CERIAS Tech Report 2007-03

ELLIPTIC CURVE FACTORING METHOD VIA FFTS WITH DIVISION

by Zhihong Li

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

ELLIPTIC CURVE FACTORING METHOD VIA FFTS WITH DIVISION POLYNOMIALS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Zhihong Li

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2006

To my parents and my wife

ACKNOWLEDGMENTS

I wish to express my great gratitude to my advisor, Professor Samuel S. Wagstaff, Jr. whose expertise, understanding, encouragement, and patience, added considerably to my graduate study. I appreciate his profound knowledge in number theory and cryptography, and his guidance in my research. Discussion with him about problems and ideas is very valuable .

I am also grateful to Professor Bradley Lucier, Professor Freydoon Shahidi and Professor William Heinzer for being on my graduate committee. Their support and advice are highly appreciated.

I would like to thank my parents and my wife, Han for the support they provided me through my entire life. Their confidence on me and love helped me a lot during hard times.

TABLE OF CONTENTS

Page

ABSTRACT

Li, Zhihong. Ph.D., Purdue University, December, 2006. Elliptic Curve Factoring Method via FFTs with Division Polynomials. Major Professor: Samuel S. Wagstaff, Jr..

In 1985, H. W. Lenstra, Jr. discovered a new factoring method, the Elliptic Curve Method (ECM), which efficiently finds 20- to 30- digit prime factors. On January 15, 2001, C. P. Schnorr proposed an idea to apply division polynomials for ECM. In this thesis, we modify and implement the idea in detail, analyze the complexity of the algorithm and the probability of success.

We first extend the concept of division polynomial to a univariate case with the parameter a in the Weierstrass form $y^2 = x^3 + ax + b$ as the variable. We generalize a result about the degree of this implied univariate division polynomial. Then we discover an algorithm to efficiently generate the m-th univariate division polynomial and determine the complexity of the algorithm.

We then present the main algorithm of this thesis, which is the main result of the thesis as well. We demonstrate in both algebraic and geometric ways the sufficient conditions for the algorithm to be successful. We analyze the probability of success and prove the related results. To demonstrate the structure of the main algorithm, we divide it to several parts and introduce every part in detail. Then we propose and prove a theorem about the complexity of the main algorithm.

We also present an optimization of the main algorithm for a family of numbers with specific form. This family is of great interest as well. We show that for this family, the optimal algorithm can factor numbers even faster and also remove the memory restriction of the multiple m of the point on the elliptic curves, which is also the index of the implied division polynomial being used.

At the end, we address some issues related to the implementation of the algorithm. With the algorithm, we can find some 40-digit primes on a 1.86GHz 1066FBS PC with 4GB DDR2 SDRAM at 533MHz. It also finds some 50-digit primes. Now we are trying the algorithm on 60-digit primes and we hope that the algorithm will find some 70-digit primes.

1. Introduction

1.1 Algorithms for Factoring Integers

Integer factorization is a classical problem in number theory. It has been studied for centuries. It is stated in the Fundamental Theorem of Arithmetic that any positive integer can be uniquely factored into primes. A prime number is $p > 1$ such that the only divisor of p is 1 and p. Two main problems about prime numbers are primality proving and integer factorization. These two problems were studied by Gauss [10]. The latest algorithms for primality testing run in polynomial time while none of the integer factoring algorithms are that fast.

The solution of the integer factorization problem is not simple. Although remarkable progress has been achieved, the problem is still difficult.

Factoring large integers is a theoretical problem. However, many practical issues depend on it. One of them, which I will focus on, is cryptography or more precisely the public-key cryptosystem. RSA [18] is by far the easiest to understand and implement public-key cryptosystem. The security of RSA lies in the supposed intractability of factoring large integers. One can break RSA if one can find the factors of the integer n given as a public key.

The three most efficient factorization algorithms are Quadratic Sieve (QS), Elliptic Curve Method (ECM) and Number Field Sieve (NFS).

The NFS [12] is the fastest known method for factoring large integers. The Multiple Polynomial Quadratic Sieve (MPQS) [7] is faster than NFS for numbers up to 110 decimal digits.

ECM [13] is very similar to Pollard's $p-1$ algorithm [17]. The difference is that ECM uses elliptic curves instead of using powers of a constant modulo N in Pollard's $p-1$ algorithm. One advantage of ECM over Pollard's $p-1$ algorithm is that there are many elliptic curve groups modulo p, where p is a prime factor of N. The run time for ECM depends strongly on the size of the least prime factor p of N and weakly on the size of N, while the run time of QS and NFS depend on the size of N only. The worst case for ECM occurs when N is the product of two primes with similar magnitude. ECM works better than QS and NFS for very large numbers as long as the least prime factor of N is reasonably small. However, we have no idea about the size of the factors of N before we factor it. So a good strategy is to try ECM first after some trial division, and switch to QS or NFS in case of failure after a reasonable amount of tries.

1.2 ECM and an FFT Extension

In ECM, one chooses a random rational point $P = (x_0, y_0)$, then a. Finally, b is determined by $b = y_0^2 - x_0^3 - ax_0 \pmod N$. Suppose that $p \neq 2, 3$ is a prime divisor of N. Denote the elliptic curve with parameters a, b by $E_{a,b}$. Then the rational points on $E_{a,b}$ over $\mathbb{Z}/p\mathbb{Z}$ form an Abelian group called the Mordell-Weil group. If we denote the order of the Mordell-Weil group as $\#E_{a,b}[p]$, we have $p + 1 - 2\sqrt{p} < \#E_{a,b}[p] < p + 1 + 2\sqrt{p}$ by Hasse's theorem. Then one evaluates the multiple MP, where M is the product of the primes up to some pre-selected bound B_1, raised to some suitable power. If MP is the identity of $E_{a,b}$ modulo p but not the identity of $E_{a,b}$ modulo N, a prime factor p of N is found. If B_1 is larger than the largest prime divisor of $\#E_{a,b}[p]$, ECM will usually succeed. However, the reason why ECM is not a deterministic algorithm is that people have no idea about $\#E_{a,b}[p]$. This brings up an important issue about ECM, which is optimizing the parameter selection. Silverman and Wagstaff [22] have posed a very precise analysis about this.

Another major question is what to do if ECM fails to find a prime factor for the upper bound B_1. In this case, $\#E_{a,b}[p]$ might have the property that all its prime factors are less than B_1 except for the largest prime factor to lie between B_1 and some larger bound B_2. To discover such p, the two-step version of ECM was introduced.

Montgomery [15] applied the Fast Fourier Transform (FFT) for the second step and it is a great success. The algorithm generates two polynomials from the resulting multiples of the first step. It then finds the greatest common divisor of these two polynomial by FFT. This process either produces a polynomial that divides both polynomials or finds a factor of N.

1.3 ECM by Division Polynomials

The Weierstrass form of elliptic curve is used through this thesis. It is not hard to transform other forms to Weierstrass form

$$E_{a,b} : y^2 = x^3 + ax + b. \tag{1.1}$$

The basic idea of ECM is hoping that some multiple $m \cdot P$ is the infinity of the elliptic curve modulo p while it is not the infinity of the elliptic curve modulo N. It turns out that it is sufficient to consider the x-coordinate only. Division polynomials are introduced in an elliptic curve and defined inductively. The x-coordinate of $m \cdot P$ has the square of the m-th division polynomial as its denominator. Denote by ψ_m the m-th division polynomial. Then the above goal will be achieved if $\psi_m \equiv 0 \pmod{p}$ and $\psi_m \not\equiv 0 \pmod{N}$ for some m. That is, $\gcd(\psi_m, N)$ is non-trivial. Therefore, one can implement ECM by evaluating the division polynomials and checking the greatest common divisor between the values and N.

This thesis differs from the other work in that we evaluate division polynomials at many points all together by FFT and then check the gcd between the product and N. If the gcd is N, then we truncate the product. If the gcd is 1, we increase the index m of the division polynomial.

1.4 Organization of the Thesis

Chapter 2 goes over some basic properties of elliptic curves and introduces ECM.

We define division polynomials as well as implied univariate division polynomials of elliptic curves and demonstrate some facts about them in Chapter 3. We state and prove a theorem about their degrees and design an algorithm to efficiently generate high-order implied division polynomials.

Chapter 4 is the core chapter of this thesis. We explain the innovative main algorithm for factoring large integers by ECM thoroughly, propose some theorems about the conditions for the algorithm to be successful as well as the probability of success. Then prove the result about the complexity of the algorithm, which is the main result of this thesis. At the end of the chapter, we illustrate that the main algorithm can be optimized for prime numbers with some specific form. We then propose and analyze this optimized algorithm.

In Chapter 5 we demonstrate the main algorithm in more detail by focusing on how to implement and test it. Then we summarize the main results.

Chapter 6 suggests directions for further research.

2. Elliptic Curve Method

2.1 Elliptic Curve and Its Arithmetic

Let K and L be two fields with $K \subseteq L$. We say $\alpha \in L$ is **algebraic** over K if there exists a nonconstant polynomial

$$f(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_0$$

with $a_0, \ldots, a_{n-1} \in K$ such that $f(\alpha) = 0$.

Definition 2.1.1 *Given two fields K and L, L is said to be **algebraic** over K if every element of L is algebraic over K. A field K is called **algebraically closed** if every nonconstant polynomial $g(x)$, with coefficients in K, has a root in K. An **algebraic closure** of K is the smallest algebraically closed field containing K and denoted by $\overline{K}$.*

The general form of an elliptic curve E over a field K is given by the following cubic equation:

$$Y^2Z + a_0XYZ + a_1YZ^2 = X^3 + b_0X^2Z + b_1XZ^2 + b_2Z^3. \tag{2.1}$$

Notice that $\infty = (0, 1, 0)$ is a solution of (2.1). It is the basepoint and called a **point at infinity**. In (2.1), we require $a_0, a_1, b_0, b_1, b_2 \in K$. If we set $x = X/Z, y = Y/Z$, (2.1) will become

$$y^2 + a_0xy + a_1y = x^3 + b_0x^2 + b_1x + b_2. \tag{2.2}$$

If $char(\overline{K}) \neq 2, 3$, the replacement $x \rightarrow x - (a_0^2/4 + b_0)/3$ and $y \rightarrow y - (a_0x + a_1)/2$ gives the Weierstrass form of an elliptic curve.

Definition 2.1.2 *The **Weierstrass form** is*

$$E : y^2 = x^3 + ax + b, \tag{2.3}$$

where $a, b \in K$ and $4a^3 + 27b^2 \neq 0$. The ***discriminant*** *is $\Delta = 4a^3 + 27b^2$. The* ***j-invariant*** *is defined by*

$$j = 1728 \frac{4a^3}{4a^3 + 27b^2}.$$

Under the transform $x^* = \alpha^2 x$ and $y^* = \alpha^3 y$, with α be a nonzero element of the multiplicative group of $\overline{K}$, the j-invariant is unchanged. In fact the two elliptic curves are isomorphic over an algebraically closed field if they have the same j-invariant. However, this is not the case for the non-algebraically closed field.

The elliptic curve in Weierstrass form is defined as the set of all the points in $\overline{K} \times \overline{K}$ satisfying (2.3) and the point at infinity ∞.

Now let's consider the arithmetic of elliptic curves. If $P = (x_1, y_1), Q = (x_2, y_2)$ are two points on the elliptic curve E given by (2.3), the inverse of P is defined as $-P = (x_1, -y_1)$. The following is the idea for defining the sum. If $Q = -P$, define $P + Q = \infty$. If $P + Q \neq \infty$, then the line L through P and Q is not vertical so that it intersects with E at a third point $-R$. In this case, we define $P + Q = R$.

Let $R = (x_3, y_3)$, then we have the following formal definition for addition:

1. If $Q = \infty$, then $R = P + \infty = P$.
2. If $P = \infty$, then $R = Q + \infty = Q$.
3. If $Q = -P$, then $R = P + (-P) = \infty$.
4. If $x_1 \neq x_2$, then define
$$m = \frac{y_2 - y_1}{x_2 - x_1}, \tag{2.4}$$
$$x_3 = m^2 - x_1 - x_2, y_3 = m(x_1 - x_3) - y_1.$$
5. If $x_1 = x_2, y_1 = y_2$ and $y_1 \neq 0$, which means $P = Q, R = 2P$, then define
$$m = \frac{3x_1^2 + a}{2y_1}, \tag{2.5}$$
$$x_3 = m^2 - 2x_1, y_3 = m(x_1 - x_3) - y_1.$$
6. If $x_1 = x_2, y_1 = y_2$ and $y_1 = 0$, then $R = P + Q = 2P = \infty$.

It's not hard to prove that the addition defined above satisfies the commutativity, existence of identity, existence of inverses. There are several ways to prove the associativity. It can be done by the formulas for addition. It can also be proved by algebraic geometry. Please refer to Section 3.2 of [21] for more details.

Theorem 2.1.1 *The points on the elliptic curve E with the addition defined above forms an abelian group with identity ∞. It is called the **Mordell-Weil** group.*

Figure 2.1 is the graph of the elliptic curve determined by $y^2 = x^3 - x + 1$. And it illustrates the way that addition and scalar multiplication are done for the points on an elliptic curve. In Figure 2.1,

$$
\begin{aligned}
&P = (0,1) \quad Q = (1,1) \quad P + Q = (-1,-1)\\
&2P = (1/4, -7/8) \quad -2P = (1/4, 7/8)\\
&2P + Q = (1/4, -7/8) + (1,1) = (5,-11)\\
&P + (P + Q) = (0,1) + (-1,-1) = (5,-11)
\end{aligned}
$$

To locate $-P$ given P, just reflect P around the $x-$axis. To find $P + Q$, draw a line passing through P and Q. The intersection point of the line and the curve is $-(P + Q)$ so that $P + Q$ can be located by reflecting $-(P + Q)$ around $x-$axis. To find $2P$, draw the tangent line of the curve passing P, and the intersection point of the tangent line and the curve is $-2P$. Then we can find $2P = -(-2P)$.

Notice that a multiple of a point P can be computed by several additions and fast multiplication. The calculation of $Q = mP$ is as follows.

1. Let $i = m$, $Q = \infty$, $R = P$.

2. If i is even, let $i = i/2$. Set $R = 2R = R + R$.

3. If i is odd, let $i = i - 1$. Set $Q = Q + R$.

4. If $i \neq 0$, go to Step 2.

5. Return Q.

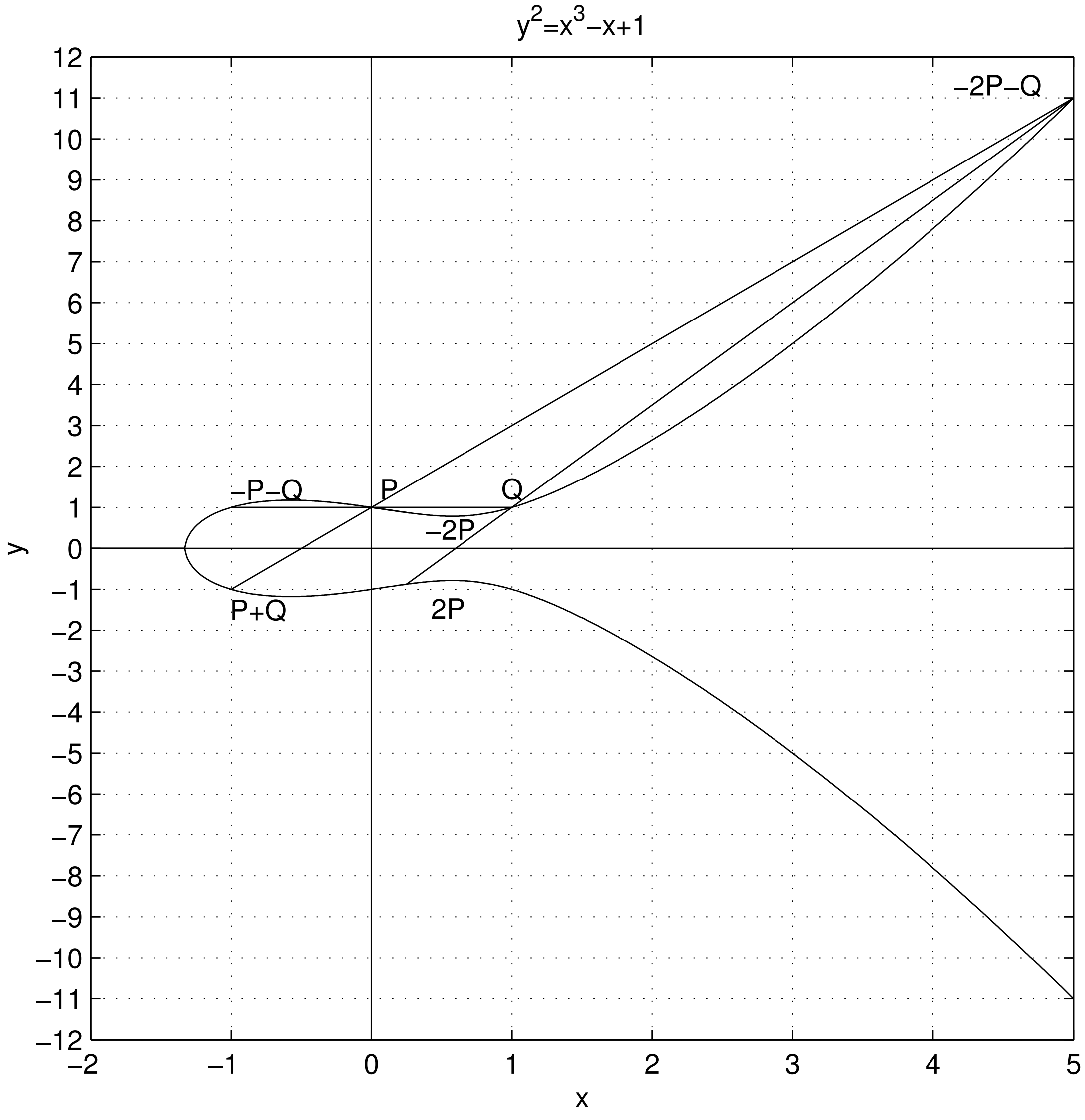

Figure 2.1. Graph of $y^2 = x^3 - x + 1$

2.2 The Order of Elliptic Curve over Finite Field

An elliptic curve over a finite field is what we deal with in ECM. Its order is very important. If p is the prime factor of N, ECM will likely succeed if all prime factors of $\#E_{a,b}[p]$ are less than some upper bound B. In this case, the order is said to be **smooth** up to B. Two questions come up now. First, how to find the order $\#E_{a,b}[p]$ of the elliptic curve over $\mathbb{F}_p$. Second, how to determine the size of the largest prime factor of an integer. We will answer the first question in this section and the second in the next section.

Theorem 2.2.1 (Hasse's Theorem) *Let $E_{a,b}[p]$ be an elliptic curve over the finite field $\mathbb{F}_p$ and its order be denoted by $\#E_{a,b}[p]$. Then*

$$p + 1 - 2\sqrt{p} \leq \#E_{a,b}[p] \leq p + 1 + 2\sqrt{p}.$$

The proof can be found in Section 5.1 of [21]. For large p, the size of $\sqrt{p}$ is much smaller than the size of p, so we can conclude that the order of the elliptic curve over $\mathbb{F}_p$ is close to p. Actually, for any integer K in interval $[p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}]$, we can choose some specific parameters a, b such that K is the order of the elliptic curve $y^2 = x^3 + ax + b$ over $\mathbb{F}_p$. Please see Deuring [8] for more details. This fact explains the advantage of ECM over Pollard's $p - 1$ method [17]. Pollard's $p - 1$ method finds the prime factor p of N by evaluating $\gcd(a^M - 1, N)$, where M is some suitable multiple of $p - 1$. So in order for the method to succeed, all prime factors of $p - 1$ have to be less than some suitable upper bound. This restricts the method to find those primes p for which $p - 1$ has no large prime factor. ECM works as long as all prime factors of $\#E_{a,b}[p]$ are less than some suitable upper bound. But the difference is that we have many choices of $\#E_{a,b}[p]$. If one curve fails, we can choose another curve.

The following theorem by Cassels [5] describes the structure of an elliptic curve over finite field.

Theorem 2.2.2 *Let $E_{a,b}[p]$ be the elliptic curve over the finite field $\mathbb{F}_p$. Then*

$$E_{a,b}[p] \simeq \mathbb{Z}_n \qquad \textit{or} \qquad E_{a,b}[p] \simeq \mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2}$$

where $n, n_1, n_2 \geq 1$ are integers and $n_1 \mid n_2$.

2.3 Smoothness and Dickman's Function

Now let's answer the second question proposed in the previous section. In ECM, the factor p will be discovered by a curve $E_{a,b}$ if the largest prime factor of $\#E_{a,b}[p]$ is smaller than B. The reason for this is that we evaluate MP with $P \in E_{a,b}$, $M = \prod_{i=1}^{n} q_i^{e_i}$, and $q_1, \ldots, q_n$ are all the primes less than B. The algorithm will succeed if MP is the infinity of $E_{a,b}[p]$ but not the infinity of $E_{a,b}[N]$. So we need to get an estimation of the size of the largest prime factor of a given integer.

Definition 2.3.1 *A positive number n is called B-**smooth** if the largest prime factor of n is less than B.*

If we denote the number of B-smooth integers in the interval $[1, x]$ as $\lambda(x, B)$, and denote by $p(x, B)$ the probability that an integer in $[1, x]$ is B-smooth, then we have $p(x, B) = \lambda(x, B)/x$.

Definition 2.3.2 ***Dickman's function*** *$\rho(t)$ for $t \geq 0$ is defined as*

$$\rho(t) = \lim_{x\to\infty} p(x, x^{1/t}) = \lim_{x\to\infty} \lambda(x, x^{1/t})/x.$$

It is easy to prove the limit exists and derive $\rho(t) = 1$ for $0 \leq t \leq 1$. Dickman [9] showed that

$$\rho'(t) = -\rho(t-1)/t \qquad \text{for} \qquad t > 1. \tag{2.6}$$

Then we can get the following result for $1 \leq t \leq 2$

$$\rho(t) = \int_0^t \rho'(s)ds = 1 - \int_1^t \frac{\rho(s-1)}{s}ds = 1 - \int_1^t \frac{1}{s}ds = 1 - \ln t.$$

By (2.6) and integrating by parts, we can get the following important theorem.

Theorem 2.3.1 *For $t > 1$, $\rho(t) = \int_{t-1}^{t} \rho(s)ds/t$.*

A good approximation by Knuth and Trabb Pardo [11] is $\rho(t) \approx t^{-t}$. Now let's consider the smoothness. If $B = n^{1/t}$, then $t = (\ln n)/(\ln B)$. We can conclude that the probability that $\#E_{a,b}[p]$ is B-smooth is

$$\rho(t) \approx t^{-t} \qquad \text{with} \qquad t = \frac{\ln \#E_{a,b}[p]}{\ln B} \approx \frac{\ln p}{\ln B}.$$

Dickman's function can also be calculated with greater accuracy. Bach and Peralta [2] gave an effective method by noticing that $\rho(t)$ is analytic on $[m-1, m]$ for any integer $m \geq 1$. There is an analytic function $\rho_m(t)$ that is equal to $\rho(t)$. Then they use a Taylor expansion for $\rho(t) = \rho_m(t) = \rho_m(m - \xi)$ on $[m-1, m]$ to get

$$\rho(t) = \sum_{i=0}^{\infty} c_i^{(m)} \xi^i$$

where the coefficients are given by

$$c_0^{(1)} = 1, \quad c_i^{(1)} = 0 \quad \text{for} \quad i > 0,$$

$$c_0^{(2)} = 1 - \ln 2, \quad c_i^{(2)} = \frac{1}{i2^i} \quad \text{for} \quad i > 0,$$

and

$$c_0^{(m)} = \frac{1}{m-1} \sum_{j=1}^{\infty} \frac{c_j^{(m)}}{j+1} \quad \text{for} \quad m > 2,$$

$$c_i^{(m)} = \sum_{j=0}^{i-1} \frac{c_j^{(m-1)}}{im^{i-j}} \quad \text{for} \quad i > 0, m > 2.$$

G. Marsaglia, Zaman and J. C. W. Marsaglia [14] gave an even more accurate method by expanding $\rho(t)$ at $m - \frac{1}{2}$ instead of m.

In the next section, we will see that a second step can be added to ECM to make it more efficient. This step 2 deals with the case that $\#E_{a,b}[p]$ is B_2-smooth but its second largest prime factor is less than B_1 ($< B_2$).

Definition 2.3.3 *An integer n is called k-**semismooth** with respect to y and z if the largest prime factor of n is $\leq y$ and its $(k+1)$st largest prime factor is $\leq z$.*

Let $\lambda_k(x, y, z)$ denote the number of k-semismooth integers in $[1, x]$. If we denote by $p_k(x, B_2, B_1)$ the probability that an integer in $[1, x]$ is k-semismooth with respect to B_2 and B_1, then we have $p_k(x, B_2, B_1) = \lambda_k(x, B_2, B_1)/x$. Define

$$\rho(t_1, t_2) = \lim_{x\to\infty} p_1(x, x^{t_2/t_1}, x^{1/t_1}) \quad \text{for} \quad t_1 > t_2 > 1.$$

Knuth and Trabb Pardo [11] gave the functional equation for $\rho(t_1, t_2)$ as

$$\rho(t_1, t_2) = \frac{1}{t_1 - t_2} \int_{t_1 - t_2}^{t_1 - 1} \rho(t) dt$$

where $\rho(t)$ is Dickman's function. The probability that the 2-step ECM with parameters B_1, B_2 succeeds in finding p is approximately $\rho(\frac{\ln p}{\ln B_1}, \frac{\ln B_2}{\ln B_1})$.

Zhang [25] gave a rigorous proof for the recurrence formulae to $\lim_{x\to\infty} p_k(x, x^t, x^s)$ and some numerical results for calculating it.

2.4 Lenstra's Algorithm

In February 1985, H. W. Lenstra, Jr. [13] invented the ECM algorithm. He published it in 1987. It is obtained from Pollard's $p - 1$ method by replacing the multiplicative group by the Mordell-Weil group. If the parameters are chosen properly, the expected number of group additions performed is $L(p)^{\sqrt{2+o(1)}}$ where

$$L(p) = \exp(\sqrt{\ln p \ln \ln p}).$$

Silverman and Wagstaff [22] suggested an optimal parameter selection technique and run-time guidelines for 2-step ECM. The worst case for ECM happens when $N = p \cdot q$ with p and q sharing the same order of magnitude. In this case, the complexity of ECM becomes $L(N)^{1+o(1)}$.

Lenstra's method revises the addition on elliptic curve over $\mathbb{Z}/N\mathbf{Z}$. Given $P = (x_1, y_1)$, $Q = (x_2, y_2)$, let $R = (x_3, y_3) = P + Q$ with $x_i, y_i < N$ for $i = 1, 2, 3$, and that $p|N$ is one of the prime factors for the number N we want to factor, the revised addition algorithm by Lenstra is

1. If $Q = \infty$, then set $R = P + \infty = P$ and stop by asserting failure and choose another point.
2. If $P = \infty$, then set $R = Q + \infty = Q$ and stop by asserting failure and choose another point.
3. Calculate $d = \gcd(x_1 - x_2, N)$ by the extended Euclidean algorithm, we will get the follow 3 subcases:
   (a) If $1 < d < N$, then set $p = d$ and stop by declaring success for factoring.
   (b) If $d = 1$, then the extended Euclidean algorithm gives $(x_1 - x_2)^{-1}$ modulo N and we can apply (2.4) to get R.
   (c) If $d = N$, then $x_1 = x_2$. In this case, calculate $d = \gcd(y_1 + y_2, N)$ by the extended Euclidean algorithm and we will get 3 more subcases here:
      i. If $1 < d < N$, then set $p = d$ and stop by declaring success for factoring.
      ii. If $d = N$, then $y_1 = -y_2 \Rightarrow Q = -P$ and set $R = \infty$ and stop by asserting the failure for factoring.
      iii. If $d = 1$, then by the symmetry, we can conclude $y_1 = y_2 \Rightarrow P = Q$. The extended Euclidean algorithm gives $(y_1 + y_2)^{-1} = (2y_1)^{-1}$ modulo N and we can apply (2.5) to get R.

In Lenstra's method, a random point $P = (x, y)$ with $1 \le x, y < N$ is selected first, then an elliptic curve $E_{a,b}[N]$ is chosen such that $P \in E_{a,b}[N]$ (this is done by choosing $a < N$ randomly and letting $b \equiv y^2 - x^3 - ax \pmod{N}$). Then one calculates $R = \prod_{i=1}^{l} q_i^{e_i} \cdot P$, where $q_1, \ldots, q_l$ are all the primes less than some suitable upper bound B and $e_1, \ldots, e_l$ are suitable powers. A good choice for the e_i is the largest e so that $q_i^e < \sqrt{N}$. Compute R by the formulas $P_0 = P$, $P_j = q_j^{e_j} \cdot P_{j-1}$ for $1 \le j < l$, and $R = P_l$. Use fast multiplication to compute $q_j^{e_j} \cdot P_{j-1}$. Each point addition or doubling in the fast multiplication is done using Lenstra's revised addition algorithm. In case N is factored in Step 3(a) or 3(c)i, then of course R will not be computed.

If a failure is asserted in some step, then change a (x, y can also be changed but not necessarily) and start over. If R is successfully computed, save it and start step 2 of ECM.

Now let's go back to the addition algorithm. Case 1 and case 2 will never happen for factoring algorithm since the algorithm will be stopped with asserting a failure when the ∞ of $E_{a,b}[N]$ is hit. Case 3(b) and case 3(c)iii will lead to the next step of computing R for the current curve. Case 3(c)ii asserts a failure and asks for a new curve. Case 3(c)i means $x_1 = x_2$ and $y_1 = -y_2$ if we reduce them to $E_{a,b}[p]$. That is, $k \cdot P = -P$ in $E_{a,b}[p]$ for some k. Then $(k+1) \cdot P = \infty$ in $E_{a,b}[p]$. Case 3(a) means $x_1 = x_2$ in $E_{a,b}[p]$. That is, $k \cdot P = P$ or $k \cdot P = -P$ in $E_{a,b}[p]$. If $k \cdot P = P$, then $(k-1) \cdot P = \infty$. So if the algorithm succeeds, some multiple of P must hit the infinity of the elliptic curve over $\mathbb{F}_p$ but not the infinity of the curve over $\mathbb{Z}/N\mathbb{Z}$.

2.5 The Second Step of ECM

Lenstra pointed out the possibility and necessity of adding a second step to his algorithm. Both Montgomery [16] and Brent [3] proposed methods to implement this second step. Both methods use the output of the first step in case of its failure. It's a point $R = m \cdot P$ on $E_{a,b}[N]$. Assume that $\#E_{a,b}[p]$ is 1-semismooth with respect to B_2 and B_1 and that the largest prime factor of $\#E_{a,b}[p]$ is q with $B_1 < q < B_2$. Then $\#E_{a,b}[p] | mq \Rightarrow (mq) \cdot P = \infty$ of $E_{a,b}[p]$. So $q \cdot R$ is the infinity over $E_{a,b}[p]$.

In the second step of each method, a sequence of multiples of R, $\{n_i \cdot R\}_{i=1}^{r}$, is computed so that for each prime $q_1 \in [B_1, B_2]$ there exist integers i and j with $q_1 | (n_i - n_j)$. When this happens, $n_i \cdot R$ and $n_j \cdot R$ are the same modulo p, and then p can be found if its largest prime factor q equals q_1 since these two points are unlikely to be the same modulo N. If we consider the x-coordinates $\{x_i\}_{i=1}^{r}$ of this sequence of points, the problem we will face in step 2 is to find x_i and x_j such that $\gcd(x_i - x_j, N)$ is non-trivial. Montgomery [15] constructed a polynomial from these values and applied the FFT to implement checking for a match.

2.6 Optimal Parameter Selection

When applying ECM to factor a large number N, since we have no idea about the size of $\#E_{a,b}[p]$ or p, we have to find a way to optimally choose the parameter B_1, B_2 and the number of curves L. Silverman and Wagstaff [22] elaborated on this issue and proposed a good way for choosing parameters and changing them after failure. They constructed a table and suggested values for B_1 and B_2 for each size of prime factor p expected to be found.

Up to now, we reviewed the basic idea of Lenstra's ECM and the second step as well as the way to select parameters B_1, B_2. To finish this chapter and make things clearer, let's quote the guidelines to implement ECM, provided by Silverman and Wagstaff [22].

1. Do trial division up to some prime around $\ln^2 N$.

2. Guess a possible size for the least prime factor of N based on the effort expended so far to try to factor N.

3. Select B_1 and B_2 based on this guess and run a 2-step ECM. If it succeeds, then stop the algorithm. If it fails to find a prime factor, choose another curve and run ECM with B_1 and B_2 again until the algorithm succeeds or a certain number of curves have been tried.

4. If the algorithm fails at the end of Step 3, go back to Step 2.

5. If the previous 4 steps have not succeeded for a reasonable runtime, switch ECM to MPQS.

This thesis tells how to do Step 3 efficiently with many curves computed simultaneously. Our algorithm is most useful for factoring number N known to have no small prime factors.

3. Division Polynomials

3.1 Definition and Properties

The key point at which the elliptic curve factoring method proposed in this thesis differs from Lenstra's method is the use of division polynomials. Division polynomials are defined inductively.

Definition 3.1.1 *For the Weierstrass form $y^2 = x^3 + ax + b$, the m-th* ***division polynomial*** $\psi_m(a, b, x, y) \in \mathbb{Z}[a, b, x, y]$ *by*

$$\psi_1 = 1, \qquad \psi_2 = 2y, \tag{3.1}$$

$$\psi_3 = 3x^4 + 6ax^2 + 12bx - a^2, \tag{3.2}$$

$$\psi_4 = 4y(x^6 + 5ax^4 + 20bx^3 - 5a^2x^2 - 4abx - 8b^2 - a^3), \tag{3.3}$$

$$\psi_{2m+1} = \psi_{m+2}\psi_m^3 - \psi_{m-1}\psi_{m+1}^3 \qquad (m \geq 2), \tag{3.4}$$

$$2y\psi_{2m} = \psi_m(\psi_{m+2}\psi_{m-1}^2 - \psi_{m-2}\psi_{m+1}^2) \qquad (m \geq 3). \tag{3.5}$$

Further, two more polynomials are defined by

$$\phi_m = x\psi_m^2 - \psi_{m+1}\psi_{m-1}, \tag{3.6}$$

$$4y\omega_m = \psi_{m+2}\psi_{m-1}^2 - \psi_{m-2}\psi_{m+1}^2. \tag{3.7}$$

Recall that $\Delta = 4a^3 + 27b^2$.

Theorem 3.1.1 *For the division polynomial ψ_m, we have*

1. *ψ_m, ϕ_m, $y^{-1}\omega_m$ (for m odd) and $(2y)^{-1}\psi_m$, ϕ_m, ω_m (for m even) are polynomials in $\mathbb{Z}[a, b, x, y^2]$. Using the Weierstrass form $y^2 = x^3 + ax + b$, these polynomials may be considered in $\mathbb{Z}[a, b, x]$.*

2. *The highest order terms with respect to x of $\phi_m(a,b,x)$ and $\psi_m(a,b,x)^2$ are x^{m^2} and $m^2x^{m^2-1}$ respectively.*

3. *If $\Delta \neq 0$, then $\phi_m(x)$ and $\psi_m(x)^2$ are relatively prime.*

4. *If $\Delta \neq 0$ and $P = (x, y)$, then*

$$m \cdot P = \left(\frac{\phi_m(a,b,x)}{\psi_m(a,b,x)^2}, \frac{\omega_m(a,b,x)}{\psi_m(a,b,x)^3} \right). \tag{3.8}$$

5. *The endomorphism of $E_{a,b}$ given by multiplication by m has degree m^2.*

The proof of these properties can be found in Section 9.5 of Washington [24]. (3.8) is the most important to the factoring method presented in this thesis. We sketch the proof of it here.

If $\omega_1, \omega_2 \in \mathbb{C}$ are linearly independent over $\mathbb{R}$, then call

$$L = \mathbb{Z}\omega_1 + \mathbb{Z}\omega_2 = \{n_1\omega_1 + n_2\omega_2 | n_1, n_2 \in \mathbb{Z}\}$$

a **lattice**. Then $\mathbb{C}/L$ is a torus. A **doubly periodic function**

$$f : \mathbb{C} \to \mathbb{C} \cup \infty$$

is one for which

$$f(z + \omega) = f(z) \quad \text{for all} \quad z \in \mathbb{C} \quad \text{and all} \quad \omega \in L.$$

Now we consider a special doubly periodic function, called **Weierstrass $\wp$-function** and defined by

$$\wp(z) = \wp(z; L) = \frac{1}{z^2} + \sum_{\omega \in L, \omega \neq 0} \left(\frac{1}{(z-\omega)^2} - \frac{1}{\omega^2} \right). \tag{3.9}$$

In fact every doubly periodic function can be written as a rational function of $\wp$ and its derivative $\wp'$. By setting

$$G_k = \sum_{\omega \in L, \omega \neq 0} \omega^{-k},$$

we can rewrite

$$\wp(z) = \frac{1}{z^2} + \sum_{j=1}^{\infty}(2j+1)G_{2j+2}z^{2j}. \tag{3.10}$$

And it is not hard to prove by direct calculation that

$$\wp'(z)^2 = 4\wp(z)^3 - 60G_4\wp(z) - 140G_6.$$

Therefore, we can regard $(\wp(z), \frac{1}{2}\wp'(z))$ as a point on an elliptic curve over $\mathbb{C}$. Then we have the following important theorem about elliptic curves.

Theorem 3.1.2 *Let L be a lattice and $E(\mathbb{C})$ be the elliptic curve $y^2 = 4x^3 - 60G_4x - 140G_6$ over $\mathbb{C}$. The map*

$$\mathbb{C}/L \to E(\mathbb{C})$$

$$z \mapsto (\wp(z), \wp'(z)), 0 \mapsto \infty$$

is a group isomorphism.

The proof can be found in Section 9.2 of Washington [24].

Now to finish the proof of (3.8), we define

$$f_m(z)^2 = m^2 \prod_{u\in(\mathbb{C}/L)[m], u\neq 0} (\wp(z) - \wp(u))$$

and can prove that $f_m(z)$ is a doubly periodic function. There are some relationships between $\wp(z)$ and $f_m(z)$ such as

$$\wp(nz) = \wp(z) - \frac{f_{n-1}(z)f_{n+1}(z)}{f_n(z)^2}, \tag{3.11}$$

$$f_{2n+1} = f_{n+2}f_n^3 - f_{n+1}^3 f_{n-1}, \tag{3.12}$$

$$\wp' f_{2n} = f_n(f_{n+2}f_{n-1}^2 - f_{n-2}f_{n+1}^2), \tag{3.13}$$

$$f_n(z) = \psi_n(\wp(z), \frac{1}{2}\wp'(z)), \tag{3.14}$$

$$\wp'(nz) = \frac{f_{2n}}{f_n^4}. \tag{3.15}$$

(3.8) can be proved for an elliptic curve over $\mathbb{C}$ by noticing

$$(x, y) = (\wp(z), \frac{1}{2}\wp'(z)), \quad m \cdot (x, y) = (\wp(nz), \frac{1}{2}\wp'(nz))$$

for some z.

For an elliptic curve $E_{a,b}$ over a field K with non-zero characteristic (not 2), the proof is done by constructing a homomorphism between R_M and $K(x,y)$, where $R = \mathbb{Z}[\alpha, \beta, X, Y]$ for α, β, X being independent transcendental elements of $\mathbb{C}$, M is a maximal ideal of R, and R_M is the localization of R at M. Then the multiple of (X, Y) on the elliptic curve $Y^2 = X^3 + \alpha X + \beta$ over R_M is equivalent with the multiple of (x, y) on the elliptic curve $E_{a,b}[K]$. It has been proved that (3.8) holds for the former case. Hence, (3.8) holds for elliptic curve over any field with characteristic not equal to 2.

3.2 Implied Univariate Division Polynomials

Provided that the base field K is infinite, notice that for the same point $P = (x, y)$, we can find infinitely many elliptic curves passing through it by changing the parameter a. So can we vary the elliptic curves passing through some fixed point by varying a. If we choose and fix the point $P = (x, y)$, then we can regard the division polynomials as polynomials with a being the independent variable since b can be expressed by $b = y^2 - x^3 - ax$. This polynomial is called **the m-th implied univariate division polynomial** and denoted by $\bar{\psi}_m(a)$. It is obvious that (3.8) is also true for $\bar{\psi}_m(a)$.

In the factoring algorithm I am presenting in this thesis, $\bar{\psi}_m$ plays an important role. Choose a number a. Then by (3.8) the value $\bar{\psi}_m^2(a)$ gives the denominator of the x-coordinate of $m \cdot P$. This value will tell us whether $m \cdot P$ is the infinity point of the elliptic curve with a as the parameter. To evaluate $\bar{\psi}_m(a)$ efficiently, we need to know the degree of the implied univariate division polynomial. I have deduced the following theorem about the degree of $\bar{\psi}_m$.

Theorem 3.2.1 *If $\deg(\bar{\psi}_m)$ denotes the degree of $\bar{\psi}_m$, then*

$$\deg(\bar{\psi}_m) = \begin{cases} (\frac{m}{2} - 1)(\frac{m}{2} + 1) = \frac{m^2}{4} - 1 & \textit{if } m \textit{ is even} \\ \frac{m-1}{2}\frac{m+1}{2} = \frac{m^2-1}{4} & \textit{if } m \textit{ is odd} \end{cases} \tag{3.16}$$

Proof For future reference, let's write out $\bar{\psi}_m$ for $m = 1, 2, 3, 4$.

$$\begin{aligned} \bar{\psi}_1 &= 1, \qquad \bar{\psi}_2 = 2y, & (3.17)\\ \bar{\psi}_3 &= -a^2 + 6x^2a + 12xb + 3x^4 \\ &= -a^2 + 6x^2a + 12x(y^2 - x^3 - ax) + 3x^4 \\ &= -a^2 - (6x^2)a + (12xy^2 - 9x^4), & (3.18) \end{aligned}$$

$$\begin{aligned} \bar{\psi}_4 &= 4y(x^6 + 5x^4a + 20x^3b - 5x^2a^2 - 4xab - 8b^2 - a^3) \\ &= 4y(x^6 + 5x^4a + 20x^3(y^2 - x^3 - ax) - 5x^2a^2 - 4xa(y^2 - x^3 - ax) \\ &\quad -8(y^2 - x^3 - ax)^2 - a^3) \\ &= -(4y)a^3 - (36x^2y)a^2 + (48xy^3 - 108x^4y)a \\ &\quad +(144x^3y^3 - 108x^6y - 32y^5). & (3.19) \end{aligned}$$

We see that (3.16) is true for $\bar{\psi}_1$, $\bar{\psi}_2$, $\bar{\psi}_3$, $\bar{\psi}_4$.

Assume (3.16) is true for $m \leq 2k$. It suffices to prove that (3.16) is true for $m = 2k + 1$ and $m = 2k + 2$.

1). $\bar{\psi}_{2k+1} = \bar{\psi}_{k+2}\bar{\psi}_k^3 - \bar{\psi}_{k-1}\bar{\psi}_{k+1}^3$

If k is even,

$$\begin{aligned} \deg(\bar{\psi}_{2k+1}) &= \max(\deg(\bar{\psi}_{k+2}) + 3\deg(\bar{\psi}_k), \deg(\bar{\psi}_{k-1}) + 3\deg(\bar{\psi}_{k+1})) \\ &= \max\left(\frac{(k+2)^2}{4} - 1 + 3\left(\frac{k^2}{4} - 1\right), \frac{(k-1)^2 - 1}{4} + 3\frac{(k+1)^2 - 1}{4}\right) \\ &= \max(k^2 + k - 3, k^2 + k) = k^2 + k = \frac{(2k+1)^2 - 1}{4}. \end{aligned}$$

If k is odd,

$$\begin{aligned} \deg(\bar{\psi}_{2k+1}) &= \max(\deg(\bar{\psi}_{k+2}) + 3\deg(\bar{\psi}_k), \deg(\bar{\psi}_{k-1}) + 3\deg(\bar{\psi}_{k+1})) \\ &= \max\left(\frac{(k+2)^2 - 1}{4} + 3\frac{k^2 - 1}{4}, \frac{(k-1)^2}{4} - 1 + 3\left(\frac{(k+1)^2}{4} - 1\right)\right) \\ &= \max(k^2 + k - 4, k^2 + k) = k^2 + k = \frac{(2k+1)^2 - 1}{4}. \end{aligned}$$

2). $\bar{\psi}_{2k+2} = (2y)^{-1}\bar{\psi}_{k+1}(\bar{\psi}_{k+3}\bar{\psi}_k^2 - \bar{\psi}_{k-1}\bar{\psi}_{k+2}^2)$

If k is even,

$$\begin{aligned}
\deg(\bar{\psi}_{2k+2}) &= \deg(\bar{\psi}_{k+1}) + \max(\deg(\bar{\psi}_{k+3}) + 2\deg(\bar{\psi}_k), \deg(\bar{\psi}_{k-1}) + 2\deg(\bar{\psi}_{k+2})) \\
&= \frac{(k+1)^2-1}{4} + \\
&\quad \max\left(\frac{(k+3)^2-1}{4} + 2\left(\frac{k^2}{4}-1\right), \frac{(k-1)^2-1}{4} + 2\left(\frac{(k+2)^2}{4}-1\right)\right) \\
&= \frac{k^2+2k}{4} + \max\left(\frac{3k^2+6k}{4}, \frac{3k^2+6k}{4}\right) \\
&= k^2+2k = \frac{(2k+2)^2}{4} - 1.
\end{aligned}$$

If k is odd,

$$\begin{aligned}
\deg(\bar{\psi}_{2k+2}) &= \deg(\bar{\psi}_{k+1}) + \max(\deg(\bar{\psi}_{k+3}) + 2\deg(\bar{\psi}_k), \deg(\bar{\psi}_{k-1}) + 2\deg(\bar{\psi}_{k+2})) \\
&= \frac{(k+1)^2}{4} - 1 + \\
&\quad \max\left(\frac{(k+3)^2}{4} - 1 + 2\frac{k^2-1}{4}, \frac{(k-1)^2}{4} - 1 + 2\frac{(k+2)^2-1}{4}\right) \\
&= \frac{k^2+2k-3}{4} + \max\left(\frac{3k^2+6k+3}{4}, \frac{3k^2+6k+3}{4}\right) \\
&= k^2+2k = \frac{(2k+2)^2}{4} - 1.
\end{aligned}$$

Now, by induction, (3.16) is true for every m. This proves the theorem. ■

3.3 Algorithm to Effectively Generate Implied Division Polynomials

We need to evaluate the m-th implied univariate division polynomial in the factoring algorithm. Since division polynomials get large rapidly in the sense of the sizes of both coefficients and the degree as m gets larger, we need to find a technique to control this growth in order to save the storage in the memory.

First of all, the large numbers are stored in 1-dimensional arrays with the first element indicating the size and sign of the number and the radix being some appropriate large number such as 2^{30}. Assume the size of the array to be 50. To control the size of the coefficients, we implement all the calculations modulo N. To store the

m-th implied univariate division polynomial, we need a 2-dimensional array with the size $\frac{m^2}{4} \times 50$. Obviously we cannot store all the implied division polynomials up to some index m. The good news is that we only need several implied division polynomials with lower index in order to deduce the one with higher index. After carefully inspecting the inductive definition, we find that only 7 adjacent implied division polynomials have to be stored. Therefore, we can use dynamic memory allocation to save storage. The algorithm is given below.

Algorithm 3.3.1 *(Defining the m-th implied division polynomial) Given a positive integer m and a pair $(x, y) \in \mathbb{Z}_N \times \mathbb{Z}_N$, the algorithm generates a 2-dimensional array which represents the coefficients of the $m-$th implied division polynomial.*

```
1. [Initialization]
   Factor $m-3, \dots, m+3$ to the form of $d \cdot 2^e$ with $d, e \in \mathbb{N}$ and $d$ being odd.
   From these seven $d$ values, select the smallest and call it $d^*$;
   round=e (corresponding to $d^*$);
   if ($d^*$==1) {New $(\psi_1, \psi_2, \psi_3, \psi_4, \psi_5, \psi_6, \psi_7)$ and initialize them by setting $\psi_i = \bar{\psi}_i$ for $\bar{\psi}_i$ in (3.17), (3.18), (3.19), (3.4), (3.5);}
   if ($d^*$==3) {New $(\psi_1, \psi_2, \psi_3, \psi_4, \psi_5, \psi_6, \psi_7)$ and initialize them by setting $\psi_1$=1 and $\psi_i = \bar{\psi}_{i-1}$ for $i = 2, \dots, 7$ in (3.17), (3.18), (3.19), (3.4), (3.5);}
   else {Find $\bar{\psi}_{d^*-3}, \bar{\psi}_{d^*-2}, \bar{\psi}_{d^*-1}, \bar{\psi}_{d^*}, \bar{\psi}_{d^*+1}, \bar{\psi}_{d^*+2}, \bar{\psi}_{d^*+3}$ by calling algorithm 3.3.1;
   New $(\psi_1, \psi_2, \psi_3, \psi_4, \psi_5, \psi_6, \psi_7)$ and initialize them by
   $(\bar{\psi}_{d^*-3}, \bar{\psi}_{d^*-2}, \bar{\psi}_{d^*-1}, \bar{\psi}_{d^*}, \bar{\psi}_{d^*+1}, \bar{\psi}_{d^*+2}, \bar{\psi}_{d^*+3})$; }

2. [Loop]
   for (i=0; i<round; i++) {New $(\psi'_1, \psi'_2, \psi'_3, \psi'_4, \psi'_5, \psi'_6, \psi'_7)$ with size of $(d^*2^i-2)(d^*2^i-1), (d^*2^i-1)^2, (d^*2^i-1)(d^*2^i), (d^*2^i)^2, (d^*2^i)(d^*2^i+1), (d^*2^i+1)^2, (d^*2^i+1)(d^*2^i+2)$;
   Map $(\psi_1, \psi_2, \psi_3, \psi_4, \psi_5, \psi_6, \psi_7)$ to $(\psi'_1, \psi'_2, \psi'_3, \psi'_4, \psi'_5, \psi'_6, \psi'_7)$ by the relation described in (3.4),(3.5);
```

```
    Free $(\psi_1, \psi_2, \psi_3, \psi_4, \psi_5, \psi_6, \psi_7)$;
    New $(\psi_1, \psi_2, \psi_3, \psi_4, \psi_5, \psi_6, \psi_7)$ with the same size as current
    $(\psi'_1, \psi'_2, \psi'_3, \psi'_4, \psi'_5, \psi'_6, \psi'_7)$;
    Copy $(\psi'_1, \psi'_2, \psi'_3, \psi'_4, \psi'_5, \psi'_6, \psi'_7)$ to $(\psi_1, \psi_2, \psi_3, \psi_4, \psi_5, \psi_6, \psi_7)$;
    Free $(\psi'_1, \psi'_2, \psi'_3, \psi'_4, \psi'_5, \psi'_6, \psi'_7)$; }
3. [Selecting $\bar{\psi}_m$]
    Locate $\bar{\psi}_m$ by checking the degree of the 7 polynomials;
    Free $(\psi_1, \psi_2, \psi_3, \psi_4, \psi_5, \psi_6, \psi_7)$;
    return $\bar{\psi}_m$.
```

We must now show that Algorithm 3.3.1 is correct.

Theorem 3.3.1 *Algorithm 3.3.1 computes the m-th implied division polynomial.*

Proof Given $\bar{\psi}_l, \ldots, \bar{\psi}_{l+6}$ for some positive integer l, we can get $\bar{\psi}_{2l+3}, \ldots, \bar{\psi}_{2l+9}$ in the first step by (3.4),(3.5). By the same argument, we can get $\bar{\psi}_{4l+9}, \ldots, \bar{\psi}_{4l+15}$ in the second step. Now I claim that we can get $\bar{\psi}_{index-3}, \ldots, \bar{\psi}_{index+3}$ in the i-th step, where

$$\text{index} = 2^i l + 3(1 + 2 + 2^2 + \ldots + 2^{i-1}) + 3 = 2^i l + 3\frac{2^i - 1}{2 - 1} + 3 = 2^i(l + 3).$$

We can prove this statement by induction. By the argument at the beginning, it is true for $i = 1, 2$. Assume it is true for the i-th step. This means that we have 7 implied division polynomials $\bar{\psi}_{index-3}, \ldots, \bar{\psi}_{index+3}$ computed, where

$$\text{index} = 2^i l + 3(1 + 2 + 2^2 + \ldots + 2^{i-1}) + 3 = 2^i l + 3\frac{2^i - 1}{2 - 1} + 3 = 2^i(l + 3).$$

By (3.4) and (3.5), we can compute $\bar{\psi}_{2(index-3)+3}, \ldots, \bar{\psi}_{2(index-3)+9}$ in the $(i+1)$-th step. By simplification, these 7 implied division polynomials are $\bar{\psi}_{2\cdot index-3}, \ldots, \bar{\psi}_{2\cdot index+3}$. Therefore the corresponding value for index is

$$index^* = 2 \cdot index = 2 \cdot 2^i(l + 3) = 2^{i+1}(l + 3).$$

This finishes the proof of the claim.

Notice that if any of $\bar{\psi}_{m-3}, \ldots, \bar{\psi}_{m+3}$ can be computed, $\bar{\psi}_m$ can be computed also since the difference of indices is at most 3. Step 1 of the algorithm is just to make sure that the index has the form $2^k(l+3)$.

In step 2, $l = d^* - 3$. Since the number of steps in the loop is $round = e$, the 7 implied division polynomials we get at the last step of the loop are $\bar{\psi}_{m^*-3}, \ldots, \bar{\psi}_{m^*+3}$ with $m^* = d^* \cdot 2^e$. And it has been shown that $\bar{\psi}_m$ is one of them. So the algorithm works correctly. ■

The polynomial multiplication in this algorithm can be implemented by fast Fourier transform (FFT). Please see Chapter 4 for more details about FFT. Since dynamical memory allocation is used a lot in the algorithm, the memory consumption is reduced from $O(m^3)$ to $O(m^2)$ and storage is the main issue when we are dealing with division polynomials.

In Algorithm 3.3.1, we apply the dynamic programming technique in the computation. The algorithm uses only $O(m^2 \ln m)$ arithmetic operations including additions, subtractions and multiplications. We state this as the following theorem and prove it. This result is important for concluding the time complexity of the main algorithm in the thesis, which is Algorithm 4.1.1 in Chapter 4.

Theorem 3.3.2 *For any positive integer m, Algorithm 3.3.1 can compute the m-th implied division polynomial $\bar{\psi}_m$ by $O(m^2 \ln m)$ arithmetic operations.*

To generate $\bar{\psi}_m$, we need to generate up to 5 implied division polynomials with indices around $m/2$. For these 5 adjacent polynomials to be generated, we need to compute up to 7 implied division polynomials with indices around $m/4$. So we only need to go through $\ln m$ recursion levels. The misleading way is to think that in each recursion level we need to apply FFT to do polynomial multiplications for polynomials with degree $O(m^2)$, so that the total time cost of the algorithm is $O(m^2(\ln m)^2)$. This is not correct since in the early recursion levels, the degree of the implied division polynomials are quite small. The following is the proof of the theorem.

Proof Without losing generality, we assume $m = 2^e$. Then there are e loops. In the i-th loop, the degree of the 7 implied division polynomials is $O(2^{2i})$. We implement the polynomial multiplication by FFT. For degree 2^{2i} polynomials, the FFT can be done by $O(2^{2i} \ln 2^{2i}) = O(i \cdot 2^{2i})$ arithmetic operations. Therefore, the total time needed by the algorithm is

$$O(2 \cdot 2^{2 \cdot 2}) + O(3 \cdot 2^{2 \cdot 3}) + \cdots + O(e \cdot 2^{2e}) = O(e \cdot 2^{2e}) = O(m^2 \ln m).$$

$\blacksquare$

3.4 Evaluating m-th Implied Division Polynomial

At the next stage of factoring after we get the m-th implied division polynomial by algorithm 3.3.1, we evaluate $\bar{\psi}_m$ at $m^2/4$ values of a. We will choose $a = 2^i$ for $0 \leq i < m^2/4$. For these values the evaluation can be done effectively by FFT. The polynomial evaluation algorithms will be discussed in the next chapter. In this section, I will only briefly discuss the feasibility of the factoring algorithm presented in this thesis. More details can be found in later chapters.

Given $P = (x, y)$, by evaluating $\bar{\psi}_m$ at $m^2/4$ values of a, we actually try to find $m \cdot P$ for $m^2/4$ different elliptic curves at the same time. In Lenstra's method, m is the product of all the primes less than some upper bound B_1 raised to some suitable power. But only one elliptic curve is tried at a time. An intuitive conclusion is that m in the algorithm shown in this thesis should be much smaller than the m in Lenstra's ECM since we try many elliptic curves all at a time. Theoretically this conclusion is also reasonable. As discussed in chapter 2, ECM will succeed if $\#E_{a,b}[p]$ is B_1-smooth. By trying $m^2/4$ curves all together, we hope that some curve has its order over $\mathbb{F}_p$ being very smooth. That is $\#E_{a,b}[p]$ is B_1-smooth for some reasonably small B_1. Then the scalar multiple m does not have to be very large. We can choose

m to be the product of the primes less than B_1. Recalling Dickman's function $\rho(t)$, we need to inspect the distribution of $\rho(t)$ for large t. Zhang [25] gave some good result about this.

Another concern is how hard it is to have $\#E_{a,b}[p]$ being very smooth given randomly chosen a. Lenstra [13] proved that $\#E_{a,b}[p]$ is well distributed in $[p+1-2\sqrt{p}, p+1+2\sqrt{p}]$ when the point $P=(x,y)$ is fixed, a is chosen randomly and $b=(y^2-x^3-ax) \bmod N$.

We will give some rigorous discussion about these issues in Chapter 4.

4. ECM with Division Polynomials

4.1 Factoring Algorithm by ECM with Division Polynomials

C. P. Schnorr [19] proposed a new idea to apply division polynomials in ECM on January 15, 2001. When it was proposed, it was believed that the new method required too much memory to work. In this thesis, we reconsider this new algorithm for factoring a large number N by ECM with division polynomials, solve the memory problem and make it work in a practical way. Now we state the main outline of the algorithm in this section. We will analyze the feasibility and go through some more details in the later sections.

Algorithm 4.1.1 *(Factoring N by ECM with division polynomials) Given a positive integer N, the algorithm finds a factor p of N or stops by asserting a failure to factor N.*

1. [Selecting the point and s]

Choose $0 < x, y < N$ and a small prime s.

2. [Setting bound]

Choose an appropriate upper bound B. Let R be the set of primes less than B. Define $m = \prod_{r \in R} r^{e(r)}$, where $e(r)$ is some suitable exponent.

3. [Generating implied division polynomials]

Generate $\bar{\psi}_m$ by Algorithm 3.3.1.

4. [Pretesting]

Compute $g = \gcd(s^i - 1, N)$ for $0 \le i < \frac{m^2}{4}$. If $1 < g < N$, declare success and the factor $p = g$; If $g = N$, go to Step 1; If $g = 1$, go to Step 5.

5. [Polynomial evaluation]

Evaluate $\bar{\psi}_m$ *at* $s^0, s^1, \ldots, s^{\frac{m^2}{4}-1}$ *by Algorithm 4.4.1. Denote* $g_i \equiv \bar{\psi}_m(s^i)$ $(\mathrm{mod}\ N)$ *for* $0 \le i < \frac{m^2}{4}$*.*

6. [Locating p]

Compute $g = \gcd(\prod_{i=0}^{\frac{m^2}{4}-1} g_i, N)$*. If* $1 < g < N$*, declare success and the factor* $p = g$*; If* $g = 1$*, go to Step 2 and increase* B*; If* $g = N$*, split the product* $\prod_{i=0}^{\frac{m^2}{4}-1} g_i$ *into two products each with half of the factors. Repeat Step 6 with the two shorter products as input. The splitting procedure is implemented the way as in Step 7.*

7. [Binary Splitting]

Split $\prod_{i=0}^{\frac{m^2}{4}-1} g_i$ *by* $f_1 = \prod_{i=0}^{\frac{m^2}{8}-1} g_i$ *and* $f_2 = \prod_{i=\frac{m^2}{8}}^{\frac{m^2}{4}-1} g_i$*. If* $1 < \gcd(f_i, N) < N$ *for* $i = 1$ *or* 2*, call success and the factor* $p = \gcd(f_i, N)$ *for the respective* i*. If* $\gcd(f_1, N)$ *(or* $\gcd(f_2, N)$*)=* N*, split* f_1 *(or* f_2*) and rename* $f_1 = \prod_{i=0}^{\frac{m^2}{16}-1} g_i$ *(or* $f_1 = \prod_{i=\frac{m^2}{8}}^{\frac{3m^2}{16}-1} g_i$*) and* $f_2 = \prod_{i=\frac{m^2}{16}}^{\frac{m^2}{8}-1} g_i$ *(or* $f_2 = \prod_{i=\frac{3m^2}{16}}^{\frac{m^2}{4}-1} g_i$*). Then repeat Step 7. In case further splitting is necessary, split the polynomial to two new polynomials with half of the degree of the previous polynomial similar as the argument stated above.*

The following theorem states that under certain circumstances, the algorithm will successfully find a non-trivial divisor of N.

Theorem 4.1.1 *Let* $N, B \in \mathbb{Z}$*,* $R = \{r \text{ is a prime} : r \le B\}$*,* $e(r) = \lceil \frac{\ln B}{\ln r} \rceil$*,* $m = \prod_{r \in R} r^{e(r)}$ *and* $x, y \in \mathbb{Z}/N\mathbb{Z}$*. Let* d *be a small prime number, and* $a_i = d^i \bmod N$*,* $b_i = y^2 - x^3 - a_i x \bmod N$ *for* $i = 0, 1, \ldots, \frac{m^2}{4} - 1$*. Then* $P = (x, y) \in E_{a_i, b_i}[N]$*. Suppose that* N *has prime divisors* p *and* q *satisfying the following conditions*

1. $\gcd(6(4a_i^3 + 27b_i^2), N) = 1$ *for* $i = 0, 1, \ldots, \frac{m^2}{4} - 1$*;*

2. $\exists a_i$ *such that each prime number* r *dividing* $\#E_{a_i, b_i}[p]$ *satisfies* $r \le B$ *so that* $r \mid m$*;*

3. for the same a_i as in Condition 2, m is not divisible by the largest prime number dividing the order of $P_q^{a_i,b_i}$, which is the point P considered in $E_{a_i,b_i}[q]$.

Then Algorithm 4.1.1 is successful in finding a non-trivial divisor of N.

Proof From *2*, if B is chosen sufficiently large, we can get $(m \cdot P)_p^{a_i,b_i} = \infty_p$. By (3.8), $\bar{\psi}_m(a_i) \equiv 0 \pmod{p}$.

From *3*, $(m \cdot P)_q^{a_i,b_i} \neq \infty_q$. By (3.8), $\bar{\psi}_m(a_i) \not\equiv 0 \pmod{q}$. Therefore, p divides $\gcd(\bar{\psi}_m(a_i), N)$. If $\gcd(\bar{\psi}_m(a_i), N) \neq N$, the algorithm succeeds in finding a divisor not less than p of N. If $\gcd(\bar{\psi}_m(a_i), N) = N$, since *2* and *3* are for the same a_i, after sufficient truncation, a non-trivial divisor not less than p of N will be found. ■

In the algorithm, we have $m^2/4$ different elliptic curves. It is important to study the relationships between those curves before we use them to implement the factoring algorithm. There is a type of map between elliptic curves, which is called isogeny.

Definition 4.1.1 *An **isogeny** between two elliptic curves E_1 and E_2 over a field K is a homomorphism from $E_1(\overline{K})$ to $E_2(\overline{K})$ that is given by rational functions.*

By the above definition, we can see that $\zeta(P + Q) = \zeta(P) + \zeta(Q)$ and $\zeta(m \cdot P) = m \cdot \zeta(P)$ for all P, Q in E_1.

There is an important result about the structure of elliptic curves.

Lemma 4.1.1 *Let n_1 and n_2 be odd integers with $\gcd(n_1, n_2) = 1$. Let E be an elliptic curve defined over $\mathbb{Z}_{n_1 n_2}$. Then there is a group isomorphism*

$$E[n_1 n_2] \simeq E[n_1] \oplus E[n_2].$$

For more details of the proof, please refer to Section 2.10 of [24].

Theorem 4.1.2 *If p is a prime divisor of N and there exist i, j with $0 \leq i < j < m^2/4$ such that $E_{a_i,b_i}[p]$ is isogenous to $E_{a_j,b_j}[p]$ and $(m \cdot P)_p^{a_i,b_i} = \infty_p^{a_i,b_i}$. Assume also that there exists an isogeny ζ between $E_{a_i,b_i}[p]$ and $E_{a_j,b_j}[p]$ such that $P = (x, y)$ is a fixed point under ζ, that is $\zeta(P) = P$. Furthermore, if $E_{a_i,b_i}[N]$ is not isogenous to $E_{a_j,b_j}[N]$, then Algorithm 4.1.1 is successful in finding a non-trivial divisor of N.*

Proof Apply ζ to both sides of $(m \cdot P)_p^{a_i,b_i} = \infty_p^{a_i,b_i}$, we can get

$$(\zeta(m \cdot P))_p^{a_j,b_j} = (\zeta(\infty_p^{a_i,b_i}))_p^{a_j,b_j} = \infty_p^{a_j,b_j}.$$

The second equality is because ζ is an isogeny. Then $(\zeta(m \cdot P))_p^{a_j,b_j} = (m \cdot \zeta(P))_p^{a_j,b_j}$. Notice that P is a fixed point under ζ. Therefore,

$$(m \cdot P)_p^{a_j,b_j} = (m \cdot \zeta(P))_p^{a_j,b_j} = \infty_p^{a_j,b_j}. \tag{4.1}$$

If $(m \cdot P)_N^{a_i,b_i} \neq \infty_N^{a_i,b_i}$, then by (3.8), p divides $\bar{\psi}_m(a_i)$ but N does not divide $\bar{\psi}_m(a_i)$. So Algorithm 4.1.1 succeeds in finding a non-trivial divisor of N.

Now assume that $(m \cdot P)_N^{a_i,b_i} = \infty_N^{a_i,b_i}$. By Lemma 4.1.1,

$$E_{a_i,b_i}[N] \simeq E_{a_i,b_i}[p] \oplus E_{a_i,b_i}[N/p]$$

$$E_{a_j,b_j}[N] \simeq E_{a_j,b_j}[p] \oplus E_{a_j,b_j}[N/p].$$

Let $P_1 \in E_{a_i,b_i}[p]$ and $P_2 \in E_{a_i,b_i}[N/p]$. Expand ζ to ζ^*

$$\begin{aligned} \zeta^* : E_{a_i,b_i}[N] &\longrightarrow E_{a_j,b_j}[N] \\ (P_1, P_2) &\longmapsto (\zeta(P_1), \zeta(P_2)). \end{aligned}$$

We have

$$(m \cdot P)_N^{a_j,b_j} = m \cdot \zeta^*(P)_N^{a_j,b_j} = (\zeta^*(m \cdot P))_N^{a_j,b_j} = \zeta^*(\infty_N^{a_i,b_i}) \neq \infty_N^{a_j,b_j}.$$

The last equality is true because $E_{a_i,b_i}[N]$ is not isogenous to $E_{a_j,b_j}[N]$. Therefore, N does not divide $\bar{\psi}_m(a_j)$. By (4.1), we can see that p divides $\bar{\psi}_m(a_j)$. So Algorithm 4.1.1 is successful in finding a non-trivial divisor of N. ■

From Theorem 4.1.1, for some chosen B, if one of the $m^2/4$ curves has B-smooth order over $\mathbb{F}_p$ and condition *3* in Theorem 4.1.1 is satisfied, then the algorithm is successful. Condition *3* can be fulfilled by choosing B properly. So in order to study the probability of success, we need to know the probability of a number to be B-smooth and some properties of Dickman's function $\rho(t)$. For definition and notation, please refer to Section 2.3. Here we denote $\lambda(x, x^{1/t}) = \lambda_1(x, x^{1/t})$.

From results of Knuth and Trabb Pardo [11], we know that

$$\lambda(x, x^{1/t}) = x\rho(t) + \frac{x\sigma(t)}{\ln x} + O(\frac{x}{\ln^2 x}) \tag{4.2}$$

where

$$\sigma(t) = (1-\gamma)\rho(t-1)$$

and $\gamma = 0.5772\ldots$ is Euler's constant. It was shown in Knuth [11] that if ρ is a good approximation to the smoothness distribution, then σ is a good approximation to the semismoothness distribution.

We can use this to deduce the probability of success of Algorithm 4.1.1. But we can get a more insightful view from the following result by Canfield, Erdös and Pomerance [4].

Lemma 4.1.2 *If* $\epsilon > 0$ *is arbitrary and* $3 \leq t \leq (1-\epsilon)\ln x/\ln\ln x$, *then*

$$\lambda(x, x^{1/t}) = x \cdot \exp\left[-t\left(\ln t + \ln\ln t - 1 + \frac{\ln\ln t - 1}{\ln t} + E(x,t)\right)\right]$$

where

$$|E(x,t)| \leq c_\epsilon \frac{(\ln\ln t)^2}{\ln^2 t}$$

with c_ϵ *being a constant that depends only on the choice of* ϵ.

Now let's state and prove one of the most important theorems in this thesis. It is about the probability of Algorithm 4.1.1 to be successful.

Theorem 4.1.3 *Let* N, p, B *and* m *be defined as in Algorithm 4.1.1 and* $B = p^{1/t}$ *so that* $t = \ln p/\ln B$. *Then the success probability of Algorithm 4.1.1 is* $1-(1-\varepsilon(t))^{m^2/4}$ *where*

$$\varepsilon(t) = t^{-t} \cdot \eta(t) \tag{4.3}$$

with

$$\eta(t) \approx e^{(1-\ln\ln t)(t+t/\ln t)}. \tag{4.4}$$

Proof By Theorem 4.1.1, we can see that the probability of Algorithm 4.1.1 to succeed is the same as the probability that at least one elliptic curve has its order to be B-smooth.

There are $4\lfloor\sqrt{p}\rfloor+1$ integers in the interval $(p+1-2\sqrt{p}, p+1+2\sqrt{p})$. Now let's count the number of integers which are B-smooth in this interval. By the definition of $\lambda(x)$, we can get that this number is $\lambda(p+1+\lfloor 2\sqrt{p}\rfloor, B)-\lambda(p+1-\lfloor 2\sqrt{p}\rfloor-1, B)$. Here we use the approximation $(p+1+\lfloor 2\sqrt{p}\rfloor)^{1/t} \approx (p+1-\lfloor 2\sqrt{p}\rfloor-1)^{1/t} \approx p^{1/t} = B$. By Lemma 4.1.2, we have

$$\begin{aligned}
&\lambda(p+1+\lfloor 2\sqrt{p}\rfloor, B)-\lambda(p+1-\lfloor 2\sqrt{p}\rfloor-1, B)\\
&= [(p+1+\lfloor 2\sqrt{p}\rfloor)-(p+1-\lfloor 2\sqrt{p}\rfloor-1)]\cdot e^{-t\left(\ln t+\ln\ln t-1+\frac{\ln\ln t-1}{\ln t}\right)}\cdot e^{E(x,t)}\\
&\approx (4\lfloor\sqrt{p}\rfloor+1)\cdot t^{-t}\cdot e^{-t(\ln\ln t-1)(1+1/\ln t)}\\
&= (4\lfloor\sqrt{p}\rfloor+1)\varepsilon(t).
\end{aligned}$$

Therefore the probability that a random integer chosen from $(p+1-2\sqrt{p}, p+1+2\sqrt{p})$ is B-smooth is $(4\lfloor\sqrt{p}\rfloor+1)\varepsilon(t)/(4\lfloor\sqrt{p}\rfloor+1)=\varepsilon(t)$.

Now we assume that the distribution of orders over $\mathbb{F}_p$ of different elliptic curves for $a=s^0,\dots,s^{m^2/4-1}$ are independent. If not, we can use another way of choosing a such as choosing $m^2/4$ a values randomly. Then we will use a different algorithm for polynomial evaluation with the same type of complexity. See Section 4.4 for more information.

Since we are assuming that different elliptic curves have independent probabilities of having their orders over $\mathbb{F}_p$ being B-smooth, the probability that at least one curve has B-smooth order over $\mathbb{F}_p$, which is the probability that Algorithm 4.1.1 is successful, is

$$\begin{aligned}
&1-\text{the probability that any curve has non-}B\text{-smooth order over }\mathbb{F}_p\\
&=1-(1-\varepsilon(t))^{m^2/4}.
\end{aligned}$$

■

Let's put some more consideration to $\varepsilon(t)$ before we move on. Notice

$$f(t)=\ln(\varepsilon(t))=-t\ln t+\ln\eta(t)=-t\ln t+t-t\ln\ln t+\frac{t}{\ln t}-\frac{t\ln\ln t}{\ln t}.$$

We have

$$f'(t) = -\ln t - \ln\ln t - \frac{2}{\ln^2 t} - \frac{\ln\ln t}{\ln t} + \frac{\ln\ln t}{\ln^2 t}$$
$$f''(t) = \frac{1}{t}\left(\frac{5}{\ln^3 t} - \frac{1}{\ln^2 t} - 1 - \frac{2\ln\ln t}{\ln^3 t} + \frac{\ln\ln t}{\ln^2 t} - \frac{1}{\ln t}\right).$$

Let $v = \ln t$, then

$$f'(t) = g_1(v) = -v - \ln v - \frac{2}{v^2} - \frac{\ln v}{v} + \frac{\ln v}{v^2}$$
$$f''(t) = g_2(v) = e^{-v}\left(\frac{5}{v^3} - \frac{2\ln v}{v^3} - \frac{1}{v^2} + \frac{\ln v}{v^2} - \frac{1}{v} - 1\right).$$

Notice

$$\begin{aligned} g_1(v) &= -v - \frac{2}{v^2} - \ln v(1 + \frac{1}{v} - \frac{1}{v^2}) \\ &= -v - \frac{2}{v^2} - \ln v(1 + \frac{1+\sqrt{5}}{2v})(1 + \frac{1-\sqrt{5}}{2v}). \end{aligned}$$

If $v > 1$ and $1 + \frac{1-\sqrt{5}}{2v} > 0$, i.e. $v > \frac{\sqrt{5}-1}{2} \approx 0.6180 \Leftrightarrow t > 1.8553$, $g_1(v) < 0$. In the algorithm, t is normally greater than 1.8553. Therefore, $f'(t) < 0$, so that $f(t)$ is decreasing as t increases. Now set $g_2(v) = 0$ to find the inflection point candidates of $f(t)$. We can solve the equation numerically. By using Matlab and checking the change of signs of $g_2(v)$ while v passes the candidate points from left to right, we can see that $t = 3.5177$ is the inflection point. Therefore, $f(t)$ is decreasing and concave up for $t \in (1.8553, 3.5177)$ and concave down for $t \in (3.5177, \infty)$. In the algorithm, t is normally greater than 3.5177. So we can conclude that while B increases, the probability of success of Algorithm 4.1.1 will increases. Now let's see whether we can find a specific range of B values such that the success probability can have a lower bound. The following corollary of Theorem 4.1.3 answers this question.

Corollary 4.1.1 *Let N, p, B and m be defined as in Algorithm 4.1.1 and $t = \ln p/\ln B$. Then the probability of Algorithm 4.1.1 to be successful is at least* $1 - \exp((-m^2(t^{-t})^{1+\ln\ln t/\ln t - 1/\ln^2 t})/4)$.

Proof Consider $\varepsilon(t)$. We can get

$$\begin{aligned} \ln\varepsilon(t) = -t\ln t + \ln\eta(t) &= -t\ln t + t - t\ln\ln t + \frac{t}{\ln t} - \frac{t\ln\ln t}{\ln t} \\ &= -t\ln t(1 + \frac{t\ln\ln t}{\ln t} - \frac{1}{\ln^2 t}) + t(1 - \frac{\ln\ln t}{\ln t}). \end{aligned}$$

Since $1 - \frac{\ln\ln t}{\ln t} = \frac{\ln t - \ln\ln t}{\ln t} > 0$ and $\frac{1}{\ln t} > 0$ for $t > 1$. we have $\ln \varepsilon(t) > -t\ln t(1 + \frac{\ln\ln t}{\ln t} - \frac{1}{\ln^2 t})$. So that

$$\begin{aligned}\varepsilon(t) >& e^{-t\ln t(1+\ln\ln t/\ln t - 1/\ln^2 t)} \\ =& (t^{-t})^{1+\ln\ln t/\ln t - 1/\ln^2 t}.\end{aligned}$$

By Taylor expansion and applying the above inequality, we have

$$(1-\varepsilon(t))^{m^2/4} \le e^{-m^2\varepsilon(t)/4} < \exp((-m^2(t^{-t})^{1+\ln\ln t/\ln t - 1/\ln^2 t})/4).$$

By Theorem 4.1.3, the probability of success of Algorithm 4.1.1 is greater than $1 - \exp((-m^2(t^{-t})^{1+\ln\ln t/\ln t-1/\ln^2 t})/4)$. This concludes the proof. ■

In the following sections, we will analyze Algorithm 4.1.1 in some more details. We will go over the algorithms to do FFT and polynomial evaluations for both geometric progression and random chosen values. Then at the end of this chapter, we will come back to Algorithm 4.1.1 by analyzing its complexity and proposing a more efficient version for some specific type of p values.

4.2 The Fast Fourier Transform and its Applications

The Fast Fourier Transform (FFT) is a fast algorithm to compute the discrete Fourier transform (DFT) of a vector. It is vastly used in both science and engineering. DFT can be defined over both the complex numbers and a commutative ring R.

Definition 4.2.1 *An element $\omega \in R$ is called a* ***principal D-th root of unity*** *if*

1. $\omega \neq 1$.
2. $\omega^D = 1$.
3. $\sum_{k=0}^{D-1} \omega^{ke} = 0$, *for* $1 \le e < D$.

Definition 4.2.2 *Let $a = (a_0, a_1, \ldots, a_{D-1})$ be a vector with elements from a commutative ring R, and suppose D has a multiplicative inverse in R. Let ω be a principal D-th root of unity. The vector $DFT(a) = (a_0^*, a_1^*, \ldots, a_{D-1}^*)$, where $a_i^* =$*

*$\sum_{k=0}^{D-1} a_k\omega^{ik}$ for $0 \le i < D$, is called the **discrete Fourier transform of a**. The vector $DFT^{-1}(a) = (a'_0, a'_1, \ldots, a'_{D-1})$, where $a'_i = D^{-1}\sum_{k=0}^{D-1} a_k\omega^{-ik}$ for $0 \le i < D$, is called the **inverse discrete Fourier transform of a**.*

It's not hard to prove that $DFT(DFT^{-1}(a)) = DFT^{-1}(DFT(a)) = a$. The Fourier transform and inverse Fourier transform of a length-D vector can be computed in $O(D^2)$ time. However, we have a much faster algorithm with time $O(D \ln D)$, which is called the fast Fourier transform (FFT). If we consider the polynomial $f(x) = \sum_{i=0}^{D-1} a_i x^i$, then finding $DFT(a)$ is equivalent to evaluating $f(x)$ at D points $\omega^0, \omega^1, \ldots, \omega^{D-1}$. To evaluate $f(x)$ at some given point ω^i, we can divide $f(x)$ by $x - \omega^i$. The remainder is $f(\omega^i)$.

Instead of dividing $f(x)$ by each $x - \omega^i$, FFT does the division after multiplying some $x - \omega^i$ together. Notice that after multiplying the $x - \omega^i$ together in pairs, we will get $D/2$ polynomials. Then we do the same multiplication to these $D/2$ polynomials, and so on. Finally we can get two polynomials, q_1 and q_2. They are both with degree $D/2$. Now we divide f by q_1 and by q_2. Denote the remainder by r_1 and r_2 respectively. If $(x - \omega^i) \mid q_1$, then it is not hard to prove that $f \equiv r_1 \pmod{x - \omega^i}$. Similar result is true for q_2. Now dividing a polynomial of degree D by a linear term is reduced to dividing a polynomial of degree $D/2 - 1$ by the same term.

The principal D-th root of unity ω has a property that can make the multiplication of $x - \omega^i$ have a good form. Let $d_0, d_1, \ldots, d_{D-1}$ be a permutation of $\omega^0, \omega^1, \ldots, \omega^{D-1}$. Consider the case of $D = 2^\alpha$. If D is not a power of 2, we can add some higher degree term to the polynomial to make D a power of 2. For integer $0 \le \beta \le \alpha$ and an integer multiple of 2^β s with $0 \le s < D$, we define a polynomial

$$q_{s,\beta} = \prod_{i=s}^{s+2^\beta-1} (x - d_i).$$

All these $q_{s,\beta}$'s form a binary tree, with $q_{0,0} = x - d_0, q_{1,0} = x - d_1, \ldots, q_{D-1,0} = x - d_{D-1}$ as the leaf nodes and $q_{0,\alpha} = \prod_{i=0}^{D-1}(x - d_i)$ as the root. By simple calculation we can prove that

$$q_{s,\beta} = q_{s,\beta-1} \cdot q_{s+2^{\beta-1},\beta-1}. \tag{4.5}$$

This result tells us that each node is the product of its two children. Constructing these polynomials is the way we multiply the $x - \omega^i$ together in pairs.

Now we need to know how to rearrange $\omega^0, \omega^1, \ldots, \omega^{D-1}$ to get $d_0, d_1, \ldots, d_{D-1}$ with some desirable property. Let $b_0 b_1 \cdots b_{\alpha-1}$ be the binary representation of the integer $0 \le i < D$. Define $\widetilde{i}$ to be the integer whose binary representation is $b_{\alpha-1} b_{\alpha-2} \cdots b_0$. Let $d_i = \omega^{\widetilde{i}}$, then by induction, we can prove that

$$q_{s,\beta} = x^{2^\beta} - \omega^{\widetilde{s/2^\beta}}. \tag{4.6}$$

We need to find the remainder of $f(x)$ when divided by $q_{s,0}$ for each $0 \le s < D$. In FFT, we compute the remainder $r_{s,\beta}$ of $f(x)$ when divided by each $q_{s,\beta}$. We run the loop from $\beta = \alpha - 1$ to $\beta = 0$. For $\beta = 0$, s will traverse all through $0, 1, \ldots, D-1$. From (4.5), we can prove that $r_{s,\beta-1}$ is the same as the remainder of $r_{s,\beta}$ when divided by $q_{s,\beta-1}$ and $r_{s+2^{\beta-1},\beta-1}$ is the same as the remainder of $r_{s,\beta}$ when divided by $q_{s+2^{\beta-1},\beta-1}$.

The degree of $r_{s,\beta}$ is $2^\beta - 1$. By (4.6), $q_{s,\beta-1}$ and $q_{s+2^{\beta-1},\beta-1}$ are both binomials $x^{2^{\beta-1}} - c$ with degree $2^{\beta-1}$ and c be the corresponding constant. Let $r_{s,\beta} = \sum_{i=0}^{2^\beta-1} r_i x^i$. Then we have

$$\begin{aligned} r_{s,\beta} &= \sum_{i=0}^{2^{\beta-1}-1} r_{i+2^{\beta-1}} x^{i+2^{\beta-1}} + \sum_{i=0}^{2^{\beta-1}-1} r_i x^i \\ &= \left(\sum_{i=0}^{2^{\beta-1}-1} r_{i+2^{\beta-1}} x^i \right) (x^{2^{\beta-1}} - c) + \sum_{i=0}^{2^{\beta-1}-1} c r_{i+2^{\beta-1}} x^i + \sum_{i=0}^{2^{\beta-1}-1} r_i x^i \\ &= \left(\sum_{i=0}^{2^{\beta-1}-1} r_{i+2^{\beta-1}} x^i \right) (x^{2^{\beta-1}} - c) + \sum_{i=0}^{2^{\beta-1}-1} (r_i + c r_{i+2^{\beta-1}}) x^i. \end{aligned}$$

So the remainder of $r_{s,\beta}$ when divided by $x^{2^{\beta-1}} - c$ is $\sum_{i=0}^{2^{\beta-1}-1} (r_i + c r_{i+2^{\beta-1}}) x^i$.

From the above analysis, we can get the following algorithm to implement FFT. For more details, please refer to Section 7.2 of [1]. The algorithm is for $D = 2^\alpha$ for some integer α, but it works well for other D values just by adding sufficiently many zeros after a_{D-1} in the vector a.

Algorithm 4.2.1 *(Finding a discrete Fourier transform of a vector a by FFT) Given a vector $a = (a_0, a_1, \ldots, a_{D-1})$ with $D = 2^\alpha$ for some integer α, the algorithm finds the discrete Fourier transform of a $DFT(a) = (a_0^*, a_1^*, \ldots, a_{D-1}^*)$, where a_i^* for $0 \le i < D$ is defined as in Definition 4.2.2.*

```
1. [Initialization]
   Set $r_{0,\alpha} = \sum_{i=0}^{D-1} a_i x^i$.
2. [Outer loop]
   for ($\beta = \alpha - 1$; $\beta \ge 0$; $\beta --$){
3. [Inner loop] for ($s = 0$; $s < D$; $s = s + 2^{\beta+1}$){
   Compute the remainders $r_{s,\beta+1} = \sum_{i=0}^{2^{\beta+1}-1} a_i x^i$;
   Generate $\widetilde{s/2^\beta}$;
   Define $r_{s,\beta} = \sum_{i=0}^{2^\beta-1}(a_i + \omega^{\widetilde{s/2^\beta}} a_{i+2^\beta})x^i$;
   Define $r_{s+2^\beta,\beta} = \sum_{i=0}^{2^\beta-1}(a_i + \omega^{\widetilde{s/2^\beta}+D/2} a_{i+2^\beta})x^i$;}
   }
4. [finalization]
   Set $a_{\tilde{i}}^* = r_{i,0}$ for $0 \le i < D$.
```

By Definition 4.2.2, if we replace ω by ω^{-1} in step *3* and divide $a_{\tilde{i}}^*$ by D in step *4*, we can get the algorithm to implement inverse FFT. We can to prove that the time complexities of Algorithm 4.2.1 and the corresponding algorithm for inverse FFT are both $O(D \ln D)$.

There are different ways to implement FFT. We can do it recursively, by bit operations. See Crandall and Pomerance [6] for more details.

4.3 Cyclic Convolutions

We need to evaluate the division polynomial at many points. A good way to achieve this is to use the cyclic convolution.

Definition 4.3.1 *Let $x = (x_0, x_1, \ldots, x_{D-1})$ and $y = (y_0, y_1, \ldots, y_{D-1})$ be two vectors with D elements. The* ***cyclic convolution*** *$x \otimes y$ is defined to be a vector $z = (z_0, z_1, \ldots, z_{D-1})$ with D elements where*

$$z_k = \sum_{i+j \equiv k \pmod{D}} x_i y_j \qquad (0 \le k \le D-1).$$

If we consider vectors x, y as the coefficients of two $D-1$-degree polynomials f, g with respect to the same variable t, we can prove that $z = x \otimes y$ gives the coefficients of the polynomial $h = fg \pmod{(t^D - 1)}$.

If we define an operation $\star$ by $x \star y = (x_0y_0, x_1y_1, \ldots, x_{D-1}y_{D-1})$, we have

$$x \otimes y = DFT^{-1}(DFT(x) \star DFT(y)). \tag{4.7}$$

Therefore cyclic convolution can be done by 2 Fourier transforms and 1 inverse Fourier transform. So that the complexity of cyclic convolution is $O(D \ln D)$.

4.4 Polynomial Evaluation

In Algorithm 4.1.1, we need to evaluate the m-th implied division polynomial at $m^2/4$ points. Crandall and Pomerance [6] (Section 9.6.3) gives several methods for different cases. In this thesis, we will use the following algorithm.

Algorithm 4.4.1 *(Evaluating a polynomial at several values with geometric progression) Given a polynomial $f(x) = \sum_{i=0}^{D-1} a_i x^i$ with degree $D-1$ and a set of values $\{s^0, s^1, \ldots, s^{D-1}\}$ with s being some small prime number, the algorithm finds D values $\{f(s^0), f(s^1), \ldots, f(s^{D-1})\}$.*

```
1. [Initialization]
   Choose the smallest value $D^* = 2^k$ such that $D^* \ge 2D$;
```

```
    Redefine $a_i$ by $a_i = a_i s^{i(i+1)/2}$ for $0 \le i < D$;
    Zero-pad $a = (a_i)_{0 \le i < D^*}$ so that $a$ is a vector with length $D^*$;
    Define another vector $b = (s^{(D^*/2-i-1)(i/2-D^*/4)})$ for $0 \le i < D^*$.
2. [Cyclic Convolution]
    Let $c = a \otimes b$.
3. [Finalization]
    Return $(f(s^i)) = (s^{i(i-1)/2} c_{D^*/2+i-1})$ for $0 \le i < D$.
```

Now let's see how the algorithm works.

Theorem 4.4.1 *Algorithm 4.4.1 evaluates $f(x)$ at $\{s^0, \ldots, s^{D-1}\}$ correctly.*

Proof Notice that

$$\frac{i(i+1)}{2} + \frac{j(j-1)}{2} - ij = \frac{(i-j)[(i-j)+1]}{2}.$$

Therefore

$$f(s^j) = \sum_{i=0}^{D^*-1} a_i s^{ij} = s^{j(j-1)/2} \sum_{i=0}^{D^*-1} (a_i s^{i(i+1)/2}) s^{(j-i)(i-j+1)/2}.$$

Now we just need to prove that

$$\sum_{i=0}^{D^*-1} (a_i s^{i(i+1)/2}) s^{(j-i)(i-j+1)/2} = c_{D^*/2+i-1} \tag{4.8}$$

for all $0 \le j < D$. By the definition of cyclic convolution, the right-hand side of (4.8) is

$$\begin{aligned} c_{D^*/2+j-1} &= \sum_{i=0}^{D^*-1} (a_i s^{i(i+1)/2}) b_{D^*/2+j-1-i} \\ &= \sum_{i=0}^{D^*-1} (a_i s^{i(i+1)/2}) s^{(D^*/2-(D^*/2+j-1-i)-1)(D^*/2+j-1-i)/2-D^*/4} \\ &= \sum_{i=0}^{D^*-1} (a_i s^{i(i+1)/2}) s^{(i-j)(j-1-i)/2}. \end{aligned}$$

This is the left-hand side of (4.8). So Algorithm 4.4.1 is correct. ■

For length D vectors, the complexity of FFT is $O(D \ln D)$. So that the complexity of Algorithm 4.4.1 is also $O(D \ln D)$ since the main part of Algorithm 4.4.1 is the convolution which essentially is two times of FFT and one time of inverse FFT .

In the proof of Theorem 4.1.3, we require the probability distribution of successfully factoring N with different elliptic curves to be independent. We assumed the independence for the geometrical progression parameters for simplicity. However, randomly choosing a is a more convincing way. These arguments rely on the following conjecture.

Conjecture 4.4.1 *The probability distribution of Algorithm 4.1.1 to be successful for any two elliptic curves with parameters a and a' is independent as long as the two elliptic curves have different orders over $\mathbb{F}_p$ for the prime factor p of N.*

If a more general way to choose the values is desirable such as choosing them totally randomly, which means we cannot use Algorithm 4.4.1 to implement polynomial evaluation any more, we can still do so with Algorithm 9.6.7 in [6]. Let's demonstrate the rough idea of the algorithm here. Assume that D is a power of 2 and we want to evaluate $f(x)$ with degree $D-1$ at D values $x_0, x_1, \ldots, x_{D-1}$. Define

$$g_1(x) = (x - x_0)(x - x_1) \ldots (x - x_{D/2-1}),$$

$$g_2(x) = (x - x_{D/2})(x - x_{D/2+1}) \ldots (x - x_{D-1}).$$

Divide $f(x)$ by $g_1(x)$ and $g_2(x)$, we can get

$$f(x) = q_1(x)g_1(x) + r_1(x) = q_2(x)g_2(x) + r_2(x).$$

Then $f(t_i) = r_1(t_i)$ for $0 \le i < D/2$ and $f(t_i) = r_2(t_i)$ for $D/2 \le i < D$. Therefore, we reduce the polynomial degree in the evaluation problem to half of the original. By setting a break-over threshold and doing the evaluation recursively, we can implement the following algorithm with a good complexity.

Algorithm 4.4.2 *(Evaluating a polynomial at random values) Given a polynomial $f(x) = \sum_{i=0}^{D-1} a_i x^i$ with degree $D-1$ and a set of values $X = \{x_0, x_1, \ldots, x_{D-1}\}$, the algorithm finds D values $\{f(x_0), f(x_1), \ldots, f(x_{D-1})\}$.*

```
1. [Initialization]
   Choose the smallest value $D^* = 2^k$ such that $D^* \geq 2D$;
   Zero-pad $a = (a_i)_{0 \leq i < D^*}$ so that $a$ is a vector with length $D^*$;
   Zero-pad $X$ at the end so that $X$ has length $D^*$;
   Set threshold $\tau = 4$.

2. [Recursion]
   $eval(f, X)\{$
   Let $d$ be the degree of $f$;
   If $(d \leq \tau)$, return $(f(x_0), f(x_1), \ldots, f(x_{d-1}))$;
   $L = (x_0, x_1, \ldots, x_{d/2-1})$;
   $R = (x_{d/2}, x_{d/2+1}, \ldots, x_{d-1})$;
   $g_1(x) = \prod_{i=0}^{d/2-1}(x - x_i)$;
   $g_2(x) = \prod_{i=d/2}^{d-1}(x - x_i)$;
   $r_1(x) = f(x) \pmod{g_1(x)}$;
   $r_2(x) = f(x) \pmod{g_2(x)}$;
   Return $eval(r_1, L) \cup eval(r_2, R)$;
   $\}$
```

In the algorithm, *eval* is the recursive function and $\cup$ is the concatenation operator. Algorithm 4.4.2 runs a bit longer than Algorithm 4.4.1 for the same D.

Theorem 4.4.2 *Algorithm 4.4.2 correctly evaluation a polynomial $f(x)$ with degree $D-1$ at D random values with time complexity $O(D \ln^2 D)$.*

Proof Let's prove that Algorithm 4.4.1 works first. Without losing generality, let's assume $D = D^* = 2^k$. We will use induction on D.

If $D = 4$, then the algorithm returns $f(x_0), f(x_1), f(x_2), f(x_3)$ directly. Suppose the algorithm works for $D = 2^i$ with $2 < i < k$. Now let's prove that it also works for $D = 2^{i+1}$. When the function *eval* is first entered, $d = D = 2^{i+1}$ and we have

$$L = (x_0, x_1, \ldots, x_{2^i-1}), \quad R = (x_{2^i}, x_{2^i+1}, \ldots, x_{2^{i+1}-1}),$$

$$r_1(x) = f(x) \pmod{g_1(x) = \prod_{j=0}^{2^i-1}(x - x_j)},$$

$$r_2(x) = f(x) \pmod{g_2(x) = \prod_{j=2^i}^{2^{i+1}-1}(x - x_j)}.$$

So when $eval(r_1, L)$ and $eval(r_2, R)$ are entered, they both have $d = 2^i$. By the induction assumption and the argument before the algorithm, we see that $eval(r_1, L)$ and $eval(r_2, R)$ return $f(x_0), f(x_1), \ldots, f(x_{D/2-1})$ and $f(x_{D/2}), f(x_{D/2+1}), \ldots, f(x_{D-1})$ respectively. Together, they return the values of $f(x)$ on those D points. By induction, we see that Algorithm 4.4.2 works.

Now let's prove the complexity to be $O(D \ln^2 D)$. It suffices to assume $D = 2^k$. The steps that take most of the operations are expanding $g_1(x), g_2(x)$ and computing $r_1(x), r_2(x)$. First of all, we prove that expanding $g(x) = \sum_{i=0}^{D-1}(x - x_i)$ takes $O(D \ln^2 D)$ field operations.

We calculate $(x - x_{2i})(x - x_{2i+1})$ for $i = 0, \ldots, 2^{k-1} - 1$. At this first step, the required numbers of additions and multiplications are both $1 * D/2 = 2^{k-1}$ so that the total number of field operations is 2^k. At the second step, there are $2 * 2^2 * D/2^2$ additions and multiplications totally. It's not hard to see that at the i-th step, we need to do $2 * i^2 * D/2^i$ operations. The total number of operations we need to expand $g(x)$ is

$$2\sum_{i=1}^{k} \frac{i^2 D}{2^i} = 2\sum_{i=1}^{k} i^2 2^{k-i} < 2(k^2 \sum_{i=1}^{k-1} 2^i) = 2(k^2(2^k - 1)) < \frac{2}{\ln^2 2}(D \ln^2 D).$$

Therefore it is $O(D \ln^2 D)$.

In Section 8.5 of [1], Aho, Hopcroft, and Ullman prove that the total number of operations to implement polynomial remaindering modulo a polynomial with degree D is $O(D \ln D)$. So the totally operation we need in Algorithm 4.4.2 is

$$\sum_{i=0}^{k-2} \frac{1}{2^i} O(D \ln^2 D) = O(D \ln^2 D).$$

$\blacksquare$

4.5 Complexity of the Factoring Algorithm

After our review of important related issues, now let's return to the main factoring algorithm in this thesis. The following crucial theorem tells the complexity of Algorithm 4.1.1.

Theorem 4.5.1 *If we choose values in a geometric progression to do polynomial evaluation, Algorithm 4.1.1 needs $O(m^2 \ln m)$ arithmetic operations to succeed or report failure.*

Proof By Theorem 3.3.2, the complexity of step 3 in Algorithm 4.1.1 is $O(m^2 \ln m)$. From Theorem 3.2.1, we can see that the degree of the m-th implied division polynomial is at most $m^2/4 - 1$. Therefore by Theorem 4.4.1 it will take $O(m^2 \ln m)$ arithmetic operations to do the polynomial evaluation. So the total number of arithmetic operations Algorithm 4.1.1 needs is $O(m^2 \ln m)$. This finishes the proof. ■

By Theorem 4.4.2, we can get the following result.

Corollary 4.5.1 *In more general case, if we choose random values and implement polynomial evaluation by Algorithm 4.4.2, then Algorithm 4.1.1 requires $O(m^2 \ln^2 m)$ arithmetic operations to succeed or report failure.*

4.6 Optimization of the Factoring Algorithm for Some Specific Prime Numbers

In Schnorr's proposal [19], the memory cost is left arguable. Algorithm 4.1.1 also has the same problem. The integer m is the multiple of the point $P = (x, y)$ while $m^2/4$ is the number of curves being used. It is also the degree of the m-th implied division polynomial. Since every coefficient is stored as an array of integers, if m gets large, which is what the program requires since it is the multiple as well, the memory cost to store the m-th implied division polynomial will get huge and a memory overflow may happen.

Intuitively, even though the multiple and the number of curves do have some relation, it should not be as explicit as shown by m vs $m^2/4$. In this section, we modify Algorithm 4.1.1 so that we could isolate the multiple and the number of curves. Before bringing up this innovative idea, let's demonstrate the advantage of the Algorithm 4.1.1, which is also one of the reasons we developed it. Algorithm 4.1.1, as well as the optimized one in this section, is appealing since it separates the factoring process to two parts: division polynomial generation and polynomial evaluation. The former takes longer than the latter. Therefore, we can apply parallel computation to optimize the algorithm. This will save a lot of time.

Notice that the m-th implied division polynomial has degree at most $m^2/4$. To evaluate $\bar{\psi}_m$ at $m^2/4$ points, we use fast Fourier transform. Therefore, if we can control the degree of the implied division polynomials, we can reduce the memory cost as well as the runtime.

Consider a family $\mathcal{F}_n$ of numbers with the form $s^n \pm 1$ where s is some small prime number such as $2, 3, 5, 7, 11$. These numbers are interesting and their study is important for the development of computational number theory. Two types of numbers known as the Mersenne numbers $M_p = 2^p - 1$ and the Fermat numbers $F_n = 2^{2^n} + 1$ are in $\mathcal{F}_n$. Some good results have been concluded for these numbers and primality testing of them is of great interest.

The factoring algorithm in this thesis can be optimized for $\mathcal{F}_n$. Let $N = s^n \pm 1 \in \mathcal{F}_n$, for some small prime number s, be the number we want to factor. Consider a degree n polynomial $x^n \pm 1$. We can get that

$$(s^i)^n \equiv 1 \pmod{N} \quad \text{for } 0 < i < n \tag{4.9}$$

for $N = s^n - 1$ and

$$(s^i)^n \equiv -1 \pmod{N} \quad \text{for } 0 < i < n \text{ and } i \text{ is odd} \tag{4.10}$$

for $N = s^n + 1$. In the implied division polynomial generating algorithm (Algorithm 3.3.1), if we do every polynomial operation modulo $m(x) = x^n \pm 1$, then the result of

Algorithm 4.1.1 will not change since p divides N and (4.9), (4.10). For the Fermat number $N = 2^{2^n} + 1$, we can make the degree even smaller. Since we evaluate the m-th implied division polynomial modulo $m(x) = x^{2^{n-1}} + 1$ at 2^{n-1} points 2^i for $0 \le i < 2^{n-1}$, we can see that N divides $m(2^i)$ for $0 < i < 2^{n-1}$ and i being odd.

Algorithm 4.6.1 *(Factoring N by ECM with division polynomials with N in some specific form and $\lfloor \frac{m^2}{4n} \rfloor$ curves) Given a positive integer $N = s^n \pm 1$ with some small prime number s, the algorithm finds the factor p of N or stops by asserting a failure to factor N.*

1. [Setting bound]

Choose an appropriate upper bound B, let R be the set of primes less than B. Define $m = \prod_{r \in R} r^{e(r)}$, where $e(r) \ge 0$ is some suitable exponent. Also define a product calculation variable $\pi = 1$ and a variable $\kappa = 0$ to record the number of loops.

2. [Selecting the point]

Choose $0 < x, y < N$.

3. [Generating implied division polynomials]

Generate $\bar{\psi}_m$ by calling modified Algorithm 3.3.1 with doing all the calculations modulo $x^n \pm 1$.

4. [Pretesting]

Compute $g = \gcd(s^i - 1, N)$ for $0 \le i < n$. If $1 < g < N$, call success and the factor $p = g$; If $g = 1$, go to Step 5.

5. [Polynomial evaluation]

Evaluate $\bar{\psi}_m$ at $s^0, s^1, \ldots, s^{n-1}$ by Algorithm 4.4.1. Denote $g_i \equiv \bar{\psi}_m(s^i) \pmod{N}$ for $0 \le i < n$.

6. [Locating p]

Define a temporary variable $\tau = \pi$ *and set* $\pi = \pi \cdot \prod_{i=0}^{n-1} g_i$. *Compute* $g = \gcd(\pi, N)$. *If* $1 < g < N$, *call success and the factor* $p = g$*; If* $g = 1$, *go to Step 7; If* $g = N$, *split* $\prod_{i=0}^{n-1} g_i$, *set* $\pi = \tau$, *and repeat Step 6 with the truncated product as the input value. The splitting procedure is implemented the same way as Step 7 in Algorithm 4.1.1.*

7. [Going back for another loop]

Set $\kappa = \kappa + 1$. *If* $\kappa = \lfloor \frac{m^2}{4n} \rfloor$, *go to Step 8. If* $\kappa < \lfloor \frac{m^2}{4n} \rfloor$, *go back to Step 2 and choose a different pair of* $0 < x, y < N$.

8. [Renewing]

If the program runs within the expect runtime, go back to Step 1 and choose a larger B. *If not, quit and report failure.*

In the algorithm, Step 7 is executed in order to make sure we try $m^2/4$ curves. Algorithm 4.6.1 calls Algorithm 3.3.1 $\lfloor \frac{m^2}{4n} \rfloor$ times. And in each call, the polynomials are all with degree n. By similar argument as the proof of Theorem 3.3.2, the total time for division polynomial generation is $m^2 O(n \ln n \ln m)/(4n) = O(m^2 \ln m \ln n)$. The total time taken by polynomial evaluation is $m^2 O(n \ln n)/(4n) = O(m^2 \ln n)$. Summing them up, we get the following theorem.

Theorem 4.6.1 *Algorithm 4.6.1 will factor* $N \in \mathcal{F}_n$ *with* $O(m^2 \ln m \ln n)$ *arithmetic operations or report failure.*

For large m, $m^2/4$ is very large. In most cases, we don't have to try that many curves. So we can ease the restriction of the number of curves a bit to get the following modified algorithm.

Algorithm 4.6.2 *(Factoring* N *by ECM with division polynomials with* N *in some specific form and* k *curves) Given a positive integer* $N = s^n \pm 1$ *with some small* *prime number s, the algorithm finds the factor p of N or stops by asserting a failure to factor N.*

In Step 1, choose an integer k. All the other parts of Step 1 are the same as Step 1 in Algorithm 4.6.1. Step 2-6 and Step 8 are all the same as the ones in Algorithm 4.6.1, Step 7 is the same as Step 7 in Algorithm 4.6.1 except for changing $\lfloor \frac{m^2}{4n} \rfloor$ to $\lfloor \frac{k}{n} \rfloor$.

By the similar arguments above Theorem 4.6.1 and in the proof of Corollary 4.1.1, we can get the following theorem about the complexity of Algorithm 4.6.2.

Theorem 4.6.2 *Algorithm 4.6.2 will factor $N \in \mathcal{F}_n$ with $O(k \ln m \ln n)$ arithmetic operations by using k elliptic curves or report failure. Let $t = \ln p / \ln B$, then the probability of success is at least $1 - e^{-k(t^{-t})^{1+\ln\ln t/\ln t - 1/\ln^2 t}}$.*

Notice that in Algorithm 4.1.1, 4.6.1 and 4.6.2, s does not have to be a prime. In fact, s can be any number as long as it has an inverse in the arithmetic domain. From the argument in Section 5.2, we can see that if s is not a multiple of any of the prime numbers we used to implement Chinese remainder theorem, then s can be used. Please refer to Section 5.2 for more details. Practically we choose s to be some small number in Algorithm 4.1.1. For Algorithm 4.6.1 and 4.6.2, we choose s to be the same as in $N = s^n \pm 1$.

This optimized algorithm makes the choice of the multiple m not being restricted by the memory cost so that we can use very large m to increase the probability of success without increasing the size of required memory at the same time.

5. Implementation and Results

5.1 Multiple-Precision Arithmetic

While implementing the algorithms in the thesis, most of the numbers are greater than 2^{32}. We need a multiple-precision mechanism to deal with this. I used the Multi-Precise Arithmetic Package written by R. D. Silverman. C. Zhang modified it to support negative multi-precision integers. I modified some of the routines so that they could deal with some special cases such as zero exponent.

In the package, all the multiple-precision integers are stored in an array with some radix such as 2^{30}, with most significant digits to the left and least to the right. The absolute value of first element in the array, denoted by array[0], is the length of the array including array[0]. The sign of it is the sign of the integer.

5.2 Chinese Remainder Theorem and Its Application to the Algorithm

Since our purpose is to factor some large integer N, we can take the remainder modulo N and keep all the numbers in the algorithm less than N.

Theorem 5.2.1 *Let $m_1, \ldots, m_t$ be positive integers with $\gcd(m_i, m_j) = 1$ for any $1 \leq i, j \leq t$ and $M = \prod_{i=1}^{t} m_i$. Let $r_1, \ldots, r_t$ be any t integers. Then the system comprising the k congruences*

$$x \equiv r_i \pmod{m_i}, \quad 1 \leq i \leq t$$

has a unique solution in $[0, M-1]$. Furthermore, this solution is given explicitly by the least nonnegative residue modulo M of

$$\sum_{i=1}^{t} r_i s_i M_i$$

where $M_i = M/m_i$ and $s_i = M_i^{-1} \pmod{m_i}$ for $1 \leq i \leq t$.

The complexity of implementing Chinese remainder theorem is $O(t(\ln M)^3)$. See Section 5.3 of Wagstaff [23] for details.

The factoring algorithm in this thesis does all the computation modulo N. The reason we do this is that the coefficients of the implied division polynomials will get extremely large when the index is getting large and we need to restrict the values of them within some upper bound. This technique is similar with the one we use in Section 4.6 to optimize the algorithm for some integers with specific form.

The routines in the multiple-precision package to implement modular arithmetic are faster and easier if the modulus is a single-precision number rather than a multiple-precision number. So what we did is generating several primes $p_1, \ldots, p_t$ less than 2^{30} such that $\prod_{i=0}^{t} p_i > N$. We implement Algorithm 3.3.1, 4.1.1, 4.2.1, 4.4.1 all modulo p_i for $1 \le i \le t$. Then use Chinese remainder theorem to get the resulting value modulo N.

In Algorithm 4.4.1, the size of the vector in the FFT is D^*. To ensure that the D^*-th root of unity exists modulo p_i, we use the following scheme to choose the p_i.

1. By the size of N, determine the total number of primes that will be needed. Denote it by t.

2. Set $r = \lfloor 2^{29}/D^* \rfloor$.

3. Test the numbers $D^* * (r + j) + 1$ for $j = 0, 1, 2, \ldots,$ for primality until t such primes have been found. Call them $p_1, \ldots, p_t$.

5.3 Results

The algorithm is implemented on a 1.86GHz 1066FBS PC with 4GB DDR2 SDRAM at 533MHz.

Within several minutes Algorithm 4.6.2 succeeded in finding a 13-digit prime $p|N$ where $p = 5625767248687$ and $N = 2^{139} - 1$. The parameters are $B = 146$ and $k \approx 10^4$.

With $B = 719$ and $k \approx 10^5$, a 20-digit prime

$$p = 86656268566282183151$$

is found for $N = 2^{149} - 1$ with 24 minutes.

By applying the optimized method in Section 4.6, $N = 5^{430} + 1$ is successfully factored within 1 week. The prime factor of it is a 47-digit number

$$p = 29523508733582324644807542345334789774261776361.$$

The parameters used are $B = 166810$ and $k \approx 10^9$.

By using $B = 12915$ and $k \approx 10^7$, the algorithm also factors $N = 2^{353} + 1$ in about 20 hours. The prime factor is 37-digit

$$p = 3803909572078746837295094051706948091.$$

A 48-digit prime factor

$$p = 694579497316894264425661243659806371972188318857$$

from $N = 2^{373} + 1$ was found in about 10 days. The parameters are $B = 63096$ and $k \approx 10^{10}$. For this number N, the first prime factor found by the algorithm is 60427. Then I tried the algorithm in Section 4.6 on $N/60427$ and it succeeded. Therefore, we can see that the method in Section 4.6 works as long as the number we want to factor divides some number in $\mathcal{F}_n$.

My goal is to find some prime factor with more than 60 digits. Compare this success with the Web Page `http://wwwmaths.anu.edu.au/~brent/ftp/champs.txt` of Richard Brent. That page lists the all-time champions for factoring with conventional ECM. It represents the greatest successes of hundreds of people using tens of thousands of computers for hundreds of thousands of hours. It shows that the first 48-digit prime discovered by ECM was found less than ten years ago. At that time, factorers had already spent tens of thousand of hours trying ECM. The current ECM record is a 67-digit prime factor.

5.4 Future Research Goals

We propose a new extension of ECM by using division polynomials. This algorithm only has one step. So far if we have a report for failure, we just start the whole algorithm all over for another B. But if we could add a second step to reuse the information we get from the failure, we can save a lot of resources and make the algorithm more efficient.

Another open problem is about the choice of the point. I'm still working on finding a relation between the point (x, y) and the property of the elliptic curve. Does the choice of (x, y) really not matter? Since every elliptic curve we use in the algorithm has to pass through (x, y), it has to have some specific property such that we can have enough elliptic curves with different order modulo p. But how to rationalize this property?

The last problem is the way we choose elliptic curves. That is the way we choose s in Algorithm 4.4.1. If we are dealing with $N \in \mathcal{F}_n$ or N dividing some number in $\mathcal{F}_n$, then the best choice of s is just the same as the base of the specific number in $\mathcal{F}_n$. But what s should we choose if we are trying to factor a random N? By saying random, I mean we have no exact idea of the form of N after applying trial division to eliminate the small factors. At least we should pick s so that the elliptic curves we get have enough different orders modulo p. But we have no idea about p. We can use some methods such as Shanks' [20] baby-step-giant-step algorithm to get some idea about the order. But I think this problem is still open and possesses some research interest.

6. Summary

We have demonstrated the new factoring method by applying division polynomials to ECM. It is a one step algorithm. Choosing a point P and an integer m, the algorithm finds the denominator in the x-coordinate of $m \cdot P$ on many elliptic curves at the same time. This process is implemented by evaluating the m-th implied division polynomial on many values. At the end, it evaluates the greatest common divisor of multiplication of the polynomial evaluation result and the number N which is to be factored.

We also proposed an optimization of the main algorithm, which works for numbers with the form $s^n \pm 1$ with s being some prime number. With this optimized algorithm, we can choose much larger m without wasting much memory and the speed of the algorithm is faster.

With this new method, we can find 40-digit prime factors and several 50-digit prime factors. We are working on 60-digit primes and we expect to find some 70-digit prime factor soon.

LIST OF REFERENCES

LIST OF REFERENCES

[1] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] E. Bach and R. Peralta. Asymptotic Semismoothness Probabilities. *Math. Comp.*, 65:1701-1715, 1996.

[3] R. P. Brent. Some integer factorization algorithms using ellpitic curves. *Research Report CMA-R32-85*, Center for Mathematical Analysis, The Australian National University, September 1985.

[4] E. R. Canfield, P. Erdös and C. Pomerance. On a problem of Oppenheim concerning "Factorisatio Numerorum". *J. Number Theory*, 17:1-28, 1983.

[5] J. Cassels. Diophantine equation with special reference to elliptic curves. *J. London Math. Soc.*, 41:193-291, 1966.

[6] R. Crandall and C. Pomerance. *Prime Numbers: A Computational Perspective.* Springer-Verlag, New York, 2001.

[7] J. A. Davis and D. B. Holdridge. Factorization using the quadratic sieve algorithm. In D. Chaum, editor, *Advances in Cryptology-CRYPTO'83*, pages 103-113, Plenum Press, New York, 1984.

[8] M. Deuring. Die Typen der Multiplikatorenringe elliptischer Funktionenkörper. *Abh. Math. Sem. Hansischen Univ.*, 14:197-272, 1941.

[9] K. Dickman. On the frequency of numbers containing prime factors of a certain relative magnitude. *Ark. Mat., Astronomi och Fysik*, 22A, 10:1-14, 1930.

[10] C. F. Gauss. *Disquisitiones Arithmeticae.* Yale University Press, New Haven, English edition, 1966.

[11] D. Knuth and L. Trabb Pardo. Analysis of a simple factorization algorithm. *Theoret. Comput. Sci.*, 3:321-348, 1976.

[12] A. K. Lenstra and H. W. Lenstra, Jr. *The Development of the Number Field Sieve.* Springer-Verlag, New York, 1993.

[13] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *Ann. of Math.*, 126:649-673, 1987.

[14] G. Marsaglia, A. Zaman, and J. C. W. Marsaglia. Numerical solution of some classical differential-difference equations. *Math. Comp.*, 53:191-201, 1989.

[15] P. Montgomery. *An FFT extension of the elliptic curve method of factorization.* PhD thesis, University of California, Los Angeles, 1992.

[16] P. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Math. Comp.*, 48:243-264, 1987.

[17] J. M. Pollard. Theorems on factorization and primality testing. *Proc. Cambridge Philos. Soc.*, 76:521-528, 1974.

[18] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. A. C. M.*, 21(2):120-126, 1978.

[19] C. P. Schnorr. *FFT-ECM for Finding Small Prime Factors.* Fachbereich Mathematik/Informatik, Universität Frankfurt, Germany. January 15, 2001.

[20] D. Shanks. Class number, a theory of factorization, and genera. In *1969 Number Theory Institute, Stony Brook, N.Y.*, volume 20 of *Proc. Sympos. Pure Math.*, pages 415-440. Amer. Math. Soc., 1971.

[21] J. Silverman. *The Arithmetic of Elliptic Curves.* Springer-Verlag, 1986.

[22] R. D. Silverman and S. S. Wagstaff, Jr. A practical analysis of the elliptic curve factoring algorithm. *Math. Comp.*, 61:445-462, 1993.

[23] S. S. Wagstaff, Jr. *Cryptanalysis of Number Theoretic Ciphers.* Chapman and Hall/CRC, Boca Raton, 2003.

[24] L. C. Washington. *Elliptic Curves: Number Theory and Cryptography.* Chapman and Hall/CRC, Boca Raton, 2003.

[25] C. Zhang. *An extension of the Dickman function and its application.* PhD thesis, Purdue University, June 2002.

VITA

VITA

Zhihong Li, was born on March 8, 1978 in Xingtai, Hebei, P.R.China.

He went to the Mathematics Department of Peking University in 1995, majoring in mathematics. He got his bachelor's degree in July of 1999. He started pursuing his Ph.D. in mathematics at Purdue University in 1999.

He earned his Master of Science degree in Mathematics (December, 2004) during the time he was studying for his Ph.D. at Purdue University.