.
Article, 27-Oct-2013.

[5] Andreas Enge; Bilinear pairings on elliptic curves.
Article, 14-Feb-2014.

Appendices

A. Main program

```
#import the objects:
    #elliptic curves
    #points
    #identitypoints
#import millers algorithm
from EllipticCurves import *
import MillersAlgorithm

#create the elliptic curve
E = EllipticCurve(30,34,631)

#create the generating points in
#E[N], the N-torsion group of E.
P = Point(E, 617, 5)
Q = Point(E, 121, 244)
N = 5

#create the auxiliary point S
S = Point(E, 0, 36)

#calculate the pairing and print the result
print(MillersAlgorithm.pairing(E,P,Q,S,N))
```

B. Elliptic Curves

```
#Elliptic curves as objects
#Curves on Weierstrass form
#y^2 = x^3 + ax + b, a and b in F
#F finite field, hence the modulus
class EllipticCurve(object):

    #initialising the curve
    def __init__(self, a, b, mod):
        self.a = a
        self.b = b
        self.mod = mod
        self.discriminant = (-16 * (4 * a*a*a + 27 * b * b)) % mod

        #check that the curve is smooth
        if not self.isSmooth():
            raise Exception("The curve %s is not smooth!" % self)

    #check that the curve is smooth
    def isSmooth(self):
        return self.discriminant != 0

    #check that the point is on the curve
    def testPoint(self, x, y):
        return (y*y % self.mod)
                == ((x*x*x + self.a * x + self.b) % self.mod)

    #print the equation of the curve, overwrite function
    def __str__(self):
        return ("y^2 = x^3 + %sx + %s mod %s"
                        % (self.a, self.b, self.mod))

    #print the curve, overwrite function
    def __repr__(self):
        return str(self)
```

```
    #check if to curves is equal, overwrite function
    def __eq__(self, other):
        return (self.a, self.b) == (other.a, other.b)

#Point of elliptic curves as objects
class Point(object):

    #initialise the point
    def __init__(self, curve, x, y):
        self.curve = curve
        self.x = x
        self.y = y

        #check if the point is on the given curve
        if not curve.testPoint(x,y):
            raise Exception("The point %s is not on the given
                                curve %s!" % (self, curve))

    #calculate multiplicative inverses
    #in the finite field F
    #return the inverse of b modulo a
    def inverse(self,a,b):
        a0 = a
        b0 = b
        t0 = 0
        t = 1
        r = a0 % b0
        q = (a0-r) / b0

        while r > 0:
            temp = (t0 - q*t) % a
            t0 = t
            t = temp
            a0 = b0
            b0 = r
            r = a0 % b0
            q = (a0-r) / b0
        return (t % a)

    #print the coordinates of the point, overwrite function
    def __str__(self):
        return ("(%r, %r)" % (self.x, self.y))
```

```
#print the point, overwrite function
def __repr__(self):
    return str(self)


#calculate the inverse of a point, overwrite function
def __neg__(self):
    return Point(self.curve, self.x % self.curve.mod,
                             -self.y % self.curve.mod)

#add the point to another point, overwrite function
def __add__(self, Q):

    #check that the curves is the same
    if self.curve != Q.curve:
        raise Exception("Can not add points on different curves!")

    #check if the other point is the identity
    if isinstance(Q, Identity):
        return self

    #initialise the coordinates for calculation
    x_1, y_1, x_2, y_2, mod
      = self.x, self.y, Q.x, Q.y, self.curve.mod

    #check if the points are equal
    if (x_1, y_1) == (x_2, y_2):
        #if y = 0 it is its own inverse
        if y_1 == 0:
            return Identity(self.curve)

        #calculate the slope of the tangent line through the point
        temp = (2 * y_1 % mod)
        m = ((3 * x_1 * x_1 + self.curve.a) % mod)
                              * self.inverse(mod,temp)

    #if not equal
    else:
        #if x1=x2 and y1!=y1, they are inverses
        if x_1 == x_2:
            return Identity(self.curve)
```

```
            #calculate the slope of the line through the points
            temp = ((x_2 - x_1) % mod)
            m = ((y_2 - y_1) % mod) * self.inverse(mod,temp)

        #calculate the coordinates of the sum of the two points
        x_3 = (m*m - x_2 - x_1) % mod
        y_3 = (m*(x_3 - x_1) + y_1)

        #return the sum
        return Point(self.curve, x_3 , -y_3 % mod)

    #subtract a point from this point, overwrite function
    def __sub__(self, Q):
        return self + -Q

    #add the point to itself n times, overwrite function
    def __mul__(self, n):

        #check that n is an integer
        if not isinstance(n, int):
            raise Exception(n is not an integer!)

        #check if n is positive
        if n < 0:
            return -self * -n

        #check if n = 0
        if n == 0:
            return Identity(self.curve)

        #initialise the points
        Q = self
        R = self if n & 1 == 1 else Identity(self.curve)

        #add the points n times
        i = 2
        while i <= n:
            Q += Q
            if n & i == i:
                R += Q
            i = i << 1
        return R
```

```
#commutative multiplication, overwrite function
def __rmul__(self, n):
    return self * n

#list the coordinates, overwrite function
def __list__(self):
    return [self.x, self.y]

#check if two points are equal, overwrite function
def __eq__(self, other):
    if type(other) is Identity:
        return False
    return self.x, self.y == other.x, other.y

#check if two points is not equal, overwrite function
def __ne__(self, other):
    return not self == other

#get point ,overwrite function
def __getitem__(self, index):
    return [self.x, self.y][index]
```

```
#The identity of an elliptic curve as object
#instanse of points of elliptic curves
class Identity(Point):

    #initialise the point
    def __init__(self, curve):
        self.curve = curve

    #it is its own inverse
    def __neg__(self):
        return self

    #print the point, overwrite function
    def __str__(self):
        return Identity

    #add the identity to another point
    def __add__(self, Q):

        #check that the points are on the same curve
        if self.curve != Q.curve:
            raise Exception("Can not add points on different curves!")

        #return the other point
        return Q

    #add the identity to itself n times
    def __mul__(self, n):

        #check that n is an integer
        if not isinstance(n, int):
            raise Exception("Can not scale a point by something
                                which is not an int!")
        else:

            #return itself
            return self

    #chech if another point is the identity as well
    def __eq__(self, other):
        return type(other) is Identity
```

C. Millers Algorithm

```
#calculate the pairing
#input:
    #The ellipcit curve E
    #The generationg point P of E[N]
    #The generationg point Q of E[N]
    #The auxiliary point S
    #The order N of P and Q
#output: the weil pariring of P and Q
#e_N(P,Q) = (f_P(Q+S)/f_P(S))/(f_Q(P-S)/f_Q(-S))
#f_P and f_Q are divisorsfunctions
def pairing(E,P,Q,S,N):
    mod = E.mod
    a = millers(E,P,Q,(Q+S),N)
    b = millers(E,P,Q,S,N)
    c = millers(E,Q,Q,(P-S),N)
    d = millers(E,Q,Q,(-S),N)
    e = a*inverse(mod,b) * inverse(mod, c*inverse(mod,d))

    return (e % mod)
```

```
#millers algorithm
#based on the alternative definition
#of the weil pairing
#input:
    #The ellipcit curve E
    #The generationg point P of E[N]
    #The generationg point Q of E[N]
    #A point dependent on the auxiliary point S
    #The order N of P and Q
#output: the divisorfunctionf_P evaluated in S
#pseudocode and derivation of the
#algorithm in the project report
def millers(E,P,Q,S,N):
        mod = E.mod
        T = P
        f = 1
        n = bin(N)
        t = len(n)- 3
        for i in range(t - 1, -1, -1):
            f = f * f * h(E,T,T,S)
            T = 2*T
            a = int(n[t-i+2])
            if a == 1:
                f = f * h(E,T,P,S)
                T = T + P
        return (f % mod)
```

```
#the function h calculate the slope of
#the lines connection P and Q
def h(E,P,Q,S):
    X = S.x
    Y = S.y
    x_1 = P.x
    y_1 = P.y
    x_2 = Q.x
    y_2 = Q.y
    mod = E.mod
    a = E.a

    #check if the points are equal
    if (x_1, y_1) == (x_2, y_2):

        #check if the line is vertical
        if y_1 == 0:
            #m=-1 is choosen as constant
            m = -1

        #if not vertical
        else:

            #calculate the slope of the tangent line at P
            temp = (2 * y_1 % mod)
            m = ((3 * x_1 * x_1 + a) * inverse(mod,temp)) % mod

    #if not equal
    else:

        #check if inverses and vertical
        if x_1 == x_2:
            m = -1

        #if not inverses
        else:

            #calculate the slope of the line through P and Q
            temp = ((x_2 - x_1) % mod)
            m = ((y_2 - y_1) * inverse(mod,temp)) % mod
```

```
    #if vertical
    if m == -1:
        return (X - x_1) % mod

    #if not vertical
    else:

        #calculate the slope
        Y = (Y - y_1 - m*(X - x_1)) % mod
        temp = (X + x_1 + x_2 - m*m) % mod
        X = inverse(mod,temp)

        return (X*Y % mod)

#calculate multiplicative inverses
#in the finite field F
#return the inverse of b modulo a
def inverse(a,b):
  a0 = a
  b0 = b
  t0 = 0
  t = 1
  r = a0 % b0
  q = (a0-r) / b0

  while r > 0:
      temp = (t0 - q*t) % a
      t0 = t
      t = temp
      a0 = b0
      b0 = r
      r = a0 % b0
      q = (a0-r) / b0
  return (t % a)
```