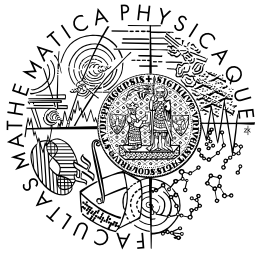


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Václav Zvoníček

Schoof's algorithm for Weierstrass curves

Department of Algebra

Supervisor of the bachelor thesis: prof. RNDr. Aleš Drápal, CSc., DSc.

Study programme: Computer Science

Study branch: General Computer Science

Prague 2023

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to express my deepest gratitude to Professor Aleš Drápal for his invaluable support and guidance during the process of writing my thesis. Professor Drápal's expertise in the field of elliptic curves and his patient guidance has been crucial to the success of this work. I am grateful for his constant support, encouragement, and insightful feedback. Thank you, Professor Drápal, for being an exceptional supervisor.

Title: Schoof's algorithm for Weierstrass curves

Author: Václav Zvoníček

Department: Department of Algebra

Supervisor: prof. RNDr. Aleš Drápal, CSc., DSc., Department of Algebra

Abstract: Schoof's algorithm is the starting point for the most efficient methods for determining the number of rational points on an elliptic curve defined over a finite field. The goal of this thesis is to introduce the subject of elliptic curves, with the emphasis on Weierstrass curves over a finite field, to describe Schoof's algorithm and its time complexity, and, finally, to implement it in C++ with the support of NTL. The implementation provides a user with a reasonably fast utility for determining the order of Weierstrass curves over finite fields of size up to 128 bits.

Keywords: elliptic curve, Weierstrass curve, Hasse's theorem, division polynomial, Schoof's algorithm

Contents

Introduction

Many cryptographic settings [1] and their proper configuration rely on the discrete logarithm problem in the group of an elliptic curve defined over a finite field (ECDLP). The reason for this is that the properties of elliptic curves allow these settings to reach an excellent level of security while keeping the amount of necessary resources low, in contrast to other techniques. According to [2], for example, using a 4096-bits key in the RSA protocol achieves the same level of security as a key of size about 7% of that in a setting based on elliptic curves.

The difficulty of the discrete logarithm problem (DLP) depends on the group in which it is considered. While the problem is trivial in $(\mathbb{Z}_n, +)$, breaking the DLP in $E(\mathbb{F}_q)$ is considered impossible for certain types of elliptic curves. It is obvious that choosing a good candidate for security-based systems based on ECDLP depends on the properties of the group $E(\mathbb{F}_q)$. Knowing its order is clearly on the list of properties we should be interested in, because once we know the order, we can make several conclusions about the difficulty of the ECDLP. For instance, due to Pohlig-Helmann [1], it is advised to use a prime-order subgroup of the group of an elliptic curve.

René Schoof published an algorithm [3] for finding the order of $E(\mathbb{F}_q)$ which runs in $\mathcal{O}(\log^8 q)$ time. This was a substantial enhancement at the time, however, it was later significantly improved by Atkin and Elkies. The resulting algorithm, called the SEA algorithm [2], runs in $\mathcal{O}(\log^6 q)$ time and thus makes the original algorithm obsolete. In spite of this, Schoof's original algorithm was groundbreaking as a solution to the problem of determining the order of $E(\mathbb{F}_q)$, and is therefore worth discussing in detail.

First three chapters of this thesis introduce the mathematical background essential to comprehend Schoof's algorithm, encompassing the definition of a Weierstrass curve, torsion groups, division polynomials, and elliptic curves over finite fields. Chapter 4 provides a full description of Schoof's algorithm along with its time complexity. Chapter 5 then presents our implementation of Schoof's algorithm in C++ with the support of NTL (A Library for doing Number Theory) [5].

1. Elliptic Curves

1.1 Weierstrass Curve

We start with the definition of an elliptic curve. From the algebraic point of view, an elliptic curve is a curve of genus one. This can be rephrased in a less abstract way by saying that such a curve is birationally equivalent to a smooth *Weierstrass curve* [4], which is a more suitable definition for our purposes, especially when addressing their arithmetic properties.

Definition 1.1.1. A Weirstrass curve E over a field K is the set

$$E = \{(x, y) \in \mathbb{A}^2 \mid y^2 + yg(x) = f(x)\} \cup \{\infty\},$$

where $g(x) = a_1x + a_3$ and $f(x) = x^3 + a_2x^2 + a_4x + a_6$ for some $a_1, \ldots, a_6 \in K$. The equation

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1.1}$$

is called the *Weierstrass equation*.

The reason for including the special point ∞ is based on the interpretation of an elliptic curve as a group and can be identified with the point at infinity in the projective plane, see below. The other points on the curve are called *affine points*. An affine point $(x, y) \in \mathbb{A}^2(K)$ is called *K-rational* or simply *rational* if the field K is clear from the context.

Now, though using the Weierstrass equation (1.1) is possible, the equation can be further simplified when $\mathrm{char}(K) \neq 2, 3$. Since we will be mostly interested in elliptic curves over finite fields of large characteristics, this assumption will not limit us at all.

Suppose $\mathrm{char}(K) \neq 2$ and let E be a Weierstrass curve given by (1.1). Rewrite it as

$$y^2 + g(x)y = f(x), \tag{1.2}$$

where $g(x) = a_1x + a_3$ and $f(x)$ is the polynomial on the RHS of (1.1). The LHS of (1.2) can be completed to a square as follows:

$$\left(y + \frac{g(x)}{2}\right)^2 = f(x) + \frac{g(x)^2}{4}.$$

Using the substitution $\tilde{y} = (y + g(x)/2)$, which is defined since $\mathrm{char}(K) \neq 2$, and after a little rearrangement, we obtain

$$\tilde{y}^2 = x^3 + a_2'x^2 + a_4'x + a_6'$$

for some new constants $a_2', a_4', a_6' \in K$. Therefore, a Weierstrass curve may be considered in the form $y^2 = f(x)$ for some monic cubic polynomial $f(x) \in K[x]$ whenever $\mathrm{char}(K) \neq 2$.

Moreover, if we assume $\text{char}(K) \notin \{2, 3\}$, then we can get rid of the term x^2 in $f(x)$. Here we are using the fact that cubing the expression $\tilde{x} = (x + a_2'/3)$ produces the term containing x^2. Thus

$$\tilde{y}^2 = \tilde{x}^3 + \left(a_4' - \frac{(a_2')^2}{3}\right)\tilde{x} + \left(a_6' + \frac{2(a_2')^3}{27} - \frac{a_2'a_4'}{3}\right).$$

In this way we have transformed (1.1) to the equation

$$E : y^2 = x^3 + Ax + B \tag{1.3}$$

for some $A, B \in K$. Equation (1.3) is sometimes referred to as the *short Weierstrass equation* of E.

Depending on A, B, the elliptic curve can have various properties. In this text, the most important property we will require is the *smoothness* of an elliptic curve.

Proposition 1.1.1. *A Weierstrass curve E given by $y^2 = f(x)$, where $f(x) \in K[x]$ is monic cubic polynomial, defined over a field K with $\text{char}(K) \neq 2$ is smooth if and only if f is separable.*

Proof. A Weierstrass curve is smooth at (x_0, y_0) if and only if not all partial derivatives vanish, namely when

$$\frac{\partial(y^2 - f(x))}{\partial x}(x_0, y_0) \neq 0 \quad \text{or} \quad \frac{\partial(y^2 - f(x))}{\partial y}(x_0, y_0) \neq 0$$

holds.

Let us find all points at which the curve is not smooth. The conditions about partial derivatives become

$$f'(x) = 0 \quad \text{and} \quad 2y = 0. \tag{1.4}$$

Now we see that E is not smooth at $(x_0, y_0) \in E$ if and if $y_0 = 0$ and $f'(x_0) = 0$. Since $(x_0, y_0) \in E$, the constraints imposed by (1.4) imply that E is smooth at every point if and only if f has no multiple roots in $\overline{K}$. □

Proposition 1.1.1 can be restated by using the discriminant of $f(x)$ [6]. Assume $\text{char}(K) \notin \{2, 3\}$, so $f(x) = x^3 + Ax + B$. Under this assumption, $f(x)$ is separable if and only if

$$4A^3 + 27B^2 \neq 0.$$

Example 1.1.1. The elliptic curve E given by $y^2 = x^3 - x + 1$ over the finite field $\mathbb{F}_{11}$ is smooth since $4 \cdot (-1) + 27 \neq 0$. In fact, it is smooth at all 9 points of the elliptic curve; the points are listed in Table 1.1.

In the rest of the text, we will only work with smooth elliptic curves given by a short Weierstrass equation, i.e. those given by (1.3) with the constraint $4A^3 + 27B^2 \neq 0$.

1.2 The Group of an Elliptic Curve

Once we have the set of points of an elliptic curve, it is natural to study this set from the algebraic point of view. In particular, there exists an operation on the set of points of an elliptic curve which, quite surprisingly, gives rise to an abelian group with the neutral element ∞. We shall now give a definition of this operation and derive some facts about it.

Definition 1.2.1. Let $P, Q \in E$ be affine points on an elliptic curve E, let $\ell \subseteq \mathbb{A}^2$ be the line through these points if $P \neq Q$, and the tangent line of E at P when P and Q coincide. If $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, the result of $P \oplus Q$ is then defined as follows:

(a) $(x_1 = x_2)$

- If $P \neq Q$ or $y_1 = y_2 = 0$, set $P \oplus Q = \infty$,
- otherwise there exists at most one point $T \in \ell \cap E, T \neq P$. If no such point T exists, set $T = P$. Then $P \oplus Q = R$, where $R \in E$ is the second point of intersection of the elliptic curve E and the vertical line through T.

(b) $(x_1 \neq x_2)$

- $(|\ell \cap E| = 3)$ let $T \in (\ell \cap E) \setminus \{P, Q\}$,
- $(|\ell \cap E| = 2)$ let $T \in \{P, Q\}$ be the point at which ℓ is the tangent line at T.

Then $P \oplus Q = R$, where $R \in E$ is the second point of intersection of the elliptic curve E and the vertical line through T if such R on E exists. Otherwise, set $R = T$.

(c) $R \oplus \infty = \infty \oplus R = R$ for all $R \in E$.

The definition can be rephrased by viewing an elliptic curve in the projective plane $\mathbb{P}^2$ with axes x, y and using the fact, which will be mentioned later, that, geometrically, $\infty = (0 : 1 : 0)$ represents a vertical direction – the affine lines passing through the point ∞ are precisely those parallel to the y-axis. Then, for some points P, Q on an elliptic curve E, $P \oplus Q$ can be thought of as the reflection of the point R over the x-axis, where the reflection of ∞ is again ∞. Figure 1.1 illustrates the result of $P \oplus Q$ when $P \neq Q$ and $P = Q$.

It turns out $(E(K), \oplus)$ yields a group:

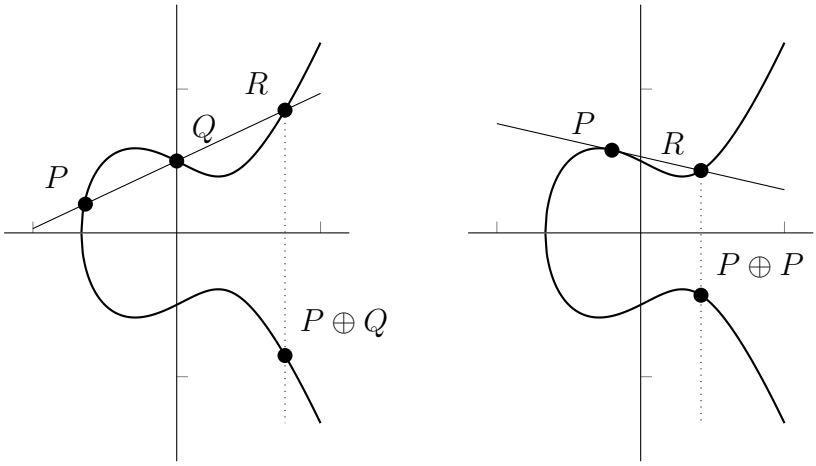
Theorem 1.2.1. *Let $E(K)$ be the set of K-rational points of an elliptic curve E over a field K together with ∞. Then*

(a) (Commutativity) $\forall P, Q \in E(K) : P \oplus Q = Q \oplus P$,

(b) (Neutral element) $\forall P \in E(K) : \infty \oplus P = P \oplus \infty = P$,

(c) (Inverses) $\exists(\ominus P) \in E(K) : P \oplus (\ominus P) = \infty$,

(d) (Associativity) $\forall P, Q, R \in E(K) : (P \oplus Q) \oplus R = P \oplus (Q \oplus R)$.



(a) Addition of two distinct points

(b) Addition of a point to itself

Figure 1.1: Addition of points on an elliptic curve

In other words, $(E(K), \oplus, \ominus, \infty)$ *forms an abelian group.*

Proof. The properties (a)-(b) follow immediately from the definition of $\oplus$. To prove (c), notice the inverse element of an affine point $(x, y) \in E(K)$ is $(x, -y) \in E(K)$, the inverse element of ∞ is again ∞. However, proving associativity of $\oplus$ is a rather tedious task when working with the formulas for addition, see [1]. □

From now on, let $E(K)$ denote the abelian group of K-rational points with the neutral element ∞ and operation $\oplus$.

Based on the Weierstrass equation, we can derive simple formulas for determining $P \oplus Q$. We omit the proof as it is quite a straightforward calculation, details are outlined in [1].

Proposition 1.2.1. *Let E be an elliptic curve given by the Weierstrass equation $y^2 = x^3 + Ax + B$ over a field K with* $\mathrm{char}(K) \in \{2, 3\}$ *and let $P, Q \in E$ be affine points on E, so $P = (x_1, y_1), Q = (x_2, y_2)$. Then $R = P \oplus Q$ is defined as follows:*

(1) If $x_1 = x_2$ and $y_1 \neq y_2$, then $R = \infty$.

(2) If $Q = P = (x_1, y_1)$ and $y_1 = 0$, then $R = \infty$.

(3) (Doubling formula) If $Q = P = (x_1, y_1)$ and $y_1 \neq 0$, then $R = (x_3, y_3)$ is given by

$$x_3 = \lambda^2 - 2x_1, \quad y_3 = \lambda(x_1 - x_3) - y_1, \quad \text{where } \lambda = \frac{3x_1^2 + A}{2y_1}.$$

(4) (Addition formula) Otherwise $R = (x_3, y_3)$ is an affine point satisfying

$$x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda(x_1 - x_3) - y_1, \quad \text{where } \lambda = \frac{y_1 - y_2}{x_1 - x_2}.$$

The case when either P or Q is a point at infinity follows from the definition of $\oplus$.

For adding a point P on an elliptic to itself repeatedly, we adopt the following convention:

$$\begin{aligned}
[n]P &= \overbrace{P \oplus P \oplus \cdots \oplus P}^{n \text{ times}} && \text{if } n > 0,\\
[0]P &= \infty, &&\\
[n]P &= \underbrace{(\ominus P) \oplus (\ominus P) \oplus \cdots \oplus (\ominus P)}_{|n| \text{ times}} && \text{otherwise if } n < 0.
\end{aligned}$$

Example 1.2.1. Viewing the elliptic curve E given by $y^2 = x^3 - x + 1$ over $\mathbb{F}_{11}$ as the group $E(\mathbb{F}_{11})$ and using the points $P = (3, 5), Q = (5, 11)$, we can find new rational points of E by applying addition formulas; for example

$$[2]P = (10, 1) \quad \text{and} \quad P \oplus Q = (1, 1).$$

Table 1.1 enumerates all points on the elliptic curve E.

Point $P = (x, y)$	**Inverse** $\ominus P = (x, -y)$
$(5, 0)$	$(5, 0)$
$(0, 1)$	$(0, -1)$
$(-1, 1)$	$(-1, -1)$
$(1, 1)$	$(1, -1)$
$(3, 5)$	$(3, -5)$

Table 1.1: The list of points on the elliptic curve E given by $y^2 = x^3 - x + 1$ over $\mathbb{F}_{11}$.

One can easily verify that this group is cyclic with $P = (0, 1)$ as its generator. Moreover, the group contains 10 points (including ∞ not listed in the table), so it is isomorphic to the additive group $\mathbb{Z}_{10}$.

1.3 Projective Elliptic Curve

So far we have worked with elliptic curves in the affine plane $\mathbb{A}^2$. Replacing $\mathbb{A}^2$ with the projective plane $\mathbb{P}^2$ means adding a variable to the Weierstrass equation, so that the resulting polynomial is homogeneous. Introducing an extra variable to the Weierstrass equation, which will be mostly of no use in the upcoming sections, will help us understand the notion of the point at infinity.

Let E be an elliptic curve given by the Weierstrass equation defined over a field K and consider it as an equation in $K[X, Y, Z]$. First, we homogenize the equation to get

$$E : Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3. \tag{1.5}$$

In this case, the elliptic curve E is the set of points $(x : y : z) \in \mathbb{P}^2$ satisfying (1.5). There is no need to explicitly involve the point ∞ in the set since it can be identified with one of the points in $\mathbb{P}^2$. Notice the affine version of the equation can be attained by setting $x = X/Z$ and $y = Y/Z$.

To see that the point ∞ is a projective point, namely a point at infinity, set $Z = 0$ to obtain

$$0 = X^3. \tag{1.6}$$

It is now clear that the only point at infinity on E is $\infty = (0 : 1 : 0)$, for at least one coordinate of a projective point in the projective plane must be non-zero.

1.4 Endomorphisms

Let E, E' be two smooth projective elliptic curves over a field K. A *morphism* $\psi : E \mapsto E'$ defined over K is a map that may be represented by $(F_1 : F_2 : F_3)$ where $F_i \in K[X, Y, Z], 1 \leq i \leq 3$ are homogeneous polynomials of the same degree such that, except for finitely many points, for any $P \in E$ there exists $1 \leq j \leq 3$ with $F_j(P) \neq 0$. The morphism ψ does not have a unique representation: $\psi = (F_1 : F_2 : F_3) = (G_1 : G_2 : G_3)$ where $G_1, G_2, G_3 \in K[X, Y, Z]$ if and only if $F_iG_j - F_jG_i$ vanishes on E for $1 \leq i < j \leq 3$. It may be proved [6] that if $P \in E$ is a point such that $F_i(P) = 0$ for all $1 \leq i \leq 3$, then there exist $G_i, 1 \leq i \leq 3$ such that $\psi = (G_1 : G_2 : G_3)$ and $G_i(P) \neq 0$ for at least one i. This means that the morphism ψ is defined at every point P on E.

In the context of elliptic curves, a special attention is drawn to a specific type of morphism called *isogeny*. An isogeny is a morphism $\phi : E \mapsto E'$ such that $\phi(\infty_E) = \infty_{E'}$, where $\infty_E, \infty_{E'}$ are the neutral elements in the groups $E(K), E'(K)$ respectively. Moreover, it can be shown [1] that isogenies are actually homomorphisms between groups $E(\overline{K})$ and $E'(\overline{K})$ (notice the definition of an isogeny involves a necessary condition for a map to be a homomorphism). Another interesting result [4] related to isogenies says that for any two isogenies ψ_1, ψ_2 from E to E', the map $\psi_1 \oplus \psi_2$ defined as $(\psi_1 \oplus \psi_2)(P) = \psi_1(P) \oplus \psi_2(P)$ for any point $P \in E$ is also an isogeny.

In this section, we will mainly deal with *endomorphisms*, the isogenies mapping $E(K)$ to itself. In particular, a special emphasis in our analysis of endomorphism will be laid on *multiplication-by-n* endomorphisms, which send a point $P \in E(\overline{K})$ to $[n]P$, where n is a positive integer. Later in Chapter 2, we will discover that there exists an explicit formula for computing $[n]P$ for any positive integer n and point $P \in E(\overline{K})$.

Maps between elliptic curves are usually in the form of *rational maps*. Since we will mostly work with affine curves we shall now define what is a rational map of two affine curves, and then explain how such a map may be converted to a morphism of correspoding projective curves.

Affine rational maps are represented by $(r_1/s_1, r_2/s_2)$ for some polynomials $r_i, s_i \in K[x, y], 1 \leq i \leq 2$. Every rational map of this form can be expressed as a morphism and vice versa.

To convert a rational map represented by $(r_1/s_1, r_2/s_2)$ to a morphism, choose polynomials $R_i(X, Y, Z), S_i(X, Y, Z)$ with $\deg(R_i) = \deg(S_i)$ and $\gcd(R_i, S_i) = 1$ satisfying $R_i(x, y, 1)/\ S_i(x, y, 1) = r_i(x, y)/s_i(x, y)$ for $1 \leq i \leq 2$. It follows that $(R_1/S_1 : R_2/S_2 : 1)$ represents $(r_1/s_1, r_2/s_2)$ in the projective plane. A morphism corresponding to the given rational map can be taken to be $(R_1S/S_1 : R_2S/S_2 : S)$, where $S = \mathrm{lcm}(S_1, S_2)$. On the other hand, a morphism $(A_1 : A_2 : A_3)$

can be transformed to a rational map in the affine plane by taking the map $(\pi(A_1)/\pi(A_3), \pi(A_2)/\pi(A_3))$, where $\pi : K[X,Y,Z] \mapsto K[X,Y]$ is defined as follows:

$$\pi(X) = X, \quad \pi(Y) = Y, \quad \pi(Z) = 1.$$

It is now clear that the situation when the denominator of every representative of a rational map is zero corresponds to the case when the point $(X : Y : 0) \in \mathbb{P}^2$ is being mapped, in our case, to the point ∞.

Let $\phi : E(\overline{K}) \mapsto E(\overline{K})$ be an endomorphism defined for any point $P = (x, y) \in E(\overline{K})$ as

$$\phi(P) = \phi(x, y) = (R_1(x, y), R_2(x, y)), \tag{1.7}$$

where $R_i = r_i(x, y)/s_i(x, y), 1 \leq i \leq 2$, are rational functions with $r_1, r_2, s_1, s_2 \in K[x, y]$. The definition of rational maps as in (1.7) can be adjusted by replacing each y^2 on the RHS of (1.2) and doing a few algebraic changes, leaving us with

$$r_i(x, y) = \frac{p_1^{(i)}(x) + p_2^{(i)}(x)y}{q^{(i)}(x)}$$

for some $p_j^{(i)}, q_j^{(i)} \in K[x, y]$. Moreover, by applying some basic facts about homomorphisms, we may even assume that there exist rational functions $r_1 = u_1(x)/v_1(x), r_2 = u_2(x)/v_2(x)$ satisfying

$$\phi(P) = (r_1(x), r_2(x)y)\,. \tag{1.8}$$

Indeed,

$$\ominus\phi(x, y) = \phi(\ominus(x, y)) = \phi(x, -y)$$

implies

$$\begin{aligned} r_1(x, -y) &= r_1(x, y), \\ r_2(x, -y) &= -r_2(x, y) \end{aligned}$$

from which it follows that $p_2^{(1)} = 0$ and $p_1^{(2)} = 0$ (we are assuming $\mathrm{char}(K) \neq 2$). Hence $\phi(P)$ may be written as in (1.8).

This shows that the x-coordinate of an endomorphism is always given by a rational function in only one variable. This will turn out to be useful from the computational point of view later in the discussion of Schoof's algorithm.

Not only are the endomorphism of elliptic curves defined everywhere, but they also turn out to be surjective, the only exception being the *trivial endomorphism* $[0] : E(\overline{K}) \mapsto E(\overline{K})$ sending every point to ∞ [1].

Theorem 1.4.1. *Every non-trivial endomorphism $\phi : E(\overline{K}) \mapsto E(\overline{K})$ of an elliptic curve E defined over a field K is surjective.*

Apart from multiplication-by-n endomorphisms, the *q^{th}-power Frobenius map* ϕ_q is another example of an endomorphism. As we will discover later, the Frobenius map is crucial to Schoof's algorithm.

Definition 1.4.1. Let E be an elliptic curve over a finite field $\mathbb{F}_q$. The Frobenius map $\phi_q : E(\overline{\mathbb{F}}_q) \mapsto E(\overline{\mathbb{F}}_q)$ applies the Frobenius automorphism $\phi_q : \overline{\mathbb{F}}_q \mapsto \overline{\mathbb{F}}_q$ to each coordinate. In other words,

$$\phi_q(P) = \begin{cases} \infty & \text{if } P = \infty, \\ (x^q, y^q) & \text{otherwise, where } P = (x, y) \in E(\overline{\mathbb{F}}_q). \end{cases} \tag{1.9}$$

It is easy to show that ϕ_q is actually an endomorphism E, so referring to the Frobenius map on an elliptic curve as the Frobenius endomorphism on the elliptic curve is justified:

Proposition 1.4.1. *The Frobenius map $\phi_q : E(\overline{\mathbb{F}_q}) \mapsto E(\overline{\mathbb{F}_q})$ is an endomorphism on an elliptic curve E defined over a field $\mathbb{F}_q$.*

Proof. Let the elliptic curve E be given by the equation $y^2 = f(x)$ for some monic cubic $f \in \mathbb{F}_q[x]$. First, we need to show that $\phi_q(P) \in E(\overline{\mathbb{F}}_q)$ for each point $P \in E(\overline{\mathbb{F}}_q)$. Since by definition $\phi_q(\infty) = \infty$, we may assume $P = (x, y) \in \mathbb{A}^2$. Raising the equation for E to q and using the fact that $(a + b)^q = a^q + b^q$ in $\overline{\mathbb{F}}_q$ and $a^q = a$ for all $a \in \mathbb{F}_q$ gives

$$(y^q)^2 = (x^3 + Ax + B)^q = (x^q)^3 + Ax^q + B.$$

Hence $\phi_q(P) \in E$. Finally, since ϕ_q is an isogeny, it immediately follows that ϕ_q is a homomorphism. □

Repeated composition of Frobenius maps as $\phi_q \circ \cdots \circ \phi_q$, the composition being applied n-times, yields again a Frobenius map, namely ϕ_{q^n}. Therefore, it makes sense to use the notation $\phi_q^n = \phi_q \circ \cdots \circ \phi_q$, where the Frobenius is composed with itself n times.

Similarly as in the finite fields, it holds that $P \in E(\mathbb{F}_q)$ if and only if $\phi_q(P) = P$. Hence, the group $E(\mathbb{F}_q)$ can be characterised by the kernel of $\phi_q \ominus [1]$, see [1].

Proposition 1.4.2. *Let E be an elliptic curve defined over the finite field $\mathbb{F}_q$. Then* $\mathrm{Ker}(\phi_q - 1) = E(\mathbb{F}_q)$.

Proof. A point $P \in E(\mathbb{F}_q)$ lies in $\mathrm{Ker}(\phi_q - 1)$ if and only if $\phi_q(P) = P$, which occurs if and only if $P \in E(\mathbb{F}_q)$. □

In Chapter 3, we will see a connection between the Frobenius endomorphism and the group of an elliptic curve defined over a finite field.

2. Torsion Groups

2.1 Torsion Groups

Definition 2.1.1. Let E be an elliptic curve defined over a field K and let n be a positive integer. The n-torsion subgroup of $E(\overline{K})$, denoted by $E[n]$, is defined as

$$E[n] = \{P \in E(\overline{K}) \mid [n]P = \infty\}.$$

The natural question is, how to find all such points of order dividing a given n. We will now look at the trivial cases when $n \in \{2, 3\}$ to explore the idea behind searching for all points in $E[n]$.

Let

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

be an elliptic curve given by the Weierstrass equation defined over a field K with $\text{char}(K) \notin \{2, 3\}$. Trivially, $\infty \in E[n]$ for any positive integer n. For $n = 2$, our task is to find all involutions of the group $E(\overline{K})$, in other words, all affine points $P \in E(\overline{K})$ satisfying $[2]P = \infty$, which is equivalent to $P = \ominus P$. Using the fact that $\ominus P = (x, -y)$ if $P = (x, y) \in E(\overline{K})$ is an affine point, it must be the case that $y = 0$, so the points of $E[2]$ are determined by the x-coordinate. The x-coordinates are exactly the roots of $f(x)$. Hence, the 2-torsion subgroup of $E(\overline{K})$ is

$$E[2] = \{(r, 0) \mid f(r) = 0\} \cup \{\infty\}.$$

In addition, $|E[2]| = 3 + 1 = 4$, which follows from separability of the polynomial $f(x)$. Also notice that the group $E(\mathbb{F}_q)$ contains no rational point of order 2 if and only if f is irreducible over K. This remark will turn out to be useful when describing Schoof's algorithm in Chapter 4.

To solve the case for $n = 3$, observe that a point $P \in E[3] \setminus \{\infty\}$ is subject to the equality $[2]P = \ominus P$. Assuming $f(x) = x^3 + Ax + B$, we can use the formulas for addition to show the equality holds for a point $P = (x, y)$ if and only if

$$3x^4 + 6Ax^2 + 12Bx - A^2 = 0. \tag{2.1}$$

It may be proved [1] that (2.1) is separable, from which we can conclude that there are exactly 8 different affine points $P = (x, y) \in E[3]$ as each of the four different roots of (2.1) gives rise to two points ($y = 0$ is only possible for points in $E[2]$). Hence, after including the point ∞, we have $|E[3]| = 9$.

The following theorem [1] answers our question of the structure of $E[n]$ for any positive integer n.

Theorem 2.1.1. *Let E be an elliptic curve defined over a field K with characteristic $\text{char}(K) = p$. Let n be a positive integer. Then*

$$E[n] \cong \mathbb{Z}_n \oplus \mathbb{Z}_n \tag{2.2}$$

if $p = 0$ or $p \nmid n$. Otherwise, by writing n as $n = p^k n'$ so that $p \nmid n'$, the following holds:

$$E[n] \cong \mathbb{Z}_{n'} \oplus \mathbb{Z}_{n'} \quad \text{or} \quad E[n] \cong \mathbb{Z}_n \oplus \mathbb{Z}_{n'}. \tag{2.3}$$

It follows from Theorem 2.1.1 that $E[n]$ forms a group generated by some $\alpha_1, \alpha_2 \in E(\overline{K})$. Furthermore, $\phi_q(E[n]) \subseteq E[n]$ for any endomorphism $\phi : E \mapsto E$, so $\phi_q \restriction_{E[n]}$ can be described, in similar fashion as a linear map, by a matrix $(\phi)_n$ over $\mathbb{Z}_n$: if $\phi(\alpha_1) = a\alpha_1 + b\alpha_2$ and $\phi(\alpha_2) = c\alpha_1 + d\alpha_2$, then

$$(\phi)_n = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathbb{Z}_n^{2\times 2}. \tag{2.4}$$

2.2 Division Polynomials

It might be tempting to conclude, after investigating the trivial cases, that a point $P \in E(\overline{K}) \setminus \{\infty\}$ belongs to $E[n]$ if and only if there exists a polynomial $\psi_n \in K[x, y]$ such that $\psi_n(P) = 0$. Indeed, we will now see that there exists a polynomial $\psi_n \in K[x, y]$ for any positive integer n, called *nth division polynomial*, satisfying this property [1].

Assume the Weierstrass curve E is given by $y^2 = f(x)$ over a field with characteristic p. Let n be an integer not divisible by p. Notice that when $P = (x, y) \in E[n]$, then also $\ominus P = (x, -y)$ and vice versa, since $E[n]$ is a group. Hence, when n is odd, every root of ψ_n represents the x-coordinate of a point and its inverse. Therefore, $\deg \psi_n = (n^2 - 1)/2$. On the other hand, when n is even, $E[n]$ contains points of order two, so in this case we have $(n^2 - |E[2]|)/2 + |E[2]|$, but we already know that $|E[2]| = 4$, so we have $\deg \psi_n = (n^2 + 4)/2$.

Lastly, it remains to answer the question of how to actually construct the polynomials ψ_n. Fortunately, according to the proposition below, we can do so by applying a recursive rule.

Proposition 2.2.1. *For an elliptic curve E given by $y^2 = x^3 + Ax + B$ defined over a field K, the division polynomial ψ_n can be defined recursively as follows:*

$$\begin{aligned}
\psi_0 &= 0, \\
\psi_1 &= 1, \\
\psi_2 &= 2y, \\
\psi_3 &= 3x^4 + 6Ax^2 + 12Bx - A^2, \\
\psi_4 &= 4y(x^6 + 5Ax^4 + 20Bx^3 - 5A^2x^2 - 4ABx - 8B^2 - A^3), \\
\psi_{2n+1} &= \psi_{n+2}\psi_n^3 - \psi_{n-1}\psi_{n+1}^3 \quad \text{when } n \geq 2, \\
\psi_{2n} &= \frac{\psi_n}{2y}(\psi_{n+2}\psi_{n-1}^2 - \psi_{n-2}\psi_{n+1}^2) \quad \text{when } n \geq 3.
\end{aligned}$$

The nth division polynomials, defined as above, characterize the points in $E[n]$:

Theorem 2.2.1. *Let E be an elliptic curve defined over a field K and let $P \in E(\overline{K}) \setminus \{\infty\}$. Then $P \in E[n]$ if and only if $\psi_n(P) = 0$.*

The division polynomial can also be utilized to explicitly express the coordinates of $[n]P$ for any positive integer n and a point $P \in E(\overline{K})$ satisfying $[n]P \neq \infty$.

Theorem 2.2.2. *Let E be an elliptic curve over a field K and let $P \in E(\overline{K})$. Then*

$$[n]P = \left(\frac{\theta_n(x)}{\psi_n^2(x,y)}, \frac{\omega_n(x,y)}{\psi_n^3(x,y)} \right),$$

where

$$\theta_n = x\psi_n^2 - \psi_{n+1}\psi_{n-1} \text{ and}$$
$$\omega_n = \frac{\psi_{n+2}\psi_{n-1}^2 - \psi_{n-2}\psi_{n+1}^2}{4y}$$

for any positive integer n such that $[n]P \neq \infty$.

The case $[n]P = \infty$ can be treated separately since, by Theorem 2.2.1, it occurs if and only if $\psi_n(P) = 0$. The following proposition becomes useful in the analysis of the time complexity of Schoofs' algorithm.

Proposition 2.2.2. $\deg \theta_n(x)$ *and* $\deg \psi_n^2(x)$ *satisfy*

$$\deg \theta_n(x) = n^2 \text{ and}$$
$$\deg \psi_n^2(x) = n^2 - 1.$$

It follows from Proposition 2.2.2 that the division polynomial ψ_n is separable for any odd n not divisible by the characteristic of the field: by Theorem 2.1.1, the number of affine points in $E[n]$ equals $|E[n]| - 1 = n^2 - 1$, so there are $(n^2-1)/2 = \deg \psi_n$ distinct x-coordinates among the points in $E[n]$.

In fact, the division polynomials may also be defined as univariate polynomials $\bar{f}_n \in K[x]$ [4], sharing the property with polynomials ψ_n that $P \in E[n] \setminus \{\infty\}$ if and only if $\bar{f}_n(P) = 0$:

$$\bar{f}_n = \begin{cases} \psi_n & \text{if } n \text{ is odd, and} \\ \psi_n/2y & \text{if } n \text{ even.} \end{cases}$$

Their recursive definition is given below.

$$\begin{aligned}
\bar{f}_0 &= 0, \\
\bar{f}_1 &= 1, \\
\bar{f}_2 &= 1, \\
\bar{f}_3 &= 3x^4 + 6Ax^2 + 12Bx - A^2, \\
\bar{f}_4 &= 2(x^6 + 5Ax^4 + 20Bx^3 - 5A^2x^2 - 4ABx - 8B^2 - A^3), \\
\bar{f}_{2n+1} &= \begin{cases} \bar{f}_{n+2}\bar{f}_n^3 - 16(x^3 + Ax + B)^2\bar{f}_{n-1}\bar{f}_{n+1}^3 & \text{when } n \geq 3 \text{ is odd,} \\ 16(x^3 + Ax + B)^2\bar{f}_{n+2}\bar{f}_n^3 - \bar{f}_{n-1}\bar{f}_{n+1}^3 & \text{when } n \geq 2 \text{ is even,} \end{cases} \\
\bar{f}_{2n} &= \bar{f}_n(\bar{f}_{n+2}\bar{f}_{n-1}^2 - \bar{f}_{n-2}\bar{f}_{n+1}^2) \quad \text{for } n \geq 2.
\end{aligned}$$

3. Elliptic Curves over Finite Fields

As we have already seen, the subgroup $E[n]$ of the group of an elliptic curve defined over an arbitrarily large field K can be briefly described as the direct sum $\mathbb{Z}_n \oplus \mathbb{Z}_n$. If we turn our attention to the case when K is finite, the group $E(K)$ of rational points becomes a torsion group as well. Moreover, $E(\mathbb{F}_q)$ is clearly a subgroup of $E[n] \cong \mathbb{Z}_n \oplus \mathbb{Z}_n$ for a sufficiently large n divisible by the order of $E(\mathbb{F}_q)$. Hence, by applying the structure theorem for finite abelian groups [7], $E(\mathbb{F}_q) \cong \mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2}$ for some integers n_1, n_2 satisfying $n_1 \mid n_2$. The theorem below [4] imposes yet another restriction on n_1:

Theorem 3.0.1. *Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$. Then there exist integers $n_2 \geq n_1 \geq 1$ satisfying $n_1 \mid n_2$ and $n_1 \mid q-1$ such that*

$$E(\mathbb{F}_q) \cong \mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2}.$$

In the rest of this chapter, our attention turns to estimating the order of $E(\mathbb{F}_q)$, the importance of which derives from cryptography applications in which it is used, for example, for checking desired properties of the group order of $E(\mathbb{F}_q)$. In the context of counting points on an elliptic curve, the order of $E(\mathbb{F}_q)$ is usually denoted $\#E(\mathbb{F}_q)$.

Each element $x \in \mathbb{F}_q$ gives rise to at most two $y \in \mathbb{F}_q$ such that (x, y) is a point on an elliptic curve. Hence, including the neutral element ∞, there are at most $2q + 1$ rational points on the curve. In fact, the number of rational points lies somewhere around $q + 1$. Hasse's theorem [1], a major theorem applied to the problem of counting rational points, says how close around $q + 1$ the value $\#E(\mathbb{F}_q)$ lies.

Theorem 3.0.2 (Hasse, 1933)**.** *The number of rational points $\#E(\mathbb{F}_q)$ on an elliptic curve defined over a finite field $\mathbb{F}_q$ satisfies*

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

The quantity $t = q+1-\#E(\mathbb{F}_q)$ is sometimes referred to as the *trace of Frobenius*. The name originates from the next theorem, which relates the Frobenius endomorphism to the quantity t.

Theorem 3.0.3. *Let E be an elliptic curve over a finite field $\mathbb{F}_q$ and let $\phi_q : E \mapsto E$ denote the Frobenius endomorphism on E. Then*

$$\phi_q^2 \ominus [t]\phi_q \oplus [q] = [0], \tag{3.1}$$

and the value of t that satisfies (3.1) is unique. Moreover, the quantity t satisfies the congruence

$$t \equiv \mathrm{Trace}((\phi_q)_n) \pmod{n} \tag{3.2}$$

for all integers n coprime to q.

Remark. Although we do not provide a complete proof of Theorem 3.0.3, we at least show the part about the trace of Frobenius t for prime n (Schoof's algorithm uses only $E[n]$ for n prime). Taking the matrix $(\phi_q)_n$, introduced in Chapter 2, we find the characteristic polynomial of $(\phi_q)_n$:

$$\det\left(\lambda I_2 - (\phi_q)_n\right) \equiv \det\begin{pmatrix} \lambda - a & b \\ c & \lambda - d \end{pmatrix} \equiv \lambda^2 - (a+d)\lambda + (ad - bc) \pmod{n}. \tag{3.3}$$

By the Cayley-Hamilton theorem, the matrix $(\phi_q)_n$ satisfies

$$(\phi_q)_n^2 \oplus (a+d)(\phi_q)_n \oplus [ad - bc] \equiv \mathbf{0} \pmod{n}, \tag{3.4}$$

where $\mathbf{0}$ stands for the zero matrix. Equation (3.4) states that (3.1) holds for all points in $E[n]$. Combining (3.1) with (3.4) and the uniqueness of t yields

$$\mathrm{Tr}((\phi_q)_n) \equiv a + d \equiv t \pmod{n}$$

for all n coprime to q.

Hasse's theorem can be used to attack the problem of finding $\#E(\mathbb{F}_q)$ by observing that $\#E(\mathbb{F}_q)$ lies in the interval of length $4\sqrt{q}$. Therefore, if there exists a point of order $k > 4\sqrt{q}$, then exactly one multiple of k lies in the interval $[\#E(\mathbb{F}_q) - 2\sqrt{q}, \#E(\mathbb{F}_q) + 2\sqrt{q}]$.

However, the existence of such a point is not guaranteed. Nevertheless, a related result [2] tells us that there is always such a point on E or on its so-called *(quadratic) twist* whenever the finite field $\mathbb{F}_p$ is sufficiently large. Before stating the theorem, it is necessary to become familiar with the definition of a *twist* of an elliptic curve.

Definition 3.0.1. A (quadratic) twist E^d of an elliptic curve E given by $y^2 = x^3 + Ax + B$ over a field K is an elliptic curve of the form

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

where $d \in K^*$ is a quadratic non-residue in K^*.

While determining the order of an elliptic curve E over $\mathbb{F}_q$ might be challenging for some instances, it may happen that the opposite is true for E^d. When this occurs, the proposition below can be used to determine $\#E(\mathbb{F}_q)$ from the knowledge of $\#E^d(\mathbb{F}_q)$. The proof [2] requires basic knowledge of quadratic residues [8].

Proposition 3.0.1. *Let E be an elliptic curve over $\mathbb{F}_q$ and let E^d be its twist, with $d \in \mathbb{F}_q^*$ not a square in $\mathbb{F}_q^*$. Then*

$$\#E(\mathbb{F}_q) + \#E^d(\mathbb{F}_q) = 2(q+1).$$

Proof. The theorem can be proved simply by going through all elements in $E(\mathbb{F}_q)$ and counting their contribution to the sum $\#E(\mathbb{F}_q) + \#E^d(\mathbb{F}_q)$.

Let $g(x) = x^3 + Ax + B$ and $g_d(x) = g(x/d)/d^3$ correspond to the polynomials of the right-hand-side of the Weierstrass equation for E and E^d, respectively. Now the proof splits into three cases for an element $x \in \mathbb{F}_q$.

- If $g_d(x) = 0$ for $x \in \mathbb{F}_q$, then $g(x/d) = 0$, so $(x, 0) \in E^d(\mathbb{F}_q)$ and $(x/d, 0) \in E(\mathbb{F}_q)$.
- If $g_d(x)$ is a non-zero quadratic residue in $\mathbb{F}_q$, then there exist two $y's$ such that $y^2 = g_d(x)$. On the other hand, there is no $y \in \mathbb{F}_q$ for which $y^2 = g(x/d) = d^3 g_d(x)$; assuming $g_d(x) = w^2$ for some $w \in \mathbb{F}_q$, we would have $d = (d^{-2}y^2w^{-2})$, contrary to d being a quadratic non-residue.
- Finally, if $g_d(x)$ is a quadratic non-residue in $\mathbb{F}_q$, then there exists no $y \in \mathbb{F}_q$ satisfying the relation $y^2 = g_d(x)$, but there exist two $y's$ for which $y^2 = g(x/d)$, since $g(x/d)$ is a residue; $g(x/d) = d^{-3}g_d(x)$ is a product of two non-residues, which always yields a quadratic residue.

In total, every $x \in \mathbb{F}_q$ contributes two counts to the sum $\#E(\mathbb{F}_q) + \#E^d(\mathbb{F}_q)$, so adding the neutral element ∞ counted twice for each curve yields the theorem. □

Theorem 3.0.4. *There exists a point P on an elliptic curve E defined over the finite field $\mathbb{F}_p$ or on its twist E^d with an order larger than $4\sqrt{p}$ whenever $p \geq 230$.*

Theorem 3.0.4 resolves the case when the order of the field is prime. However, once we have determined $\#E(\mathbb{F}_q)$, the order of $E(\mathbb{F}_{q^n})$ can be computed explicitly.

Proposition 3.0.2. *If $x^2 - tx + q = (x - \alpha)(x - \beta)$, where t is the trace of Frobenius of an elliptic curve E over the finite field $\mathbb{F}_q$, then*

$$\#E(\mathbb{F}_{q^n}) = q^n + 1 - (\alpha^n + \beta^n)$$

for any integer $n \geq 1$.

Proof. We will show that the polynomial

$$\begin{aligned} f(x) &= (x - \alpha^n)(x - \beta^n) = x^2 - (\alpha^n + \beta^n)x + (\alpha\beta)^n \\ &= x^2 - (\alpha^n + \beta^n)x + q^n. \end{aligned}$$

is the characteristic polynomial of ϕ_{q^n} on E, thus proving that $\tilde{t} = \alpha^n + \beta^n$ is the trace of Frobenius for E over $\mathbb{F}_{q^n}$. Write

$$x^{2n} - (\alpha^n + \beta^n)x^n + q^n = f(x^n) = (x^n - \alpha^n)(x^n - \beta^n).$$

Notice that the polynomial $f(x^n)$ is divisible by $(x-\alpha)(x-\beta)$ since we can write

$$\begin{aligned} x^n - \alpha^n &= (x - \alpha)(x^{n-1} + \cdots + 1) \text{ and} \\ x^n - \beta^n &= (x - \beta)(x^{n-1} + \cdots + 1). \end{aligned}$$

Assume $\alpha^n + \beta^n$ is an integer (otherwise $\#E(\mathbb{F}_q)$ would not be an integer). Under this assumption, which will be proved as a lemma below, and from the observation above, there exists a polynomial $g(x) \in \mathbb{Z}[x]$ satisfying the equality

$$x^{2n} - (\alpha^n + \beta^n) + q^n = f(x^n) = g(x)(x^2 - tx + q). \tag{3.5}$$

Now comes the part where we make use of Theorem 3.0.3. Plugging the Frobenius endomorphism ϕ_q into (3.5) gives

$$\phi_{q^n}^2 \ominus [\alpha^n \oplus \beta^n]\phi_{q^n} \oplus [q^n] = g(\phi_q)(\phi_q^2 \ominus [t]\phi_q \oplus [q]) = [0].$$

Moreover, the same theorem claims that there exists a unique element $\tilde{t}$ satisfying $\phi_{q^n}^2 \ominus [\tilde{t}]\phi_{q^n} \oplus [q^n]$, which is simultaneously equal to $q^n + 1 - \#E(\mathbb{F}_{q^n})$ and $\alpha^n + \beta^n$, so

$$\#E(\mathbb{F}_{q^n}) = q^n + 1 - (\alpha^n + \beta^n).$$

□

To complete the proof, it remains to show $\alpha^n + \beta^n \in \mathbb{Z}$. The following lemma not only proves this claim but also suggests how to recursively compute $\alpha^n + \beta^n$.

Lemma 1. *If we define $s_n = \alpha^n + \beta^n$ as a sequence, then $s_0 = 2, s_1 = t$ and $s_{n+1} = ts_n - qs_{n-1}$ for each $n \geq 2$.*

Proof. α is a root of the characteristic polynomial $x^2 - tx + q$ of the Frobenius endomorphism on an elliptic curve E over $\mathbb{F}_q$, hence $\alpha^2 = t\alpha - q$; the same relation holds for β. Consequently, by multiplying the relation by α^{n-1}, the relation becomes $\alpha^{n+1} = t\alpha^n - q\alpha^{n-1}$ for α, and similarly, by replacing α with β, we obtain $\beta^{n+1} = t\beta^n - q\beta^{n-1}$. Adding the expressions for α and β yields

$$\alpha^{n+1} + \beta^{n+1} = t(\alpha^n + \beta^n) - q(\alpha^{n-1} + \beta^{n-1}) = ts_n - qs_{n-1}.$$

□

4. Schoof's Algorithm

In cryptography settings based on elliptic curves, several restrictions are imposed on the curves in use. This is because there exist quite effective attacks [1] on particular families of curves, so these curves must be avoided in order to improve the security of the setting of interest. To give an example, MOV attack [9] uses the Weil pairing [6] to translate the elliptic curve discrete logarithm problem to the discrete logarithm problem in a finite field $\mathbb{F}_{q^n}$, where $\mathbb{F}_q = \mathbb{F}_{p^m}$ is the field over which the elliptic curve is defined and n is the smallest non-negative integer such that $p \mid q^n - 1$. Hence, the MOV attack is a threat to those curves for which small such n can be found. Anomalous curves are another example of curves for which there is an attack [10] that renders them insecure.

Apart from these attacks, the number of rational points on an elliptic curve is also an important aspect for deciding whether the given elliptic curve is a good candidate for cryptographic applications, because it gives us an idea of possible orders in the group. The point counting problem will be the subject of this chapter. We will describe Schoof's algorithm [3][4] for counting the number of rational points on an elliptic curve defined over a finite field, which forms a basis for the most efficient versions of the algorithms solving the problem. After explaining the algorithm, we will inspect its time complexity in general, which will in turn allow us to establish the time complexity for particular implementations of multiplication in the ring of polynomials over $\mathbb{F}_q$.

4.1 Schoof's Algorithm

Suppose our task is to determine $\#E(\mathbb{F}_q)$, where $q = p^n, p > 3$ and E is an elliptic curve given by the Weierstrass equation $y^2 = x^3 + Ax + B$ over $\mathbb{F}_q$. Schoof's algorithm tackles this problem by taking advantage of Hasse's theorem, by which $\#E(\mathbb{F}_q) = q + 1 - t$ where $|t| \leq 2\sqrt{q}$ and so t lies within exactly one range of length $4\sqrt{q}$.

Let $\ell_1 < \ell_2 < \cdots < \ell_k$ be primes distinct from the characteristic p such that

$$m = \prod_{i=1}^{k} \ell_i > 4\sqrt{q}. \tag{4.1}$$

By Hasse's theorem, only one multiple of $\hat{t} \equiv t \pmod{m}$ satisfies $|\hat{t}| \leq 2\sqrt{q}$, so that multiple must be equal to the trace of Frobenius t. Hence, if $t \pmod{\ell_i}$ is known for all $1 \leq i \leq k$, the Chinese Remainder Theorem can be used to recover t. We therefore need to find a way of establishing t modulo each ℓ_i.

Assume first $\ell = 2$. To determine $t \pmod 2$, notice that $\#E(\mathbb{F}_q) \equiv 0 \pmod 2$ if and only if the group $E(\mathbb{F}_q)$ contains a rational point of order 2. As we have seen in Chapter 2, this occurs if and only if $x^3 + Ax + B$ has a root in $\mathbb{F}_q$. Since q is assumed to be odd, the condition on $\#E(\mathbb{F}_q)$ being even is equivalent to saying that $t \equiv 0 \pmod 2$, which follows from Hasse's theorem:

$$\#E(\mathbb{F}_q) = q + 1 - t \equiv 0 \pmod 2 \quad \iff \quad t \equiv 0 \pmod 2.$$

To decide if $x^3 + Ax + B$ has a root in $\mathbb{F}_q$, the greatest common divisor of $x^3 + Ax + B$ and the polynomial $x^q - x$, whose roots match exactly the elements

of $\mathbb{F}_q$, is computed. If the gcd equals 1, then $x^3 + Ax + B$ has no roots in $\mathbb{F}_q$, which means it is irreducible over the finite field and $t \equiv 1 \pmod 2$. Otherwise $t \equiv 0 \pmod 2$.

The core of Schoof's algorithm is to find $t \pmod \ell$ for an odd prime ℓ. Recall the Frobenius endomorphism ϕ_q on elliptic curves satisfies the following relation:

$$\phi_q^2 \ominus [t]\phi_q \oplus [q] = [0] \quad \iff \quad [t]\phi_q = \phi_q^2 \oplus [q]. \tag{4.2}$$

The Frobenius endomorphism restricted to points in $E[\ell]^* = E[\ell] \setminus \{\infty\}$ is a linear bijection since it is a one-to-one linear map of dimension 2. Moreover, since ℓ is prime, $\phi_q(P)$ for $P \in E[\ell]^*$ is again a point of order ℓ. As a consequence, the relation (4.2) can be rewritten when restricted only to points in $E[\ell]^*$ for some prime $\ell \neq p$ as

$$[t_\ell]\phi_q(P) = \phi_q^2(P) \oplus [q_\ell]P, \tag{4.3}$$

for all $P \in E[\ell]^*$, where $t_\ell = t \pmod \ell$ and $q_\ell = q \pmod \ell$ are the least non-negative representatives of $[t]_\ell$ and $[q]_\ell$. Because of the same reasons stated in the paragraph above, if the relation (4.3) holds for one point, then it holds for all $P \in E[\ell]^*$. The value of t_ℓ can therefore be retrieved by successive testing of the equality of expressions

$$[\tau]\phi_q(P) \text{ and } \phi_q^2(P) \oplus [q_\ell]P \tag{4.4}$$

for all $\tau \in \{0, 1, \ldots, \ell - 1\}$ and *some* point P. Once the expressions are equal, we have found $t_\ell = \tau$. In fact, since $[\tau]P$ and $[-\tau]P$ only differ by the sign of the y-coordinate, we need only test values $\tau \in \{0, 1, \ldots, (\ell - 1)/2\}$.

Knowing all points of E satisfying (4.3) would allow us to simply verify if at least one of them is a point of order ℓ. Unfortunately, finding all points satisfying (4.3) is not computationally efficient – we would need to consider points in $E[\ell]^*$ and since these are the roots of the ℓth division polynomial $\bar{f}_\ell$, working with the finite field of order $q^{\deg \bar{f}_\ell} = q^{\mathcal{O}(\ell^2)}$ might be required. Instead, Schoof's algorithm works with a general point $(x, y) \in E[\ell]^*$ and performs all necessary computations as if x, y where variables in $\mathbb{F}_q[x, y]$. Recall the point $(x, y) \in E$ is in $E[\ell]^*$ if and only if $\bar{f}_\ell(x) = 0$.

Before we can proceed with the algorithm for an odd prime ℓ, we need to decide which addition formula to use. To decide if $\phi_q^2(P) = [\pm q_\ell]P$ for some $P \in E[\ell]^*$, it suffices to compare the x-coordinates of $\phi_q^2(P)$ and $[\pm q_\ell]P$, the former being equal to x^{q^2}, and the latter can be expressed as a rational function $r(x)/s(x)$ for some $r, s \in \mathbb{F}_q[x]$. By multiplying the equality of the x-coordinates by s and subtracting r, we obtain a polynomial equation $\bar{h}_X = 0$ for some $\bar{h}_X \in \mathbb{F}_q[x]$. The equality $\phi_q^2(P) = [\pm q_\ell]P$ then holds for some point $P \in E[\ell]^*$ if and only if $\gcd(\bar{h}_X, \bar{f}_\ell) \neq 1$.

Let us now split the description of the algorithm into two branches, based on what formulas for point addition will be used to compute the expression $\phi_q^2(x, y) \oplus [q_\ell](x, y)$ symbolically. The details of the construction of several polynomials mentioned in the descprition will now be omitted and given after summarizing the algorithm.

1. $\phi_q^2(P) \neq [q_\ell]P$ and $\phi_q^2(P) \neq \ominus[q_\ell]P$ for every $P \in E[\ell]^*$

The sum of $\phi_q^2(x,y)$ and $[q_\ell](x,y)$ in this case is computed by the addition formula for two distinct points and yields an affine point, denote it by (x',y'). Using notation $(x_i,y_i) = [i](x,y)$, we now need to decide if there exists $\tau \in \{1,\ldots,\ell-1\}$ such that $(x',y') = (x_\tau^q, y_\tau^q)$ for some point $(x,y) \in E[\ell]^*$. We will show that the equality holds if and only if $h_X(x) \equiv 0 \pmod{\bar{f}_\ell}$ for some polynomial $h_X \in \mathbb{F}_q[x]$ described below.

To start with, x' can be found by using the addition formula as

$$x' = \left(\frac{y^{q^2} - y_{q_\ell}}{x^{q^2} - x_{q_\ell}}\right)^2 - x^{q^2} - x_{q_\ell}.$$

Secondly, x_τ^q may be expressed by using the formulas for $[n](x,y)$ for a point $(x,y) \in E^*[\ell]$, mentioned in Chapter 2, and applying the Frobenius endomorphism on the result. The relation $x' = x_\tau^q$ can be further modified by substituting $y^2 = x^3 + Ax + B$ to y^i for $i \geq 2$. This is the case for the x-coordinate because the first coordinates of x' and x_τ^q may be expressed as rational functions in x only. Finally, after multiplying the equation by a common multiple of the denominators of the rational functions in the equation, an equation of the form $h_X(x) = 0$ results. It remains to prove that all the computations yielding h_X may be performed modulo $\bar{f}_\ell$.

To see this, recall that for all odd integers, which is especially the case of the primes we consider now, the division polynomial $\psi_\ell = \bar{f}_\ell$ is a polynomial in $\mathbb{F}_q[x]$. Moreover, the only roots of the polynomial are exactly the points in $E[\ell]^*$, and the polynomial is separable, as we have seen in Chapter 2. A crucial corollary of this observation is that testing $h_X(x) = 0$ reduces to deciding whether

$$h_X(x) \equiv 0 \pmod{\bar{f}_\ell} \tag{4.5}$$

since, as has been remarked earlier, if the equality holds for a single point of order ℓ, it holds for all points of $E[\ell]$.

Suppose $\tau \in \{1,\ldots,(\ell-1)/2\}$ satisfying (4.5) is known. However, the equality of the x-coordinates of x' and x_τ^q is not sufficient for the points (x',y'), (x_τ^q, y_τ^q) to be equal; we only know that $(x',y') = (x_\tau^q, y_\tau^q)$ or $(x',y') = \ominus(x_\tau^q, y_\tau^q)$. The former case occurs if and only if $y'/y = y_\tau^q/y$ for all $P = (x,y) \in E[\ell]^*$ ($y \neq 0$ since ℓ is assumed to be odd). The expressions $y'/y, y_\tau^q/y$ can be modified in the same manner as we saw before to yield rational functions in x only, and, after multiplying this equation by a common multiple of the denominators of the rational functions, equation $h_Y(x) \equiv 0 \pmod{\bar{f}_\ell}$ for some $h_Y \in \mathbb{F}_q[x]$ results. Hence, if

$$y'/y - y_\tau^q/y \equiv 0 \pmod{\bar{f}_\ell},$$

then $t_\ell \equiv \tau \pmod{\ell}$, otherwise $t_\ell \equiv -\tau \pmod{\ell}$. This finishes the part for the case when the addition formula for distinct points has to be used. The part to which we now draw our attention not only completes the description of the algorithm, but also reveals what happens if $\tau \equiv 0 \pmod{\ell}$.

2. $\phi_q^2(P) = [q_\ell]P$ or $\phi_q^2(P) = \ominus[q_\ell]P$ for some $P \in E[\ell]^*$

The existence of a point $P \in E[\ell]^*$ satisfying $\phi_q^2(P) = \ominus[q_\ell]P$ immediately gives t_ℓ; $[t_\ell](x^q, y^q) = \phi_q^2(P) \ominus [q_\ell]P = \infty$, and since ℓ is prime, we have $t_\ell \equiv 0 \pmod{\ell}$. Deciding whether $\phi_q^2(P) = \ominus[q_\ell]P$ for some $P \in E[\ell]^*$ can be solved similarly as before by computing $-y_{q_\ell}$ (the y-coordinate of $\ominus[q_\ell]P$), using the division polynomials and transforming the equality to a polynomial one in the form $\bar{h}_Y = 0$ for some $\bar{h}_Y \in \mathbb{F}_q[x]$. The equality then holds for some point in $E[\ell]^*$ if and only if $\gcd(\bar{h}_Y, \bar{f}_\ell) \neq 1$.

Consider now the case when $\phi_q^2(P) = [q_\ell]P$ for some point $P \in E[\ell]^*$. The relation (4.3) for P changes to

$$[t_\ell]\phi_q(P) = \phi_q^2(P) \oplus [q_\ell]P = [2q_\ell]P,$$

which implies $\phi_q(P) = [2q_\ell/t_\ell]P$. This in turn gives

$$[q_\ell]P = \phi_q^2(P) = \phi_q([2q_\ell/t_\ell]P) = [4q_\ell^2/t_\ell^2]P, \tag{4.6}$$

so

$$q_\ell \equiv \frac{4q_\ell^2}{t_\ell^2} \pmod{\ell} \quad \iff \quad t_\ell^2 \equiv 4q_\ell \pmod{\ell},$$

which means that q_ℓ must be a quadratic residue modulo ℓ, i.e. $q_\ell = \tau^2$ for some $\tau \in \mathbb{F}_q$. Therefore, we either have $t_\ell \equiv 2\tau \pmod{\ell}$ or $t_\ell \equiv -2\tau \pmod{\ell}$, that is, either $\phi_q(P) = [\tau]P$ or $\phi_q(P) = \ominus[\tau]P$. To determine which of these two relations holds, we decide if $y^q = y_\tau$ for some $P = (x, y) \in E[\ell]^*$. If not, then $y^q = -y_\tau$. The equation $y^q = y_\tau$ can be transformed to a polynomial one of the form $g_Y = 0$ for some $g_Y \in \mathbb{F}_q[x]$ by multiplying it by the denominator of y_τ. Then the equality $\phi_q(P) = [\tau]P$ holds for some $P \in E[\ell]^*$ if and only if $\gcd(g_Y, \bar{f}_\ell) \neq 1$. Therefore, if the greatest common divisor is non-trivial, we have $t_\ell \equiv 2\tau \pmod{\ell}$. Otherwise g_Y and $\bar{f}_\ell$ are coprime, so $y^q = -y_\tau$ and therefore $t_\ell \equiv -2\tau \pmod{\ell}$.

It remains to describe in detail the construction of the polynomials $h_X, h_Y, \bar{h}_X$, $\bar{h}_Y$ and g_Y. We start with $\bar{h}_X$, which determines whether there exists a point $P \in E[\ell]^*$ such that $\phi_q^2(P) = [\pm q_\ell]P$. Let $\ominus[q_\ell](x, y) = (r_{1,q_\ell}/s_{1,q_\ell}, -y \cdot r_{2,q_\ell}/s_{2,q_\ell})$, where the polynomials $r_{i,q_\ell}, s_{i,q_\ell}, 1 \leq i \leq 2$ are computed by using the formula in Theorem 2.2.2. Since $\phi_q^2(x, y) = (x^{q^2}, y^{q^2}) = (x^{q^2}, y(x^3 + Ax + B)^{(q^2-1)/2})$, we have

$$\bar{h}_X \equiv x^{q^2} s_{1,q_\ell} - r_{1,q_\ell} \pmod{\bar{f}_\ell}$$

and similarly we can obtain $\bar{h}_Y$:

$$\bar{h}_Y \equiv s_{2,q_\ell}(x^3 + Ax + B)^{(q^2-1)/2} + r_{2,q_\ell} \pmod{\bar{f}_\ell}.$$

Next, we express $(x', y') = \phi_q^2(x, y) \oplus [q_\ell](x, y)$. Using the division polynomials we may write $[q_\ell](x, y) = (r_{1,q_\ell}/s_{1,q_\ell}, y \cdot r_{2,q_\ell}/s_{2,q_\ell})$, where $r_{i,q_\ell}, s_{i,q_\ell} \in \mathbb{F}_q[x], 1 \leq i \leq 2$ are given by the formula in Theorem 2.2.2. The x-coordinate of x' can now be expressed as

$$x' = \left(\frac{y^{q^2} - y_{q_\ell}}{x^{q^2} - x_{q_\ell}}\right)^2 - x^{q^2} - x_{q_\ell} = y^2 \left(\frac{y^{q^2-1} - r_{2,q_\ell}/s_{2,q_\ell}}{x^{q^2} - r_{1,q_\ell}/s_{1,q_\ell}}\right)^2 - x^{q^2} - \frac{r_{1,q_\ell}}{s_{1,q_\ell}}$$
$$= \frac{(x^3 + Ax + B)(s_{2,q_\ell}(x^3 + Ax + B)^{(q^2-1)/2} - r_{2,q_\ell})^2 s_{1,q_\ell}^2}{s_{2,q_\ell}^2 (x^{q^2} s_{1,q_\ell} - r_{1,q_\ell})^2} - x^{q^2} - \frac{r_{1,q_\ell}}{s_{1,q_\ell}}$$
$$= \frac{(x^3 + Ax + B)(s_{2,q_\ell}(x^3 + Ax + B)^{(q^2-1)/2} - r_{2,q_\ell})^2 s_{1,q_\ell}^3}{s_{1,q_\ell} s_{2,q_\ell}^2 (x^{q^2} s_{1,q_\ell} - r_{1,q_\ell})^2}$$
$$- \frac{(s_{2,q_\ell}(x^{q^2} s_{1,q_\ell} - r_{1,q_\ell}))^2 (x^{q^2} s_{1,q_\ell} + r_{1,q_\ell})}{s_{1,q_\ell}(s_{2,q_\ell}^2 x^{q^2} s_{1,q_\ell} - r_{1,q_\ell})^2} = \frac{r_{x'}(x)}{s_{x'}(x)}$$

and

$$y' = \left(\frac{y^{q^2} - y_{q_\ell}}{x^{q^2} - x_{q_\ell}}\right)(x_{q_\ell} - x') - y_{q_\ell} = y \left(\frac{y^{(q^2-1)/2} - r_{2,q_\ell}/s_{2,q_\ell}}{x^{q^2} - r_{1,q_\ell}/s_{1,q_\ell}}\right)\left(\frac{r_{1,q_\ell}}{s_{1,q_\ell}} - \frac{r_{x'}}{s_{x'}}\right)$$
$$- \frac{r_{2,q_\ell}}{s_{2,q_\ell}} = \frac{(s_{2,q_\ell}(x^3 + Ax + B)^{(q^2-1)/2} - r_{2,q_\ell})(r_{1,q_\ell} s_{x'} - s_{1,q_\ell} r_{x'})}{s_{1,q_\ell} s_{x'} (x^{q^2} s_{1,q_\ell} - r_{1,q_\ell})}$$
$$- \frac{(r_{2,q_\ell} s_{x'} (x^{q^2} s_{1,q_\ell} - r_{1,q_\ell}))}{s_{1,q_\ell} s_{x'} (x^{q^2} s_{1,q_\ell} - r_{1,q_\ell})} = \frac{r_{y'}(x)}{s_{y'}(x)}.$$

Let

$$[\tau]\phi_q(x, y) = (x_\tau^q, y_\tau^q) = \left(\left(\frac{r_{1,\tau}(x)}{s_{1,\tau}(x)}\right)^q, y\left(\frac{r_{2,\tau}(x)}{s_{2,\tau}(x)}\right)^q\right)$$
$$= \left(\frac{r_{1,\tau}(x^q)}{s_{1,\tau}(x^q)}, y \cdot \frac{r_{2,\tau}(x^q)}{s_{2,\tau}(x^q)}(x^3 + Ax + B)^{(q-1)/2}\right)$$

where $r_{i,\tau}, s_{i,\tau} \in \mathbb{F}_q[x], 1 \leq i \leq 2$ are polynomials computed by the formulas for $[\tau](x, y)$. The last equality holds because ϕ_q is a homomorphism on polynomials over the finite field $\mathbb{F}_q$.

The polynomial h_X is obtained by modifying the equality $x' = x_\tau^q$ so that the equality takes the form of a polynomial one. Thus

$$h_X \equiv r_{x'} s_{1,\tau}^q - r_{1,\tau}^q s_{x'} \pmod{\bar{f}_\ell}$$

and similarly

$$h_Y \equiv r_{y'} s_{2,\tau}^q - s_{y'} r_{2,\tau}^q (x^3 + Ax + B)^{(q-1)/2} \pmod{\bar{f}_\ell}.$$

Finally, we construct g_Y for checking if $\phi_q(P) = [\tau]P$ or $\phi_q(P) = [-\tau]P$ for some point $P \in E[\ell]^*$. At this stage of the algorithm we may suppose the x-coordinates of $\phi_q^2(P)$ and $[\pm\tau]P$ are equal, so the polynomial g_Y compares only their y-coordinates. Since $\phi_q(x, y) = (x^q, y^q) = (x^q, y(x^3 + Ax + B)^{(q-1)/2})$, by letting $[\tau](x, y) = (r_{1,\tau}/s_{1,\tau}, y \cdot r_{2,\tau}/s_{2,\tau})$ we can immediately see, after cancelling y, that

$$g_Y \equiv (x^3 + Ax + B)^{(q-1)/2} s_{2,\tau} - r_{2,\tau} \pmod{\bar{f}_\ell}.$$

Schoof's algorithm is summarized in the pseudocode block below.

Algorithm: Schoof's algorithm

Input : Prime power $q = p^n, n \geq 1$, elliptic curve E given by the short Weierstrass equation $y^2 = f(x)$, where $f(x) = x^3 + Ax + B$, over the finite field $\mathbb{F}_q$.

Output: The order of $E(\mathbb{F}_q)$

$S \leftarrow \{\}$; $t_\ell \leftarrow 0$
// Determine $t \pmod 2$
if $\gcd(x^q - x, f) = 1$ **then** $t_\ell \leftarrow 1$
$S \leftarrow S \cup \{(t_\ell, 2)\}$
// Determine $t_\ell \equiv t \pmod \ell$ for odd prime ℓ
$m \leftarrow 2$; $\ell \leftarrow 3$
while $m \leq 4\sqrt{q}$ **do**
    $q_\ell \leftarrow q \pmod \ell$ so that $|q_\ell| < \ell/2$
    Precompute division polynomials modulo $\bar{f}_\ell$, $x^q, y^q, x^{q^2}, y^{q^2}$ modulo $\bar{f}_\ell$ and division polynomials evaluated at (x^q, y^q) modulo $\bar{f}_\ell$
    Construct $\bar{h}_X(x)$
    if $\gcd(\bar{h}_X, \bar{f}_\ell) \neq 1$ **then**
        Construct $\bar{h}_Y(x)$
        if $\gcd(\bar{h}_Y, \bar{f}_\ell) \neq 1$ **then**
            $t_\ell \equiv 0 \pmod \ell$
        else
            /* Here it is guaranteed that $q_\ell = \tau^2$ for some $\tau \in \mathbb{F}_\ell$ and $\phi_q(P) = [\pm\tau]P$ for some $P \in E[\ell]^*$ */
            Find $\tau \in \mathbb{F}_\ell$ such that $q_\ell \equiv \tau^2 \pmod \ell$
            Construct $g_Y(x)$
            if $\gcd(g_Y, \bar{f}_\ell) \neq 1$ **then** $t_\ell \equiv 2\tau \pmod \ell$
            else $t_\ell \equiv -2\tau \pmod \ell$
    else
        /* At this point we may suppose the points $\phi_q^2(P)$ and $[q_\ell]P$ are distinct for all $P \in E[\ell]^*$ */
        for $\tau = 1$ **to** $(\ell - 1)/2$ **do**
            Construct $h_X(x)$
            if $h_X \equiv 0 \pmod{\bar{f}_\ell}$ **then**
                Construct $h_Y(x)$
                if $h_Y \equiv 0 \pmod{\bar{f}_\ell}$ **then** $t_\ell \equiv \tau \pmod \ell$
                else $t_\ell \equiv -\tau \pmod \ell$
                break
        end
        $S \leftarrow S \cup \{(t_\ell, \ell)\}$
        $\ell \leftarrow$ nextprime(ℓ)
        if $\ell = p$ **then** $\ell \leftarrow$ nextprime(ℓ)
end
Obtain $\hat{t} \equiv t \pmod m$ by applying CRT to the congruence system given by S. Choose $\hat{t}$ such that $|\hat{t}| \leq 2\sqrt{q}$.
return $q + 1 - \hat{t}$

4.2 Time Complexity of Schoof's Algorithm

From the description of Schoof's algorithm, one may easily guess that most time of the algorithm is spent on the arithmetics of polynomials over the finite field $\mathbb{F}_q$. We will first give a generalized time complexity of Schoof's algorithm in terms of the time complexity of relevant operations in $\mathbb{F}_q$ and $\mathbb{F}_q[x]$. Once we have a general formula for the time complexity of Schoof's algorithm, we will prove a basic upper bound on the complexity. Finally, we will briefly mention that better upper-bounds of the complexity can be theoretically attained.

To start with, we need the following theorem [11] to give an estimate on the number of loops of the main cycle in the pseudocode above.

Theorem 4.2.1. *(The Prime Number Theorem) If $\pi(x)$ denotes the number of primes less than or equal to $x \in \mathbb{R}$, then $\pi(x)$ satisfies*

$$\pi(x) \sim \frac{x}{\ln x}$$

in the sense that

$$\lim_{x\to\infty} \frac{\pi(x)}{\frac{x}{\ln x}} = 1.$$

It immediately follows from Theorem 4.2.1 that there are about $\mathcal{O}(\log n/\log\log n) \subseteq \mathcal{O}(\log n)$ primes of size at most $\log n$ among the first $\log n$ natural numbers.

Before we begin with the analysis of the time complexity of Schoof's algorithm, let us for the purpose of the analysis denote by $M(q)$ and $m(q)$ the costs of multiplication in $\mathbb{F}_q[x]$ and $\mathbb{F}_q$, respectively. Throughout the proof, the notions "steps" and "bit operations" will be freely interchanged, the latter being the more precise of the two.

Theorem 4.2.2. *Schoof's algorithm finds the order of $E(\mathbb{F}_q)$ in $\mathcal{O}(M(q)\log^2 q)$.*

Proof. The main loop (starting at line 5) of Schoof's algorithm processes $\mathcal{O}(\log q)$ primes; $\mathcal{O}(\log 4\sqrt{q}) = \mathcal{O}(\log q)$ primes are needed to make m large enough, and their existence is expected by the previous observation, which says that there are $\mathcal{O}(4\log\sqrt{q}) = \mathcal{O}(\log q)$ primes of size $\mathcal{O}(\log q)$ among the first $\log 4\sqrt{q}$ natural numbers. For the rest of the proof, we will assume ℓ is a prime of size $\mathcal{O}(\log q)$.

What makes Schoof's algorithm fast is the use of the small degree of division polynomials. In particular, we know that $\deg \bar{f}_\ell \in \mathcal{O}(\ell^2) = \mathcal{O}(\log^2 q)$, which is also true for all polynomials used in the formula for the multiplication-by-n endomorphism. Moreover, even though x^q, y^q, x^{q^2} and y^{q^2} have degrees linear or even quadratic in q, computing the powers by applying the binary exponentiation mod $\bar{f}_\ell$ (in case of y's after the substitution of the equation of the elliptic curve) requires only $\mathcal{O}(M(q)\log q)$ steps. Hence, the number of steps performed in a single run of the main loop is fully determined by the number of multiplications carried out in $\mathbb{F}_q[x]/(\bar{f}_\ell)$. Therefore, we shall now look more closely at how many products are needed in various parts of the algorithm:

- Precomputing necessary division polynomials by using the recursive formulas for further computations costs $\mathcal{O}(\ell) = \mathcal{O}(\log q)$ multiplications in $\mathbb{F}_q[x]$. Hence the total cost of this part is $\mathcal{O}(M(q)\log q)$.

- Computing $x^q, x^{q^2}, y^q, y^{q^2} \bmod \bar{f}_\ell$ requires $\mathcal{O}(\log q)$ multiplications, giving the total complexity of $\mathcal{O}(M(q)\log q)$.
- After computing the Frobenius maps above, the construction of the polynomials $h_X, h_Y, \bar{h}_X, \bar{h}_Y$ requires only a constant number of multiplications. The construction can thus be accomplished by performing $\mathcal{O}(M(q))$ steps.
- The Euclidean algorithm for computing the greatest common divisor of polynomials of degree $\mathcal{O}(\ell^2)$ performs $\mathcal{O}(\log \ell^2) = \mathcal{O}(\log\log q)$ remainder evaluations, the cost of which is negligible.
- Finding the square root mod ℓ can be achieved by simple trial-and-error, yielding the time complexity of $\mathcal{O}(m(q)\log q) \subseteq \mathcal{O}(M(q)\log q)$.

Finally, nextprime runs in almost constant time because of the size of primes the algorithm considers, and recovering t by using the Chinese Remainder Theorem can also be achieved fairly quickly ($\mathcal{O}(\log q)$ multiplications and inverses in $\mathbb{F}_q$, which is negligible in comparison to the rest).

To summarize, precomputing division polynomials and the Frobenius maps takes $\mathcal{O}(M(q)\log q)$ steps, the part spanning lines 8-17 requires $\mathcal{O}(M(q))$ steps, and the body of the for loop (lines 20-25), being repeated exactly $(\ell-1)/2 \in \mathcal{O}(\log q)$, runs in $\mathcal{O}(M(q))$. We therefore reach the conclusion that the overall time complexity of Schoof's algorithm is $\mathcal{O}(\log q \cdot (M(q)\log q)) = \mathcal{O}(M(q)\log^2 q)$. □

We can now discuss the running time of Schoof's algorithm for specific types of algorithms used for multiplication in $\mathbb{F}_q$ and $\mathbb{F}_q[x]$. The most straightforward implementation of multiplication for polynomials of degree at most d costs $\mathcal{O}(d^2)$ multiplications of elements in $\mathbb{F}_q$, and the same operations in $\mathbb{F}_q$ require $\mathcal{O}(\log^2 q)$ bit operations. Consequently, the overall time complexity of multiplication of polynomials used in Schoof's algorithms of degree $d \in \mathcal{O}(\log^2 q)$ is $\mathcal{O}(\log^6 q)$. We have just showed the following:

Theorem 4.2.3. *Schoof's algorithm finds the order of $E(\mathbb{F}_q)$ in $\mathcal{O}(\log^8 q)$.*

Of course, more advanced algorithms for multiplication in $\mathbb{F}_q[x]$ which are asymptotically much faster than the trivial one can be considered. As an example, the algorithm described in [12] multiplies two polynomials of degree $\log^2 q$ in $\mathcal{O}(\log^3 \log\log q)$ bit operations, under the assumption of a hypothesis regarding the least prime in an arithmetic progression. Plugging this complexity into Theorem 4.2.2 causes the overall running time of Schoof's algorithm to drop to $\mathcal{O}(\log^5 q \log\log q)$.

5. Implementation of Schoof's Algorithm

Now that we have explained Schoof's algorithm for counting rational points on an elliptic curve, we can move forward to describe our implementation [13] of the algorithm. Firstly, we will list technologies we decided to use for the implementation and argue why this choice was appropriate one. Next, the design of the program, involving the structures, classes and organization of the code, will be discussed. The discussion will be followed by investigation of the performance of the implementation, which will be supported by measurements of its actual speed on elliptic curves over finite fields of gradually larger sizes.

5.1 Dependencies

To start with, C++ has been chosen as the major and only programming language for the purposes of the implementation. Since the standard of the language does not offer an efficient multi-precision arithmetics on arbitrarily large numbers as well as a support for finite fields and polynomials, an external library for these operations and objects must have be provided. From the list of plausible libraries, two most popular stand out: FLINT [14] (a C++ wrapper, to be precise) and NTL [5].

Schoof's algorithm depends heavily on the implementation of various polynomial operations over finite fields, mainly multiplication, exponentiation, and Euclidean algorithm for the greatest common divisor. We therefore need to compare both libraries especially on these operations.

Fortunately, NTL's main page [15] explicitly gives a comparison of NTL and FLINT, which evaluates the performance of NTL and FLINT in these polynomial operations. The tables in Figures 5.1 to 5.4 display their relative speed as the ratios FLINT-time/NTL-time.

$k/1024$	$n/1024$												
	1/4		1/2		1		2		4		8		16
1/4	**2.76**	**2.48**	**2.84**	**2.57**	**2.54**	**2.46**	**2.61**	**2.48**	**2.59**	**2.53**	**2.55**	**2.30**	**2.52**
1/2	1.45	1.57	1.56	1.79	1.74	**2.07**	**2.08**	**2.33**	**2.44**	**2.21**	**2.54**	**2.32**	**3.26**
1	1.07	1.12	1.11	1.24	1.22	1.42	1.40	1.86	1.85	1.99	**2.94**	**2.26**	**2.83**
2	0.83	0.85	0.84	0.90	0.88	0.98	0.97	1.20	1.17	1.63	1.60	1.75	**2.17**
4	0.98	1.00	0.96	1.00	1.00	1.00	0.99	1.07	1.06	1.23	1.14	1.43	1.41
8	1.05	1.04	1.03	1.02	1.00	0.98	0.97	0.95	0.94	1.02	0.98	0.95	0.94
16	0.96	0.97	0.97	0.97	0.96	0.96	0.94	0.93	0.91	0.87	0.85	0.91	0.89

Figure 5.1: Multiplication in $\mathbb{F}_p[x]$: n = degree bound, k = #bits in p

$k/1024$	$n/1024$												
	1/4		1/2		1		2		4		8		16
1/4	**4.31**	**3.73**	**4.25**	**3.74**	**4.00**	**3.57**	**3.94**	**3.61**	**4.04**	**3.66**	**3.88**	**3.36**	**3.88**
1/2	**2.22**	**2.27**	**2.36**	**2.61**	**2.68**	**3.12**	**3.21**	**3.49**	**3.84**	**3.27**	**3.94**	**3.44**	**5.09**
1	1.62	1.64	1.68	1.81	1.85	**2.13**	**2.15**	**2.79**	**2.83**	**3.00**	**4.58**	**3.20**	**4.31**
2	1.28	1.24	1.27	1.34	1.34	1.49	1.51	1.82	1.83	**2.39**	**2.54**	**2.48**	**3.22**
4	1.50	1.47	1.49	1.47	1.50	1.47	1.50	1.60	1.63	1.77	1.84	**2.08**	**2.07**
8	0.76	0.78	0.78	0.79	0.79	0.76	0.78	0.81	0.84	0.87	0.88	1.01	0.99
16	0.58	0.57	0.58	0.58	0.58	0.58	0.58	0.60	0.59	0.61	0.59	0.66	0.65

Figure 5.2: Multiplication in $\mathbb{F}_p[x]/(f)$: n = degree bound, k = #bits in p

$k/1024$	$n/1024$												
	1/4		1/2		1		2		4		8		16
1/4	**4.14**	**3.66**	**4.31**	**3.72**	**3.90**	**3.62**	**4.01**	**3.68**	**4.11**	**3.81**	**4.12**	**3.66**	**4.18**
1/2	**2.38**	**2.43**	**2.51**	**2.77**	**2.85**	**3.36**	**3.47**	**3.71**	**4.05**	**3.58**	**4.20**	**3.90**	**5.27**
1	1.78	1.82	1.87	**2.03**	**2.05**	**2.33**	**2.42**	**3.11**	**3.15**	**3.43**	**4.93**	**3.58**	**4.79**
2	1.43	1.43	1.51	1.51	1.55	1.67	1.70	**2.05**	**2.06**	**2.82**	**2.92**	**2.85**	**3.45**
4	1.62	1.65	1.65	1.61	1.67	1.61	1.66	1.77	1.80	1.89	**2.01**	**2.33**	**2.31**
8	0.85	0.89	0.86	0.89	0.86	0.90	0.87	0.89	0.94	0.97	0.94	1.08	1.09
16	0.63	0.64	0.64	0.64	0.65	0.64	0.63	0.66	0.66	0.67	0.66	0.73	0.72

Figure 5.3: Squaring in $\mathbb{F}_p[x]/(f)$: n = degree bound, k = #bits in p

$k/1024$	$n/1024$												
	1/4		1/2		1		2		4		8		16
1/4	1.51	1.60	1.84	1.86	**2.18**	**2.10**	**2.52**	**2.31**	**2.75**	**2.46**	**2.96**	**2.59**	**3.11**
1/2	1.40	1.47	1.58	1.64	1.75	1.77	1.88	1.89	**2.06**	**2.07**	**2.28**	**2.25**	**2.45**
1	1.22	1.30	1.31	1.38	1.41	1.45	1.49	1.53	1.60	1.63	1.69	1.73	1.83
2	1.13	1.24	1.17	1.28	1.22	1.29	1.25	1.33	1.30	1.37	1.36	1.42	1.42
4	1.16	1.30	1.27	1.37	1.33	1.40	1.37	1.45	1.42	1.45	1.47	1.48	1.52
8	1.00	1.14	1.01	1.13	1.02	1.10	1.02	1.08	1.00	1.06	1.00	1.03	0.98
16	0.95	1.08	0.92	1.04	0.89	0.99	0.87	0.97	0.86	0.94	0.85	0.91	0.84

Figure 5.4: Computing GCDs in $\mathbb{F}_p[x]$: n = degree bound, k = #bits in p

The degree of polynomials considered in Schoof's algorithm is $\mathcal{O}(\log^2 q)$, and since the elliptic curves used nowadays [17] are mostly defined over finite fields of order about 256 bits, the tables clearly suggest that NTL is considerably faster than FLINT when considering operations of our interest. We have therefore created an implementation of Schoof's algorithm in C++ supported by NTL.

5.2 Source Code Organization

The source code of the implementation of Schoof's algorithm is divided into two files: `ellc.cpp` and `formulas.cpp`. The former file contains major part of the implementation while the latter provides formulas for Schoof's algorithm, namely for computing

- division polynomials and division polynomials modulo $\bar{f}_\ell$,

- $x^q, x^{q^2}, y^q, y^{q^2} \pmod{\bar{f}_\ell}$
- $[n]P \pmod{\bar{f}_\ell}$ for n not divisible by ℓ,
- $[n]P \pmod{\bar{f}_\ell}$ for n not divisible by ℓ, and
- $\phi_q^2(P) \oplus [q_\ell]P$,

where $P = (x, y) \in E(\mathbb{F}_q)$ and ℓ is an odd prime. These formulas mostly perform some computations on an elliptic curve, so they are quite specific. For this reason, they have been moved to this file to improve the code's readability.

There are three other files located in `src` directory: `demo.cpp`, `utils.cpp` and `tests.cpp`. `demo.cpp` demonstrates usage of counting points on a Weierstrass curve; `utils.cpp` contains only two simple functions to set polynomials' coefficients and serves as a file for implementing other helper functions. Finally, `tests.cpp` contains customizable tests on which this implementation of Schoof's algorithm was executed.

5.3 Implementation Details

As already indicated, the core of the implementation is located in `ellc.cpp`. Two classes appear in the corresponding header file `ellc.h`: `WEllipticCurve` and `WECComp`. `WEllipticCurve` stands for the class used to store data about the Weierstrass curve over the given finite field.

5.3.1 WEllipticCurve

The class `WEllipticCurve` is very simple: an object of `WEllipticCurve` keeps the order of the finite field, the Weierstrass equation and its coefficients. As for the interface, there is only one non-trivial method, and that is `count_points()`, which implements Schoof's algorithm. The body of this method basically copies the pseudocode of Schoof's algorithm we saw in Chapter 4 and should be straightforward and easy to understand. Nevertheless, there are two things to note to make the body completely clear to the user:

- Checking for $q \leq 2\lfloor\sqrt{q}\rfloor$ is equivalent to testing for $q^2 \leq 4q$, which avoids the use of floating-point numbers and thus having to import a separate library for their support.
- The number of division polynomials needed in one cycle of Schoof's algorithm is at most $\ell+2$, which follows from the formulas used for computation of multiples of a point $(x, y) \in E[\ell]^*$ – the largest index appearing in these formulas is $\ell + 2$.

5.3.2 WEEComp

`WECComp`'s interface consists of methods initializing division polynomials, division polynomials modulo a fixed ℓ-th division polynomial, and also division polynomials modulo ℓ-th division polynomial, evaluated at a point (x^q, y^q) for $(x, y) \in E[\ell]^*$. The presence of the last two methods is necessary; otherwise we would

need to work with polynomials of larger degrees than $\mathcal{O}(\ell^2)$ in the first case, and the evaluation would cost more than generating the division polynomials in the second case.

Apart from the methods for initializing the division polynomials, there are also methods for computing addition of points in $E[\ell]$, the details are well described in the code.

Finally, there are methods for computing the polynomials $h_X, h_Y, \bar{h}_X, \bar{h}_Y$, and g_Y. Their names do not match those we introduced in Chapter 4, see the comments in the code that clarify which polynomial is which.

5.4 Testing

To ensure that the implementation always gives the correct answer for an arbitrary smooth Weierstrass curve over finite field with characteristic distinct from 2 and 3, it was tested against various types of Weierstrass curves. Input data for these tests were created automatically by the Sage script `simple_ec_gen.sage`, which makes calls to SageMath [16] functions for finding the order of the given elliptic curve. It is thus required to have installed SageMath to generate random Weierstrass curves using this script.

The script contains the following function for generating input data:

- `fixed_fq_test`(n, p, e, F) – generates n non-equivalent smooth Weierstrass curves (non-equivalent in terms of j-invariants, see below) given by the Weierstrass equation over the finite field $F = \mathbb{F}_{p^e}$, where $p \notin \{2, 3\}$. The function was mainly used for detecting errors in the implementation.

- `create_test`$(n, \text{nbits}, \text{outf})$ – produces n non-equivalent smooth Weierstrass curves over the finite fields $\mathbb{F}_p$, where $p \in [2^{\text{nbits}-1}, 2^{\text{nbits}})$ and appends them to the file outf. This function was used to generate input data for performance measurements, see below.

5.5 Performance Results

To begin with, all measurements were performed on HP Pavilion Laptop Model 15-eg2051nc, with Intel Core i7-1260P CPU and Arch Linux operating system. The tests devised for performance measurements included only elliptic curves over finite fields $\mathbb{F}_p$ for arbitrary primes p within the range from 4 bits to 128 bits. Not only creating such tests was simpler than generating elliptic curves over extension fields of prime fields, but also elliptic curves found in practice are essentially only considered over $\mathbb{F}_p$, see [17]. For each size of the prime p, a fixed number of mutually non-equivalent elliptic curves (over the algebraic closure of $\mathbb{F}_p$) were generated, say n, so that we could observe the influence of the value of n on the average running time of the algorithm. The equivalence was checked by comparing the *j-invariants* [1] of the curves. The plot in Figure 5.5 displays the results of our measurement for $n \in \{10, 15\}$.

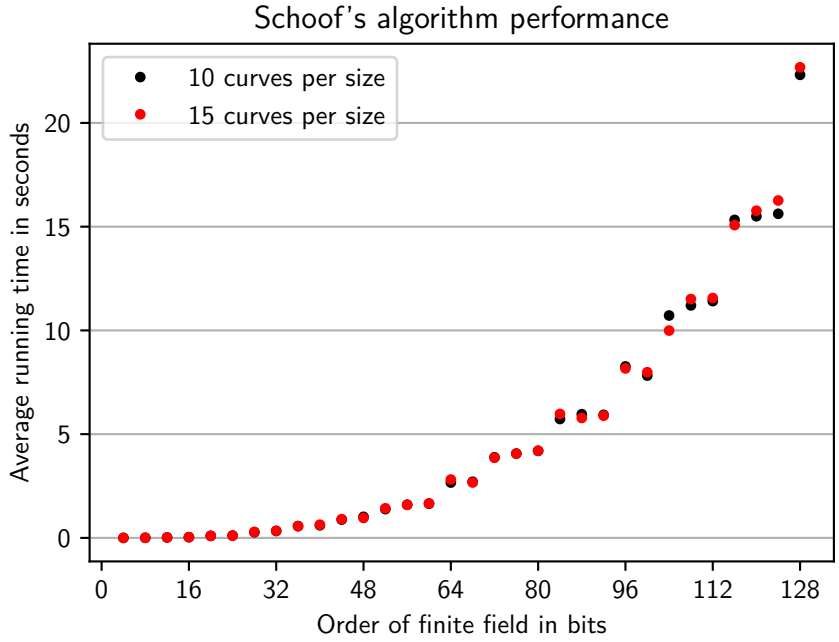


Figure 5.5: Average running time of our implementation of Schoof's algorithm executed on elliptic curves over finite field $\mathbb{F}_p$, where p ranges from 4 bits up to 128 bits.

Once we have these data, it is natural to inspect the actual trend of the measured running time. For this reason, we extended input to involve primes of size up to 156 bits to increase credibility of the estimation of the algorithm's expected running time. This test produced the following plot in Figure 5.6. We also used scipy's `curve_fit` function [18] to find the best fitting curve for our data, which we set to be in the form $y = ax^b + c$ for some parameters $a, b, c \in \mathbb{R}$ since the total time complexity is expected to be in the form of a power of $\log q$.

It should not be surprising that the fitted curve in Figure 5.6, given by $y = 6.18\text{e}{-07}x^{3.57} + 0.347$, differs significantly from the theoretical result about the time complexity of Schoof's algorithm derived in Chapter 4. Leaving aside the hardware's influence on the running time such as retrieving data from caches, accessing memory or optimizations on CPUs, the cost of polynomial operations does not match those we assumed in proving the theoretical time complexity.

For example, the documentation of NTL says that the polynomial arithmetic of univariate polynomials over a finite field $\mathbb{F}_p$ is implemented using the FFT, combined with the Chinese Remainder Theorem. In this way, assuming the time complexity of multiplication of field elements is still $\mathcal{O}(\log^2 q)$, the cost of multiplying polynomials of degree $\mathcal{O}(\log^2 q)$ drops from $\mathcal{O}(\log^4 q)$ to $\mathcal{O}(\log^2 \log \log q)$, yielding the total time complexity of Schoof's algorithm at $\mathcal{O}(\log^6 \log \log q)$. Similarly, a different method for modular exponentiation, namely a sliding window method [19], is used by NTL.

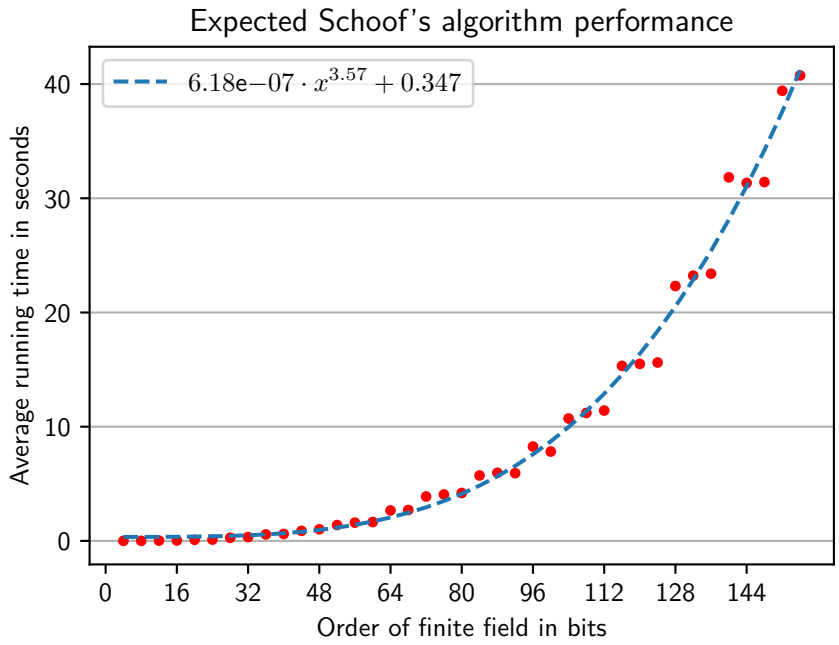


Figure 5.6: Expected running time of the implementation of Schoof's algorithm as a power function of the size of a finite field.

Conclusion

As demonstrated in this thesis, Schoof's algorithm requires an understanding of not only the group of elliptic curves and the addition applied to points on the curve, but also more advanced mathematical concepts such as division polynomials and rational maps on elliptic curves.

After absorbing these topics, we were able to implement the algorithm in its basic version, without additional enhancements such as those proposed by Elkies and Atkins or parallelization. Despite this, we have developed a useful tool for determining the order of elliptic curves over finite fields, which produces results in approximately 20 seconds for curves over finite fields of sizes about 128 bits.

In the future, we plan to extend our implementation by adding these proposed optimizations, aiming to improve the performance of the implementation and come closer to that of mathematical software systems such as SageMath.

Bibliography

[1] Lawrence C. Washington. *Elliptic curves : Number Theory and Cryptography*. Chapman & Hall/CRC, 2008.

[2] Ian F Blake, G Seroussi, and Nigel P Smart. *Elliptic curves in cryptography*. Cambridge University Press, 1999.

[3] René Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of computation*, 44(170):483–494, 1985.

[4] Aleš Drápal. Algorithms on Elliptic Curves. `https://dl1.cuni.cz/pluginfile.php/1574923/mod_resource/content/1/gralg.pdf`, 2022.

[5] NTL: A Library for doing Number Theory. `https://libntl.org/`, 2021.

[6] Joseph H Silverman. *The Arithmetic of Elliptic Curves*. Springer Science & Business Media, 03 2013.

[7] David S Dummit and Richard M Foote. *Abstract Algebra*. John Wiley & Sons, 07 2003.

[8] Ian Stewart and David Tall. *Algebraic number theory and Fermat's last theorem*. CRC Press, 2015.

[9] A.J. Menezes, T. Okamoto, and S.A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39:1639–1646, 1993.

[10] N. P. Smart. The Discrete Logarithm Problem on Elliptic Curves of Trace One. *Journal of Cryptology*, 12:193–196, 06 1999.

[11] D. J. Newman. Simple Analytic Proof of the Prime Number Theorem. *The American Mathematical Monthly*, 87:693, 11 1980.

[12] David Harvey and Joris van der Hoeven. Polynomial Multiplication over Finite Fields in Time $O(n \log n)$. *Journal of the ACM*, 69, 04 2022.

[13] Václav Zvoníček. Zvoníček Václav / Schoofs algorithm · GitLab. `https://gitlab.mff.cuni.cz/zvonicev/schoofs-algorithm`, 2023.

[14] FLINT: Fast Library for Number Theory. `https://www.flintlib.org/index.html`, 11 2023.

[15] Victor Shoup. NTL vs FLINT. `https://libntl.org/benchmarks.pdf`, 02 2021.

[16] SageMath Mathematical Software System - Sage. `https://www.sagemath.org/`, 2023.

[17] Joppe W Bos, J Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, and Eric Wustrow. Elliptic curve cryptography in practice. In *Financial Cryptography and Data Security: 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers 18*, pages 157–175. Springer, 2014.

[18] Scipy documentation — scipy v1.8.1 manual. `https://docs.scipy.org/doc/scipy/`, 02 2023.

[19] C.K. Koç. Analysis of sliding window techniques for exponentiation. *Computers & Mathematics with Applications*, 30, 11 1995.