ATKIN'S ECPP (**E**lliptic **C**urve **P**rimality **P**roving) ALGORITHM

OSMANBEY UZUNKOL

OCTOBER 2004

ATKIN'S ECPP (Elliptic Curve Primality Proving) ALGORITHM

A THESIS SUBMITTED TO
DEPARTMENT OF MATHEMATICS
OF
TECHNICAL UNIVERSITY OF KAISERSLAUTERN

BY

OSMANBEY UZUNKOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF MATHEMATICS

October 2004

ABSTRACT

ATKIN'S ECPP ALGORITHM

Uzunkol, Osmanbey
M.Sc., Department of Mathematics
Supervisor: Prof. Dr. Gerhard Pfister

October 2004, cxxiii pages

In contrast to using a strong generalization of Fermat's theorem, as in Jacobi-sum Test, Goldwasser and Kilian used some results coming from Group Theory in order to prove the primality of a given integer $N \in \mathbf{N}$. They developed an algorithm which uses the group of rational points of elliptic curves over finite fields. Atkin and Morain extended the idea of Goldwasser and Kilian and used the elliptic curves with CM (complex multiplication) to obtain a more efficient algorithm, namely Atkin's ECPP (elliptic curve primality proving) Algorithm. Aim of this thesis is to introduce some primality tests and explain the Atkin's ECPP Algorithm.

Keywords: Cryptography, Algorithms, Algorithmic Number Theory.

ÖZ

Hergün bir yere konmak ne güzel,
Bulanmadan donmadan akmak ne hoş,
Dünle beraber gitti cancağızım!
Ne kadar söz varsa düne ait,
Şimdi yeni şeyler söylemek lazım...

.....**Mevlana Celaleddini'i Rumi**.....

I would like to thank first of all to my supervisor Prof. Dr . Gerhard Pfister for his help before and during this work. Secondly, I would like to thank also Hans Schönemann and Raşit Şimşek for their computer supports in computer algebra system SINGULAR and programming language C++, respectively. At the end, I want to thank to my family, especially my mother, and my darling Şermin Çamdalı, without them this thesis would not have been realized.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Primality Tests

Primality testing, distinguishing prime numbers from composite ones, goes back to very old research of number theory due to Eratosthenes who came up with the first primality testing to efficiently generate the set of prime numbers from 1 to N in $\mathcal{O}(N\ log\ log\ N)$ arithmetic steps by means of the so-called *sieve of Eratosthenes* method in 3^{rd} century BC. Since 17^{th} century, mathematicians have been studying primality testing methods in order to distinguish prime numbers from composites and factor integers. Fermat's little theorem enable us to recognize most of the composite integers. Owing to the improvements in the area of Data security and cryptography, in particular public key cryptography, the importance of finding *big* primes has been dramatically increased. Using the extended ideas of Fermat's theorem, Sollovay and Strassen, in 1977, and Miller and Rabin, in 1980, were developed *probabilistic* primality tests. Although, the answer `prime` in these tests is not always true, even cannot be proven mathematically, these tests especially Miller-Rabin test have been used in public key cryptography to build cryptosystems based on factorization of integers, such as RSA and its variants, and discrete logarithm problem (DLP), such as Diffie-Hellman and ElGamal. Furthermore, the answer `composite` is always true in both of these primality tests. Hence, in some texts they are called *compositeness* tests.

The first *general* purpose primality testing algorithm was designed by Adleman, Pomerence and Rumely [10]. The running time of this algorithm was proved to be $\mathcal{O}((log\ N)^{c.log\ log\ log\ N})$ for some effective $c > 0$. Although, this algorithm is *fast* polynomial, has expected polynomial complexity, the first appearance was

not suited to use in practice. Afterwards this algorithm was made practical and simplified by H. W. Lenstra and Cohen in [32], and then implemented by Cohen and A. K. Lenstra in [14].

1.2 ECPP Algorithms

In 1985, H. W. Lenstra introduced the usage of elliptic curves in factorization of integers. After that Goldwasser and Kilian developed an algorithm with the hope of finding a primality test with the help of groups of rational points of elliptic curves over finite fields. Their algorithm is called ECPP (**e**lliptic **c**urve **p**rimality **p**roving) algorithm, which uses the DOWN-RUN strategy of the elliptic curve analog of $N-1$ primality testing method together with a theoretical algorithms due to Schoof. They showed that under reasonable hypothesis on the distribution of primes in short intervals, the expected running time of ECPP is $\mathcal{O}(log^{12}\ N)$. After the previous success at producing proofs of compositeness, as in Sollovay-Strassen and Miller-Rabin, this algorithm produces short proofs of primality. This proof is called hence *certificate* of primality.

The major difficulty in the ECPP algorithm of Goldwasser & Kilian is to find the size of the group of rational points of elliptic curves by means of the theoretical algorithms due to Schoof. Although, some progress has been made in the direction of making Schoof's algorithm practical by Atkin in [48] and Elkies in [29], Atkin and Morain have found a better idea. They used elliptic curves with complex multiplication, abbreviated by elliptic curves with CM, instead of using randomly chosen elliptic curves. Their algorithm, namely Atkin's ECPP, is very practical and used to prove the primality of the *Titanic* numbers, numbers which have more than 1000-digits, by using weeks of workstation time. Moreover, as for the Goldwasser-Kilian algorithm, it is easy to verify the correctness of the result for second programmer, i. e. it gives a certificate of primality for the prime candidate N which enable to recheck the primality of N much faster.

1.3 About Thesis...

The aim of this thesis is to explain and overview the important primality testing methods and Atkin's ECPP algorithm. It consists of the following chapters:

- We will start with basic facts coming from algebra and number theory. In particular, we will explain imaginary quadratic number number fields, quadratic forms and Cornacchia's algorithms, which are the basic algebraic results used in our primality tests and ECPP algorithm.

- In chapter 3, elliptic curves and their arithmetic related with primality proving will be intensively explained and the necessary background will be covered. In this chapter, we will also see how one can deal with problems coming together with the computations of the groups of rational points of elliptic curves over different fields and in particular over finite fields $\mathbf{F}_q$, where $q = p^r$, $r \in \mathbf{N}$ and p is a prime. Further, we will also shortly overview the curves with CM and their relation between imaginary quadratic number fields and forms.

- We will overview some primality testing methods in chapter 4. Furthermore, we will introduce the so-called Poclingston's theorem and $N-1$ (resp. $N+1$) primality testing method, which is a primality testing algorithm based on the full or partial factorization of $N-1$ (resp. $N+1$), where N is a prime candidate. This method will enable us to introduce the general idea and DOWN-RUN strategy of the ECPP algorithms that we will also give in a general form.

- In chapter 5, we will explain our ECPP algorithms of Goldwasser-Kilian and Atkin, respectively. We will overview the approaches of dealing with *point counting problem* of elliptic curves over finite fields, such as the theoretical algorithm of Schoof used in Goldwasser-Kilian ECPP algorithm and CM-method in our Atkin's ECPP algorithm. Moreover, we will see different methods and approaches to improve and optimize the Atkin's algorithm at the end of the chapter.

- The running time analysis of both Goldwasser-Kilian and Atkin will be summarized in chapter 6.

- At the end, we will give implementations of some primality testing methods and some parts of Atkin's ECPP algorithm together with the examples. The computer package LiDiA was used to implement and give examples of these algorithms in the programming language *C++*. Some of the number theoretical functions and primality tests were implemented in computer algebra system *SINGULAR*, too.

chapter 2

Some results from Algebra & Number Theory

In this chapter, we will briefly explain the necessary theory and background coming together with elementary number theory, quadratic forms and imaginary quadratic number fields. All subjects have enormous literature. We begin with some arithmetical results which are basics of our discussions later on. At the first, basic arithmetical tasks related to prime numbers will be introduced. We will give the Fermat's little theorem which will be a basic theoretical result in our all primality testing and proving algorithms. Secondly we will review the preliminaries of algebra and algebraic number theory which are related to our $N-1$ (resp. $N+1$) and ECPP Algorithms. Especially the quadratic forms will be the basic analogous number-theoretic result of our ECPP Algorithms.

2.1 Introductory arithmetics

We will review in this section some arithmetical properties coming from Algebra which are necessary to develop and define our theoretical results for prime numbers.

Definition 2.1. Let $\mathcal{R}$ be a ring and $a, b \in \mathcal{R}$. Then b is called divisible by a or a divides b if there exists $c \in \mathcal{R}$ with $b = a.c$.

Remark 2.1. $a \mid b$ (if a divides b), otherwise $a \nmid b$ (if a does not divide b). Furthermore, we have; $0 \mid a \Leftrightarrow a = 0$.

Definition 2.2. $a \in \mathcal{R}$ is called an *identity* if $a \mid 1$, i. e. $\exists c \in \mathcal{R}$ such that $1 = a.c$. So by definition c is also a an *identity* and $c = a^{-1}$ is then the multiplicative inverse of a in $\mathcal{R}$.

Definition 2.3. $a, b \in \mathcal{R}$ are called *associate* if $a \mid b$ and $b \mid a$ simultaneously.

Corollary 2.1. $(\mathcal{R}^*, .)$ is an abelian group.

Proof: See reference [17].

Definition 2.4. Let $a \in \mathcal{R}$, $a \neq 0$ and a is not an identity element. Then a is called *irreducible* if from the decomposition $a = b.c$, $b, c \in \mathcal{R}$ we get $b \in \mathcal{R}^*$ or $c \in \mathcal{R}^*$. If a is not irreducible, then it is called *composite*.

Definition 2.5. Let $p \in \mathcal{R}$, $p \neq 0$ and p is not an identity element. Then p is called *prime element* or just *prime* if from $p \mid b.c$ with $b, c \in \mathcal{R}$ we get $p \mid b$ or $p \mid c$.

Remark 2.2. In general *prime* $\neq$ *irreducible*

Question: When we have equality?

Proposition 2.1. If $\mathcal{R}$ is an integral domain, then each prime element $a \in \mathcal{R}$ is irreducible.

Proof: See reference [17].

But again for a general integral domain the inverse of the above proposition is false (irreducible $\not\Rightarrow$ prime). In fact, we need an analog algebraic structure which have a division property like integers. Our structure must be in this sense euclidean rings.

Proposition 2.2. Let $\mathcal{R}$ be a euclidean ring, then each irreducible element $a \in \mathcal{R}$ is prime.

Proof: See reference [17].

As we see from the above two propositions we can conclude that in euclidean rings *primes* = *irreducibles* (as every euclidean ring is an integral domain).

Uniqueness of decomposition of prime elements

Lemma 2.1.1. *Let $\mathcal{R}$ be a euclidean ring, $a \in \mathcal{R}$, $a \neq 0$. If $a \notin \mathcal{R}^*$, then there exists a prime element which divides a.*

Proof: see reference [17].

Theorem 2.1. Let $\mathcal{R}$ be a euclidean ring, $a \in \mathcal{R}$, $a \neq 0$.

1. Either $a \in \mathcal{R}^*$ or a can be written as a product of prime elements, i.e. $\exists s \in \mathbb{N}$ and prime elements $p_1, \ldots, p_s$ so that

$$a = \prod_{i=0}^{s} p_i.$$

2. Let $a = \prod_{j=0}^{t} q_j$ be an another factorisation of a. Then $s = t$ and there exists a permutation of the set $\{1, \ldots, s\}$ such that p_i & $q_{\pi(i)}$ are associate.

Actually above theorem tells us that each element of a euclidean ring can be written in terms of prime elements (equivalently irreducible elements). We will use this idea in our special case $\mathcal{R} = \mathbf{Z}$ in the next section so as to conclude that every natural number has a unique prime number decomposition!

We will end this basic section with the definition of *a greatest common divisor* and the general version of so-called *euclidean algorithm*.

Definition 2.6. Let $\mathcal{R}$ be an integral domain, $a, b \in \mathcal{R}$, d be a common divisor of a & b. Then d is called *a greatest common divisor*, abbreviated by *gcd*, of a & b, if $d' \mid d$ for all common divisor d' of a & b.

Lemma 2.1.2. *Let $\mathcal{R}$ be a principal ideal domain, $a, b \in \mathcal{R}$ and $I = (a, b)$ be an ideal of $\mathcal{R}$ which is generated by a and b. If $I = (d)$ with $d \in \mathcal{R}$, then d is a gcd of a and b.*

Proof: see reference [17].

Theorem 2.2. Euclid Algorithm Let $\mathcal{R}$ be a euclidean ring with euclid function ψ, $a, b \in \mathcal{R}$ $b \neq 0$, $I = (a, b)$.
Set $r_0 = a$; $r_1 = b$ and $\forall i \geq 0$ $r_i = q_i.r_{i+1} + r_{i+2}$ such that $r_{i+2} = 0$ or $\psi(r_{i+2}) < \psi(r_{i+1})$.
Then $\exists$ a smallest natural integer κ with $r_\kappa \neq 0$ but $r_{\kappa+1} = 0$ and $I = (r_\kappa)$.

Proof: See reference [17].

This theorem implies that for a given $a, b \in \mathcal{R}$ it is possible to find a *gcd* d of a and b. Note that there exist $x, y \in \mathcal{R}$ such that $d = ax + by$. Also note that in the general case *gcd* is not unique!

Remark 2.3. One can also give for this general case the so-called *extended euclidean algorithm* which simultaneously computes d, x and y. See reference [17] and [46].

At the we will give our first algorithm, namely *extended GCD algorithm* for integers, in a pseudo-code.

ALGORITHM:extended GCD
Input: Given $a, b \in \mathbf{Z}$ with $a \geq b$,
Output: $d = gcd(a, b)$ and integers x, y satisfying $ax + by = d$,

1. If $b = 0$, then set $d \leftarrow a$, $x \leftarrow 1$, $y \leftarrow 0$, and return(d, x, y);
2. Set $x_2 \leftarrow 1$, $x_1 \leftarrow 0$, $y_2 \leftarrow 0$, $y_1 \leftarrow 1$;
3. While $b > 0$ do the following:
   (a) $q \leftarrow \lfloor a/b \rfloor$, $r \leftarrow a - qb$, $x \leftarrow x_2 - qx_1$, $y \leftarrow y_2 - qy_1$,
   (b) $a \leftarrow b$, $b \leftarrow r$, $x_2 \leftarrow x_1$, $y_2 \leftarrow y_1$, and $y_1 \leftarrow y$,
4. Set $d \leftarrow a$, $x \leftarrow x_2$, $y \leftarrow y_2$, and return(d, x, y).

At the end of this section, we are going to review the so-called *Chinese Remainder Theorem*, abbreviated by *CRT* for the general case.

Theorem 2.3. Chinese Remainder Theorem Let $\mathcal{R}$ be a euclidean ring, $t \in \mathbf{N}$ and $m_1, \cdots, m_t \in \mathcal{R}$ such that m_i and m_j are coprime to each other for $i \neq j$. Then we have

$$\mathcal{R}/m\mathcal{R} \cong \bigoplus_{i=1}^{t} \mathcal{R}/m_i\mathcal{R}.$$

Proof: see reference [17].

Corollary 2.2. Let $\mathcal{R}$ be a euclidean ring, $t \in \mathbf{N}$ and $m_1, \cdots, m_t \in \mathcal{R}$ such that m_i and m_j are coprime to each other for $i \neq j$. Furthermore, assume that $a_1, \cdots, a_t \in \mathbf{R}$ are given. Then $\exists\ x \in \mathcal{R}$ with $x \equiv\ a_i\ mod(m_i)$ for $1 \leq i \leq t$. Additionally, $x\ mod\left(m = \prod_{i=1}^{t} m_1 \cdots m_t\right)$ is uniquely determined.

Proof: immediate by CRT!

2.2 Fermat's theorem and Special prime numbers

In this section we are going to revisit some properties of *prime numbers*. Afterwards, *Fermat's little theorem* and related results will be discussed. At the end we will see some *special prime numbers*. Their primality can be tested more easier as we will see in Chapter 4.

Hier we have $\mathcal{R} = \mathbf{Z}$. Obviously $\mathbf{Z}$ is euclidean. Further its prime elements, which are > 0, are called w.l.o.g. *prime numbers* (Prime elements of $\mathbf{Z}$ are p and $-p$, where p is a prime number).

Theorem 2.4. Euclid There are infinitely many prime numbers.

Proof: Let $\mathcal{M}$ be the set of all prime numbers. $\mathcal{M} \neq \emptyset$ as $2 \in \mathcal{M}$. Let now $p_1, \ldots, p_k$ be the distinct prime numbers and $k \geq 1$. Let $Q = 1 + \prod_{1=0}^{k} p_i$ by lemma 2.1.1 $\exists P$ which divides Q.
Let $P = p_i \Rightarrow P \mid Q - \prod_{i=0}^{k} p_i \Rightarrow P \mid 1 \Rightarrow$ contradiction to the assumption that p_i is prime $\forall i$, $1 \leq i \leq k$.

By theorem 2.1 we can conclude that each natural number n has a unique prime factorisation. Then $\exists s \in \mathbf{N}$ and prime numbers $p_1, \ldots, p_s$ such that

$$n = \prod_{i=0}^{s} p_i.$$

Notation: If we write down the same primes together we get

$$n = \prod_{i=0}^{t} p_i^{e_i},$$

$p_i \neq p_j$ if $i \neq j$, $e_i \in \mathbf{N}$. This is called *canonical prime factor decomposition* of n.

Theorem 2.5. Prime number Theorem Let $\pi(x)$ be number of primes smaller than 'x' where $x \in \mathbf{R}$. Then

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

Remark 2.4.

1. Some cryptosystems like RSA needs big prime numbers, such as prime numbers of exactly 128 bits in its binary representations. How many does such primes exist?
   By using Prime number theorem, we have approximately $\pi(2^{128}) - \pi(2^{127}) = \frac{2^{128}}{128.\log 2} - \frac{2^{127}}{127.\log 2}$ ($\approx 1,906.10^{36}$ primes).
2. This theorem, as we see above, will be very useful in running time analysis of our algorithm in chapter 6.

Now we will give the definition of Legendre- and Jacobi-Symbols;

Definition 2.7. Legendre-Symbol Let $p > 2$ be a prime number and $a \in \mathbf{Z}$ then

$$(\frac{a}{p}) = \begin{cases} -1 & if \quad a\ is\ a\ quadratic\ non-residue \\ 0 & if \quad gcd(a,n) \neq 1 \\ +1 & if \quad quadratic\ a\ is\ a\ residue \end{cases}$$

Of course we can generalize this definition for any odd integer N instead of a prime number p.

Definition 2.8. Jacobi-Symbol Let $N = \prod_{i=0}^{k} p_i^{e_i}$ be an odd integer with all p_i's prime and $a \in \mathbb{Z}$. Then $(\frac{a}{N})$ is defined as follows:

$$(\frac{a}{N}) = (\frac{a}{p_1})^{e_1}(\frac{a}{p_2})^{e_2} \dots (\frac{a}{p_k})^{e_k}.$$

ALGORITHM:Jacobi-Symbol

Input: Given $a, N \in \mathbb{Z}$,

Output: Legendre resp. Jacobi-Symbol $(\frac{a}{N})$,

1. (**Test** $N = 0$) if $N = 0$ output 0 if $\mid a \mid \neq 1$,if $\mid a \mid = 1$ then terminate the algorithm;

2. (**Remove 2's from** N) if $gcd(a, N, 2) = 1$, output 0 and terminate the algorithm.Otherwise,
   (a) $v \leftarrow 0$ and while v is even set $v \leftarrow v + 1$ and $N \leftarrow N/2$
   (b) if v is even $k \leftarrow 1$, otherwise set $k \leftarrow (-1)^{\frac{(a^2-1)}{8}}$
   (c) if $N < 0$ set $N \leftarrow -N$ and if in addition $a < 0$ set $k \leftarrow -k$;

3. (**finished?**) (Here clearly N is odd and positive)
   (a) if $a = 0$ then output 0 if $b > 1$, k if $b = 1$
   (b) otherwise set $v \leftarrow 0$ and while a is even do $v \leftarrow v + 1$ and $a \leftarrow a/2$
   (c) if v is odd set $k \leftarrow (-1)^{\frac{(b^2-1)}{8}} k$;

4. (**Apply reciprocity law**) $k \leftarrow (-1)^{\frac{(a-1)(b-1)}{4}} k$. Then $r \leftarrow \mid a \mid$ and $a \leftarrow b \; mod(r)$, $b \leftarrow r$ and GOTO 3;

As we already said that our basic number theoretical result is due to Fermat. Fermat's little theorem and the generalization of its inverse (actually inverse is of course not true!) will be our basic results in our primality testing and afterwards proving algorithms. We will review the results coming with Fermat's theorem.

Theorem 2.6. Fermat's Little Therem Let p be a prime number and $a \in N$ with $gcd(a, p) = 1$ then $a^{p-1} \equiv 1 \; mod(p)$.

Proof See reference for example [17].

Definition 2.9. A composite number N is said to be *pseudoprime* to the base a, denoted by $psp(a)$, $a \in Z$, $1 \leq a \leq N-1$, if $a^{N-1} \equiv 1 \; mod(N)$.

Definition 2.10. A composite number N is said to be *euler pseudoprime* to the base a, denoted by $epsp(a)$, $a \in Z$, $1 \leq a \leq N-1$, if $a^{\frac{N-1}{2}} \equiv \left(\frac{a}{n}\right) mod(N)$.

Proposition 2.3. For all odd number $a \in N$ there are infinitely many $psp(a)$'s and $epsp(a)$'s.

Proof See reference [17].

Proposition 2.4. let p be an odd prime , $p-1 = 2^l.m$, where m is odd, $a \in Z$ with $gcd(a,p) = 1$. Then either $a^m \equiv 1 \; mod(p)$ or there exists some j with $0 \leq j \leq l$ such that $a^{2^j.m} \equiv -1 \; mod(p)$

Proof See reference [17].

Definition 2.11. If N is composite and the above conditions are satisfied for $a \in Z$ then N is called a *strong pseudoprime* to the base a, abbreviated by $spsp(a)$.

Special Prime Numbers

Lemma 2.2.1. *Let $a, b \in \mathbf{N}$ and $a \mid n$. Then $2^a - 1 \mid 2^n - 1$.*

Proof: See reference [17].

Corollary 2.3. If $\mathcal{M}_n = 2^n - 1$ is a prime, then n is a prime. Such primes are called *Mersenne Primes*

Proof: Trivial by above lemma.

Lemma 2.2.2. *If $\mathcal{M}_p$ is prime then $n = 2^{p-1}.\mathcal{M}_p$ is a perfect number. Conversely each even perfect number n has the form $n = 2^{p-1}.\mathcal{M}_p$ where $\mathcal{M}_p$ is a Mersenne prime*

Proof: Due to Euclid see reference [17].

Lemma 2.2.3. *If $N = 2^m + 1$ is prime then $m = 2^n$. The numbers $\mathcal{F}_n = 2^{2^n}$ are called Fermat numbers.*

We will discuss in Chapter 4, how for these special prime numbers we can apply our tests.

2.3 Results from Algebraic Number Theory

In this section we will give the necessary background from the theory of algebraic number theory. We are going to explain quadratic forms and their relation with imaginary quadratic number fields, which are basics of the theory of *Complex Multiplication*, abbreviated by CM-theory, that we will see in next chapter. These results together with CM-theory will enable us to give $N - 1$ analog of a prime number test actually our algorithm due to Atkin. Most of the results needed to introduce our algorithm and related literature and references will also be given. Note that up to now we have just introduced the number theoretical results which are basics of all Primality testing algorithms. We also introduce in Chapter 4 the so-called *Jacobi Sum Test* which is also a *true primality testing algorithm*. However, we will not give the prerequisite theory of that algorithm as it is outside the scope of this thesis. The necessary background can be covered from the book of Cohen [1].

Arithmetic in Quadratic Number Fields

Let $D \in \mathbf{Z}$, let $\sqrt{D} \in \mathbf{C}$ be a root of the polynomial $X^2 - D$. Then

$$\mathbf{Q}(\sqrt{D}) := \{a + b.\sqrt{D} \mid a, b \in \mathbf{Q}\} \subseteq \mathbf{C}.$$

Then $\mathbf{Q}(\sqrt{D})$ is a field, namely quadratic number field. If $D < 0$ ($D > 0$), then it is called complex (real resp.).

Remark 2.5. $\mathbf{Q}(\sqrt{D})$ is a Q-vector space of dimension 2, a basis for example $P_1 = (1, 0)$ and $P_2 = (0, \sqrt{D})$.

WLOG we can assume that D is a squarefree integer. i. e. $D \not\equiv 0\ mod(4)$. Consider just $D \equiv 1, 2, 3\ mod(4)$.

Definition 2.12. The map $\sigma : \mathbf{Q}(\sqrt{D}) \to \mathbf{Q}(\sqrt{D})$;
$\sigma(a + b.\sqrt{D}) = a - b.\sqrt{D}$ is called *conjugate*.

Here are some properties of the conjugate map σ :

1. $\sigma(\alpha \pm \beta) = \sigma(\alpha) \pm \sigma(\beta)$, $\forall \alpha, \beta \in \mathbf{Q}(\sqrt{D})$,
2. $\sigma(\alpha.\beta) = \sigma(\alpha).\sigma(\beta)$, $\forall \alpha, \beta \in \mathbf{Q}(\sqrt{D})$,
3. $\sigma(\frac{\alpha}{\beta}) = \frac{\sigma(\alpha)}{\sigma(\beta)}$, $\forall \alpha, \beta \in \mathbf{Q}(\sqrt{D})$ provided that $\beta \neq 0$,
4. $\sigma(\beta) = 0$ if and only if $\beta = 0$.

Theorem 2.7. Let D be a square free integer. Then $\mathbf{Q}(\sqrt{D})$ is a field with $\mathbf{Q} \subseteq \mathbf{Q}(\sqrt{D}) \subseteq \mathbf{C}$. Each $\alpha \in \mathbf{Q}(\sqrt{D})$ has a unique representation of the form

$$\alpha = a + b.\sqrt{D}, a, b \in \mathbf{Q}.$$

The map σ is an automorphism of $\mathbf{Q}(\sqrt{D})$ which fixes $\mathbf{Q}$ pointwise.

Proof: See reference. [17].

Definition 2.13. $\alpha' = \sigma(\alpha)$ Norm-function: $N(\alpha) = \alpha.\alpha'$ and trace-function: $T(\alpha) = \alpha + \alpha'$

Hence for a given $\alpha = a + b.\sqrt{D}$ we have $N(\alpha) = a^2 - b^2.D$ and $T(\alpha) = 2a$.

Theorem 2.8.

1. The function $S : \mathbf{Q}(\sqrt{D}) \to \mathbf{Q}$ is an epimorphism of additive groups.
2. if $\alpha \in \mathbf{Q}(\sqrt{D})$, then we have $N(\alpha) = 0$ if and only if $\alpha = 0$ (as $D \neq \square$) the function $S^* : \mathbf{Q}(\sqrt{D})^* \to \mathbf{Q}^*$ is a homomorphism of multiplicative groups.

Now let $\alpha = a + b\sqrt{D} \in \mathbf{Q}(\sqrt{D})$ then we get a relation

$$\alpha^2 = a^2 + 2ab\sqrt{D} + b^2D = \underbrace{2a}_{S(\alpha)}(a + b\sqrt{D}) - \underbrace{a^2 - b^2D}_{N(\alpha)}$$

$\Rightarrow \alpha^2 - S(\alpha).\alpha + N(\alpha) = 0.$

Corollary 2.4. Every $\alpha \in \mathbf{Q}(\sqrt{D})$ is a root of a polynomial in $\mathbf{Q}[x]$ of degree ≤ 2. A monic polynomial $\mu_\alpha \in \mathbf{Q}[x]$ with minimal degree such that $\mu_\alpha(\alpha) = 0$ is called *minimal polynomial* of α. Further, μ_α is uniquely determined.

Definition 2.14. Let $\alpha \in \mathbf{Q}(\sqrt{D})$. Then α is called an *algebraic integer* if $\mu_\alpha \in \mathbf{Z}[x]$.

Now we have actually three cases for minimal polynomials of having algebraic integers:

- Let $\alpha \in \mathbf{Z} \Rightarrow \mu_\alpha = x - \alpha \in \mathbf{Z}[x]$,
- Let $\alpha \in \mathbf{Q} - \mathbf{Z} \Rightarrow \mu_\alpha = x - \alpha \notin \mathbf{Z}[x]$,
- Let $\alpha \in \mathbf{Q}(\sqrt{D}) - \mathbf{Z}$; α is an algebraic integer $\Leftrightarrow$ both $T(\alpha) \in \mathbf{Z}$ and $N(\alpha) \in \mathbf{Z}$.

Lemma 2.3.1. *Let $\alpha \in \mathbf{Q}(\sqrt{D})$. Then α is algebraic integer if and only if α has the form*

$$\alpha = \begin{cases} \frac{1}{2}(a + b\sqrt{D}), a, b \in \mathbf{Z} & if \quad a \equiv b \; mod(2) \; for \; D \equiv 1 \; mod(4) \\ a + b\sqrt{D}, a, b \in \mathbf{Z} & if \quad D \equiv 2, 3 \; mod(4) \end{cases}$$

Proof: See reference [17].

Let $\mathcal{O}_D$ be the set of all algebraic integers of $\mathbf{Q}(\sqrt{D})$. Then we have the following corollary and theorem;

Corollary 2.5. $\mathcal{O}_D$ is a ring. Moreover, it is an integral domain.

Proof: trivial by above lemma.

Theorem 2.9. $\mathcal{O}_D$ is a free $\mathbf{Z}$-Module of Rank 2, $\mathcal{O}_D$ has $\mathbf{Z}$-Basis $(1, w_D)$ with

$$w_D = \begin{cases} \sqrt{D} & if \quad D \equiv 2,3 \ mod(4) \\ \frac{1}{2}(1+\sqrt{D}) & if \quad D \equiv 1 \ mod(4) \end{cases}$$

Moreover, $\mathcal{O}_D \cong \mathbf{Z} \oplus w_D.\mathbf{Z}$.

Definition 2.15. Imaginary quadratic discriminant of an imaginary quadratic number field $\mathcal{K} = \mathbf{Q}(\sqrt{-d})$, $d > 0$, square free is equal to

$$D = \begin{cases} d & if \quad D \equiv 3 \ mod(4) \\ 4d & if \quad D \not\equiv 3 \ mod(4) \end{cases}$$

.

Corollary 2.6. The set of all algebraic numbers in $\mathcal{K}$ can be given by the following isomorphy;

$$\mathcal{O}_K = \begin{cases} \mathbf{Z} + \mathbf{Z}\sqrt{-d} & if \quad d \not\equiv 3 mod(4) \\ \mathbf{Z} + \frac{-1+\sqrt{-d}}{2}\mathbf{Z} & if \quad d \equiv 3 mod(4) \end{cases}$$

.

Definition 2.16. Order of an imaginary quadratic number field An order $\mathcal{O}$ in $\mathcal{K}$ is a subring of $\mathcal{K}$ which is as a $\mathbf{Z}$-Module finitely generated and of maximal rank $n = deg(K)$.

Definition 2.17. An ideal α of $\mathcal{O}$ is a sub-$\mathcal{O}$-module, i. e. a sub-$\mathbf{Z}$-module of $\mathcal{O}$ such that every $r \in \mathcal{O}$ and $i \in \alpha$ we have $r.i \in \alpha$.

Definition 2.18. An ideal α is said to be *principal ideal* of $\mathcal{O}$, if there exists $x \in \mathcal{K}$ such that $\alpha = x\mathcal{O}$. Furthermore $\mathcal{O}$ is a principal ideal domain (PID) if $\mathcal{O}$ is an integral domain (for orders it is always the case) and if every ideal α of $\mathcal{O}$ is a principal ideal.

Assume $\mathcal{O}_K$ is principal order of an imaginary quadratic number field $\mathcal{K}$. Then $\mathcal{O} \subset \mathcal{O}_K$.

Definition 2.19. Two ideals α, β are said to be equivalent, if there exist a, b $\in \mathcal{K}^*$ with $a.\alpha = b.\beta$.

Equivalent classes form an ideal class $\mathcal{U}$. Every ideal class of $\mathcal{O}_D$ has an ideal of the form;

$$a\mathbf{Z} + \frac{-b + \sqrt{-D}}{2}\mathbf{Z} \; for \; a \in \mathbf{N}, b \in \mathbf{Z}$$

with additionally $c = (b^2 + D)/4a \in \mathbf{Z}$ *and* $gcd(a, b, c) = 1$.

Definition 2.20. A *fractional ideal* i of $\mathcal{O}$ is a non-zero submodule of $\mathcal{K}$ such that there there exists a non-zero integer d with di an ideal of $\mathcal{O}$.

Definition 2.21. Let i be a fractional ideal of $\mathcal{O}$. We say that i is *invertible* if there exists a fractional ideal j of $\mathcal{O}$ such that $\mathcal{O} = ij$. Such an ideal j is then called *inverse* of the ideal i.

Lemma 2.3.2. *Let i be a fractional ideal, and set*

$$i' = \{x \in \mathcal{K}, xi \subset \mathcal{O}\},$$

then i is invertible if and only if $ii' = \mathcal{O}$. Moreover, if this equality is true, then i' is unique inverse of i and denoted by i^{-1}.

Proof: immediate!

2.4 Quadratic Forms

In this section we will introduce the quadratic forms and their relations with imaginary quadratic number fields and discriminants. In fact, the idea is that quadratic forms & invertible fractional ideals of imaginary quadratic order $\mathcal{O}$ are the same structure in an imaginary quadratic number field $\mathcal{K}$. This equivalence will enable us to find an another one in the next chapter, the equivalence between lattices $\Lambda \subset \mathbf{C}$ and hence the equivalence between elliptic curves over $\mathbf{C}$, which will be then particularly used in CM-theory and at the end in our ECPP algorithm.

Definition 2.22. A *binary quadratic form* over $\mathbf{Z}$ is a map:
$f : \mathbf{Z}^2 \to \mathbf{Z}$ $f(x, y) = ax^2 + bxy + cz^2$, $a, b, c \in \mathbf{Z}$ with discriminant $D = 4ac - b^2$. This form is called *primitive* if $gcd(a, b, c) = 1$. Hence, it can be identified as a triple, i. e. $f = (a, b, c)$.

Remark 2.6. Quadratic orders can be also represented in terms of matrices

$$f(x, y) = (x, y) \underbrace{\begin{pmatrix} a & b/2 \\ b/2 & c \end{pmatrix}}_{M_f} \begin{pmatrix} x \\ y \end{pmatrix}$$

Definition 2.23. Two binary quadratic forms f & g are said to be equivalent, abbreviated by $(f \sim g)$, if $\exists \mathcal{A} \in SL(2, \mathbf{Z})$ with $M_g = \mathcal{A}^{-1}.M_f.\mathcal{A}$.

This is obviously an equivalence relation.

Definition 2.24. Binary quadratic forms f in equivalence classes $[f]$ form a group called *form class group*, and denoted by $Cl(D)$.

Let's analyse this equivalence classes a little bit more;

Corollary 2.7. Each equivalence class contains exactly one form (a, b, c) with $a, b\& c$ are relatively prime and satisfy the following;
$\mid b \mid \leq a \leq c$ & ($\mid b \mid = a$ or $a = c \Rightarrow b > 0$). Such a form is called *reduced*.

Proof: See reference [11].

Theorem 2.10. Let $\mathcal{O}_K$ be a principal order of an imaginary quadratic number field $\mathcal{K}$, and D be an imaginary quadratic discriminant of $\mathcal{O}_K$. Then define a map;

$$\phi : f(x, y) = ax^2 + bxy + c^2 \longmapsto a\mathbf{Z} + \frac{-b + \sqrt{-D}}{2}\mathbf{Z}$$

with $D = b^2 - 4ac$.
Then ϕ is a bijection between form class group $Cl(D)$ and the ideal class group $Cl(\mathcal{O})$.

Proof: See reference [11].

This theorem says us that ;
$\mid Cl(D) \mid = \mid Cl(\mathcal{O}_D) \mid$ which implies that $h(\mathcal{O}_D) = h(D)$

Review: $h(\mathcal{O}_D)$ (and $h(D)$) is by definition the cardinality of $Cl(\mathcal{O}_D)$ (cardinality of $Cl(D)$, respectively).

Definition 2.25. Let D be an imaginary quadratic discriminant, $n \in \mathbf{Z}$ and $f = (a, b, c)$ be a quadratic form with discriminant D. If $\exists (x, w) \in \mathbf{Z}^2$ such that

$$n = ax^2 + bxw + w^2 = f(x, w)$$

Then n can be represented by means of the function f. Such an n is called *Norm* of an element of $\mathcal{O}_D$ if there exists $\pi \in \mathcal{O}_D$ with $n = \pi\bar{\pi}$.

Question: When have we such a form and how can we compute it?

Lemma 2.4.1. *Let D be an imaginary quadratic discriminant $n \in \mathbf{Z}$. There exists $\pi \in \mathcal{O}_D$ with $n = \pi\bar{\pi}$ if and only if $4n = t^2 + Dy^2$ has a solution $(t, y) \in \mathbf{Z}^2$.*

We will end this section to explain the method to compute these diaphontine equations. Note that it is not always the case that such a solution exists. In order to find such a pair (t, y), we can use so-called **Cornaicchia's Algorithm** which also give us a chance to know whether or not such a solution exists. This algorithm computes essentially for a given rational the continued fraction of the square root. This idea was also used so as to factor the integers. Note that finding such a pair is equivalent to solving $p = \tilde{x}^2 + d\tilde{y}^2$ for a prime number p. Note that this algorithm will be also used in CM-Theory of elliptic curves for a given discriminant D.

ALGORITHM: Cornaicchia's Algorithm
Input: Given a square free integer D and a prime number p,
Output: A solution to $p = \tilde{x}^2 + d\tilde{y}^2$, if exists,

1. Let $p/2 < x_0 < p$ be solution to $x^2 \equiv -D \; mod(p)$;

2. $p \leftarrow q_0 x_0 + x_1$, $k \leftarrow 0$;

3. Until $x_k^2 < p \leq x_{k-1}$ do;
   (a) $x_k \leftarrow q_{k+1}xk + 1 + x_{k+2}$ $k \leftarrow k + 1$,
   (b) $\tilde{x} \leftarrow x_k$, $\tilde{y} \leftarrow \sqrt{(p - x_k^2)/d}$,

4. if $\tilde{y} \in \mathbf{Z}$ return $(\tilde{x}, \tilde{y})$, else return '*No Solution*';

One can also modify the algorithm to get a more efficient method. Here is the modified version of **Cornacchia** due to Cohen. In this method we also do not need to use $(\tilde{x}, \tilde{y})$ transformation;

ALGORITHM: Modified Cornaicchia's Algorithm
Input: Given a square free integer D and a prime number p such that $D \equiv 0$ or $1\ mod(4)$ and $\mid D \mid < 4p$,
Output: A solution to $4p = x^2 + dy^2$, if exists,

1. (**Case** $p = 2$) If $D + 8$ is a square of a natural number return $(\sqrt{D+8}, 1)$, otherwise return '*No Solution*';

2. (**Test if it is residue**) using Jacobi-Algorithm compute $k \leftarrow \left(\frac{D}{p}\right)$. If $k = -1$ return '*No Solution*';

3. (**Compute square root**) Compute an integer x_0 such that $x_0^2 \equiv D\ mod(p)$ and $0 \leq x_0 < p$;
   (a) Set $x_0 \leftarrow p - x_0$ if $x_0 \neq D\ mod(2)$,
   (b) Set $a \leftarrow 2p$, $b \leftarrow x_0$, and $l \leftarrow \lfloor 2\sqrt{p} \rfloor$

4. (**Euclidean Algorithm**) if $b > l$, set $r \leftarrow a\ mod\ b$, $a \leftarrow b$, $b \leftarrow r$ and **GOTO** step 4;

5. (**Test Solution**) If $\mid D \mid$ does not divide $4p - b^2$ or if $C = (4p - b^2)/ \mid D \mid$ is not the square of an integer return '*No Solution*'. Otherwise return $(x, y) = (b, \sqrt{c})$.

CHAPTER 3

Preliminaries from the arithmetic of Elliptic curves

In this chapter, we will introduce some important results coming together with the arithmetic of elliptic curves over different fields. Elliptic curves have an extensive literature as they are used in many branches of both theoretical and applied mathematics and are closely related with the theory of elliptic functions, from which they derive their name. Elliptic curves have been used and studied in the recent in the proof of *Fermat's last Theorem*. They have been also used in factorization of integers, cryptography, and as in our case primality proving for more than two decades. Elliptic curves have an extensive usage in cryptography, in particular *public key Cryptography*, since it is possible to reach a reasonable security when we compare with other cryptosystems like RSA. Furthermore the same level of security can be gained with a reasonable smaller key sizes, and hence of smaller memory and processor requirements. Additionally, they give a chance to construct cryptosystems based on *Discrete Logarithm Problem*, abbreviated by DLP. In primality proving, an analog method like $N-1$ tests will be developed by means of complex multiplication (CM-) method of of elliptic curves.

3.1 Some results from the theory of Elliptic Curves

Definition 3.1. Let $F \in \mathbf{R}[x, y]$ be a polynomial of degree d ($F \neq 0$). Then

$$C := \{(x, y) \in \mathbf{R}^2 \mid F(x, y) = 0\}$$

is called a *curve* of degree d.

Let $\mathcal{K}$ be a field and $\bar{\mathcal{K}}$ be an algebraic closure of $\mathcal{K}$, let further $(x_0, x_1, \ldots, x_n) \in \bar{\mathcal{K}}^{n+1} - \{0\}$. Define $(x_0 : x_1 : \ldots : x_n)$ as a unique line between 0 and $(x_0, x_1, \ldots, x_n)$.

Definition 3.2. The *n-dimensional projective space* over $\bar{\mathcal{K}}$ is the set;

$$\mathbf{P}^n = \{(x_0 : x_1 : \ldots : x_n) \mid (x_0, x_1, \ldots, x_n) \in \bar{\mathcal{K}}^{n+1} - \{0\}\}.$$

Notation: n-dimensional projective space over $\bar{\mathcal{K}}$ is also denoted by $\mathbf{P}^n_{\bar{\mathcal{K}}}$.

Definition 3.3. 1. A point $(x_0 : x_1 : \ldots : x_n) \in \mathbf{P}^n$ is said to be a *k-rational point*, if there exists $\lambda \in \mathcal{K}^*$ so that

$$\lambda(x_0 : x_1 : \ldots : x_n) \in \bar{\mathcal{K}}^{n+1}.$$

The set of all k-rational points of $\mathbf{P}^n$ is abbreviated by $\mathbf{P}^n(\mathcal{K})$.

2. $(x_0 : x_1 : \ldots : x_n)$ are called *homogene coordinates* of $\mathbf{P}^n$.

Definition 3.4. A subset $C \in \mathbf{P}^2$ is called an *algebraic curve*, if $\exists$ a non-constant homogene polynomial $F \in \bar{\mathcal{K}}[x, y, z]$ such that $C = V(F)$.

Remark 3.1. Define $L_z := V(z) \subset \mathbf{P}^2$, so that

$$L_z = \{(x : y : z) \in \mathbf{P}^2 : z = 0\} \cong \mathbf{P}^1.$$

Then there is an isomorhism
$\varphi_z : U_z := \mathbf{P}^2 - L_z \to \bar{\mathcal{K}}^2$ such that $\binom{(x:y:z)\to(x/z,y/z)}{(s:t:1)\leftarrow(s,t)}$
Then U_z is called *affine part* of $\mathbf{P}^2$ and $L_z \cong \mathbf{P}^1$ is called *line at infinity*.

Definition 3.5. Let $C \in \mathbf{P}^2$ be an algebraic curve. The affine part of C is the subset

$$C' := C \cap U_z.$$

Definition 3.6. An algebraic curve of degree d is said to be *smooth*, if there exists a polynomial $F \in \mathcal{K}[x, y, z]$ of degree d such that
$C = V(F)$ and $(\frac{\partial F}{\partial x}(P), \frac{\partial F}{\partial y}(P), \frac{\partial F}{\partial z}(P)) \neq (0, 0, 0)\ \forall P \in C$.

Notation: Let $i_p(C, L) :=$ denote the number of points of $C \bigcap L$, where L is another line (resp. curve).

Theorem 3.1. Bezout's theorem Let C_1, C_2 be two smooth algebraic curves in $\mathbf{P}^2$. Then

$$\sum_{P \in C_1 \bigcap C_2} i_p(C_1, C_2) = deg(C_1).deg(C_2).$$

Proof: see reference [18].

Definition 3.7. Let $C = V(F)$ be a smooth algebraic curve of degree d, $deg(F) = d$. Then *Hesse-curve* of C is the set

$$H_C := V(det(\frac{\partial^2 F}{\partial x_i \partial x_j})_{0 \leq i,j \leq 2}) \subset \mathbf{P}^2$$

.

Remark 3.2.

- For $d \leq 2$, we have $det(\frac{\partial^2 F}{\partial x_i \partial x_j})$ is constant, and hence not a curve in the concept of the definition 3.4.
- if $det(\frac{\partial^2 F}{\partial x_i \partial x_j}) \neq 0$, then H_C is also a smooth algebraic curve of degree $3d(d-2)$ if $d \geq 3$.

Definition 3.8. A point $P \in C$ is called an *inflection point*, if there exists a tangent L of C at the point P such that $ip(C, L) \geq 3$. Then L is called *inflection tangent* of C at P.

Theorem 3.2. (Let $char(K) \neq 2$) $P \in C$ is an inflection point if and only if $P \in C \cap H_C$.

Proof: See reference [18].

Remark 3.3. By Bezout's theorem, there are at most $3d(d-2)$ and at least 1 inflection points over an algebraic curve C.

Definition 3.9. Let C be a smooth algebraic curve of degree 3 and $\mathcal{O} \in C$ be an inflection point. Define

$$+ : \begin{pmatrix} C \times C \longrightarrow C \\ (P, Q) \mapsto P + Q \end{pmatrix}$$

as follows;
Let $P\bar{Q}$ be the line connecting $P\&Q$ (tangent if $P = Q$), Then by Bezouts we have $P\bar{Q} \cap C = \{P, Q, R\}$ (with multiplicity). Let $O\bar{R}$ be the line connecting $O\&R$ then $O\bar{R} \cap C = \{O, R, C\}$.
Define $P + Q := S$.

Properties of the function '+':

1. $\forall P \in C$ we have $P + \mathcal{O} = P$(as $\mathcal{O}$ is an inflection point)
2. $\forall P, Q \in C$ we have $P + Q = Q + P$.
3. $\forall P \in C$ consider $P\bar{\mathcal{O}} \cap C = \{P, \mathcal{O}, Q\}$.Set: $-P := Q$
   since: $P(\bar{-P}) \cap C = \{P.\mathcal{O}, -P\}$ and $\mathcal{O}\bar{\mathcal{O}} \cap C = \{\mathcal{O}, \mathcal{O}, \mathcal{O}\}$ as $\mathcal{O}$ is an inflection point, we have $P + (-P) = \mathcal{O}$.
4. $\forall P, Q, R \in C$: we have $(P + Q) + R = P + (Q + R)$

Proof: See reference [16]. Hence we can conclude that $(C, +)$ is an abelian group.

Definition 3.10. An *Elliptic Curve* is the pair $(E, +)$ where E is a smooth algebraic curve of degree 3 and $+$ is a group structure like above.

Remark 3.4. The choice of $\mathcal{O}$ as an inflection point is not necessary. However if we choose $\mathcal{O}$ as an inflection point we have the following nice property:
$P + Q + R = \mathcal{O} \Leftrightarrow P, Q$ and R are collinear.

Remark 3.5. Let $m \in \mathbf{Z}$. Then we define;

$$m.P = \begin{cases} \underbrace{P + P + \cdots + P}_{m-times} & if \quad m > 0 \\ 0 & if \quad m = 0 \\ -(\mid m \mid).P & if \quad m < 0 \end{cases}$$

Theorem 3.3. Weierstrass Normal Form Let E be an elliptic curve and $\mathcal{O}$ be an inflection point, let also $E = V(F)$ with $deg(F) = 3$. Then there exists a

projective transformation φ_A

$$\varphi_A : \mathbf{P}^2 \to \mathbf{P}^2$$

so that $\tilde{E} = V(\tilde{F})$ with

$$\tilde{F}(X,Y,Z) = F(\varphi_A(X,Y,Z)) = Y^2Z + a_1XYZ + a_3YZ^2 - x^3 - a_2X^2Z - a_4XZ^2 - a_6Z^3$$

Moreover, $\varphi_A^{-1}(\mathcal{O}) = (0:1:0)$ is the unique point at infinity of $\tilde{E}$.

Proof: See reference [16].

Remark 3.6. With a coordinate transformation, elliptic curve E' is an affine curve, i. e.

$$E' = E \cup U_z = \{(x,y) \in \bar{\mathcal{K}}^2 : y^2 + a_1XY + a_3Y = x^3 + a_2X^2 + a_6\}$$

plus a point at infinity $\mathcal{O} = (0:1:0)$.

Remark 3.7. For $char(K) \neq 2, 3$ there is an easier normal form with respect to a coordinate exchange. Then E has a form for $E = V(F)$

$$F(X,Y,Z) = Y^2 - X^3 - aXZ^2 - bZ^3 \; and \; \exists a, b \in \bar{\mathcal{K}}$$

that means;

$$E = \{(X,Y) \in \bar{\mathcal{K}}^2 : Y^2 = X^3 + aX + b\} \cup \{(0:1:0)\}$$

Proof:See [6].

Lemma 3.1.1. *Let E be an elliptic curve, then E is smooth if and only if $4a^3 + 27b^2 \neq 0$.*

Proof: trivial...

Theorem 3.4. Let $char(\bar{\mathcal{K}}) \neq 2, 3$. Further let $E = V(F)$ be an elliptic curve with $F(X,Y,Z) = Y^2Z - X^3 - aXZ^2 - bZ^3$ with $a, b \in \bar{\mathcal{K}}$ (particularly: $4a^3 +$

$27b^2 \neq 0$). Moreover, assume that $P, Q \in E$, $P, Q \neq 0$ and $P \neq -Q$ and `w.l.o.g.` we have $P = (x_1, y_1, 1)$ & $Q = (x_2, y_2, 1)$. Then we have;

$$P + Q = (\lambda^2 - x_1 - x_2 : -\lambda(\lambda^2 - x_1 - x_2) - \mu : 1)$$

with $\lambda := \frac{y_2 - y_1}{x_2 - x_1}$ and $\mu := \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1}$ if $P \neq Q$. If we have $P = Q$. Then ;

$\lambda := \frac{3x_1^2 + a}{2y_1}$ and $\mu := \frac{-x_1^3 + ax_1 + 2b}{2y_1}$.

Proof: See [16].

Remark 3.8. If we have $y_1 = 0 \Rightarrow P = -P$ as we have $-(x : y : 1) = (x : -y : 1)$.

Remark 3.9. Of course for the general Weierstrass normal form, i. e. if we have $char(\bar{\mathcal{K}}) = 2, 3$, the generalized version of this addition process can be applied. See [7].

Definition 3.11. An endomorphism ϕ of an elliptic curve E is a map $\phi : E \to E$ with $\phi(\infty) = \infty$. The set of all endomorphisms of an elliptic curve forms a ring and is abbreviated by $End(E)$.

For the structure of the endomorphism ring $End(E)$ we have three choice, namely;

1. $End(E) = \mathbf{Z}$ (not possible for curves over finite fields),
2. $End(E)$ is an order of an imaginary quadratic number field,
3. $End(E)$ is the maximal order of a quarternion algebra.

Remark 3.10. We will discuss the second case in detail in CM-theory later!

Definition 3.12. If we have $\mathbf{Z} \subsetneq End(E)$, then we say that E has *complex multiplication*, denoted by E with CM.

Definition 3.13. The j-invariant of an elliptic curve E with $\Delta \neq 0$ is defined as a constant $j(E) = 1728.\frac{(4a)^3}{\Delta}$ where $\Delta = 4a^3 + 27b^2$.

Lemma 3.1.2. *Let E_1 and E_2 be two elliptic curves over an algebraically closed field $\mathcal{K}$. Then E_1 and E_2 are isomorphic if and only if $j(E_1) = j(E_2)$.*

Proof: see [6].

Remark 3.11.
1. If we have $j = 0$ (resp. $j = 1728$), we have $a = 0$ and $b = 1$ (resp. $a = 1$ and $b = 0$).
2. One can also show that for every $j \in \mathcal{K}$, there exists an elliptic curve E with $j(E) = j$.

3.2 Elliptic curves over C

Definition 3.14. Given a lattice Λ in $\mathbf{C}$ with $\Lambda = \{nw_1 + mw_2 \mid n, m \in \mathbf{Z}\}$ with $w_1, w_2 \in \mathbf{C}^*$, $\frac{w_1}{w_2} \notin \mathbf{R}$.

1. A meromorphic function f is said to be *elliptic* if $f(z+w) = f(z)$, $\forall w \in \Lambda$.
2. *Weierstrass $\wp$-function* associated to Λ is given by

$$\wp(z; w) = z^{-2} + \sum_{w \in \Lambda \setminus \{0\}} \left((z-w)^{-2} - w^{-2}\right)$$

.

Then we have

$$\wp(z; \Lambda) : z \mapsto \begin{cases} \wp(z; w) = z^{-2} + \sum_{w \in \Lambda \setminus \{0\}} \left((z-w)^{-2} - w^{-2}\right) & if \quad z \notin \Lambda \\ \infty \quad if \quad z \in \Lambda \end{cases}$$

.

Theorem 3.5. Weierstrass $\wp$-function is meromorphic, and it is elliptic (double-periodic) and satisfies the following differential equation;

$$\wp'(z)^2 = 4\wp(z)^3 - g_2(\Lambda)\wp(z) - g_3(\Lambda)$$

with the constants $g_2(\Lambda) = 60 \sum_{w \in \Lambda \setminus \{0\}} \frac{1}{w^4}$ and $g_3(\Lambda) = 140 \sum_{w \in \Lambda \setminus \{0\}} \frac{1}{w^6}$.

ProofSee reference [6].

Definition 3.15. Two lattices Λ_1 and Λ_2 are said to be *homothetic*, if there exists $\lambda \in \mathbf{C}$ such that $\Lambda_1 = \lambda\Lambda_2$.

Definition 3.16. The j-invariant of a lattice $\Lambda$is a complex number

$$j(\Lambda) = 1728.\frac{g_2(\Lambda)^3}{g_2(\Lambda)^3 - 27g_3(\Lambda)^2} = 1728.\frac{g_2(\Lambda)^3}{\Delta(\Lambda)}$$

.

Theorem 3.6. Two lattices Λ_1 and Λ_2 are homothetic if and only if they have the same j-invariant.

Proof: See reference [6].

Theorem 3.7. Let $E = (a, b)$ be an elliptic curve over $\mathbf{C}$. Then there exists a uniquely defined lattice $\Lambda \subset \mathbf{C}$ so that $a = -g_2(\Lambda)/4$ and $b = -g_3(\Lambda)/4$ respectively.

Proof: See [6].

Proposition 3.1. Let $\mathcal{O}$ be an order in an imaginary quadratic number field $\mathcal{K}$, i be an invertible fractional ideal. Then i can be considered as a lattice over $\mathbf{C}$.

Proof: See [6].

Theorem 3.8. Let Λ be a lattice, and $\wp(z;\Lambda)$ the Weierstrass $\wp$-function. Furthermore, assume we have $\alpha \in \mathbf{C}\backslash\mathbf{Z}$. Then the followings are equivalent;

1. $\wp(\alpha z)$ is a rational function.
2. $\alpha\lambda \subset \Lambda$.
3. There exists an order $\mathcal{O}$ in an imaginary quadratic number field and Λ is homothetic to an invertible fractional ideal of that order $\mathcal{O}$.

Proof: See [6] and [5].

Theorem 3.9. Let E be an elliptic curve over $\mathbf{C}$ and Λ be the corresponding lattice. Then;

$$End(E) \cong \{\alpha \in \mathbf{C} \mid \alpha\Lambda \subset \Lambda\}.$$

By the above two theorems we have;
$End(E)$ of an elliptic curve E is an order $\mathcal{O}$ of an imaginary quadratic number field if and only if the lattice Λ is homothetic to an invertible fractional ideal i of $\mathcal{O}$.

$\Rightarrow \exists$ bijective map between set of invertible fractional ideals of imaginary quadratic orders and set of isomorphy classes of an elliptic curve.

$\Rightarrow$ particularly we have $j(i) = j(E)$, where $j(i)$ and $j(E)$ are j-invariants of imaginary quadratic order $\mathcal{O}$ and elliptic curve E, respectively.

Theorem 3.10. Let $\mathcal{O}$ be an order in an imaginary quadratic number field $\mathcal{K}$, i be an invertible fractional ideal of the order $\mathcal{O}$. Then $j(i)$ is an algebraic integer of degree maximal $h(\mathcal{O})$.

Proof: See [5].

Review: By previous chapter we have $h(\mathcal{O})$ as the cardinality of the ideal class group $Cl(\mathcal{O})$.

Let now $\mathcal{K}$ be an imaginary quadratic number field with maximal imaginary quadratic order $\mathcal{O}_K$. Then one can represent the ideal class group $Cl(\mathcal{O}_K)$ of $\mathcal{O}_K$ with the represantatives;

$$i_1, \cdots, i_{h_D}$$

where represantatives are lattices over $\mathbf{C}$. Hence $j(i_1), \cdots, j(i_{h_D})$ will determine the j-invariants.
Let $\mathcal{L}$ be the smallest Galois extension of $\mathcal{K}$, in which each $j(i_s)$, $0 \leq s \leq h_D$ is contained.
$\Rightarrow$ we have then an isomorphism $\mathbf{C}/i_s \rightarrow E_{i_s}(\mathbf{C})$.

Let now E be an elliptic curve over $\mathbf{C}$ with $End(E) \cong \mathcal{O}_D$ and $i_1, \cdots, i_D$ are elements of the ideal class group $Cl(\mathcal{O}_D)$. Let further ε denote the elliptic curve over $\mathcal{L}$ and be given as follows;

$$j_0 = j(i_k)$$

for some $k \in \{1, \cdots, h_D\}$ with $\kappa = j_0/(1728 - j_0)$ and $\varepsilon = (3\kappa, 2\kappa)$.
$\Rightarrow$ Lattices of E and ε are homothetic!

Theorem 3.11. Let $\mathcal{K}$ be an imaginary quadratic number field, $\mathcal{O}_K$ be the ring of algebraic integers in $\mathcal{K}$. Then there exist exactly h_D isomorphy classes elliptic curves with complex multiplication (CM) with $\mathcal{O}_K$ where D is the imaginary quadratic discriminant of $\mathcal{O}_K$. Moreover, they can be defined over *Hilbert Class Field* H_K. Class polynomial of a Hilbert Class Field is

$$H_D(X) = \prod_{s=1}^{h_D} (X - j(i_s))$$

.

Proof and details: see reference [6], [5] and their references.

3.3 Elliptic curves over finite fields $\mathbf{F}_q$

Definition 3.17. Let E be an elliptic curve over a finite field $\mathbf{F}_q$. The q^{th}-power of *Frobenius map* is defined by

$$\varphi : \begin{cases} E(\overline{\mathbf{F}_q}) \to E(\overline{\mathbf{F}_q}) \\ \underset{\mathcal{O} \mapsto \mathcal{O}}{(x,y) \mapsto (x^q, y^q)} \end{cases}$$

It is also easy to see that φ maps points on E to points on E, i. e. it respects the group law. In fact φ is a group endomorphism of E over $\mathbf{F}_q$.

Corollary 3.1. 1. Let E be an elliptic curve over $\mathbf{F}_q$. then we have

$$\mid E(\mathbf{F}_q) \mid = q + 1 - t$$

where t is the trace of Frobenius at q.

2. The Frobenius endomorphism φ and the trace of Frobenius satisfy the following functional equation

$$\varphi^2 - [t]\varphi + [q] = [0]$$

that is for a given point $P = (x, y)$ on the curve (given in affine coordinates) we have the following functional equality

$$(x^{q^2}, y^{q^2}) - [t](x^q, y^q) + [q](x, y) = \mathcal{O}$$

.

Note: addition and substaction are curve operations!
Proof: see reference [6].

Theorem 3.12. Hasse Let p be a prime number and $q = p^n$, $n \in \mathbf{N}$. Let also E be an elliptic curve over $\mathbf{F}_q$. Then

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

.

Proof: See [6].

Definition 3.18. Let E be an elliptic curve over $\mathbf{F}_p$, where p is a prime number. Let also $| \ E(\mathbf{F}_p) \ | = p + 1$. Then E is called *supersingular*.

Lemma 3.3.1. *If E is a supersingular elliptic curve over $\mathbf{F}_p$. Then $End(E)$ is the maximal order of a quarternion algebra.*

As we already discussed we have just 2 choices for the endomorphism ring of an elliptic curve over finite fields. (As it is the case for finite fields that $End(E) \ncong \mathbf{Z}$)). The second case is the case of above lemma and the important case is explained by means of the following theorem;

Theorem 3.13. Let E be an elliptic curve over $\mathbf{F}_p$, where p is a prime number. The endomorphism ring of E is an imaginary quadratic order if and only if $\mid E(\mathbf{F}_p) \mid \neq p+1$.

Proof: see [6].

Remark 3.12. Let $\Upsilon \in \mathcal{O}_D$ be a prime ideal with $\mathcal{O}/\Upsilon \cong \mathbf{F}_p$. Furhermore, $\varepsilon = (\alpha, \beta)$ is a non-supersingular elliptic curve over $\mathcal{L}$ and let $E = (\alpha \ mod \ \Upsilon, \beta \ mod \ \Upsilon)$ be the reduced curve modulo Υ.

A curve ε has a *'good'* reduction modulo Υ if E is again a non-supersingular elliptic curve. It means

$$j(\varepsilon) \ mod(\Upsilon) = j(E).$$

Theorem 3.14. If $j \in \mathbf{F}_p$, $j \neq 0, 1728$, then there exist maximal two isomorphy classes of elliptic curves over $\mathbf{F}_p$. If $j = 0$ (resp. $j = 1728$), then there exist maximal 6 (resp. 4) isomorphy classes.

Proof: see also for details [6].

Theorem 3.15. Deuring Let $\mathcal{O}_K$ be a maximal imaginary quadratic order. Moreover, let ε be an elliptic curve over H_D with $End(\varepsilon) \cong \mathcal{O}_K$ and let further p be a prime ideal of degree 1 (i. e. $p = \overline{p}$) and $(P) = p.\overline{p}$ for that P has a 'good' reduction. Then, $End(\varepsilon) \cong End(E)$, *i. e.* $End(E) = \mathcal{O}_K$.

Proof: see [6] and [5].
$\Rightarrow \exists$ some $\pi \in \mathcal{O}_K$ with $p = \pi.\overline{\pi}$, and then

$$\mid E(\mathbf{F}_p \mid = p + 1 - (\pi + \overline{\pi})$$

.

3.4 Fast Point Addition and Multiplication

At this position, we will give some algorithms and methods to find a *random* point on a given elliptic curve and discuss the efficient fast point addition and

multiplication methods appeared in our computations later.

By Hasse's theorem, we can conclude that for a given q we have a range of $4\sqrt{q}$ about the value $q+1$. In order try to find a *random* point for a given elliptic curve over $\mathbf{F}_q$, we can use the following algorithm due to [7] with almost uniform distribution of elements of $\mathbf{F}_q$.

ALGORITHM: Determining a *random* point in $\mathbf{F}_q$
Input: An elliptic curve over $\mathbf{F}_q$,
Output: A 'random' point $P \in \mathbf{F}_q$,

1. Do the the following;
   (a) Pick a *random* $x \in \mathbf{F}_q$;
   (b) Substitute x for X in the Weierstrass form of the curve E;
   (c) Try to find out Y;
   (d) If such y's can be found, choose one and set $P = (x, y)$.
2. Until such a point P is found
3. Return P.

For prime fields $\mathbf{F}_p$, each orders occur with an almost uniform distribution for details see reference [43].

3.4.1 Point Addition

We will concentrate our attention only on $p > 3$ not for caharacteristics 2 or 3, as we do not need such characteristic in our furher discussions (for fields of characteristic 2 of cryptographic interest, there are analogus methods as we describe here for details see [7]). Our obeservations in point addtion and multiplication are also based on the results from [7] and its references.

Affine Coordinates

Review: As we already discussed for the fields characteristic is not equal to 2 or 3, for a given two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, given in affine coordinates, such that $P_1, P_2 \neq \mathcal{O}$ and $P_1 \neq -P_2$ (these two conditions can be trivially checked), we have $P_1 + P_2 = P_3 = (x_3, y_3)$ and this can be compute as follows

1. If $P_1 \neq P_2$,
   (a) $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$,
   (b) $x_3 = \lambda^2 - x_1 - x_2$,
   (c) $y_3 = (x_1 - x_3).\lambda - x_3 - y_1$.
2. If $P_1 = P_2$,
   (a) $\lambda = \frac{3x_1^2 + a}{2y_1}$,
   (b) $x_3 = \lambda^2 - 2x_1$,
   (c) $y_3 = (x_1 - x_3).\lambda - x_3 - y_1$.

- For $P_1 \neq P_2$, we have one field inversion and three field multiplications, which can be abbreviated by $1\mathcal{I} + 3\mathcal{M}$.
- For $P_1 = P_2$, we have one field inversion and four field multiplications, which can be abbreviated by $1\mathcal{I} + 4\mathcal{M}$.

Remark 3.13. We can neglect in this case the cost of field addition and multiplication by small constants (for example in the computation of λ for the case $P_1 = P_2$).

Projective Coordinates

As we already explained, too, a projective point $P = (X, Y, Z)$ satisfies the following Weierstrass equation

$$Y^2Z = X^3 + aXZ^2 + bZ^3.$$

If $Z \neq 0$, it corresponds to the affine point $(X/Z, Y/Z)$. Hence, it turns out that other projective representations can lead to more efficient implementations. In particular, we will prefer *'weighted'* projective representation, i. e. (X, Y, Z) will correspond to $(X/Z^2, Y/Z^3)$ whenever $Z \neq 0$. It is equivalent then to using projective curve of the form

$$Y^2 = X^3 + aXZ^4 + bZ^6.$$

Then the zero point $\mathcal{O} = (\gamma^2, \gamma^3, 0)$ for some $\gamma \in \mathbf{F}_q^*$. One can easily see that conversion from affine to projective is trivial, while conversion in the other direction costs $1\mathcal{I} + 4\mathcal{M}$.

The key observation in connection with weighted projective coordinates is that point addition can be done using field multiplication only, with no inversions required. The total costs is then $16\mathcal{M}$ (for details see [7])

3.4.2 Fast Point Multiplication

Here point multiplication on the group of rational points of elliptic curves is the special case of the problem of exponentiation of in general abelian groups (as we use multiplication symbol for general setting of abelian groups). Therefore, this problem corresponds to the related *shortest addition chain* for integers, i. e. starting from 1, and computing at each step the sum of two previous result, what is the least number of steps required to obtain k?
Certain characteristics of the elliptic curve version of the problem must also be taken into account to obtain faster computational methods and hence algorithms, although general methods of exponentiation can be used to solve point multiplication problem.

We will not give details in the analysis of these algorithms that we will give below. These can be found in [7] For the sake of correctness, when analyzing the complexity of the section, and for simplicity we will consider the case the

field of char. 2 which can easily be extended for our case $p > 3$, where p is a prime.

ALGORITHM: Point Multiplication: Binary Method
Input: A point P and an l-bit integer $k = \sum_{j=0}^{l-1} k_j 2^j$,
Output: $Q = [k].P$,

1. $Q \leftarrow \mathcal{O}$;
2. For $j = l - 1$; to 0 do;
   (a) $Q \leftarrow [2]Q$;
   (b) if $k_j = 1$ then $Q \leftarrow Q + P$;
   (c) $j \leftarrow j - 1$.
3. Return Q.

Our second method is *m-ary method*, where $m = 2^r$ for some $r \geq 1$ and hence binary method is the special case of this method corresponding to $r = 1$.

ALGORITHM: Point Multiplication: m-ary Method
Input: A point P and an integer $k = \sum_{j=0}^{d-1} k_j m^j$, $k_j \in \{0, 1, \cdots, m-1\}$
Output: $Q = [k].P$,

1. Precomputation
   (a) $P_1 \leftarrow P$;
   (b) For $i = 2$; to $m - 1$ do;
      i. $P_i \leftarrow P_{i-1} + P$ (we have $P_i = [i].P$);
      ii. $i \leftarrow i + 1$.
   (c) $Q \leftarrow \mathcal{O}$;
2. Main Loop
   (a) For $j = d - 1$; to 0 do;
      i. $Q \leftarrow [m]Q$ (this requires r doublings.);

ii. $Q \leftarrow Q + P_{k_j}$;

iii. $j \leftarrow j - 1$.

(b) Return Q.

It can be easily verified that the algorithm computes $[k].P$, following the Horner's rule

$$[m](\cdots[m]([m]([k_{l-1}]P) + [k_{l-2}]P) + \cdots) = [k]P.$$

The doubling in the main-loop can be exploited to obtain additional savings: by splitting the computation of $[m]Q$ into two different stages, one can skip also the multiples of P in the precomputation. this leads to an improvement on the m-ary method, this modified version of m-ary method will be our third algorithm.

ALGORITHM: Point Multiplication: m-ary Method
Input: A point P and an integer $k = \sum_{j=0}^{d-1} k_j m^j$, $k_j \in \{0, 1, \cdots, m-1\}$
Output: $Q = [k].P$,

1. Precomputation

   (a) $P_1 \leftarrow P$, $P_2 \leftarrow [2]P$;

   (b) For $i = 2$; to $(m-2)/2$ do;

      i. $P_{2i+1} \leftarrow P_{2i-1} + P_2$;

      ii. $i \leftarrow i + 1$.

   (c) $Q \leftarrow \mathcal{O}$;

2. Main Loop

   (a) For $j = d - 1$; to 0 do;

      i. If $k_j \neq 0$ then do

         A. Let s_j, h_j be such that $k_j = 2^{s_j} h_j$

         B. $Q \leftarrow [2^{r-s_j}]Q$ (this requires r doublings.);

         C. $Q \leftarrow Q + P_{h_j}$;

D. $j \leftarrow j - 1$.

ii. Else $s_j \leftarrow r$

A. $Q \leftarrow [2^{s_j}]Q$;

B. $j \leftarrow j - 1$.

(b) Return Q.

Note that a slightly modified version of horner's method proves the correctness of this algorithm, too. At this stage we will once more generalize the method and give the so-called *Sliding Window Method*.

ALGORITHM: Point Multiplication: Sliding Window Method
Input: A point P and an integer $k = \sum_{j=0}^{l-1} k_j 2^j$, $k_j \in \{0, 1\}$
Output: $Q = [k].P$,

1. Precomputation

(a) $P_1 \leftarrow P$, $P_2 \leftarrow [2]P$;

(b) For $i = 1$; to $2^{r-1} - 1$ do;

i. $P_{2i+1} \leftarrow P_{2i-1} + P_2$;

ii. $i \leftarrow i + 1$.

(c) $j \leftarrow l - 1$, $Q \leftarrow \mathcal{O}$;

2. Main Loop

(a) While $j \geq 0$ do

i. If $k_j = 0$ then

A. $Q \leftarrow [2]Q$,

B. $j \leftarrow j - 1$.

ii. Else do:

A. Let t be the least integer such that $j - t + 1 \leq r$ and $k_t = 1$

B. $h_j \leftarrow (k_j kj - 1 \cdots k_t)_2$,

C. $Q \leftarrow [2^{j-t+1}]Q + P_{h_j}$,

D. $j \leftarrow t-1$.

(b) Return Q.

It is time now to mention the so-called *Signed Digit representations*, it is the case that the subtraction in the group of rational pints of elliptic curves has virtually the same cost as addition. For canonical curve representations the negative of a point $P=(x,y)$ is $(x,x+y)$ in characteristic two, and $(x,-y)$ in odd characteristic. This leads us to reduce the number of curve operations by means of addtion-subtraction chains in point multiplication method.

Consider now integer representations of the form $k=\sum_{j=0}^{l} s_j 2^j$, where $s_j \in \{-1,0,1\}$. It is said to be then *binary signed digit*, abbreviated by SD, representation. This representation includes all integers $0 \leq k \leq 2^{l+1}-1$ along with their negatives. This redundancy can be traded off for a sparsity constraint which results more efficient point multiplication algorithms.

Definition 3.19. An SD representation is said to be *sparse*, if it has no adjacent non-zero digits, that is $S_j s_{j+1}=0 \ \forall \ j \geq 0$. A sparse SD representation is also called a *non-adjacent form*, denoted by NAF.

Lemma 3.4.1. *Every integer k has a unique NAF. The NAF has the lowest weight among all SD representations of k, and it is at most one digit longer than the shortest SD representations of k.*

Proof: See references also for more details [39], [38] Chapter 10, [37].

Our next algorithm computes the NAF of a non-negative integer given in binary representation.

ALGORITHM: Conversion to NAF

Input: An l-bit integer $k=\sum_{j=0}^{l-1} k_j 2^j$, $k_j \in \{0,1\}$

Output: $k=\sum_{j=0}^{l} s_j 2^j$, $s_j \in \{-1,0,1\}$

1. $c_0 \leftarrow 0$;

2. For $j=0$; to l do;

(a) $c_{j+1} \leftarrow \lfloor (k_j + k_{j+1} + c_j)/2 \rfloor$ (assuming that $k_i = 0$ for $i \geq l$);

(b) $s_j \leftarrow k_j + c_j - 2c_{j+1}$;

(c) $j \leftarrow j + 1$.

3. Return $(s_l s_{l-1} \cdots s_0)$.

It is by Morain and Olivos showed that NAFs have fewer zeros than binary representations, i.e. they proved that in expected weight of a NAF of length l is $l/3$. For details see [36].

Now we will give a slightly generalized algorithm sliding window method, namely *Signed m-ary Window Decomposition.*

ALGORITHM: Signed m-ary Window Decomposition
Input: An l-bit integer $k = \sum_{j=0}^{l-1} k_j 2^j$, $k_j \in \{0, 1\}$, $k_l = 0$
Output: A sequence of pairs $\{(b_i, e_i)\}_{i=0}^{d-1}$

1. $d \leftarrow 0$, $j \leftarrow 0$;

2. While $j \leq l$ do;

   (a) If $k_j = 0$

      i. $j \leftarrow j + 1$

   (b) Else do

      i. $t \leftarrow min\{l, j + r - 1\}$, $h_d \leftarrow (k_t k_{t-1} \cdots k_j)_2$.

      ii. If $h_d > 2^{r-1}$ then do:

         A. $b_d \leftarrow h_d - 2^r$,

         B. increment the number $(k_l k_{l-1} \cdots k_{t+1})_2$ by 1.

      iii. Else

         A. $b_d \leftarrow h_d$,

         B. $e_d \leftarrow j$, $d \leftarrow d + 1$, $j \leftarrow t + 1$

3. Return the sequence $(b_0, e_0), (b_1, e_1), \cdots, (b_{d-1}, e_{d-1})$

Proof of the correctness of algorithm: The correctness of the algorithm is verified inductively by ascerting the the condition

$$k = \sum_{i=0}^{d-1} b_i 2^{e_i} + \sum_{j'=j}^{l} k_{j'} 2^{j'}$$

each time the loop in Step 2 is checked. As $j > l$, the second sum of the above equation vanishes, giving the desired decomposition of k, then the proof is straightforward. The only observation here is that the condition in step **b.ii** holds, then **b.ii.A** substracts 2^{j+r} from the sum in equation and **b.ii.B** adds it back as we have $t = j + r - 1$.

Now after having the sequence of $\{(b_i, e_i)\}_{i=0}^{d-1}$, the modification of the sliding window method is easy to construct.

ALGORITHM: Signed m-ary Windows
Input: A point P, and such that $k = \sum_{i=0}^{d-1} b_i 2^{e_i}$
Output: $Q = [k]P$.

- `Precomputation`
  1. $P_1 \leftarrow P$, $P_2 \leftarrow [2]P$
  2. For $i = 1$ to $2^{r-2} - 1$ do
     (a) $P_{2i+1} \leftarrow P_{2i-1} + p_2$,
     (b) $i \leftarrow i + 1$
  3. $Q \leftarrow P_{b_{d-1}}$.
- `Main Loop`
  1. For $i = d - 2$ to 0
     (a) $Q \leftarrow [2^{e_{i+1}-e_i}]Q$,
     (b) If $b_i > 0$ then $Q \leftarrow Q + P_{b_i}$,
     (c) Else $Q \leftarrow Q - P_{-b_i}$,

(d) $i \leftarrow i - 1$.

2. $Q \leftarrow [2^{e_0}]Q$,
3. Return Q

Using the analysis analog to that of unsigned window scheme, one can derive the correctness of the algorithm for details see [7].

3.5 Point Counting and Other Problems

Remark 3.14. There are also other concepts of elliptic curves coming together with *primality proving algorithms and public key cryptography*, some of these are followings

- Determining the Group Order, *point counting algorithm*, of rational points of elliptic curves over finite fields. For these there are lots of approaches,
  - *Baby Step-Giant Step* **Schoof's Algorithms**, ant its variants e. g. due to Etkies' and Atkin's based on the variation of Hasse's theorem.
  - Counting the points by means of constructing non-supersingular elliptic curves with complex multiplication.
- Discrete Logarithm for elliptic curves, abbreviated by **ECDLP**, which gives a change to obtain analogus cryptographic protocols like *ElGamal and Diffie-Hellmann*. Note that according to Pohling-Hellmann approach solving DLP modulo n, where n is order of an abelian group $\mathcal{G}$,
  $n = p_1^{e_1}.p_2^{e_2} \cdots P_r^{e_r}$ is equivalent to solving each DLP modulo p_i, where p_i's are prime for $1 \leq i \leq r$.

Note: We will introduce the concepts of *Point Counting Problem* in details in Chapter 5 & 6. Because of the fact that primality proving algorithms of Goldwasser & Killian and Atkin's are principally based on the *Point Counting Problem* and hence Schoof's approach (or its variants) and CM-method, respectively.

We will also see computationally efficient methods of the above concept in a detail and try to get efficient methods in particular for curves with complex multiplication. Furthermore, we will also see how the results coming from the arithmetic of elliptic curves over $\mathbf{C}$ or $\mathbf{Q}$ and the results of Algebraic Number Theory that we developed in the former chapter will be applied in our primality testing and proving algorithms.

For more general treatment of the theory of arithmetic of elliptic curves and their usage in extensive areas of both applied and theoretical mathematics can be found in [6] and [5]. Furthermore, for the usage of elliptic curves particularly in Cryptography, Coding theory and Factorization of integers see [1] and [38].

chapter 4

Primality Testing and Proving Algorithms

In this chapter, we are going to introduce and explain different primality testing & proving algorithms. We will start with a historical prime listing method, namely sieve of *Eratoshenes*, which will give us a possibility up to a given bound, say it up to 10^{16}, to store prime numbers and therefore enables us to verify whether a given prime candidate N, $N < 10^{32}$, is prime by means of just checking whether it is divisible by any of the list element.

Question

- What is a *Primality Test / Primality Proof*?
- What is the meaning of *certificate* in our context?

Definition 4.1. 1. If our algorithm gives us a possibility to reprove the primality of the candidate mathematically then we say that our algorithm is a *Primality Proving algorithm*.

2. If with our algorithm it is not possible to recheck mathematically that our candidate is prime, then we say that our algorithm is a *Primality Testing algorithm*. In that case we have also two possibilities;

   (a) An algorithm is said to be a *True Primality Testing*, if it can determine with mathematical certainty that our candidate is prime.

   (b) An algorithm is said to be a *Probabilistic Primality Testing*, if our candidate is a *probable prime*.

3. If it is possible to reprove the primality of our prime candidate, then we say that it has a *(primality) certificate*.

4.1 Prime Number Generation

In this section, we will explain a very historical primality listing methods. For generalization and modern methods see [2].

ALGORITHM: Sieve of Eratosthenes
Input: A natural number B, $B > 2$, and the set $\mathcal{L} = \{2\}$
Output: Prime numbers between $[2, B]$,

1. List the numbers $n \in [2, B]$, for example $a_i = i$, $2 \leq i \leq B$ and initialise $p = 2$;
2. While $p < B$ do:
   (a) Start with p and delete all n's of the form $k.p$ with $k \geq 2$;
   (b) Find the smallest non-deleted number p' with $p' > p$, then $\mathcal{L} = \mathcal{L} \cup \{p'\}$;
   (c) $p \leftarrow p'$ and goto b;
3. Return $\mathcal{L}$.

Remark 4.1. One can improve the efficiency in the followings

- Considering just $i \in [2, B]$, or better the numbers coprime to $210 = 2.3.5.7$.
- One can also generalize the sieve to the intervals $[A, B]$, such as $[A, A+10^6]$.

4.2 Trial-division Method

The easiest way to test whether a number is prime or not is the so-called *trial division*. If a given number $N \in \mathbf{N}$ is not divisible by any of the primes $\leq \sqrt{N}$, then we can conclude that N is prime. (As if $N = a.b$, $1 < a \leq b < N \Rightarrow a^2 \leq a.b = N \Rightarrow p^2 \leq N$ for primes $p \mid a$) Of course, since the complexity is

$\mathcal{O}(\sqrt{N})$, it is not possible for big numbers to use this method (say it more than 100-digits).But one can use trial-division to factor an integer to test whether we have small factors or not. Therefore, ideally it can also be used at the first up to a given bound in before terminating our more advanced algorithms later. One can choose a bound B and by means of sieve of Eratosthenes store the primes in a list to see at the first if our prime candidate is divisible by any element of this list. Of course if it is the case, we can conclude that our candidate is composite and divisible by that element of our previously chosen list.

4.3 Fermat's Primality Test

According to the Fermat's last theorem, one can consider that whether we have somehow an inverse of Fermat so that we can apply a *Primality Testing* based on this inverse. But unfortunately such an inverse does not exist. But we can have a *compositeness Test* owing to the following Corollary of Fermat.

Proposition 4.1. A number N is composite if there exists $a \in N$ such that $a^{N-1} \not\equiv 1 \; mod(N)$.

Proof : trivial by Fermat's theorem.

Definition 4.2. The composite number N which passes Fermat's test is called *Carmichael* Number. The smallest such a number is $561 = 3.11.17$

Theorem 4.1. There are infinitely many Carmichael Numbers.

By above theorem it is clear that Fermat test can not guarentee at all whether the checked candidate is a true prime. However this test enables us to detect most of the composite numbers at the first. Furthermore, by means of generalization of Fermat's and trying to get a generalized version of inverse of Fermat's theorem will give us a change to introduce the $N-1$, $N+1$, and hence at the end Jacobi-sum and ECPP algorithms. There are also other *probabilistic primality tests* by means of generalizing Fermat. In the next section we will deal with such algorithms. They are also used with some modification in cryptographic protocols

when the true primality testing and proving algorithms are not available or not required to guarantee a reasonable security of the considered cryptosystem.

4.4 Probabilistic Primality Testing Algorithms

Now it is time to give the probabilistic primality tests, namely Solovay-Strassen and Miller-Rabin Tests. The idea is to use the definitions of pseudoprimality, euler pseudoprimality and at the end strong pseudoprimality.

4.4.1 Solovay-Strassen Probabilistic Primality Test

Now Solovay-Strassen Algorithm will be given. This and Miller-Rabin Algorithms are actually probabilistic algorithms. The answer `composite` is always true , hence they are called in some texts *Compositeness Algorithms*. Moreover, if the candidate is prime then the answer is always `'prime'`. However, it is possible to have a false answer `'prime'`, though the candidate may actually be `'composite'`.

ALGORITHM:Solovey-Strassen

1. $N \in N$, N is odd;
2. choose a number $t \in N$ randomly and for $1 \leq i \leq t$ choose a_i randomly;
3. For $1 \leq i \leq t$ compute
   (a) $c_i \equiv \left(\frac{a_i}{n}\right) mod(N)$ with reciprocity law
   (b) $b_i \equiv a_i^{\frac{N-1}{2}} mod(N)$;
4. if $b_i \neq a_i$ for some i,then Return: **'composite'** ;
5. else Return: **'possibly prime'**;

Proposition 4.2. Solovay-Strassen error-probability bound Let N be an odd composite integer. The probability that the Solovay-Strassen test declares N to be `'prime'` is less than $(\frac{1}{2})^t$.

4.4.2 Miller-Rabin Probabilistic Primality Test

We just generalize the above idea of Sollovay-Strassen by considering strong pseudoprimes instead of Euler pseudoprimes (because $spsp < epsp < psp$). We therefore introduce the Miller-Rabin Test, which is also called *strong pseudoprimality test*.

ALGORITHM:Miller-Rabin

1. $N \in N$, N is odd, set $N = 1 + 2^l.m$, where m is odd;
2. choose a number $t \in N$ randomly and for $1 \leq i \leq t$ choose a_i randomly;
3. Test for $1 \leq i \leq t$
   (i)whether $a_i^m \equiv 1\ mod(N)$
   (ii)or there exists j with $0 \leq j \leq l$ such that $a^{2^j.m} \equiv -1\ mod(N)$;
4. if both (i) & (ii) are false, Return:**'composite'**;
5. else Return: **'possibly prime'**;

Proposition 4.3. Miller-Rabin error-probability bound Let N be an odd composite integer. The probability that the Miller-Rabin test declares N to be `'prime'` is less than $(\frac{1}{4})^t$.

Remark 4.2. fixed bases in Miller-Rabin a strategy that is sometimes employed is to fix the bases a in the Miller-Rabin algorithm to be the first few primes (composite integers may be ignored using iteratively the Miller-Rabin), instead of choosing them at random. Note that trial division and the so-called prime list may be used at the first to reduce the unnecessary computation in Miller-Rabin algorithm, i. e. choose a bound B and list all primes $p < B$ and use at the first trial division to guarantee that the given candidate is not divisible by any of these primes $< B$. For instance, if one chooses $B = 256$, then %80 of composite odd numbers can be discarded before employing Miller-Rabin.

Remark 4.3. Let's compare Sollovay-Strassen & Miller-Rabin algorithms according to [2]:

1. The Soolovay-Strassen test is computationally more expensive.

2. The Sollovay-Strassen test is harder to implement as it also involves Jacobi symbol computations.

3. The error probability for Sollovay-Strassen is bounded by $(\frac{1}{2})^t$, while the error probability for Miller-Rabin is bounded by $(\frac{1}{4})^t$.

Therefore we can conclude that there is no need to use Sollovay-Strassen instead of Miller-Rabin.

4.5 $N-1$ Primality Testing Algorithms

There are also some primality testing algorithms based on the *inverse* of the Fermat's theorem. Indeed, our aim here is to get an inverse of Fermat by using some additional informations. Actually if we can factor the number $N-1$ completely or partially one can give a proof of primality with some extra informations. We will now give the most important proposition of both $N-1$ (resp. $N+1$) primality testing algorithm and our ECPP Algorithms.

Theorem 4.2. Lucas Let $a, N \in \mathbf{N}$ with $gcd(a, N) = 1$. If $a^{N-1} \equiv 1\ mod(N)$, but $a^{\frac{N-1}{d}} \not\equiv 1\ mod(N)$ for every divisor $d > 1$ of $N-1$, then N is prime.

Proofsee [17].

Theorem 4.3. Let $N \in \mathbf{N}$, $N-1 = \prod_{i=1}^{t} p_i^{e_i}$. If there exists $a \in \mathbf{N}$ with $a^{N-1} \equiv 1\ mod(N)$ but $a^{\frac{N-1}{p_i}} \not\equiv 1\ mod(N)$, then N is prime.

Proofsee [17].

Theorem 4.4. Let $N \in \mathbf{N}$, $2 \mid N$, and $N-1 = \prod_{i=1}^{t} p_i^{e_i}$ for $1 \leq i \leq t$. If there exists a_i with $a_i^{N-1} \equiv 1\ mod(N)$, $a_i^{\frac{N-1}{i}} \not\equiv 1\ mod(N)$, then N is prime.

Proof see [17].

Note that in order to apply any of the above theorem, we need the *complete* factorization of $N-1$. Mostly, it is not possible to have such complete factorization, but we have one of the divisor of $N-1$ which satisfies some properties then

we can also test the primality of the candidate N.

Pocklingston's theorem will be the center of such *partial* factorization of $N-1$, as we will see, we can get also depending on this theorem other testing criterias.

Proposition 4.4. Pocklington's Theorem Let p be a prime divisor N-1. Assume that we can find an integer a_p such that $a_p^{N-1} \equiv 1\ mod(N)$and $(a_p^{N-1} - 1, N) = 1$. Then if d is any divisor of N, we have $d \equiv 1\ mod(p^{\alpha_p})$, where p^{α_p} is the largest power of p which divides N-1.

proof: It is enough to look at all prime divisors of N.Now if p is a prime divisor of N, we have $a_p^{d-1} \equiv 1\ mod(d)$, since a_p is coprime to N hence to d. On the other hand , we have $a_p^{\frac{N-1}{p}} \not\equiv 1\ mod(d)$, as $(a_p^{N-1} - 1, N) = 1$. If e is the exact order of a_p modulo d, then we have e divides d-1 but not $\frac{(N-1)}{p}$ but e divides N-1. hence $p^{a_p} \mid e \mid d-1$, hence $d \equiv 1\ mod(p^{a_p})$.

Proposition 4.5. Assume that we can write $N-1 = F.U$ where $(F, U) = 1$, F is completely factored and $F > \sqrt{N}$. Then, if for each prime dividing N we can find an a_p satisfying the conditons of Pocklington, then N is prime. Conversely, if N is prime , for any prime p dividing N-1, one can find a_p satisfying the conditions of Pocklington.

Corollary 4.1. Assume that we can write $N - 1 = F.U$ where $(F, U) = 1$, F is completely factored, all the prime divisors of U are greater then B, and $B.F \geq \sqrt{N}$. Then, for each p dividing F we can find an a_p satisfying conditions of Pocklington, and if furthermore we can find an a_U cuch that $a_U^{N-1} \equiv 1\ mod(N)$ and $(a_U^F - 1, N) = 1$, then N is prime. Conversely, if N is prime, then a_p and a_U can be always found.

4.5.1 Test

Now we will give $N-1$ algorithm by means of a pseudocode. At the first we will give an algorithm for the case that we can have the full factorization of $N-1$, in the second case we concentrate our attention on partially factorized case.

ALGORITHM:$N-1$ Test (Fully factorized $N-1$)

Input: $N \in N$, N is odd, the bases a_i, $0 \leq i \leq r$, which passed Miller-Rabbin test,

Output: `'prime'`, or `'composite'`.

1. Set $i = 0$ (m_0 first base);
2. Determine s, t such that $N - 1 = m_i^{2^s} t$;
3. While $i < k$ do:
   (a) if $m_i^t \equiv 1 \; mod(N)$, then $i \leftarrow i + 1$,
   (b) while $k < s$ do:
      - If $m_i^{2^k} t \equiv -1 \; mod(N)$, $i \leftarrow i + 1$ and STOP,
      - $k \leftarrow k + 1$,
   (c) If $k = s$ return `'composite'`.
4. Return `'prime'`.

Now if we cannot factorize the $N-1$ fully we will just test the conditions of Pocklington in the following algorithm:

ALGORITHM:$N-1$ Test (Partially factorized $N-1$)

Input: $N \in N$, N is odd, P is a prime divisor of $N-1$, and $Q = \prod_{i=0}^{n} p_i^{q_i}$ is a divisor of $N-1$ with prime factors p_i's such that $N - 1 = Q.P$ with $Q < P$.

Output: `'prime'`, or `'composite'`.

1. Choose an natural number $a \in \mathbf{N}$ such that $1 < a < N$,
2. If $a^{N-1} \not\equiv 1 \; mod(N)$, then GOTO 1,
3. If $gcd(a^Q - 1, N) \neq 1$, then GOTO 1,
4. Return `'prime'`.

4.5.2 certificate

The so-called DOWN-RUN method to get the pairs (i, m_i), together with exponents we get the so-called *certificate*. In fact if we store these datas, for a second programmer it is possible to verify the primality or compositeness of the result once more.

4.5.3 Special primes

Easiest form is $N - 1 = 2^m$
Let $n \in \mathbf{N}$ be an odd integer. We know that

$$\frac{x^n - 1}{x - 1} = 1 + x + \cdots + x^{n-1}$$

Let us substitute x with $-x$, then we have
$\frac{(-x)^n-1}{-x-1} = 1 - x + \cdots + (-x)^{n-1} \Rightarrow \frac{x^n+1}{x+1} = 1 - x + \cdots + x^{n-1} \Rightarrow x + 1 \mid x^n + 1$, if we have $n \equiv 1\ mod 2$.
Let $m = bu$, where u is odd, then $x := 2^b \Rightarrow 2^b + 1 \mid 2^{bu} + 1 = N$. According to lemma 2.2.3 we have then together with theorem 3.3 that if $a \in \mathbf{Z}$ with $a^{F_n-1} \equiv 1\ mod(F_n)$, and $a^{(F_n-1)/2} \not\equiv 1\ mod(F_n)$, then we can conclude that F_n is prime, where $F_n = 2^{2^n} + 1$.

Theorem 4.5. Peppin Let $n \geq 1$. Then F_n is prime if and only if
$3^{2^{2^n-1}} \equiv -1\ mod(F_n)$.

Theorem 4.6. Let $n \geq 2$ and $p \mid F_n$, p is prime. Then we have $p \equiv -1\ mod(2^{n+2})$.

Note: Proofs of the above two theorem see [17].

With the help of the Poclington's theorem we can conclude also the following results

Theorem 4.7. Let $N = K.2^n + 1$, $2 \nmid K$, $0 < K < 2^n + 2$. Then N is prime if and only if $\exists\ a \in \mathbf{Z}$ with $a^{\frac{N-1}{2}} \equiv -1\ mod(N)$.

Theorem 4.8. Let $N = K.2^n + 1$, $n \geq 2$, K is odd and $3 \nmid K$. Furthermore, assume that we have $0 < K < 2^n + 2$. Then N is prime if and only if $3^{\frac{N-1}{2}} \equiv -1 \; mod(N)$.

4.6 $N + 1$ Primality Testing Algorithms

Our aim is now to find a similar Test for $N + 1$, when the factorization of $N - 1$ is not partially or fully possible.

Remark 4.4. Let p, q be two integers, so that $p^2 - 4.q$ is not a square, then $x^2 + px + q$ have two distinct zeros $r_{1,2} = \frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}$. If we consider r_1 we get the following recursive formula ;

Proposition 4.6. Powers of r can be given in the following way

$$r_1^m = \frac{V(m) + U(m).\sqrt{p^2 - 4q}}{2}$$

where $V(m)$ and $U(m)$ can be calculated recursively as follows :

$$U(0) = 0, U(1) = 1, U(m) = pU(m-1) + qU(m-2),$$

$$V(0) = 2, V(1) = p, U(m) = pV(m-1) - qV(m-2),$$

U and V are lucas sequences of p and q.

Proof: see see [17].

Proposition 4.7. *(Lucas Test)* Let N be an odd integer. If there exists δ with $(\frac{\delta}{N}) = -1$ and for every prime factor r of N+1 there exist some p and q with $p^2 - 4q = \delta$ such that

$$U(N+1) \equiv 0 \; mod(N) \; \& \; U(\frac{n+1}{r}) \not\equiv 0 \; mod(N)$$

are satisfied, then N is prime.

Proof: see [17].

4.6.1 Test

Now we will give $N-1$ algorithm by means of a pseudocode (only for partially factorized case).

ALGORITHM:$N+1$ Test (Partially factorized $N+1$)
Input: $N \in N$, N is odd, P is a prime divisor of $N+1$, and $Q = \prod_{i=0}^{n} p_i^{q_i}$ is a divisor of $N+1$ with prime factors p_i's such that $N+1 = Q.P$ with $Q < P$.
Output: `'prime'`, or `'composite'`.

1. Choose r, s such that $gcd(r, s) = 1$,
2. If $(\frac{r-4s}{N}) \neq -1$, then GOTO 1,
3. If $V(\frac{N+1}{2}) \not\equiv 0 \; mod(N)$, then GOTO 1,
4. If $V(\frac{Q}{2}) \not\equiv 0 \; mod(N)$, then GOTO 1,
5. Return `'prime'`.

4.6.2 certificate

As in the case of $N-1$, we can get a certificate for $N+1$ tests in terms of the parameters r, s. As we know that here r depends on the number s, we can choose also $r = 1$ if s is even, and $r = 2$ if s is an odd integer.

4.6.3 Special primes

We know by Corollary 2.2 that if $N = 2^n - 1$ is prime, then n must also be prime. Recall that we say such primes *Mersenne Primes*. One can use the so-called $N+1$ tests much more efficiently.

4.7 ECPP Algorithms

As we have already seen from the above two sections, if we have the partial factorization of $N-1$ or $N+1$, there is a version of *inverse of Fermat's little*

theorem. With some additional requirements it is not actually a difficult problem to be able to test the primality of a given number. It is also clear that these algorithms are somewhat special, as they need the factorization of $N-1$ resp. $N+1$. The first general purpose primality **proving** algorithm was introduced in 1979 by Addleman, Pomerence and Rumely. This algorithm is called *Jacobi Sum Test* and based on the group rings of cyclotomic extensions. The idea of this test is therefore also to do with a generalization of Fermat's little theorem. The running time of this algorithm is $\mathcal{O}((logN)^{c.log\ log\ logN})$ for some constant $c > 0$. Hence, it is almost a polynomial time algorithm. However the algorithm was not practical. Cohen and Lenstra developed a practical version of this algorithm and is of use also in practice.

The reason why ECPP Algorithms are superior in practice is that it gives a *short primality certificate* (or *certificate of primality*), that is once given the parameters of the algorithm it is much more easy for the second party to verify the result (in our case *prove* or *disprove* the primality of the number which was considered to be prime). Jacobi Sum test cannot give such a certificate, that is second programmer has to write and execute the entire test once more, though it is slightly faster than our ECPP Algorithms up to 600-700 digits.

4.7.1 Test

Here we will explain the algorithm based on elliptic curves over finite fields, instead of using the suitably strong generalizations of Fermat's theorem, we will use the group of rational points of elliptic curves over $\mathbf{F}_N$ itself. Now at the first step we will morally certain that our number N is prime, i. e. it has passed the probabilistic primality proving tests such as Sollavay-Strassen and Miller-Rabin. Hence we will work as if N was a prime, assuming for example that each non-zero element modulo N is invertible. In the event that there exist some non-invertible elements modulo N, we will immediately stop the algorithm and give a non-trivial factor of N by taking a GCD with N (for example by using extended GCD Algorithm). We will therefore consider an elliptic curve over $\mathbf{Z}/N\mathbf{Z}$. It

means that we consider a Weierstrass equation $y^2 = x^3 + ax + b$ $a, b \in \mathbf{Z}/N\mathbf{Z}$, $(4a^3 + 27b^2) \in (\mathbf{Z}/N\mathbf{Z})^*$ (it is not necessary to consider general Weierstrass equation as $(N, 6) = 1$). Furthermore, we will perform the group operation of rational points of our elliptic curve as if N was a prime. Hence, we can assume from now on that all computations can be performed without any problems.

Our basic strategy in our ECPP Algorithm is the so-called DOWN-RUN strategy of the following theorem, which is the elliptic curve analog of the theorems of $N - 1$ and $N + 1$ tests.

Theorem 4.9. Let N be a positive integer coprime to 6 and different from 1. Let E be an elliptic curve modulo N. Assume that we know an integer m and a point $P \in E(\mathbf{Z}/N\mathbf{Z})$ satisfying the following conditions.

1. There exists a prime divisor q of m such that

$$q > (\sqrt[4]{N} + 1)^2$$

2. $m.P = O_E = (0 : 1 : 0)$

3. $(\frac{m}{q}).P = (x : y : t)$ with $t \in (\mathbf{Z}/N\mathbf{Z})^*$

Then N is prime.

proof: Assume that N was not a prime Let p be the smallest prime divisor of N in $E(\mathbf{Z}/p\mathbf{Z})$,the image of P has order a divisor of m but not divisor of m/q as $t \in (\mathbf{Z}/p\mathbf{Z})^*$. Since q is a prime q divides the order of the image of P in $E(\mathbf{Z}/p\mathbf{Z})$, i.e. $q \leq \mid E(\mathbf{Z}/p\mathbf{Z}) \mid$. By Hasse's Thm. $q < (\sqrt{p} + 1)^2$. Since $p \leq \sqrt{N}$ (as being the smallest prime by assumption) we get $q < (\sqrt[4]{N} + 1)^2$ which is trivially a contradiction to our assumption.

Now we are facing with 3 main problems to solve here namely,

- How to choose the elliptic curve?
- How to find P?

- How to choose $m \in \mathbf{N}$?

Now the following proposition gives the answer of the second and third question.

Proposition 4.8. Let $m =\mid E(\mathbf{Z}/p\mathbf{Z}) \mid$. If for a prime number q dividing m

$$q > (\sqrt[4]{N} + 1)^2$$

is satisfied then there exists a point $P \in E(\mathbf{Z}/N\mathbf{Z})$ s.t. $m.P = O_E = (0 : 1 : 0)$ and $(\frac{m}{q}).P = (x : y : t)$ with $t \in (\mathbf{Z}/N\mathbf{Z})^*$. Note that $m =\mid E(\mathbf{Z}/p\mathbf{Z}) \mid$ can be computed as if N was a prime.

The ECPP (elliptic curve primality proving) algorithms is given then as follows;

ALGORITHM:ECPP
INPUT: a number $N \in \mathbf{Z}$, whose primality will be (dis)proved.
OUTPUT: If N is composite , a divisor of N, if N is prime return `'prime'`.

1. choose a non-supersingular elliptic curve E over $\mathbf{Z}/N\mathbf{Z}$
2. $m \leftarrow\mid E \mid$;
3. choose a prime number $q > (\sqrt[4]{N} + 1)^2$ such that $q \mid m$;
4. Choose a **'random'** point $P \in E$;
   4.1 If $m.P \neq 0$ go to 4;
   4.2 If $\frac{m}{q}.P = 0$ go to 4;
   4.3 if there exists an error return the divisor of N (extended GCD algorithm)
5. return `'prime'`.

As we see in this algorithm, we reached an algorithm analog to the one that we developed in $N-1$ and $N+1$ tests. As we already explained these tests are *special* since they need the partial or full factorization of $N-1$ or $N+1$. However, if we change the group $\mathbf{F}_N$ with the group of elliptic curves over $\mathbf{F}_N$, we can reach a general primality proving algorithm, which also, in contrast to Jacobi sum test

and its variants, gives us a chance to reprove or disprove the result that we have tested. For further discussions on *short prime certificates* see the next chapter. Further, the idea is unchanged, namely the so-called DOWN-RUN strategy. We get intermediate *prime candidates*, and applying the DOWN-RUN strategy, we get the following;

Remark 4.5. The primality of a number N can be reduced to the primality of the smaller prime candidates $q < N$ by means of the above ECPP Algorithm. Moreover, applying this algorithm recursively, we can prove the primality of the given prime candidate N just by trial-division or $N - 1$ test, as we can get an intermediate prime candidate $q < 10^{16}$.

After introducing our algorithm, we have to deal with the following problems, which are actually coming together with the results of the theory of arithmetic of elliptic curves over $\mathbf{C}$ or $\mathbf{F}_N$ in Chapter 3, and the results from Number Theory and Algebraic Number Theory in Chapter 2.

- How can we choose elliptic curves?
- How can one compute the cardinality of the group of rational points of elliptic curves over finite fields, whose characteristic is **'big'**?
- Is this algorithm practical?

Our basic difficulty in our algorithm is to find $m =\mid E(\mathbf{Z}/p\mathbf{Z}) \mid$. We will discuss in detail the methods (both theoretic and practical) in the next chapter. After that we will give the complexity analysis of the Algorithm in chapter 6 and in chapter 7 we will give the implementation details.

As we already see, we need in this case also an elliptic curve E over $\mathbf{Z}_N$ with the property that it has a prime divisor, which satisfies the condition $> (\sqrt[4]{N}+1)^2$. Furthermore, we need also a point P on this curve which satisfies the conditions of the Theorem 4.9. That means we have a probabilistic algorithm, although the result is always true and reprovable. Note that in case N is not a prime, it is

not possible to terminate the algorithm, but if we insert the condition of GCD with the intermediate result, we will get a divisor of N and which also gives us a chance to detect the divisor of N, for the case N is composite.

4.7.2 certificate

As we already witnessed in $N-1$ and $N+1$ tests, we can apply in ECPP algorithm also the so-called DOWN-RUN strategy here we get recursively the intermediate prime candidates:

$$N_0 = N, N_1(= q), \cdots, N_i, \cdots$$

Definition 4.3. $N_0 = N(= q_0), N_1(= q_1), \cdots, N_i(= q_i), \cdots$ with corresponding elliptic curves E_i and the cardinalities m_i, are called **primality certificate** by means of ECPP.

In contrast to the Jacobi sum test, we can get with these certificate candidates a *certificate* algorithm, which verifies the result in a much more little time than original algorithm.

Remark 4.6. As we already explained this, algorithm gives a *short certificate* by means of m_i's . It gives a possibility that anybody can prove to his or her satisfaction the primality of N using much less work than executing the original Algorithm.

4.8 Primality Testing in reality

We can of course combine all of the algorithms discussed in this chapter so as to obtain a true primality testing method, which may be used in cryptographic protocols such as Diffie-Helmann and ElGammal.

ALGORITHM:A Combined Primality Testing

INPUT: a number $N \in \mathbf{Z}$, whose primality will be (dis)proved,

OUTPUT: If N is composite , a divisor of N, if N is prime return `'prime'`.

1. Test, if N is coprime to 30. If not return $\{$`'`N`' has a divisor` $gcd(N, 30)\}$;
2. Test, if N is divisible by any elements p of prime list up to $< 10^{16}$ (trial division) if yes return $\{$`'`N`' has a divisor` $p\}$;
3. Test, whether $2^{N-1} \equiv 1 \ mod(N)$ (*almost* all composite number can be therefore detected), if not return $\{$`'`N`' has a divisor` $p\}$;
4. Factorise $N-1$ and terminate $N-1$ algorithm, if we have answer `'composite'` return $\{$`'`N`' has a divisor` $p\}$;
5. Factorise $N+1$ and terminate $N+1$ algorithm, if we have answer `'composite'` return $\{$`'`N`' has a divisor` $p\}$;
6. Terminate Miller-Rabbin Algorithm with a given bound (parameter), if we have answer `'composite'` return $\{$`'`N`' has a divisor` $p\}$;
7. terminate ECPP algorithm, if we have answer `'composite'` return $\{$`'`N`' has a divisor` $p\}$;
8. return $\{$`'prime'`$\}$.

Note that one can also use in step 7 Jacobi sum test, if there is no need to certify the primality of the number. The algorithms of Jacobi sum and ECPP have been also combined methods based on the concept of *'dual elliptic primes'*(see [28]).

CHAPTER 5

ECPP (Elliptic Curve Primality Proving) ALGORITHMS

We have basically explained the theory of primality testing algorithms and ECPP algorithms in the last chapter. Our basic problem now is to determine, as we discussed, the order of the group of rational points of elliptic curves over finite fields. One method is due to Schoof. His algorithm is a Baby Step-Giant Step algorithm based on the theorem of Hasse, which computes $m =\mid E(\mathbf{Z}/p\mathbf{Z}) \mid$ in time $O(log^8 N)$. The ECPP Algorithm, which uses the Schoof's algorithm so as to find $m =\mid E(\mathbf{Z}/p\mathbf{Z}) \mid$, is due to Goldwasser and Kilian. It was showed that under reasonable hypothesis on the distribution of prime numbers in short intervals, the expected running time of the algorithm is $O(log^{12} N)$, hence is polynomial in $logN$.

The theoretical advance has been made by Adleman and Huang, by proving the following theorem.

Theorem 5.1. There exists a probabilistic polynomial time algorithm which can prove or disprove that a given number N is prime.

Their idea is to use, in addition to elliptic curves, Jacobians of curves of genus 2, and a similar algorithm to the one like Goldwasser and Kilian. Although, their algorithm is not practical, they proved the above theorem, see [25], i. e. they found an algorithm which runs in polynomial time. Of course both Goldwasser and Kilian and Jacobi Sum tests are not of that type since only the expected running times are polynomial, but the worst case may not be!

5.1 Goldwasser-Kilian ECPP Algorithm

Goldwasser & Killian used the same idea of the ECPP-Algorithm that we explained in last chapter. They use the so-called *Schoof Algorithms* to find the cardinality of the group of $E(\mathbf{F}_N)$.

5.1.1 Schoof's Algorithms

Problem: To find $\mid E \mid$. For every point $P \in E(\mathbf{F}_q)$ with $n = ord(P)$ we have a $k \in \mathbf{N}$ such that $k.n = \mid E \mid$.
$\Rightarrow$ By Hasse's theorem $k.n \in [q+1-2\sqrt{q}, q+1+2\sqrt{q}] =: I$.

Observation: If $ord(P) > 4\sqrt{q}$, then there exists exactly one $k \in \mathbf{N}$ such that $k.n \in I$ (and hence $k.n = \mid E \mid$).

Baby-Step Giant-Step Algorithm

Idea:

- Choose a 'random' point $P \in E(\mathbf{F}_q)$ and verify that (in apprx. $\sqrt{q}$ steps) $ord(P) > 4\sqrt{q}$.
- Determine the unique number $k.n \in I$ with $k.n.P = \mathcal{O}$ (and hence $\mid E \mid$).

ALGORITHM:A Baby-Step Giant-Step

1. **Initial Step**;
   - Set $h := 2\sqrt{q}$, hence $\mid E \mid \in [q+1-h, q+1+h]$,
   - Define $t' := trace(\phi) - h$. Then $\mid E \mid = q+1-trace(\phi) = q+1-h-t'$ with $t' \in [0, 2h]$,
   - Set $m := \lceil \sqrt{2h} \rceil = \lceil 2\sqrt[4]{q} \rceil \Rightarrow \exists a, b \in \{0, 1, \cdots, m-1\}$ such that $t' = am + b$.
2. **Baby-Step**

(a) For $b < m$:

i. compute $b.P$ and store in a list,

ii. $b \leftarrow b + 1$.

3. **Giant-Step**

(a) For $b < m$:

i. Compute $(q + 1 + h - am).P$,

ii. Compare, if $(q+1+h-am).P = b.P$. If so return $\{t' = am+b\}$,

iii. $b \leftarrow b + 1$.

4. Return $\{q + 1 - t' + h\}$.

By the construction we have $am + b = t' \in [0, 2h] \Rightarrow trace(\phi) = t' - h \Rightarrow$ $\mid E \mid = q + 1 - trace(\phi) = q + 1 - t' + h$, which proves hence the correctness of the above algorithm.

Schoof's Algorithm

In this case we will try to find the trace $t = trace(\phi)$, where ϕ the Frobenious. As we know from the functional equation $\mid E(\mathbf{F}_q) \mid = q + 1 - t$.

Idea: Compute at the first t modulo p for all p prime.
Chinese Remainder Theorem: throughout the equations $t\ mod(p_i)$ for $i = 1, \cdots, k$, we can determine $t\ mod(p_1 \cdots p_k)$ uniquely.

We know that $t \in [-2\sqrt{q}, 2\sqrt{q}]$ (by Hasse) so that t is uniquely determined by $tmod(p_i)$ for $i = 1, \ldots, k$ if $p_1 \cdots p_k > 4\sqrt{q}$. It is meaningfull to start with

$$p_1, p_2, p_3, \cdots = 2, 3, 5, \cdots$$

ALGORITHM: Schoof's Algorithm

1.Step: Trying to find a p-torsion point P on E, where p is a prime ($P \neq 0$), not neceserally $P \in E(\mathbf{F}_q)$. We can find this point by means of modular polynomials.
Let $[m] : E \to E$ is a morphism, i. e. there exist homogene polynomials θ_m, w_m, ψ_m such that $[m].(P) = (\theta_m : w_m : \psi_m)$.

These polynomials are known and written recursively as (assuming `W.L.O.G.`) $z = 1$:
$\psi_0(X,Y) = 0$, $\psi_3(X,Y) = 3X^4 + 6aX^2 + 3bX - a^2$
$\psi_1(X,Y) = 1$, $\psi_4(X,Y) = (2X^2 + 10bX^4 - 10a^2X^2 - 2baX - 2a - b^2)\psi_2$
$\psi_2(X,Y) = Y$, $\psi_{2m+1}(X,Y) = \psi_{m+2}.\psi_m^3 - \psi_{m-1}\psi_{m+1}^2$, $m > 0$
$\psi_{2m}(X,Y) = \frac{1}{\psi_2}(\psi_{m+2}\psi_{m+1}^2 - \psi_{m-2}\psi_{m+1}^2)\psi_m$.

P is a p-torsion point $\Leftrightarrow [P](P) = \mathcal{O} = (0 : 1 : 0)$ is the unique point at infinity $\Leftrightarrow \psi_p(P) = 0$.
In fact: P is a p-torsion point on $E \Leftrightarrow Y^2 = X^3 + aX + b$ and $\psi_m(X,Y) = 0$. `W.L.O.G.` we can assume that ψ_m is linear (substitude the non-linear terms with the first equation). Then we can solve ψ_m with respect to Y, namely $h_m(X)$.
Therefore: $P = (x : y : 1) \in E$ with $p.P = \mathcal{O} \Leftrightarrow h_m(x) = 0$ & $Y^2 = X^3 + aX + b$.
2.Step: For $\psi = 0, \cdots, p-1$, test whether now $\psi\phi(P) = \phi^2(P) + q.P$, where ϕ is Frobenious and P is p-torsion point.

If **YES**: $t = \psi \ mod(p)$ then by Corollary 2.1 part 2, we have a functional equation

$$\phi^2 + q - t\phi \equiv 0$$

so we have $\psi\phi(P) = \phi^2(P) + q.P = t.\phi(P) \Rightarrow \phi((\psi - t)p) = 0 \Rightarrow (\psi - t)p \in Ker(\phi)$

$$Ker(\phi) = \{(x : y : z) \mid (x^q, y^q, z^q) = (0 : 1 : 0)\} = \{(0 : 1 : 0)\} = \{\mathcal{O}\}$$

$\Rightarrow (\psi - t).p = 0 \Rightarrow p \mid \psi - t \Rightarrow t \equiv \psi \ mod(p)$.

Hence we get t modulo p, then by means of Chinese Remainder Theorem (CRT), we will get $t\ mod(p_1 \cdots p_k)$, and hence $\mid E \mid = q + 1 - t$. Note that this algorithm was improved by Atkin and Elkies for details see [29], [48].

5.1.2 ECPP Algorithm (Goldwasser-Kilian)

In this section, we will give the algoritm due to Goldwasser & Kilian in a pseudocode due to Cohen [1].

ALGORITHM:ECPP Algorithm (Goldwasser-Kilian)
INPUT: a number $N \in \mathbf{Z}$, whose primality will be (dis)proved,
OUTPUT: If N is composite , a divisor of N, if N is prime return *TRUE*.

1. **Initialize** Set $i \leftarrow 0$ and $N_i \leftarrow N$.

2. **Is N_i small?** If $N_i < 2^{16}$, trial divide N_i by the primes from the list up to 2^{15}. If N_i is not prime GOTO step 9.

3. **Choose a random curve** Choose a and b at 'random' in $\mathbf{Z}/N_i\mathbf{Z}$, and check that $4a^3 + 27b^2 \in (\mathbf{Z}/N_i\mathbf{Z})^*$. Let E be the elliptic curves whose affine Weierstrass equation is $y^2 = x^3 + ax + b$.

4. **Use Schoof** Using Schoof's Algorithm, compute $m \leftarrow \mid E(\mathbf{Z}/N_i\mathbf{Z}) \mid$. If Schoof's algorithm fails GOTO step 9.

5. **Is m OK?** Check whether $m = 2q$ where q passes the Miller-Rabbin test (or more generally, trial divide m up to a small bound, and check that the remaining factor q passes the Miller-Rabbin test and is larger than $(\sqrt[4]{N_i} + 1)^2)$. if this is not the case GOTO step 3.

6. **Find** P Choose at 'random' $x \in \mathbf{Z}/N_i\mathbf{Z}$ until the Legendre (or Jacobi) symbol $(\frac{x^3+ax+b}{N_i})$ is equal to 0 or 1 (this will occur after a few trial at most). Then compute $y \in \mathbf{Z}/N_i\mathbf{Z}$ such that $y^2 = x^3 + ax + b$. if there is a failure GOTO step 9.

7. **Check P?** Compute $P_1 \leftarrow m.P$ and $P_2 \leftarrow (m/q).P$. If during the computations some division is impossible GOTO 9. Otherwise, check that $P_1 = \mathcal{O}_E$, i. e. that $P_1 = (0 : 1 : 0)$ in projective coordinates. If $P_1 \neq \mathcal{O}_E$, GOTO step 9. finally if $P_2 = \mathcal{O}_E$, GOTO step 6.

8. **Recurse** Set $i \leftarrow i + 1$ and $N_i \leftarrow q$ and GOTO step 2.

9. **Backtrack** (*We are here only when N_i is not prime, which is unlikely occurence.*) If $i = 0$, output a message saying that N is `'composite'` and terminate the algorithm. Otherwise, set $i \leftarrow i - 1$ and GOTO step 3.

Remark 5.1. As stated in the algorithm, if N is not a prime, the algorithm may run indefinitely and so should perpaps not be called *algorithm* in this sense.

5.1.3 certificate

The results that we introduced in the last chapter can be without any reserve applicable.

5.2 Atkin's ECPP Algorithm

We see in this chapter and chapter 3 that the basic problem to be able to deal with our primality proving algorithms is that we have to find the size of the group of rational points of elliptic curves over a finite field $\mathbf{F}_q$. This problem is solved in Goldwasser-Kilian algorithm using the theoretical algorithm due to Schoof. However, Schoof's Algorithm and its variants seem almost impossible to implement. We will therefore use the properties of elliptic curves over finite fields related to complex multiplication, that we introduced in chapter 3.

5.2.1 Generating Elliptic Curves with Complex Multilication

As we already defined in chapter 3, an elliptic curve has *Complex Multiplication* (CM), if $End(E)$ is strictly bigger than $\mathbf{Z}$. By theorem 3.10, $j(\tau)$ is an algebraic

integer of degree h_D. If $\mathbf{Z}[\tau]$ is the maximal order of some imaginary quadratic number field $\mathcal{K}$, then $H = K(j(\tau))$ is an extension of $\mathcal{K}$ of degree h_D. This is actually the maximal unramified abelian extension of $\mathcal{K}$. As we already explained in theorem 3.10, H is called the *Hilbert class field* of $\mathcal{K}$, i. e. it is a field under which every ideal in $\mathbf{Z}[\tau]$ will become principal when considered as an ideal in $\mathbf{Z}_H$.

It is required to find the Hilbert class polynomial, $H_D(x)$, of theorem 3.10. An intermediate approach will here be used to do this. Set $q = e^{2i\pi\tau}$, and

$$\Delta(\tau) = q\left(1 + \sum_{n\geq 1}(-1)^n\left(q^{n(3n-1)/2} + q^{n(3n+1)/2}\right)\right)^{24}.$$

This can be computed as if it was written. Then we will use the well-known theorem on modular forms that

$$g_2^3 - 27g_3^2 = \left(\frac{2\pi}{w_2}\right)^{12}\Delta.$$

Now the formula that we will use for computing $j(\tau)$ is

$$j(\tau) = \frac{(256f(\tau)+1)^3}{f(\tau)},$$

where $f(\tau) = \frac{\Delta(2\tau)}{\Delta(\tau)}$. Now we will give the algorithm to find the Hilbert class Polynomial due to Cohen.

ALGORITHM: Hilbert Class Polynomial

Input: Given a negative discriminant D

Output: The monic polynomial of degree h_D in $\mathbf{Z}[X]$ of which $j((D+\sqrt{D})/2)$ is a root.

1. **Initialize** Set $P \leftarrow 1$, $b \leftarrow Dmod(2)$ and $B \leftarrow \lfloor\sqrt{\mid D \mid /3}\rfloor$;
2. For $j = 0$; to l do;
3. **Initialize** a Set $t \leftarrow (b^2 - D)/4$ and $a \leftarrow max(b, 1)$;

4. **Test** If $a \nmid t$ GOTO step 4. Otherwise compute $j \leftarrow j((-b+\sqrt{D})/(2a))$ using the above formulas. Now if $a = b$ or $a^2 = t$ or $b = 0$ set $P \leftarrow P.(X - j)$, else set $P \leftarrow P.(X^2 - 2Re(j)X + \mid j \mid^2)$;

5. **Loop on** a Set $a \leftarrow a + 1$. If $a \leq t$, GOTO step 3;

6. **Loop on** b Set $b \leftarrow b + 2$, if $b \leq B$ GOTO step 2, otherwise coefficients of P to the nearest integer, output P and terminate the algorithm.

We need to state a remark due to Cohen:

Remark 5.2. The final coefficients of P (known to be integers) must be computed with an error at most 0.5 For this, we need to make a priori estimate on the size of coefficients of P. In practise, we look at the constant term, which usually not far from being the largest. This term is equal to the product of values $j((-b+\sqrt{D})/(2a))$ over all reduced forms (a, b, c), and the modulus of this is approximately equal to $e^{\pi\sqrt{|D|}/(2a)}$ hence the modulus of the constant term is relatively close to 10^k, where

$$k = \frac{\pi\sqrt{\mid D \mid}}{ln(10)} \sum \frac{1}{a},$$

where the sum running over all reduced forms (a, b, c) of discriminant D.

Construction of elliptic curves with CM

Instead of taking 'random' elliptic curves as in Goldwasser-Kilian algorithm, we will choose elliptic curves with complex multiplication by an order of an imaginary quadratic number field $\mathcal{K} = \mathbf{Q}(\sqrt{D})$ where N, our prime candidate, splits as a product of two elements. This will enable us apply the theorem 3.15 due to Deuring which will give us immediately the cardinality of $E(\mathbf{Z}/N\mathbf{Z})$.

In this case we will work as if N was prime, too. We must find a negative discriminant D such that N splits as a product of two elements (so as to ensure ourselves that the curve is not supersingular), and hence N is not inert in $\mathcal{K}$. This can be acchieved by means of Cornacchia's algorithm of Chapter 2. Applying

this algorithm repetitively to get a solution of $x^2 + Dy^2 = 4N$ we can find such a discriminant D. Once such a pair (x, y) is found than we have

$$\pi = \frac{x + y\sqrt{D}}{2},$$

now by applying Deuring, we get

$$m =\mid E(\mathbf{Z}/N\mathbf{Z} \mid= N + 1 - \pi - \bar{\pi} = N + 1 - x.$$

We know that $m = p + 1 - t$, where t is the trace of Frobenius. Recall that $t = \pi + \bar{\pi}$, where π is an element of norm N. A solution by means of Cornacchia of $x^2 + Dy^2 = 4N$ means that

$$\pi = \pm\frac{x + y\sqrt{D}}{2}.$$

Then, the order $N + 1 + x$ will be the order of quadratic twist of E. Hence, by theorem 3.14 in general case we have just these two elliptic curves up to isomorphism (E and its quadratic twist), which actually proves the following lemma:

Lemma 5.2.1. *Suppose E and E' have the same j-invariant but not isomorphic to each other over a field $\mathcal{F}_p$, where p is a prime. If $j \neq 0$ and $j \neq 1728$, then E' is quadratic twist of E and if $E = p + 1 - t$ and $E' = p + 1 + t$.*

Proposition 5.1. Let $w(D)$ be the numbers of roots of unity in the imaginary quadratic order of discriminant D, hence $w(D) = 2$ if $D < -4$ (as in the above case!), $w(-4) = 4$ and $w(-3) = 6$. Then there exist exactly $w(D)$ isomorphism classes of elliptic curves modulo N with CM by the imaginary quadratic order of discriminant D

Proof: immediate from thm. 3.14.

Above proposition corresponds to the factorisation $N = (\zeta\pi)(\bar{\zeta\pi})$, where ζ runs over all $w(D)$-th roots of unity (note that this will in paricular correspond to the above situation $\zeta = \pm 1$ if $D < -4$).

Our second aim is to write down explicitely the equation of these elliptic curves. Since N splits in the order of discriminant D, we have $w(D) \mid N-1$ and there exist $(N-1)/2$ values of $g \in \mathbf{Z}/N\mathbf{Z}$ ($(N-1)/3$ if $D=-3$) such that $g^{(N-1)/p} \neq 1$ for each prime dividing $w(D)$. Choosing one values of g we get the following equations of elliptic curves

Lemma 5.2.2. *Let $c = j/(1728-j)$, where $j = j(\frac{D+\sqrt{D}}{2})$.*

1. *If $D < -4$ we have the following affine equation of ellitic curve E*
   $y^2 = x^3 - 3cg^{2k}x + 2cg^{2k}$ *for $K = 0$ or 1.*

2. *If $D = -4$ we have*
   $y^2 = x^3 - g^k x$ *for $0 \leq k \leq 3$.*

3. *If $D = -3$ we have*
   $y^2 = x^3 - g^k$ *for $0 \leq k \leq 5$.*

Now at the end we will explain how we can find the roots of Hilbert class polynomials over $\mathbf{Z}/N\mathbf{Z}$. This is the problem of factoring, and hence finding the roots of, polynomials over finite fields. We will introduce the so-called *Berlekamp's algorithm* to solve this problem, which is the generalisation of Gauss elemination in Linear Algebra. For details see [46] and [2].

ALGORITHM: Berlekamp's Algorithm
Input: Given a square-free polynomial $f(x)$ of degree n in $\mathbf{Z}/N\mathbf{Z}[x]$.
Output: The factorization of $f(x)$ into monic irreducible polynomials.

1. For each i, $0 \leq i \leq n-1$, compute the polynomial;

$$x^{iN} mod(f(x)) = \sum_{j=0}^{n-1} N_{ij} x^j.$$

   Note that each N_{ij} is an element of $\mathbf{Z}/N\mathbf{Z}$.

2. Form the $n \times n$ matrix Q whose (i,j)-entry is N_{ij};

3. Determine a basis $v_1, \cdots, v_t$ for the null space of the matrix $(Q - I_n)$, where I_n is the $n \times n$ identity matrix. The number of irreducible polynomials of $f(x)$ is then precisely t;

4. Set $F \leftarrow \{f(x)\}$. (F is the set of factors of $f(x)$ found so far; their product is equal to f(x).);

5. For i from 1 to t do;

   (a) For each polynomial $h(x) \in F$ such that $degh(x) > 1$ do the following:

      i. compute $gcd(h(x), v_i(x) - \alpha)$ for each $\alpha \in \mathbf{Z}/N\mathbf{Z}$,

      ii. Replace $h(x)$ in F by all those polynomials in the gcd computations whose degrees are ≥ 1.

   (b) $i \leftarrow i + 1$.

6. Return the polynomials in F as the factors of $f(x)$.

5.2.2 ECPP ALGORITHM (Atkin)

Now we will introduce our ECPP Algorithm due to Atkin in form of pseudocode. As we saw in the proceeding section, our aim is now, instead of finding 'random' elliptic curves and applying our DOWN-RUN strategy as in Goldwasser-Kilian, to apply the the theory of complex multiplication for elliptic curves and to use elliptic curves with complex multiplication in our general DOWN-RUN strategy. Our aim here is that we will give at the first the algorithm and in the following sections we are going to try to enhance the efficiency of the algorithm and to try to optimize some of the computational problems coming together with our algorithm and elliptic curve generation with CM in finite fields with large prime characteristic.

ALGORITHM:ATKIN'S ECPP

INPUT: a number $N \in \mathbf{Z}$, whose primality will be (dis)proved,

OUTPUT: If N is composite , a divisor of N, if N is prime return *TRUE*.

1. **Initialize** Set $i \leftarrow 0$, $n \leftarrow 0$ and $N_i \leftarrow N$.

2. **Is N_i small?** If $N_i < 2^{16}$, trial divide N_i by the primes from the list up to 2^{15}. If N_i is not prime GOTO step 14.

3. **Choose next discriminant** Let $n \leftarrow n+1$ and $D \leftarrow D_n$.If $(\frac{D}{N}) \neq 1$, GOTO step 3. Otherwise, use Cornacchia's Algorithm to find a solution,if exists, of the equation $x^2 + \mid D \mid y^2 = 4N$. if no such solution exists, GOTO step 3.

4. **Factor** m For $m = N+1+x, m = N+1-x$ (and in addition for $m = N+1+2y, m = M+1-2y$ if $D = -4$, and $m = N+1+(x+3y), m = N+1-(x+3y)$if $D = -3$), then factor m.

5. **Does a suitable m exist?** If, using the proceeding step, for at least one value of m we can find a q dividing m which passes the Miller-Rabbin test and $> (\sqrt[4]{N_i}+1)^2$, then GOTO step 6, otherwise GOTO step 3.

6. **Compute the elliptic curve** If $D = -4$, set $a \leftarrow -1$, $b \leftarrow 0$. If $D = -3$, set $a \leftarrow 0$, $b \leftarrow -1$. Otherwise compute the minimal polynomial $H_D \in \mathbf{Z}[X]$ of $j((D+\sqrt{D})/2)$. Then reduce H_D modulo N_i and let j be one of the roots of $\bar{H_D} \equiv H_D mod(N_i)$. Then set $c \leftarrow j/(1728-j)\ mod(N_i)$, $a \leftarrow -3c\ mod(N_i)$, $b \leftarrow 2c\ mod(N_i)$.

7. **Find** g By making several 'random' choices of g. find g such that g is a quadratic non-residue modulo N_i, and in addition if $D = -3$, $g^{(N_i-1)/3} \not\equiv 1\ mod(N_i)$.

8. **Find** P Choose at 'random' $x \in \mathbf{Z}/N_i\mathbf{Z}$ until Legendre (resp. Jacobi) symbol $(\frac{x^3+ax+b}{N_i})$ is equal to 0 or 1 (this will occur in a few trial at most). Then compute $y \in \mathbf{Z}/N_i\mathbf{Z}$ such that $y^2 = x^3+ax+b$ (if this algorithm fails GOTO step 14) Finally set $k \leftarrow 0$.

9. **Find right curve** Compute $P_2 \leftarrow (m/q).P$ and $P_1 \leftarrow q.P_2$ on the curve whose affine coordinate is $y^2 = x^3+ax+b$. If during the computations some

division was impossible, GOTO step 14. If $P_1 = \mathcal{O} = (0 : 1 : 0)$ GOTO step 12.

10. Set $k \leftarrow k + 1$. If $k \geq w(D)$ GOTO step 14, else if $D < -4$ set $a \leftarrow ag^2$, $b \leftarrow bg^3$, if $D = -4$ set $a \leftarrow ag$, if $D = -3$ set $b \leftarrow bg$ and GOTO step 8.

11. **Find a new** P Choose at 'random' $x \in \mathbf{Z}/N_i\mathbf{Z}$ until Legendre (resp. Jacobi) symbol $(\frac{x^3+ax+b}{N_i})$ is equal to 0 or 1 (this will occur in a few trial at most). Then compute $y \in \mathbf{Z}/N_i\mathbf{Z}$ such that $y^2 = x^3 + ax + b$ (if this algorithm fails GOTO step 14). If $P_1 \neq \mathcal{O} = (0 : 1 : 0)$ then GOTO step 10.

12. **Check** P If $P_2 = \mathcal{O}_E$, GOTO step 2.

13. **Recurse** Set $i \leftarrow i + 1$, $N_i \leftarrow q$ and GOTO step 2.

14. **Backtrack** (We are here when N_i is not prime, which is very unlikely.) If $i = 0$, output a message saying that N is composite and terminate the algorithm. Otherwise, set $i \leftarrow i - 1$ and GOTO step 3.

As seen in the algorithm, the basic difference comparing with Goldwasse-Kilian is to find the elliptic curves with CM, after that we try to use the same $N - 1$ analog DOWN-RUN strategy, that is applying this algorithm we will get a list of *probable primes* and will try to prove the primality of smaller numbers and by means of this number we will conclude the primality of our actual prime candidate N.

5.2.3 Problems and Approaches

We are going to discuss in this section the problems that we have to deal with coming together with our above algorithm and introduce some approaches to make this algorithm more practical.

Factoring m

Firstly, we have to check whether m satisfies the the condition which will enable us to apply the Theorem 4.9, i. e. that m is not prime, but its largest prime factor q is larger than $(\sqrt[4]{N}+1)^2$. Since we introduced a practical algorithm, we have to deal with this problem much more seriously than in Goldwasser-Kilian test. We will use firstly the trial divide m up to a much higher bound, and then we will use much serious *factorization algorithms* such as Pollard-ρ and $p-1$ to factor m. Here are some approaches of Atkin to solve factorization problem:

Pollard's ρ: It is reasonable to find all factors less than 10^8 with this method. We decide to make 10^5 iterations of this method. Atkin accumulated the iterates of the function and do only two *gcd*'s. See [42], [41] and [11].

ECM: One can use the algorithm as described in [41] with the parametrizations of [42] and [41] for having curves with some prescribed small divisors. One of the basic problem is the storage. One concentrates just on the numbers $< 10^{700}$ see [11].

Pollard's $p-1$: Note that this is reasonable when testing the *Cunningham* numbers which have often the property of being congruent to ± 1 modulo some large known prime integer. So one can spend a little time to see whether we have a possibility to get a factor of m of that type.

If m is not suitable to apply the conditions of theorem 4.9, we have still one more change, that is we can use the other elliptic curves up to isomorphism as we introduced in proposition 5.1 and lemma 5.2.2.

Which curve are we in?

We have $w(D)$ elliptic curves modulo N, where D is an imaginary quadratic discriminant (Corollary 5.1). However, a priori only one of these curves up to isomorphism corresponds a suitable value of m, and it is not clear which one of

these. For $D = -3$ or $D = -4$ it is easy to see a recipe which one is actually the *right curve* (As they are corresponding cases of $j = 0$ and $j = 1728$). For $D < -4$, such a recipe is almost impossible to find. What we can do is simply to compute $m.P$ for our suitable m and a 'random' P, $(P \neq \mathcal{O})$, on one of these two elliptic curves. If this is not equal to the identity, in projective coordinates $m.P \neq \mathcal{O} = (0 : 1 : 0)$. If this is equal to identity we cannot conclude that we are on the right curve, but as P has been randomly chosen, we can probably still use the curve to satisfy the hypothesis of theorem 4.9. note also that, we do not need to prove mathematically that our curve is the right one, since our aim is just to satisfy the conditions of the theorem by means of a an elliptic curve over $\mathbf{Z}/N\mathbf{Z}$.

What is a *good* discriminant D?

In order to obtain the equation of the curve, it is necessary to find the values of j modulo N. This is very very difficult if the class number h_D is large. Hence, we have to start with imaginary quadratic discriminant D whose corresponding class numbers h_D is as small as possible.

Remark 5.3. Some people have expressed concerns that using small class numbers h_D in cryptographic purposes. Because, the resulting curve may be then more amenable to some future attacks than more general field $\mathcal{K}$. On average it is expected that the class number of h_D will grow as $\mathcal{O}(\sqrt{D})$, so small class numbers are in some sence *special* as we explained in our ECPP algorithm. This may cause possible a future, as yet unknown, attacks to try to solve discrete logarithm problem (DLP), and hence cryptosystems, based on the groups of rational points of elliptic curves constructed with CM-method. For details see [7].

Let p be a prime number. Then according to chapter 1 that p is a norm in $\mathbf{Q}(\sqrt{D})$ if and only if p is represented by the principal form of $\mathcal{H}(D)$. For practical purposes as we introduced above, we can assume that $D < 10^{16}$ with $h_D \leq 50$. They form a set $\mathcal{D}$ We have then presumably $\mid \mathcal{D} \mid = 10628$ (For details [11]).

What happens if we have a theoretical failure?

We explained in chapter 3 that we have a complex multiplication, i. e. if $End(E)$ is strictly larger than $\mathbf{Z}$, then we have two cases, namely

1. $End(E)$ is an order of an imaginary quadratic number field,
2. $End(E)$ is the maximal order of a quarternion algebra.

Our curve construction corresponds the first case, i. e. we constructed elliptic curves which are an order of an imaginary quadratic number field. However, if we face with the case 2 (supersingularity case), then it is possible to have a theoretical failure:

If $q \leq (\sqrt[4]{N}+1)^2$, then we cannot apply our theorem. In paricular, it cannot be used when the number of points, m, is a perfect square and $\mathbf{Z}/N\mathbf{Z}$ is isomorhic to $(\mathbf{Z}/M\mathbf{Z})\times(\mathbf{Z}/M\mathbf{Z})$ with $m = M^2$. This is exactly the case 2, i. e. if $M \mid N-1$. We have also then by Hasse

$$\sqrt{N} - 1 \leq M \leq \sqrt{N} + 1$$

putting $\lfloor\sqrt{N}\rfloor = a$ and $N = a^2 + r$, with $0 < r < 2a + 1 \Rightarrow a \leq M \leq a + 1$. Suppose after that at the first $M = a$. Then as $M \mid N - 1$ we have

$$a \mid a^2 + r - 1$$

that is $a \mid r - 1$. Then we have two cases.

- When $r = 1$, one has $N = a^2 + 1$ and E has complex multiplication by $\mathbf{Q}(D)$ with $D = (m - N - 1)^2 - 4N = -4a^2$.
- When $r > 1$ as $0 < r < 2a + 1$, we have $r - 1 = a$ and thus $N = a^2 + a + 1$. It is then easy to see that E has complex multiplication by $\mathbf{Q}(\sqrt{-3})$. For details see [11].

Hilbert Polynomials

Proposition 5.2. The norm of j in $\mathbf{Q}(j)$, which is the same as $\mathcal{H}_D(0)$, is the cube of an integer in $\mathbf{Z}$.

Proof see [11].

It is worth remarking here that we will not need to prove mathematically that our computations regarding j are correct, as the ECPP algorithm and corresponding proof depends only on our calculations on the curves. We described a method and algorithms in section 5.2.1 with special emphasis on the numerical value of the function $j(\tau)$.

Atkin has checked and verified the result of the above proposition whether $\mathcal{H}_D(0)$ is a cube of an integer with error bound 0.5, as we have explained in 4.2.1 for the discriminant $D = 23$ and got the following Hilbert class polynomial

$$\mathcal{H}_{23}(X) = X^3 + 3491750X^2 - 5151296875X + 23375^3.$$

Weber Polynomials

The coefficients of $\mathcal{H}_D(X)$ of j become larger if the class number h_D grows. Although, one can afterwards reduce the results modulo N, to compute these coefficients, we will need to use high precision computations of the values of $j(\tau)$ for every quadratic irrational τ corresponding to reduced imaginary quadratic form of discriminant D. Since these computations are independent of N, one of the solution might be that results of these will be stored before going into algorithm, but again as the coefficients are very large that even for a moderately sized list we would need an enormous amount of storage.

In order to avoid such kinds of computational challenges, we are going to use meromorphic functions which are closely related to the function $j(\tau)$ and which have analogous arithmetic properties. These functions are called *Weber functions* or *Weber polynomials*. Results are due to [11] and [7].

Define the following Weber functions, using Dedekind's η-function, $\eta(z)$:

$$h(\tau) = \zeta_{48}^{-1}\frac{\eta((\tau+1)/2)}{\eta(\tau)}, h_1(\tau) = \frac{\eta(\tau/2)}{\eta(\tau)}, h_2(\tau) = \sqrt{2}\frac{\eta(2\tau)}{\eta(\tau)},$$

$$\gamma_2(\tau) = \frac{h(\tau)^{24}-16}{h(\tau)^8}, \gamma_3(\tau) = \frac{(h(\tau)^{24}+8)(h_1(\tau)^8 - h_2(\tau)^8)}{h(\tau)^8},$$

where $\zeta_n = e^{2\pi i/n}$. These functions are not all algebraically independent because they are all related to j via the equations (for more details see [11]);

$$j = \frac{(h^{24}-16)^3}{h^{24}} = \frac{(h_1^{24}+16)^3}{h_1^{24}} = \frac{(h_2^{24}+16)^3}{h_2^{24}} = \gamma_2^3 = \gamma_3^2 + 1728.$$

Weber calls $\mu(\tau)$ a class invariant if $\mu(\tau)$ lies in the Hilbert class field of $\mathbf{Q}(j)$. Clearly $j(\tau)$ is a class invariant. Furher, with Weber functions one can determine much more class invariants. These give rise to polynomials, abbreviated usually by $\mathcal{W}_D(X)$, using almost the same idea to compute $\mathcal{H}_D(X)$. Finding roots of these polynomials, which have considerably small coefficients in general, will allow us to recover the j-invariants.

Let $-D$ be an imaginary quadratic discriminant and d be a square free positive integer such that $\mathbf{Q}(\sqrt{-D}) = \mathbf{Q}(\sqrt{-d})$. Then we can apply the following conditions in turn

- If $D \equiv 3 \; mod(6)$ use $\mu = \sqrt{-D}.\gamma_3(\tau)$,
- If $D \equiv 7 \; mod(8)$ use $\mu = h(\tau)/\sqrt{2}$,
- If $D \equiv 3 \; mod(8)$ use $\mu = h(\tau)$,
- If $D \equiv \pm 2 \; mod(8)$ use $\mu = h_1(\tau)/\sqrt{2}$,
- If $D \equiv 5 \; mod(8)$ use $\mu = h(\tau)^4$,
- If $D \equiv 3 \; mod(8)$ use $\mu = h(\tau)^2/\sqrt{2}$.

The only problem here is that if we have $D \equiv 3 \; mod(8)$ and $D \not\equiv 3 \; mod(6)$. In that case the degree of Weber polynomial $\mathcal{W}_D(x)$ is then $3h_D$ not h_D. So it is

a better idea then not to use such discriminants.

We get above for the case $D = 23$ the following Hilbert class polynomial

$$\mathcal{H}_{23}(X) = X^3 + 3491750X^2 - 5151296875X + 23375^3.$$

If we use the Weber polynomials instead of Hilbert class polynomials we get

$$\mathcal{W}_{23}(X) = X^3 - X - 1.$$

The above example shows that how we can benefit by using Weber polynomials instead of Hilbert class polynomials.

Furher discussions

Some improvement can be done by examining the possible splitting property of the rational primes in the quadratic extension $\mathcal{K} = \mathbf{Q}(\sqrt{-d})$ and its Hilbert class field. This reduces the following lemma due to [7]:

Lemma 5.2.3. *Let d be a square free integer and p such that we can find a solution to the diaphontine equation*

$$p = x^2 + dy^2.$$

Then we have the followings;

1. *If $p \equiv 3\ mod(8)$ then $D \equiv 2, 3$ or $7\ mod(8)$.*
2. *If $p \equiv 5\ mod(8)$ then $D \equiv 1\ mod(2)$.*
3. *If $p \equiv 7\ mod(8)$ then $D \equiv 3, 6$ or $7\ mod(8)$.*

In particular, we must have $(\frac{-d}{p}) = (\frac{-D}{p}) = 1$.

As we introduced earlier one can perform the *Berlekamp's* algorithm to factor the polynomials $\mathcal{H}_D$ over $\mathbf{Z}/N\mathbf{Z}$. However, this can be expensive, since for a given N, the complexity of such a computation is basically proportional to the

square of the degree of the polynomial, i. e. in our case $\sim h_D$. These explain why we discarded the case $D \equiv 3\ mod(8)$, since in this case, we might work on polynomials of degree $3h_D$. The methods to factor the polynomials over their genus field and other computational remarks related to these can be found in detail in the articel of Atkin. See [11].

5.2.4 certificate

As we introduced both in general ECPP algorithm and in ECPP algorithm of Goldwasser & Kilian, it is also possible to verify the result of the algorithm, if we have built a sequence of intermediate probable primes together with the found elliptic curves and its number of points and a point on it satisfying the requirements of theorem. This is as we already explained is a *certificate* of primality. This is generalization of the ideas of Pratt & Pomerence, see [45] and [44]. For example, Kaltofen and Valente agreed on the certification of 222-digit prime. They also checked the 1226-digit record.

5.3 Remarks

We introduced the Atkin' ECPP algorithm. In contrast to the ECPP algorithm of Goldwasser and Kilian, this algorithm performs well in practice, since it is ample to use this algorithm to prove the primality of numbers from 100 to thousands decimal digits and more. It is possible with current technology of computers to test the arbitrary integers up to 400 digits in a few days on a single SUN 3/60 workstation with this algorithm. Numbers with less than 800 digits can be performed in about one week of real time using distributed process on about 10 workstations.

However, there are also some remaining uncertainties to find the best strategy for applying methods to larger probable prime inputs. The general operations which should be implemented efficiently and optimally , and whose timings on a particular machine are relavent to the strategy are:

- Sieving and subsequent factorization of the number of points of groups of rational points of an elliptic curve,
- Exponentiaton modulo a large prime p (and equivalent square roots, pseudoprime tests),
- Exponention on elliptic curves modulo a large prime p,
- Solution of polynomial equation congruences modulo a large prime p.

CHAPTER 6

ANALYSIS

In this chapter, we are going to analyze the running time complexities of our ECPP algorithms due to Goldwasser-Kilian and Atkin, respectively.

6.1 Preliminaries

The following two theorems due to Heath-Brown and Lenstra, respectively, allow us to analyze our algorithm for uniformly distributed inputs.

Theorem 6.1. Heath-Brown Call an integer y sparse if there are less then $\sqrt{y}/2\lfloor log\ y \rfloor$ primes in the interval $[y, y + \lfloor\sqrt{y}\rfloor]$. Then there exist a constant α such that for sufficiently large x,

$$\mid \{y : y \in [x, 2x],\ y\ is\ sparse\} \mid < x^{5/6} log^{\alpha}\ x.$$

Proof: see [9].

Theorem 6.2. Lenstra Let $p > 5$ be a prime. Let,

$$S \subseteq [p + 1 - \lfloor\sqrt{p}\rfloor, p + 1 + \lfloor\sqrt{p}\rfloor].$$

If a curve given by (A, B), $A,\ B \in \mathbf{Z}$ in Weierstrass normal form, over $\mathbf{Z}_p$ is chosen uniformly, then,

$$prob(\mid (A, B) \mid \in S) > \frac{c}{log\ p}.\frac{\mid S \mid -2}{2\lfloor\sqrt{p}\rfloor + 1},$$

where c is a some fixed constant.

Proof: see [9].

Essentially the size of a 'random' group is at most $\mathcal{O}(1/log\ p)$ times less likely to have a particular property as a randomly selected integer in

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

provided that $\mid S \mid > 2$.

6.2 Analysis

We are going to firstly have a look at the analysis of random elliptic curve generation, which satisfies the conditions of our main theorem 4.9. We now analyse the running time of elliptic curve generation part satisfying our condition in terms of the number of points in an appropriate interval around $p/2$. Define $S(p)$ by

$$S(p) := \left\{ q : q \in \left[\frac{p+1-\lfloor\sqrt{p}\rfloor}{2}, \frac{p+1+\lfloor\sqrt{p}\rfloor}{2} \right],\ q\ is\ prime \right\}.$$

Lemma 6.2.1. *Let $p > 5$ be a k-bit prime, and suppose that $\mid S(p) \mid = \mathcal{O}(\sqrt{p}/log^c\ p)$. Then prime number generation will run for expected $\mathcal{O}(k^{c+8})$ steps before it terminates.*

Proof: see [9].

Lemma 6.2.2. *Let $p > 5$ be a prime, and let (A, B) be chosen uniformly from curves over $\mathbf{Z}_p$. Let also $S(p)$ be defined as above. Then*

$$prop(\mid (A,B) \mid\ is\ twice\ a\ prime) > \frac{c}{log\ p}.\frac{\mid S(p) \mid -2}{2\lfloor\sqrt{p}\rfloor} + 1,$$

where c is some fixed constant.

Proof: see [9].

After introducing the basic facts which was used to analyse the ECPP algorithm of Goldwasser & Kilian, we are going to give the theorems which and their prove can be found in [32] and [24] in details.

Theorem 6.3. Suppose that there exist two positive constants c_1 and c_2 such that the number of primes in the interval $[x; x + \sqrt{2x}]$ $(x \geq 2)$ is greater than $c_1\sqrt{x}(logx)^{-c_2}$. Then Goldwasser-Kilian algorithm proves the primality of N in expected time $\mathcal{O}((log\ N)^{10+c_2})$.

Theorem 6.4. There exists two positive constants c_3 and c_4 such that for all $k \geq 2$, the proportion of prime numbers N of k-bits for which the expected time of Goldwasser-Kilian algorithm is bounded by $c_3(log\ N)^{11}$ is at least $1 - c_8 2^{-k^{\frac{1}{loglog\ k}}}$.

At the end we can summarize the analyzis of the algorithm as follows:

1. Given an input of length k, the algorithm produces a certificate of primality that is of length $\mathcal{O}(k^2)$, and requires $\mathcal{O}(k^4)$ steps to verify.

2. The algorithm terminates in expected polynomial time on every prime number, provided that the following conjecture is true:

$$CONJECTURE:\ (\exists c_1, c_2 > 0)\pi(x + \sqrt{x}) - \pi(x) \geq \frac{c_2\sqrt{x}}{log^{c_1}\ x},$$

   $for\ x\ sufficiently\ large.$

3. There exist constants c_1 and c_2 such that for all k sufficiently large, the algorithm will terminate in expected $c_1 k^{11}$ time for all but at most,

$$\frac{2^k}{2^{k^{c_2/log\ log\ k}}},$$

   of the inputs. In other words, the algorithms can be proved to run quickly on all but vanishingly small fraction of the prime numbers.

As for Goldwasser-Kilian algorithm, we have only the heuristic analysis cited in [33]. Atkin and Morain found that the running time of Atkin's ECPP is roughly $\mathcal{O}((log\ N)^{6+\epsilon})$. for some $\epsilon > 0$. Other implementation details and practical considerations to make this algorithm more practical and more optimized was briefly discussed in chapter 5, and for further details see [11].

chapter 7

Implementation, LiDIA classes and Conclusion

7.1 Implementation details

7.1.1 SINGULAR source codes Examples

We will start with our *implementations* with a prime listing algorithm (like sieve of Eratosthenes) in programming language SINGULAR by using the prime funtion.
Note: Limitation for the integers in SINGULAR (upper bound) is 2147483647.

```
.....................................................................
//The following function computes the list of primes given
//in lower and upper bound from bigger to smaller
//Written by Osmanbey Uzunkol...

proc primelist(int a,int b){
  list myprime;
  int temp=prime(b);
  myprime[1]=temp;
  int temp2=temp;

     if (a!=1) {//if..

        for(int i=2;temp>a;i++) {//if.
    temp=prime((temp2-1));
```

```
     if ((temp>a)&& (temp>2) ) {
    myprime[i]=temp;
    temp2=temp;        }

                                  }//end of if.
    return(myprime);//end of if..
                }
     else {
   list myprime2=primelist((a+1),b);
   int b=size(myprime2);
   list myprime3;

     for (int i=1;i<=b;i++) {
   myprime3[i]=myprime2[i];
                           }
         myprime3[b+1]=2;

           return(myprime3);
}
   }
```

Example: Let us illustrate how this function works with a small example:

```
> primelist(45353109,45353264);
[1]:
   45353237
[2]:
   45353207
[3]:
   45353201
[4]:
```

```
   45353183
[5]:
   45353173
[6]:
   45353149
[7]:
   45353111
```

.....

.....

```
//Following programme tests whether the given prime candidate N
//passes the trial-division algorithm as explained in the thesis
//It computes the primes up to 32768 in a list and then tries to
//divide the candidate any of this list elements. If one of the element
//divides N then returns the prime, and proves that this number is
//composite together with its divisor. If it returns 1 then we have a
// (probable) prime. Written by Osmanbey Uzunkol...

 ring r=0,x,lp; //in order to use the type number to compute big integers
 proc isdivisible(number N) {
    list l=primelist(1,32768); //note that this option can be changed
    int temp=size(l);          //depending on the situation...
    int i=1;
     while(i<=temp) {

       if ((N mod l[i])==0){
       if (l[i]!=N){
          return(l[i]);
                        }
       else        {
          return(1);
                   }
```

```
                        }//end of if

      i++ ;
                }
    return(1);
 }
```

.....

Example: The following example illustrates and prove that given number is divisible by the prime 3251.

.....

```
> isdivisible(19928823733665155266771716253621762652673663536671619848758724 41);
// ** redefining b **
3251
```

.....

Example: The following example shows that the given candidate 153298644105116578975167048104782789635951257974397 is (probable) prime. Although it is actually not, since 153298644105116578975167048104782789635951257974397 = 9661373.158671696150346932030434026410928125470314 89.

.....

```
> isdivisible(153298644105116578975167048104782789635951257974397);
// ** redefining b **
1
```

.....

.....

```
//The following fuction computes the Legendre symbol for primes and Jacobi
//symbol for integers in general as explained in the thesis.
//Written by Osmanbey Uzunkol..

 ring r=0,x,lp;
```

```
proc Jacobi(number a,number n) {
 int i;
 int s;
 number temp=a;

 a=a mod n;
  if (a==0)    {
     return(0);}
  if (a==1)    {
     return(1);}
  if (gcd(a,n)>1) {
     return(0);   }
  else {
   while(temp mod 2==0) {
      temp=(temp/2);
      i++;
                         }
   if (i%2==0){ s=1;}
   else {
    if((n mod 8)==1||(n mod 8)==7) {
       s=1;                         }
    else    {
       s=-1;}
        }
       if ( ((n mod 4)==3) && ((temp mod 4)==3) ) {
          s=(-s);                                  }

       if (temp!=1) {
          number n1=n mod temp;
          int temp1=s*Jacobi(n1,temp);
           return(temp1);
                    }
```

```
          else {
           if (i%2==0)    {
               return(1); }
           else {
            if((n mod 8)==1||(n mod 8)==7) {
               return(1);                  }
           else           {
              return(-1); }
           }
          }
     }//end of else
}
```

.....

Example: Some examples of Jacobi funtion: For the first example as $gcd(14414442441, 1626616611101713) = 7$, we have;

.....

```
> Jacobi(14414442441,1626616611101713);
0
```

.....

Second and third one illustrate the case of quadratic residue and non-residue, respectively.

.....

```
> Jacobi(64553633552,3232442001);
1
> Jacobi(77575775,288287376);
-1
```

.....

.....

```
//The following function may be used instead of primelist function in
```

```
//fermattest and Sollovay-Strassen Test.
//Written by Osmanbey Uzunkol..

 proc base(int a){
  list l;
  int j=1;
   for(j=1;j<=a;j++){
     l[j]=j+1;
                    }
  return(l);
                 }
.....................................................................................

.....................................................................................
//This function tests the primality (actually compositeness)
//of the prime candidate N by using Fermat primality test
//as explained in the thesis with the chosen prime base elements
//between the below given bounds.Upper bound can be changed
//within the limitations of integers in SINGULAR.
//Note that with a fast multiplication methods as we explained
//for elliptic curves the efficiency can be speeded up. If
//the answer takes much time than one has to use C++ version.
//Written by Osmanbey Uzunkol.

ring r=0,x,lp;
proc fermattest(number N){
 if(isdivisible(N)>1){
  return(0);}
 else{
 number r;
 number temp;
 number exp=(N-1);
```

```
 number j;
 list l=primelist(32768,33768);
 int k=size(l);
 int i;
   for(i=1;i<=k;i++) {
     r=1;
    for(j=1;j<=exp;j=j+1){
      temp=l[i]*r;
      r=temp mod N;
                          }

  if(r mod N!=1 mod N){
    return(0);                          }
                           }
  return(1);
}
}
```

Example: The compositeness of the candidate 64366536553727646563 was proven by calling the fermattest:

```
> fermattest(64366536553727646563);
// ** redefining b **
0
```

Furhermore, the (probable) primality of the following integer, which is a divisor of the above integer, was also proven. However, it takes lots of time as the multiplication procedure is slow in SINGULAR. Actually, if it takes more than 1 minute, the answer is probably a probable prime but it is a better idea to check it in C++ version.

```
.....................................................................
> fermattest(3608596543910279);
// ** redefining b **
1
.....................................................................

.....................................................................
//This function tests the primality (actually compositeness)
//of the prime candidate N by using Sollovay-Strassen primality test
//as explained in the thesis with the chosen prime base elements(not random)
//between the below given bounds.Upper bound can be changed
//within the limitations of integers in SINGULAR.
//Note that with a fast multiplication methods as we explained
//for elliptic curves the efficiency can be speeded up. If
//the answer takes much time than one has to use C++ version.
//Written by Osmanbey Uzunkol.

ring r=0,x,lp;
proc sollovay_strassen (number N){
 if(isdivisible(N)>1){
  return(0);}
 else{
 number r;
 number temp;
 number exp=((N-1)/2);
 number j;
 list l=primelist(32768,33768);
 int k=size(l);
 int i;
   for(i=1;i<=k;i++) {
     r=1;
    for(j=1;j<=exp;j=j+1){
```

```
      temp=l[i]*r;
      r=temp mod N;
                        }

  if(r mod N!=Jacobi(l[i],N) mod N){
    return(0);                          }
                         }
  return(1);
}
}
```

Example: We will give two examples one is composite the other is (probable) prime. However, the caution that we mentioned in `fermattest` is also the case here.

```
> sollovay_strassen(42423425161662655252544155 1421);
// ** redefining b **
0
```

```
> sollovay_strassen(91346224180575661878056249);
// ** redefining b **
1
```

7.1.2 $C++$ source codes, used classes and Examples

In this section, we will see some implementation of primality tests with the help LiDIA C++ library for Computational Number Theory. At the end by means of

primeproof method developed by J. Hechler, see [30] for LIDIA, the implementation of ECPP algorithm will be given in an example file written in C++ with examples.

Here with the help of **primelist** function a list of primes in a given lower/upper bounds:

```
.....................................................................................
//This programme computes a list of prime numbers given in
//lower and upper bound. It is almost the same as wriitten
//in SINGULAR, but it is more efficient
//due to fast computational capability of LiDIA
//written by Osmanbey Uzunkol

#include <LiDIA/prime_list.h>
#include <iostream>
using namespace LiDIA;
using namespace std;
unsigned long * primelist(unsigned long lower,unsigned long upper){
  unsigned long *prime;
  prime_list primelist(lower,upper);
  int i;
  i=primelist.get_number_of_primes ();
  int j=0;
  prime[j]=primelist.get_first_prime();
  unsigned long temp=prime[i];
  while (j<i)
    {
      j++;
      prime[j] = primelist.get_next_prime();
      temp=prime[j];
    }
  return(prime);
```

```
}

int size_primelist(unsigned long lower,unsigned long upper){
   unsigned long *prime;
   prime_list primelist(lower,upper);
   int i;
   i=primelist.get_number_of_primes ();
   return(i);
}

int main()
{
unsigned long lower, upper, prime;
cout << "Please enter lower bound: "; cin >> lower ;
cout << "Please enter upper bound: "; cin >> upper ;
cout << endl;
 prime_list p_list(lower, upper);
 prime = p_list.get_first_prime();
 while (prime)
   {
     cout << prime << endl;
     prime = p_list.get_next_prime();
   }
 return 0;
}
```

.....

Example: the same example that we gave for `primelist` function for SINGULAR (note that in this time list is from smaller prime to bigger:

.....

```
        Please enter the lower bound: 45353109
```

```
        Please enter the upper bound: 45353264

        45353111
        45353149
        45353173
        45353183
        45353201
        45353207
        45353237
.....................................................................

.....................................................................
int isdivisible( unsigned long N){
 unsigned long  p;
 prime_list pl(2, 10000000);
 for(p = pl.get_first_prime();p<1000000;p=pl.get_next_prime){
  if (N%p==0) return(0);
  else return(1);                                          }
}
#include <LiDIA/prime_list.h>
#include <iostream>
using namespace LiDIA;
using namespace std;
int main()
{
  unsigned long N;
  cout<<"enter the possible prime <10^14"<< N; cin>>N;
  cout<<endl;

  cout<<isdivisible(N)<<endl;
  return 0;
}
```

```
.....

.....
//This example programme computes Legendre-Jacobi symbol
//as in the SINGULAR...
//exaple written by Osmanbey Uzunkol..
#include <LiDIA/bigint.h>
#include <iostream>
using namespace LiDIA;
using namespace std;
int main(){
  //In this programme we will compute
  //Jacobi/Legendre symbol of (a/n)

  bigint a,N;
  cout<<"Please enter the value of a: "; cin>>a;
  cout<<"Please enter the value of N: "; cin>>N;
  int i=jacobi(a,N);
  cout<<"The Jacobi(Legendre)-symbol is: "<<i<<endl;
  return(0);
}
.....
```

Example: As in the case of singular, three cases of Jacobi-symbol will be given as examples:

```
.....
please enter the value of a: 71766727289393844949949948949
please enter the value of n: 822726661555615516666166171771001O1
The Jacobi(Legendre)-symbol is: 1
.....

.....
```

```
please enter the value of a: 6546647437473783882871
please enter the value of n: 61616525266266161177111172782271
The Jacobi(Legendre)-symbol is: -1
.....

.....
please enter the value of a: 10786846916586309307182037205410965
please enter the value of n: 10000000018115255258718628525431780916556709302l
The Jacobi(Legendre)-symbol is: 0
.....

.....
//The source code of fermattest of LiDIA will be given in this programme
//example file written by Osmanbey Uzunkol..
#include <LiDIA/bigint.h>
#include <iostream>
using namespace LiDIA;
using namespace std;
int fermattest(const bigint & n)
{
bigint tmp_a, tmp_n, res;
register int a = 2;

if (n < 2)
return 0;

if ((n == 2) || (n == 3) || (n == 5) || (n == 7))
return 1;

tmp_n.assign(n);
dec(tmp_n);

while (a <= 7) {
```

```
if (!remainder(n, a))
return 0;
else {
tmp_a.assign(a);
power_mod(res, tmp_a, tmp_n, n);
if (!res.is_one())
return 0;
else {
if (a == 2)
a += 1;
else
a += 2;
}
}
}
return 1;
}
int main()
{
bigint N;
cout<<"please enter the prime candidate N : "; cin>>N;
cout<<endl;
if(fermattest(N)) cout<<N<<" is (probably) prime"<<endl;
else cout<<N<<" is a composite number"<<endl;
 return 0;
}
```

.....

Example: The following two examples will show composite and (probable) prime cases, respectively

.....

```
please enter the prime cadidate N: 434355252566363355424522636636635535 1
```

```
434355252566363355424522636636635535l is a composite number
.....

.....
please enter the prime cadidate N: 2543015553938550490663
2543015553938550490663 is a (probably) prime
.....

.....
//The source code of fermattest of LiDIA will be given in this programme
//example file written by Osmanbey Uzunkol..
#include <LiDIA/bigint.h>
#include <iostream>
using namespace LiDIA;
using namespace std;
bool
bigint::is_prime (const int b1) const
{
static long a[10] = {3, 5, 7, 11, 13, 17, 19, 23, 29, 31};
long b, j, ok, i, k = 0, sx;

if (!longify(sx)) {
// can not be converted to long
if (sx <= 0)
return false;

if (sx == 2)
return true;

if (sx <= 31) {
for (i = 0; i < 10; i++)
if (sx == a[i])
return true;
```

```
return false;
}
}

if (is_le_zero() || is_even())
return false;

if (b1 <= 0)
lidia_error_handler("is_prime", "#tests <= 0");

if (b1 > 9)
b = 9;
else
b = b1;

bigint erg;
bigint H(37), Q(*this);
Q.dec();

bigint N_minus1(Q);

while (Q.is_even()) {
Q.divide_by_2();
k++;
}

for (i = 0; i <= b; i++) {
power_mod(erg, bigint(a[i]), Q, *this);

if (!erg.is_one() && erg.compare(N_minus1)) {
j = k;
```

```
ok = 0;

while ((j > 0) && !ok) {
square(erg, erg);
remainder(erg, erg, *this);

if (!erg.compare(N_minus1)) ok = 1;

j--;
}

if (!ok) {
return false;
}
}
}

for (; i <= b1; i++) {
if (!compare(H))
return true;

power_mod(erg, H, Q, *this);

if (!erg.is_one() && erg.compare(N_minus1)) {
j = k;
ok = 0;

while ((j > 0) && !ok) {
square(erg, erg);
remainder(erg, erg, *this);

if (!erg.compare(N_minus1)) ok = 1;
```

```
j--;
}

if (!ok) {
return false;
}
}

H = H.next_prime();
}

return true;
}



int main(){
bigint N;
cout<<"Please enter the prime cadidate :"; cin>>N;
if (is_prime(N,10))
  cout<<N<<" is (probable) prime"<<endl;
else
 cout<<N<<" is composite"<<endl;
return 0;
}
.....

.....
please enter the prime cadidate :6636536366366363636555626266263101
6636536366366363636555626266263101 is composite
.....
```

```
.....................................................................
please enter the prime cadidate: 449017159180743482307697943
449017159180743482307697943 is (probable) prime
.....................................................................

.....................................................................
//A factorization example for use in N-1 or N+1
//tests when we can factorize the intger fully.
//Note that there are other factorizetion functions that
//enable us to test the primality of N if the
//partial factorization is available for N-1 or N+1.

#include <LiDIA/rational_factorization.h>
#include <iostream>
using namespace LiDIA;
using namespace std;
int main()
{
rational_factorization f;
bigint n;
cout << "\n Please enter a number: ";
cin >> n;
f.assign(n);
f.factor();
if (f.is_prime_factorization())
cout<<"\n Prime Factorization: " << f << endl;
else
cout<<"\n Factorization: "<< f << endl;
return 0;
}
.....................................................................
```

Example: an example of prime factorization of integer N in the form (p_i, e_i)

where

$$N = \prod_{i=1}^{k} p_i^{e_i} \; where \; p_i's \; are \; prime;$$

```
.....................................................................
Please enter a number: 722373666447774889299438843747747773738
Prime Factorization: [(2,1),(137,1),(281,1),(4787,1),(393373,1),
(532724413961699228710902 7,1)]
.....................................................................
```

We will see the `Cornacchia` algorithms written in the `bigint` class of LiDIA due to the pseudocode (Modified Cornacchia's algorithms) that we gave in chapter 2.

```
.....................................................................
bool cornacchia (bigint & x, bigint & y, const bigint & DD, const bigint & p)
{
bigint x0, a, b, l, r;
bigint D, tmp2, D_abs, p_four;
bigint rr;
bool r_is_sqr;
long s;

if (!DD.is_negative())
lidia_error_handler("cornacchia", "D not negative");

s = (4 - (DD.least_significant_digit() & 3)) & 3; // DD mod 4

shift_left(p_four, p, 2);
D.assign(DD);

if (!is_prime(p) || !p.is_positive() || p == 2)
lidia_error_handler("cornacchia", "no odd prime number");

if ((-D) >= p_four)
```

```
lidia_error_handler("cornacchia", "|D| >= 4*p");

if (s != 1 && s != 0)
lidia_error_handler("cornacchia", "D != 0 or 1 mod 4");

if (jacobi(D, p) == -1) {
return false; // (D/p) = -1 -->no solution
}
else {
ressol(x0, p+D, p);
if (x0.is_even() != D.is_even())
subtract(x0, p, x0);

shift_left(a, p, 1);
b.assign(x0);

shift_left(l, p, 2);
sqrt(l, l); // l = floor(2*sqrt(p)) = floor(sqrt(4p))

while (b > l) {
remainder(r, a, b);
a.assign(b);
b.assign(r);
}

square(l, b);
subtract(a, p_four, l);
// a = 4p - b^2 now

D.negate();
remainder(r, a, D);
```

```
if (!r.is_zero()) {
return false;
}
else {
divide(r, a, D);
r_is_sqr = is_square (rr, r);
if (!r_is_sqr) {
return false;
}
else {
x.assign(b);
y.assign(rr);
return true;
}
}
}
}
```

.....

Now at the end, we will use the primeproof class (see for more details [30] and [20]) to write down an example file in C++ and give examples of $N-1$, $N+1$ and ECPP algorithms;

.....

```
#include <iostream>
#include <LiDIA/bigint.h>
#include <LiDIA/prime_proof.h>
#include <LiDIA/certificate.h>
using namespace LiDIA;
using namespace std;
int main()
{
```

```
bigint N;
cout << "Please enter the prime candidate : ";
cin >>N;
prime_proof proof;
certificate c;
proof.set_verbose(true);
proof.set_ecpp_mode(1);
proof.set_prime(N);
bool success = proof.prove_prime();
 if(success)
cout<<N<<" is prime"<<endl;
else cout<<N<<" is not prime"<<endl;
 return 0;
}
```

.....

Examples:

.....

```
Please enter the prime candidate : 2142973051
 Primelength:10
 Make the SPP test
SPP: n-1=2^l*q, l: 1 q: 1071486525
SPP: Test was succesful
2142973051 is prime
```

.....

.....

```
Please enter the prime candidate : 13893427619861410061536716578910836681937718817738892987377829919983877565300220
17738892987377829919983877565300220018817272773017726637727727 1
 Primelength:109
13893427619861410061536716578910836681937718817738892987377829919983877565300220
```

```
018817272773017762663772772 71 is not prime
.....................................................................................

.....................................................................................
Please enter the prime candidate : 614367291872088173452017745619376378920198837
625678299199827782991001663551783901009882771645209188465517290019872626615671 88
172726637818991998827728919910000938388299288399288829992888919771771888188811 77
11000118811717
 Primelength:219
614367291872088173452017745619376378920198837625678299199827782991001663551783 90
100988277164520918846551729001987262661567188172726637818991998827728919910000 93
838829928839928882999288891977177188818881177110001188117 17 is not prime
.....................................................................................
```

We will finish this section by giving a *primality proof* by using ECPP in an easy to understand form due to [7]. Note that it is easy to verify and get *certificate* by means of intermediate datas below;

```
.....................................................................................
126765060022822940149670320 5653
169317673849406496638751929789    53542864930901413159140235 5077
122311651710723437189087960 8558  34881870097669254769721966 5601
126765060022823077635754418 6344
176476322298420571611003 7
-------------------------------------

176476322298420571611003 7
123710600901914193475439 7   82473733934609462316959 8
498566265383685655850376    16981609587630133894156 26
176476322298158772974796 8
21321838780409719
-------------------------------------
```

```
21321838780409719
5979072666605065    11093328037873283
12289991207526417   5086330291908954
21321839059327264
636820759
```

. . . . . .

Using DOWN-RUN, we got the number 636820759. By calling the primelist function, we can easily see that this number is prime by means of trial division. Hence, we proved the primality of 126765060022822940149670320565 3 by reducing the primality of it to the prime 636820759.

7.2 Conclusion

The aim of this thesis was to introduce modern primality testing methods and to explain the so-called Atkin's ECPP Algorithm. Most of the primality tests were covered by means of explaining the necessary theoretical background coming from algebra, number theory and arithmetic of elliptic curves and by means of introducing the methods in an algorithmic approach. Furthermore, we introduced intensively the computational problems and solutions coming together with the theory of elliptic curves, in particular curves with CM. Additionally, approaches to make the ECPP algorithm more practical were also discussed. At the end of thesis, several programming examples of primality tests are given together with the examples of Atkin's ECPP in computer algebra system SINGULAR and programming language C++, with the help of using LiDIA computer package for computational number theory.

REFERENCES

REFERENCES

[1] H. Cohen, *A Course in Computational Algebraic Number Theory*, Springer-Verlag, Second corrected Printing, 1995,

[2] A.J.Menezes, P.C. van Oorschot, S.A. Vanstone. Kornberg, *Handbook of Applied Cryptography*, CRC Press, 1996,

[3] J. Buchmann, *Einführung in die Kryptographie*, Springer-Verlag, 1999,

[4] M. O. Rabin, *Probabilistic Primality Testing*, J. Number Theory, 12(1980) 128-138,

[5] J. Silverman, *Advanced topics in the arithmetic of Elliptic Curves*, Springer-Verlag, 1994,

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

[7] I. Blake, G. Seroussi, N. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press, 1999,

[8] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, 1987,

[9] S. Goldwasser, J. Kilian, *Primality Testing Using Elliptic Curves*,

[10] L. M. Adleman, C. Pomerance, R. Rumely *On distinguishing prime numbers from composite numbers*, Ann. Math. 117, 173-206,

[11] A. O. L. Atkin, F. Morain, *Elliptic curves and Primality Proving*, Math. Comp. , 61, 29-67, 1993,

[12] M.Agraval, N.Kayal, N.Sexana, *Primes is in* $\mathcal{P}$, paper August 2002,

[13] D. Cox, *Primes of the form* $x^2 + ny^2$, John Wiley and Sons, 1989,

[14] H. Cohen, A. K. Lenstra *Implementation of a new primality test*, Math. Comp. 48, 177(1987), 103-121,

[15] S. Galbraith, J. McKee, *The probability that the number of points on an elliptic curve over a finite field is prime*, Journal of the London Mathematical Society, 62, no. 3, pg. 671-684, 2000,

[16] *Geometrische Methoden in der Kryptographie*, Vorlesungsskript TU-Kaiserslautern, SS 2003,

[17] *Primzahltests and Kryptographie I-II*, Vorlesungsskript TU-Kaiserslautern, WS 2003-2004 and SS 2003,

[18] *Algebraische Geometrie I*, Vorlesungsskript TU-Kaiserslautern, WS 2002-2003,

[19] F. Morain, *Building cyclic elliptic curves modulo large primes*, Eurocrypt'91, pg. 531-543,

[20] LIDIA C++ library for computational number Theory, *www.informatik.tu-darmstadt.de/TI/LiDIA /Wellcome.html*,

[21] S. B. Lippman, *C++ Primer*, 2.Edition, Addison-Wesley,

[22] G.-M. Greuel, G. Pfister, H. Schoenemann *A Computer Algebra System for Polynomial Computations*, FB Mathematik TU-Kaiserslautern,

[23] M. Welschenbach, *Kryptographie in C und C++*, 2. Auflage, Springer-Verlag,

[24] S. Goldwasser, J. Kilian, *almost all primes can be quickly certified*, in 18th Annual Symposium on Foundations of Computer Science, IEEE, pg. 316-329, Berkeley, California, May 1986,

[25] L. M. Adleman, M. A. Huang *Recognizing primes in random polynomial time*, in Proceedings 19th STOC(1986) ACM Press, pg. 462-469, New-york City, May 25-27, 1987,

[26] E. R. Berlekamp, *Factoring polynomials over large finite fields*, Math. Comp. 24, 111 (1970), 713-735,

[27] W. Bosma, *Primality testing using elliptic curves*, Tech. Rep. 85-12, Math. Instituut, Universiteit van Amsterdam, 1985,

[28] P. Mihailescu, F. Morain, *Dual elliptic primes and Cyclotomy primality proving*,

[29] N. D. Elkies, *Elliptic and modular curves over finite fields and related computational issues*, in *Computational perspectives on Number Theory: Proceedings of a Conference in honor of A. O. L. Atkin*, American Mathematical Society International Press, 7, 1998,

[30] J. Hechler, *Primzahlnachweis mit Hilfe elliptischer Kurven*, Diplomarbeit, Okt. 2003 TU-Darmstadt,

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

[32] H. Cohen,H. W. Lenstra, *Primality testing and Jacobi sums*, Math. comp. 42, 165(1984), 297-330,

[33] A. K. Lenstra, H. W. Lenstra, *JR. Algorithms in number theoty*, In Handbook of Theoretical Computer Science, J. van Leeuwen, Ed. vol. A: Algorithms and Complexity. North Holland, 1990, ch. 12, pp. 674-715,

[34] K. Ireland, M. Rosen, *A classical introduction to modern number theory*, vol. 84 of Graduate texts in Mathematics, Springer, 1982,

[35] F. Morain, *Elliptic curves, primality proving and some titanic primes*, 1989, (1992),

[36] F. Morain, J. Olivos *Speeding up the computations on an elliptic curve using addition-substraction chain*, Info. Theory Appl. 24, 531-543, 1990,

[37] W. E. Clark, J. J. Liang *On arithmetic weight for a general radics representation of integers*, IEEE Trans. Info. Theory, 19, 823-826, 1973,

[38] C. H. van Lint, *Introduction to Coding Theory*, Springer-Verlag, 1982,

[39] G. Reitwiesner, *Binary Arithmetic*, Adv. in Comp., 1, 231-308, 1960,

[40] P. Ribenboim, *The book of prime number records*, Third edition Springer 1995,

[41] R. P. Brent, *Some integer factorization using elliptic curves*, Australian Computer Science Communications 8(1986), 149-163,

[42] P. L. Montgomery, *Speeding the Pollard and elliptic curve methods of factorization*, Math. Comp. 48, 177 (January 1987), 243-264,

[43] A. O. J. McKee, *Subtleties in the distribution of the numbers of points on elliptic curves over a finite field*, J. LMS,

[44] C. Pomerence, *Very short primality proofs*, Math. Comp. 48, 177 (1987), 315-322,

[45] V. R. Pratt, *Every prime has sufficient certificate*, SIAM J. Comput. 4 (1975), 214-220,

[46] *Einführung in die Computer Algebra*, Vorlesungsskript WS 2002-2003,

[47] F. Morain , *Distributed primality proving and the primality of* $(2^{3539}+1)/3$, In Advanced Cryptology - EUROCRYPT'90 (1991), I. B. Demgard, Ed., vol. 473 of Lect. Notes in Computer Science, Springer-Verlag, pp. 110-123. Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques, Aarhus, Denmark, May 21-24, 1990,

[48] O. Atkin *The number of points on an elliptic curve modulo a prime*, manuscript (1991),

[49] T. Oetiker, H. Partl, I. Hyna, E. Schlegl, *A not very short Introduction to LATeX*, Version 1.0, November 16, 1994.