Elliptic curves in the modern age

(3rd millennium)

Jens Berlips

Elliptic curves in the modern age

(3rd millennium)

Jens Berlips

August 21, 2006

draft date: August, 2006.

Typeset by the author with the LaTeX 2_ε Documentation System.

To my grandmother – may she rest in peace

Contents

1 Introduction **10**

2 Credits **12**

3 Preliminiaries **13**

3.1 Affine space 13
3.2 Projective space 14
3.3 From projective to Affine space and vice verse 15
3.4 Notation 15
3.5 Cubic plane curve 16
3.5.1 Intersection of curves 17

4 Elliptic Curves **18**

4.1 Addition on $E(k)$ 20
4.2 General theory 24
4.2.1 Order 24
4.2.2 Torsion points 25
4.2.3 Division polynomials 26

5 Practical computational considerations **30**

5.1 Binary ladder 30

6 Different parametrization **31**

6.1 Affine coordinates 31
6.2 Projective coordinates 32
6.3 Montgomery coordinates 33

7 Finding the order **35**

7.1 Schoof's method 36
7.1.1 The Frobenius endomorphism 36
7.1.2 Division polynomials and Schoof's method 37
7.1.3 Schoof's method explained 37

8 Factorization **40**
8.1 Factorization methods 41
8.2 Pollard $p - 1$ 41
8.3 Smooth numbers 43
8.4 Ideas of factorization 43
8.5 The general method 44
8.6 Elliptic curve method 45
8.6.1 Elliptic Curves over $\mathbb{Z}/N\mathbb{Z}$ 45
8.6.2 Algorithm explaination 46

9 Primality proving **47**
9.1 Certificates 48
9.2 Elliptic Curve primality proving explained 48

10 Getting down to implementation **49**

11 Ending words **50**

12 Source Code Listing **51**

List of Figures

1 $y = x^2$ intersect $y = 3$ with multiplicity 2 18
2 $E(\mathbb{R})$ defined by $y^2 = x^3 - 60$ 20
3 Illustration of the group law on $E(\mathbb{R})$ defined by $y^2 = x^3 - 60$ 21
4 9 Intersecting lines 23

Listings

1 Group binary ladder (field.py) 51
2 Elliptic curve Affine addition (field.py) 51
3 Elliptic curve Affine inverse (field.py) 52
4 Elliptic curve Montgomery arithmetic (field.py) 52
5 Elliptic curve trivial count for $\#E$ (field.py) 53
6 Elliptic curve Jacobi-method for $\#E$ (field.py) 53
7 Schoof's method for $\#E$ (field.py) 54
8 Factorization - trial division (field.py) 57
9 Factorization - Pollard $p - 1$ (field.py) 57
10 Factorization - ECM (field.py) 57

1 Introduction

> “The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extent that it would be superfluous to discuss the problem at length. ... Further, the dignity of the science itself seems to require that every possible means be explored for the solution of a problem so elegant and so celebrated.” - C. F. Gauss. [Gau]

Elliptic curves are becoming more and more important, not only as cryptographical applications useful for the industry but also as an important tool in mathematical theory (for example it was studied by Andrew Wiles when proving Fermat’s last theorem). In cryptography, where the industry requires shorter public keys for embedded systems, elliptic curve cryptograph (ECC) is simply harder to attack at a given key size and gives for that reason better protection.

As for mathematical purposes *each* elliptic curve defines an abelian group. Hence the theory creates a rich structure of an almost inexhaustible pool of abelian groups.

This last property has come to have remarkable significance in number theory. On one hand due to Hendrik W. Lenstra’s ingenious generalization of the Pollard $p-1$ (see section 8.2) to elliptic curves, we have the so called elliptic curve method, ECM, (see section 8.6). A probabilistic factorization algorithm dependent on the length of the smallest factor. On the other hand we also have a very fast primality proving algorithm (see section 9) with a natural extension which makes it possible to create primality certificates. Each certificate describes a proof of primality for a given prime and can be verified almost instantaneously.

Many of the algorithms described in this thesis exploits in some sense the random structure of elliptic curves - especially the order of the group is a common algorithmic problem to lay down. Thanks to R. Schoof’s idea (see section 7.1) this is now possible even for elliptic curves with order up to 100 digits.

I aim to explain and implement those algorithms in a way accessible for an undergraduate student with appropriate mathematical background. All implementations are made in the pseudo-code-like platform independent language Python (see `http://www.python.org` and section 10).

I strongly encourage the reader to look at some of the algorithms and compare them to the corresponding Python source code (referenced at the end of each algorithm).

2 Credits

First I would like to thank my advisor, Pär Kurlberg, who helped to give me the theoretical ground on which I could make my own conclusions – and then helping me to finaly modify things to ultimate correctness. I would also like to thank Christian Lundkvist who took time for excellent explaination of some of the algebraic geometric aspects of Schoof's algorithm. For some correction of the English I would like to give my sincere thanks to my father Leo Berlips. Then also William Stein for his excellent software SAGE making fast polynomial arithmetic possible.

Last but not least, R. Schoof, D.J. Bernstein and H. Cohen. All of whom I had the pleasure of meeting in this year's (2006) winter school at the University of Arizona.

3 Preliminiaries

Let us first have a look at some basic algebraic definitions used frequently throughout this thesis.

Definition 1 (Finite field)**.** A finite field is a field with finite cardinality (i.e., the number of elements), and a finite field with order p is denoted $\mathbb{F}_p$.

We can identify $\mathbb{F}_p$ with $\mathbb{Z}/p\mathbb{Z}$ by an isomorphism and we will use $\mathbb{F}_p$ and $\mathbb{Z}/p\mathbb{Z}$ interchangable. In order for $\mathbb{F}_p$ to exist, it is required that p is a prime number.

Definition 2 (Characteristic)**.** The characteristic of a field k, written $\text{char}(k)$, is the smallest $p \in \mathbb{Z}^+$ s.t. $p \cdot 1_k = 0_k$, or 0 if no such finite p exist. Here 1_k and 0_k is the identity and zero element in k respectively .

For example $\text{char}(\mathbb{F}_p) = p$ and $\text{char}(\mathbb{R}) = 0$. The characteristic of a field is a very important. One aspect is that some change of variables is possible only when $\text{char}(k)$ fulfills some specific requirements. A simple example is $y^2 = x^2 + ax + 1$. Completing this square is not possible in a field of characteristic 2 as that would require ”division by zero”.

3.1 Affine space

An Affine space is a set of points where you can subtract points to get a vector (the line from one point to another) or add vector to point to get another point, but you cannot add points, since there is no origin (see [Ful] or [Wik1]). By $\mathbb{A}^n(k)$ or simply $\mathbb{A}^n$ (if k is understood) we mean the n-fold cartesian product,

$$\mathbb{A}^n(k) = \underbrace{k \times \cdots \times k}_{\text{n times}}$$

By an Affine line we shall mean $\mathbb{A}^1(k)$ and $\mathbb{A}^2, \mathbb{A}^3$ the Affine plane and surface respectively. If $F \in k[x, y]$ is a polynomial, we call the zero-set,

$$V(F) = \left\{(x, y) \in \mathbb{A}^2 \mid F(x, y) = 0\right\}$$

an *affine plane curve.*

Example 1. Let $\mathbb{R}[x, y] \ni F(x, y) = y - x^2$ then $V(F)$ is the Affine plane curve called a *cubic curve.*

3.2 Projective space

Suppose we want to introduce the notion of a "geometric infinity", that is some some sort of "completeness", in the Affine space above. The reason for this could be that we want a robust answer to questions of whether certain asymptotic curves intersect or not. Let for example $y^2 = x^2 + a$ and $y = x + b$ $(a, b \in k)$ be two plane Affine curves. Now as $x \to \infty$ both curves are asymptotic, but do they intersect in some sense? One way of enlarging the Affine space so those two curves *do* intersect is by identifying each tuple $(x, y) \in \mathbb{A}^2$ with the point $(X, Y, Z) = (X, Y, 1) \in \mathbb{A}^3$. Then if we further identify $(X, Y, 1)$ with the line through $(0, 0, 0)$ and itself we have that every line through $(0, 0, 0)$ can be uniquely identified with one such point (x, y) except for lines in the plane $Z = 0$.

These Affine lines with $Z = 0$ are in the "completed" Affine space (or projective space) simply called *projective points.* When referred to those lines in Affine space we will call them *points at infinity.* More formally,

Definition 3 (Projective n-space)**.** The projective n-space $\mathbb{P}^n(k)$ over a field k is defined as,

$$\mathbb{P}^n(k) = (k^{n+1} - \{(0, 0, \ldots, 0)\}) / \sim$$

Where $x \sim y$ if $\exists t \neq 0$ such that $y = tx$.

If k is understood we will write $\mathbb{P}^n$. To separate from Affine coordinates we write $[X_1 : X_2 : \cdots : X_{n+1}]$ for points in $\mathbb{P}^n$.

As for answering our question of the two asymptotic curves above let us introduce the notion of homogenization.

Definition 4 (Homogenization)**.** A homogenization f^* in $k[X, Y, Z]$ of a polynomial f in $k[x, y]$ is a polynomial with the property $f^*(tX, tY, tZ) = tf^*(X, Y, Z)$, $t \in k$, where $f(X, Y) = f^*(X/Z, Y/Z, 1)$. Define,

$$f^* = Z^{\deg(f)} \cdot f(X/Z, Y/Z)$$

It follows from the definition that homogeneous polynomials are defined as sums of monomials with the same degree. Since $f^*(tX, tY, tZ) = 0 \Leftrightarrow tf^*(X, Y, Z) = 0$ it is natural to say that (X, Y, Z) is equivalent (or homogenous) to (tX, tY, tZ). It follows that the zero-set (the curve) for homogenized polynomials are subsets (called projective curves) of a projective space whereas the zero-set for non-homogenized polynomials are subsets (Affine curves) of an Affine space .

3.3 From projective to Affine space and vice verse

If the projective coordinate $Z \neq 0$, all triples $[X : Y : Z]$ have an Affine representation by the canonical map $\mathbb{P}^2 \ni [X : Y : Z] \mapsto (X/Z, Y/Z) \in \mathbb{A}^2$. If $Z = 0$, there is no Affine representation for those projective points $[X : Y : 0]$.

Note. It is worth to stress the fact that *all* projective points $[X : Y : Z]$ are can be interpreted as *lines* in $\mathbb{A}^3$.

As for projective curves they are very similar to Affine curves (by definition) except they are defined by a *homogenized* polynomial. In this way we can through the canonical change of variables above map a projective curve into an Affine curve and vice verse (keeping track of the point at infinity).

Example 2. Continuing from the example in section 3.2 the homogenized polynomials are $Y^2 = X^2 + aZ^2$ and $Y = X + bZ$. The points at infinity correspond to the plane $Z = 0$ where

$$\begin{aligned} Y^2 &= X^2 \\ Y &= X \end{aligned}$$

Further we have that in the plane $Z = 0$ they actually intersect in projective space! Using Affine representation this is not true, but we then introduce the notion of intersection ”at infinity”.

3.4 Notation

Throughout this thesis I will use the following notations,

- k is a field (ex. $k = \mathbb{F}_p$ or $k = \mathbb{C}$) and $k[x_1, \ldots, x_n]$ is the ring of polynomials in $x_1, \ldots, x_n$.
- p is a prime number and also $q = p^i$, $i = 1, 2, \ldots$.
- N is a composite number with $N = \prod_{i=0}^{k} p_i^{\alpha_i}$.
- Affine coordinates will be represented by lower-case letters, x, y and projective coordinates by capital letters X, Y, Z. They will relate implicitly to each-other by the change of variables $x = X/Z$ and $y = Y/Z$ if $Z \neq 0$.

3.5 Cubic plane curve

A general *cubic affine plane curve* is defined by the zero-set to $F(x, y)$,

$$F(x, y) = ax^3 + bx^2y + cxy^2 + dy^3 + ex^2 + fxy + gy^2 + hx + j$$

The curve is called singular (or non-smooth) in (x_1, y_1) if $\frac{dF}{dx} = \frac{dF}{dy} = 0$ at (x_1, y_1) or equivalently there is no tangent-line defined at (x_1, y_1). Recall that the tangentline at (x_1, y_1) for F is $\frac{dF}{dx}(x - x_1) + \frac{dF}{dy}(y - y_1) = 0$. By definition (4) there is also a natural extension to the projective representation,

$$F(X, Y, Z) = aX^3 + bX^2Y + cXY^2 + dY^3 + eX^2Z + fXYZ + gY^2Z + hXZ^2 + jZ^3$$

A smooth *curve*, $\mathcal{C}$, is a plane curve (either Affine or projective) that is defined by a non-singular polynomial. If this polynomial is F then

$$\mathcal{C} = V(F)$$

Where $V(F)$ is the zero set to F.

It can be shown using Riemann-Roch theorem [Sil86, p 37-41] that any *non-singular cubic curve* (for our purposes this is enough, the interested reader can refer to the notion of *genus* in [Sil86, p. 39] describing this more accurately) has a representation in the much simpler Weierstrass equation,

$$F : y^2 + a_1xy + a_2y = x^3 + b_1x^2 + b_2x + b_3$$

If $\mathrm{char}(k) \neq 2$ we can complete the square on the left hand side by substituting y for $\frac{1}{2}(y - a_1x - a_3)$ to get

$$F : y^2 = 4x^3 + (a_1^2 + 4b_1)x^2 + (4b_2 + a_1b_3)x + (a_2^2 + 4b_3) \tag{1}$$

where $a_i, b_i \in k$. If further $\mathrm{char}(k) > 3$ another similar change of variables transforms (1) into,

$$F : y^2 = x^3 + ax + b, \quad a, b \in k \tag{2}$$

this equation is called *Weierstrass form* and will be often used when referring to cubic curves. For this reason it is important to know when the curve defined by (2) is non-singular (i.e., so that it defines a *smooth curve*). Calculating the partial derivatives to (2), the defined curve is singular if and only if $2y = 0$ and $\frac{dF}{dx} = 0$. These vanish simultaneously exactly when $y = 0$ (since $\mathrm{char}(k) \neq 2$) and x is a multiple root of (2). This last condition is categorized by the discriminant for the cubic F called Δ_F. For F this discriminant is $\Delta_F = -16(4a^3 + 27b^2)$ and then F is singular if and only if $\Delta_F = 0$.

3.5.1 Intersection of curves

As we soon will see, defining a grouplaw on a curve is close related to characterizing the intersection points between curves (lines, cubics etc). In projective geometry this theory is easier and more robust than in Affine geometry. A very important (and necessary for our purposes) theorem in projective geometry is due to Bézout - a French mathematician in the 18:th century.

Before introducing Bézout's theorem we need to somewhat understand *intersection numbers* (see figure 1) for projective curves. If C, D are two projective plane curves not sharing a common curve, then intersection number at a point $x \in C \cap D$, counting multiplicites will be denoted $i(C, D; x)$.

The intersection number $i(C, D; x)$ is defined as $\dim R_x(k^3)/(C, D)$, where R_x is the set of rational functions defined at x (see [Ful, p. 74-85] for more details and proofs).

Theorem 1 (Bézout's theorem for curves). *Let C and D be projective curves defined by polynomials of degree c respectively d over an algebraically closed field k. Assume C and D do not share a common component (i.e, they contain no common curve). Then $C \cap D$ is finite, and more precisely, if $i(C, D; x) \geq 1$ counts the multiplicity (see above and figure 1) at each inter-*

section point x we have that,

$$\sum_x i(C, D; x) = cd$$

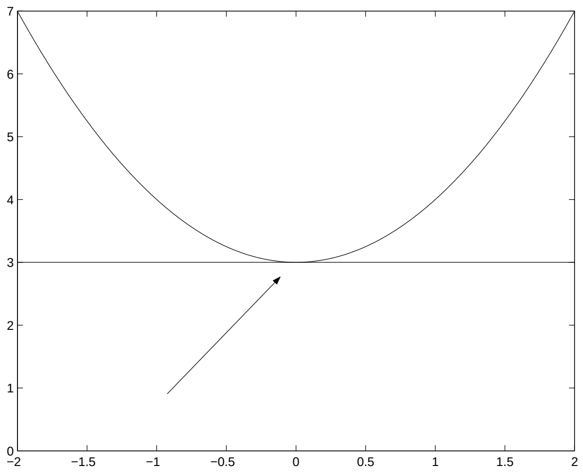

Figure 1: $y = x^2$ intersect $y = 3$ with multiplicity 2

Note. Consider again the curves from example 2. Then those two curves are of degree 2 respectively 1 and share no common curve, moreover we have that

$$\sum_x i(C, D; x) = 2$$

But they only intersect at infinity! This intersection point must for that reason have multiplicity 2.

It should be emphasized that Bézout's theorem can only be applied to *projective curves*! Moreover, we shall see that Bézout's theorem is needed to prove the group law on elliptic curves and for this reason it is *one* motivation behind the choice of introducing the notion of projective space.

4 Elliptic Curves

The principal objects we will study in this thesis are planar smooth cubic curves, more commonly referred to as *elliptic curves*. (This is not the whole truth, but again, without the notion of *genus* [Sil86, p. 39], it is hard to categorize it better.)

Definition 5 (Elliptic Curve)**.** A projective curve E, over a field k with $\mathrm{char}(k) \neq 2, 3$, written $E(k)$ defined by the non-singular Weierstrass form,

$$E_{a,b}(k) : ZY^2 = X^3 + aZ^2X + bZ^3 \tag{3}$$

is called a *projective elliptic curve.*

Note. We will write $E(k)$ if a, b is understood and simly E when k is understood. This is the projective definition of E. For simplicity we will also often refer to the Affine curve as E, we shall see that the projective representation and the affine representation are one and the same if necessary adjustment are made to the affine space.

Let us have a look at the differences between the Affine and projective representation of $E(k)$.

There are two major reasons why projective coordinates are necessary, first Bezout's theorem (theorem 1) is applicable only in projective coordinates and we need that to prove the group law, secondly it is necessary in order for the group law to be defined that we include the point at infinity. In projective corodinates this corresponds to points in the plane $Z = 0$ (see section 3.2) defined over (3). If $Z = 0$ in (3), then $X = 0$ (this projective point correspond to all vertical lines in the Affine space) and we see that E intersects $Z = 0$ in exactly one point, namely $[0 : 1 : 0]$ (homogenous to all points $[0 : Y : 0]$). This point will be introduced as $\mathcal{O}$ in Affine space. Summarized, in projective space we have a robust algebraic structure including the point at infinity whereas in Affine coordinates this point must be "kept track of seperately". When implementing affine elliptic curves we actually do this!

We refer to elliptic curves $E(k) \subset \mathbb{P}^2$ as the set of points $P = (x, y) \in A^2$ (with projective equivalent $[x : y : 1]$) satisfying (2) together with a point $\mathcal{O} = [0 : 1 : 0] \in \mathbb{P}^2$ called the point at infinity. With this view both Affine and projective coordinates can be used interchangeable.

Elliptic curves intersect the projective points $Z = 0$ in only one point, this follows from the fact that there is only one asymptotic vertical line in any elliptic curve, see for example figure 2.

When $\mathbb{R}$ is replaced with a finite field, the elliptic curve will not be as smooth as figure 2 but it still give some understanding of the geometric aspects of an elliptic curve.

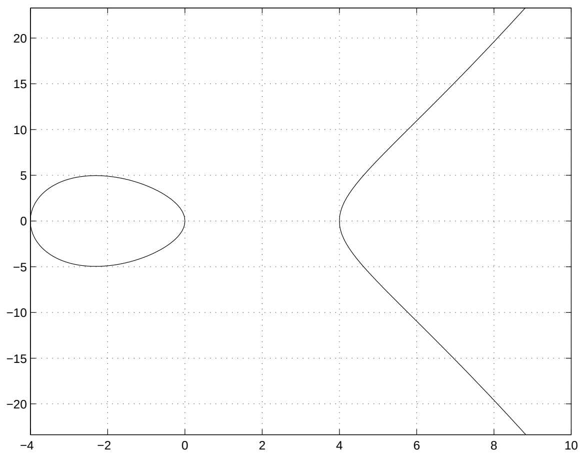

Figure 2: $E(\mathbb{R})$ defined by $y^2 = x^3 - 60$

4.1 Addition on $E(k)$

Let P, $Q \in E(k)$ with $P, Q \neq \mathcal{O}$ and L the unique line connecting P, Q and a third point of intersection, R also on $E(k)$ (if $P = Q$, L is the tangentline to $E(k)$ at P). The main question here is of course: is R well-defined?

This question is answered by Bezout's theorem which tell us that R is well-defined in $\mathbb{P}^2$, but not in $\mathbb{A}^2$ since it may equal the point at infinity $\mathcal{O}$. As we will see it is of great importance that R is well-defined and for this reason we need to introduce projective coordinates and the point at infinity to $\mathbb{A}^2$.

Define a map from $E \times E \longrightarrow E$ given by $\varphi : (P, Q) \to R$, then φ acts *similarly* to an addition on $E(k)$ except there is no identity (any two points give raise to a unique third point not equal any of the two first). To remedy this, we introduce the point at infinity $\mathcal{O}$ and define $\bigoplus$ on $E(k)$ as follows:

$$P \bigoplus Q = \varphi(\mathcal{O}, \varphi(P, Q)) \tag{4}$$

Using $\bigoplus$ we can define the group law on $E(k)$. (In fact it is easy to see that by redefining $\bigoplus$ we can actually have any point as the identity).

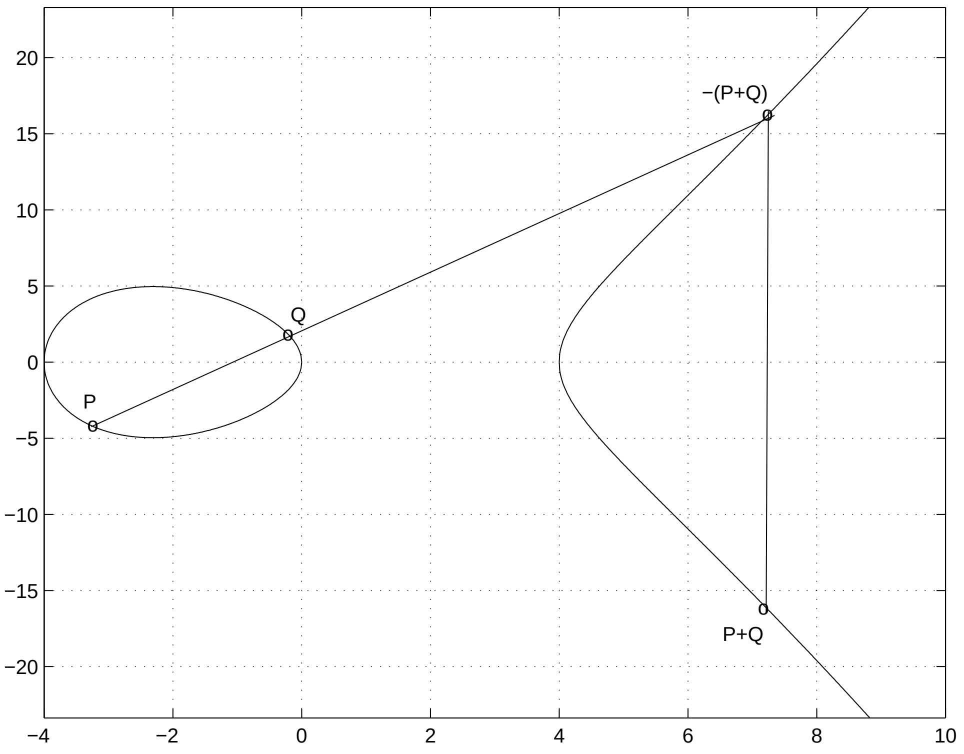

Figure 3: Illustration of the group law on $E(\mathbb{R})$ defined by $y^2 = x^3 - 60$

Theorem 2 (Elliptic Curve Group Structure)**.** *An elliptic curve $E(k)$, over a field k, forms an abelian group with the operation $\bigoplus$ and the point $\mathcal{O}$ being the identity.*

Proof. We must prove the following statements. let P,Q,$R \in E(k)$,

(I) $\bigoplus$ is associative, $(P \bigoplus Q) \bigoplus R = P \bigoplus (Q \bigoplus R)$.

(II) $\bigoplus$ is commutative, that is $E(k)$ is abelian and $P \bigoplus Q = Q \bigoplus P$,

(III) There exist an identity $\mathcal{O} \in E(k)$ such that $P \bigoplus \mathcal{O} = \mathcal{O} \bigoplus P = P$, $\forall P \in E(k)$.

(IV) P has an inverse $-P \in E(k)$ such that $P \bigoplus -P = \mathcal{O}$

Two point combinations (P, Q) and (Q, P) both defines the same unique line (see for example [Ful] or [Wik2]) and it follows from the definition of φ that $\varphi(P, Q) = \varphi(Q, P)$, concluding (II).

Define the inverse of $P = [X_1 : Y_1 : Z_1]$ as $-P = [X_1 : -Y_1 : Z_1]$. The line L intersecting $E(k)$ in P and $-P$ is defined by the projective equation $L = \{X : X = ZX_1\}$ and by Bezout's theorem it must interesect $E(k)$ in a third point. If $Z = 0$ it follows that $X = 0$ and $Y = t$, and on the projective plane this is $\mathcal{O}$, concluding (IV) and (III). Only statement (I) is difficult - the associativity. To prove this we need the following lemma.

Lemma 1 (Nine associated points). *Let $\mathcal{C}$ be an irreducible cubic curve, $\mathcal{C}'$ and $\mathcal{C}''$ cubics. If $\mathcal{C}' \cap \mathcal{C} = \bigcup_{i=1}^{9} P_i$, where P_i are non-singular points on $\mathcal{C}$, and suppose $\mathcal{C}'' \cap \mathcal{C} = (\bigcup_{i=1}^{8} P_i) \cup Q$, then $Q = P_9$.*

Proof. See [Ful, p. 124]. □

With these tools in Algebraic geometry we can continue proving theorem 2. The idea is to connect the points $P + (Q + R)$ and $(P + Q) + R$ through 9 constructed intersecting lines (spanning 2 cubic curves) then we use Bezout"s theorem and the associativity theorem above to show that they must be equal.

Define three lines L_1,L_2, L_3 and I_1,I_2,I_3 the following way,

$$\begin{array}{rcccccc} L_1 \cap E(k) & = & P & \cup & Q & \cup & -(P+Q) \\ L_2 \cap E(k) & = & (P+Q) & \cup & R & \cup & -((P+Q)+R) \\ L_3 \cap E(k) & = & (Q+R) & \cup & \mathcal{O} & \cup & -(Q+R) \\ & & & & & & \\ I_1 \cap E(k) & = & (P+Q) & \cup & \mathcal{O} & \cup & -(P+Q) \\ I_2 \cap E(k) & = & Q & \cup & R & \cup & -(Q+R) \\ I_3 \cap E(k) & = & P & \cup & (Q+R) & \cup & -(P+(Q+R)) \end{array}$$

Each line intersects exactly three points on $E(K)$, and is well-defined. It is illustrated in this picture,

Those first three lines defines the cubic $L_1 \cdot L_2 \cdot L_3$, intersecting $E(k)$ at eight points namely:

$$\begin{array}{cccc} P, & Q, & P+Q, & Q+R, \\ R, & \mathcal{O}, & -(P+Q), & -(Q+R) \end{array}$$

and by Bezout's theorem, we know there must be a ninth point of intersection, call this point $U \in E(k)$. Using lemma 1 (recall that any elliptic curve

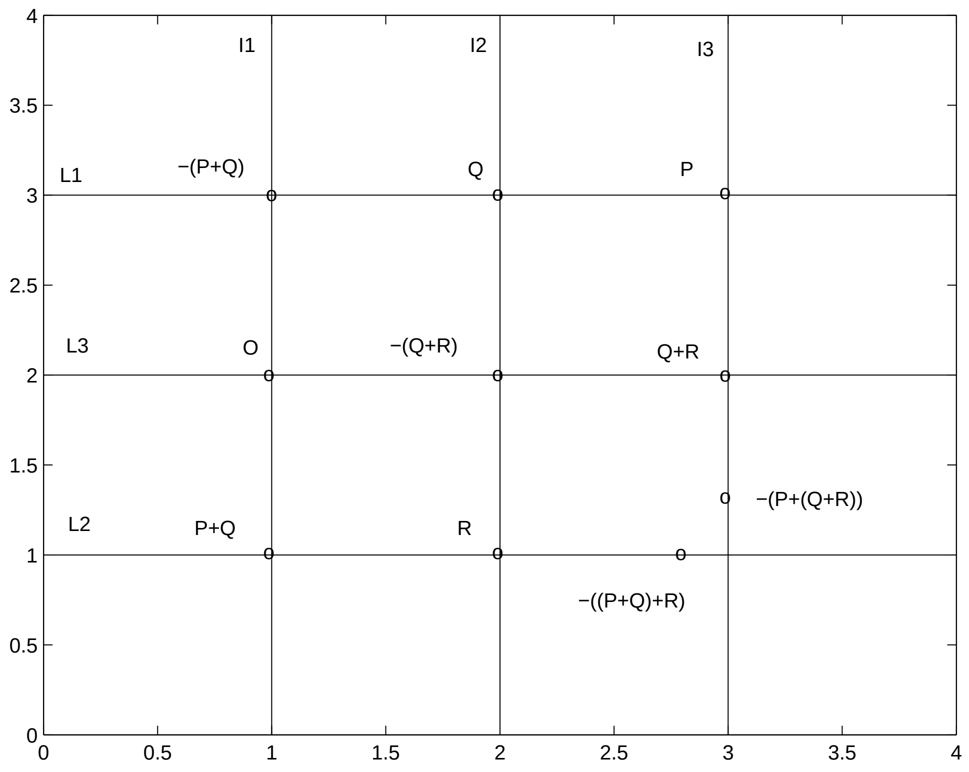

Figure 4: 9 Intersecting lines

is an irreducible cubic curve) we know that any other cubic passing through those eight points also intersects U.

The last three lines defines the cubic $I_1 \cdot I_2 \cdot I_3$. It also intersects those eight points above and must therefore pass through U, but it also passes through an additional point $-(P + (Q + R))$. But a cubic not containing a common curve with $E(k)$ cannot intersect $E(k)$ in 10 points, thus two points must be equal. By definition U is not equal to any of the first 8 points and must therefore equal $-(P + (Q + R))$.

By symmetry in arguments we could just as well have started with the cubic $I_1 I_2 I_3$ and then deduced that $U = -((P + Q) + R)$. Thus,

$$-((P + Q) + R) = U = -(P + (Q + R))$$

which concludes the proof for $\bigoplus$. $\square$

We write down the concrete algebraic operations for adding and doubling points on $E(\mathbb{F}_p)$. In Affine coordinates those operations are:

1. $-P = (x_1, -y_1)$
2. if $P \neq -Q$ then $P + Q = (x_3, y_3)$, where

$$\begin{aligned} x_3 &= m^2 - a - x_1 - x_2 \\ y_3 &= m(x_3 - x_1) + y_1 \end{aligned}$$

   The slope m of the tangentline is defined as:

$$\begin{aligned} m &= (y_2 - y_1)/(x_2 - x_1) \quad & (x_2 \neq x_1) \\ m &= (3x_1^2 + 2ax_1 + b)/2y_1 \quad & (x_1 = x_2) \end{aligned}$$

   and finally if $x_1 = x_2$ and $y_1 = 0$ then $P + Q = \mathcal{O}$.

3. if $P = -Q$, then $P + Q = \mathcal{O}$.

Note. From here on we will drop the symbol $\bigoplus$ and instead use $+$ and $+(-P)$ will be written simply $-P$ for group operations on $E(k)$. A common operation on $E(k)$ is multiplication by elements in $\mathbb{Z}$. Any abelian group is a $\mathbb{Z}$-module and therefore nP, $n \in \mathbb{Z}$ is canonically defined. It is custom to define nP through the map $[n] : E \longrightarrow E$ as

$$P \mapsto \underbrace{P + \ldots + P}_{\text{n times}}$$

with $[0]P = \mathcal{O}$ and $[-n]P = -([n]P)$.

4.2 General theory

To proceed to more advanced topics on elliptic curves and especially the algorithms we need to further develop our theory.

4.2.1 Order

An elliptic curve (definition 5) over a finite field $\mathbb{F}_p$ obviously has a finite order. The order is clearly bounded above by $2p+1$ because there is p values for x and each y have a maximum of two solutions in x, plus the point at infinity.

However, it is possible to write down a formula for the bound as a function $\#E_{a,b}(\mathbb{F}_p) : (a,b) \to \mathbb{Z}$, written

$$\begin{aligned}\#E_{a,b}(\mathbb{F}_p) &= 1 + \sum_{x\in\mathbb{F}_p} (1 + \left(\frac{x^3+ax+b}{p}\right)) \\ &= p + 1 + \sum_{x\in\mathbb{F}_p} \left(\frac{x^3+ax+b}{p}\right) = p + 1 - a_p\end{aligned}$$

From here on we define $a_p = -\sum_{x\in\mathbb{F}_p}\left(\frac{x^3+ax+b}{p}\right)$ and as usual $\left(\frac{a}{p}\right)$ is the Jacobi symbol (which is the same as quadratic character χ of an element in $\mathbb{F}_p$). Recall that $\left(\frac{a}{p}\right) = 1$ if a is a quadratic residue mod p and -1 otherwise, also $\left(\frac{0}{p}\right) = 0$.

A nice way of guessing the bound is to assume that $x^3 + ax + b$ is a random function and we give an heuristic bound on the sum by a random walk on $\mathbb{Z}$, starting at 0. After p steps the expected (in probabilistic sense) distance walked is roughly of order $\sqrt{p}$ [Wol1].

A sharp version of this was proven by Helmut Hasse.

Theorem 3 (Hasse). *Let $\#E(\mathbb{F}_p)$ be the order of the elliptic curve $E(\mathbb{F}_p)$, then*

$$|\#E(\mathbb{F}_p) - p + 1| \leq 2\sqrt{p} \tag{5}$$

Proof. See [Sil86, p. 131-133]. □

4.2.2 Torsion points

A torsion point (or division point) is a point $P \in E(k)$ of finite order. A torsion point of order n (i.e, a point $P \in E$ s.t. $[n]P = \mathcal{O}$) is called a n-torsion point. The set of n-torsion points is called $E(k)[n]$ or when k is understood plainly $E[n]$.

$$E(k)[n] = \{P \in E(k) : [n]P = \mathcal{O}\}$$

The set of n-torsion points naturally defines a subgroup in $E(k)$. Since k may not be algebraically closed we can expect $E[n]$ defined over $\overline{k}$, the algebraic closure of k (i.e., the extension field containining all roots for all polynomials in $k[x]$) to be different (larger). To understand the set of torsion points we need to interpret the torsion points in terms of the function $[n]$:

$$E[n] = \ker([n])$$

This way of looking on $[n]$ is the subject for the next section.

4.2.3 Division polynomials

We will see that there is a polynomial whose roots are exactly the torsion points. This is an essential part of Schoof's algorithm in section 7.1.

If $P = (x, y) \in E_{a,b}(\mathbb{F}_p)$ then,

$$[2]P = (x', y') \left(\frac{(3x^2 + a)^2 - 4xy^2}{4y^2}, \frac{3x^2 + a}{2y}(x - x') - y \right) \tag{6}$$

using the formulas from section 4.1. We have that $[2]P = \mathcal{O}$ if and only if $4y^2 = 0$ and it follows that $P \in E(\mathbb{F}_p)[2]$. In the following section we will investigate and prove the existance of a recursion equation for polynomials f_n, whose roots are exactly the torsion points. We saw an example above of f_2 where $f_2^2 = 4y^2$ and and it follows that the 2-torsion points are exactly the roots of $y^2 = x^3 + ax + b$, as expected.

Theorem 4 (Division polynomials)**.** *Let $E_{a,b}(\mathbb{F}_p)$ be an elliptic curve, then there is a polynomial $f_n \in \mathbb{F}_p[x, y]$ depending on $E_{a,b}(\mathbb{F}_p)$ such that if $P = (x, y) \in E_{a,b}(\mathbb{F}_p)$ and $[n]P = \mathcal{O}$ then $f_n(x, y) = 0$. Moreover f_n can be constructed recursively using the following relations:*

$$\begin{aligned} f_{2n+1} &= f_{n+2}f_n^3 - f_{n+3}^3 f_{n-1} && (7) \\ f_{2n} &= f_n(f_{n+2}f_{n-1}^2 - f_{n-2}f_{n+1}^2)/y && (8) \end{aligned}$$

Note. For $E(\overline{\mathbb{F}})$ the statement above is an equivalence, $[n]P = \mathcal{O} \Leftrightarrow f_n(x, y) = 0$.

Proof. I will give an outline of the proof over $\mathbb{C}$ and mention how it can be extendable to $\mathbb{F}_p$. For a more detailed proof see [Lan, p. 33-43].

An elliptic curve $E(\mathbb{C})$ can be be considered as $\mathbb{C}/\mathbf{L}$, where $\mathbf{L}$ is a lattice

$$\mathbf{L} = \{a\omega_1 + b\omega_2 \mid a, b \in \mathbb{R}\}$$

(see [Lan, p. 3]). An Elliptic function is a $\mathbf{L}$-periodic meromorphic function defined on the whole complex plane. If f is $\mathbf{L}$-periodic then $f(z+\omega) = f(z)$ when $z \in \mathbb{C}$ and $\omega \in \mathbf{L}$. The lattice $\mathbf{L}$ together with an *elliptic function*, the so-called Weierstrass $\wp$-function has an algebraic meaning. The $\wp$ function satisfies Weierstrass' differential equation:

$$\wp' = 4\wp^3 + a\wp + b \tag{9}$$

with a, b depending on $\mathbf{L}$, and where

$$\wp(z) = \frac{1}{z^2} + \sum_{\omega \in L \setminus \{0\}} \left[\frac{1}{(z-\omega)^2} - \frac{1}{\omega^2} \right]$$

We give (9) an algebraic meaning by identifying the point $(\wp(z), \wp'(z))$ with (x, y) on the Weierstrass equation

$$y^2 = 4x^3 + ax + b \tag{10}$$

and moreover this identification defines, through $\wp$, an isomorphism between $\mathbb{C}/\mathbf{L}$ and the affine curve (over $\mathbb{C}$) defined by (10), with the point at infinity added. Looking at an elliptic curve in this way the group law is trivial since for $z_1, z_2 \in \mathbb{C}$, $P = (\wp(z_1), \wp'(z_1))$ and $Q = (\wp(z_2), \wp'(z_2))$ we define

$$P + Q = (\wp(z_1 + z_2), \wp'(z_1 + z_2))$$

and the group law over the elliptic curve is induced by the group law over $\mathbb{C}/\mathbf{L}$. We also have that when $\wp$ has a pole (i.e., when $z = \omega \in \mathbf{L}$), $(\wp, \wp')$ is exactly the point at infinity.

We are interested in the n-torsion points $u \in \mathbb{C}/\mathbf{L}$ with $nu = 0$. If for example $n = 2$ then we have the 2-torsion points

$$0, \frac{\omega_1}{2}, \frac{\omega_2}{2}, \frac{\omega_1 + \omega_2}{2} \tag{11}$$

and those mapped into the curve by $(\wp, \wp')$ will go to 2-torsion points on the elliptic curve.

Let us now consider the family of functions $\{f_n\}$ defined as

$$f_n(z)^2 = n^2 \prod_{\substack{u \in \mathbb{C}/\mathbf{L} \\ nu=0, u \neq 0}} (\wp(z) - \wp(u))$$

If n *is odd* all factors in the product occur with multiplicity 2, since the values $\pm u$ are not congruent mod L, and because $\wp$ is even they map to the same value. This will also hold when n is even except for non-trivial points u for which $2u \equiv 0$. At those points $\wp(z) - \wp(u)$ will be a double zero and these correspond exactly to the roots of $\wp'(z)^2$. Because solutions $\wp'(z) = 0$ corresponds to exactly those points, we can write

$$\wp'(z)^2 = 4 \prod_{\substack{2u=0 \\ u\neq 0}} (\wp(z) - \wp(u))$$

Hence the product defining f_n^2 is a perfect square and since f_n is defined in $\wp$ and $\wp'$ we use a theorem [Lan, p. 7] saying that any polynomial in $\wp$ and $\wp'$ are known to be elliptic functions.

1. n **odd:** $f_n(z) = P_n(\wp(z))$, where P_n is a polynomial of degree $(n^2 - 1)/2$ with leading coefficient n.

2. n **even:** $f_n(z) = \frac{1}{2}\wp' P_n(\wp(z))$, where P_n is a polynomial of degree $(n^2 - 4)/2$ with leading coefficient n.

To create a recursive relation let us consider the function,

$$\wp_n(z) = \wp(nz) - \wp(z)$$

$\wp_n(z)$ has poles at the zeros of f_n^2 with the same multiplicity, namely 2, and moreover it has zeros at points z for which $(n+1)z \equiv 0$ or $(n-1)z \equiv$ is satifsfied (because $nz = z$ or $nz = -z$ is equivalent to $\wp_n(nz) = \wp(z)$ since $\wp$ is even). The function

$$\frac{f_n^2(\wp(nz) - \wp(z))}{f_{n+1}f_{n-1}}$$

will have no poles or zeros in $\mathbb{C}/\mathbf{L}$ hence it must be constant (Louville's theorem [Wun, p. 194]).

By expanding around 0 this constant is -1 (see [Lan, p. 34]) and it follows that

$$\wp(nz) = \wp(z) - \frac{f_{n+1}f_{n-1}}{f_n^2} \tag{12}$$

Now consider

$$\wp_n - \wp_m = \wp(nz) - \wp(mz) = f_{n+1}f_{n-1}f_m^2 - f_{m+1}f_{m-1}f_n^2$$

for $n, m \in Z$ and $m > n$. We see that $\wp_n - \wp_m$ vanish at points u such that

$$(m \pm n)u \equiv 0$$

with multiplicity 1 ($\wp_n$ and $\wp_m$ vanish at $n \pm 1$ and $m \pm 1$ respectively). But neither f_n or f_m have a zero at these points, since$mu, nu \not\equiv 0$. Hence these points must be the zeros of

$$\underbrace{f_{n+1}f_{n-1}f_m^2}_{(m\pm n)\equiv 0} - \underbrace{f_{m+1}f_{m-1}f_n^2}_{(n\pm m)\equiv 0}$$

Finally we note that $f_{n+m}f_{m-n}$ has the same zeros with only a pole at 0. Hence they must be constant multiples of each other, but expanding around 0 we see that this constant is 1 showing that:

$$f_{n+1}f_{n-1}f_m^2 - f_{m+1}f_{m-1}f_n^2 = f_{n+m}f_{m-n}$$

By setting $(n, m) = (n+1, n)$ and $(n, m) = (n+1, n-1)$, obtain the following recursive formulas

$$f_{2n+1} = f_{n+2}f_n^3 - f_{n+3}^3 f_{n-1} \tag{13}$$

$$\wp' f_{2n} = f_n \left(f_{n+2}f_{n-1}^2 - f_{n-2}f_{n+1}^2 \right) \tag{14}$$

Now if E is defined over $\mathbb{N}$ we can use the addition formulas to show that f_1, f_2, f_3 and f_4 have integer coefficients. By applying the recursive formulas above we can inductively deduce that also f_n must have integer coefficient, proving the theorem. □

From (12) and (14) we can find an expression for $[n]P$ in terms of the division polynomials, i.e. if $P = (x, y)$ then

$$[n](x, y) = \left(x - \frac{f_{n+1}f_{n-1}}{f_n^2}, \frac{f_{n+2}f_{n-1}^2 - f_{n-2}f_{n+1}^2}{yf_n^3} \right) \tag{15}$$

5 Practical computational considerations

Many algorithms in computational number theory have complexity relying on the efficiency in calculating the exponent of an element in a group. The naive way of calculating g^N, $N \in \mathbb{Z}$ in a group could be done by simply multiplying the element with itself, requiring N operations. But there is a much better approach: *the binary ladder* with complexity $O(\log(N))$. Let us in the forthcoming text in the context of elliptic curves only consider abelian groups.

5.1 Binary ladder

A binary ladder expands N in the numeric base-2 (the binary base). If $N = N_n N_{n-1}, \ldots, N_0$, where N_i is i:th binary bit of N we can write the N recursively,

$$N(i) = 2N(i-1) + N_i$$

and then $N = N(n)$ moreover we have that,

$$N(i)g = 2N(i-1)g + N_i g$$

Using this representation of N it follows that each step in the recursion takes one double and one addition, thus calculating Ng can be done in $O(\ln(N))$ doublings and additions. This leads us to our first algorithm:

Algorithm 1 (Binary ladder)**.**
Usage: $\mathbf{G}$ is an abelian group, $g \in \mathbf{G}$, $N \in \mathbb{Z}^+$
Output: $N \cdot g \in G$
Complexity: $O(\ln(N))$
Python: listing 1

```
1: $g_0 = 1$
2: $g_1 = g$
3: for $j = \ln(N) - 1$ to $0$ do
4:     if $N_j = 1$ then
5:         $g_0 = g_1 + g_0$
6:     end if
7:     $g_1 = g_1 + g_1$
8: end for
9: return $g_0$
```

As we will see in section 6.3, for elliptic curve arithmetic doubling a point is generally less time-consuming than adding, especially in the Montgomery parametrization. We see that by ”replacing” additions with doublings we could make the ladder more efficient.

Note. If we are for example considering the binary ladder we see that replacing additions with doublings is the same as replacing ones with zeros in the binary expansion.

6 Different parametrization

For the interest of algorithm efficiency it should be emphasized that the computational aspects in implementing the elliptic curve arithmetic depends highly on the parametrization, and below is a listing of some options. For the mathematician they are all the same, but for a computational number theorist they will be very different!

- Affine coordinates
- Projective coordinates
- Montgomery coordinates.

6.1 Affine coordinates

Enough have been said about curves and elliptic curves to develop a set of algorithms to implement the group structure compuationally. Let $E(\mathbb{F}_p)$ be an elliptic curve with $p > 3$ and $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be points, not necessarily different, on $E(\mathbb{F}_p)$. Then if $E(\mathbb{F}_p)$ is defined by (2) the following algorithms defines a group law over $E(\mathbb{F}_p)$.

Algorithm 2 (Elliptic curve addition - affine addition)**.**
Input: $P = (x_p, y_p), Q = (x_q, y_q) \in E(\mathbb{F}_p)$
Output: $P + Q \in E(\mathbb{F}_p)$
Complexity: 2 multiplications, 7 additions and one field inverse
Python: listing 2

```
1: if $P = \mathcal{O}$ then
2:     return $Q$
3: end if
4: if $Q = \mathcal{O}$ then
```

```
 5:     return $P$
 6: **end if**
 7: **if** $x_p = x_Q$ **then**
 8:     **if** $y_P = y_Q$ **then**
 9:         $m = (3x_P^2 + a)/2y_P$
10:     **else**
11:         return $\mathcal{O}$
12:     **end if**
13: **else**
14:     $m = (y_Q - y_P)/(x_Q - x_P)$
15: **end if**
16: $x_{P+Q} = m^2 - x_P - x_Q$
17: $y_{P+Q} = m(x_P - x_{P+Q}) - y_P$
18: return $P + Q = (x_{P+Q}, y_{P+Q})$
```

Algorithm 3 (Affine inverse)**.**
Usage: $P = (x_p, y_p) \in E(\mathbb{F}_p)$
Complexity: $O(1)$
Python: listing 3

```
1: return $(x_P, -y_P)$
```

As seen in algorithm 2, addition requires one field inverse. This calculation is asymptotically slower than for example integer multiplication and it would be profitable if we could avoid this. By using projective coordinates, this is in fact possible.

Note. Usually we will in complexity analysis refer to addition as **A**, Multiply with **M** and Inverse with **I**.

Here are some *ideas* of how long time it takes for a modern computer in year 2006 to do various arithmetic calculations I will write some down here:

- **Addition:** Adding two 200-digits numbers can be done about 10^7 times in a few seconds.
- **Multiplication:** Multiplying the same numbers 10^7 will instead take roughly a minute.
- **Inverse:** Calculating the inverse in $\mathbb{F}_p$ for p of the same size can be done 10^6 times in a few seconds.

6.2 Projective coordinates

One major problem with Affine coordinates in a computational perspective is the fact that we need to calculate an inverse with the gcd-operation to evaluate the slope. This can be avoided by using projective coordinates and instead represent the same point in projective coordinates where that inverse is superfluous. Consider the Affine point $P = (x, y)$. Calculating $[2]P$ then involves calculating the slope m

$$m = \frac{3x^2 + a}{2y} \tag{16}$$

and $[2]P$ will be on the k^2-rational form $(\frac{g(x,y)}{h(x,y)}, \frac{v(x,y)}{w(x,y)})$. But this rational Affine expression can be mapped on the projective point

$$\left[\frac{g(x,y)}{h(x,y)} : \frac{v(x,y)}{w(x,y)} : 1\right] = [g(x,y) : v(x,y) : h(x,y)w(x,y)]$$

so we do not need to compute an inverse in $\mathbb{F}_p$!

This idea can be retired further, and a special case of the projective parametrization is the so-called Montgomery parameterization, which to this date is the fastest known.

6.3 Montgomery coordinates

As explained by Montgomery in [Mon] there is a special projective parametrization with very fast arithmetic properties, which exploits the fact that the X-coordinate contains all information about the Y-coordinate except for at most a sign (see (1)). For this reason it is in its original form only suitable for some specific applications.

To derive this parametrization and its algorithms lets consider the elliptic curve defined over $\mathbb{F}_p$ by the affine cubic equation:

$$y^2 = b_3x^3 + b_2x^2 + b_1x + b_0 \tag{17}$$

with $b_i \in \mathbb{F}_p$ and $b_3 \neq 0$. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points on $E(\mathbb{F}_p)$ with difference $P_- = P_1 - P_2 = (x_-, y_-)$ and sum $P_+ = (x_+, y_+)$ then it is rather easy to deduce a formula for x_-x_+

$$x_-x_+ = \frac{(x_1x_2 - 1)^2}{x_1x_2} \tag{18}$$

Further if $b_0 = 0, b_1 = b_3, b_2 = A/B$ and $b_3 = 1/B$ with $x = X/Z$, $y = Y/Z$ in (17) we obtain the (projective) elliptic curve,

$$E(k) : BY^2Z = X(X^2 + AXZ + Z^2) \tag{19}$$

(which is well-defined only if $B \neq 0$ and $A \neq \pm 2$). On this curve we can find an expression for the projective coordinates X_+ and Z_+:

$$X_+ = Z_-(X_1X_2 - Z_1Z_2)^2 \tag{20}$$
$$Z_+ = X_-(X_1Z_2 - Z_1X_2)^2 \tag{21}$$

and the following formula for $[2]P_1 = (X_2 : Z_2)$, given by Montgomery:

$$X_2 = (X_1^2 - Z_1^2)^2 \tag{22}$$
$$Z_2 = 4X_1Z_1\big((X_1 - Z_1)^2 + \frac{A+1}{4}(4X_1Z_1)\big) \tag{23}$$

By (20) and (21), X_+ and Z_+ can be calculated with 6 multiplications if their difference X_-, Z_- is known. Further, we can double P_1 in only 5 multiplications if $(A+1)/4$ is known.

Note. We will only write $[X : Z]$ to denote a projective point with Montgomery parametrization because the Y coordinate is not needed in this choice of arithmetic.

Algorithm 4 (Montgomery add).
Input: $P = (X_1 : Z_1)$, $Q = (X_2 : Z_2)$ and $R = P - Q = (X_- : Z_-) \in E(\mathbb{F}_p)$
Output: $P + Q \in E(\mathbb{F}_p)$
Complexity: $8\mathbf{M} + \mathbf{2A}$
Python: listing 4

```
1: $X_3 = Z_-(X_1X_2 - Z_1Z_2)^2$
2: $Z_3 = X_-(X_1Z_2 - Z_1X_2)^2$
3: return $(X_3 : Z_3)$
```

Algorithm 5 (Montgomery double).
Input: $P = (X_1 : Z_1) \in E(\mathbb{F}_p)$
Output: $[2]P \in E(\mathbb{F}_p))$
Complexity: $7\mathbf{M} + \mathbf{4A}$
Python: listing 4

```
1: $X_2 = (X_1^2 - Z_1^2)^2$
2: $Z_2 = 4X_1Z_1\big((X_1 - Z_1)^2 + \frac{A+1}{4}(4X_1Z_1)\big)$
3: return $(X_2 : Z_2)$
```

Algorithm 6 (Montgomery ladder).
Input: $P = (X_1 : Z_1) \in E(\mathbb{F}_p)$ and $N \in Z$
Output: $[N]P \in E(\mathbb{F}_p)$
Complexity: $O(log(N))$
Python: listing 4

```
1: if $n = 0$ then
2:     return $\mathcal{O}$
3: end if
```

```
4: if $n = 1$ then
5:     return $P$
6: end if
7: if $n = 2$ then
8:     return $[2]P$
9: end if
10: $Q = P$
11: $R = [2]P$ //Uses algorithm 5
12: for $j = \text{nbits}(n) - 2$ to $0$ do
13:     if $n_j = 1$ then
14:         $Q = R + Q : P$ //Uses algorithm 4 with parameter $P = (X_-, Z_-)$
15:         $R = [2]R$ //Uses algorithm 5
16:     else
17:         $R = Q + R : P$ //Uses algorithm 4 with parameter $P = (X_-, Z_-)$
18:         $Q = [2]Q$ //Uses algorithm 5
19:     end if
20: end for
21: return $Q$
```

Note: nbits(n) is the number of bits in an integer n and n_j is the j:th bit of n.

7 Finding the order

Finding all points of an elliptic curve $E(\mathbb{F}_p)$ is quite easy if p is small, we just verify which tuples in the cartesian product $\mathbb{F}_p^2$ satisfy the elliptic equation [CrP, p. 350-359].

Algorithm 7 (Trivial method)**.**
Input: An elliptic curve $E(\mathbb{F}_p)$
Output: $\#E(\mathbb{F}_p)$
Complexity: $O(p^2)$
Python: listing 5

```
1: $k := 1$ //Include $\mathcal{O}$
2: for all $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ do
3:     if $y^2 = x^3 + ax + b$ then
4:         $k := k + 1$
5:     end if
6: end for
7: return $k$
```

Implementing algorithm 7 is simple and requires no overhead in terms of precalculations, or any significant dependencies on hard-to-write code - but as usual simplicity has its price on speed. The trivial method requires $O(p^2)$ operations (one loop through $\mathbb{F}_p$ for each element in $\mathbb{F}_p$).

Another simple algorithm for calculating $\#E$ is the Jacobi method and follows directly from section 4.2.1.

Algorithm 8 (Jacobi method to calculate $\#E(k)$).
Usage: For $E(\mathbb{F}_p)$ with $p \in [3, 10^7]$
Input: An elliptic curve $E(\mathbb{F}_p)$
Output: $\#E(\mathbb{F}_p)$
Complexity: $O(p \ln^2(p))$
Python: listing 6

```
1: $k := 1$ //Include $\mathcal{O}$
2: for all $x \in \mathbb{F}_p$ do
3:     if $\left(\frac{x^3+ax+b}{p}\right) = 1$ then
4:         $k := k + 1$
5:     end if
6: end for
7: return $k$
```

The Jacobi method obviously scales much better because we can evaluate $\left(\frac{a}{p}\right)$ in only $O(p \ln^2(p))$ operations - we can calculate almost the double amount of digits! But in cryptographical calculations we are faced with calculating the order of elliptic curves, $E(\mathbb{F}_p)$ with $p > 10^{40}$ (about 128-bit number). In our next section, we lay down a method capable of this.

7.1 Schoof's method

René Schoof published his paper [Sch] 1985 in which he revolutionized the efficiency of calculating $\#E$ over a finite field. The algorithm itself is quite short and concise, but the actual implementation contains most of the basic algorithms in algorithmic number theory.

Schoof's idea is to calculate $\#E(\mathbb{F}_p) \pmod{l}$ for many small primes l and then finally use the chinese remainder theorem to combine the results. In order to understand how this is done we need to introduce the *Frobenius endomorphism* and then finally find an application for our beloved *division polynomials* (see section 4.2.3).

7.1.1 The Frobenius endomorphism

Let $E(\mathbb{F}_p)$ be an elliptic curve, over which we have a group of endomorphisms, $\mathrm{End}(E)$ (i.e., homomorphism from a group to itself, $\mathrm{End}(E)$ always contain $\mathbb{Z}$). A non-trivial endomorphism in this group is the so-called Frobenius endomorphism:

$$\Phi_p : (x, y) \mapsto (x^p, y^p) \tag{24}$$

It is easy to see that this map restricted to $E(\mathbb{F}_p)$ is the identity (Fermat's little theorem), and for that reason also an automorphism (isomorphism from a group to itself), but it is non-trivial that it actually defines a automorphism on $E(\overline{\mathbb{F}}_p)$ (there is no p-th root of unity in $\overline{\mathbb{F}}_p$ and actually $x^{p-1} = 1$ if $x \in \overline{\mathbb{F}}_p$). *But why is it important to consider the algebraic closure?*

The Frobenius endomorphism also satisfies the quadratic equation $x^2 - a_p x + p$

$$\Phi_p^2(P) - [a_p]\Phi_p(P) + [p]P = \mathcal{O} \tag{25}$$

(see [CrP, p. 352]) for all $P \in E(\overline{\mathbb{F}}_p)$ and especially for $P \in E(\overline{\mathbb{F}}_p)[n]$.

7.1.2 Division polynomials and Schoof's method

Let $E(\mathbb{F}_p)$ be an elliptic curve defined by definition 5 and $\{f_n\}$, $n > 0$ be the set of division polynomials, depending on E. Each f_n has $\deg(f_n)$ number of roots and all of them corresponds to a n-torsion point of the elliptic curve. The problem is that not all roots of f_n are defined over $\mathbb{F}_p$. Thus we must consider the finite extension of $\mathbb{F}_p$ with respect to the roots of f_n. Mathematically we do this by considering $\overline{\mathbb{F}}_p$, the algebraic closure of $\mathbb{F}_p$. But computationally this is impossible, since $\overline{\mathbb{F}}_p$ is uncountable.

In the next section we will see *why we need all n-torsion points* and also *how we can calculate with them (without really computing them)*, to finally explain Schoof's algorihtm.

7.1.3 Schoof's method explained

Combining these two tools we can finally explain the beautiful algorithm.

Restricting (25) to $E(\mathbb{F}_p)[n]$, the following hold:

$$\Phi_p(P)^2 - [a_p \bmod n]\Phi_p(P) + [p \bmod n]P = \mathcal{O} \tag{26}$$

Because for elements $P \in E(\overline{\mathbb{F}}_p)[n]$, if $k = k' + ln$, $k, l \in \mathbb{Z}$, we have that

$$[k]P = [k']P + [ln]P = [k']P$$

motivating why it is possible to reduce our original equation mod n.

Now if $n = \ell$ is a prime number we define $\mathbb{F}_p[x,y]/(f_n(x), x^3 + ax + b - y^2) = \mathbb{T}_{n,p}$, the finite field $\mathbb{F}_p$ extended with the roots of f_n and with elements on the elliptic curve. Computationally this means that considering points in the finite extension field $\mathbb{T}_{n,p}$ is the same as computing with polynomials.

Rewriting (26) with $P = (x, y) \in \mathbb{T}_{n,p}$ we have,

$$(x^{p^2}, y^{p^2}) + [p \bmod n]P = [a_p \bmod n](x^p, y^p) \tag{27}$$

Note. The extension field $\mathbb{T}_{n,p} \neq E[n]$ but by definition x is a root to f_n and $y^2 = x^3 + ax + b$ - thus x defines a n-torsion point (except for the sign of y), motivating (27).

Now we can pin-point the essence of Schoof's algorithm: Everything in (27) is known except a_p, but we find it by trial and error!

Using the chinese remainder theorem, after finding a_p for sufficiently many ℓ, we can determine a_p modulo $\prod \ell$. One might think that it is necessary to find a_p up to modulo in order p, but by Hasse's theorem

$$|a_p| \leq 2\sqrt{p} \tag{28}$$

and it follows that it is enough to evaluate a_p up to $2\sqrt{p}$.

For $\ell = 2$ it is possible to do better, in terms of speed. A 2-torsion point correspond to points on $E(\mathbb{F}_p)$ where $y = 0$ (this can be seen either geometrically or for example by expanding $[2](x, y)$ as in (29)). Plugin $y = 0$ in (2)

$$0 = x^3 + ax + b \tag{29}$$

that is, a point $(x, y = 0) \in E(\mathbb{F}_p)[2]$ must be a root to that equation over $\mathbb{F}_p$. To check if any such roots exist it is enough to recall that $x^p - x = 0$ is satisfied if and only if $x \in \mathbb{F}_p$. It then follows that if there exist $x \in \mathbb{F}_p$ such that (29) is true then the following holds:

$$\gcd(x^p - x, x^3 + ax + b) \neq 1 \tag{30}$$

Now, if this equation holds then $E(F_p)$ has a non-trivial 2-torsion subgroup and then $2|\#E$ thus $a_p \equiv 0 \pmod 2$.

Algorithm 9 (Schoof's method)**.**
Usage: For $E(\mathbb{F}_p)$ with $p \in [10^5, 10^{100}]$
Complexity: $O(\ln^8(p))$
Python: listing 7
Precalculations: An optimal set of primes L s.t. a_p can be uniqely calculated.

```
1:  $K = \{\emptyset\}$ //Set of equations: $a \equiv b \pmod{\ell}$
2:  **for all** $l \in L$ **do**
3:     **if** $x = 2$ **then**
4:        **if** $\gcd(x^p - x, x^3 + ax + b) = 1$ **then**
5:           $K = K \cup \{a_p \equiv 0 \pmod 2\}$
6:        **else**
7:           $K = K \cup \{a_p \equiv 1 \pmod 2\}$
8:        **end if**
9:     **else**
10:       $u(X) = x^p \pmod{\Psi_l}$
11:       $v(X) = (x^3 + ax + b)^{(p-1)/2} \pmod{\Psi_l}$    $//= y^{p-1} \pmod{\Psi_l}$
12:       $P_0 = (u(x), yv(x))$    $//P_0 = (x^p, y^p)$
13:       $P_1 = (u(x)^p, yv(x)^{p+1})$    $//P_1 = (x^{p^2}, y^{p^2})$
14:       $P_2 = [p \pmod 2](x, y)$
15:       **if** $P_1 + P_2 = \mathcal{O}$ **then**
16:          $K = K \cup \{l, 0\}$    $//a_p \equiv 0 \pmod l$
17:          **next**
18:       **else**
19:          $Q = P_0$
20:          **for all** $k \in [1, l/2]$ **do**
21:             **if** $x(P_1 + P_2) = x_Q$ **then**
22:                **if** $y(P_1 + P_2) = y_Q$ **then**
23:                   $K = K \cup \{a_p \equiv k \pmod l\}$
24:                **else**
25:                   $K = K \cup \{a_p \equiv l - k \pmod l\}$    $//P_1 + P_2 = -Q$
26:                **end if**
27:             **end if**
28:             $Q = Q + P0$ $//Q = [k]P_0$
29:          **end for**
30:       **end if**
31:    **end if**
32: **end for**
33: **return** unique $a_p$ for which all equations in $K$ are satisfied (using CRT).
```

8 Factorization

Let us begin this section with a continuation of the first quote in the introduction, this time by Lenstra:

> ”Until recently, the subject of primality testing and factorization was not taken seriously by most mathematicians. Nowadays, a change in this attitude is noticeable. Partly, this change is due to the introduction of more sophisticated mathematical techniques than were used before. Indeed, the use of elliptic curves, which is the main topic of this lecture, has been referred to as the first application of 20-th Century mathematics to the problem of prime factor decomposition.” - H. W. Lenstra, Jr. [Len2]

The most basic theorem in arithmetic acts as the origin of the foundation for this very chapter.

Theorem 5 (The fundamental theorem of arithmetic)**.** *Every positive integer $N > 1$ can be written as a product of primes, and beside from permutations of the prime-numbers this representation is unique.*

Proof. See [Gio, p. 10]. □

It is now, given a number N, natural to ask whether we can find this unique representation - this is called the factoring problem.

Definition 6 (The integer factorization problem)**.** Given a positive integer N, find all prime factors. That is write

$$N = p_1 p_2 \cdots p_m$$

where p_i is not necessarily distinct.

Even though this problem sounds trivial, for example $667 = 23 \cdot 29$, something you easily do in your head it is far from obvious when the number is bigger, try 999983 for example! Did you fail? *Hint: Is it prime?*

It is interesting to note that even if the exact difficulty of the factorization problem is not known, there is no mathematical foundation for the belief that factoring is a hard problem. But in fact, on the other hand no-one has found any suggestions that it is not!

There is an active research in the area of quantum computing which theoretically predicts that it should be possible to solve the prime factorization problem on a Quantum computer in polynomial time using Shor's algorithm [Sho]. Only time will tell if this theory is practically possible. See for example [CrP, p. 418-424] or [Eke] for more on this topic.

8.1 Factorization methods

The first method to solve the factorization problem is trial division. It tries to divide an integer N with all positive integers $k \leq \sqrt{N}$.

Algorithm 10 (Trial division).
Usage: $N \in \mathbb{Z}^+$
Complexity: $O(\sqrt{N})$
Python: listing 8

```
1: for all $k \in [2, \sqrt{N}]$ do
2:     while $N \equiv 0 \pmod{k}$ do
3:         $N = N/k$
4:         output: $k$.
5:     end while
6: end for
```

This algorithm is deterministic and will not fail, but it requires $O(\sqrt{N})$ operations, and as N grows the allocation of such amount of compuational power is unfeasible on even the best supercomputer. Note that for small N it is an excellent algorithm, on a modern computer (2006) we can expect to find factors in order of about size 10^9 in roughly a minute. But can we do better?

Lets begin describing one of the most simple non-trivial algorithm, Pollard $p-1$.

8.2 Pollard $p-1$

Let as usual N be a composite integer, and assume p is an unknown prime divisor to N. Choose $a \in \mathbb{Z}/N\mathbb{Z}$. Then if $a^t \equiv 1 \pmod{p}$ it is quite likely that $\gcd(a^t - 1, N)$ is a non-trivial factor of N.

To explain how this works let us take a look at the group $\mathbb{Z}/N\mathbb{Z}$,

$$\mathbb{Z}/N\mathbb{Z} \simeq \mathbb{Z}/p_1^{\alpha_1}\mathbb{Z} \times \mathbb{Z}/p_2^{\alpha_2}\mathbb{Z} \times \cdots \times \mathbb{Z}/p_l^{\alpha_l}\mathbb{Z}$$

Assume that we have found a t such that $p_i - 1 | t$. Then, by Fermat's little theorem, we have that $a^t \equiv 1 \pmod{p_i}$ hence $\gcd(a^t - 1, n)$ is a non-trivial factor of N.

If t is constructed as

$$t = \prod_p p^{\frac{\ln B}{\ln p}}$$

where the product is taken over all primes $p \leq B$, then we are guaranteed that $p_i - 1 | t$ if $p_i - 1$ is B-smooth.

The see when we will find a non-trivial factor let us consider two scenarious

- $p_i - 1$ is B-smooth for all i, thus $a^t \equiv 1 \pmod{N}$ and $\gcd(a^t - 1, n) = n$.
- When $a \in \mathbb{F}_p$ have a finite order being B-smooth it may happen that $a^t \equiv 1 \pmod{p_i}$ even though $p_i - 1$ is *not* B-smooth.

If we are faced with the first condition we simply try another, smaller B.

The second scenario is actually good for us, because if we find such an a it is very likely that this a won't have the same order in *all* groups $\mathbb{Z}/p_i\mathbb{Z}$ that is B-smooth, thus we will actually succeed with somewhat better probability (however, observe that the best known algorithm for finding the order of an element in $\mathbb{Z}/N\mathbb{Z}$ depends on the non-trivial factorization of $\phi(N)$).

Algorithm 11 (Pollard $p - 1$)**.**
Usage: $N \in \mathbb{Z}^+$
Complexity: $O(\ln^3 N)$
Python: listing 9

```
1: $s = \gcd(a^{\mathrm{lcm}(B)-1}, N)$
2: **if** $1 < s < N$ **then**
3:     return $s$
4: **end if**
5: return "FAIL"
```

Note. As of this algorithm only the pseudo-code to find the first factor (not necessarily prime) is included. The extension into finding all factors is the same as in algorithm 10.

The implementation above is a probabilistic algorithm, depending on the probability that $p_i - 1$ is B-smooth, for $p_i | N$. For fix k the algorithm will take

$O(\ln^3 N \cdot k \ln\ln k)$ operations since the gcd operation is of order $O(\ln^2 N)$ and binary exponentiation is $O(\ln N)$. Calculating lcm (least common multiple) requires $O(k \cdot \ln\ln k)$ operations with the sieve of Eratosthenes. If assuming constant k (not depending on N) we get that each iteration takes $O(\ln^3 N)$ operations.

It is now clear that if we are unlucky and no prime divisor has smooth order (section 8.3), the Pollard $p-1$ algorithm will fail. Unless we're lucky and find a, p_i with $\operatorname{ord}(a, p_i)$ small. However, this is unlikely.

Let us continue our adventure by looking at some more general ideas behind modern factoring algorithms.

8.3 Smooth numbers

As seen above, the structure of a numbers' prime composition is of great importance for factorization algorithms. For example a random integer is expected to have one large prime factor and a couple of small ones. For some integers, it may be so that they only have small prime factors. And some have few large. It is obvious that the factorization of those different number have different complexity.

Definition 7 (B-smoothness)**.** A positive integer n is called B-smooth if none of its prime factors exceeds B.

Let further $\psi(k, B)$ be the number of B-smooth numbers less than n. Then the probability that a random positive integer in $[1, k]$ is B-smooth is $\psi(k, B)/k$,

And as we saw in Pollard $p-1$ and shall see in the elliptic curve method, those numbers play a fundamental role in the theory of factorization.

Theorem 6 (Probability for smoothness)**.** *The probability that a random integer $k \in [1, x]$ is $x^{1/u}$-smooth, is about u^{-u}.*

Proof. See [Can]. □

8.4 Ideas of factorization

First let us briefly summarize two common methods to factor integers, in fact those two constitutes the foundation for all factorization algorithms currently known:

1. Brute force methods (trial divisions with various modifications).
2. Finding ”congruence collisions”.

The first was described above so let us take a look at the second . Let say we have found N to be the difference of two squares $N = x^2 - y^2$ then $y^2 \equiv x^2 \pmod{N}$. Then if $x \equiv y \pmod{p_i}$ and $x \not\equiv y \pmod{N}$ we have that $x - y$ is a non-trivial factor of N (not necessarily prime).

8.5 The general method

A method for generating algorithms can be described more generally: If the following two properties hold for a group then an algorithm for factoring integers can be created.

Let $\mathbf{G}(N)$ be a group defined ”with respect to some integer N” and suppose there is some homomorphism $\Phi : \mathbf{G}(N) \longrightarrow \mathbf{G}(p)$, p being prime dividing N, but not necessarily known (this is not quite enough, $\mathbf{G}(N)$ and $\mathbf{G}(p)$ must be ”naturally” defined, for example defined through polynomials or rational functions). Especially we need to be able to split $\mathbf{G}(N)$ using the chinese remainder theorem. If we have found x and y in $\mathbf{G}(N)$ with $x \neq y$ such that $\Phi(x) = \Phi(y)$, then a non-trivial factor of N can be found. How? Let us make some examples.

Let’s clarify this with an example,

Example 3 (Fermat method)**.** Let $\mathbf{G}(7 \cdot 5) = \mathbb{Z}/35\mathbb{Z}$ and $x = \lceil\sqrt{35}\rceil = 6$, if $y = 1$ then $x + y = 6 + 1 = 0 \pmod{7}$ but $6 + 1 = 7 \pmod{7 \cdot 5}$ and we have $\gcd(6 + 1, 35) = 7$, a non-trivial factor of 35. We also have that $x^2 - y^2 = 35$.

And finally let us have a look at another example:

Example 4 (Pollard $p - 1$ (last step))**.** Let $\mathbf{G}(7 \cdot 5) = (\mathbb{Z}/35\mathbb{Z})^*$ be a multiplicative group. Also let $x = 2^4$, $y = 1$. We could for example let $\Phi : (\mathbb{Z}/35\mathbb{Z})^* \longrightarrow (\mathbb{Z}/5\mathbb{Z})^*$, then $x \neq y \pmod{35}$, but $\Phi(x) = \Phi(y)$. And we can find the factor by $\gcd(16 - 1, 35) = 5$. See algorithm 11 for similarities.

8.6 Elliptic curve method

Using the ideas from Pollard $p-1$ in the context of elliptic curves we can explain the elliptic curve method (ECM) for factoring integers. First we need to introduce elliptic curves over a composite modulo.

8.6.1 Elliptic Curves over $\mathbb{Z}/N\mathbb{Z}$

It is useful to present some idea of how the elliptic curve $E(\mathbb{Z}/N\mathbb{Z})$ "look like" when N is not prime.

Definition 8. Let N be a positive integer coprime to 6. We define the elliptic curve $E(\mathbb{Z}/N\mathbb{Z})$ (called elliptic pseudo-curve) as the projective curve defined by

$$E_{a,b}(k) : ZY^2 = X^3 + aZ^2X + bZ^3$$

for $a, b \in \mathbb{Z}/N\mathbb{Z}$ and $4a^3 + 27b^2$ is invertible modulo N.

The group structure is preserved by the chinese remainder theorem (because the curve is defined by a polynomial), so

$$E(\mathbb{Z}/N\mathbb{Z}) \cong E(\mathbb{Z}/p_1^{\alpha_1}\mathbb{Z}) \times \cdots \times E(\mathbb{Z}/p_k^{\alpha_k}) \tag{31}$$

Here $N = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$. But in affine coordinates the group contains a little bit more complicated structure when N is composite. Let us consider a point $P = [X : Y : Z] \in E(\mathbb{Z}/N\mathbb{Z})$. Either $\gcd(Z, N) = 1$ and there exist an affine representation of P. But if $\gcd(Z, N) > 1$ there is no affine representation of P (when we reduce it modulo some p, $p|N$ we will get a point in $E(\mathbb{Z}/p\mathbb{Z})$ that is the point at infinity).

If we define the affine group arithmetic in $E(\mathbb{Z}/N\mathbb{Z})$ we will have to "secretely" add some points at infinity whenever the group arithmetic fail (because some elements are not invertible). If we instead define the group arithmetic in projective coordinates everything will work out just fine (the group law is correct [Coh, p. 477-479]).

However, for the purpose of factoring we actually welcome this complication! We are actually only interested in such points P such that the Z-coordinate shares common divisor with N (for which the affine arithmetic fails or $\gcd(Z, N) \neq 1$). Because for such points we have found a non-trivial factor of N! Let us consider an example of this:

Example 5. Let $N = 10$ and define the affine curve $E(\mathbb{Z}/10\mathbb{Z})$ by,

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

then it contains the following points,

$$\begin{gathered}(0,1),(2,1),(3,1),(4,3)\\(4,7),(5,1),(7,1),(8,1)\end{gathered}$$

Now if we try to add $P = (0,1)$ and $Q = (2,1)$ in $E(\mathbb{Z}/N\mathbb{Z})$ we end up with a division by zero since the sum involves calculating $(2y_P)^{-1} = 2^{-1}$ in $\mathbb{Z}/10\mathbb{Z}$ which is not defined (as $\gcd(2,10) > 1$). What has happened is that the canonical reduction mod 2 of $P+Q$ into $P+Q \in E(\mathbb{Z}/2\mathbb{Z})$ is not an affine point - that is $P+Q$ is the point at infinity in $E(\mathbb{Z}/2\mathbb{Z})$.

In more general terms we did the following: In $E(\mathbb{Z}/N\mathbb{Z})$ we have that $P \neq -Q$, but when reducing mod 2, let $\Phi : E(\mathbb{Z}/N\mathbb{Z}) \to E(\mathbb{Z}/2\mathbb{Z})$. It follows that $\Phi(P) = (0,1)$ and $\Phi(Q) = (0,1)$. Becuase $-(0,1) = (0,1)$ we have that $\Phi(P) = \Phi(-Q)$ - and a non-trivial factorization can be found. Please note that the actual reduction is *not* necessary because the affine arithmetic will simply fail. If we work with projective coordinates we can get similar results by checking if $\gcd(N, Z) > 1$. Finding any such point is exactly what ECM to do.

8.6.2 Algorithm explaination

Let us take a look how H. Lenstra algorithm [Len] exploits the elliptic curves defined over $E(\mathbb{Z}/N\mathbb{Z})$, where N is a composite integer, to create a factorization method completely analogues to the $p-1$ method.

Let $P \in E(\mathbb{Z}/N\mathbb{Z})$, this point can be reduced into each one of $E(\mathbb{Z}/p_i\mathbb{Z})$ by simply reducing modulo p_i. If one finds an integer B such that $\#E(\mathbb{Z}/p_i\mathbb{Z})$ divides B for exactly one i, then $[B]P = \mathcal{O}$ in $E(\mathbb{Z}/p_i\mathbb{Z})$ but not in $E(\mathbb{Z}/N\mathbb{Z})$. (Otherwise P would generate a sub-group with order strictly bigger than $\#(\mathbb{Z}/p_i\mathbb{Z})$ which is impossible). This mean that the computation will fail in Affine coordinates (similar to example 5), or if we use projective coordinates the Z-coordinate will have a common factor with N - both will result in a non-trivial factor of N. The advantage of this method compared to Pollard

$p-1$ is that we can choose another curve very easy, and hope that this new curve has order, $\#E$ that is B-smooth.

Algorithm 12 (Elliptic Curve Method (Affine))**.**
Usage: $N \in \mathbb{Z}$
Python: listing 10
Precalculations: A list L of all primes up to B.

```
1: $B = 1000$ //Or some other practical limit
2: $m = \prod_{i=0}^{k} p^{\frac{\ln B}{\ln p}}$
3: Create a random curve $E$ and a random point $P \in E$
4: if $[m]P$ Failed then
5:     Catch element $g$ whose inverse was undefined.
6:     return $\gcd(g, N)$
7: else
8:     goto 2
9: end if
```

Note. When doing calculations in projective coordinates the arithmetic will never fail, for this reason we must have another way of finding a non-trivial factor. To do this we calculate $\gcd(Z, N)$ where Z is the projective Z coordinate (see section 3.2).

9 Primality proving

Trial division (see algorithm 10) can of course be used to test small numbers for proving primality, but for larger numbers there are better methods.

There is a method due to ideas of E. Lucas, from 1876.

Theorem 7 (Lucas theorem)**.** *If a, N are integers with $N > 1$, and*

$$a^{N-1} \equiv 1 \pmod{N}$$

but $a^{(N-1)/q} \not\equiv 1 \pmod{N}$ for every prime $q|N-1$, then N is prime.

Proof. See [CrP, p. 173]. □

Again (analogues to Pollard $p-1$) the algorithm depends on the smoothness of $N-1$, something that is very improbable for large N. However, as in ECM we can get around this by using elliptic curves.

Theorem 8 (Goldwasser-Kilian)**.** *Let $N > 1$ be a natural number and* $\gcd(6, N) = 1$, *and let K, m be natural numbers with $K|m$. Now consider the* *elliptic pseudo-curve*[1] $E(\mathbb{Z}/N\mathbb{Z})$. *Assume there exist a point* $P \in E(\mathbb{Z}/N\mathbb{Z})$ *s.t.* $[m]P$ *is well-defined and moreover,*

$$[m]P = \mathcal{O}$$

For all prime q *dividing* K *we can carry out the curve operations to find,*

$$[m/q]P \neq \mathcal{O}$$

Then for every prime p *dividing* N*,*

$$\#E(\mathbb{F}_p) \equiv 0 \pmod{K}$$

In particular if also $K > (N^{1/4}+1)^2$ *then* N *must be prime!*

Proof. Let p be a prime factor of N. Because $[m/q]P \neq \mathcal{O}$ we have that $[K]P \neq \mathcal{O}$. But because $K|m$ we have that K must divide the order of P and then also the order of the group. If further $K > (N^{1/4}+1)^2$ then $\#E(\mathbb{F}_p) > (N^{1/4}+1)^2$ and Hasse theorem 3 implies that $\#E(\mathbb{F}_p) < (p^{1/2}+1)^2$. We conclude from the two relations that $p^{1/2} > N^{1/4}$ or equivalently $p > N^{1/2}$ for all primes p. As N has all its prime factors larger than its square root it must be prime. □

9.1 Certificates

If you consider the Goldwasser-Kilian theorem above you see that it ends with a relation "*if* $K > (N^{1/4}+1)^2$ *then* N *is prime*". Thus we could recursively store relations: $R_1 = (N, K_1), R_2 = (K_1, K_2), \ldots, R_i = (K_i, p)$. Primality for K_i follows from relation R_i and primality for K_{i-1} follows from R_{i-1} and R_i and so forth, recursively.

Because $K_i < K_{i-1}$ (at least a factor 2 smaller) the recursion will terminate quite fast. This chain of relations are called a prime certificate for N.

9.2 Elliptic Curve primality proving explained

Let us now use theorem 8 to create an elliptic curve prime proving algorithm.

[1]We're not quite sure N is prime until after the algorithm is done.

If m equals the order of some random elliptic pseudo-curve over $\mathbb{Z}/N\mathbb{Z}$ (calculated with for example Schoof's algorithm, see 7.1) then,

$$[m]P = \mathcal{O}$$

as required, assume we can find the factorization of m on the form $m = F \cdot K$ s.t. F is a product of small primes and K is a probable prime with $K > (N^{1/4}+1)^2$. If something failed we know N is composite! If the factorization could not be found, hit another curve!

But if we were lucky (it will happen fairly often) then we check that,

$$[m/K]P \neq \mathcal{O}$$

and we got a proof of primality for N.

Algorithm 13 (Elliptic Curve primality test).
Usage: Probable prime N

```
1: create a random pseudo-curve $E(\mathbb{Z}/N\mathbb{Z})$
2: $m = \#E$ //Through algorithm 9
3: if not possible to find a probable prime $K$ and integer $F$ s.t. $K \cdot F = m$ and
   $F > (N^{1/4}+1)^2$ then
4:    goto 1
5: end if
6: Find a point $P$ on $E$
7: $Q = [m/K]P$
8: if $Q = \mathcal{O}$ then
9:    goto 6
10: end if
11: if $[K]Q \neq \mathcal{O}$ then
12:    return $N$ is composite
13: end if
14: return $K$ is prime $\Rightarrow$ $N$ is prime
```

Note: *If any part of the algorithm fails (undefined, invalid etc) then output composite.*

10 Getting down to implementation

I choose Python for its simplicity and pseudo-like syntax. It has native support for large-integer multiplication (even if it is not that efficient) it made it possible for early trial-and-error approaches to get a feel for numerical algorithms in general. A simple Fermat primality test could be implemented in a few lines of code. And ECM, using montgomery coordinates, in about 50! Very impressive.

Everything was written with an object oriented way, with hierarchies behind common mathematical objects, the field class inherit group class and so forth.

The code itself is about 2000 lines long and includes plenty of tests cases where you can learn how it works. I think it is quite self-explanatory.

11 Ending words

The reader may now think that complete factoring of one integers is actually the only problem that concerns the factorization problem. But this is not true, sometimes, especially as an application to more complex factorization algorithms we are faced with a sub-problem: Given a set of random integers, find as many complete factored smooth numbers as possible. Thus we try to maximize (#factored numbers)/time instead of minimize the time to factor a given integer.

This last interesting aspect was investigated in Arizona Winter School [AWS], year 2006, under the supervision of D.J. Bernstein. Today we find these smooth numbers with for example the sieve of erastothenes, but it is very memory inefficient. A proposed better approach is to use for example ECM and trial division. Both algorithms are very memory efficient which opens up a new method where small embedded parallell computers are used to solve those problems.

12 Source Code Listing

Here you can find a subset of functions included in the source code for this thesis. The full Python module can be found at:
http://www.berlips.com/exjobb/field.tgz.

Some remarks on the syntax used in the code:

- x denotes an element and $x.G$ is the field/group containing x.
- G is a field/group with many properties, for example $x.G.one()$ could be used to get the identity in G. For more options, see *field.py*.
- $ZmodN$ is the group $\mathbb{Z}/N\mathbb{Z}$ (includes both the abelian and multiplicative structure).
- EC is an Affine elliptic curve group. Note: we use the convention $x = True$ denotes the point at infinity.

Listing 1: Group binary ladder (field.py)

```
# Binary ladder,
# calculates x^k
def __pow__(x, k):
  pow=x
  curr=x.G.one()          # Find the multiplicative identity
          # in the group G containing x

  while k!=0:
    if k&1:
      curr = curr*pow

    pow = pow*pow
    k = k>>1

  return curr
```

Listing 2: Elliptic curve Affine addition (field.py)

```
# Affine addition
# P = [x1,y1], Q=[x2,y2]
# P,Q elliptic points
def add(self, P,Q):
  # Calculate P+Q on an elliptic curve E
  # check for identity elements.
  if P == self.zero_:
    return Q
  if Q == self.zero_:
    return P
  x1,y1 = P
  x2,y2 = Q

  if P == Q:
    if y1.is_zero():
      return True
    # al(pha) is the tangent slope at P
    al = (3*x1*x1 + self.a)/(2*y1)
    x3 = al*al - 2*x1
    y3 =  al*(x1-x3)-y1
  else:
    if x1 == x2:
      return True
    # al(pha) is the slope of the line between P and Q
    al = (y2 - y1)/(x2 - x1)
    x3 = al*al - x1 - x2
    y3 =  al*(x1 - x3)-y1

  return [x3,y3]
```

Listing 3: Elliptic curve Affine inverse (field.py)

```
# Affine inverse
# P = [x,y]
def add_inv(self, P):
  return [P[0], -P[1]]
```

Listing 4: Elliptic curve Montgomery arithmetic (field.py)

```
# Montgomery arithmetic over (4a+10)y^2 = x^3 + ax^2+x
def ecmdouble(self,P):
  (x,d) = P
  return  (x*x-d*d)**2,  4*x*d*(x*x+self.a*x*d+d*d)
def ecmadd(self, P, Q):
  (x,d) = P
  (x1,d1) = Q
  return ( 4*(x*x1 - d*d1)**2,  8*(x*d1 - d*x1)**2 )
def mul(self, r, P):
  """ calculate r*P
  """

 Q = self.ecmdouble(P)
   bit = r.numdigits(2)
  for b in xrange(bit-2, -1,-1):
    if r.getbit(b):
      P = self.ecmadd(P,Q)
      Q = self.ecmdouble(Q)
    else:
      Q = self.ecmadd(Q,P)
      P = self.ecmdouble(P)

  return P
```

Listing 5: Elliptic curve trivial count for $\#E$ (field.py)

```
# Trivial count for elliptic curve (self).
# self.R with parameters self.a and self.b
#
# self.R.N is the cardinality of the field self.R
def trivial_count(self):
  R = self.R
  a = self.a
  b = self.b

  count=1   # include point at infinity

  for x in xrange(0,R.N):
    x = R(x)
    for y in xrange(0,R.N):
      y = R(y)
      if y*y == x*x*x + a*x + b:
        count+=1
  return count
```

Listing 6: Elliptic curve Jacobi-method for $\#E$ (field.py)

```
# Jacobi-count for an elliptic curve (self)
# The elliptic curve is defined over
# self.R with parameters self.a and self.b
#
# self.R.N is the cardinality of the field self.R
def jacobi_count(self):
  R = self.R
  a = self.a
  b = self.b

  count=1   # include point of infinity
  for x in xrange(0,R.N):
    x = R(x)
    ysqr = (x*x*x + a*x + b);
    if ysqr.is_zero():
      count+=1
    if ysqr.is_quadratic_residue() == 1:
      count+=2

  return count
```

Listing 7: Schoof's method for $\#E$ (field.py)

```
# Schoofs method calculating the order
# of an elliptic curve (self) defined over
# the field self.R
#
# Outputs all equations #E = k (mod l)
# on the form (k,l)
def schoof(self):
    R = self.R
    K = Poly(R)
    Y2 = K([self.b, self.a, 0, 1])
    K.quotient(Y2.x)
    X = K([0,1])

    h = X**R.N - X

  # [Check l=2]

    if (h&Y2).degree() != 1:
      print((2,1))
```

```
else:
    print((2,0))

prime_list = base.prime_generate(3, 1000)

# Find maximum prime number (l) needed:
prod = 2
n = 0
n4sqrt = 4*base.isqrt_greater(R.N)

for l in prime_list:
    if prod > n4sqrt:
        break
    prod *= l
    n+=1

del prime_list[n:]
psi = self.division_polynomials(K, l)

prod = 1

# [Check other prime numbers l in list]
for lidx in xrange(len(prime_list)):

    l = prime_list[lidx]
    psi_l = psi[int(l+1)]

    pt = R.N % l # reduced N modulo l
    pi = pt + 1  # only used for indexing

    K.quotient(K.make_monic(psi_l.x))

    ELC = EC(K, K([self.a]), K([self.b]))
    Y2 = K([self.b, self.a, 0, 1])
    X = K([0,1])

    u = X ** R.N
    v = Y2**((R.N - 1)/2)

    P0 = ELC([u,v])
    P1 = ELC([u**R.N, v**(R.N+1)])
```

```

        # P2 = (D/G, E/H)
        if pt % 2 == 0:
          D = X*(psi[pi]**2*Y2) - (psi[pi-1] * psi[pi+1])
          G = psi[pi]**2*Y2
          E = (psi[pi+2]*psi[pi-1]**2 -
               psi[pi-2]*psi[pi+1]**2)*K([~R(4)]) # note Y
          H = psi[pi]**3*Y2**2
        else:
          D = X*(psi[pi]**2) - Y2*(psi[pi-1] * psi[pi+1])
          G = psi[pi]**2
          E = (psi[pi+2]*psi[pi-1]**2 -
               psi[pi-2]*psi[pi+1]**2)*K([~R(4)]) # note Y
          H = psi[pi]**3


        # Add P2 + P1
        # P1 = [D, G, E, H]
        # P2 = [D', 1, E', 1]
        P12 = self.add_torsion_rational(
             [ P1.x[0], K([1]), P1.x[1], K([1]) ],
             [D,G,E,H], Y2)

        if P12 == True:
          print((l,0))
          continue

        (Dp, Gp, Ep, Hp) = P12

        P00 = [P0.x[0], K([1]), P0.x[1], K([1])]
        P03 = P00

        # Try all a_p:
        for k in xrange(1,l/2+2):
          if (P03[0]*P12[1] - P03[1]*P12[0]).is_zero():
            if (P03[2]*P12[3] - P03[3]*P12[2]).is_zero():
              print((l,k))
              break
            print((l,l-k))
            break
          P03 = ELC.add_torsion_rational(P03, P00, Y2)
```

Listing 8: Factorization - trial division (field.py)

```
# Trivial factorization of a positive integer N
# up to a bound B and using a precalculated list
# of primes 'primes'.
def factor_trial(N, primes=None, B=None):
  """ Returns smallest factors of a number using trial division
    factors are upper bound by B """
  factors = []
  if primes == None:
    primes = base.prime_generate(B)

  for p in primes:
    while N%p == 0:
      N = N/p
      factors.append(p)
  return factors
```

Listing 9: Factorization - Pollard $p-1$ (field.py)

```
# Tries to find a factor using the method of pollard p-1
# B : the least common multiple of the integers up to some
# bound, computed using lcm.
def factor_pmin1(N, B=None):
    for a in [2, 3, 5]:
        x = a**B
        g = gcd(x-1, N)
        if g != 1 and g != N:
            return g
    return N
```

Listing 10: Factorization - ECM (field.py)

```
# N is a positive integer to be factored
# B is the stage one bound
def factor_ecm(N, B=None):
  """ Lenstras algorithm for finding a factor in N,
  based on Elliptic curve arithmetics
  """
  if B==None:
    B = 10000
  C = 10
  R = ZmodN(N)
```

```

g6 =base.gcd(N,6)
if g6 != 1:
  return N/g6

# Generate prime list
primes = base.prime_generate(1000)

del primes[0:2] # remove p=2,3 from the list as we require gcd(N, 6)=1

# generate a a_i for each p_i s.t. p_i^a_i > B
pna = [] #pna = prime n alpha
for p in primes:
  pna.append([p, int(math.log(B)/math.log(p))])

while C>0:
  E = EC(R, 0,0)
  P=E.random_elt_curve()

  g = base.gcd(E.discriminant().x,N)
  if g==N: continue;
  if g>1: return g

  # Using affine coordinates
  for pa in pna:
    for j in xrange(pa[1]):
      try:
        P = pa[0]*P
      except ZeroDivisionError, g:
        return N/base.gcd(N, g.args[0])
```

References

[AWS] ”Arizona Winter School 2006”,
http://swc.math.arizona.edu/oldaws/06GenlInfo.html

[Can] E. Canfield, P.Erdös, and C. Pomerance. *”On a problem of Oppenheim concerning ”factorisatio numerorum””*. J. Number Theory, 17:1-28, 1983

[CrP] Crandall, R. and Pomerance, C. *”Prime numbers - A computational perspective (second edition)”* Springer Science+Business Media, Inc. (2005)

[Coh] Cohen, Henri *”A Course in Computational Algebraic Number Theory”* Springer-Verlag Berlin Heidelberg, 1993

[Eke] A. Ekert, R. Jozsa *”Shor's quantum algorithm for factorizing numbers”* Reviews of Modern Physics, pages 733–753, July 1996.

[Ful] Fulton, W. *”Algebraic Curves: An introduction to algebraic geometry”,* Addison-Wesley Publishing Co., Inc (1969).

[Gau] Gauss, C. F *”Disquisitiones Arithmeticae (1801) Article 329”*

[Gio] Gioia, Anthony A. *”The theory of numbers : an introduction”,* Dover edition (2001)

[Lan] Lang, S. ”Elliptic curves - diophantine analysis”, Springer-Verlag, New York Heidelberg Berlips (1978)

[Len] H. W. Lenstra, Jr. *”Factoring integers with Elliptic Curves” The Annarls of Mathematics*, 2nd Ser., Vol. 126, No.3 (Nov.,1987), 649-673

[Len1] H. W. Lenstra. *”Algorithms in algebraic number-theory”* Bulletin (new series) of the American Mathematical Society [0273-0979] LENSTRA year:1992 vol:26 iss:2 pg:211 -244

[Len2] H. W. Lenstra. *”Elliptic curves and number-theoretic algorithms”* Tech. Rep. Report 86-19, Math. Inst., Univ. Amsterdam, 1986.

[Mon] "An FFT extension of the Elliptic Curve Method of Factorization", PhD thesis, University of California, Los Angeles (1992)

[Sch] Schoof, R. *"Elliptic Curves Over Finite Fields and the Computation of Square Roots* mod *p"*, *Mathematics of computation,* Vol. 44, No. 170 (Apr., 1985), 483 – 494

[Sho] P. W. Shor in *Proc. 35th Annual Symposium on the Foundations of Computer Science, edited by S. Goldwasser (IEEE Computer Society Press, Los Alamitos, California, 1994), p. 124.*

[Sil86] Silverman, Joseph H. *"The Arithmetic of Elliptic Curves",* New York, Springer-Verlag Inc (1986).

[Wik1] Wikipedia (en) *"Affine geometry"* http://en.wikipedia.org/wiki/Affine_geometry.

[Wik2] Wikipedia (en) *"Projective geometry"* http://en.wikipedia.org/wiki/Projective_geometry.

[Wik3] Wikipedia(en) "Liouville's theorem (complex analysis)",http://en.wikipedia.org/wiki/Liouville%27s_theorem_%28complex_analysis%29

[Wol1] Weisstein, E. W. *"Books about Brownian Motion."* www.ericweisstein.com/encyclopedias/books/BrownianMotion.html.

[Wun] Wunsch, D. "Complex Variables with Applications (second edition), Addison Wesley Publishing Company Inc.