2008

Counting points on elliptic curves over Zp

Suresh Sundriyal

Counting Points on Elliptic Curves over $\mathbb{Z}_p$

Suresh Sundriyal
Department of Computer Science
Rochester Institute of Technology

January 25, 2008

Committee
Stanisław P. Radziszowski, Chair
Christopher Homan, Reader
Warren R. Carithers, Observer

ROCHESTER INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

Counting Points on Elliptic Curves over $\mathbb{Z}_p$

Prof. Stanisław P. Radziszowski, Chair

Prof. Christopher Homan, Reader

Prof. Warren R. Carithers, Observer

Date

Abstract

The mathematics of elliptic curves has been studied since ancient times. These mathematical structures have found applications in varied fields. More recently, there has been a growing interest in applying these structures to cryptography. Various such applications have been proposed. Elliptic curve based cryptographic algorithms have been shown to provide greater security with shorter key sizes than the conventional RSA based cryptosystems, making them more memory efficient and less processor intensive.

One of the fundamental requirements of all such elliptic curve cryptographic algorithms is that the order of the group of points satisfying the elliptic curve meet a certain set of requirements. However, finding the group order is not a trivial task.

There are numerous special cases of elliptic curves where determining the group order is trivial. This thesis however, deals with the study of general point counting algorithms and their performances, which are applicable to all the curves. Possible improvements to the algorithms are provided where possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Elliptic curves are defined as the set of solutions to a cubic equation having two variables, over various domains. Consider the following equation:

$$y^2 = x^3 + Ax + B \tag{1.1}$$

If the equation 1.1 has distinct roots then the graph of such an equation defines an elliptic curve of interest to cryptography. The above equation is called the simplified Weierstrass equation. Such structures over finite fields $\mathbb{F}_p$, where p is usually a large prime, have very rich theory and they have found uses in many fields, including number theory and cryptography.

These structures were used in the proof of Fermat's last theorem by Andrew Wiles [26]. Hendrik Lenstra formulated an algorithm, which employed elliptic curves to factorize numbers, and is one of the fastest factorization algorithms today, along with the Number Field Sieve method [14].

In 1985 Neal Koblitz [13] and Victor Miller [16] independently suggested the use of elliptic curves in cryptography. Since then various public key cryptographic algorithms, employing elliptic curves, have been formulated.

Most of these algorithms are based on the elliptic curve discrete log problem (ECDLP). An attractive feature of the elliptic curve cryptography (ECC) algorithms is that there is no known sub-exponential algorithm to solve the discrete log problem for elliptic curves. This ensures the security of these algorithms. For more information on ECDLP, see [3].

In comparison to the RSA algorithm, ECC provides the same amount of security with shorter key lengths [3]. Due to the smaller key sizes, these algorithms tend to have lower memory and processor utilization. This makes ECC algorithms ideal for embedded environments, where resources are scarce.

The selection of a secure curve to be used in the elliptic curve algorithms involves knowing the curve order or, in simple words, the total number of points on the said curve. Unfortunately, there is to date no efficient way to count the number of points on a general curve. Major research has been done into the field of point counting on elliptic curves and there are new and exciting results being published quite frequently.

This thesis studies many of these point counting algorithms and also looks at ways to improve their performance. The basic point counting algorithms are studied in Chapter 2 and the advanced algorithms are studied in Chapter 3.

Another way to employ secure curves is to construct curves with the desired characteristics. Chapter 4 provides an overview of the curve construction algorithm.

Before moving on to the more advanced topics, the following sections present the elliptic curve fundamentals required for the rest of this thesis.

1.1 Elliptic curve fundamentals

1.1.1 Weierstrass equation

We shall be studying nonsingular cubic curves defined by the simplified Weierstrass equation, given by (1.1).

The Weierstrass equation defines elliptic curves over a field $\mathbb{F}_p$ as long as the characteristic of $\mathbb{F}_p$ is greater than 3 [7].

The equation (1.1) is said to be nonsingular if it has distinct roots [25]. Suppose a, b and c are the roots of the above equation. This implies that:

$$\begin{aligned} y^2 &= (x-a)(x-b)(x-c) && (1.2)\\ &= x^3 - (a+b+c)x^2 + (ab+ac+bc)x - abc && (1.3) \end{aligned}$$

Comparing the coefficients of the above equation with the Weierstrass equation we can see that the coefficient of x^2 is zero. Thus, we can write

$$a + b + c = 0 \tag{1.4}$$

Now, let us assume that a and b are the same root. Substituting this in (1.4) we get:

$$\begin{aligned} 2a + c &= 0 && (1.5)\\ \Rightarrow a &= \frac{-c}{2} && (1.6) \end{aligned}$$

Substituting the (1.6) in equation (1.3) results in:

$$y^2 = x^3 - \frac{3}{4}c^2x - \frac{1}{4}c^3. \tag{1.7}$$

Substituting $A = \frac{-3}{4}c^2$ and $B = \frac{-1}{4}c^3$ such that $y^2 = x^3 + Ax + B$, we have

$$4A^3 = -\frac{27}{16}c^6 = -27B^2 \tag{1.8}$$

This implies that if the Weierstrass equation has a double root then $4A^2 + 27B^2 = 0$. Thus we can say that if the Weierstrass equation is to have three unique roots then it should satisfy the following condition:

$$4A^3 + 27B^2 \neq 0. \tag{1.9}$$

The negative of the left hand side of the equation (1.9) is also called the *discriminant* of the elliptic curve equation.

1.1.2 Elliptic curve arithmetic

Consider an elliptic curve defined by (1.1) over a field $\mathbb{F}_p$ of characteristic greater than 3, (Refer [A.3] on page 50, for an explanation of finite fields). We shall represent such a curve as follows:

$$E\left(\mathbb{F}_p\right) = \{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p : y^2 = x^3 + Ax + B\} \cup \{O\} \tag{1.10}$$

where O is the point at infinity. Take two points, $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, and the point at infinity denoted by O. We define the commutative operation $(+)$ and the inverse operation $(-)$ as follows [24]:

1. $-O = O$;
2. $-P = (x_1, -y_1)$;
3. $O + P_1 = P_1$;
4. if $P_2 = -P_1$; then $P_1 + P_2 = O$;
5. if $P_2 \neq -P_1$, then $P_1 + P_2 = (x_3, y_3)$, with

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2, \\ y_3 &= \lambda(x_1 - x_3) - y_1, \end{aligned}$$

   where the slope λ is defined by

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{if } x_2 \neq x_1, \\ \\ \frac{3x_1^2 + A}{2y_1}, & \text{if } x_2 = x_1. \end{cases}$$

   Note: in $\mathbb{Z}_p$, the division is performed by multiplying with the inverse of the denominator modulo p.

6. Multiplication of a point $P \in E$ with an integer n is defined as $[n]P = P + P + \ldots + P$, where the number of P on the right is equal to n.

Theorem 1.1.1 $(\mathbb{F}_p, +)$ *is a group*

- *(commutativity)* $P_1 + P_2 = P_2 + P_1$ *for all* P_1, P_2 *on E, [25].*
- *(identity) By definition,* $P + O = P$ *for all points P on E, [25].*
- *(inverse) For every point* $P(x, y)$ *on the curve E there is a point* $P'(x, -y)$*, [25]. Plugging in the values for these in the formulae above we can see that* $P+P' = O$*.*
- *(associativity) It can be shown that* $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$ *for all points* P_1, P_2, P_3 *on E. The proof of associativity is quite complex and beyond the scope of this discussion. For further details refer to [25].*

From the above discussion it becomes clear that the points on E form an additive abelian group.

1.1.3 *j*-invariant

For an elliptic curve defined by the equation 1.1, the j-invariant of the equation is defined as,

$$j = j(E) = 1728\frac{4A^3}{4A^3 + 27B^2} \tag{1.11}$$

There are two special cases for the j-invariant,

1. $j = 0$; in this case the Weierstrass equation is written as $y^2 = x^3 + B$.
2. $j = 1728$; in this case the Weierstrass equation is written as $y^2 = x^3 + Ax$.

Once we know the value of the j-invariant, where ($j \neq 0$ and $j \neq 1728$), we can easily find the value of the parameters A and B with the following equations:

$$A = \frac{-3j}{j - 1728} \tag{1.12}$$

$$B = \frac{2j}{j - 1728} \tag{1.13}$$

Thus if we know the j-invariant of a curve then we can denote the curve as,

$$y^2 = x^3 + \frac{-3j}{j - 1728}x + \frac{2j}{j - 1728}$$

The j-invariant also tells us whether the two curves are isomorphic over an algebraically closed field. If two different elliptic curves defined over a field $\mathbb{F}$ have the same j-invariant, then they are said to be twists of each other.

For further information about finite field arithmetic and elliptic curve fundamentals, refer [7], [25] and [1].

1.1.4 Hasse's theorem

Finding the curve order of elliptic curves is not a trivial task, however Hasse's theorem gives us a bound on the curve order. Hasse's theorem states that for a curve E defined on $\mathbb{Z}_p$, the curve order lies between,

$$[\, p + 1 - 2\sqrt{p}\,, p + 1 + 2\sqrt{p}\,]$$

For very large values of p, this interval can be quite exhaustive but it does narrow the search space and with some clever methods, which we shall see in chapter 2, this search can be performed efficiently.

In order to verify Hasse's bound with an example consider an elliptic curve $y^2 + x + 28$, with $p = 19$. According to Hasse's theorem, the order of this curve should lie within $[11, 29]$. We can easily calculate the points on this curve, with the help of one of the algorithms mentioned in chapter 2, and verify that the actual curve order is 26, which is within the Hasse's bound.

A complete proof of Hasse's theorem is given in [25].

Chapter 2

Counting Points on Elliptic Curves

2.1 Overview

The number of points on an elliptic curve is defined as the cardinality of the set of pairs of points that lie in the finite field $\mathbb{F}_p$ over which the elliptic curve is defined, and which satisfy the equation of the elliptic curve, plus an additional point O to represent the point at infinity. The order of the elliptic curve plays an important role in the selection of the curve for cryptographic uses.

Many of the cryptographic standards select a secure curve based on the curve statistics provided by the curve order. One of the most common requirements is that the order of the curve defined under a finite field $\mathbb{F}_p$ not be a product of small primes. It is usually preferred that the group order be a multiple of large primes or that the group order at least contain a large prime of the order of at least 2^{160}.

SEA algorithm, discussed in chapter 3, is the most efficient algorithm for point counting available today, having a complexity of $O\left(\log^5 p\right)$ [3]. This is generally sufficient for most present day cryptographic usage, but as the speed of the processors keep increasing, the number of bits required in cryptographic keys also increases. Thus, we need better algorithms for point counting to increase the level of security of elliptic curve cryptography (ECC) protocols.

This chapter studies a couple of basic point counting algorithms and their performance and explores new ideas to improve them. The more advanced algorithms are dicussed in chapter 3.

During the course of analyzing the algorithms, it was found that Shanks-Mestre algorithm was the one which lends itself well to computational optimization. Numerous improvements to the Shanks-Mestre algorithm are presented in this chapter and their performance statistics are given.

2.2 Naïve algorithm

It makes sense to start off with the basic algorithm, since it helps to explain the concepts of point counting on an elliptic curve. Although the naïve algorithm is not much more than counting points on the curve one by one, there are a couple of number theoretic shortcuts that we can apply to the whole process. There are better algorithms to accomplish the same task, but they are quite complex. It is much easier to employ the naïve algorithm for counting points on elliptic curves defined modulo primes of up to 25-bits.

Before continuing with the discussion, the reader is advised to read the explanation of the Legendre symbol in (A.8), used in the summation of (2.1).

Consider an elliptic curve $y^2 = x^3 + Ax + B$ defined over a field $\mathbb{F}_p$. If p is a prime, then the order of such an elliptic curve, denoted by $\#E(\mathbb{F}_p)$, is given by the following expression [25]:

$$\#E(\mathbb{F}_p) = p + 1 + \sum_{x \in \mathbb{F}_p} \left(\frac{x^3 + Ax + B}{p} \right) \tag{2.1}$$

Equation (2.1), also known as Lang-Trotter method is explained in detail in [25]. The $\left(\frac{x^3+Ax+B}{p}\right)$ in equation (2.1) represents Legendre symbol, see (A.8). In order to understand the equation (2.1), let us dissect it, element by element. There are p points in the field $\mathbb{F}_p$. For every quadratic residue $y^2 \in \mathbb{F}_p$ (i.e., $\left(\frac{y^2}{p}\right) = 1$), there are two points on the curve, namely $P_1 = (x_1, y_1)$ and $P_2 = (x_1, -y_1)$ and hence a point has to be added to p. Similarly, for every non-quadratic residue $y^2 \in \mathbb{F}_p$ (i.e., $\left(\frac{y^2}{p}\right) = -1$), there are no points on the elliptic curve and hence one point has to be subtracted from p. For every $x \in \mathbb{F}_p$ where $\left(\frac{y^2}{p}\right) = 0$, there is one point on the curve, i.e. $(x, 0)$. Thus, we know the reasoning behind the $p + \sum_{x \in \mathbb{F}_p} \left(\frac{x^3+Ax+B}{p}\right)$ part of the equation.

In addition to the above three situations the point O is implicitly assumed to be on the elliptic curve and hence 1 has to be added to the total to get the complete equation (2.1). Now we are in a position to formulate the first point counting algorithm.

Algorithm 2.2.1: NAÏVE ALGORITHM(A, B, p)

```
procedure LEGENDRE(x, A, B, p)
  y ← x^3 + Ax + B
  l ← y^((p−1)/2) (mod p)
  comment: Euler's criterion (A.7)
  return (l)

main
  t ← 0
  for x ← 0 to (p − 1)
    do t ← t + LEGENDRE(x, A, B, p)
  return (p + 1 + t)
```

Pseudocode taken from [25, 24].

2.2.1 Performance analysis of the naïve algorithm

As can be inferred from the explanation, as well as the algorithm, the naïve algorithm goes through all the points $x \in \mathbb{F}_p$ and checks how many valid values for y exist. This is the simplest form of the point counting algorithm, but with a complexity of $O(p)$, it is not efficient for large values of p. For a field $\mathbb{F}_p$ having an extremely large characteristic p, the iteration of the loop is formidable. If we were to consider counting points on a cryptographic curve, the calculations would be in the magnitude of 2^{160} and above, which would be impossible to perform within reasonable time.

2.2.2 Statistics and results

In order to test the performance of the naïve algorithm, an implementation was written in C++. The Number Theory Library [21], which provides an efficient implementation of various number theory functions was used for the calculating the Legendre symbol. The timing tests were done on an Intel P4, dual-core machine with 1 GB of RAM. In order to improve the performance, the optimization flags provided by the GNU C++ compiler were used.

The following graph shows the time taken by the algorithm to finish, plotted against the increasing values of number of bits of the characteristic p defining the curve. The curve used was $y^2 = x^3 + x + 28$.

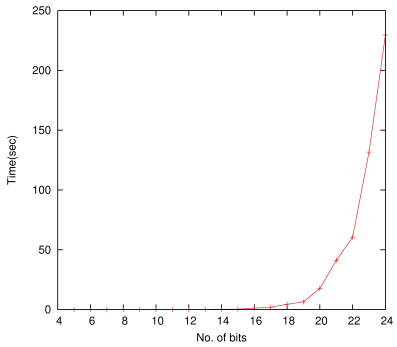

Figure 2.1: Performance of naïve algorithm.

The numerical data relating to the performance of the algorithm is given in the table 2.1.

Naïve algorithm			
Bit-size (lg p)	Prime (p)	Time (sec)	Number of points
5	19	0.046	26
8	241	0.046	223
10	659	0.046	653
12	2707	0.046	2644
14	10009	0.266	10089
16	35051	1.282	35187
18	252937	4.625	252765
20	529043	17.64	527878
21	1622669	40.985	1625052
22	2317919	60.547	2315065
23	7029599	131.687	7028642
24	10162723	229.547	10166426

Table 2.1: Performance statistics of naïve algorithm.

It is obvious that it is not feasible to count points on curves defined over large finite fields using this algorithm. However, for fields defined for 20-bit primes or less, the naïve algorithm is the most practical choice considering the ease of implementation and the fact that it will also generate all the coordinates if they are needed. This feature of the algorithm is useful for educational purposes.

2.3 Shanks-Mestre algorithm

The Shanks-Mestre algorithm is the end product of two different ideas by Daniel Shanks and Jean-Francois Mestre. Shank's Baby-Step Giant-Step algorithm [20] has been successfully employed to calculate discrete logarithms in the modular fields $\mathbb{Z}_p$. Using the same basic principles, and by employing a theorem by Mestre, we can modify the discrete log Baby-Step Giant-Step algorithm to calculate the points on an elliptic curve.

The Baby-Step Giant-Step algorithm makes for a nice point counting algorithm with a time complexity of $O\left(p^{\frac{1}{4}}\right)$, however it also has a space complexity associated with it which also equals $O\left(p^{\frac{1}{4}}\right)$. So, although it is faster than the naïve algorithm, it also utilizes a lot of system memory.

The adaptation of Shank's algorithm to elliptic curves is based on a theorem due to Hasse regarding the bound on the order of the elliptic curve [25]. Hasse's theorem states that the number of points on an elliptic curve defined over a finite field $\mathbb{F}_p$, henceforth denoted by $\#E(\mathbb{F}_p)$, lies in the interval:

$$[\, p + 1 - 2\sqrt{p}\,, p + 1 + 2\sqrt{p}\,] \tag{2.2}$$

These bounds imply that the exact order of the group can be given by the following equation:

$$\#E(\mathbb{F}_p) = p + 1 + l, \qquad \textit{where} \quad l \in [-2\sqrt{p}, 2\sqrt{p}] \tag{2.3}$$

Let us write $k = \beta + \gamma W$ where $W = \lceil \sqrt{2}p^{\frac{1}{4}} \rceil$, β ranges over $[0, W-1]$ and γ ranges from $[0, W]$. Now by the group law we know that for any $P \in E(\mathbb{F}_p)$ we have,

$$\begin{aligned} [\#E(\mathbb{F}_p)]P = O \quad &\Longleftrightarrow \quad [p + 1 + \beta \pm \gamma W]P = O \\ &\Longleftrightarrow \quad [p + 1 + \beta]P = \mp[\gamma W]P \end{aligned} \tag{2.4}$$

Now we form a list of x-coordinates of the points, such that

$$\{[p + 1 + \beta]P : \beta \in [0, \ldots, W - 1]\} \tag{2.5}$$

Call this list A, and a second list of x-coordinates list B such that

$$\{[\gamma W]P : \gamma \in [0, \ldots, W]\} \tag{2.6}$$

We then sort these two lists and look for a match. If a match is found, we can check to see which one of $[p + 1 + \beta \pm \gamma W]P$ is the point at infinity.

However, we are not done at this point. We have found a multiple of P for which $P = O$. By Lagrange's theorem (A.9) we know that the order of a point always divides the group order. This means that we have found the order of the point P or its multiple. This may not necessarily be the curve order. If it so happens that the point under consideration has low order, Shank's method will find multiple matches in the list. However, if the order of the point is greater than $4\sqrt{p}$, the algorithm will find the unique match. For details into this approach, refer to [25].

Occasionally it may also happen that all the points on the curve will have low order, and in such a case the algorithm will never find a unique match. The way around this conundrum is based on a result by Mestre [7].

Before we proceed further, let us define the twist of an elliptic curve. The twist of an elliptic curve $E_{a,b}$ is defined by $E_{a',b'}$ where $a' = g^2a, b' = g^3b$ for any quadratic nonresidue (A.6) $g \in \mathbb{F}_p$. The twist of an elliptic curve is isomorphic to the elliptic curve $E_{a,b}$. For more details refer [3].

Theorem 2.3.1 (Mestre) *For an elliptic curve $E(\mathbb{F}_p)$ and its twist $E'(\mathbb{F}_p)$, which is governed by a quadratic non-residue modulo p,*

$$\#E(\mathbb{F}_p) + \#E'(\mathbb{F}_p) = 2p + 2$$

Furthermore, when $p > 457$ there exists a point P on at least one of the curves such that $|P| > 4\sqrt{p}$. If $p > 229$ at least one of the two curves possess a point P such that the only integer $m \in (p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p})$ having $[m]P = O$ is the actual curve order.

The importance of this theorem lies not in the result that $\#E(\mathbb{F}_p)+\#E'(\mathbb{F}_p) = 2p+2$ but in the second part of the statement which states that there exists a point on at least one of the curves with the property that the single match m is the curve order.

Thus, we need to modify the Baby-Step Giant-Step algorithm to discard the original point and start off with another point on one of the curves until the intersection of the two lists produces only a single match. Finding such a point randomly is not hard since the number of points satisfying the criteria exceed $cp/\ln\ln p$ [7].

Another interesting observation to be made is the comment regarding p being greater than 229. There are cases when p is less than 229 where there does not exist a single point meeting the required criteria [7]. Thus, for primes less than 229 we have to resort to the naïve algorithm to get the curve order.

Keeping in mind the above discussion, we are now in a position to formulate a pseudocode version of the Shanks-Mestre algorithm. The pseudocode is provided as follows. The $x(P)$ is used to denote the x-coordinate of a particular point on the curve.

Algorithm Shanks - Mestre

Input: a, b, p
Output: $\#E_{a,b}$

1. [Check magnitude of p]
   if ($p \leq 229$) return $p + 1 + \sum_{x \in \mathbb{F}_p} \left(\frac{x^3+ax+b}{p}\right)$; //naïve algorithm.

2. [Initialize Shank search]
   Find a quadratic nonresidue g ($mod\ p$);
   $W = \lceil \sqrt{2}p^{\frac{1}{4}} \rceil$; //giant-step parameter.
   $(c, d) = (g^2a, g^3b)$; //twist parameter.

3. [Mestre Loop]
   Choose random $x \in [0, p-1]$;
   $\sigma = Legendre\left(\frac{x^3+ax+b}{p}\right)$;
   if ($\sigma == 0$) goto [Mestre Loop];
   if ($\sigma == 0$) $E = E_{a,b}$ //set original curve.
   else{
   $E = E_{c,d}$;
   $x = gx$; //set twist curve and valid x
   }
   Define an initial point $P \in E$ to have $x(P) = x$;
   $S = Shanks(P, E)$; //Shanks intersection.
   if ($\#S \neq 1$) goto [Mestre Loop]; //exactly 1 match is needed.
   Set s as the unique element of S;
   $\beta = ind(A, s); \gamma = ind(B, s)$; //find indices of unique match.
   Choose sign in $t = \beta \pm \gamma W$ such that $[p + 1 + t]P = O$ on E;
   return $p + 1 + \sigma t$; //desired order of $E_{a,b}$

4. Function *Shanks*()
   Shanks(P,E) {
   $A = \{x([p + 1 + \beta]\, P) : \beta \in [0, W - 1]\}$; //baby steps.
   $B = \{x([\gamma W]\, P) : \gamma \in [0, W]\}$; //giant steps.
   return $A \cap B$;
   }

Pseudocode taken from [7].

We are only interested in the x-coordinates for the matches. For every point x that we find there are two possible values for y. This ambiguity does not affect the algorithm in any way, since once we know the point x, we can easily find the point y by substituting it in the Weierstrass equation (1.1). Since the x-coordinates are the only ones that need to be stored, the two lists can be ordered by increasing values of x and the search can thus be optimized.

After the explanation of Shanks-Mestre algorithm it must be mentioned that the algorithm is probabilistic in nature and there are chances that the randomly chosen point will not have the desired properties. In such a case the algorithm will take longer than expected to reach a conclusion. There is a way to remedy this problem and we shall be discussing it in the next section concerning the improvements to this algorithm.

2.3.1 Improvements to Shanks-Mestre algorithm

It is easy to see that the algorithm is a definite improvement over the naïve algorithm. However, it is not without its own set of problems. The algorithm requires us to find point multiples and this itself is cumbersome to begin with. To add to this, if we are to recall the elliptic curve arithmetic, point addition requires us to calculate inverses in the modular field. Finding modular inverses is a time consuming process and it is closely related to the Extended Euclidean algorithm, which has a time complexity of $O(log^2 p)$ [12]. This can be quite crippling to the performance of the algorithm.

Just to show the degrading effect the modular inverse operation has on the algorithm, the *InvMod()* function provided by the Number Theory Library (NTL) [21] was iterated a million times over numbers with increasing number of bits. The graph of the same is shown below.

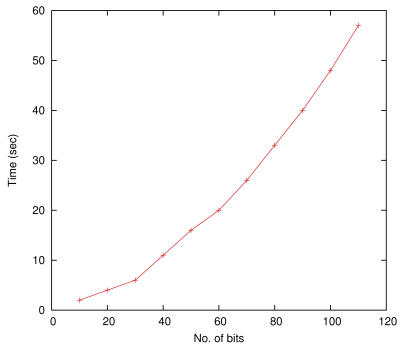

Figure 2.2: Performance of the Modular Inverse operation in the NTL Library.

Looking at the graph above, the debilitating effect on the performance of the algorithm is quite clear. If we are to improve its performance, one of the first things we need to do is to somehow reduce the number of modular inverse operations in the algorithm. The following ways were used in the implementation of the algorithm to optimize the performance:

2.3.1.1 Methods to reduce modular inverse operations

1. **Projective Coordinates**
   A detailed explanation and proofs relating to the various coordinate systems that can be used with the elliptic curve algebra can be found in [3]. Here we shall only be discussing their use with regards to Shanks-Mestre algorithm.

Instead of working in the standard (x, y) coordinate system it is possible to represent the points in a projective coordinate system where the points are represented in projective (X, Y, Z) coordinates. One of the upshots of using such coordinate systems is that arithmetic with the help of such coordinate systems does not require the need for modular inverses. Several coordinate systems have been proposed for use in elliptic curve arithmetic and depending on the requirement and the ease of implementation, any one of the systems can be chosen.

Looking at the Shanks-Mestre algorithm we notice that throughout the algorithm we are only keeping track of the x-coordinate and discarding the y-coordinate. This ambiguity in the y-coordinate is acceptable in Shanks-Mestre algorithm and this property leads us to the selection of Montgomery projective coordinates.

Montgomery coordinates represent the point in $[X : Z]$ coordinates. The $[X : Z]$ refers to the X and Z coordinates of the point. In order to convert a point in the (x, y) coordinate system to Montgomery coordinates, we take $X = x$, $Z = 1$ and represent the point as $[X : 1]$, notice that the y coordinate is discarded. One can recover the x-coordinate of the point when $Z \neq 0$ as $x = X/Z$. The point at infinity ($\mathcal{O}$) is defined by the pair $[0 : 0]$. The Montgomery coordinates are discussed in detail in [7].

Suppose we have two points, P_1 and P_2, such that P_1 and P_2 are not $\mathcal{O}$, and $P_1 \neq P_2$. If $P_1 = [X_1, Z_1]$ and $P_2 = [X_2, Z_2]$, we have $P_1 + P_2 = [X_+, Z_+]$ and $P_1 - P_2 = [X_-, Z_-]$. Then the addition function is defined as:

$$\begin{aligned} X_+ &= Z_-((X_1X_2 - AZ_1Z_2)^2 - 4B(X_1Z_2 + X_2Z_1)Z_1Z_2) \\ Z_+ &= X_-(X_1Z_2 - X_2Z_1)^2 \end{aligned}$$

We denote this addition function by *addh*() and represent it as:

$$[X_+ : Z_+] = addh([X_1 : Z_1], [X_2 : Z_2], [X_- : Z_-])$$

Similiarly, we denote the double function as *doubleh*() and represent it as,

$$[X_+ : Z_+] = doubleh([X_1 : Z_1])$$

where,

$$\begin{aligned} X_+ &= Z_-((X_1^2 - AZ_1^2)^2 - 4B(2X_1)Z_1^3) \\ Z_+ &= 4Z_-(X_1^3 + AX_1Z_1^2 + BZ_1^3) \end{aligned}$$

Thus, the use of Montgomery coordinates allows us to avoid the use of modular inverses. However, after finding the point multiples, we do require the use of modular inverses to calculate $x = XZ^{-1}$ to create ordered lists. It should be noted that several operations in *addh*() and *doubleh*() functions are repeated and should be stored in order to optimize the functions.

2. **Reducing point multiplication**
Point multiplication, i.e. nP in Montgomery coordinates, can be performed by employing the following multiplication ladder:

Algorithm: Elliptic Curve Multiplication Ladder.

Input: n, P
Output: nP

1. [Initialize]
   if ($n == 0$) return O.
   if ($n == 1$) return $[X : Z]$
   if ($n == 2$) return $doubleh([X : Z])$

2. [Begin Montgomery adding/doubling ladder]
   $[U : V] = [X : Z]$
   $[T : W] = doubleh([X : Z])$

3. [Loop over bits of n, starting with next-to-highest]
   for ($B - 2 \geq j \geq 1$) {
      if ($n_j == 1$) {
         $[U : V] = addh([T : W], [U : V], [X : Z])$;
         $[T : W] = doubleh([T : W])$;
      } else {
         $[T : W] = addh([U : V], [T : W], [X : Z])$;
         $[U : V] = doubleh([U : V])$;
      }
   }

4. [Final Calculation]
   if ($n_0 == 1$) return $addh([U : V], [T : W], [X : Z])$;
   return $doubleh([U : V])$;

Pseudocode taken from [7].

The multiplication ladder calls the *addh*() as well as *doubleh*() function multiple times in order to calculate a single point multiple nP. If we take a look at the list generation function we see that in list A every point is just an increment of P over the previous point. In list B every point is an increment of WP over the previous point. We just need to call the multiplication function to find only the first two points in each list and we can find the successive points by adding P for list A and WP for list B to the previous point. Thus, we can effectively reduce the amount of calls to the multiplication ladder function to four. This provides an enormous improvement to the algorithm. This method is not limited to Montgomery coordinates and should effectively work in all the coordinate systems.

3. **Computing two modular inverses simultaneously**
   This method is based on the observation that if we take $\frac{1}{xy}$ then $y^{-1} = x \times \frac{1}{xy}$ and $x^{-1} = y \times \frac{1}{xy}$. As can be seen, if we use the above method, two modular inverses have been reduced to one inverse and three multiplication functions [6]. We have already seen the performance of the modular inverse function in graph 2.2.

   The following graph shows the statistics related to the $MulMod()$ function provided by the NTL library:

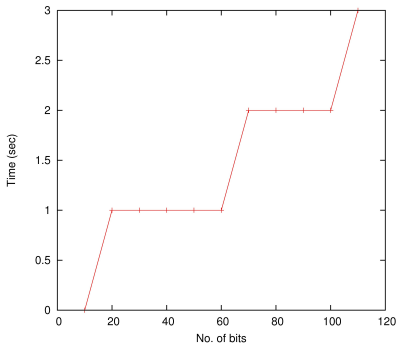

Figure 2.3: Performance of the Modular Multiply operation in the NTL Library.

   The comparison of statistics reveals that the modular multiplication is many times faster than the modular inverse function. Replacing one inverse operation with three multiply operations gives a massive performance boost. We can also try the same trick by calculating n multiple points and simultaneously calculate their inverses, thus reducing the number of inverses by n. It would be interesting to see at what point the performance due to this trick starts to deteriorate.

4. **Eliminating inverses completely**
   We could theoretically remove the modular inverses completely by generating lists by storing both the X and the Y value. Thus if we wished to compare two points P1$[X_1 : Z_1]$ and P2$[X_2 : Z_2]$, we would need to check for $X_1Z_2 = X_2Z_1$.

   This method, in effect, would remove the need for the modular inverse function completely but then we would lose the benefit of having ordered lists. This would effectively make searching through huge lists an extremely time consuming procedure. However, if one had an efficient search function in place, this method is worth trying out.

2.3.1.2 Reduction in memory usage

1. **Selection of data structures**
   Shanks-Mestre algorithm has the same space complexity as time complexity, namely of the order of $O(p^{\frac{1}{4}})$. This implies that it is a fine trade off between the memory usage and the computation time. The two lists generated by the Baby-Step Giant-Step algorithm are huge and require a lot of system memory to store

simultaneously. There is not much that can be done to improve this situation except to employ the best data structure to store these lists or store the data in files, which is not without its own problems as explained in section (2.3.2).

While writing the program various data structures were experimented with. The program was tested with the C++ STL containers like Map and Sets, which store their elements in a sorted manner and employ binary tree structures in order to do so. The program was also tested with STL Vectors, which work like arrays, and need to be sorted after they have been populated, in order to generate ordered lists.

It was expected that storing all the elements in a Vector and then sorting the Vector would perform better than associative containers like Maps or Sets, which require comparing elements at every addition and need to balance the binary tree structure as well. However, it was surprisingly found that Sets provide the best data performance to the algorithm.

In the case of Vectors, the speed of addition of elements to the lists was nullified by the amount of time it took to sort these massive lists. This again is a programming specific issue and if one were to come up with an extremely useful sorting algorithm or some sort of hashing algorithm then Vectors would definitely be worth experimenting with.

2. **Storing only one list**
   This particular optimization resulted in massive performance gains. Since the first list was already generated, the second list was never created. Every time an element was created it was compared with the elements in the first list using binary search and, if there was no match, it was discarded. If there was a match a counter was incremented, the index was stored and the point itself was discarded. This resulted in the memory consumption being reduced by half and it also saved on the time consumed to insert the element into a data structure.

2.3.1.3 Number Theoretic improvements

Currently, if the algorithm comes across a point P with a low order, thus resulting in multiple matches, the algorithm discards the point and starts all over again.

Coming across multiple matches just means that we have hit upon either the point order or a multiple of point order P. Once we have found this point order m, or its multiple, we just need to find its prime factors and keep eliminating each prime, until we find the correct point order. After finding this point order we find all its multiples that lie inside the Hasse's bound. One of these multiples has to be the curve order. A property of curve orders is that any point multiplied by the curve order yield the point at infinity O. Now to find the correct point order, we just need to find a couple of random points and multiply them by the multiples of the found point order to see which one yields O for all the points. This multiple is the curve order.

This method is described in detail in [25]. This method has not been tested during the course of investigation of this thesis.

2.3.2 Memory constraints

As has been mentioned previously, Shanks-Mestre algorithm is a fine balance between memory constraints and speed. If we try and optimize one, the other one suffers. To increase the size of the prime field for which we wish to find the order of the curve, if we write out the generated lists out to a file instead of storing them in the RAM, we free up the RAM for more calculations to be done.

A program was written to write out the generated lists to files and then these files were sorted using the Unix *sort* command. These files were then searched for a match using a program written in Perl. This resulted in massive I/O operations to the hard disk, thus slowing down the performance of the program. However, it was able to calculate the order of the curves with characteristics ranging up to 100 bits. The performance improvements due to this method can be seen in table 2.4.

2.3.3 Performance analysis of Shanks-Mestre algorithm

Shanks-Mestre requires about $O(p^{\frac{1}{4}})$ operations to generate the Baby-Step Giant-Step parameters. The searching of the lists takes another $O(p^{\frac{1}{4}} \ln p)$ operations, thus amounting to a total complexity of $O(p^{\frac{1}{4}} \ln p)$. This is certainly much better than the $O(p)$ complexity of the naïve algorithm.

Apart from the time complexity there is also the space complexity of storing two very large lists that is associated with Shanks-Mestre. The space complexity of the algorithm is again $O(p^{\frac{1}{4}})$. In the tests performed it was noted that it was impossible to continue with the tests after about 2^{94} bit primes, unless we write out the generated lists to a file and then use external methods to sort them and compare them as explained in section (2.3.2) and (2.4).

2.3.4 Statistics and results

In order to test the performance of the Shanks-Mestre algorithm, the algorithm was implemented in C++ and NTL library [21] was used for it's optimized implementation of various number theory functions. The standard C++ STL containers were used to create the list of generated points. The optimizations provided by the GNU C++ compiler were used to optimize the programs. The performance tests were done on an Intel P4, dual-core machine with 1 GB of RAM. The primes used in the tests were generated using the prime generation function provided by NTL library, which generates primes of required bit lengths.

The statistics of the Shanks-Mestre algorithm are broken into two parts. The first one showing the algorithm without any of the improvements, employing plain affine coordinates and an NAF multiplication ladder, explained in detail in [24]. The second part consisting of the algorithm written using the improvements mentioned above and using Montgomery coordinates and Montgomery multiplication ladder.

The figure and the statistics shown in figure 2.4 and table 2.2 are for the algorithm written in C++ without any improvements. The graph shows the performance of the Shanks-Mestre algorithm without the improvements, with respect to the number of

bits in the characteristic prime. The table shows the timing statistics along with the characteristic prime and the output of the algorithm.

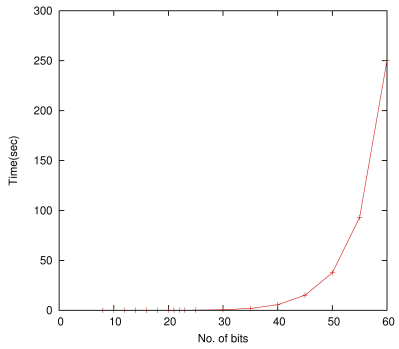

Figure 2.4: Performance of Shanks-Mestre without improvements.

Shanks-Mestre algorithm			
Bit-size (lg p)	Prime (p)	Time (sec)	Number of points
8	241	0.046	223
10	659	0.046	653
12	2707	0.046	2644
14	10009	0.077	10089
16	35051	0.077	35187
18	252937	0.093	252765
20	529043	0.109	527878
21	1622669	0.171	1625052
22	2317919	0.187	2315065
23	7029599	0.187	7028642
25	21192287	0.265	21198894
30	621863321	0.624	621867781
35	18476397883	1.874	18476139303
40	887824074947	5.656	887824401408
45	27137934029177	15.093	27137926160511
50	713233189761401	37.640	713233192312264
55	20335297264092913	93.076	20335297309264699
60	662867092149154843	250.218	662867092559546848

Table 2.2: Performance of Shanks-Mestre algorithm without improvements.

The statistics 2.3 and the graph 2.5 below show the performance of the Shanks-Mestre algorithm with all the improvements, employing the Montgomery coordinates and the Montgomery multiplication ladder. The graph shows the performance of the algorithm with respect to the increasing number of bits in the characteristic prime. The table 2.3 lists the time taken and the output for a particular prime. Note that this is still far from the sizes required for cryptographic uses.

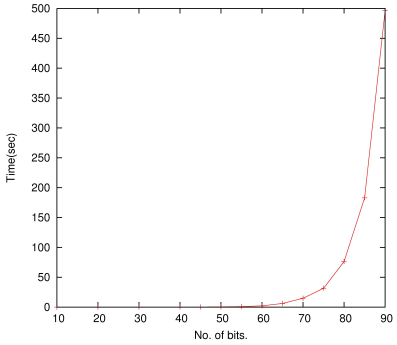

Figure 2.5: Performance of Shanks-Mestre algorithm with improvements.

Bit-size (lg p)	Prime (p)	Time (sec)	Number of points
10	659	0.046	653
20	529043	0.046	527878
30	621863321	0.046	621867781
40	887824074947	0.250	887824401408
45	27137934029177	0.328	27137926160511
50	713233189761401	0.516	713233192312264
55	20335297264092913	0.984	20335297309264699
60	662867092149154843	2.172	662867092559546848
65	31589708485749285569	6.219	31589708479346282562
70	997463535420662492537	14.875	997463535449084778301
75	20451098129904754707631	31.640	20451098130015922912267
80	63251236420635436737845 3	76.391	63251236420636172483864 0
85	21017378458678237957618993	183.031	21017378458673457484584654
90	83350151264634661024873 0337	496.203	83350151264632108751569 1600

Table 2.3: Performance of Shanks-Mestre algorithm with improvements.

As is expected, the performance of the algorithm increases dramatically with all the previously suggested improvements put in place. However, the performance is also limited by the memory usage, which also increases dramatically with the increase in size of the prime. This inadvertently limits the performance of the algorithm, until the CPU starts flushing out the contents of the RAM to the page file to free up more RAM.

The third set of statistics given in 2.6 and 2.4 are for a version of the algorithm that utilizes the benefits of Montgomery coordinates but copies the lists out to a file and uses external programs to sort the list and find a match. This program does have performance drawbacks but does not suffer from extreme memory usage. As long as time is not an issue this program can be used to perform calculations on enormous primes. In our tests the program was made to count points on a curve that was about 96-bits long and it took approximately 2.5 hours to calculate the order of the curve.

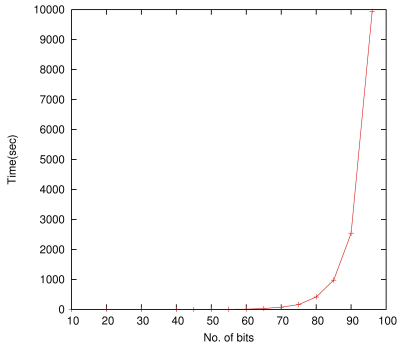

Figure 2.6: Performance of Shanks-Mestre algorithm with memory improvements.

On comparison, it is seen that the idea of writing the lists out to a file does degrade the performance of the algorithm in general but is still faster than the plain algorithm employing only the affine coordinates. The main improvement that this enhancement provides is not in the speed but in the number of bits of prime field that the algorithm can work with. Curve orders for fields defined under 96 bit primes were successfully calculated using this algorithm in just over 2 hours. The lists for theses curves were written out to a file which amounted to about 2GB of data. Were these lists to be stored in the RAM, the calculations would either have been impossible or the size of the RAM would have to be increased. After writing out the files at short intervals the RAM usage for the program was reduced to about 250MB at any given point in time. Although the program was only checked for 96 bit primes, given enough time it should work even for a large prime. The process can also be parallelized for further improvements.

The graph 2.7 shows a composite of the graphs with and without the various improvements for comparison.

Bit-size (lg p)	Prime (p)	Time (sec)	Number of points
10	659	0.359	653
20	529043	0.359	527878
30	621863321	0.563	621867781
40	887824074947	0.781	887824401408
45	27137934029177	1.171	27137926160511
50	713233189761401	2.203	713233192312264
55	20335297264092913	4.812	20335297309264699
60	662867092149154843	11.203	662867092559546848
65	31589708485749285569	31.375	31589708479346282562
70	997463535420662492537	75.016	997463535449084778301
75	20451098129904754707631	163.359	20451098130015922912267
80	632512364206354367 378453	415.516	632512364206361724 838640
85	2101737845867823795 7618993	979.172	21017378458673457 484584654
90	833501512646346610 248730337	2543.453	833501512646321087 515691600
96	7269182735545241350 3006695883	9938.226	7269182735545256703 8667788774

Table 2.4: Performance of Shanks-Mestre algorithm with memory improvements.

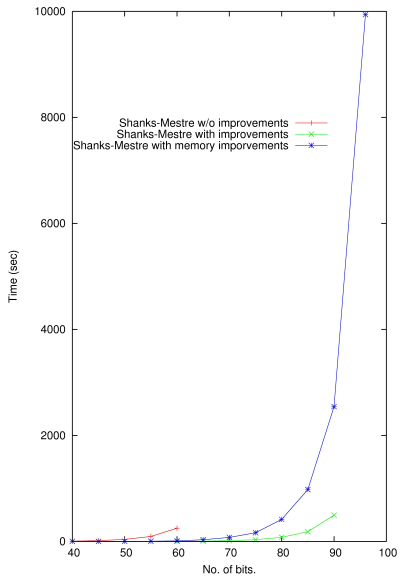

Figure 2.7: Comparison graphs for Shanks-Mestre.

Chapter 3

Advanced Point Counting Algorithms

3.1 Overview

Until now we have been studying point counting algorithms with complexities of $O(p)$ and $O\left(p^{\frac{1}{4}} \log p\right)$. These are theoretically sound algorithms but not very useful for large values of p like 2^{160} as needed in cryptographic applications. In this chapter we will be looking at a polynomial-time point counting algorithm due to Schoof [18], which has a complexity of $O\left(\ln^8 p\right)$ and Schoof-Elkies-Atkin (SEA) algorithm which is an improvement to Schoof's original algorithm by Elkies [10] and Atkin [18], having a complexity of $O\left(\ln^5 p\right)$. These algorithms are extremely dense and borrow heavily from the fields of geometry, linear algebra and abstract algebra.

3.2 Schoof's Algorithm

Before we begin with the explanation of Schoof's algorithm it would be best to review the various theorems that the algorithm utilizes.

3.2.1 Background to Schoof's algorithm

3.2.1.1 Frobenius Endomorphism [25]

Let $\mathbb{F}_q$ be a finite field with algebraic closure $\overline{\mathbb{F}}_q$, where $q = p^k$, then we define the Frobenius map Φ_q as follows:

$$\Phi_q : \overline{\mathbb{F}}_q \rightarrow \overline{\mathbb{F}}_q \tag{3.1}$$

$$x \mapsto x^q \tag{3.2}$$

What this means is that when we consider points on the curve where the points are on the algebraic closure $\overline{\mathbb{F}}_q$ of the field $\mathbb{F}_q$, raising the point to the p-th power is a field

automorphism. In even simpler terms, raising the coordinates of a point $(x, y) \in E(\overline{\mathbb{F}}_q)$ takes this point to another point in $E(\overline{\mathbb{F}}_q)$ [25].

Lemma 3.2.1 *Let E be defined over $\mathbb{F}_q$, where $q = p^k$, and let $(x, y) \in E(\overline{\mathbb{F}}_q)$. Then,*

1. $\Phi_q(x, y) \in E(\overline{\mathbb{F}}_q)$
2. $(x, y) \in E(\mathbb{F}_q) \iff \Phi_q(x, y) = (x, y)$

PROOF In order to prove the above we need to be aware of the fact that $(a + b)^q = a^q + b^q$ where q is a power of the characteristic of the field. We also need that $a^q = a$ for all $a \in \mathbb{F}_q$. Both are standard theorems in field theory [25].

By the Weierstrass equation we have,

$$y^2 = x^3 + ax + b \tag{3.3}$$

with $a, b \in \mathbb{F}_q$. Raise the equation to the qth power.

$$(y^2)^q = (x^3 + ax + b)^q \tag{3.4}$$
$$(y^q)^2 = (x^q)^3 + a^q x^q + b^q \ \because\ (a + b)^q = a^q + b^q \tag{3.5}$$
$$(y^q)^2 = (x^q)^3 + ax^q + b \ \because\ a, b \in \mathbb{F}_q \tag{3.6}$$

This shows that (x^q, y^q) lies on E thus proving 1.

The 2^{nd} point can be inferred from the fact that if $x, y \in \mathbb{F}_q$ then $x^q = x$ and $y^q = y$.

3.2.1.2 Division Polynomials

If we recall the section on elliptic curve arithmetic 1.1.2 it is clear that the coordinates of the sum $P_1 + P_2$ of two points on the curve are rational functions of the coordinates of the points P_1 and P_2. With repeated application of these formulae we can deduce that the multiplication by n of a particular point P can be expressed in terms of a rational function. This rational function is known as the division polynomial. We shall be discussing some important properties of division polynomials which are relative to our discussion at hand. For an in-depth coverage of division polynomials refer to [15] or [3].

The nth-division polynomials are polynomials whose value is zero for the points having order n on an elliptic curve (where the curve is defined by the Weierstrass equation). Consider $E[n]$ to be a set of points whose order is equal to or divides n defined over an elliptic curve, $E[n] = \left\{P \in E(\overline{\mathbb{F}}_p) \mid nP = O\right\}$. Plugging the x and y coordinates of any of these points into the division polynomial Ψ_n makes the value of Ψ_n equal to zero.

Continuing with this definition, the division polynomials Ψ_n of an elliptic curve are elements of $\mathbb{F}_p(x, y)$, modulo the curve, with the property that $\Psi_n(x, y) = 0$ if and only if $(x, y) \in E[n]$. These polynomials are defined as follows [7]:

$$\begin{aligned}
\Psi_{-1} &= -1, \\
\Psi_0 &= 0, \\
\Psi_1 &= 1, \\
\Psi_2 &= 2Y, \\
\Psi_3 &= 3X^4 + 6aX^2 + 12bX - a^2, \\
\Psi_4 &= 4Y(X^6 + 5aX^4 + 20bX^3 - 5a^2X^2 - 4abX - 8b^2 - a^3), \\
\Psi_{2n} &= \Psi_n(\Psi_{n+2}\Psi_{n-1}^2\Psi_{n-2}\Psi_{n+1}^2)/(2Y), \\
\Psi_{2n+1} &= \Psi_{n+2}\Psi_n^3 - \Psi_{n+1}^3\Psi_{n-1}.
\end{aligned}$$

Algebraic verification of division polynomials:
In order to see why the division polynomial properties hold, let us try to prove that they hold for Ψ_3. For a complete proof please refer [3]. Consider a point $P(x, y) \in E[3]$. This implies that $3P = O$. As a consequence $2P = -P$, which means that the x-coordinate of $2P$ and $-P$ are the same. Thus, we can use the formulas given in section 3.2.1.2 to calculate the x-coordinate of $2P$ as follows [15]:

$$x_3 = \lambda^2 - x_1 - x_2 \tag{3.7}$$

but since $x_1 = x_2$ we get,

$$x_3 = \lambda^2 - 2x_1 \tag{3.8}$$

Since we are finding out the value of $2P$ we can use the formula for λ given in (1.1.2) on page (3) for the case when $x_2 = x_1$.

$$x_3 = \frac{(3x_1^2 + a)^2}{4y_1^2} - 2x_1 \tag{3.9}$$

Since we know that the x-coordinates of $2P$ and $-P$ are the same we can equate the right hand side of the above equation to x_1, as follows:

$$x_1 = \frac{(3x_1^2 + a)^2}{4y_1^2} - 2x_1 \tag{3.10}$$

Simplifying this equation gives:

$$12x_1y_1^2 = 9x_1^4 + 6ax_1^2 + a^2 \tag{3.11}$$

Substituting $y^2 = x^3 + ax + b$ from the Weierstrass equation,

$$12(x_1^4 + ax_1^2 + bx_1) = 9x_1^4 + 6ax_1^2 + a^2 \tag{3.12}$$

Further simplifying and substituting x_1 with x gives:

$$3x^4 + 6ax^2 + 12bx - a^2 = 0 \tag{3.13}$$

which is the equation for Ψ_3, meaning that $2P = -P$, if $\Psi_3 = O$. In other words, P has order 3, or $3P = O$. We could similarly prove the same for Ψ_4, or with much more work, for any other higher case.

Example of empirical verification of division polynomials:
Consider an elliptic curve defined by the equation $y^2 = x^3 + x + 29 \pmod{71}$. One can check directly that the point (20,5) has order 3 on this elliptic curve. Substituting the value of the x and y-coordinates in the equation for Ψ_3 we get:

$$\begin{aligned} \Psi_3 &\equiv 3x^4 + 6ax^2 + 12bx - a^2 \pmod{71} \\ \Psi_3(20,5) &\equiv 3(20^4) + 6(20^2) + 336(20) - 1 \pmod{71} \\ &\equiv 489119 \pmod{71} \\ &\equiv 0 \pmod{71} \end{aligned}$$

Similiarly Ψ_6 for the above elliptic curve is given by the following equation:

$$\begin{aligned} \Psi_6 &\equiv (16xy + 39x^2y + x^3y + 6x^4y + 59x^5y + 40x^6y \\ &\quad +57x^7y + 10x^8y + 54x^9y + 40x^{10}y + 53x^{12}y + \\ &\quad 2x^{13}y + 2x^{14}y + 6x^{16}y) \pmod{71} \\ \Psi_6(20,5) &\equiv 19679090827898548919600 \pmod{71} \\ &\equiv 0 \pmod{71} \end{aligned}$$

The result for Ψ_3 proves that the property of the division polynomial holds and the result for Ψ_6 reaffirms the obvious that $E[3] \subset E[6]$, or to state it in general terms, $E[n] \subset E[l]$, so long as $n \mid l$.

Properties of division polynomials [7]:

1. The division polynomial $\Psi_n(X, Y)$ is a polynomial in X alone for n odd, and for n even it is Y times a polynomial in X.

2. For n odd and not a multiple of p, we have $deg(\Psi_n) = (n^2 - 1)/2$.

3. For n even and not a multiple of p, we have the degree of Ψ_n in the variable X is $(n^2 - 4)/2$.

4. If $(x, y) \in E(\overline{\mathbb{F}}_p) \setminus E[n]$, then

$$[n](x,y) = \left(x - \frac{\Psi_{n-1}\Psi_{n+1}}{\Psi_n^2}, \frac{\Psi_{n+2}\Psi_{n-1}^2 - \Psi_{n-2}\Psi_{n+1}^2}{4y\Psi_n^3} \right) \tag{3.14}$$

Detailed proofs of these properties can be found in [3].

3.2.2 Explanation of Schoof's algorithm

Schoof's algorithm breaks down the order $\#E$ for many small primes l such that $\prod l > 4\sqrt{p}$, and then reconstructs the order modulo p using the Chinese remainder theorem. For more details on Chinese remainder theorem, look at section (A.10).

Schoof's algorithm begins by first examining the special case of $\#E(mod\ 2)$. By the property of the group order, the group order is the least common multiple of the order of the elements of the group. Thus, for the group order to be even, at least one of the elements in the group has to have an order of 2. Since a point $P \neq O$ has $2P = O$ if and only if it is of the form $P = (x, 0)$, meaning that the equation $x^3 + ax + b$ has roots in $\mathbb{F}_p$. This can be confirmed by taking $\gcd(x^p - x, x^3 + ax + b)$.

For the remaining cases for which $l > 2$, Schoof's algorithm employs a generalized method. From the Frobenius endomorphism discussed in Lemma 3.2.1 we already know that for $(x, y) \in E(\overline{\mathbb{F}}_p)$, $\Phi(x, y) = (x^p, y^p)$. Another theorem regarding the Frobenius endomorphism states that: If the order of the elliptic curve group $E(\mathbb{F}_p)$ is $p + 1 - t$ then

$$\Phi^2(P) - [t]\Phi(p) + [p]P = O \tag{3.15}$$

for every point $P \in E(\overline{\mathbb{F}}_p)$ [7]. This implies that the Frobenius endomorphism satisfies a quadratic equation with the trace as t.

Now consider only those points P for which $[n]P = O$, n being any positive number. It is a subgroup of $E(\overline{\mathbb{F}}_p)$ and Φ maps $E[n]$ to itself, as previously discussed in (3.2.1.2. Thus we have:

$$\Phi^2(P) - [t \bmod n]\Phi(p) + [p \bmod n]P = O \tag{3.16}$$

The above equation forms the basis of Shoof's algorithm. Schoof's algorithm calculates all the values in the above equation (recall that $[n]P$ can be calculated using 3.14), except t. Then it uses trial and error until the exact value of t is found which satisfies the above equation. It does so by employing the division polynomials which simulate both elliptic multiplication and pick out the n-torsion points. [See [25] for definitions.]

The Schoof's algorithm searches for $t \pmod l$ by substituting $n = l$ in equation 3.16, for numerous small primes l. After finding sufficient values of t modulo small primes l, such that the multiple of various l is greater than $4\sqrt{p}$, the value of the actual t can be found using the Chinese remainder theorem (A.10).

Algorithm: Schoof's Algorithm
Let p be a prime such that $p > 3$. For a curve $E_{a,b}$ this algorithm returns the value of $t(\text{mod } l)$, where l is in a set of much smaller primes than p and the curve order is $\#E = p + 1 - t$. For more details refer to [7] and [3].

Input: a, b, p
Output: $\#E_{a,b}$

1. [For loop]
   For each l in the set of small primes, repeat steps 2 and 3.

2. [Check $l = 2$]
   if ($l == 2$) {
   $g(X) = \gcd(X^p - X, X^3 + aX + b)$; //Polynomial gcd in $F_p[X]$.
   if ($g(X) == 1$) return 0; //$T \equiv 0 \pmod 2$, so order $\#E$ is even.
   return 1; $\#E$ is odd.
   }

3. [Analyze relation (7.10)]
   $\overline{p} = p \bmod l$;
   $u(X) = X^p \bmod (\Psi_{l,p})$ //$\Psi_{l,p} = \Psi_l \pmod p$
   //That is, $v(X) = Y^{p-1} \bmod (\Psi_{l,p})$.
   //$P_0 = (X^p, Y^p)$.
   $P_0 = (u(X), Yv(X))$
   $P_1 = (u(X)^p \bmod (\Psi_l, p), Yv(X)^{p+1} \bmod (\Psi_l, p))$
   //$P_0 = (X^{p^2}, Y^{p^2})$.
   Cast $P_2 = [\overline{p}](X, Y)$ in rational form.
   if ($P_1 + P_2 == O$) return 0; //$\#E = p + 1 - t$ *with* $t \equiv 0 \pmod l$
   $P_3 = P_0$
   for ($1 \le k \le l/2$){
   if (X-coordinates of ($P_1 + P_2$) and P_3 match) {
   if (Y-coordinates also match) return k;
   return $l - k$;
   }
   $P_3 = P_3 + P_0$;
   }

4. Find out the value of t for all the l and find the exact value of t using the Chinese remainder theorem. The actual order can be calculated and confirmed with the help of Hasse's bound.

Pseudocode taken from [7].

In Schoof's algorithm, the Ψ_l are calculated directly from the definition of division polynomials.

3.2.3 Proposed improvements to Schoof's algorithm

3.2.3.1 Using Shank's BSGS algorithm

If the last part of the Shank's algorithm is analyzed, where the algorithm searches from 1 to $l/2$ in order to find a match such that $P_1 + P_2 = kP_3$, we notice that for large values of l this becomes impossible, since we would have to calculate $l/2$ steps. It should also be noted how similar this is to Shank's original Baby-Step Giant-Step algorithm which was discussed in the Shanks-Mestre algorithm.

The problem consists of finding a match such that $P_1 + P_2 = kP_3$, where k ranges from 1 to $l/2$; we are in effect looking for a solution to the equation given in (3.16). The complexity of this search is of course $O(l/2)$. By employing Shank's algorithm to this problem we can get this down to $O((l/2)^{1/2+\epsilon})$.

In order to use Shank's algorithm, write $k = \beta + \gamma W$, where $W = \lceil \sqrt{l/2} \rceil$, now we can rewrite the equation as,

$$P_1 + P_2 = kP_3 \quad \Leftrightarrow \quad P_1 + P_2 = [\beta + \gamma W]P_3 \tag{3.17}$$

$$\Leftrightarrow \quad P_1 + P_2 + \beta P_3 = \gamma W P_3 \tag{3.18}$$

This is the form of Shank's Baby-Steps Giant-Steps algorithm that we are used to. Now we just have to construct two lists, namely,

$$\begin{aligned} A &= \{(P_1 + P_2 + \beta P_3) : \beta \in [0, W-1]\} \textit{ and} \\ B &= \{(\gamma W P_3) : \gamma \in [0, W]\}. \end{aligned}$$

After constructing these two lists we search for a match, and on finding the match, check for which one of $\beta \pm \gamma W$ matches our search criteria. This time around though we do not have to search for a unique match.

We can rewrite the part in the above algorithm from the for loop in step 3 onwards as follows.

1. Construct list A= $\{(P_1 + P_2 + \beta P_3) : \beta \in [0, W-1]\}$
2. Construct list B= $\{(\gamma W P_3) : \gamma \in [0, W]\}$
3. $s = A \cap B$
4. $\beta = ind(A, s)$ and $\gamma = ind(B, s)$
5. Choose $k = \beta \pm \gamma W$ such that $P_1 + P_2 + \beta P_3 == \gamma W P_3$
6. if (X-coordinates of $(P_1 + P_2)$ and P_3 match) {
   if (Y-coordinates also match) return k;
   return $l - k$;
   }

This method will allow us to use slightly bigger prime l, which should reduce the number of primes needed to find the solution. This will in turn help to increase the speed of the algorithm.

3.2.3.2 Eliminating polynomial inverses.

Due to the use of division polynomials, the points during the calculations happen to be polynomial rationals in the form of $P = (A/B, C/D)$, where $A, B, CandD$ are polynomials defining Ψ_l. This would require us to calculate polynomial modular inverses, and as mentioned before, calculating inverses is a tedious and time consuming process. To get around this, we do not compute the polynomial inverses and store the points in the rational form.

When we need to find matches between two points, namely $P_1 = (A/B, C/D)$ and $P_2 = (E/F, G/H)$, we can instead search to see if $AF = EB$ and $CH = GD$. This would definitely cut down on the time required to do the calculation. More details regarding this approach can be found in [7].

3.2.4 Performance analysis of Schoof's algorithm

Schoof's algorithm has a complexity of $O(log^8 p)$. This is highly reduced from the complexity of Shanks-Mestre algorithm but is still not spectacularly fast. One of the major drawbacks with the algorithm is the fact that the degree of division polynomials grows quickly and it becomes computationally intensive to work with polynomials of such a high degree, which brings down the performance of Schoof's algorithm.

3.2.5 Statistics and results

The implementation of Schoof's tested with this algorithm is due to Mike Scott. The algorithm is implemented in C++ with the MIRACL library [19]. The primes were generated with the help of the prime generation function provided by NTL library [21]. The following graph 3.1 show the performance of Schoof's algorithm with respect to the growing number of bits in the characteristic field. The table 3.1 shows the actual results obtained with the help of the program.

As can be seen from the table and the statistics, the algorithm can perform reasonably well for large primes. In fact it can calculate the curve order for fields defined under 190-bit primes in under 6 minutes.

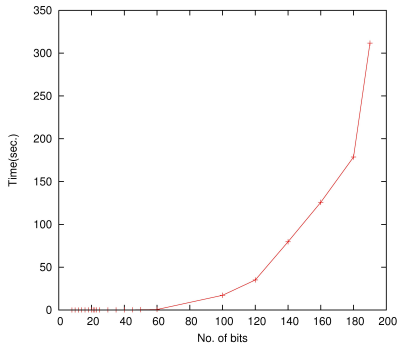

Figure 3.1: Performance of Schoof's algorithm with respect to lg p.

Schoof's algorithm			
Bit-size (lg p)	Prime (p)	Time (sec)	Number of points
8	241	0.046	223
10	659	0.046	653
12	2707	0.046	2644
14	10009	0.047	10089
16	35051	0.047	35187
18	252937	0.047	252765
20	529043	0.047	527878
21	1622669	0.047	1625052
22	2317919	0.047	2315065
23	7029599	0.047	7028642
25	21192287	0.047	21198894
30	621863321	0.047	621867781
35	18476397883	0.109	18476139303
40	887824074947	0.078	887824401408
45	27137934029177	0.141	27137926160511
50	713233189761401	0.343	713233192312264
60	662867092149154843	0.516	662867092559546848
100	633825300114114 700748351602943	17.125	633825300114113 716721032026723
120	664613997892457936 451903530140172297	35.078	664613997892457937 691477220447505570
140	696898287454081973 172991196020261297 062043	79.672	696898287454081973 171943196159173626 461648
160	730750818665451459 101842416358141509 827966271787	125.766	730750818665451459 101844067027284258 604550307509
180	766247770432944429 179173513575154591 809369561091801397	178.632	766247770432944429 179173512967424312 791504461706278412
190	78463771692333509 54794736779009583 02012794430558004 314147	311.750	78463771692333509 54794736778893965 13561226976274126 625920

Table 3.1: Performance statistics of Schoof's algorithm

3.3 Schoof-Elkies-Atkins (SEA) algorithm

3.3.1 Introduction to SEA algorithm

We saw in the previous section that one of the main limiting factor in the Schoof's algorithms was the degree of the division polynomials employed. René Schoof published the paper describing the algorithm [18] in 1985 and since then there have been some major improvements. The algorithm as it stands now, with a couple of improvements from Elkies in 1987 [10] and Atkin in 1988 [18], have reduced the complexity of the algorithm from $O\left(\ln^8 \ p\right)$ to $O\left(\ln^5 \ p\right)$. The algorithm in its present form has been dubbed the Schoof-Elkies-Atkins (SEA) algorithm. With the help of this algorithm, order of curves over $\mathbb{F}_p$ with p having 500 digits have been calculated.

Since the basic limitation of Schoof's algorithm was the degree of the division polynomials, it is not surprising that the improvements made by Elkies and Atkins to the algorithm deal with reducing the degree of the division polynomials employed.

Recall that in Schoof's algorithm we select a set of l-primes such that $\prod l > 4\sqrt{p}$. The first step in the SEA algorithm is to identify whether the prime l is an Elkies prime or an Atkin prime, the details of which are given a bit further. From our previous discussion (3.16) we already know that the Frobenius map Φ modulo l satisfies a quadratic characteristic equation of the form:

$$u^2 + t_l u + \overline{p} = 0 \tag{3.19}$$

If the discriminant of the above equation $\Delta = t^2 - 4\overline{p}$ is a quadratic residue in $\mathbb{F}_l$ we say that l is an Elkies prime, otherwise it is considered an Atkin prime. Although t is unknown, we can determine whether l is an Elkies prime by taking $\gcd(x^p - x, \Phi_l(x, j))$, where j is the j-invariant of the curve E, defined as

$$j(E) = 1728\frac{4A^3}{4A^3 + 27B^2}4 \tag{3.20}$$

If the degree of $\gcd(x^p - x, \Phi_l(x, j))$ is either 1, 2 or $l + 1$ then the prime is an Elkies prime and if it is 0 then it is an Atkin prime.

3.3.1.1 Elkies primes

Once it has been determined that the prime is an Elkies prime it can be established that the discriminant Δ_t has two roots, say λ and μ [3], which are eigenvalues of the Frobenius map Φ modulo l. Thus the equation (3.19) reduces to the following:

$$u^2 + t_l u + \overline{p} \quad = \quad (u - \lambda)(u - \mu) \tag{3.21}$$

$$\tag{3.22}$$

This in turn implies that $\overline{p} = \lambda\mu$, thus:

$$t_l = \lambda + \frac{\overline{p}}{\lambda} \tag{3.23}$$

In order to find the value of t_l we need to find the value of λ, which is an eigenvalue of the Frobenius map. To find the value of λ we just have to go through all the values of $1 \ldots l-1$ which satisfy the following equation:

$$\Phi_l(x, y) = (x^p, y^p) = \lambda(x, y) \tag{3.24}$$

Note that in this method all the calculations are done modulo a factor f_l of the division polynomial Ψ_l, which has a degree of $(l-1)/2$ as compared to $(l^2-1)/2$ of the division polynomial [3]. This improves the performance on l which are Elkies prime.

3.3.1.2 Atkin primes

In 1988, Atkin found a way to use the non-Elkies primes efficiently. It has been found that almost half the primes are Elkies primes and thus the use of just Elkies primes in the algorithm does not increase the number of primes that much. However, with larger primes the degree of the polynomials used also increases. Thus the best compromise is to use Elkies primes along with a few Atkin primes.

If l is found to be an Atkin prime then the characteristic polynomial of Φ has two roots $\lambda, \mu \in \mathbb{F}_{l^2} - \mathbb{F}_l$. The element $\lambda/\mu = \gamma_r$ is an element of order r in $\mathbb{F}_{l^2}$. All elements of order r can be found by first finding a generator g for $\mathbb{F}^*_{l^2}$ and then computing $\gamma_r = g^{i(l^2-2)/r}$, where $i \in 1, \ldots, r-1$ is co-prime to r. After finding all the possible values of γ_r we can find the possible values of $t \pmod l$ using the following equations:

$$t = (\lambda + \mu) \pmod l \tag{3.25}$$

$$\overline{p} = \lambda\mu \pmod l \tag{3.26}$$

$$\gamma_r = \lambda/\mu. \tag{3.27}$$

With these things in mind we are now in a position to state the pseudocode for the SEA algorithm.

3.3.1.3 Schoof-Elkies-Atkin (SEA) algorithm pseudocode

Algorithm 3.3.1: SEA(a, b, p)

Input: a, b, p
Output: $\#E_{a,b}$
$M \leftarrow 1$
$l \leftarrow 2$
$A \leftarrow \{\}$ //Set of Atkin primes.
$B \leftarrow \{\}$ //Set of Elkies primes.
while $M < 4\sqrt{p}$
do
- Decide whether l is an Elkies Prime or an Atkin Prime
- **if** l is an Elkies prime
  - **then**
    - Determine the polynomial $F_l(x)$.
    - Find an eigenvalue, λ, modulo l.
    - $t \leftarrow \lambda + \overline{p}/\lambda \bmod l$.
    - $E \leftarrow E \cup \{(t, l)\}$.
  - **else**
    - Determine $t \bmod l$ with the method in (3.3.1.2)
    - $A \leftarrow A \cup \{(t, l)\}$.
- $M \leftarrow M \times l$.
- $l \leftarrow nextprime(l)$. //$nextprime(l)$ gives the next prime from l.

Recover t using the sets A and E and employing CRT.
return $(q + 1 - t)$

Pseudocode taken from [3].

3.3.2 Improvements to SEA algorithm

3.3.2.1 Working with mostly Elkies primes

It is generally noticed that working with Elkies primes is much quicker than working with Atkin primes. However, working with mostly Elkies primes requires us to find the trace of Φ modulo larger primes than if we were working with both the Elkies and Atkin primes.

A way around it is to mostly use Elkies primes and a handful of Atkin primes so as to maintain optimum performance [6].

3.3.3 Performance analysis of SEA algorithm

Schoof's algorithm had a complexity of $O(\log^8 p)$, mostly due to the fact that it required us to work with polynomials of high degree. The improvements provided by the use of Elkies and Atkin primes allow us to work with much smaller polynomials and as a result the complexity of the original Schoof's algorithm is reduced to about $O(\log^5 p)$ [3]. This can be further reduced by the use of quicker multiplication methods.

3.3.4 Statistics and results

The following graph and table give the performance and the results obtained for the implementation of SEA algorithm in C++ by Mike Scott [19]. The curve used was, as usual, $x^3 + x + 28$. From the graph and the table it is clear that the improvements by the use of Elkies and Atkin primes does increase the performance of the basic Schoof's algorithm. It was found that curve orders for curves defined over fields of characteristic 240 bits were found out in under five minutes. This is sufficient for most of today's cryptographic needs.

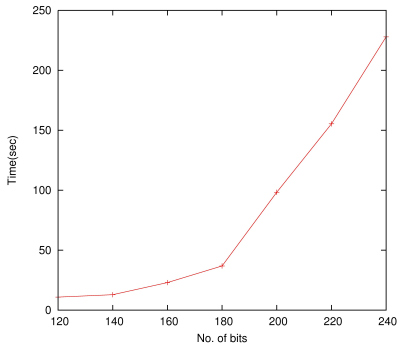

Figure 3.2: Performance of SEA algorithm.

Looking at the performance statistics of the SEA algorithm in table 3.2 we can see that the SEA algorithm is much more efficient than all the algorithms studied so far. The graph 3.3 shows the comparison of SEA algorithm to the previously studied algorithms and it is obvious that SEA algorithm is a vast improvement over the rest of them.

SEA algorithm			
Bit-size (lg p)	Prime (p)	Time (sec)	Number of points
120	664613997892457936 451903530140172297	10.862	664613997892457937 691477220447505570
140	696898287454081973 172991196020261297 062043	12.891	696898287454081973 171943196159173626 461648
160	730750818665451459 101842416358141509 827966271787	22.969	730750818665451459 101844067027284258 604550307509
180	766247770432944429 179173513575154591 809369561091801397	36.844	766247770432944429 179173512967424312 791504461706278412
200	803469022129495137 770981046170581301 261101496891396417 650789	98.265	803469022129495137 770981046170893117 655490093291821879 961375
220	8424983333484574935 8334422146936345855 1160763204392890034 487820473	155.344	8424983333484574935 8334422146936450313 2088854031797480115 479932309
240	8834235323891921647 9164875037145925791 3741948437809479060 803100646309917	227.922	8834235323891921647 9164875037145925918 2526634310061095285 277024846365455

Table 3.2: Performance statistics of SEA algorithm.

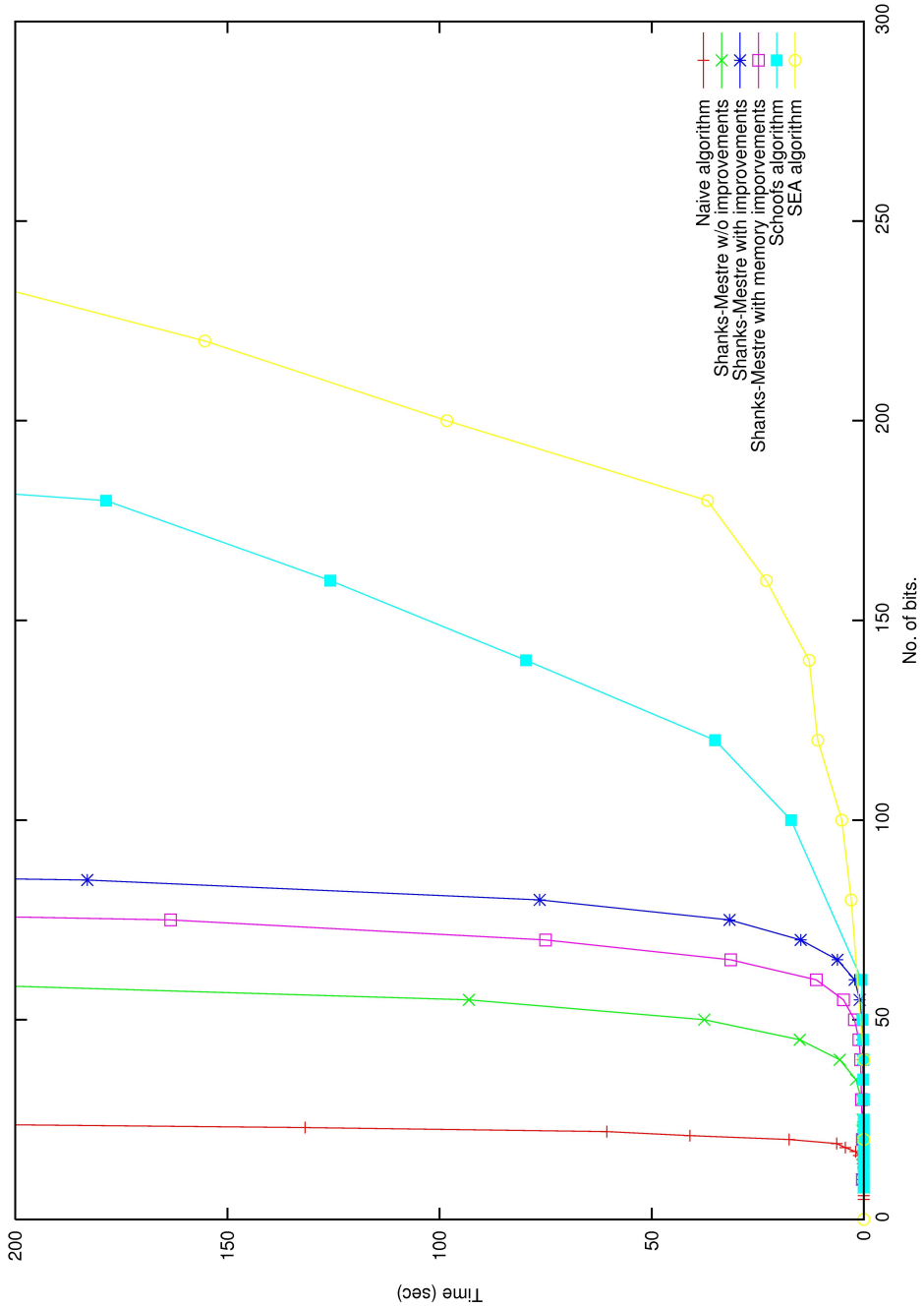

Figure 3.3: Comparison of various algorithms.

3.4 Special Cases

Most of the algorithms studied so far are generalized point counting algorithms. However, there are numerous special cases where finding the group order is trivial. These cases lead to an interesting study all by themselves. Two such important cases are briefly given in this section.

3.4.1 Subdfield Curves

Once we know the order of a small finite field $E(\mathbb{F}_q)$, it is trivial to find the curve order of $E(\mathbb{F}_{q^n})$ for all n [25].

Theorem 3.4.1 *Let $\#E(\mathbb{F}_q) = q+1-t$. Represent, $X^2 - tX + q = (X-\alpha)(X-\beta)$. Then,*

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

for all $n \geq 1$.

A detailed proof of theorem 3.4.1 is given in [25]. This special case gives rise to some interesting possibilities. We can easily find the order of $E(\mathbb{F}_q)$, where q is a small prime, using one of the above mentioned algorithms and then use theorem 3.4.1 to find the curve order for $E(\mathbb{F}_{q^n})$, where n is large.

This method is extensively used in elliptic curves over binary fields, where $q = 2$. The binary fields lend themselves to efficient hardware implementations.

3.4.2 Family of curves

This section presents a special family of curves for which there is an explicit formula for finding the curve order [25]. If the elliptic curve is defined by the equation

$$y^2 = x^3 - ax$$

and $a \not\equiv 0 \pmod{p}$, then the theorem 3.4.2 gives the explicit curve order.

Theorem 3.4.2 *Let p be an odd prime and the $a \not\equiv 0 \pmod{p}$. If the curve E is defined by*

$$y^2 = x^3 - ax$$

1. *If $p \equiv 3 \pmod{4}$, then $\#E(\mathbb{F}_p) = p + 1$.*
2. *If $p \equiv 1 \pmod{4}$, represent $p = a^2 + b^2$, where a, b are integers with b even and $a + b \equiv 1 \pmod{p}$. Then,*

$$\#E(\mathbb{F}_p) = \begin{cases} p+1-2a & \text{if } a \text{ is a fourth power mod } p \\ p+1+2a & \text{if } a \text{ is a square mod } p \text{ but not a 4th power mod } p \\ p+1 \pm 2b & \text{if } a \text{ is not a square mod } p \end{cases}$$

A detailed proof of the theorem can is given in [25].

Chapter 4

Curve Construction

4.1 Overview

To this point we have studied algorithms that tell us what the order of the curve is, given the curve parameters a, b and p. As we have seen, even the best of these algorithms has a time complexity of about $O(\log^5 p)$. Such algorithms are at best suited for constructing curves over fields defined over primes of up to 500 bits. This is more than enough for today's cryptographic usage. However, as time progresses and the computers become faster at crunching numbers, this requirement is bound to grow and in the future such algorithms may become too slow.

The current method of finding suitable curves itself is a bit redundant. We first select a large enough prime field and then arbitrarily select the parameters a and b. We count the number of points on the curve and then check to see if the curve order is suitable or not.

There is an alternative to this process. The algorithm to be discussed next allows us to enter a prime of a particular length, but not a and b, and then lets us know the order of the curves that can be constructed with the help of the given prime. Optionally it can also return the values of a and b for the curve. Thus, we know beforehand, the order of the curves before we even know the parameters a and b. If we find a suitable curve order we can continue with the algorithm to find the a and b or discard the value and move on to the next curve order.

There have been concerns over such curves not being suitable for cryptographic usage and there have been talks about such curves being susceptible to some unknown future discrete log algorithm. To date there has been only very limited evidence supporting these claims [7].

The ideas of curve construction were formulated on the basis of work done by A.O.L. Atkins and F. Morain in the area of elliptic curve based primality testing algorithm [2]. The theory behind curve construction is closely related to the complex multiplication of elliptic curves over the fields of rational numbers. A detailed study of this method is beyond the scope of this thesis but an overview is presented. For further details refer to [7] and [3].

4.2 Atkin-Morain curve construction

4.2.0.1 Fundamental Discriminant, Hilbert Class Number and Hilbert Class Polynomial

A negative integer D is considered a fundamental discriminant if the odd part of D is square-free and $|D| \equiv 3, 4, 7, 8, 11, 15 \pmod{16}$. With every discriminant D there is an associated class number $h(D)$, known as the Hilbert class number, and a Hilbert class polynomial T_D. The Hilbert class polynomial has the property that it has degree equal to $h(D)$ and its discriminant is equal to D. Furthermore, its coefficients consist of integers.

4.3 Atkin-Morain curve generation algorithm

Let n to be a positive integer. The map $[n]$ that maps a point P on E to $[n]P$ belongs to endomorphisms of E, denoted by End(E). Thus, the ring End(E) contains an isomorphic copy of the ring of integers. However, sometimes there are endomorphisms of E that do not correspond to an integer. In such a case it is not isomorphic to a ring of integers but to an order in an imaginary quadratic number field. Such curves are said to have complex multiplications, or CM curves [7].

Consider an elliptic curve E defined over the rationals. If this curve, considered over the complex numbers, has complex multiplications in $\mathbb{Q}(\sqrt{D})$, where D is a negative integer, and if we take $p > 3$, a prime that does not divide the discriminant of E, then it is quite easy to find the order of $E_{a,b}(\mathbb{F}_p)$.

The process of generating the curves is simple. We select a prime number p for which we wish to construct a curve $E_{a,b}(\mathbb{F}_p)$. Next, we choose a negative discriminant less than -4 such that $(\frac{D}{p}) = 1$. This is to ensure that $\mathbb{Q}(\sqrt{D})$ exists for the chosen p. The discriminant is taken less than -4 because -3 and -4 are special cases that give j-invariant equal to 1728 and 0. Now we find if there are any solutions to the equation $4p = u^2 + |D|v^2$. This can be easily be determined with the help of Cornacchia-Smith algorithm, which is listed later in section 4.4.1. If such u, v do not exist then we select another D. On the other hand if such a pair exists for the Diophantine equation, the curve order is given by $p + 1 \pm u$. One of the curve orders belongs to the curve $E_{a,b}(\mathbb{F}_p)$ and another to its twist $E'_{a,b}(\mathbb{F}_p)$

At this point we can check to see if the curve orders are to our satisfaction and if they are not then we move on to another D. If the curve order is acceptable we proceed with the calculation of Hilbert polynomial (T_D) associated with the Hilbert class of D. We then find an integral root of T_D which gives us the j-invariant of the curve. On knowing this j-invariant we can find out the parameters a and b from the equations 1.12 and 1.13. The twist of the curve can be found out easily by first finding a g, which is a quadratic nonresidue modulo p, then $a' = g^2 a$ and $b' = g^3 b$.

Finding which curve order belongs to which curve is just a matter of generating a couple of random points on the curves and checking to see if their multiple with the order is equal to O or not.

This method is most efficient for generating curves defined over fields with very

large characteristics. The only time consuming part of the whole algorithm is the generation of the Hilbert polynomial T_D and finding the class numbers. Luckily, many discriminants have been already classified by their class numbers and T_D, for almost all of these discriminants, are also available. Thus, it just becomes a matter of creating a lookup table.

4.4 Curve finding algorithm

4.4.1 Cornachhia-Smith algorithm

The Cornacchia-Smith algorithm [7] is used to find a solution to the Diophantine equation $4p = u^2 + |D|v^2$. If a solution to the equation exists then the algorithm returns the solution, else it states that there are no solutions.

Algorithm 4.4.1 (Cornacchia-Smith)
Given a prime p and $D \equiv 0, 1 \pmod 4$ and $-4p < D < 0$, this algorithm returns whether the solution exists or not and if it does then the solution to the Diophantine equation $4p = u^2 + |D|v^2$.

Input: p
Output: u, v

1. [Case $p = 2$]
   if (p== 2){
      if ($D + 8$ is a square) return ($\sqrt{D+8}$);
      return{};
   }

2. [Test for solvability]
   if ($(\frac{D}{p}) < 1$) return {no solution};

3. [Initial square root]
   $x_0 = \sqrt{D} \pmod p$;
   if ($x_0 \not\equiv D \pmod 2$) $x_0 = p - x_0$;

4. [Initialize Euclid Chain]
   $(a, b) = (2p, x_0)$;
   $c = \lfloor 2\sqrt{p} \rfloor$;

5. [Euclid Chain to find the GCD]
   while ($b > a$) $(a, b) = (b, a \bmod b)$;

6. [Final Report]
   $t = 4p - b^2$;
   if ($t \not\equiv 0 \pmod{|D|}$) return {no solution};
   if ($t/|D|$ not a square) return;
   return $(\pm b, \pm\sqrt{t/|D|})$;

Pseudocode taken from [7].

4.4.2 Modified Atkin-Morain algorithm

As mentioned earlier, there is an exhaustive list [7] of a large number of discriminants along with their Hilbert polynomials T_D already available. In fact, a complete list for $h(D) \leq 16$ is available. Therefore, we can make lookup tables for these instead of calculating them when needed. The original Atkin-Morain algorithm [7] has been modified to employ these lookup tables rather than search for these polynomials, which can be time consuming.

Algorithm 4.4.2 (Modified Atkin Morain) *The algorithm takes in a prime number p and returns all the possible curve orders and parameters for suitable discriminants D in the lookup table.*

Input: p
Output: $E_{a,b}, \#E_{a,b}$

1. [Establish discriminant list and Hilbert polynomial list]
   $disc$ = {List of a few discriminants}
   pol = {List of respective Hilbert class polynomials.}

2. [Loop over representations]
   for $(D \in disc)$\{
   if $((\frac{D}{p} = 1)$\{
   Attempt to solve $4p = u^2 + |D|v^2$, via Cornacchia-Smith algorithm,
   on failure, jump to next $|D|$;
   \}
   else \{
   Jump to next $|D|$;
   \}
   Calculate a suitable quadratic nonresidue g of p;

3. [Finding the curve order]
   return $\{p + 1 \pm u\}$
4. [Finding the curve parameters]
   Select T_D from list pol respective to $|D|$.
   $S \equiv T \bmod p$;
   Obtain a root $j \in \mathbb{F}_p$ of S;
   $c = j(j - 1728)^{-1} \bmod p$;
   $r = -3c \bmod p$;
   $s = 2c \bmod p$;

5. [Return two curve-parameters]
   return $\{(a, b)\} = \{(r, s), (rg^2 \bmod p, sg^3 \bmod p)\}$
   \}

Pseudocode taken from [7].

This algorithm is quickly able to compute the elliptic curve parameters and their orders. The more exhaustive the lists, the more the number of elliptic curves the algorithm will generate. To make the processes quicker we can calculate the r and s parameters beforehand, and instead of making a lookup table for T_D, we can make a lookup table for r and s and do away with the need to find a root every time.

Since the algorithm returns two curves and two curve orders, it is also necessary to find out which order belongs to which curve. This is as easy as finding a couple of points on one of the curves and testing which curve order takes all the points to O. This then is the curve order for that particular curve and the other order belongs to the other curve.

4.5 Results

The following table shows a couple of curves generated by the algorithm 4.4.2 and their curve orders. The curves were generated for NIST prime $2^{192} - 2^{64} - 1$ (B) and their orders were verified with the help of an implementation of Schoof's algorithm written by Mike Scott [19]. The timing tests for this algorithm were not done since it takes a couple of seconds to come up with the results.

D	A	B	#E
-11	6277101735386680763 8357894232076664160 83908700149801463295 4159807182707468360 2710206888549004382 38894215393900114656	46584751438539776 42092629831012047 55260297955962511 49724124354093586 8347958	6277101735386680763 8357894233373564793 61799363334922742515 6277101735386680763 8357894230779763528 8357894230779763528
-43	6277101735386680763 8357894232076662099 24270530479027425279 1686211845626390982 8385606419617264718 48176813197085554621	36029148864059248 53907718144102400 0 20745705284379766 71395558450758628 03488418190394410 7829526	6277101735386680763 8357894233501076524 29941749766549548452 6277101735386680763 8357894230652251797 37875651014100374108
-67	6277101735386680763 8357892652453885226 54521600029365409279 1788017361257523964 5124277215235259826 43694628638449438414	76414926181008778 07226617388948774 73218730368000 36173442778309364 26757646103556137 28698353540036607 335892	6277101735386680763 8357894230526868439 95542594291093702252 6277101735386680763 8357894233626459881 8357894233626459881

Table 4.1: Curves generated over NIST prime $2^{192} - 2^{64} - 1$.

Appendix A

Elementary Number Theory

This chapter briefly explains some of the key number theory topics which are frequently discussed in the thesis. It is by no means a complete overview of these topics and the reader should look at [1], [8], [7] and [24], for a complete overview and proofs.

A.1 Greatest Common Divisor

Greatest common divisor of two integers a and b is defined as the largest positive integer that divides both a and b. It is denoted by $\gcd(a, b)$. GCD can be usually calculated with the help of Euclidean algorithm [24].

A.2 Groups

A group $\mathbb{G}$ is a set of elements with a binary operation $(*)$ defined over them. A group satisfies the following three properties [8, 24, 25]:

- The binary operation is associative in $\mathbb{G}$. This means that for any elements $a, b, c \in \mathbb{G}$, $(a * b) * c = a * (b * c)$.
- G has an identity element e, with respect to $*$. This implies that there is an element e such that for any $a \in \mathbb{G}$, $a * e = e * a = a$.
- Each element in $\mathbb{G}$ has an inverse with respect to $*$, i.e., for every $a \in \mathbb{G}$, there is an element a^{-1}, such that, $a * a^{-1} = a^{-1} * a = e$.

The order of the group $\mathbb{G}$, denoted by $|\mathbb{G}|$, is simply the number of elements in the group. If there is a finite number of elements then the group is said to be finite. The group $\mathbb{G}$ is said to be abelian if and only if, for all $a, b \in \mathbb{G}$; $a * b = b * a$.

A.3 Fields

A field $\mathbb{F}$ is a set of elements with two binary operations, namely addition, denoted by (+) and multiplication, denoted by $(\cdot)$. A field has the following properties [8, 24, 25]:

- $(\mathbb{F}, +)$ is an abelian group with an identity denoted by 0.
- $(\mathbb{F}^*, \cdot)$ is an abelian group with an identity denoted by 1, where $\mathbb{F}^* = \mathbb{F} - \{0\}$.
- The elements of the field exhibit distributive law, i.e. $(a + b) \cdot c = a \cdot c + b \cdot c$).

  The order of the field $|\mathbb{F}|$ is defined as the number of elements in the set and the field is said to be finite if the order is finite. There exists a finite field of order q, if and only if q is a prime power, i.e $q = p^k$, for some prime p. In such cases, p is said to be the characteristic of the field.

A.4 $\mathbb{Z}_n$

$\mathbb{Z}_p$ is defined as a set of positive integers modulo n.

A.5 $\mathbb{Z}_n^*$

$\mathbb{Z}_n^*$, is a group, with multiplication as the group operation and the elements of the group are all the integers having a multiplicative inverse modulo n.

A.6 Quadratic residues

Consider two coprime integers a and m, with m a positive integer. a is said to be a quadratic residue $\pmod m$, if and only if there exists an x, such that:

$$x^2 \equiv a \pmod m$$

A.7 Euler's criterion

Euler's criterion is used to find whether a number is a quadratic residue modulo a prime or not. Consider a number a, where $\gcd(a, p) = 1$ for an odd prime p. a is said to be a quadratic residue modulo p if:

$$a^{\frac{p-1}{2}} \equiv 1 \pmod p$$

a is said to be a quadratic nonresidue modulo p if,

$$a^{\frac{p-1}{2}} \equiv -1 \pmod p$$

A.8 Legendre symbol

Legendre symbol is a concise representation of Euler's criteria. Consider an integer a and an odd prime p. The Legendre symbol captures whether a is a quadratic residue modulo p or not. The formal definition of Legendre symbol can be stated as follows [7, 24]:

Definition A.8.1 (Legendre symbol) *Suppose p is an odd prime. For any integer a, define the* Legendre Symbol $\left(\frac{a}{p}\right)$ *as follows:*

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } a \equiv 0 \,(mod\ p) \\ 1 & \text{if } a \text{ is a quadratic residue modulo } p \\ -1 & \text{if } a \text{ is a quadratic non-residue modulo } p \end{cases}$$

In order to represent whether a is a quadratic residue modulo a composite number m or not, Jacobi symbols are used. More information regarding the Jacobi symbols can be found in [7].

A.9 Lagrange's theorem

Lagrange's theorem states that for any finite group $\mathbb{G}$ with order g, the order of any subgroup $\mathbb{G}'$ of $\mathbb{G}$, divides g, i.e. $|\mathbb{G}'| \Big| g$.

A.10 Chinese remainder theorem

Let $p_1, p_2, \ldots, p_n$ be distinct prime numbers and let $m = p_1^{k_1} \cdot p_2^{k_2} \cdots p_n^{k_n}$. Then the set of congruences,

$$x \equiv a \pmod{p_1^{k_1}}$$

$$x \equiv a \pmod{p_2^{k_2}}$$

$$\ldots$$

$$x \equiv a \pmod{p_n^{k_n}}$$

has a simultaneous solution, which is unique modulo m [1].

Appendix B

NIST primes

The FIPS 186-2 document,published by NIST, recommends 10 finite fields for use with the elliptic curves. Specifically, the document recommends 5 prime fields and 5 binary fields for cryptographic usage.

The recommended prime fields are as follows:

$$\begin{aligned} p_{192} &= 2^{192} - 2^{64} - 1 \\ p_{224} &= 2^{224} - 2^{96} + 1 \\ p_{256} &= 2^{256} - 2^{224} + 2^{192} + 2^{9}6 - 1 \\ p_{384} &= 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1 \\ p_{521} &= 2^{521} - 1. \end{aligned}$$

These primes have the property that they can be expressed as sums or differences of small powers of 2. Apart from p_{521} all the powers in the expressions are multiples of 32. These properties lead to extremely fast reduction algorithms which are suitable for machines having register sizes equal to 32.

As a small example take $p = p_{192} = 2^{192} - 2^{64} - 1$ and let c be an integer such that $0 \leq c \leq p^2$. Let $c = c_5 2^{320} + c_4 2^{256} + c_3 2^{192} + c_2 2^{128} + c_! 2^{64} + c_0$ be the base-64 representation of c. We can then reduce the higher powers of 2 in c using the congruences,

$$2^{192} \equiv 2^{64} + 1 \pmod{p} \tag{B.1}$$

$$2^{256} \equiv 2^{128} + 2^{64} \pmod{p} \tag{B.2}$$

$$2^{320} \equiv 2^{128} + 2^{64} + 1 \pmod{p}. \tag{B.3}$$

We thus obtain,

$$\begin{aligned} c \equiv\ & c_5 2^{128} + c_5 2^{64} + c_5 \\ & + c_4 2^{128} + c_4 2^{64} + \\ & + \qquad + c_3 2^{64} + c_3 \\ & + c_2 2^{128} + c_1 2^{64} + c_0 \pmod{p} \end{aligned}$$

Hence, c modulo p can be obtained by adding four 192-bit numbers and taking mod p at each step.

The recommended binary fields for elliptic curves are $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$, $\mathbb{F}_{2^{283}}$, $\mathbb{F}_{2^{409}}$ and $\mathbb{F}_{2^{571}}$. For each of these binary fields, a randomly chosen elliptic curve and a Koblitz curve [11] are also recommended.

For more details regarding the NIST primes, refer to [11].

Appendix C

Resources

All the programs were written using C++ and Perl. The programs were tested on a Pentium 4, 3.33MHz dual core machine with 1GB of RAM. All the programs were written by the author except where stated.

The following opensource software and libraries were used during the development and testing:

- **GNU C Compiler (GCC)** This is the standard suite of compilers that is bundled with all the major Unix/Linux distributions. The compiler optimization flags were used during the compilation of the programs to improve their performance.
- **Number Theory Library (NTL)** This is a C++ library written by Victor Shoup, to perform number theoretic computations. The library is opensourced and can be downloaded from `http://www.shoup.net`.
- **Cygwin** This is a windows port of some of the common Unix utilities like BASH and GCC.
- **MIRACL library** The MIRACL library [19] is a C++ library providing multiprecision arithmetic and a variety of number theoretic functions with a bias towards cryptographic usage. The library also provides highly optimized implementations of Schoof's algorithm and SEA algorithm, which were used in this thesis to test their performance.
- **LiDIA** Another number theory library providing multi-variate polynomial functions. The library is opensource and can be downloaded from `http://www.cdc.informatik.tu-darmstadt.de/TI/LiDIA/`.
- **GNU Multiprecision Library (GMP)** GMP is a C library for doing multiprecision arithmetic. The library is highly optimized for performance and it was used in places to improve the calculations that would normally have been very slow with NTL library. GMP can be downloaded from `http://gmplib.org`.

Bibliography

[1] Marlow Anderson and Todd Feil. *A First Course in Abstract Algebra*. Chapman & Hall/CRC, 2005.

[2] A. O. L. Atkin and F. Morain. Elliptic curves and primality proving. *Mathematics of Computation*, 61:29–68, 1993.

[3] Ian F. Blake, Gadiel Seroussi, and Nigel P. Smart, editors. *Elliptic Curves in Cryptography*. Cambridge University Press, 1999.

[4] Ian F. Blake, Gadiel Seroussi, and Nigel P. Smart, editors. *Advances in Elliptic Curve Cryptography*. Cambridge University Press, 2005.

[5] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag, 1993.

[6] Henri Cohen and Gerhard Frey. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chanpman & Hall/CRC, 2006.

[7] Richard Crandall and Carl Pomerance. *Prime Numbers, A Computational Perspective*. Springer, 2006.

[8] Wilbur E. Deskins. *Abstract Algebra*. Dove, 1966.

[9] L. Diwaghe. Remarks on the Schoof-Elkies-Atkin Algorithm. *Mathematics of Computation*, Volume 67, 1998.

[10] Noam D. Elkies. The existence of infinitely many supersingular primes for every elliptic curve over q. *Invent. Math.*, 89:561–568, 1987.

[11] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[12] Tom Harrison. Euclidean algorithm. `http://en.wikipedia.org/wiki/Euclidean_algorithm`.

[13] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[14] Hendrik W. Lenstra. Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673, 1986.

[15] John J. McGee. René Schoof's Algorithm for Determining the Order of the Group of Points on an Elliptic Curve over a Finite Field. Master's thesis, Virginia Polytechnic Institute and State University, 2006.

[16] Victor Miller. Use of elliptic curves in cryptography. *CRYPTO 85*, 218:417–426, 1985.

[17] Greg Musiker. Schoof's Algorithm for Counting Points on E($\mathbb{F}_q$), 2005.

[18] René Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of Computation*, Volume 44, 1985.

[19] Mike Scott. Miracl Library. `http://indigo.ie/~mscott/`.

[20] Daniel Shanks. Class number, a theory of factorization and genera. *Proc. Sympos. Pure Math.*, 20:415–440, 1966.

[21] Victor Shoup. Number Theory Library. `http://www.shoup.net/ntl`.

[22] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge, 2005.

[23] Joseph H. Silverman and John Tate. *Rational Points on Elliptic curves*. Springer, 1992.

[24] Douglas R. Stinson. *Cryptography, Theory and Practice*. Chapman & Hall/CRC, 2006.

[25] Lawrence C. Washington. *Elliptic Curves, Number theory and Cryptography*. Chapman & Hall/CRC, 2003.

[26] Andrew Wiles. Modular elliptic curves and fermat's last theorem. *Annals of Mathematics*, 141:443–551, 1995.