STUDY ON ELLIPTIC AND HYPERELLIPTIC CURVE METHODS FOR INTEGER FACTORIZATION

楕円および超楕円曲線法による素因数分解の研究

by

Takayuki Yato

八登崇之

A Senior Thesis

卒業論文

Submitted to

Department of Information Science

Faculty of Science

The University of Tokyo

on February 15, 2000

in Partial Fulfillment of the Requirements

for the Degree for Bachelor of Science

Thesis Supervisor: Hiroshi Imai　今井 浩

Title: Associate Professor of Information Science

ABSTRACT

Today factoring integers becomes more important because the security of RSA public-key cryptosystem is based on its intractability. Various methods for doing this have been developed such as Quadratic Sieve and Number Field Sieve.

This thesis focused on Elliptic Curve Method (ECM), which is the fastest of all the algorithms whose running time depends on the smallest prime factor of a given integer. And the effects of so far proposed improvements for this method were confirmed through experiments.

Furthermore, use of hyperelliptic curves for factoring was considered and implemented. As a result, the inferiority over ECM was observed. And the order of Jacobian groups was also considered.

論 文 要 旨

今日、素因数分解の重要性が増している。それは RSA 公開鍵暗号系の安全性が素因数分解の難しさに立脚しているからである。これに対して二次ふるい (QS) や数体ふるい (NFS) 等の種々の方法が考案されている。

本論文では、実行時間が対象の数の最小素因数の大きさに依存するアルゴリズムの中で最高速である楕円曲線法 (ECM) を主に扱った。そして実験を通してこれまでに提案されている改良法の効果を確認した。

さらに、超楕円曲線を用いた素因数分解について考察および実装をして、その結果楕円曲線法よりも劣ることが観察された。さらにヤコビアン群の位数について考察を行った。

Table of Contents

1. Introduction **1**

1.1 Algorithms for Integer Factorization · · · 1

1.2 Outline of This Thesis · · · 3

2. Elliptic Curve Method **5**

2.1 Elliptic Curves · · · 5

2.2 Addition Law · · · 6

2.3 $p-1$ Method · · · 8

2.4 Elliptic Curve Method · · · 9

2.4.1 Reduction modulo n · · · 9

2.4.2 Algorithm · · · 10

2.5 Montgomery's Improvement · · · 12

2.5.1 Montgomery-form Curves · · · 12

2.5.2 Addition Law · · · 13

2.5.3 2nd Step · · · 14

2.5.4 Selection of Curves · · · 15

3. Implementation of ECM **17**

3.1 Experiments · · · 17

3.2 Results · · · 18

3.3 Evaluation · · · 18

3.4 Remarks: Influence of Cache Misses · · · 21

4. Factorization Using Hyperelliptic Curves **23**

4.1 Background · · · 23

4.2 Definitions · · · 24

4.3 Factoring Method · · · 26

4.3.1 Principle · · · 26

4.3.2 Algorithm · · · 27

4.3.3 Remark: Relation to ECM · · · 27

4.3.4 Examples · · · 28

4.4 Order of Groups · · · 30

4.5 Implementation · · · 31

4.6 Evaluation · · · 32

5. Conclusions **34**

References **36**

Acknowledgements

I am very grateful to Prof. Hiroshi Imai for his advice and encouragement. I would also like to thank other members of Imai laboratory.

List of Tables

3.1 Running time and number of successes (ECM-1) · · · · · · · · · · · · 19
3.2 Running time and number of successes (ECM-2) · · · · · · · · · · · · 20

4.1 Running time and number of successes (HECM) · · · · · · · · · · · · 32

List of Figures

2.1 An elliptic curve on the real plane · 7

Chapter 1

Introduction

1.1 Algorithms for Integer Factorization

Integer factorization is one of the problems that have been long considered in the world of the number theory. In the last few decades, together with the rapid progress of computer technology, methods for factoring integers efficiently were studied, and as a result some such algorithms were invented.

One of the important applications of integer factorization is RSA public-key cryptosystem. The security of the cryptosystem depends on the intractability of factoring integers.

Presently the algorithms listed below are widely known.

Trial Division The simplest algorithm, which continues divisibility tests for a composite number n by integers starting from 2 until a factor is found. Of course, we can skip even numbers larger than 2, but in any case this method becomes impractical when all of prime factors of n is over 12 digits in decimal.

ρ *Method* (by Pollard [11]) Using the Birthday Paradox, this method finds integers x_1 and $x_@$ such that $x_1 \equiv x_2 \pmod p$ (where p is a prime divisor of n) and obtains $p = \mathrm{GCD}(x_1 - x_2, n)$.

$p-1$ Method (by Pollard [10]) On the basis of Fermat's Little Theorem $a^{p-1} \equiv 1 \pmod{p}$, it finds a factor. Described in Chapter 2.

Elliptic Curve Method (ECM) (by H. W. Lenstra, Jr. [7]) This method makes use of the property of groups of points on elliptic curves to find a factor. It is the main theme of this thesis and described in Chapter 2 in detail.

Quadratic Sieve (QS) (by Pomerance [12]) This method collects relations of the form $s^2 \equiv t^2 \pmod{n}$ and finds a factor by using them. To obtain such relations, a kind of sieve is used.

Number Field Sieve (NFS) (by A. K. Lenstra et al. [6]) This method uses a "factor base" on the integer ring over a fixed algebraic field, instead of that of (ordinary) primes as in QS.

Time complexity of some algorithms is dependent on the given composite number n, and that of others is dependent on the smallest prime factor p of n. As for the algorithms mentioned above, the average (or worst-time) complexity is shown below [1]:

$*$ Dependent on n	
QS	$L_n[1/2, 1.020]$
NFS (General)	$L_n[1/3, 1.901]$
$*$ Dependent on n	
Trial Division	$O(p)$
ρ Method	$O(\sqrt{p})$
$p-1$ Method	$O(p')$
ECM	$L_p[1/2, 1.414]$

(p' is the largest prime divisor of $p-1$.) Here the function $L_x[\gamma, c]$ is defined as follows:

$$L_x[\gamma, c] = \exp((c + o(1))(\log x)^{\gamma}(\log\log x)^{1-\gamma}) \tag{1.1}$$

And an algorithm that has the time complexity $L_n[\gamma, c]$ for some c and $\gamma < 1$ (where n is the input integer) is called a *subexponential time algorithm*. Note that $L_n[0; c] = O((\log n)^c)$ (polynomial time) and $L_n[1; c] = O(n^c)$ (exponential time with respect to the input length $\log n$).

As we see in the table, the best-known algorithm for factoring integers is NFS, asymptotically and practically for very large composite numbers (over 120 digits). However, ECM can find relatively small factors (less than 50 digits) of so large integers that NFS cannot treat, because ECM does not have a limitation with respect to n. For this reason, ECM is still an important technique for factorization at the present time. ECM was first introduced by H. W. Lenstra, Jr. [7], and since then various improvements have been devised. This thesis mainly treats this method and analyze the effect of such improvements.

1.2 Outline of This Thesis

In this thesis, we consider integer factorization by using ECM through experiments with computer.

First, we implemented ECM.

- the original algorithm by Lenstra [7]
- the improved algorithm by Montgomery [9] and Suyama, without 2nd step
- the improved algorithm by Montgomery and Suyama, with 2nd step

From these implementations we confirmed the following well-known facts:

- Use of Montgomery-form curves rather than Weiestrauss-form ones makes computation about two times faster.

- Choosing suitable curves increases the probability that factors are found (about 40%).

- 2nd Step largely increases the probability that factors are found with a small increase in running time.

Second, we investigated factoring using hyperelliptic curves through experiments. Using hyperelliptic curves for factoring is analyzed by Lenstra, Pila and Pomerance [8] from theoretical interests, and theoretically it is conjectured to be *not* so good as ECM. On the other hand, in cryptography algebraic curves used are extended from elliptic to hyperelliptic, and experimental analysis of the order of Jacobian groups of hyperelliptic curves, which is the key point in factorization using hyperelliptic curves, has a close relation to security analysis of hyperelliptic curve cryptosystems. We thus investigated factoring using hyperelliptic curves with the objective that we evaluate quantitatively how *bad* it is and consider some simple examples about the order of hyperelliptic curves, even though it is not practically useful. As a result, we found the following facts:

- Factorization using hyperelliptic curves takes over ten times as much time as ECM, and the probability of success is by far smaller than ECM.

- We confirmed the fact that the order of Jacobian groups of hyperelliptic curves of genus 2 defined over $\mathbf{F}_p$ is approximately equal to p^2.

Now we outline this thesis. In Chapter 2 we describe ECM and its improvement. In Chapter 3 we implement ECM and experiment on it. In Chapter 4 we investigate factoring using hyperelliptic curves. Last in chapter 5 we conclude.

Chapter 2

Elliptic Curve Method

2.1 Elliptic Curves

We start with the definition of elliptic curves.

Definition 2.1 Let $\mathbf{F}$ be a field. An *elliptic curve* E over $\mathbf{F}$ is a curve that is given by an equation of the following form:

$$E: \; y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \quad (a_i \in \mathbf{F}) \tag{2.1}$$

where E must be smooth.

We let $E(\mathbf{F})$ denote the set of points $(x, y) \in \mathbf{F}^2$ that satisfy this equation, along with a *point at infinity* denoted O. The condition that the curve E be smooth is that there be no points of $E(\mathbf{F})$ where both partial derivatives vanish.

If the characteristic of $\mathbf{F}$ is neither 2 nor 3, then by a few changes of variables we can obtain the following:

$$y^2 = x^3 + ax + b, \quad (a, b \in \mathbf{F}). \tag{2.2}$$

In this case, the condition that the curve be smooth is equivalent to requiring that the cubic on the right have no multiple roots. This holds if and only if the

discriminant of $x^3 + ax + b$, which is $-(4a^2 + 27b^3)$, is nonzero. In the next few sections we will treat curves of the form (2.2) (Weierstrass-form).

2.2 Addition Law

Let E be an elliptic curve of the form (2.2). In this section, we define an addition law on the set of points on E. In order to describe this visually, we think of elliptic curves over the real numbers.

Definition 2.2 Let E be an elliptic curve over the real numbers given by equation (2.2), and let P and Q be two points on E. We define the negative of P (denoted by $-P$) and the sum $P + Q$ according to the following rules:

- $-O = O$, $O + P = P$. (O serves as the zero element.)
- If $P = (x, y)$, then $-P = (x, -y)$.
- If neither P nor Q is equal to O, then $P + Q$ is defined as follows:
  - If P and Q have different x-coordinates, then we let R to be the third point where the line $\overline{PQ}$ intersects the curve E. (There is exactly one such point.) If $\overline{PQ}$ is tangent to E at P (or Q), we take $R = P$ (or Q). Then we define $P + Q$ to be $-R$.
  - If $P = Q$, then we let R to be the only other point where the tangent line to E at P intersects E, and define $2P = -R$. (If the tangent line has a "double tangency" at P, then R is taken to be P.)
  - If $P = -Q$, then $P + Q = O$.

  (Note that if P has the same x-coordinate with Q, P must be equal to either Q or $-Q$.)

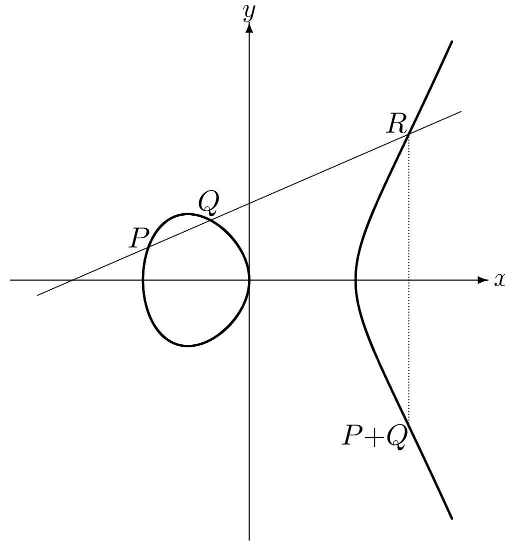

Figure 2.1: An elliptic curve on the real plane

Next we consider representing this operation as coordinate calculation. Let $P_1 = (x_1, y_1)$. $P_2 = (x_2, y_2)$, and $P_1 + P_2 = (x_3, y_3)$. Then x_3 and y_3 are expressed by the following rational formulae in terms of x_1, x_2, y_1, y_2:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned} \tag{2.3}$$

$$\text{where } \lambda = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & (P \neq \pm Q) \\ (3z_1^2 + a)/2y_1 & (P = Q) \end{cases}$$

Using these algebraic formulae, we can define an addition of points on an elliptic curve over any field of characteristic $\neq 2, 3$.

Theorem 2.1 *The set of points on an elliptic curve (including O) is an Abelian group with respect to the above operation.*

The only group law that is not an immediate consequence of the above-mentioned rules is the associative law. That can be proved from a fact from the projective geometry of cubic curves.

By the notation nP, we mean P added to itself n times if n is positive, and $-P$ added to itself $|n|$ times if n is negative. To compute nP, we use the repeated doubling, that is, after generating $2P, 2^2P, 2^3P, \ldots$ we sum up some of them according to the binary representation of k.

2.3 $p-1$ Method

Before proceeding to Lenstra's elliptic curve method, we give a classical factoring technique which is analogous to Lenstra's method. This is called Pollard's $p-1$ method.

Suppose that we want to factor a composite number n, and p is some (as yet unknown) prime factor of n. If p happens to have the property that $p-1$ has no large prime divisor, then the following algorithm very likely finds p:

1° Choose a bound B and set

$$k = \prod_{p:\text{prime}, p \leq B} p^{\alpha_p}, \quad \text{where } \alpha_p = \max\{\alpha \mid p^\alpha \leq B\}$$

(Note that B is the least common multiple of all integers $\leq B$.)

2° Choose an integer a between 1 and $n-2$ and compute $a^k \bmod n$ by the repeated squaring method. Using the result, compute $d = \text{GCD}(a^k - 1, n)$ by the Euclidean algorithm.

3° If d is not a non-trivial divisor of n, go back to 2°.

To explain when this algorithm will work, suppose that p is a prime divisor of n such that $p-1$ is a product of small prime powers. Then it follows that k is a multiple of $p-1$ (because it is a multiple of all of the prime powers in the factorization of $p-1$), and so, by Fermat's Little Theorem, we obtain $a^k \equiv 1 \pmod{p}$.

Then we can get a non-trivial factor of n by computing $\mathrm{GCD}(a^k - 1, n)$, unless every other prime divisor q of n has the property $a^k \equiv 1 \pmod q$.

Example We factor $n = 542669$ by this method, choosing $B = 8$ (and hence $k = 2^3 \cdot 3 \cdot 5 \cdot 7 = 840$) and $a = 2$. We find that $2^{840} \bmod n = 119565$, and $\mathrm{GCD}(119564, n) = 421$. This leads to the factorization $542669 = 421 \cdot 1289$.

However, this algorithm fails when all of the prime divisors p of n have $p-1$ divisible by a relatively large prime (or prime power).

Example Let $n = 488781$. This number has the factorization $488781 = 387 \cdot 1263$. And we have $387 - 1 = 2 \cdot 193$ and $1263 - 1 = 2 \cdot 631$. So unless k is divisible by 191 (that is, we choose $B \geq 191$), we are unlikely to find a non-trivial divisor.

The main weakness of Pollard's $p-1$ method is that it depends on the structure on the group $\mathbf{F}_p^*$ (more precisely, the various such groups as p runs through the prime divisors on n). (Note that Fermat's Little Theorem roots on the fact that the order of $\mathbf{F}_p^*$ is $p-1$.) If all of those groups have an order divisible by a large prime, this method does not work.

Lenstra's method overcomes the weakness. By working with elliptic curves over $\mathbf{F}_p$, we can use a variety of groups, and we can realistically hope to find one whose order is not divisible by a large prime power.

2.4 Elliptic Curve Method

2.4.1 Reduction modulo n

When we use elliptic curves for factoring, we must compute modulo n, since we do not yet know the prime divisors of n. Let p to be a prime divisor of n and the following proposition holds:

Proposition 2.1 *If* $a \equiv b \pmod{n}$*, then* $a \equiv b \pmod{p}$*.*

Because of the fact, when we compute modulo n and reduce the result modulo p, the final result is almost always the same as the result when we compute modulo p from the beginning. This property means that computing coordinates of points on a elliptic curve E is equivalent to considering E which coordinates are reduced by each prime divisor p at a time. Let $(E \bmod p)$ denote E reduced by p in this way.

The only exception of the property is division. To compute a/b modulo n, we must find the inverse of b by the extended Euclidean method and multiply a to it modulo n. If the inverse does exist, then the property mentioned above holds and the quotient obtained is congruent to a/b also modulo p. However, if $b \equiv 0 \pmod{p}$, we cannot divide a by b modulo p. In this case, when we attempt to divide a by b modulo n, the extended Euclidean algorithm fails to give the inverse of b and returns p as the g.c.d. of b and n (unless there is another prime divisor q of n such that $b \equiv 0 \pmod{q}$).

Again we consider the computation of kP on a curve E. In the sequence of computations, the above-mentioned situation occurs when $kP = O$ on $(E \bmod p)$. That is, if the sum of two points is O, these two points are negative of each other, thus have the same x-coordinate. Therefore if we let x_1 and x_2 be the x-coordinate of the points (modulo n), then $x_1 \equiv x_2 \pmod{p}$, so that we attempt to apply the addition law (2.3) (which involves division by $(x_1 - x_2)$) and then inverse calculation fails. As a result, we can find a non-trivial divisor of n as $\mathrm{GCD}(x_1 - x_2, n)$, unless every other prime divisor q of n has the property $x_1 \equiv x_2 \pmod{q}$.

2.4.2 Algorithm

Now we are ready to give the description of algorithm.

1° Choose a bound B and set

$$k = \prod_{p:\text{prime}, p \le B} p^{\alpha_p}, \quad \text{where } \alpha_p = \max\{\alpha \mid p^\alpha \le B\} \tag{2.4}$$

2° Choose an integer a and b between 1 and $n-1$ and consider the elliptic curve $E: y^2 = x^3 + ax + b$ modulo n. In the case $\mathrm{GCD}(4a^3 + 27b^2, n) > 1$, choose a, b again if this value is n, and otherwise we succeed in finding a non-trivial divisor of n and we're done.

3° We choose a random point $(x, y) \in E$, and attempt to compute kP.

4° If we fail to obtain an inverse of a denominator at any stage of the computation, then we find a divisor of n as the g.c.d. of n and the denominator. (If the result is n, go back to 2°.)

5° If kP is actually calculated without failure, go back to 2° because we fail to find a divisor.

Let $\#E$ denote the number of points (including O) on the elliptic curve E (that is, the *order* of the group of points on E). From the basic theorem of the group theory — *the order of a group is divisible by the order of an element* — the following theorem holds:

Theorem 2.2 *Let E be an elliptic curve, P a point on E, and $n = \#E$. Then $nP = O$.*

In the previous section, we observed that when $kP = O$ on $(E \bmod p)$ for a prime divisor p of n the computation of kP fails and a non-trivial divisor of n (probably p itself) is found. Theorem 2.2 claims that if k is a multiple of the order $\#(E \bmod p)$ then $kP = O$ on $(E \bmod p)$. Thus if we take k as (2.4), we can find a non-trivial divisor of n when n has a such prime divisor p that has the property

that $\#(E \bmod p)$ is the product of primes all $\leq B$. Here we define a term that describes the property.

Definition 2.3 Let B be a positive number. An integer is said to be B-*smooth* if it is not divisible by any prime greater than B.

Concerning the number of points on elliptic curves, the following theorem holds:

Theorem 2.3 (Hasse)

$$p + 1 - 2\sqrt{p} \leq \#(E \bmod p) \leq p + 1 + 2\sqrt{p}$$

Unlike $p - 1$ Method, the value of $\#(E \bmod p)$ varies near around $p + 1$ according to the parameters a and b, so that we have a chance to find a curve whose order is not divisible by a large prime power, thus obtain a non-trivial divisor of n.

2.5 Montgomery's Improvement

2.5.1 Montgomery-form Curves

The problem about Lenstra's method is that it involves many inverse calculations, which are done by the extended Euclidean algorithm and require many divisions. Montgomery [9] succeeded in avoiding inverse calculations in such a way that we describe here.

First, we use elliptic curves of the following form (called Montgomery-form):

$$E : by^2 = x^3 + ax^2 + x, \quad b(a^2 - 4) \neq 0. \tag{2.5}$$

Moreover, we consider them on the *projective plane*

$$\begin{aligned} \mathbf{P}^2(\mathbf{Z}/n\mathbf{Z}) &= \{[X : Y : Z] \mid X, Y, Z \in \mathbf{Z}/n\mathbf{Z}\}/\sim. \\ & [X : Y : Z] \sim [\lambda X : \lambda Y : \lambda Z] \end{aligned} \tag{2.6}$$

(In other word, we represent coordinates with rational numbers, as $x = X/Z$, $y = Y/Z$.) In this coordinate system, (2.5) is expressed as follows:

$$E : bY^2Z = X^3 + aX^2Z + XZ^2. \tag{2.7}$$

And the "point at infinity" O has the coordinates $[0 : 1 : 0]$, which is the only point on the curve with the Z-coordinate zero. Therefore whether $nP = O$ on $E \bmod p$ is judged by the condition that the Z-coordinate of nP be congruent to zero modulo p.

2.5.2 Addition Law

For a certain point P, we write $nP = [X_n : Y_n : Z_n]$. We compute $(l+m)P$ from lP and mP using the following formulae:

when $l \neq m$

$$\begin{cases} X_{l+m} = Z_{l-m}(X_l X_m - Z_l Z_m) \\ Z_{l+m} = X_{l-m}(X_l Z_m - Z_l X_m) \end{cases} \tag{2.8}$$

when $l = m$

$$\begin{cases} X_{2l} = (X_l^2 - Z_l^2)^2 \\ Z_{2l} = 4X_l Z_l (X_l^2 + aX_l Z_l + Z_l^2) \end{cases} \tag{2.9}$$

Y-coordinates are omitted since they are not used.

The formula (2.8) needs the coordinates of $(l-m)P$. Thus we need a special method for computing multiples of points. Here is a method of computing kP for a point P [2] (we assume k is odd, since for k equal to a power of 2 kP is easily computed by using (2.9) repeatedly):

1° Let

$$k = k_0 + k_1 \cdot 2 + k_2 \cdot 2^2 + \cdots + k_r \cdot 2^r, \quad k_i \in \{0,1\}, k_0 = k_r = 1$$

be the binary representation of k.

2° Let $S = P$, $T = 2P$, $U = -P$.

3° For $i = 1, \dots, r$ do the following:

when $k_i = 1$

$$S := S + T \text{ (using } U\text{)}, \; T := 2T \text{ (}U\text{ is unchanged)},$$

when $k_i = 0$

$$U := U - T \text{ (using } S\text{)}, \; T := 2T \text{ (}S\text{ is unchanged)}.$$

4° Then we have $S = kP$.

2.5.3 2nd Step

Using the method mentioned so far, we can find a prime divisor k when $\#(E \bmod p)$ is B-smooth. Here we consider the case that p has exactly one such prime number that is larger than B (but not larger than B', which is from 40 to 100 times as large as B). The operation mentioned from now on is called *2nd step* (while the operation so far is called *1st step*).

Assume $\#(E \bmod p)$ equal to qm, where m is the product of primes not larger than B and q is a larger prime. Let $R = mP$. Since $(qm)P = O \pmod p$, we have $qR = O \pmod p$. Thus we can find q by checking if $qR = O$ for all the primes q that satisfy $B < q \le B'$, although this method is not efficient.

To make it more efficient, we use the following method. Set $q = 420t \pm s$ $(0 < s < 210)$, and we have $\mathrm{GCD}(q, s) = 1$. Since $Z(qR) = 0 \pmod p$, it follows from the addition law

$$Z(420tR)X(\pm sR) - X(420tR)Z(\pm sR) = 0 \pmod p.$$

Using $X(\pm sR) = X(sR)$, $Z(\pm sR) = Z(sR) \pmod p$, we have

$$Z(420tR)X(sR) - X(420tR)Z(sR) = 0 \pmod p,$$

hence

$$x(420tR) - x(sR) = 0 \pmod p,$$

where $x = X/Z$. We utilize this fact.

We generate in advance the polynomial

$$f(x) = \prod_{\substack{0<s<210 \\ \mathrm{GCD}(s,210)=1}} (x - x(sR)) \pmod p \tag{2.10}$$

and compute $f(x(420tR))$ for each t in the range $0 < t < B'/420$. Then one of the values must be zero because $q \leq B'$. Consequently, by performing the calculation modulo n, we obtain $\mathrm{GCD}(f(x(420tR)), n) = p$ and thus find a prime divisor. (This kind of technique is called *Baby-step Giant-step Method*.)

Here the number $210 = 2 \cdot 3 \cdot 5 \cdot 7$ is chosen because it is a number n such that $\varphi(n)/n$ is small. We can use a larger number, for example $2310 = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11$, for speeding up.

2.5.4 Selection of Curves

Actually, the order of a curve is rarely smooth. For the reason it is significant how to choose suitable curves.

Because the efficiency of ECM depends on smoothness of the order, we can speed up the method by choosing curves whose order has some small factors which are known in advance. An arbitrary Montgomery-form curve has the order divisible by 4. In addition, by choosing a, b properly we can make the order divisible by 3 (hence by 12).

$$a = \frac{-3u^4 - 6u^2 + 1}{4u^3}, \quad b = \frac{(u^2 - 1)^2}{4uv^2}$$
$$uv(u^2 - 1)(9u^2 - 1) \neq 0$$

In order to get a point (x, y) we must determine u, v and x so that y^2 is a trivial square number. We can achieve this by setting

$$u = \frac{2r}{3r^2 - 1} \quad x = \frac{3u^2 + 1}{4u}$$
$$\Big(\text{then } y^2 = \frac{(u+1)^2 v^2}{16}\Big(\frac{3r^2+1}{3r^2-1}\Big)^2\Big)$$

We need not determine v and b actually because y is not used.

Chapter 3

Implementation of ECM

3.1 Experiments

We implemented these three kinds of ECM algorithms:

1. the original algorithm by Lenstra [7]
2. the improved algorithm by Montgomery [9] and Suyama, without 2nd step
3. the improved algorithm by Montgomery and Suyama, with 2nd step

Using each of them, we performed the following experiment.

Experiment We generate 100 composite numbers which are products of two l-decimal-digit prime numbers and apply the method to each number 20 times. And then we examine how many times in total it succeeds in finding a non-trivial divisor. We also measure running time. This procedure is carried out with various values of B (upper bound of smoothness).

The condition of the experiments is as follows:

* Machine spec: Ultra-60 360 MHz

* OS: Solaris 2.7
* Language: C++
* Compiler: gcc version 2.8.1, with option `-O2`
* Multiprecision Package: LiDiA version 1.3.3

3.2 Results

The results are shown in Tables 3.1 and 3.2. There c is the number of successes, t_f is the average running time in failure cases, and T is the expected time that is needed to obtain a non-trivial divisor. It is estimated as follows:

$$T = \frac{c}{\#\text{ of trials}} \cdot t_f$$

(Here "# of trials" $= 10 \cdot 200 = 2000$.) Values of T are parenthesized when c is less than 5, since they are unreliable. The optimal T for each k is shown in the *slanted* face (only in the case that we are confident).

3.3 Evaluation

From the experimental results, we suggest the following:

- The value of c increases with an increase of B, and it decreases with an increase of l. This value is proportional to the success probability and thus the ratio of B-smoothness of l-digit integers, since the order of elliptic curves over $\mathbf{F}_p$ is roughly equal to p from the Hasse's Theorem.

- The value of t_f increases almost linearly with B. It is explained from the fact that $\log k$ where k is given by (2.4) is approximately proportional to B, although we do not prove the fact yet.

[1. Lenstra's original algorithm]

l \ B		1000	2000	4000	8000	16000
10	c	145	220	416	548	721
	t_f	0.222	0.449	0.893	1.810	3.623
	T	3.061	4.079	4.291	6.606	10.05
12	c	26	72	131	199	334
	t_f	0.267	0.539	1.074	2.170	4.349
	T	20.55	*14.96*	16.39	21.81	26.04
14	c	9	22	33	67	110
	t_f	0.307	0.622	1.230	2.498	5.003
	T	68.22	*56.51*	74.56	74.55	90.97
16	c	3	6	8	23	44
	t_f	0.357	0.720	1.437	2.899	5.801
	T	(237.9)	240.1	359.3	*252.0*	263.6
18	c	0	1	5	6	14
	t_f	0.406	0.817	1.626	3.281	6.573
	T	—	(1634)	650.5	1093	939.0
20	c	0	1	0	1	5
	t_f	0.454	0.920	1.820	3.667	7.358
	T	—	(1839)	—	(7334)	2943

[2. Improved algorithm, without 2nd step]

l \ B		1000	2000	4000	8000	16000
10	c	274	418	613	842	1041
	t_f	0.113	0.228	0.459	0.931	1.881
	T	*0.824*	1.090	1.498	2.211	3.614
12	c	72	139	218	348	506
	t_f	0.124	0.253	0.511	1.029	2.089
	T	*3.457*	3.646	4.688	5.916	8.256
14	c	14	37	53	114	198
	t_f	0.136	0.274	0.551	1.114	2.254
	T	19.38	*14.80*	20.79	19.54	22.76
16	c	1	4	21	38	69
	t_f	0.152	0.307	0.618	1.250	2.520
	T	(303.9)	(153.7)	*58.87*	65.79	73.04
18	c	0	2	3	9	22
	t_f	0.166	0.338	0.677	1.363	2.761
	T	—	(338.4)	(451.2)	302.7	250.9
20	c	0	0	4	7	8
	t_f	0.184	0.367	0.746	1.498	3.026
	T	—	—	(372.8)	428.1	756.5

Table 3.1: Running time and number of successes (1): The values c, t_f and T are defined samely as in Table 3.2.

[3. Improved algorithm, with 2nd step]

l	$\backslash$ B	1000	2000	4000	8000	16000
	c	1097	1301	1488	1638	1715
10	t_f	0.167	0.317	0.628	1.247	2.514
	T	*0.304*	0.488	0.844	1.522	2.931
	c	410	621	803	1027	1224
12	t_f	0.189	0.362	0.711	1.408	2.822
	T	*0.920*	1.166	1.771	2.742	4.611
	c	119	213	340	511	725
14	t_f	0.205	0.393	0.766	1.521	3.056
	T	*3.451*	3.688	4.507	5.952	8.430
	c	30	73	131	209	329
16	t_f	0.233	0.447	0.872	1.735	3.462
	T	15.56	*12.23*	13.31	16.60	21.04
	c	4	14	36	86	141
18	t_f	0.256	0.490	0.959	1.895	3.796
	T	(127.8)	70.05	53.29	*44.08*	53.83
	c	3	5	10	30	45
20	t_f	0.282	0.546	1.060	2.117	4.219
	T	(188.3)	218.2	212.0	141.1	187.5

Table 3.2: Running time and number of successes (2): There c is the number of successes out of 2000 times, t_f is the average running time in failure cases (sec), and T is the expected time until success (sec).

- Running time t_f increases as the number of digits l increases. It is natural, but the ratio of increase is not so large as is expected. We do not know the reason.

- Because we do not have numbers of successes large enough at large l's, it is difficult for us to claim something about T. Nevertheless, we can see the tendency that optimal B becomes larger as l grows larger.

- Comparing 1. with 2., we find that a value of c in 2. is about 40–80% larger than that in 1. It is because choosing curves so that they have orders divisible by 12 increases the probability that orders are smooth and thus divisors are found. Moreover, a value of t_f in 2. is about 40–50% of that in 1. The decrease indicates that the use of Montgomery-form curves speeds up the computation. The value of optimal B does not change very much.

- Comparing 2. with 3., we find that 2nd Step substantially raises the probability of success with a 30–40% increase of running time. From the change in optimal T, we conclude that this improvement reduces the expected time to 20–30% of the previous value.

3.4 Remarks: Influence of Cache Misses

The original ECM requires very small amount of memory. In fact, it needs to hold only ten or some multiprecision integers, We can thus consider that all the computations are done on the cache memory. However in 2nd step of the improved algorithm, we encounter the situation that we hold a high-degree polynomial $f(X)$ on the memory and examine the value of $f(X)$ when X is substituted by each of the values $x_1, x_2, \ldots$ (where coefficients and each x_i are multiprecision integers).

Let $2g$ be the interval of the giant-step. (In Chapter 2 we took $g = 210$.) Then the degree of the polynomial is $\varphi(g)$. If we let $g = 210(= 2 \cdot 3 \cdot 5 \cdot 7)$ (hence $\varphi(g) = 48$) and the length of integers be 128 bits (40 digits in decimal), then the size of memory that is needed to hold the polynomial is

$$128 \text{ bits} \times 48 = 6144 \text{ bits} = 768 \text{ bytes}.$$

It is not so large, but if we take a larger g for speeding up, such as $g = 2310(= 210 \cdot 11)$ ($\varphi(g) = 480$), and the size of involved integers becomes 256 bits, the size of the polynomial is 15 Kbytes, and some computers may not have enough cache memory to hold it. Thus we examined influence of cache misses.

Experiment In the computation of the value of a polynomial under a modulus, every time a coefficient is read from the memory, one multiplication and one remainder calculation are performed. Thus, we measured the cache miss penalty when a 128-bit integer is read and time needed for a multiplication and a remainder calculation involving integers of that length.

Result We measured cache miss penalty per one word (= 32 bits) and it turned out to be approximately 7 ns. Hence

$$\text{penalty when a 128-bit integer is read} = 28 \text{ ns}.$$

On the other hand, time needed for operations was

$$\begin{aligned} \text{Time}(128 \text{ bits} \times 128 \text{ bits}) &= 1691 \text{ ns}, \\ \text{Time}(256 \text{ bits mod } 128 \text{ bits}) &= 5383 \text{ ns}. \end{aligned}$$

Thus we conclude that we have no problem at all about cache misses in ECM.

Chapter 4

Factorization Using Hyperelliptic Curves

4.1 Background

In the previous chapters, we discussed integer factorization using elliptic curves. Another application of elliptic curves is *elliptic curve cryptosystems*, which are based on the intractability of the *discrete logarithm problem* in groups of points on elliptic curves.

Nowadays, in the area of cryptology, using *hyperelliptic curves* is eagerly studied [4, 13], because it gives the same security level with a smaller key length as compared to cryptosystems using elliptic curves. From the fact it is expected to be possible to use hyperelliptic curves to factor integers, since ECM exploits the property of the Abelian groups in the same way as the cryptosystems. However, factorization using hyperelliptic curves has hardly ever been studied. As far as we know, the paper by Lenstra, Pila and Pomerance [8] is the only one such paper. In that paper they analyzed, from theoretical interest, a factoring algorithm that uses the Jacobian groups of hyperelliptic curves of genus 2, and concluded that the

algorithm (called *hyperelliptic curve method* by them) has the expected running time $L_p[1/2, 2]$ at most (where p is the smallest prime divisor) and thus *not* so good as the elliptic curve method, whose running time is $L_p[1/2, \sqrt{2}]$.

There are some due reasons that hyperelliptic curves are not suitable for integer factorization, and here we give one of them. Generally, computations on Jacobian groups of hyperelliptic curves take much longer time than those on groups of elliptic curves. The reason that hyperelliptic cryptosystems are still practically useful is that they can use a prime field $\mathbf{F}_p$ for smaller p than elliptic cryptosystems to attain the same security level. However, in factorization we must always compute modulo a given composite number n, and as a result we only suffer from bad influences. Some other reasons are shown in the later discussion.

Nevertheless, we believe evaluating quantitatively how *bad* it is to be worthwhile. With this objective we implemented factorization using hyperelliptic curves.

4.2 Definitions

In this section we give the brief definitions of hyperelliptic curves and its Jacobian groups. For details, see [5].

Definition 4.1 Let $\mathbf{F}$ be a field. A *hyperelliptic curve* C *of genus* g (≥ 1) *over* $\mathbf{F}$ is a nonsingular curve that is given by an equation of the following form:

$$C : \ v^2 + h(u)v = f(u) \qquad (\text{in } \mathbf{F}[u, v]) \tag{4.1}$$

where $h(u) \in \mathbf{F}[u]$ is a polynomial of degree $\leq g$, and $f(u) \in \mathbf{F}[u]$ is a monic polynomial of degree $2g + 1$.

Although a Jacobian group $\mathbf{J}$ itself is defined by considering curves over the algebraic closure of $\mathbf{F}$, in the factoring algorithm (and also in hyperelliptic cryptosystems) its subgroup $\mathbf{J}(\mathbf{F})$ is used. The group $\mathbf{J}(\mathbf{F})$ can be viewed as the set

of reduced divisors. A *divisor* is primarily defined as a formal sum of points on the curve, but here we give a definition of the reduced divisors in the *polynomial representation*, which form is used in actual computation.

Definition 4.2 Let C be a hyperelliptic curve (given by (4.1)) of genus g over a field $\mathbf{F}$. A *reduced divisor* (defined over $\mathbf{F}$) of C is defined as a form $\mathrm{div}(a, b)$, where $a, b \in \mathbf{F}[u]$ are polynomials such that

- a is monic, and $\deg b < \deg a \leq g$,
- a divides $(b^2 - bh - f)$.

In particular $\mathrm{div}(1, 0)$ is called *zero divisor.*

When we define an addition over reduced divisors as mentioned below, they form an Abelian group whose identity is the zero divisor.

1° Compute d_1, e_1 and e_2 which satisfy

$$d_1 = \mathrm{GCD}(a_1, a_2) \text{ and } d_1 = e_1 a_1 + e_2 a_2$$

2° If $d_1 = 1$, then

$$a := a_1 a_2, \quad b := (e_1 a_1 b_2 + e_2 a_2 b_1) \bmod a,$$

otherwise do the following:

- Compute d, c_1 and s_3 which satisfy

$$d = \mathrm{GCD}(d_1, b_1 + b_2 + h) \text{ and } d = c_1 d_1 + s_3(b_1 + b_2 + h).$$

- Let $s_1 := c_1 e_1$ and $s_2 := c_1 e_2$, so that

$$d = s_1 a_1 + s_2 a_2 + s_3(b_1 + b_2 + h).$$

- Let

$$a := a_1 a_2 / d^2, \quad b := (s_1 a_1 b_2 + s_2 a_2 b_1 + s_3(b_1 b_2 + f))/d \bmod a.$$

3° While $\deg a > g$ do the following:

- Let $a := (f - bh - b^2)/a$.
- Then let $b := (-h - b) \bmod a$.

4° Let $a := c^{-1}a$, where c is the leading coefficient of a.

Here we mean this group $\mathbf{J}(\mathbf{F})$ simply by the Jacobian group of the hyperelliptic curve.

4.3 Factoring Method

4.3.1 Principle

First we give principle of the factoring method using hyperelliptic curves (we call it *hyperelliptic curve method (HECM)* just like the precursors). The main difference between this method and ECM is just that we use Jacobian groups of hyperelliptic curves instead of groups of elliptic curves. That is to say, we must again compute modulo a given composite number n rather than a prime p. For simplicity, we assume n is the product of two primes p and q. From the argument in Section 2.4, we find that if a computation result modulo n is reduced modulo p (or q) it gives the result of the same computation modulo p (or q). Hence comes the following proposition:

Proposition 4.1 *Let k to be an integer, D a divisor, and we attempt to compute kD modulo n. If kD computed over $\mathbf{J}(\mathbf{F}_p)$ and $\mathbf{J}(\mathbf{F}_q)$ are $\operatorname{div}(a_p, b_p)$ and $\operatorname{div}(a_q, b_q)$ respectively, and $\deg a_p < \deg a_q$, then the computation of kD modulo n fails and a non-trivial divisor of n is found.*

Proof. Suppose the computation succeeded, and let $\operatorname{div}(a, b)$ to be the result. Let $a \bmod p$ denote a whose coefficients are reduced modulo p, then we have a mod

$p = a_p$ and a mod $q = a_q$. However, since a is monic, the degrees of both a mod p and a mod q must be equal to that of a, which leads to a contradiction. ∎

(If $\deg a_p < \deg a_q$, p should be found. But we do not yet prove that. In any case, that is not significant in the following discussion.)

4.3.2 Algorithm

Next we describe the algorithm of HECM briefly. Note that for a point $(x, y) \in \mathbf{F}^2$ on a curve $\mathrm{div}(u - x, y)$ is always a reduced divisor from the definition.

1° Choose a bound B and set k samely as in ECM. (See (2.4).)

2° Consider a "random" hyperelliptic curve C.

3° Take a random point $(x, y) \in C$, and let divisor $D = \mathrm{div}(u - x, y)$. Attempt to compute kD modulo n.

4° If the computation fails, then we find a divisor of n. Otherwise go back to 2°.

4.3.3 Remark: Relation to ECM

Next we apply Proposition 4.1 to elliptic curves (the case $g = 1$). Since a reduced divisor $\mathrm{div}(a, b)$ of an elliptic curve must satisfy $\deg a \le 1$, the proposition is expressed as follows:

Corollary 1 *Let k to be an integer, D a divisor, and we attempt to compute kD modulo n. If $kD = \mathrm{div}(1, 0)$ over $\mathbf{J}(\mathbf{F}_p)$ and $kD \neq \mathrm{div}(1, 0)$ over $\mathbf{J}(\mathbf{F}_q)$, then the computation of kD modulo n fails and a non-trivial divisor of n is found.*

A nonzero divisor $\mathrm{div}(u - x, y)$ of an elliptic curve E always has its corresponding point (x, y) on E. Therefore the set of such divisors is in one-to-one

correspondence with the set of finite points. In addition, when we correspond the zero divisor $\mathrm{div}(1,0)$ with the point at infinity O, we obtain the following familiar result about elliptic curves (by Lenstra [7]):

Corollary 2 *Let k to be an integer, P a finite point, and we attempt to compute kP modulo n. If $kP = O$ over $(E \bmod p)$ and $kP \neq O$ over $(E \bmod q)$, then the computation of kP modulo n fails and a non-trivial divisor of n (which should be p) is found.*

4.3.4 Examples

Here we give an example of the computation over Jacobian groups and factorization using them.

Example We factor $77 = 7 \cdot 11$ using the following curve of genus 2:

$$C: \ y^2 = x^5 + 3x + 40 \pmod{77}$$

and the point $P = (2, 1)$ on the curve. P corresponds to the divisor $D = \mathrm{div}(u - 2, 1)$.

First we compute multiples of D over $\mathbf{J}(\mathbf{F}_7)$:

$$
\begin{aligned}
1D &= \mathrm{div}(u+5,\ 1)\\
2D &= \mathrm{div}(u^2+3u+4,\ 3u+2)\\
3D &= \mathrm{div}(u^2+4u+4,\ u+5)\\
4D &= \mathrm{div}(u^2+3u+2,\ 2u+1)\\
5D &= \mathrm{div}(u^2+2u+5,\ 3u+5)\\
6D &= \mathrm{div}(u^2+5u+1,\ 6u+4)\\
7D &= \mathrm{div}(u^2+3u+2,\ 3u+2)\\
8D &= \mathrm{div}(u^2+6u+5,\ 4u+5)\\
9D &= \mathrm{div}(u+1,\ 1)\\
10D &= \mathrm{div}(u^2+6u+5,\ u+6)\\
11D &= \mathrm{div}(u^2+2u+1,\ 3u+2)\\
12D &= \mathrm{div}(u^2+3,\ 4u+5)\\
13D &= \mathrm{div}(u+2,\ 4)\\
14D &= \mathrm{div}(u^2+3,\ u+6)\\
15D &= \mathrm{div}(u^2+3,\ 6u+1)\\
16D &= \mathrm{div}(u+2,\ 3)
\end{aligned}
$$

$$
\begin{aligned}
17D &= \mathrm{div}(u^2+3,\ 3u+2)\\
18D &= \mathrm{div}(u^2+2u+1,\ 4u+5)\\
19D &= \mathrm{div}(u^2+6u+5,\ u+6)\\
20D &= \mathrm{div}(u+1,\ 6)\\
21D &= \mathrm{div}(u^2+6u+5,\ 3u+2)\\
22D &= \mathrm{div}(u^2+3u+2,\ 4u+5)\\
23D &= \mathrm{div}(u^2+5u+1,\ u+3)\\
24D &= \mathrm{div}(u^2+2u+5,\ 4u+2)\\
25D &= \mathrm{div}(u^2+3u+2,\ 5u+6)\\
26D &= \mathrm{div}(u^2+4u+4,\ 6u+2)\\
27D &= \mathrm{div}(u^2+3u+4,\ 4u+5)\\
28D &= \mathrm{div}(u+5,\ 6)\\
29D &= \mathrm{div}(1,\ 0)
\end{aligned}
$$

The order of D is 29. If we compute the order of the group $\mathbf{J}(\mathbf{F}_7)$, we get 58, and thus we can confirm the fact that the order of a element is divisible by the order of the group.

Next we compute multiples of D over $\mathbf{J}(\mathbf{F}_{11})$:

$$
\begin{aligned}
1D &= \mathrm{div}(u+9,\ 1)\\
2D &= \mathrm{div}(u^2+7u+4,\ 3u+6)\\
3D &= \mathrm{div}(u^2+8u+5,\ 5u+2)\\
4D &= \mathrm{div}(u^2+5u+3,\ 6u+9)\\
5D &= \mathrm{div}(u^2+3u+10,\ 8u+8)\\
6D &= \mathrm{div}(u^2+7u+7,\ 5u+7)\\
7D &= \mathrm{div}(u^2+2u+9,\ 7u+8)\\
8D &= \mathrm{div}(u^2+7u+6,\ 3u+8)\\
9D &= \mathrm{div}(u^2+5u+8,\ 10u+1)\\
10D &= \mathrm{div}(u+7,\ 8)\\
11D &= \mathrm{div}(u^2+5u+8,\ 9u+5)\\
12D &= \mathrm{div}(u^2+10u+4,\ 6u+5)\\
&\cdots\cdots\cdots\\
60D &= \mathrm{div}(u^2+8u+5,\ 6u+9)\\
61D &= \mathrm{div}(u^2+7u+4,\ 8u+5)\\
62D &= \mathrm{div}(u+9,\ 10)\\
63D &= \mathrm{div}(1,\ 0)
\end{aligned}
$$

The order of D is 63. (The order of the group $\mathbf{J}(\mathbf{F}_{11})$ is 126.)

Last we compute multiples of D modulo 77. (Of course the actual factorization algorithm only does this, and multiplication is done by using the repeated doubling.)

$$
\begin{aligned}
1D &= \mathrm{div}(u+75,\ 1)\\
2D &= \mathrm{div}(u^2+73u+4,\ 3u+72)\\
3D &= \mathrm{div}(u^2+74u+60,\ 71u+68)
\end{aligned}
$$

$$
\begin{aligned}
4D &= \mathrm{div}(u^2 + 38u + 58,\ 72u + 64) \\
5D &= \mathrm{div}(u^2 + 58u + 54,\ 52u + 19) \\
6D &= \mathrm{div}(u^2 + 40u + 29,\ 27u + 18) \\
7D &= \mathrm{div}(u^2 + 24u + 9,\ 73u + 30) \\
8D &= \mathrm{div}(u^2 + 62u + 61,\ 25u + 19)
\end{aligned}
$$

The degree of a of $9D$ is 1 over $\mathbf{J}(\mathbf{F}_7)$ and 2 over $\mathbf{J}(\mathbf{F}_{11})$, thus the computation stopped there, and as a result the divisor 7 was found.

4.4 Order of Groups

In this section we discuss the order of Jacobian groups, which is of great importance in designing secure cryptosystems. From Proposition 4.1, we know that in HECM we do not require the condition kD be equal to the zero divisor so as to get a non-trivial divisor. Nevertheless this is at least a sufficient condition. Thus it still makes sense to hope that the order of groups is smooth.

Concerning the range of the order of Jacobian groups of hyperelliptic curves, the following theorem (a counterpart of Hasse's Theorem of elliptic curves) is known (for the proof see [5]):

Theorem 4.1 *Let $\mathbf{J}(\mathbf{F}_p)$ be the Jacobian of a hyperelliptic curve of genus g defined over $\mathbf{F}_p$. Then*

$$(\sqrt{p} - 1)^{2g} \leq \#\mathbf{J}(\mathbf{F}_p) \leq (\sqrt{p} + 1)^{2g}$$

It shows the order of Jacobian groups is much larger than that of elliptic curves. As a result hyperelliptic curves have smaller possibility that the order be smooth than elliptic curves. This fact is favorable to cryptography, but not to integer factorization.

We examined orders of Jacobian groups over very small fields with brute-force enumeration: out of candidates of $\mathrm{div}(a, b)$ count ones that satisfy the definition of reduced divisors.

Example The order of the Jacobian of the curve $y^2 = x^5 + 3x^3 + 5$ over $\mathbf{F}_p$.

p	order
7	$58 = 2 \cdot 29$
11	$152 = 2^3 \cdot 19$
13	$238 = 2 \cdot 7 \cdot 17$
17	$357 = 3 \cdot 7 \cdot 17$
19	$416 = 2^5 \cdot 13$
23	$751 = 751$
29	$729 = 3^6$
31	$990 = 2 \cdot 3^2 \cdot 5 \cdot 11$
37	$1200 = 2^4 \cdot 3 \cdot 5^2$
41	$1856 = 2^6 \cdot 29$
43	$2048 = 2^{11}$
47	$2256 = 2^4 \cdot 3 \cdot 47$
53	$2300 = 2^2 \cdot 5^2 \cdot 23$
59	$3624 = 2^3 \cdot 3 \cdot 151$
61	$4937 = 4937$

The above theorem shows that the Jacobian of a hyperelliptic curve of genus 2 defined over $\mathbf{F}_p$ is approximately equal to p^2, which result is in agreement with this example.

4.5 Implementation

We implemented the algorithm of HECM, using hyperelliptic curves of the following form (genus 2):

$$y^2 = x^5 + ax^3 + bx^2 + cx + d.$$

(By setting $h(u) = 0$, we can judge whether curves are smooth (nonsingular) by the condition the right hand side have no multiple roots. Omitting x^4-term does not lose the generality.) After that we performed the following experiments.

[Hyperelliptic curve method]

$l \setminus B$		1000	2000	4000	8000	16000
6	c	64	119	212	369	522
	t_f	2.537	5.194	10.203	20.931	41.266
	T	*39.63*	43.64	48.12	56.72	79.05
8	c	0	2	7	18	43
	t_f	2.881	5.762	11.628	23.225	46.984
	T	—	(2881)	1661	1290	*1092*
10	c	0	0	1	1	3
	t_f	3.092	6.122	12.424	25.009	49.966
	T	—	—	(12423)	(25009)	(16655)

Table 4.1: Running time and number of successes: There c is the number of successes out of 1000 times, t_f is the average running time in failure cases (sec), and T is the expected time until success (sec).

Experiment We generate 100 composite numbers which are products of two l-decimal-digit prime numbers and apply the method to each number 10 times. And then we examine how many times in total it succeeds in finding a non-trivial divisor. We also measure running time. This procedure is carried out with various values of B (upper bound of smoothness).

The condition of the experiment is the same as in Chapter 3.

The results are shown in Table 4.1.

4.6 Evaluation

From the experimental results, we suggest the following:

- HECM takes over ten times as much time as ECM. It is because operations on the Jacobian need very long time. Naiveness of the implementation may be another reason.

- The probability of success fall drastically to less than 1% of ECM. We guess that the increase in orders has large influence. This result seems much worse than that expected from theory mentioned in the first section. We must investigate the cause.

Here ECM means Lenstra's original (naive) method. Combining these evaluations, we suggest that the expected running time of HECM is over 1000 times longer than that of ECM (when $l = 10$). According to Lenstra, Pila and Pomerance's theoretical result, the ratio is

$$\left.\frac{L_p[1/2, 2]}{L_p[1/2, \sqrt{2}]}\right|_{p=10^{10}} = 145$$

when $l = 10$, and our experimental result is much worse. We think that it is mainly because of the naive computations in Jacobian groups, but we must investigate the reason further.

Lenstra, Pila and Pomerance's analysis is based on the probability that the order of a Jacobian group of a hyperelliptic curve is smooth. As we have seen in the previous chapters, this is a sufficient condition of the success of HECM. However this fact does not seem to make up for the fault.

Chapter 5

Conclusions

First we implemented three kinds of ECM algorithms, from the original one to the improved one, and confirmed the following well-known facts through experiments:

- Use of Montgomery-form curves rather than Weiestrauss-form ones makes computation about two times faster.
- Choosing suitable curves increases the probability that factors are found (about 40%).
- 2nd Step largely increases the probability that factors are found with a small increase in running time.

Next we investigated factoring using hyperelliptic curves. In particular we evaluated the inferiority quantitatively through experiments, and as a result we found

- Factorization using hyperelliptic curves takes over ten times as much time as ECM, and the probability of success is less than 1% of ECM.

The results were much worse than that expected theoretically. We must investigate the cause.

In addition we also considered some simple examples of the order of Jacobian groups of hyperelliptic curves, and confirmed the fact:

- The order of Jacobian groups of hyperelliptic curves of genus 2 defined over $\mathbf{F}_p$ is approximately equal to p^2.

We want to examine the order of Jacobian groups over larger fields.

As a result, HECM was not practical. But we want to study further computations in Jacobian groups, because this is also related to hyperelliptic cryptosystem, which is known to be more practical.

References

[1] T. Izu: "Fast Computation on Elliptic Curve Method of Integer Factorization" (in Japanese), 情報処理学会研究報告, Vol.99, No.72, pp.53–60 (1999)

[2] Y. Kida and I. Makino: *Computer Number Theory in UBASIC* (in Japanese), Nihon Hyoronsha (1994)

[3] N. Koblitz: *A Course in Number Theory and Cryptography, 2nd Edition*, GTM 114, Springer-Verlag (1987)

[4] N. Koblitz: "Hyperelliptic Cryptosystems", *J. Cryptology*, Vol.1, pp.139–150 (1989)

[5] N. Koblitz: *Algebraic Aspects of Cryptography*, Springer-Verlag (1998)

[6] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse and J. M. Pollard: "The Number Field Sieve", *Proc. 22nd STOC*, pp.564–572 (1990)

[7] H. W. Lenstra, Jr.: "Factoring Integers with Elliptic Curves", *Annals of Math.*, pp.649–673 (1987)

[8] H. W. Lenstra, Jr., J. Pila, C. Pomerance: "A Hyperellictic Smoothness Test. I", *Philos. Trans. Roy. Soc. London*, Vol.345, pp.397–408 (1993)

[9] P. L. Montgomery: "Speeding the Pollard and Elliptic Curve Methods for Factorizations", *Math of Comp.*, Vol.48, pp.243–264 (1987)

[10] J. M. Pollard: "Theorems on Factorization and Primality Testing", *Proc. Cambridge Philos. Soc*, Vol.76, pp.521–528 (1974)

[11] J. M. Pollard: "A Monte Carlo Method for Factorization", *BIT*, Vol.15, pp.331-334 (1975)

[12] C. Pomerance: "The Quadratic Sieve Algorithm", *Lecture Notes in Computer Science*, Vol.209, pp.169–182 (1985)

[13] Y. Sakai and K. Sakurai: "Design of Hyperelliptic Cryptosystems in Small Characteristic and a Software Implementation over $\mathbf{F}_{2^n}$", *Advances in Cryptology – ASIACRYPT'98*, LNCS, Vol.1514, pp.80–94 (1998)

[14] Y. Sakai and K. Sakurai: "On the Efficiency of Hyperelliptic Cryptosystems — $\mathbf{J}(\mathbf{F}_p)$ vs. $\mathbf{J}(\mathbf{F}_{2^n})$ in Software Implementation —" (in Japanese) SCIS99 (1999)

[15] D. Takahashi, Y. Torii and T. Yuasa: "An Implementation of Factorization on Massively Parallel SIMD Computers" (in Japanese), 情報処理学会論文誌, Vol.36, No.11, pp.2521–2530 (1995)