國立臺灣大學理學院數學系

碩士論文

Department of Mathematics
College of Science
National Taiwan University

Master Thesis

計算二元體上橢圓曲線群之群秩：SEA 演算法的實作

Implementations of SEA Algorithm Counting the Orders of Elliptic Curve Groups over Binary Fields

林子桓

Tzu-Huan Lin

指導教授：陳君明 博士

Advisor: Jiun-Ming Chen, Ph.D.

中華民國 98 年 1 月

January, 2009

Acknowledgements

First I want to thank my thesis advisor professor Jiun-Ming Chen for his guidance and help in the last year of my graduate student career, and his answering to all my questions in this field. And I want to thank Chih-Hung Lin for his help in writing codes and all things about computers. I also want to thank Yu-Chen Wang and other classmates for their discussion with me when I had problems. Finally I want to thank my parents and my family, they always support and encourage me when I got into trouble. Without any of their help, this thesis can never be done.

摘要

目前尋找安全橢圓曲線的最好方法爲群秩計算法 (Point-counting Method)。Schoof-Elkies-Atkin 演算法 (SEA 演算法) 爲質數體上計算橢圓曲線群秩最有效率的演算法。Lercier 提出了計算二元體上同源的方法，使得 SEA 演算法也可應用基於二元體的橢圓取線上。這篇論文我們將依據 Lercier 提出的方法實作 SEA 演算法，用它來計算美國國家標準和技術研究院 (NIST) 推薦的十條曲線的群秩並觀察其效率。

關鍵字：橢圓曲線、群秩、SEA 演算法、Schoof 演算法、Elkies 質數、Atkin 質數

Abstract

The best suggested way to find secure elliptic curves is point-counting. So far Schoof-Elkies-Atkin algorithm (SEA algorithm) is the most efficient point-counting algorithm for elliptic curves over prime fields. Lercier proposed an algorithm to compute isogenies in $\mathbb{F}_{2^n}$ such that SEA algorithm can be used for binary case. In this thesis we will follow Lericier's approach to implement SEA algorithm computing the order of an elliptic curve over binary fields.

Keywords: elliptic curve, group order, SEA algorithm, Schoof algorithm, Elkies prime, Atkin prime

Contents

1 Introduction

After being proposed by Koblitz and Miller independently around 1985, the elliptic curve cryptosystem (ECC) became one of the most popular public key cryptosystems these years. The security of ECC is based on the difficulty of the discrete logarithm problem on elliptic curve groups (ECDLP). One important reason of that ECC is widely used is that there is no known sub-exponensual algorithm to solve ECDLP so far. Moreover, in comparison with RSA, another popular public key cryptosystem, ECC can have the same level of security with much shorter keys. This makes ECC suitable for environments with limited storage and power, for example, a smart card.

However, not all elliptic curves are suitable for cryptographic use. An elliptic curve is considered to be secure only when its order is nearly prime. The best way to find secure curves is point-counting. That is, pick an arbitrary curve and count points on it, then see if it satisfies the conditions we need.

Schoof's algorithm is the first polynomial time point-counting algorithm. It was presented by Schoof in 1985, and has complexity $O(\log^8 q)$. However it is extremely inefficient when the size of base field is large. One of the main reason is that the degrees of the division polynomials are so large that the calculations are impracticable. Elkies and Atkin dealt with the problem to give the Schoof-Elkies-Atkin algorithm (SEA algorithm [6]),their improvements degrades the complexity to $O(\log^5 q)$. Also thanks to the improvements of Morain, Couveignes, Dewaghe, Müller, SEA algorithm is the most efficient point-counting algorithm so far, and we can use it to generate a secure elliptic curve over prime fields in a reasonable time.

However, elliptic curves over binary fields are of more interests in cryptography. The problem is given two elliptic curves over binary fields, how to compute isogenies

between them. Couveignes gave an algorithm to deal with this problem. It works in the formal group defined by the elliptic curve. But this algorithm requires huge series computations, it turns out to be inefficient in practice.

Lercier proposed another algorithm to compute isogenies in [2]. It is based on identities satisfied by them. Lercier's algorithm has similar complexity as Couveignes's algorithm, but it is easier to implement and more efficient in practice. In this thesis we will follow Lercier's approach to compute isogenies, and implement SEA algorithm to compute the order of elliptic curves over binary fields.

The rest of the article is structured as follows: in Section 2 we give some mathematical backgrounds which are needed in Schoof's algorithm and SEA algorithm. In Section 3 we give an overview of Schoof's algorithm and SEA algorithm. In Section 4 we explain how to compute modular polynomials over binary fields and introduce Lercier's method to compute isogenies between elliptic curves. In Section 5 we compare our program with an open source implementation of Schoof algorithm given in [7]. And parts of our source code is given in the appendix.

2 Mathematical Backgrounds

In this section we introduce some mathematical backgrounds about Schoof's algorithm and SEA algorithm. Here we list some notations used a lot in this thesis.

q	a power of 2
$\mathbb{F}_q$	the finite field with q elements
$\overline{\mathbb{F}}_q$	algebraic closure of $\mathbb{F}_q$
$\overline{E}_b$	a fixed elliptic curve over $\mathbb{F}_q$ with parameter b
E_b	the set of rational points on $\overline{E}_b$
$\#S$	cardinality of the set S
$\mathcal{O}$	the point at infinity
t	the trace of Frobenius
t_l	the residue of t modulo l
T_l	the set of possible values for t modulo l
$[m]$	multiplication-by-m map
$E[m]$	m-torsion subgroup
$\mathbb{Z}_m$	cyclic group of order m
φ	Frobenius map on $\overline{E}_b$
$\psi_l(x)$	l-th division polynomial
$F_l(x)$	a factor of $\psi_l(x)$
$\Phi_l(x, y)$	l-th modular polynomial
$\mathbb{Z}$	the set of intergers
$\mathbb{C}$	the set of complex numbers

Suppose q is a power of two, and suppose a, b are two elements in $\mathbb{F}_q$. We denote $\overline{E}_{a,b}$ the elliptic curve over $\mathbb{F}_q$ defined by

$$\overline{E}_{a,b} = \{(x, y) \in \overline{\mathbb{F}}_q \times \overline{\mathbb{F}}_q \mid y^2 + xy = x^3 + ax^2 + b\} \cup \{\mathcal{O}\},$$

where $\mathcal{O}$ is the point at infinity on $\overline{E}_{a,b}$. When $b = 1$, $\overline{E}_{a,b}$ is called an Koblitz curve. Then let

$$E_{a,b} = \{(x, y) \in \overline{E}_{a,b} \mid x, y \in \mathbb{F}_q\} \cup \{\mathcal{O}\},$$

which is called the set of rational points on $\overline{E}_{a,b}$. We are trying to find the cardinality of $E_{0,b}$ for every $b \in \mathbb{F}_q^*$ (that is, $b \in \mathbb{F}_q$ with $b \neq 0$).

Remark 2.1. An elliptic curve $\overline{E}_{a,b}$ is called supersingular if $b = 0$. The difficulty of ECDLP on a supersingular curve is lowered by MOV attack [4], so we are only interested in non-supersingular curves. By Theorem 3.5 in [3], every non-supersingular elliptic curve over $\mathbb{F}_q$ is isomorphic (in group structure) to $E_{a,b}$ for some $a \in \{0, \gamma\}$ and $b \in \mathbb{F}_q^*$, where γ is a fixed element in $\mathbb{F}_q$ such that $\mathrm{Tr}(\gamma)$=1. ($\mathrm{Tr}(\gamma)$ is the trace of γ in $\mathbb{F}_q$ over $\mathbb{F}_2$.) Moreover, since (see [1], P.38)

$$\#E_{0,b} + \#E_{\gamma,b} = 2q + 2,$$

we need only to find $\#E_{0,b}$ for every $b \in \mathbb{F}_q^*$. $\overline{E}_{0,b}$ and $E_{0,b}$ are also denoted by $\overline{E}_b$ and E_b respectively.

Let b be a fixed nonzero element in $\mathbb{F}_q$, and let

$$t = q + 1 - \#E_b. \tag{1}$$

t is called the *trace of Frobenius*, it is just what we want to find out (since once t is found, then $\#E_b$ is obtained by Equation 1). Hasse's theorem gives a bound of t.

Theorem 2.2. (Hasse) *The trace of Frobenius satisfies*

$$|t| \leq 2\sqrt{q}.$$

Remark 2.3. By Hasse's theorem, if we can find the residue of t modulo l (t_l for short) for enough primes l such that $\prod l > 4\sqrt{q}$, then the exact value of t can be obtained by Chinese Remainder Theorem. How to find t_l is the main part of Schoof's algorithm and SEA algorithm, we will come back to this in the next section.

It is well-known that $\overline{E}_b$ has a group structure with the *chord-tangent operation*. With this operation we can give the definition of *multiplication-by-m maps* and *m-torsion subgroups*, which are of much importance in elliptic curve theory.

Definition 2.4. (multiplication-by-m map and m-torsion subgroup)
Let m be an integer, the multiplication-by-m map $[m] : \overline{E}_b \to \overline{E}_b$ is defined by

$$[m]P = \begin{cases} P + P + \cdots + P \text{ (m summands)}, & \text{if } m > 0, \\ \mathcal{O}, & \text{if } m = 0, \\ -(P + P + \cdots + P) \text{ ($-m$ summands)}, & \text{if } m < 0, \end{cases}$$

for every $P \in \overline{E}_b$, where '+' denotes the group operation on $\overline{E}_b$.

The *m-torsion subgroup* $E[m]$ is defined by

$$E[m] = \{P \in \overline{E}_b \mid [m]P = \mathcal{O}\}.$$

One can see that $E[m]$ is a subgroup of $\overline{E}_b$. However, we can describe the structure of $E[m]$ by the following lemma.

Lemma 2.5. *Suppose m is a positive integer with $(m, q) = 1$, then*

$$E[m] \cong \mathbb{Z}_m \oplus \mathbb{Z}_m,$$

where $\mathbb{Z}_m$ is the cyclic group of order m.

Now we give another important definition.

Definition 2.6. (Frobenius Map)

The Frobenius map φ on $\overline{E}_b$ is defined by

$$\varphi : \overline{E}_b \to \overline{E}_b : \begin{cases} (x, y) & \mapsto & (x^q, y^q), \\ \mathcal{O} & \mapsto & \mathcal{O}. \end{cases}$$

φ can be checked to be a group endomorphism of $\overline{E}_b$. Moreover, φ satisfies the following identity.

Theorem 2.7. *The Frobenius map φ on $\overline{E}_b$ satisfies*

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

That is,

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

for every $(x, y) \in \overline{E}_b$.

Finally we introduce the division polynomials.

Definition 2.8. (Division Polynomial)

For each nonnegative integer m, the m-th division polynomial $\psi_m(x) \in \mathbb{F}_q[x]$ with

respect to $\overline{E}_b$ is defined recusively by the following formulas:

$$\begin{aligned}
\psi_0 &= 0,\\
\psi_1 &= 1,\\
\psi_2 &= x,\\
\psi_3 &= x^4 + x^3 + b,\\
\psi_4 &= x^6 + bx^2,\\
\psi_{2m+1} &= \psi_{m+2}\psi_m^3 + \psi_{m-1}\psi_{m+1}^3,\ m \geq 2,\\
\psi_{2m} &= (\psi_{m+2}\psi_{m-1}^2 + \psi_{m-2}\psi_{m+1}^2)\psi_m/x,\ m > 2.
\end{aligned}$$

By the group law of elliptic curves and mathematical induction, one can check that the multiplication-by-m map $[m]$ and the division polynomial ψ_m satisfy the following equality:

$$[m]P = (x + \frac{\psi_{m-1}\psi_{m+1}}{\psi_m^2}, x + y + \frac{(x^2 + x + y)\psi_{m-1}\psi_m\psi_{m+1} + \psi_{m-2}\psi_{m+1}^2}{x\psi_m^3}), \quad (2)$$

for each $P = (x, y) \in \overline{E}_b \backslash E[m]$. Moreover, we have

Lemma 2.9. *Let $P = (x, y)$ be a point in $\overline{E}_b \backslash E[2]$ and $m \geq 2$ an integer. Then $P \in E[m]$ if and only if $\psi_m(x) = 0$.*

Now we are in a place to introduce Schoof's algorithm and SEA algorithm.

3 SEA Algorithm

In this section we give an overview of Schoof's algorithm and SEA algorithm.

3.1 Schoof's Algorithm

As mentioned before, Schoof's algorithm is an algorithm to count points on elliptic curves. It was presented by Schoof in 1985, and has time complexity $O(\log^8 q)$.

Let $b \in \mathbb{F}_q^*$ and $\overline{E}_b : y^2 + xy = x^3 + b$. By Remark 2.3, to find $\#E_b$ we need only to find t_l (the residue of t modulo l) for enough primes l.

Suppose l is an odd prime. We reduce the Frobenius map φ on $E[l]$, then by Theorem 2.7, we have

$$\varphi^2(P) - [t_l]\varphi_l(P) + [q_l](P) = \mathcal{O}$$

for every $P \in E[l]$, where $q_l = q \pmod{l}$. Or equivalently,

$$(x^{q^2}, y^{q^2}) + [q_l](x, y) = [t_l](x^q, y^q), \tag{3}$$

for every $(x, y) \in E[l]$.

The left hand side of Equation 3 can be represented by $(R_1(x, y), R_2(x, y))$ for some rational functions R_1, R_2 in $\mathbb{F}_q(x, y)(= \{f/g \mid f, g \in \mathbb{F}_q[x, y] \text{ with } g \neq 0\})$ by using group law and Equation 2. Meanwhile, by Equation 2 again, for every $\tau \in \{0, 1, 2, \ldots, l-1\}$, $[\tau](x^q, y^q)$ can also be represented by $(R_1^\tau(x, y), R_2^\tau(x, y))$ for some R_1^τ, $R_2^\tau \in \mathbb{F}_q(x, y)$.

Therefore, if we can find a value $\tau \in \{0, 1, 2, \ldots, l-1\}$ such that

$$R_1(x, y) \equiv R_1^\tau(x, y) \pmod{\psi_l(x), y^2 + xy + x^3 + b} \tag{4}$$

and

$$R_2(x, y) \equiv R_2^\tau(x, y) \pmod{\psi_l(x), y^2 + xy + x^3 + b}, \tag{5}$$

by Lemma 2.9, we must have

$$(x^{q^2}, y^{q^2}) + [q_l](x, y) = [\tau](x^q, y^q),$$

for every $(x, y) \in E[l]$, and so $t_l = \tau$.

After finding t_l for enough l such that $\prod l > 4\sqrt{q}$, by Hasse's theorem and Chinese remainder theorem (CRT) we can find the exact value of t. The following is the summarization of Schoof's algorithm.

Schoof's Algorithm

Input: an elliptic curve $\overline{E}_b : y^2 + xy = x^3 + b$

Output: $\#E_b$

1. Pick enough primes l such that $\prod l > 4\sqrt{q}$
2. For each l, find τ from $\{0,1,\ldots,l-1\}$ such that Equation 4 and 5 holds.
3. Set $t_l = \tau$.
4. Use CRT to recover the value of t.
5. return $q + 1 - t$.

Schoof's algorithm becomes extremly inefficient when q is large. The main reason is that the division polynomial ψ_l has degree $(l^2 - 1)/2$ such that we have to do too many calculations (in Step 2 of the above algorithm) when l is large. We will use another kind of polynomial F_l instead of ψ_l in SEA algorithm. But before introducing this, we should introduce modular polynomials, which plays an important role in SEA algorithm.

3.2 Modular Polynomials

Modular polynomials play an important role in SEA algorithm. Here we give the definition and some properties of them, and we will discuss how to compute them next section.

Let $\mathbb{H}$ be the upper half plane of $\mathbb{C}$ (the field of complex numbers), that is, $\mathbb{H} = \{z \in \mathbb{C} \mid \mathrm{Im}(z) > 0\}$. It can be shown that for every $\tau \in \mathbb{H}$, one can find an elliptic curve $E(\tau)$ over $\mathbb{C}$ such that

$$E(\tau) \cong \frac{\mathbb{C}}{\mathbb{Z} + \mathbb{Z}\tau}. \tag{6}$$

So we can define a map E from $\mathbb{H}$ to the set $\mathcal{I}$ of isomorphism classes of elliptic curves over $\mathbb{C}$ by

$$E : \mathbb{H} \to \mathcal{I}, \tau \mapsto E(\tau),$$

where $E(\tau)$ is any curve which satisfies Equation 6.

Clearly this map is well-defined (in fact it is onto), and if we denote $\Delta(\tau)$ and $j(\tau)$ the discriminant and j-invariant of $E(\tau)$ respectively, with denoting $q = e^{2\pi i\tau}$, $\Delta(\tau)$ and $j(\tau)$ can be represented as the following form:

$$\Delta(\tau) = q\prod_{n\geq 1}(1-q^n)^{24} = q(1+\sum_{n\geq 1}(-1)^n(q^{n(3n-1)/2}+q^{n(3n+1)/2}))^{24} \tag{7}$$

and

$$j(\tau) = \frac{(256f(\tau)+1)^3}{f(\tau)}, \tag{8}$$

where $f(\tau) = \Delta(2\tau)/\Delta(\tau)$.

Let l be an odd prime, and let

$$\Phi_l(x, j(\tau)) = (x - j(l\tau))\prod_{i=0}^{l-1}(x - j(\frac{\tau+i}{l})).$$

It can be shown that $\Phi_l(x, j(\tau)) \in \mathbb{Z}[x, j(\tau)]$. The l-th modular polynomial $\Phi_l(x, y)$ is defined by replacing $j(\tau)$ in $\Phi_l(x, j(\tau))$ by y, which is an integral polynomial with two variables.

The modular polynomials $\Phi_l(x, y) \in \mathbb{Z}[x, y]$ are symmetric, and of degree $l+1$ in each variable. To reduce it to a finite field of characteristic p, we need only to modulo each coefficient of $\Phi_l(x, y)$ by p. In our case (over a binary field), we need to modulo 2 in each coefficient of $\Phi_l(x, y)$. However the coefficients of $\Phi_l(x, y)$ are very large and difficult to compute, fortunately there are some variants of modular polynomials, for example, Müller's modular polynomials. We will not discuss them here, but next section we will give an efficient algorithm to compute modular polynomials over binary fields, which is also presented by Müller [5].

3.3 SEA Algorithm

SEA algorithm is based on Schoof's algorithm, and improved by Elkies and Atkin. As said in the end of Section 3.1, we will replace the l-th division polynomial ψ_l in Schoof's algorithm by its factor F_l. The polynomial $F_l \in \mathbb{F}_q[x]$ has degree $(l-1)/2$, with comparison ψ_l with degree $(l^2-1)/2$, the computations in $\mathbb{F}_q[x]/(F_l(x))$ are much less and so are more practicable while l is large. However only about one half of primes have this advantage. We call this kind of primes *Elkies primes*, the other half are called *Atkin primes*. Before we introduce them officially, we have to recall some mathematical facts.

Given an odd prime l, by Lemma 2.5, $E[l]$ is isomorphic to $\mathbb{Z}_l \oplus \mathbb{Z}_l$. Moreover, since l is a prime, $E[l]$ can be regarded as a two-dimensional linear space over $\mathbb{F}_l$. Consider the Frobenius map φ restricted on $E[l]$, then $\varphi : E[l] \to E[l]$ is a linear transformation. Moreover, by Theorem 2.7 we have the following lemma.

Lemma 3.1. *The Frobenius map φ is a linear transformation on $E[l]$. Moreover, the characteristic polynomial of φ*

$$\Delta_\varphi(x) = x^2 - t_l x + q_l,$$

where $t_l = t \pmod{l}$, $q_l = q \pmod{l}$.

If the characteristic polynomial Δ_φ has a root λ in $\mathbb{F}_l$, then the other solution must be q_l/λ (λ cannot be zero). So $\Delta_\varphi(x) = (x-\lambda)(x-q_l/\lambda)$. Hence

$$t_l = \lambda + \frac{q_l}{\lambda}. \tag{9}$$

So if we can find an eigenvalue λ of the Frobenius map $\varphi : E[l] \to E[l]$, then we can find t_l by Equation 9. This can only be done in the situation Δ_φ has a root in $\mathbb{F}_l$, we call a prime satisfying this condition an Elkies prime.

Definition 3.2. With respect to an elliptic curve $\overline{E}_b$, a prime l is called an *Elkies prime* if $\Delta_t = t^2 - 4q$ is a square in $\mathbb{F}_l$. Otherwise l is called an *Atkin prime*.

If l is an Elkies prime, one can find t_l by searching for an eigenvalue λ (we will discuss more details later). However, since we do not know what value t is (this is what we are trying to find out), we cannot decide which kind of prime l is by the above definition. The following theorem gives us a way to do that, which is presented by Atkin.

Theorem 3.3. (Atkin) *Let $\overline{E}_b$ be a non-supersingular elliptic curve with j-invariant j*[1]*. And suppose $\Phi_l(x, j) = h_1 h_2 \dots h_s$ is the factorization of $\Phi_l(x, j)$ in $\mathbb{F}_q[x]$ as a product of irreducible polynomials, where $\Phi_l(x, y)$ is the l-th modular polynomial over binary fields. Then there are the following possibilities for the degrees of $h_1, h_2, \dots, h_s$:*

(i) (l,1) *or* (1,1,...,1): *in both cases $t^2 - 4q \equiv 0 \pmod{l}$. In the former case we set $r = l$ and in the latter $r = 1$.*

(ii) (1,1,r,r,...,r): *in this case $t^2 - 4q$ is a square modulo l. r divides $l - 1$ and φ acts on $E[l]$ as a matrix*

$$\begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix},$$

where λ, $\mu \in \mathbb{F}_l^*$.

(iii) (r,r,...,r): *in this case $t^2 - 4q$ is not a square modulo l, r divides $l + 1$ and φ has an irreducible characteristic polynomial over $\mathbb{F}_l$.*

In all cases r is the order of φ in the projective general linear group $PGL_2(\mathbb{F}_l)$ and

[1]Recall that the j-invariant of the elliptic curve $\overline{E}_b$ is $1/b$.

the trace of Frobenius t satisfies

$$t^2 = q(\xi + \xi^{-1})^2 \pmod{l}, \tag{10}$$

for some primitive r-th root of unity ξ in $\overline{\mathbb{F}}_l$.

From the above theorem we can see that in the first two cases $t^2 - 4q$ is a square in $\mathbb{F}_l$ and is not in the third case. In other words, if the factorization of $\Phi_l(x, j)$ in $\mathbb{F}_q[x]$ satisfies case (i) or (ii), then l is an Elkies prime; otherwise l is Atkin. We can also observe that only in the first two cases $\Phi_l(x, j)$ has a factor of degree 1, this fact leads to the following corollary:

Corollary 3.4. *A prime l is an Elkies prime if and only if* $\gcd(x^q + x, \Phi_l(x, j))$ *in* $\mathbb{F}_q[x]$ *is nontrivial.*

By Corollary 3.4 we can decide a prime l is Elkies or Atkin by compute

$$\gcd(x^q + x, \Phi_l(x, j)).$$

If l is Elkies, we will use the polynomial $F_l(x)$ defined below to find an eigenvalue λ of $\varphi : E[l] \to E[l]$.

Definition 3.5. Suppose $\lambda \in \mathbb{F}_l$ is an eigenvalve of the Frobenius map $\varphi : E[l] \to E[l]$, and C_λ is a 1-dimensional eigenspace with respect to λ. Then we define

$$F_l(x) = \prod_{\pm P \in C_\lambda \setminus \{\mathcal{O}\}} (x - x(P)),$$

where $x(P)$ is the affine x-coordinate of P.

One can prove that $F_l(x) \in \mathbb{F}_q[x]$. And since $\#C_\lambda = l$, $F_l(x)$ has degree $(l-1)/2$. Lercier gave an algorithm to find F_l ([2]), we will introduce this in the next section.

As said before, the polynomial F_l can be used to find an eigenvalue λ. We want to find $\lambda \in \{1, 2, \ldots, l-1\}$ satisfying

$$\varphi(P) = [\lambda](P)$$

for each $P \in C_\lambda$, or equivalently (by Equation 2),

$$(x^q, y^q) = (x + \frac{\psi_{\lambda-1}\psi_{\lambda+1}}{\psi_\lambda^2}, x + y + \frac{(x^2+x+y)\psi_{\lambda-1}\psi_\lambda\psi_{\lambda+1} + \psi_{\lambda-2}\psi_{\lambda+1}^2}{x\psi_\lambda^3}), \quad (11)$$

for each $(x, y) \in C_\lambda$.

So by the definition of $F_l(x)$, we need to find λ such that

$$x^q \equiv x + \frac{\psi_{\lambda-1}\psi_{\lambda+1}}{\psi_\lambda^2} \pmod{F_l(x), y^2 + xy + x^3 + b}$$

and

$$y^q \equiv x + y + \frac{(x^2+x+y)\psi_{\lambda-1}\psi_\lambda\psi_{\lambda+1} + \psi_{\lambda-2}\psi_{\lambda+1}^2}{x\psi_\lambda^3} \pmod{F_l(x), y^2 + xy + x^3 + b},$$

hold, once such λ is found, by Equation 9 we can find t_l.

If l is an Atkin prime, since $\varphi : E[l] \to E[l]$ does not have an eigenvalue in $\mathbb{F}_l$, we cannot do the same thing as in the case that l is Elkies. However, by Equation 10 in Theorem 3.3, we can find candidates of $t \pmod{l}$. Here is what we do.

First of all, to determine r in Theorem 3.3, we compute

$$\gcd(x^{q^i} + x, \Phi_l(x, j)) \quad (12)$$

for $i = 2, 3, \ldots, l+1$, where j is the j-invariant of $\overline{E}_b$. The smallest i such that the gcd is equal to $\Phi_l(x, j)$ is just the value r. Moreover, by Theorem 3.3 we can see that r divides $l \pm 1$, this information can reduce some computations.

After r is found, suppose λ, μ are roots of

$$x^2 - t_l x + q_l = 0.$$

Then $\gamma_r = \lambda/\mu$ is then an element of order r in $\mathbb{F}_{l^2}$. By first finding a generator g of $\mathbb{F}_{l^2}^*$, we can compute all possibilities of $\gamma_r = \{g^{i(l^2-1)/r} \mid i = 1, 2, \ldots, r-1$ with $(i, r) = 1\}$. Then we can find a set of possible values for t_l by the following equations:

$$t_l = \lambda + \mu \pmod{l}, \tag{13}$$

$$q_l = \lambda\mu \pmod{l},$$

and

$$\gamma_r = \lambda/\mu.$$

Let d be a quadratic non-residue in $\mathbb{F}_l$ and write $\lambda = x_1 + x_2\sqrt{d}$, $\gamma_r = g_1 + g_2\sqrt{d}$ for x_1, x_2, g_1, $g_2 \in \mathbb{F}_l$. The possible values for g_1 and g_2 are already known, we want to find the possible values for x_1 and x_2. Since μ is the conjugate of λ, we have $\mu = x_1 - x_2\sqrt{d}$, and so

$$g_1 + g_2\sqrt{d} = \frac{\lambda}{\mu} = \frac{1}{q_l}(x_1^2 + dx_2^2 + 2x_1x_2\sqrt{d}).$$

Hence

$$q_l g_1 = x_1^2 + dx_2^2,$$

$$q_l g_2 = 2x_1x_2,$$

and

$$q_l = x_1^2 - dx_2^2.$$

By the first an the third equilities we get $x_1^2 = q_l(g_1 + 1)/2$, this can give at most two possibilities of x_1. And by Equation 13 we have

$$t_l = 2x_1.$$

Here we can see that there are at most $2\phi(r)$ possible values for t_l, where ϕ is the Euler ϕ-function.

After the procedures above are done for enough primes l such that $\prod l > 4\sqrt{q}$, that is, if l is Elkies we have found t_l, and if l is Atkin we have found a set T_l of possible values for t_l. We can combine these information to find t with Chinese remainder theorem (CRT) and baby-step giant-step algorithm (BSGS). The process is as follows. It was given by Müller in [5].

First we divide Atkin primes into two parts A_1 and A_2 such that their product of possible values of t_l (that is, $\prod \#T_l$) are roughly the same as each other. Let m_1, m_2 be the products of primes in A_1 and A_2 respectively. By using CRT we can find a set of possibilities for $t \pmod{m_1}$ and $t \pmod{m_2}$, say S_1 and S_2. That is,

$$t \equiv t_1 \pmod{m_1} \text{ for some } t_1 \in S_1$$

and

$$t \equiv t_2 \pmod{m_2} \text{ for some } t_2 \in S_2.$$

On the other hand, let m_3 be the product of Elkies primes and find t_3 such that

$$t \equiv t_3 \pmod{m_3}$$

by CRT. Then we can write

$$t = t_3 + m_3(m_1 r_2 + m_2 r_1) \tag{14}$$

for some integers r_1, r_2 with

$$r_1 \equiv \frac{t_1 - t_3}{m_2 m_3} \pmod{m_1}$$

and

$$r_2 \equiv \frac{t_2 - t_3}{m_1 m_3} \pmod{m_2},$$

where $t_1 \in S_1$, $t_2 \in S_2$. Since $m_1 m_2 m_3 > 4\sqrt{q}$, if we choose

$$0 \le t_3 < m_3 \text{ and } \lfloor \frac{-m_1}{2} \rfloor < r_1 \le \lfloor \frac{m_1}{2} \rfloor,$$

we must have $|r_2| \leq m_2$.

Now we pick a point P on E_b randomly. Since the group order of E_b is $q+1-t$, we must have

$$[q+1]P = [t_3 + m_3(m_1 r_2 + m_2 r_1)]P.$$

Rearranging this we get

$$[q+1-t_3]P - [r_1 m_2 m_3]P = [r_2 m_1 m_3]P.$$

Then we compute and store

$$Q_{r_1} = [q+1-t_3]P - [r_1 m_2 m_3]P$$

for possible values of r_1 with $|r_1| \leq \lfloor m_1/2 \rfloor$, this can be considered as the phase of the giant steps in BSGS. After that we compute

$$R_{r_2} = [r_2 m_1 m_3]P,$$

by the previous observation we need only to take r_2 with $|r_2| \leq m_2$. Once we find

$$Q_{r_1} = R_{r_2}$$

for some r_1, r_2, then t can be found by Equation 14.

Remark 3.6. This BSGS procedure above only works when $\#E_b$ is *nearly prime*. When $\#E_b$ has many small prime factors (this kind of curve is not of cryptographic interest), we need to choose more random points on E_b to determine the exact calue of t.

Here is the summarization of SEA algorithm.

SEA Algorithm

Input: an elliptic curve $\overline{E}_b$

Output: $\#E_b$

1. Pick enough primes l such that $\prod l > 4\sqrt{q}$
2. For each l, decide l is Elkies prime or Atkin prime by

   $\gcd(x^q + x, \Phi_l(x, j))$.
3. If l is Elkies, search an eigenvalue λ of φ by Equation 11.
4. Set $t_l = \lambda + q_l/\lambda$.
5. If l is Atkin, find candidates of t_l.
6. Use CRT and BSGS to recover the value of t.
7. return $q + 1 - t$.

---

4 Implementation of SEA Algorithm

Until now there are only two gaps left to implement SEA algorithm over binary fields. One is to find modular polynomials, the other is to find F_l. We fill the two gaps this section.

4.1 Computing Modular Polynomials

As introduced in Section 3, l-th modular polynomial $\Phi_l(x, y)$ can be used to decide a prime l is Elkies or Atkin, and is needed while we compute F_l. Here we give a method to compute $\Phi_l(x, y)$ over binary fields. This method was given by Müller in [5], and also can be found in [8].

First l-th modular polynomial $\Phi_l(x, y)$ over binary fields has the form

$$\Phi_l(x, y) = \sum_{i=0}^{l+1} \sum_{j=0}^{l+1} a_{ij} x^i y^j,$$

where $a_{ij} \in \mathbb{F}_2$.

Recall the definition in Section 3.2,

$$\Phi_l(x, j(\tau)) = (x - j(l\tau)) \prod_{i=0}^{l-1} (x - j(\frac{\tau + i}{l})), \tag{15}$$

where (by Equation 7 and 8)

$$j(\tau) = \frac{\Delta(\tau)}{\Delta(2\tau)} = q^{-1} \frac{1 + \sum_{n\geq 1} (q^{4n(3n-1)} + q^{4n(3n+1)})}{1 + \sum_{n\geq 1} (q^{16n(3n-1)} + q^{16n(3n+1)})}. \tag{16}$$

In particular, we have

$$\Phi_l(j(l\tau), j(\tau)) = 0,$$

or

$$j^{l+1}(\tau) + j^{l+1}(l\tau) = \sum_{i=0}^{l} \sum_{j=0}^{l} j^i(l\tau) j^j(\tau). \tag{17}$$

We will compare the leading power of both sides in Equation 17 to find the coefficients a_{ij}.

First observe from Equation 16, $j(\tau)$ has leading power -1. Since the q-expansion of $j(l\tau)$ is replacing q in Equation 16 by q^l (recall that $q = e^{2\pi i\tau}$), $j(l\tau)$ has leading power $-l$. Hence, the left hand side (LHS) of Equation 17 has leading power $-l(l+1)$. On the other hand, $j^i(l\tau)j^j(\tau)$ has leading power $-(li+j)$, since the right hand side of Equation 17 must also have leading power $-l(l+1)$, we need only to find i, $j \in \{0, 1, \ldots, l\}$ such that $\max\{li+j, lj+i\} = l(l+1)$. The solution is $i = l$, $j = l$, so we must have $a_{ll} = 1$. Then we substract $j^l(l\tau)j^l(\tau)$ from both sides of Equation 17 and compare their leading powers again. Let $p(L)$ be the leading power of LHS, we again want to find $i, j \in \{0, 1, \ldots, l\}$ such that $\max\{i + jl, j + il\} = -p(L)$, then set $a_{ij} = a_{ji} = 1$ (since $\Phi_l(x, y)$ is symmetric). Continue this process until the leading power of LHS is nonnegative, then all nonzero coefficients must be found. We summarize this algorithm as follows:

Algorithm to compute modular polynomials over binary fields

Input: an elliptic curve $\overline{E}_b$, a prime l

Output: $\Phi_l(x, y)$

1. Determine $j(\tau)$ with precision $l(l+1)+1$ by Equation 16
2. Compute $j^i(\tau)$ for $1 < i \leq l+1$
3. Determine $j^j(l\tau)$ for $1 \leq j \leq l+1$ by substituting q by q^l in $j^j(\tau)$
4. Set $L = j^{l+1}(\tau) + j^{l+1}(l\tau)$
5. while $(p(L) < 0)$ where $p(L)$ is the leading power of L
6. Determine (i, j) such that $\max\{li + j, lj + i\} = -p(L)$
7. Set $a_{ij} = a_{ji} = 1$
8. If $(i = j)$ set $L = L + j^i(l\tau)j^j(\tau)$
9. Else set $L = L + j^i(l\tau)j^j(\tau) + j^j(l\tau)j^i(\tau)$
10. Return $\sum_{i=0}^{l+1} \sum_{j=0}^{l+1} a_{ij} x^i y^j$

4.2 Computing Isogenies

In this section we give an algorithm to find F_l introduced in Section 3. Sometimes we call this process computing isogenies, because to find F_l is equivalent to find isogenies between elliptic curves. This algorithm was given by Lercier [2]. Lercier estimates the complexity of this algorithm is $O(l^3)$ field operations, based on heuristics and experimental evidence.

Suppose $E_{b_1} : y^2 + xy = x^3 + b_1$ is an elliptic curve over $\mathbb{F}_q$. Given an Elkies prime l and an eigenvalue λ of the Frobenius map $\varphi : E[l] \to E[l]$, recall that

$$F_l(x) = \prod_{\pm P \in C_\lambda \setminus \{\mathcal{O}\}} (x - x(P)),$$

where C_λ is the eigenspace corredponding to λ.

Note that F_l has degree $(l-1)/2$, and suppose

$$F_l(x) = \sum_{i=0}^{(l-1)/2} q_i^2 x^i. \tag{18}$$

Remark 4.1. Recall that every element in a binary field $\mathbb{F}_q$ has a unique square root, so we can assume $F_l(x)$ has the form in Equation 18.

To find q_i's, first let $j_1 = 1/b_1$, which is the j-invariant of $\overline{E}_{b_1}$. Since l is an Elkies prime, the polynomial $\Phi_l(x, j_1) \in \mathbb{F}_q[x]$ has a root in $\mathbb{F}_q$, pick one and call it j_2 (this is the j-invariant of another curve which is isogenous to $\overline{E}_{b_1}$). Let $b_2 = 1/j_2$, and let

$$\alpha = \sqrt[4]{b_1}, \beta = \sqrt[4]{b_2}.$$

For convience we assume

$$q_i = \frac{\sqrt[4]{\alpha}}{\sqrt[4]{\beta}} \sqrt{\alpha}^{d-2i} p_{d-i}. \tag{19}$$

To find F_l we need only to find these p_i's. First of all, p_0, p_d, p_{d-1}, p_{d-2} can be found by the following equalities:

$$p_0 = \sqrt[4]{\alpha^{2d} + \alpha^{2d-1} p_{d-1}}, p_d = 1, p_{d-1} = \alpha + \beta, \tag{20}$$

$$p_{d-2} = \begin{cases} p_{d-1}^4 + \alpha p_{d-1} + \alpha^2, & \text{if } d \text{ is even,} \\ p_{d-1}^4 + \alpha p_{d-1}, & \text{if } d \text{ is odd.} \end{cases}$$

For p_1, p_2,..., p_{d-3}, we can use the following three equations:

$$\sqrt[4]{\alpha} \sum_{i=0}^{k} p_i^2 p_{d-k+i}^2 \alpha^{2i} = \sqrt[4]{\beta} \sqrt{\alpha}^{d+2k} \sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} p_{k-2i} B(d-k+2i, i), \tag{21}$$

for $k = 0, 1, \ldots, d$,

$$p_k^4 = \alpha^{2d-4k-1} \sum_{i=0}^{k} p_{d-2k-1+2i} B(d-2k-1+2i, i) \alpha^{2i} + \alpha^{2d-4k} \sum_{i=0}^{k} p_{d-2k+2i} B(d-2k+2i, i) \alpha^{2i}, \tag{22}$$

for $k = 0, 1, \ldots, \lfloor (d-1)/2 \rfloor$. And

$$p_{d-k-1}^4 = \alpha \sum_{i=0}^{k} p_{d-2k-1+2i} B(d-2k-1+2i, i) \alpha^{2i} + \sum_{i=0}^{k+1} p_{d-2-2k+2i} B(d-2-2k+2i, i) \alpha^{2i}, \tag{23}$$

for $k = 0, 1, \ldots, \lfloor d/2 \rfloor - 1$, where

$$B(i, j) = \frac{i!}{j!(i-j)!} \pmod 2$$

for nonnegative integers i, j.

We rewrite the three equations by

$$p_k^2 + b_k p_k + c_k = 0, \tag{24}$$

where

$$b_k = \frac{\sqrt[4]{\beta}\sqrt{\alpha}^{d+2k}}{\alpha^{2k}\sqrt[4]{\alpha}} \tag{25}$$

and

$$c_k = \frac{\sqrt[4]{\alpha}\sum_{i=0}^{k-1} p_i^2 p_{d-k+i}^2 \alpha^{2i} + \sqrt[4]{\beta}\sqrt{\alpha}^{d+2k}\sum_{i=1}^{\lfloor k/2\rfloor} p_{k-2i}B(d-k+2i,i)}{\alpha^{2k}\sqrt[4]{\alpha}}, \tag{26}$$

$$p_{d-2k-1} = \frac{p_k^4}{\alpha^{2d-4k-1}} + \sum_{i=1}^{k} p_{d-2k-1+2i}B(d-2k-1+2i,i)\alpha^{2i} + \tag{27}$$

$$\alpha\sum_{i=0}^{k} p_{d-2k+2i}B(d-2k+2i,i)\alpha^{2i},$$

and

$$p_{d-2k-2} = p_{d-k-1}^4 + \sum_{i=1}^{k+1} p_{d-2k-2+2i}B(d-2k-2+2i,i)\alpha^{2i} + \tag{28}$$

$$\alpha\sum_{i=0}^{k} p_{d-2k-1+2i}B(d-2k-1+2i,i)\alpha^{2i}.$$

To find $p_1, p_2, \ldots, p_{d-3}$, first we set $k = 1$ in Equation 24, which gets

$$p_1^2 + b_1 p_1 + c_1 = 0.$$

So

$$p_1 = \gamma_1 + \pi_0 b_1,$$

where γ_1 is a root of $x^2 + b_1 x + c_1 = 0$ and $\pi_0 = 0$ or 1. Then we set $k = 1$ in Equation 27 and Equation 28, we can see that p_{d-3} and p_{d-4} can be represented as a polynomial in $\mathbb{F}_q[\pi_0]$.

Similarly, set $k = 2$ in Equation 24, which gets

$$p_2^2 + b_2 p_2 + c_2 = 0.$$

This time we get

$$p_2 = \gamma_2 + \pi_1 b_2,$$

where γ_2 is a root of $x^2 + b_2x + c_2 = 0$ and $\pi_1 = 0$ or 1. Note that p_2 is in $\mathbb{F}_q[\pi_0, \pi_1]$. And let $k =$ 2in Equation 27, Equation 28 in turn, p_{d-5}, p_{d-6} can also be represented as polynomials in $\mathbb{F}_q[\pi_0, \pi_1]$.

Continue this process until all p_1, p_2,. . . ,p_{d-3} are represented as polynomails in $\mathbb{F}_q[\pi_0, \pi_1, \ldots, \pi_K]$ ($K = \lceil (d-3)/3 \rceil$), then we substitute the representations of p_1, p_2,. . . ,p_{d-3} to the equations in Equation 27 and Equation 28 which are not used yet. This will give us enough equations to solve $\pi_0, \pi_1, \ldots, \pi_K$. After these unknowns are solved, p_k's are found, and so F_l is obtained.

We summarize this algorithm as follows:

Algorithm to compute isogenies

Input: $E_{b_1} : y^2 + xy = x^3 + b_1$, an Elkies prime l

Output: $F_l(x)$

1. Let $j_1 = 1/b_1$
2. Find a root of $\Phi_l(x, j_1) = 0$ in $\mathbb{F}_q$, say j_2
3. Set $b_2 = 1/j_2$
4. Set $\alpha = \sqrt[4]{b_1}$, $\beta = \sqrt[4]{b_2}$
5. Set $d = (l-1)/2$
6. Compute $p_0, p_d, p_{d-1}, p_d - 2$ by Equation 20
7. Represent $p_1, p_2, \ldots, p_{d-3}$ by $\pi_0, \pi_1, \ldots, \pi_K$
8. Put these representations into unused equations in Equation 27 and 28
9. Solve $\pi_0, \pi_1, \ldots, \pi_K$ and obtain $p_1, p_2, \ldots, p_{d-3}$
10. Compute q_i, $i = 0, 1, \ldots, d$ by Equation 19
11. Return $\sum_{i=0}^{d} q_i^2 x^i$

---

5 Experimental Results

We implemented the SEA algorithm by two steps. The first one decides which kind of prime a prime l is, then computes F_l (if l is Elkies) and stores it, we call this step *Fl*. The other one combines data got before and recovers t, we call this step *sea2* (to distinguish from SEA program with respect to prime fields).

We computed the orders of ten binary elliptic curves (five of them are Koblitz curves) recommended by NIST (National Institute of Standards and Technology) with our implementation of SEA2 and an open source of Schoof2 [7] respectively. In both SEA2 and Schoof2 we used the MIRACL(Multiprecision Integer and Rational Arithmetic C/C++ Library [7]) library. It provides big number arithmetic, finite field arithmetic, and can do computations in polynomial rings, power series ring, etc. The comparisons of performance results are listed as the table below.

Table: Comparison of two programs

Curve	Schoof2	SEA2		
		Fl	sea2	Total
B-163	42s	2s	2s	4s
B-233	9m 26s	6s	5s	11s
B-283	34m 39s	20s	24s	44s
B-409	6h 45m 48s	1m 30s	1m 11s	2m 41s
B-571	71h 54m 25s	33m 43s	7m 13s	40m 56s
K-163	17s	1s	2s	3s
K-233	1m 40s	2s	4s	6s
K-283	5m 34s	3s	16s	19s
K-409	1h 4m 18s	20s	17s	37s
K-571	10h 29m 25s	1m 9s	46s	1m 55s

Table: Computing Environments

Hostname	CPU	Cores	64-bit	OS	Memory(MB)	L2 Cache (KB)
linux7	Xeon L5420 2.5G	8	v	Linux	16384	12288

6 Conclusion

We have implemented SEA algorithm in the case of binary fields. With our program we can compute the orders of elliptic curves over binary fields reasonably fast. Moreover, since for each prime l, the procedure to find t_l is independent from each other, this program can be easily parallelized if we want to find a secure curve in a shorter time.

References

[1] I.F. Blake, G. Seroussi and N.P. Smart. *Elliptic curve in cryptography.* volume 265 of London Mathematical Society Lecture Note Series.Cambridge University Press, Cambridge, 2000.

[2] R. Lercier. *Computing isogenies in F_{2^n}.* ANTS-II: Algorithmic Number Theory, Lecture Notes Computation Science, vol. 1122, Springer-Verlag, 1996, p. 197-212

[3] A. J. Menezes. *Elliptic curve public key cryptosystems.* Springer, 1993

[4] F. A. Meneze, S. Vanstone and T. Okamoto. *Reducing elliptic curve logarithms to logarithms in a finite field.* IEEE Transactions on Information Theory, 39:1639-1646, 1993

[5] V. Müller. *Ein Algorithmus zur Bestimmung der Punktzahl elliptischer Kurven über endlichen Körpern der Charackteristik grösser drei,* Ph.D. Thesis, Universitaet des Saarlandes, 1995

[6] R. Schoof. *Counting points on elliptic curves over finite fields.* Journal de théorie des nombres de Bordeaux,7 no. 7 (1995), p.219-254

[7] Shamus Software. *Multiprecision Integer and Rational Arithmetic C/C++ Library.* http://www.shamus.ie/

[8] F. Vercauteren. *The SEA algorithm in characteristic 2.* Preprint, 2000.

A Source Code with Explanation

In the appendix we will give the part of our source code computing isogenies. The words with talic font are our explanation about every function.

A.1 Some Preprocessors

```
int M, d_Fl, num_pi, count[200], term[10000][1000];
GF2m alpha, alphaq, alphad, beta, betaq, delta, alpha2[120], eq[30][10000];
vector<GF2m> p[200], p2[200];
bool pi[100], pK[200], B[350][350], BC[12000][10000];

int main (const int argc, char * argv[]) {
    int L, a, b, c;
    ofstream ofile;
    FILE *myfptr;
    miracl *mip=&precision;
    Big A, b1;
    mip->IOBASE=10;
```

usage:
M: size of the base field.
a, b, c: the degrees of the middle three terms of the irreducible polynomial f(x)
A, B: parameters of the curve

```
    if (argc != 7 && argc != 5) {
        cout << "Usage: " << argv[0] << " M a b c A B" << endl;
        cout << "Usage: " << argv[0] << " M a A B" << endl;
        return -1;
    }
    else if (argc == 5) {
        M = atoi(argv[1]);
        a = atoi(argv[2]);
        b = 0;
        c = 0;
        A = argv[3];
```

```
        b1 = argv[4];
}
else {
        M = atoi(argv[1]);
        a = atoi(argv[2]);
        b = atoi(argv[3]);
        c = atoi(argv[4]);
        A = argv[5];
        b1 = argv[6];
}
ofile.open("fl_output.txt");
ecurve2(M,a,b,c,A,b1,TRUE,MR_AFFINE);
ofile << M << " " << a << " " << b << " " << c << " " << A << " " << b1 << endl;

cout << "M= " << M << ", B= " << b1 << endl;
get_bij();

if (M>= 56) {    // when M < 56, we use Pollard's meththod immediately
        Poly2 MP, Fl;
      GF2m j2,b2;
        int n;
        Big accum= 2, d= howmanyprimes(M);    //    accum is the product of used
                                                    primes
        ofile << d << endl;

        // When M is even, we need this delta to solve quadratic equations
        delta= 2;
        if (M%2==0) {
                while (trace(delta)==0) delta*=2;
        }

        // For Koblitz curves we will use a different method
        bool Koblitz= false;
        if (b1==1) Koblitz= true;

        // get modular pilynomials from "modpol2.txt"
        myfptr = fopen("modpol2.txt", "r");
        while((fscanf(myfptr, " %d %d", &L, &n)!=EOF) && (accum<= d)){
```

```
        cout << "L= " << setw(3) << L << ", ";
            MP= sub_j_in_modpol(b1,n,myfptr);
            if (Koblitz) {              // Koblitz case
                if (n%2==1) {    // Atkin case
                    cout << "                atkin!!" << endl;
                    Atkin();
                    continue;
                }
                else Fl= get_FlK(L,b1,1);    // Elkies case
            }
            else {    // the case B ≠ 1
                j2= Atkin_or_Elkies(MP);
                if (j2==0) {    // Atkin case
                    cout << "                atkin!!" << endl;
                    Atkin();
                    continue;
                }
                /* Elkies */
                b2= 1/j2;
                Fl= get_Fl(L,b1,b2);
            }
            accum*= L;
            cout << "elkies!!" << endl;
            /* output of FL*/
            ofile << L << endl;
            term2 *tempptr=Fl.start;
            int mytemp = tempptr->n + 1;
            while (tempptr!=NULL) {
                mytemp--;
                if (tempptr->n == mytemp) {
                    ofile << tempptr->an << endl;
                    tempptr=tempptr->next;
                }
                else ofile << "0" << endl;
            }
            ofile << endl;
        }
}
```

```
        fclose(myfptr);
        ofile.close();
        return 0;
}
```

We use index[10000] *to store the terms of the monomials*
For example,
(i) *index*[0].*deg* = 2; *index*[0].*mono*[0]= 1; *index*[0].*mono*[1]= 3;
then index[0] = $\pi_1\pi_3$
(ii) index[1].*deg* = 4; i*ndex*[1].*mono*[0]= 1; *index*[1].*mono*[1]= 3;
index[1].*mono*[2]= 4; *index*[1].*mono*[3]= 5;
then index[1] = $\pi_1\pi_3\pi_4\pi_5$

```
struct INDEX {
        int deg;
        int mono[10];
} index[10000];
```

sol_qua: solve quadratic equations when M is odd
sol_qua2: solve quadratic equations when M is even

```
GF2m sol_qua(GF2m b, GF2m c) {
       int i;
       GF2m x= c/(b*b),sol=0;
        for (i=0; i<= (M-1)/2; i++) {
                sol+= x;
                x= pow(x,4);
        }
        sol*= b;
        return sol;
}

GF2m sol_qua2(GF2m b, GF2m c) {
        int i;
        GF2m x= c/(b*b),sol=0;
```

```
        GF2m s=0, t=delta;
        x= sqrt(x);
        for (i= 0; i< M-1; i++) {
                x= sqrt(x);
                t= sqrt(t);
                s+= t;
                sol+= s*x;
        }
        sol*= b;
        return sol;
}
```

get_index: see how many monomials will be used, and give each of them an index. When we compute p_k, the monomials being used are
$\{\pi_i\pi_j...\pi_m\pi_n \mid 2i+2j+...+2m+n+(3d-2) = k\ ,\ d = deg(\pi_i\pi_j...\pi_m\pi_n)\}$

```
void get_index(int k) {
        int i,j,counta;
        int d=2,temp[10];

        counta= count[k-1]-1;
        while (d*(d+1)-1< k+1) {
                for (i=0; i< d-1; i++) temp[i]= i;
                temp[d-1]= k-3*d+2;
                for (i=0; i< d-1; i++) temp[d-1]-= 2*temp[i];
                while (temp[d-2]< temp[d-1]) {
                        counta++;
                        index[counta].deg= d;
                        for (i=0; i< d; i++) index[counta].mono[i]= temp[i];
                        temp[d-1]-= 2;
                        temp[d-2]++;
                        for (i=1; i< d; i++) {
                                if (temp[d-2]>= temp[d-1]) {
                                        temp[d-1-i]++;
                                        for (j=d-i; j< d-1; j++) temp[j]= temp[j-1]+1;
                                        temp[d-1]= k-3*d+2;
                                        for (j=0; j< d-1; j++) temp[d-1]-= 2*temp[j];
                                }
```

```
                }
            }
            d++;
        }
        counta++;
        index[counta].deg= 1;
        index[counta].mono[0]= k-1;
        count[k]= counta+1;
        return;
}
```

get_mono:
input: two monomials
return value: product of the two monomial

```
INDEX get_mono(INDEX ind1,INDEX ind2) {
        INDEX temp;
        int i,j,k;

        for (i=0; i< ind1.deg; i++) temp.mono[i]= ind1.mono[i];
        temp.deg= ind1.deg;
        for (i=0; i< ind2.deg; i++) {
            for (j=0; j< temp.deg; j++) {
                if (ind2.mono[i]==temp.mono[j]) break;
                if (ind2.mono[i]< temp.mono[j]) {
                    for (k= temp.deg; k>j; k--) temp.mono[k]= temp.mono[k-1];
                    temp.mono[j]= ind2.mono[i];
                    temp.deg++;
                    break;
                }
            }
            if (j==temp.deg) {
                temp.mono[j]= ind2.mono[i];
                temp.deg++;
            }
        }
        return temp;
}
```

get_term:
input: monomial
return value: the index refered in get_index of the monomial

```
int get_term(INDEX ind) {
        int i,j,k=0,d=2,temp[10],counta;
        bool match;
        if (ind.deg==0) return 0;
        else if (ind.deg==1) return (count[ind.mono[0]+1]-1);
        else {
                for (i=0; i< ind.deg-1; i++) k+= 2*ind.mono[i];
                k+= ind.mono[ind.deg-1]+3*ind.deg-2;

                counta= count[k-1]-1;
                while (d*(d+1)-1< k+1) {
                        for (i=0; i< d-1; i++) temp[i]= i;
                        temp[d-1]= k-3*d+2;
                        for (i=0; i< d-1; i++) temp[d-1]-= 2*temp[i];
                        while (temp[d-2]< temp[d-1]) {
                                counta++;
                                match= true;
                                for (i=0; i< ind.deg; i++) if (ind.mono[i]!=temp[i]) match= false;
                                if (match) break;
                                temp[d-1]-= 2;
                                temp[d-2]++;
                                for (i=1; i< d; i++) {
                                        if (temp[d-2]>= temp[d-1]) {
                                                temp[d-1-i]++;
                                                for (j=d-i; j< d-1; j++) temp[j]= temp[j-1]+1;
                                                temp[d-1]= k-3*d+2;
                                                for (j=0; j< d-1; j++) temp[d-1]-= 2*temp[j];
                                        }
                                }
                        }
                        if (match) break;
                        d++;
                }
```

```
            return counta;
        }
}
```

sub_j_in_modpol: read modular polynomials from "modpol2.txt", and substitute j-invariant in one of the variables

```
Poly2 sub_j_in_modpol(GF2m b1, int n,FILE* myfptr) {
        int i;
        GF2m j1;
        Poly2 MP;
        j1= ((GF2m)1)/(GF2m)b1;
        MP.clear();
        for(i = 0; i < n; i++){
            int degx, degy;
            fscanf(myfptr, " %d %d", &degx, &degy);
            MP.addterm(pow(j1, degx), degy);
        }
        return MP;
}
```

Atkin_or_Elkies: See a prime L is Elkies or Atkin

```
GF2m Atkin_or_Elkies(Poly2 MP) {
        int i;
        GF2m b,c,j2;
        Poly2Mod G, XQ, XX;

        setmod(MP);
        XX= 0;
        XX.addterm((GF2m)1,1);
        XQ= XX;
        for (i=0; i< M; i++) XQ*= XQ;

        G= gcd(XQ+XX);
        b= G.coeff(1);
        c= G.coeff(0);

        cout << "degree(G) = " << degree(G) << ", ";
```

```
        if (degree(G)==1) j2= c;
        else if (degree(G)==2) {
                if (M%2==1)       j2= sol_qua(b,c);
                else j2= sol_qua2(b,c);
        }
        else j2= 0;

        return j2;
}
```

howmanyprimes: decide how many primes we need

```
Big howmanyprimes(int M) {
        Big p= pow((Big)2,M);
        Big d;
       if (M<=256) d=pow((Big)2,64);
        else d=pow((Big)2,72);
       d=sqrt(p/d);
       if (d<256) d=256;
        return d;
}
```

get_bij : compute B(i , j) = i ! / (j! (i–j)!) mod 2 *for later use*

```
void get_bij(void) {
        int i,j;
        for (i=0; i< M/2; i++) B[i][0]= B[i][i]= true;      //    B(i,j)= i!/(j!(i-j)!) mod2
        for (i=2; i< M/2; i++) {
                for (j=1; j< i; j++) {
                        B[i][j]= B[i-1][j-1]^B[i-1][j];
                }
        }
        return;
}
```

column_exch: exchange two columns in a coefficient matrix

```
void column_exch(int i,int j,int n) {
        int k;
        GF2m temp;
        for (k=0; k< n; k++) {
                temp= eq[k][i];
                eq[k][i]= eq[k][j];
                eq[k][j]= temp;
        }
        return;
}
```

row_op: add one row to another in a coefficient matrix

```
void row_op(bool row1[], bool row2[], int c) {
        int i;
        for (i=c; i>=0; i--)    row2[i]^= row1[i];
        return;
}
```

row_exch: exchange two rows in a coefficient matrix

```
void row_exch(bool row1[], bool row2[], int K) {
        int i;
        bool c;
        for (i=0; i< K+1; i++) {
                c= row1[i];
                row1[i]= row2[i];
                row2[i]= c;
        }
        return;
}
```

A.2 Computing Isogenies

get_pk: represent p_k as a polynomial in $F_q[\pi_0, \pi_1, ..., \pi_K]$ by Equation 19, 22, 23.

```
void get_pk(void) {
        int i,j,k,m;
        GF2m bk;

        index[0].deg= 0;      // the constant term
        count[0]= 1;
        // num_pi is the number of the unknowns, here we give indices to monomials
        for (i=1; i< num_pi+1; i++) get_index(i);
        for (i=0; i< num_pi; i++) {
              k= count[(num_pi-i-1)/2];
             for (j=0; j< count[i]; j++) {
                    for (m=0; m< k; m++) {
                          INDEX u;
                          u= get_mono(index[j],index[m]);
                          term[j][m]= get_term(u);
                    }
              }
        }

        alpha2[0]= 1;      // α^2i will be used a lot later, we compute and store then here
        alpha2[1]= pow(alpha,2);
        j= d_Fl/2+1;
        for (i=2; i< j; i++) alpha2[i]= alpha2[i-1]*alpha2[1];
        bk= betaq*sqrt(alphad)/alphaq;            //    bk

        for (k=1; k< num_pi; k++) {
              bk/= alpha;                  //    bk= betaq*sqrt(alphad)/(pow(alpha,k)*alphaq)
              vector<GF2m> ck(count[k]);
              for (j=0; j< count[k]; j++) ck[j]= 0;    //    Compute c_k by Equation 21
              int s= k;
              GF2m temp= bk*alpha2[k];
              for(i=1; i<= k/2; i++) {
                    s-= 2;
                    if (B[d_Fl-s][i]) for (j=0; j< count[s]; j++) ck[j]+= p[s][j]*temp;
```

```
}

for (i=0; i< k; i++) {
        s= d_Fl-k+i;
        for (j=0; j< count[i]; j++) {
                for (m=0; m< count[s]; m++) {
                        ck[term[j][m]]+= p2[i][j]*p2[s][m];
                }
        }
}

int t1= d_Fl-2*k-1;
int t2= t1-1;
p[k]= p2[k]= p[t1]= p[t2]= ck;          //    set vector size
count[t1]= count[t2]= count[k];

p[k][--count[k]]= bk;
p2[k][count[k]]= pow(bk,2)*alpha2[k];
temp= pow(alpha2[k],2)*alpha/(alphad*alphad);
if (M%2==1) {
        for (j=0; j< count[k]; j++) {
                ck[j]/= alpha2[k];
                //    By Equation 19, pk is a root of x^2+b_k x+c_k=0.
                p[k][j]= sol_qua(bk,ck[j]);
                p2[k][j]= pow(p[k][j],2)*alpha2[k];
                p[t1][j]= pow(p[k][j],4)*temp;        //    p[d-2k-1]
                p[t2][j]= 0;                                    //    p[d-2k-2]
        }
}
else {
        for (j=0; j< count[k]; j++) {
                ck[j]/= alpha2[k];
                p[k][j]= sol_qua2(bk,ck[j]);
                p2[k][j]= pow(p[k][j],2)*alpha2[k];
                p[t1][j]= pow(p[k][j],4)*temp;        //    p[d-2k-1]
                p[t2][j]= 0;                                    //    p[d-2k-2]
        }
}
```

```
        //    Compute p[d_Fl-2*k-1], p[d_Fl-2*k-2] by Equation 22, 23
        p[t1][count[k]]= pow(bk,4)*temp;
        p[t2][count[k]++]= 0;

        s= t1;
        for(i=1; i< k+1; i++) {
            int qq= count[k-i];
            if (B[++s][i-1]) {
                temp= alpha*alpha2[i-1];
                for (j=0; j< qq; j++) p[t1][j]+= p[s][j]*temp;
            }
            if (B[s][i]) for (j=0; j< qq; j++) p[t2][j]+= p[s][j]*alpha2[i];
            if (B[++s][i]) {
                temp= alpha*alpha2[i];
                for (j=0; j< qq; j++) {
                    p[t1][j]+= p[s][j]*alpha2[i];
                    p[t2][j]+= p[s][j]*temp;
                }
            }
        }
        if (B[d_Fl][k]) p[t1][0]+= alpha2[k]*alpha;
        if (B[d_Fl][k+1]) p[t2][0]+= p[d_Fl][0]*alpha2[k+1];
        i= d_Fl-k-1; m= k/2;
        for (j=0; j< count[m]; j++) p[t2][j]+= pow(p[i][j],4)+ alpha*p[t1][j];
        for (j=count[m]; j< count[k]; j++) p[t2][j]+= alpha*p[t1][j];
        if (k< num_pi/2+1) {
            p2[t1]= p2[t2]= ck;          //    set vector size
            for (j=0; j< count[k]; j++) {
                p2[t1][j]= pow(p[t1][j],2);
                p2[t2][j]= pow(p[t2][j],2);
            }
        }

    }

for (k=num_pi; k< num_pi+1; k++) {
        bk/= alpha;            //    bk= betaq*sqrt(alphad)/(pow(alpha,k)*alphaq)
        vector<GF2m> ck(count[k]);
```

```
for (j=0; j< count[k]; j++) ck[j]= 0;
int s= k;
GF2m temp= bk*alpha2[k];
for(i=1; i<= k/2; i++) {
        s-= 2;
        if (B[d_Fl-s][i]) for (j=0; j< count[s]; j++) ck[j]+= p[s][j]*temp;
}

for (i=0; i< k; i++) {
        s= d_Fl-k+i;
        for (j=0; j< count[i]; j++) {
                for (m=0; m< count[s]; m++) {
                        ck[term[j][m]]+= p2[i][j]*p2[s][m];
                }
        }
}

p[k]= ck;      //    set vector size
count[k]--;
if (M%2==1) {
        for (j=0; j< count[k]; j++) {
                ck[j]/= alpha2[k];
                p[k][j]= sol_qua(bk,ck[j]);
        }
}
else {
        for (j=0; j< count[k]; j++) {
                ck[j]/= alpha2[k];
                p[k][j]= sol_qua2(bk,ck[j]);
        }
}
p[k][count[k]++]= bk;
if (3*k==(d_Fl-1)) break;

int t= d_Fl-2*k-1;
p[t]= ck;
count[t]= count[k];
```

```
            temp= pow(alpha2[k],2)*alpha/(alphad*alphad);
            for (j=0; j< count[k]; j++) p[t][j]= pow(p[k][j],4)*temp;      //    p[d-2k-1]
            s= t;
            for(i=1; i< k+1; i++) {
                  int qq= count[k-i];
                  if (B[++s][i-1]) {
                        temp= alpha*alpha2[i-1];
                        for (j=0; j< qq; j++) p[t][j]+= p[s][j]*temp;
                  }
                  if (B[++s][i]) for (j=0; j< qq; j++) p[t][j]+= p[s][j]*alpha2[i];
            }
            if (B[d_Fl][k]) p[t][0]+= alpha2[k]*alpha;
            if (3*k==(d_Fl-2)) break;

            p[--t]= ck;
            count[t]= count[k];
            i= d_Fl-k-1; m= k/2;
            s= t+1;
            for (j=0; j< count[m]; j++) p[t][j]= pow(p[i][j],4) + alpha*p[s][j];
            for (j= count[m]; j< count[k]; j++) p[t][j]= alpha*p[s][j];      //      p[d-2k-2]
            for(i=1; i< k+1; i++)    {
                  int qq= count[k-i];
                  if (B[++s][i]) for (j=0; j< qq; j++) p[t][j]+= p[s][j]*alpha2[i];
                  if (B[++s][i]) {
                        temp= alpha*alpha2[i];
                        for (j=0; j< qq; j++) p[t][j]+= p[s][j]*temp;
                  }
            }
            if (B[d_Fl][k+1]) p[t][0]+= alpha2[k+1];
      }
      return;
}
```

get_cf: After representing p_k's as polynomials in $F_q[\pi_0, \pi_1, \ldots, \pi_K]$, substitute them into equations which are unused in Equation 22.

```
void get_cf(int n) {
```

```
	int i,j,k,t;
	GF2m temp,temp2= alphad*alphad/(alpha*pow(alpha,4*num_pi));
	for (t=0; t< n; t++) {
		for (j=0; j< count[num_pi]; j++) eq[t][j]= 0;
		k= num_pi+1+t;
		int s= d_Fl-2*k-1;
		for (j=0; j< count[s]; j++) eq[t][j]+= p[s][j];
		for (i=1; i< k+1; i++) {
			if (B[++s][i-1]) {
				temp= alpha2[i-1]*alpha;
				for (j=0; j< count[s]; j++) eq[t][j]+= p[s][j]*temp;
			}
			if (B[++s][i]) for (j=0; j< count[s]; j++) eq[t][j]+= p[s][j]*alpha2[i];
		}
		if (B[d_Fl][k]) eq[t][0]+= alpha2[k]*alpha;
		temp2/= alpha2[2];
		for (j=0; j< count[num_pi]; j++) eq[t][j]*= temp2;
		for (j=0; j< count[k]; j++) eq[t][j]+= pow(p[k][j],4);
	}
	return;
}
```

solve_pi: solve the equations obtained before by Gaussian elimination

```
void solve_pi (int noe, int K) {
	int i,j,k,r;

	for (k=0; k< noe; k++)	{
		for (i=0; i< K; i++) {
			for (j=0; j< M; j++) {
				if (eq[k][i]%2 == 1) BC[k*M+j][i]= true;
				else BC[k*M+j][i]= false;
				eq[k][i]/= 2;
			}
		}
	}
	r=0;		//	rank
	for (i= --K; i> 0; i--) {
```

```
            j= r;
            while ((!BC[j][i]) && (j< M*noe-1))    j++;
            row_exch(BC[r],BC[j],K);
            if (!BC[r][i]) continue;
            for (j=r+1; j< M*noe; j++) if (BC[j][i]) row_op(BC[r],BC[j],i);
            r++;
        }
        for (i=0; i< num_pi; i++) {
            pi[i]= BC[r-1-i][0];
            if (pi[i]) for (j=0; j< r-1-i; j++) BC[j][0]^= BC[j][i+1];
        }
        return;
}
```

get_Fl: get F_l by using the information obtained before

```
Poly2 get_Fl(int L, GF2m b1, GF2m b2) {
        int i,j,k;
        int noe;
        GF2m q[200];
        Poly2 Q;

        d_Fl= (L-1)/2;                      // degree of F_l
        num_pi= (d_Fl-3)/3 +1;
        if (d_Fl%3==0) num_pi--;        // number of unknowns

        alpha= sqrt(sqrt(b1));      //    alpha, alpha^(1/4), alpha^d
        alphaq= sqrt(sqrt(alpha));
        alphad= pow(alpha,d_Fl);
        beta= sqrt(sqrt(b2));       //    beta, beta^(1/4)
        betaq= sqrt(sqrt(beta));

        vector<GF2m> c(1);                  // Compute p0,pd,pd–1,pd–2 by Equation 15
        count[d_Fl]= count[0]= 1;           //    p[0], p[d], p[d-1], p[d-2]
        p[d_Fl]= p2[d_Fl]= p[0]= p2[0]= c;      //    set vector size
        p[d_Fl][0]= p2[d_Fl][0]= 1;

        p[0][0]= sqrt(alphad)*betaq/alphaq;
```

```
p2[0][0]= pow(p[0][0],2);
if (d_Fl>1) {
        count[d_Fl-1]= 1;
        p[d_Fl-1]= p2[d_Fl-1]= c;
        p[d_Fl-1][0]= alpha+beta;
        p2[d_Fl-1][0]= pow(p[d_Fl-1][0],2);
}
if (d_Fl>2) {
        count[d_Fl-2]= 1;
        p[d_Fl-2]= p2[d_Fl-2]= c;
        p[d_Fl-2][0]= pow(p[d_Fl-1][0],4)+alpha*p[d_Fl-1][0];
        if (d_Fl%2==1) p[d_Fl-2][0]+= alpha*alpha;
        p2[d_Fl-2][0]= pow(p[d_Fl-2][0],2);
}

if (d_Fl< 5) {                  //    if deg(Fl)< 5 we are done
        GF2m temp= alphaq*sqrt(alphad)*alpha/betaq;
        for(k=0; k< d_Fl+1; k++) {
                temp/= alpha;
                q[k]= temp*p[d_Fl-k][0];
                Q.addterm(pow(q[k],2),k);
        }
        return Q;
}

get_pk();
noe= count[num_pi]/(M-2) +2;        //    number of equations needed
get_cf(noe);
for (i=1; i< num_pi+1; i++) column_exch(i,count[i]-1,noe);
solve_pi(noe,count[num_pi]);

int t[10000],s[100],m;
i=0;
for (m=1; m< num_pi+1; m++) {
        for (j= count[m-1]; j< count[m]; j++) {
                bool temp= true;
                for (k=0; k< index[j].deg; k++) {
                        if (!pi[index[j].mono[k]]) {
```

```
                        temp= false;
                        break;
                    }
                }
                if (temp) t[i++]= j;
            }
            s[m]= i;
        }

        if (d_Fl%3==0) num_pi++;
        for (k=1; k< num_pi; k++) {
            for (j=0; j< s[k]; j++) {
                p[k][0]+= p[k][t[j]];
                p[d_Fl-2*k-1][0]+= p[d_Fl-2*k-1][t[j]];
                p[d_Fl-2*k-2][0]+= p[d_Fl-2*k-2][t[j]];
            }
        }

        for (k=0; k< d_Fl%3; k++) {
            for (j=0; j< s[num_pi]; j++) p[num_pi+k][0]+= p[num_pi+k][t[j]];
        }
        Q.clear();      // compute q_k by Equation 14
        GF2m temp= alphaq*sqrt(alphad)*alpha/betaq;
        for(k=0; k< d_Fl+1; k++) {
            temp/= alpha;
            q[k]= temp*p[d_Fl-k][0];
            Q.addterm(pow(q[k],2),k);
        }
        return Q;
}
```

A.3 Computing Isogenies for Koblitz Curves

get_pkK: for Koblitz curves, the values of p_k's are always 0 or 1, and we don't have to compute monomials which have degree greater than 1, so we use another function

```
void get_pkK (void) {
        int i,k;

       for (k=1; k< num_pi + 1; k++) {
              pK[k]= pi[k-1];
              if (3*k==(d_Fl-1)) break;

              pK[d_Fl-2*k-1]= false;                    //    p[d-2k-1]
            for(i=1; i< k+1; i++)    {
                  if (B[d_Fl-2*k-1+2*i][i]) pK[d_Fl-2*k-1]^= pK[d_Fl-2*k-1+2*i];
                  if (B[d_Fl-2*k+2*i][i])     pK[d_Fl-2*k-1]^= pK[d_Fl-2*k+2*i];
              }
             pK[d_Fl-2*k-1]^= (pK[d_Fl-2*k]^pK[k]);
              if (3*k==(d_Fl-2)) break;

              pK[d_Fl-2*k-2]= false;                    //     p[d-2k-2]
            for(i=1; i< k+1; i++)    {
                  if (B[d_Fl-2-2*k+2*i][i])    pK[d_Fl-2*k-2]^= pK[d_Fl-2-2*k+2*i];
                  if (B[d_Fl-1-2*k+2*i][i])    pK[d_Fl-2*k-2]^= pK[d_Fl-1-2*k+2*i];
              }
            pK[d_Fl-2*k-2]^= pK[d_Fl-2*k-1]^pK[d_Fl-k-1]^B[d_Fl][k+1];
       }
       return;
}
```

get_cf1, get_cf2: In Koblitz case, the equations in Equation 22 maybe not enough, we also need equations in Equaton 23.

```
bool get_cf1(int j) {
       int i;
       int k= j+1;
       bool c= false;
       for (i=0;i< k+1;i++) {
```

```
            if(B[d_Fl-2*k-1+2*i][i]) c^= pK[d_Fl-2*k-1+2*i];
            if(B[d_Fl-2*k+2*i][i]) c^= pK[d_Fl-2*k+2*i];
        }
        c^= pK[k];
        return c;
}
bool get_cf2(int k) {
        int i;
        bool c= false;
        for (i=0;i< k+1;i++) {
            if(B[d_Fl-2*k-1+2*i][i]) c^= pK[d_Fl-2*k-1+2*i];
            if(B[d_Fl-2*k-2+2*i][i]) c^= pK[d_Fl-2*k-2+2*i];
        }
        c^= pK[d_Fl-k-1]^B[d_Fl][k+1];
        return c;
}
```

get_FlK: compute F_l for Koblitz curves

```
Poly2 get_FlK(int L, GF2m b1, GF2m b2) {
        int i,j,k;
        int noe,noe1,noe2;

        bool Fl_test= false;
        Poly2 Q;

        d_Fl= (L-1)/2;                  //   degree of Fl
        num_pi= (d_Fl-3)/3 +1;          //   number of unknowns have to be added
        if (d_Fl%3 == 0) num_pi--;

        noe1= (d_Fl-1)/2-num_pi+1;
        noe2= d_Fl/2-num_pi;
        noe= noe1+noe2;

        pK[d_Fl]= true;         //   p[0], p[d], p[d-1], p[d-2]
        pK[0]= true;
        if (d_Fl>1) pK[d_Fl-1]= false;
        if (d_Fl>2) {
```

```
        if (d_Fl%2==1)    pK[d_Fl-2]= true;
        else pK[d_Fl-2]= false;
}

if (d_Fl< 5) {                  //    if deg(Fl)< 5 we are done
        for(k=0; k< d_Fl+1; k++) Q.addterm((GF2m)pK[d_Fl-k],k);
        return Q;
}

for (i=0; i< num_pi; i++)    pi[i]= false;              //      degree 0
get_pkK();
for (j=0; j< noe1; j++) BC[j][0]= get_cf1(num_pi+j);
for (j=0; j< noe2; j++) BC[noe1+j][0]= get_cf2(num_pi+j);



for (i=0; i< num_pi; i++) {                                 //      degree 1 for Koblitz
        for (j=0; j< num_pi; j++)    pi[j]= false;
        pi[i]= true;
        get_pkK();
        for (j=0; j< noe1; j++) BC[j][i+1]= get_cf1(num_pi+j)^BC[j][0];
        for (j=0; j< noe2; j++) {
                BC[noe1+j][i+1]= get_cf2(num_pi+j)^BC[noe1+j][0];
        }
}

int r=0;
solve_piK(noe,num_pi,r);

int vio[20];
i=0;
for (j=0; j< num_pi; j++) {
        if (!BC[j][num_pi-j]) {
                for (k= num_pi-1; k>j; k--) row_exch(BC[k],BC[k-1],num_pi);
                vio[i++]= num_pi-j-1;
        }
}

int upperbound= 1 << (num_pi-r);
```

```
        i=0;
        while (!Fl_test) {
                for(j=0; j< num_pi-r; j++) pi[vio[j]]= (i & (1 << j)) && 1;
                i++;
                bool vio_test;
                for (j=0; j< num_pi; j++) {
                        vio_test= false;
                        for (k=0; k< num_pi-r; k++) if (j==vio[k]) vio_test= true;
                        if (!vio_test) {
                                pi[j]= false;
                                for (k=0; k< j; k++) if (BC[num_pi-1-j][k+1]) pi[j]^= pi[k];
                                pi[j]^= BC[num_pi-1-j][0];
                        }
                }
                get_pkK();
                Q.clear();
                for(k=0; k< d_Fl+1; k++) if (pK[d_Fl-k]) Q.addterm(1,k);
                if (i< upperbound+1) Fl_test= DP_test(Q,b1,L);
                else Fl_test= true;
                if (i==upperbound) break;
        }
        return Q;
}
```

DP_test: division polynomial test.

```
bool DP_test(Poly2 Q,GF2m b1, int L) {
        Poly2 Pf[5], G;
        Poly2Mod P[600];
        Pf[0]=0; Pf[1]=1; Pf[2]=0; Pf[3]=0; Pf[4]=0;
        Pf[2].addterm(1,1);
        Pf[3].addterm(b1,0);
        Pf[3].addterm(1,3);
        Pf[3].addterm(1,4);
        Pf[4].addterm(b1,2);
        Pf[4].addterm(1,6);
        setmod(Q);
```

```
    int m;
    for (m=0; m< 5; m++) P[m]= (Poly2Mod) Pf[m];

    m=3;
    while (2*m < L+3) {
        P[2*m-1]= P[m+1]*P[m-1]*P[m-1]*P[m-1]+ P[m-2]*P[m]*P[m]*P[m];
        P[2*m]= (P[m+2]*P[m-1]*P[m-1]+
            P[m-2]*P[m+1]*P[m+1])*P[m]*inverse(P[2]);
        m++;
    }
    G= gcd(P[L]);
    if (G==Q) return true;
    else return false;
}
```

solve_piK: solve_pi function in Koblitz case

```
void solve_piK (int noe, int K, int& r) {
    int i,j;
    r=0;        //    rank
    for (i= K; i> 0; i--) {
        j= r;
        while ((!BC[j][i]) && (j< noe-1))    j++;
        row_exch(BC[r],BC[j],K);
        if (!BC[r][i]) continue;
        for (j=r+1; j< noe; j++) if (BC[j][i]) row_op(BC[r],BC[j],i);
        r++;
    }
    return;
}
```

B Implementation Data

compiler option: -O2
Thread model: posix
gcc version 4.3.2 (Debian 4.3.2-1)

B.1 NIST Binary Curves

(1) Computing Isogenies

B-163

```
b93030@linux7:~/Work/Schoof/miracl$ time ./fl 163 7 6 3 1
298223623434385133626744665662778500 8148015875581
M= 163, b1= 298223623434385133626744665662778500 8148015875581
d= 796131459065721
L=   3, degree(G) = 0,              atkin!!
L=   5, degree(G) = 0,              atkin!!
L=   7, degree(G) = 0,              atkin!!
L=  11, degree(G) = 2, elkies!!
L=  13, degree(G) = 0,              atkin!!
L=  17, degree(G) = 0,              atkin!!
L=  19, degree(G) = 2, elkies!!
L=  23, degree(G) = 0,              atkin!!
L=  29, degree(G) = 0,              atkin!!
L=  31, degree(G) = 2, elkies!!
L=  37, degree(G) = 2, elkies!!
L=  41, degree(G) = 0,              atkin!!
L=  43, degree(G) = 0,              atkin!!
L=  47, degree(G) = 2, elkies!!
L=  53, degree(G) = 0,              atkin!!
L=  59, degree(G) = 0,              atkin!!
L=  61, degree(G) = 2, elkies!!
L=  67, degree(G) = 2, elkies!!
L=  71, degree(G) = 0,              atkin!!
L=  73, degree(G) = 2, elkies!!
L=  79, degree(G) = 0,              atkin!!
L=  83, degree(G) = 0,              atkin!!
L=  89, degree(G) = 2, elkies!!
L=  97, degree(G) = 0,              atkin!!
L= 101, degree(G) = 2, elkies!!

real    0m2.328s
user    0m2.224s
sys     0m0.104s
```

B-233

```
b93030@linux7:~/Work/Schoof/miracl$ time ./fl 233 74 1
276049798002920418707884550237789852030770725625900396439857014712337
3
M= 233, b1=
276049798002920418707884550237789852030770725625900396439857014712337
3
```

```
d= 27354868640032294882193329
L=      3, degree(G) = 0,                atkin!!
L=      5, degree(G) = 2, elkies!!
L=      7, degree(G) = 1, elkies!!
L=    11, degree(G) = 0,                 atkin!!
L=    13, degree(G) = 0,                 atkin!!
L=    17, degree(G) = 2, elkies!!
L=    19, degree(G) = 2, elkies!!
L=    23, degree(G) = 0,                 atkin!!
L=    29, degree(G) = 0,                 atkin!!
L=    31, degree(G) = 2, elkies!!
L=    37, degree(G) = 0,                 atkin!!
L=    41, degree(G) = 0,                 atkin!!
L=    43, degree(G) = 2, elkies!!
L=    47, degree(G) = 0,                 atkin!!
L=    53, degree(G) = 0,                 atkin!!
L=    59, degree(G) = 2, elkies!!
L=    61, degree(G) = 0,                 atkin!!
L=    67, degree(G) = 2, elkies!!
L=    71, degree(G) = 2, elkies!!
L=    73, degree(G) = 2, elkies!!
L=    79, degree(G) = 2, elkies!!
L=    83, degree(G) = 0,                 atkin!!
L=    89, degree(G) = 0,                 atkin!!
L=    97, degree(G) = 2, elkies!!
L= 101, degree(G) = 0,                 atkin!!
L= 103, degree(G) = 2, elkies!!
L= 107, degree(G) = 2, elkies!!
L= 109, degree(G) = 2, elkies!!
L= 113, degree(G) = 0,                 atkin!!
L= 127, degree(G) = 2, elkies!!

real        0m6.408s
user        0m5.828s
sys         0m0.576s
```

B-283

```
b93030@linux7:~/Work/Schoof/miracl$ time ./fl 283 12 7 5 1
48218135760560723740069977803990811803122700303006012701204503412059
1914644378616963829
M= 283, b1=
48218135760560723740069977803990811803122700303006012701204503412059
1914644378616963829
d= 57367317478181007276781504744917
L=      3, degree(G) = 0,                atkin!!
L=      5, degree(G) = 0,                atkin!!
L=      7, degree(G) = 2, elkies!!
L=    11, degree(G) = 0,                 atkin!!
L=    13, degree(G) = 0,                 atkin!!
L=    17, degree(G) = 0,                 atkin!!
L=    19, degree(G) = 2, elkies!!
L=    23, degree(G) = 2, elkies!!
L=    29, degree(G) = 2, elkies!!
L=    31, degree(G) = 0,                 atkin!!
L=    37, degree(G) = 2, elkies!!
```

```
L=   41, degree(G) = 0,                atkin!!
L=   43, degree(G) = 0,                atkin!!
L=   47, degree(G) = 2, elkies!!
L=   53, degree(G) = 0,                atkin!!
L=   59, degree(G) = 0,                atkin!!
L=   61, degree(G) = 2, elkies!!
L=   67, degree(G) = 2, elkies!!
L=   71, degree(G) = 0,                atkin!!
L=   73, degree(G) = 0,                atkin!!
L=   79, degree(G) = 2, elkies!!
L=   83, degree(G) = 2, elkies!!
L=   89, degree(G) = 0,                atkin!!
L=   97, degree(G) = 2, elkies!!
L= 101, degree(G) = 0,                atkin!!
L= 103, degree(G) = 2, elkies!!
L= 107, degree(G) = 2, elkies!!
L= 109, degree(G) = 2, elkies!!
L= 113, degree(G) = 0,                atkin!!
L= 127, degree(G) = 2, elkies!!
L= 131, degree(G) = 0,                atkin!!
L= 137, degree(G) = 2, elkies!!
L= 139, degree(G) = 0,                atkin!!
L= 149, degree(G) = 2, elkies!!
L= 151, degree(G) = 0,                atkin!!
L= 157, degree(G) = 0,                atkin!!
L= 163, degree(G) = 0,                atkin!!
L= 167, degree(G) = 2, elkies!!

real    0m20.771s
user    0m20.229s
sys     0m0.536s
```

B-409

```
b93030@linux7:~/Work/Schoof/miracl$ time ./fl 409 87 1
868862616340907076728177706403844252645058294790436418244386586141
118704710045649886344108090582071423185712121479358925 75
M= 409, b1=
868862616340907076728177706403844252645058294790436418244386586141
118704710045649886344108090582071423185712121479358925 75
d= 529120111857624937813183735811535109761073714132242
L=    3, degree(G) = 0,                atkin!!
L=    5, degree(G) = 0,                atkin!!
L=    7, degree(G) = 0,                atkin!!
L=   11, degree(G) = 2, elkies!!
L=   13, degree(G) = 2, elkies!!
L=   17, degree(G) = 2, elkies!!
L=   19, degree(G) = 2, elkies!!
L=   23, degree(G) = 2, elkies!!
L=   29, degree(G) = 0,                atkin!!
L=   31, degree(G) = 0,                atkin!!
L=   37, degree(G) = 2, elkies!!
L=   41, degree(G) = 2, elkies!!
L=   43, degree(G) = 2, elkies!!
L=   47, degree(G) = 0,                atkin!!
L=   53, degree(G) = 0,                atkin!!
```

```
L=    59, degree(G) = 2, elkies!!
L=    61, degree(G) = 0,                atkin!!
L=    67, degree(G) = 0,                atkin!!
L=    71, degree(G) = 2, elkies!!
L=    73, degree(G) = 0,                atkin!!
L=    79, degree(G) = 0,                atkin!!
L=    83, degree(G) = 2, elkies!!
L=    89, degree(G) = 2, elkies!!
L=    97, degree(G) = 2, elkies!!
L= 101, degree(G) = 2, elkies!!
L= 103, degree(G) = 0,                atkin!!
L= 107, degree(G) = 2, elkies!!
L= 109, degree(G) = 0,                atkin!!
L= 113, degree(G) = 0,                atkin!!
L= 127, degree(G) = 0,                atkin!!
L= 131, degree(G) = 2, elkies!!
L= 137, degree(G) = 2, elkies!!
L= 139, degree(G) = 0,                atkin!!
L= 149, degree(G) = 0,                atkin!!
L= 151, degree(G) = 2, elkies!!
L= 157, degree(G) = 2, elkies!!
L= 163, degree(G) = 0,                atkin!!
L= 167, degree(G) = 2, elkies!!
L= 173, degree(G) = 0,                atkin!!
L= 179, degree(G) = 2, elkies!!
L= 181, degree(G) = 2, elkies!!
L= 191, degree(G) = 2, elkies!!
L= 193, degree(G) = 2, elkies!!
L= 197, degree(G) = 2, elkies!!
L= 199, degree(G) = 0,                atkin!!
L= 211, degree(G) = 2, elkies!!
L= 223, degree(G) = 2, elkies!!

real    1m31.203s
user    1m30.522s
sys     0m0.564s
```

B-571

```
b93030@linux7:~/Work/Schoof/miracl$ time ./fl 571 10 5 2 1
285332924526134353556008696418155129688929877610683298089156085094
418001170112330790532601964265265353300348275302366901684288410817
251487094414061111367922534741972021721 0
M= 571, b1=
285332924526134353556008696418155129688929877610683298089156085094
418001170112330790532601964265265353300348275302366901684288410817
251487094414061111367922534741972021721 0
d=
127933392980412699402423752410905669230480363236317557909096646251
1824293476
L=     3, degree(G) = 0,                atkin!!
L=     5, degree(G) = 0,                atkin!!
L=     7, degree(G) = 2, elkies!!
L=    11, degree(G) = 2, elkies!!
L=    13, degree(G) = 2, elkies!!
L=    17, degree(G) = 0,                atkin!!
```

L= 19, degree(G) = 2, elkies!!
L= 23, degree(G) = 2, elkies!!
L= 29, degree(G) = 0, atkin!!
L= 31, degree(G) = 0, atkin!!
L= 37, degree(G) = 0, atkin!!
L= 41, degree(G) = 2, elkies!!
L= 43, degree(G) = 2, elkies!!
L= 47, degree(G) = 0, atkin!!
L= 53, degree(G) = 2, elkies!!
L= 59, degree(G) = 0, atkin!!
L= 61, degree(G) = 0, atkin!!
L= 67, degree(G) = 0, atkin!!
L= 71, degree(G) = 0, atkin!!
L= 73, degree(G) = 0, atkin!!
L= 79, degree(G) = 2, elkies!!
L= 83, degree(G) = 0, atkin!!
L= 89, degree(G) = 2, elkies!!
L= 97, degree(G) = 0, atkin!!
L= 101, degree(G) = 0, atkin!!
L= 103, degree(G) = 0, atkin!!
L= 107, degree(G) = 0, atkin!!
L= 109, degree(G) = 0, atkin!!
L= 113, degree(G) = 0, atkin!!
L= 127, degree(G) = 2, elkies!!
L= 131, degree(G) = 0, atkin!!
L= 137, degree(G) = 1, elkies!!
L= 139, degree(G) = 2, elkies!!
L= 149, degree(G) = 2, elkies!!
L= 151, degree(G) = 0, atkin!!
L= 157, degree(G) = 2, elkies!!
L= 163, degree(G) = 0, atkin!!
L= 167, degree(G) = 2, elkies!!
L= 173, degree(G) = 2, elkies!!
L= 179, degree(G) = 2, elkies!!
L= 181, degree(G) = 0, atkin!!
L= 191, degree(G) = 2, elkies!!
L= 193, degree(G) = 2, elkies!!
L= 197, degree(G) = 2, elkies!!
L= 199, degree(G) = 2, elkies!!
L= 211, degree(G) = 0, atkin!!
L= 223, degree(G) = 0, atkin!!
L= 227, degree(G) = 0, atkin!!
L= 229, degree(G) = 0, atkin!!
L= 233, degree(G) = 0, atkin!!
L= 239, degree(G) = 2, elkies!!
L= 241, degree(G) = 2, elkies!!
L= 251, degree(G) = 0, atkin!!
L= 257, degree(G) = 2, elkies!!
L= 263, degree(G) = 2, elkies!!
L= 269, degree(G) = 2, elkies!!
L= 271, degree(G) = 0, atkin!!
L= 277, degree(G) = 2, elkies!!
L= 281, degree(G) = 2, elkies!!
L= 283, degree(G) = 0, atkin!!
L= 293, degree(G) = 0, atkin!!
L= 307, degree(G) = 0, atkin!!
L= 311, degree(G) = 2, elkies!!

```
L= 313, degree(G) = 2, elkies!!
L= 317, degree(G) = 2, elkies!!
L= 331, degree(G) = 2, elkies!!
L= 337, degree(G) = 0,                  atkin!!
L= 347, degree(G) = 2, elkies!!
L= 349, degree(G) = 0,                  atkin!!
L= 353, degree(G) = 0,                  atkin!!
L= 359, degree(G) = 0,                  atkin!!
L= 367, degree(G) = 0,                  atkin!!
L= 373, degree(G) = 0,                  atkin!!
L= 379, degree(G) = 2, elkies!!
L= 383, degree(G) = 0,                  atkin!!
L= 389, degree(G) = 0,                  atkin!!
L= 397, degree(G) = 2, elkies!!

real        34m57.203s
user        33m43.374s
sys         0m1.172s
```

(2) SEA Algorithm Part

B-163

```
b93030@linux7:~/Work/Schoof/miracl/sea2lab$ time ./sea2
EP[0]= 2
t[0]= 1
NP mod 11 =       3
NP mod 19 =      13
NP mod 31 =      18
NP mod 37 =      18
NP mod 47 =      18
NP mod 61 =      37
NP mod 67 =      36
NP mod 73 =      31
NP mod 89 =      81
NP mod 101 =     80
p = 1169201309864722334562947866173026415724746034380 8
order = 11026744243059526
ordermod = 60433317538894718
TR = 1
Releasing 5 Tame and 5 Wild Kangaroos
.....................................................................
NP is 2*Prime!
NP= 2*5846006549323611672814742442876390689256843201587

real        0m3.924s
user        0m1.988s
sys         0m0.004s
```

B-233

```
b93030@linux7:~/Work/Schoof/miracl/sea2lab$ time ./sea2
EP[0]= 2
t[0]= 1
```

```
NP mod 5 =       1
NP mod 7 =       2
NP mod 17 =      5
NP mod 19 =      6
NP mod 31 =     25
NP mod 43 =     10
NP mod 59 =      9
NP mod 67 =     40
NP mod 71 =     70
NP mod 73 =     23
NP mod 79 =      1
NP mod 97 =      3
NP mod 103 =    56
NP mod 107 =     9
NP mod 109 =    36
NP mod 127 =    45
p =
13803492693581127574869511724554050904902217944340773110325048447598592
order = 102666611743167062960537 86
ordermod = 721919514870007022097525430
TR = 1
Releasing 5 Tame and 5 Wild Kangaroos
.....................................
NP is 2*Prime!
NP=
2*690174634679056378743475586227702555583981273734501355537938363448546
3

real       0m7.999s
user       0m5.188s
sys        0m0.012s
```

B-283

```
b93030@linux7:~/Work/Schoof/miracl/sea2lab$ ./sea2
EP[0]= 2
t[0]= 1
NP mod 7 =       6
NP mod 19 =      8
NP mod 23 =     12
NP mod 29 =     11
NP mod 37 =     33
NP mod 47 =     32
NP mod 61 =     51
NP mod 67 =     15
NP mod 79 =     12
NP mod 83 =     68
NP mod 97 =     75
NP mod 103 =    52
NP mod 107 =    26
NP mod 109 =     3
NP mod 127 =    91
NP mod 137 =    28
NP mod 149 =   120
NP mod 167 =    73
```

```
p =
155413511378058325673556952545881512531392547124171161700144992779
11234281641667985408
order = 356009306024364040769107395835602
ordermod = 417122038447871102863749694823942
TR = 1
Releasing 5 Tame and 5 Wild Kangaroos
.....................................................
NP is 2*Prime!
NP=
2*777067556890291628367784762729407562656962592437690488910919652 6
770044277787378692871

real    0m24.733s
user    0m24.258s
sys     0m0.044s
```

B-409

```
b93030@linux7:~/Work/Schoof/miracl/sea2lab$ time ./sea2
EP[0]= 2
t[0]= 1
NP mod 11 =    1
NP mod 13 =    6
NP mod 17 =    5
NP mod 19 =    2
NP mod 23 =   13
NP mod 37 =   26
NP mod 41 =   17
NP mod 43 =   18
NP mod 59 =   24
NP mod 71 =    7
NP mod 83 =   26
NP mod 89 =   18
NP mod 97 =   68
NP mod 101 =   78
NP mod 107 =   87
NP mod 131 =  119
NP mod 137 =  113
NP mod 151 =   93
NP mod 157 =    7
NP mod 167 =   77
NP mod 179 =   74
NP mod 181 =    1
NP mod 191 =   89
NP mod 193 =  120
NP mod 197 =   98
NP mod 211 =   29
NP mod 223 =   68
p =
132211193758049719790383061606554207965680936592856243856929759054
88115824726226916503784208794305696951824240500467166085 12
order = 31279439800404487940737038734445836965895214971677 58
ordermod = 353644194687461591052708912767398352517843868498 4198
TR = 1
Releasing 5 Tame and 5 Wild Kangaroos
.............................
```

```
NP is 2*Prime!
NP=
2*661055968790248598951915308032771039828404682964281219284648798
041577748273748052081437237621791109659798672883665675267 71

real    1m17.834s
user    1m11.440s
sys     0m0.136s
```

B-571

```
b93030@linux7:~/Work/Schoof/miracl/sea2lab$ time ./sea2
EP[0]= 2
t[0]= 1
NP mod 7 =      6
NP mod 11 =     3
NP mod 13 =     1
NP mod 19 =     6
NP mod 23 =    21
NP mod 41 =    13
NP mod 43 =     3
NP mod 53 =    21
NP mod 79 =    48
NP mod 89 =    45
NP mod 127 =   32
NP mod 137 =  107
NP mod 139 =   71
NP mod 149 =   57
NP mod 157 =   97
NP mod 167 =  123
NP mod 173 =   69
NP mod 179 =   10
NP mod 191 =  176
NP mod 193 =  140
NP mod 197 =   23
NP mod 199 =   50
NP mod 239 =   89
NP mod 241 =  218
NP mod 257 =  238
NP mod 263 =   88
NP mod 269 =  162
NP mod 277 =  180
NP mod 281 =  134
NP mod 311 =  227
NP mod 313 =  160
NP mod 317 =  219
NP mod 331 =  130
NP mod 347 =  152
NP mod 379 =   16
NP mod 397 =   95
p =
772907504603451668939070378186397468859785465941286999731447050290
303828457912084907238753316384515592492723206300435435473015732208
597531148581734693416149739396162964 6848
order =
970034550053118781795951067441961126647316372165136120077405153276
```

```
1794863602
ordermod =
105641544396289499044637429723725678203472534631045243223462822269
12750332226
TR = 1
Releasing 5 Tame and 5 Wild Kangaroos
.....
NP is 2*Prime!
NP=
2*386453752301725834469535189093198734429892732970643499865723525 1
451519142289560424536143999389415773083133881121926944486246872462
816813070234528288303332411393191105285703

real    7m16.755s
user    7m12.763s
sys     0m0.656s
```

B.2 NIST Koblitz Curves

```
b93030@bsd2:~/Work/Schoof/miracl$ ./DtoH
5846006549323611672814741753598448348329118574063
dec:      5846006549323611672814741753598448348329118574063
hex:      00000004 00000000 00000000 00020108 A2E0CC0D 99F8A5EF

b93030@bsd2:~/Work/Schoof/miracl$ ./DtoH
3450873173395281893717377931138512760570940988862252126328087024741343
dec:
3450873173395281893717377931138512760570940988862252126328087024741343
hex:      00000080 00000000 00000000 00000000 00069D5B B915BCD4
6EFB1AD5 F173ABDF

b93030@bsd2:~/Work/Schoof/miracl$ ./DtoH
3885337784451458141838923813647037813284811733793061324295874997529
81582970442260387 3
dec:
3885337784451458141838923813647037813284811733793061324295874997529
81582970442260387 3
hex:      01FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFE9AE 2ED07577
265DFF7F 94451E06 1E163C61

b93030@bsd2:~/Work/Schoof/miracl$ ./DtoH
3305279843951242994759576540163855199142023414821406096423243950228
807112892491910506732584577774580140963665906177313586 71
dec:
3305279843951242994759576540163855199142023414821406096423243950228
807112892491910506732584577774580140963665906177313586 71
hex:      007FFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
FFFFFE5F 83B2D4EA 20400EC4 557D5ED3 E3E7CA5B 4B5C83B8
E01E5FCF

b93030@bsd2:~/Work/Schoof/miracl$ ./DtoH
1932268761508629172347675945465993672149463664853217499328617625725
759571144780212268133978522706711834706712800825351461273674974066
6617311929682421 6
1709250355573368527 6673
dec:
1932268761508629172347675945465993672149463664853217499328617625725
759571144780212268133978522706711834706712800825351461273674974066
661731192968242161709250355573368527 6673
hex:      02000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 131850E1 F19A63E4 B391A8DB 917F4138
B630D84B E5D63938 1E91DEB4 5CFE778F 637C1001
```

K-163: a=1, b=1, h=2, f(x) = x^163 + x^7 + x^6 + x^3 + 1
n = 0x 00000004 00000000 00000000 00020108 A2E0CC0D 99F8A5EF

```
b93030@linux7:~/Work/Schoof/miracl/sea2lab2$ time ./fl 163 7 6 3 1 1
M= 163, B= 1
L=     3,                atkin!!
L=     5,                atkin!!
L=     7, elkies!!
L=    11, elkies!!
```

```
L=   13,                atkin!!
L=   17,                atkin!!
L=   19,                atkin!!
L=   23, elkies!!
L=   29, elkies!!
L=   31,                atkin!!
L=   37, elkies!!
L=   41,                atkin!!
L=   43, elkies!!
L=   47,                atkin!!
L=   53, elkies!!
L=   59,                atkin!!
L=   61,                atkin!!
L=   67, elkies!!
L=   71, elkies!!
L=   73,                atkin!!
L=   79, elkies!!

real    0m0.663s
user    0m0.460s
sys     0m0.160s

b93030@linux7:~/Work/Schoof/miracl/sea2lab2$ time ./sea2
EP[0]= 2
t[0]= 1
NP mod 7 =      2
NP mod 11 =     3
NP mod 23 =     12
NP mod 29 =     17
NP mod 37 =     19
NP mod 43 =     41
NP mod 53 =     36
NP mod 67 =     62
NP mod 71 =     44
NP mod 79 =     63
p = 116920130986472233456294786617302641572474603438 08
order = 3029291667134560
ordermod = 3255013894150942
TR = 1
Releasing 5 Tame and 5 Wild Kangaroos
.....................................................
NP is 2*Prime!
NP= 2*5846006549323611672814741753598448348329118574063

real    0m2.174s
user    0m2.056s
sys     0m0.004s
```

K-233: a=0, b=1, h=4, f(x) = x^233 + x^74 + 1
n = 0x 00000080 00000000 00000000 00000000 00069D5B B915BCD4 6EFB1AD5
F173ABDF

```
b93030@linux7:~/Work/Schoof/miracl/sea2lab2$ time ./fl 233 74 0 1
M= 233, B= 1
L=    3,                atkin!!
```

```
L=     5,                  atkin!!
L=     7, elkies!!
L=    11, elkies!!
L=    13,                  atkin!!
L=    17,                  atkin!!
L=    19,                  atkin!!
L=    23, elkies!!
L=    29, elkies!!
L=    31,                  atkin!!
L=    37, elkies!!
L=    41,                  atkin!!
L=    43, elkies!!
L=    47,                  atkin!!
L=    53, elkies!!
L=    59,                  atkin!!
L=    61,                  atkin!!
L=    67, elkies!!
L=    71, elkies!!
L=    73,                  atkin!!
L=    79, elkies!!
L=    83,                  atkin!!
L=    89,                  atkin!!
L=    97,                  atkin!!
L= 101,                   atkin!!
L= 103,                   atkin!!
L= 107, elkies!!
L= 109, elkies!!
L= 113, elkies!!
L= 127, elkies!!
L= 131,                   atkin!!
L= 137, elkies!!

real        0m2.133s
user        0m1.940s
sys         0m0.184s

b93030@linux7:~/Work/Schoof/miracl/sea2lab2$ time ./sea2
EP[0]= 2
t[0]= 1
NP mod 7 =        2
NP mod 11 =       4
NP mod 23 =       8
NP mod 29 =      15
NP mod 37 =      29
NP mod 43 =       3
NP mod 53 =      39
NP mod 67 =      41
NP mod 71 =      16
NP mod 79 =       2
NP mod 107 =      73
NP mod 109 =      63
NP mod 113 =      56
NP mod 127 =     119
NP mod 137 =      77
p =
13803492693581127574869511724554050904902217944340773110325048447598592
order = 33281498335308066775517546
ordermod = 746390071761036012214155 02
```

```
TR = 0
Releasing 5 Tame and 5 Wild Kangaroos
.....................................
NP is 4*Prime!
NP=
4*345087317339528189371737793113851276057094098886225212632808702474134
43

real    0m4.715s
user    0m4.236s
sys     0m0.012s
```

K-283: a=0, b=1, h=4, f(x) = x^283 + x^12 + x^7 + x^5 + 1
n = 0x 01FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFE9AE 2ED07577 265DFF7F
265DFF7F 94451E06 1E163C61

```
b93030@linux7:~/Work/Schoof/miracl/sea2lab2$ time ./fl 283 12 7 5 0 1
M= 283, B= 1
L=   3,              atkin!!
L=   5,              atkin!!
L=   7, elkies!!
L=  11, elkies!!
L=  13,              atkin!!
L=  17,              atkin!!
L=  19,              atkin!!
L=  23, elkies!!
L=  29, elkies!!
L=  31,              atkin!!
L=  37, elkies!!
L=  41,              atkin!!
L=  43, elkies!!
L=  47,              atkin!!
L=  53, elkies!!
L=  59,              atkin!!
L=  61,              atkin!!
L=  67, elkies!!
L=  71, elkies!!
L=  73,              atkin!!
L=  79, elkies!!
L=  83,              atkin!!
L=  89,              atkin!!
L=  97,              atkin!!
L= 101,             atkin!!
L= 103,             atkin!!
L= 107, elkies!!
L= 109, elkies!!
L= 113, elkies!!
L= 127, elkies!!
L= 131,             atkin!!
L= 137, elkies!!
L= 139,             atkin!!
L= 149, elkies!!
L= 151, elkies!!
L= 157,             atkin!!
L= 163, elkies!!
```

```
real        0m4.138s
user        0m3.232s
sys         0m0.136s

b93030@linux7:~/Work/Schoof/miracl/sea2lab2$ time ./sea2
EP[0]= 2
t[0]= 1
NP mod 7 =        4
NP mod 11 =       4
NP mod 23 =      12
NP mod 29 =       4
NP mod 37 =       6
NP mod 43 =      29
NP mod 53 =      31
NP mod 67 =      21
NP mod 71 =       4
NP mod 79 =      55
NP mod 107 =      13
NP mod 109 =      43
NP mod 113 =      14
NP mod 127 =      38
NP mod 137 =      61
NP mod 149 =      61
NP mod 151 =     127
NP mod 163 =      45
p =
155413511378058325673556952545881512531392547124171161700144992779
11234281641667985408
order = 976221562707951485685752726262 80
ordermod = 27372639266019025259254226285 8174
TR = 0
Releasing 5 Tame and 5 Wild Kangaroos
.....................................................
NP is 4*Prime!
NP=
4*388533778445145814183892381364703781328481173379306 1324295874997
529815829704422603873

real        0m17.758s
user        0m16.393s
sys         0m0.012s
```

K-409: a=0, b=1, h=4, f(x) = x^409 + x^87 + 1
n = 0x 007FFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFE5F
83B2D4EA 20400EC4 557D5ED3 E3E7CA5B 4B5C83B8 E01E5FCF

```
b93030@linux7:~/Work/Schoof/miracl/sea2lab2$ time ./fl 409 87 0 1
M= 409, B= 1
L=     3,                atkin!!
L=     5,                atkin!!
L=     7, elkies!!
L=    11, elkies!!
L=    13,                atkin!!
L=    17,                atkin!!
```

```
L=   19,                atkin!!
L=   23, elkies!!
L=   29, elkies!!
L=   31,                atkin!!
L=   37, elkies!!
L=   41,                atkin!!
L=   43, elkies!!
L=   47,                atkin!!
L=   53, elkies!!
L=   59,                atkin!!
L=   61,                atkin!!
L=   67, elkies!!
L=   71, elkies!!
L=   73,                atkin!!
L=   79, elkies!!
L=   83,                atkin!!
L=   89,                atkin!!
L=   97,                atkin!!
L=  101,               atkin!!
L=  103,               atkin!!
L=  107, elkies!!
L=  109, elkies!!
L=  113, elkies!!
L=  127, elkies!!
L=  131,               atkin!!
L=  137, elkies!!
L=  139,               atkin!!
L=  149, elkies!!
L=  151, elkies!!
L=  157,               atkin!!
L=  163, elkies!!
L=  167,               atkin!!
L=  173,               atkin!!
L=  179, elkies!!
L=  181,               atkin!!
L=  191, elkies!!
L=  193, elkies!!
L=  197, elkies!!
L=  199,               atkin!!
L=  211, elkies!!
L=  223,               atkin!!
L=  227,               atkin!!
L=  229,               atkin!!
L=  233, elkies!!
L=  239, elkies!!
L=  241,               atkin!!
L=  251,               atkin!!
L=  257,               atkin!!
L=  263, elkies!!

real    0m23.650s
user    0m19.669s
sys     0m0.184s

b93030@linux7:~/Work/Schoof/miracl/sea2lab2$ time ./sea2
EP[0]= 2
t[0]= 1
```

```
NP mod 7 =       4
NP mod 11 =      2
NP mod 23 =      8
NP mod 29 =      8
NP mod 37 =     20
NP mod 43 =     29
NP mod 53 =     39
NP mod 67 =     39
NP mod 71 =     40
NP mod 79 =     38
NP mod 107 =    61
NP mod 109 =    68
NP mod 113 =    94
NP mod 127 =    38
NP mod 137 =     4
NP mod 149 =    89
NP mod 151 =    92
NP mod 163 =   158
NP mod 179 =   160
NP mod 191 =    83
NP mod 193 =     4
NP mod 197 =   164
NP mod 211 =   102
NP mod 233 =    14
NP mod 239 =    67
NP mod 263 =   236
p =
1322111937580497197903830616065542079656809365928562438569297590548811582472622691650378420879430569695182424050046716608512
order = 45126125362876995145770115695399886610860137225038 4
ordermod = 109955924193908062660617169918360237953159542388274 6
TR = 0
Releasing 5 Tame and 5 Wild Kangaroos
.....
NP is 4*Prime!
NP=
4*330527984395124299475957654016385519914202341482140609642324395022880711289249191050673258457777458014096366590617731358671

real    0m18.069s
user    0m17.333s
sys     0m0.020s
```

K-571: a=0, b=1, h=4, f(x) = x^571 + x^10 + x^5 + x^2 + 1
n = 0x 02000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 131850E1 F19A63E4 B391A8DB 917F4138 B630D84B E5D63938 1E91DEB4 5CFE778F 637C1001

```
b93030@linux7:~/Work/Schoof/miracl/sea2lab2$ time ./fl 571 10 5 2 0 1
M= 571, B= 1
L=     3,              atkin!!
L=     5,              atkin!!
L=     7, elkies!!
L=    11, elkies!!
L=    13,              atkin!!
L=    17,              atkin!!
L=    19,              atkin!!
```

L= 23, elkies!!
L= 29, elkies!!
L= 31, atkin!!
L= 37, elkies!!
L= 41, atkin!!
L= 43, elkies!!
L= 47, atkin!!
L= 53, elkies!!
L= 59, atkin!!
L= 61, atkin!!
L= 67, elkies!!
L= 71, elkies!!
L= 73, atkin!!
L= 79, elkies!!
L= 83, atkin!!
L= 89, atkin!!
L= 97, atkin!!
L= 101, atkin!!
L= 103, atkin!!
L= 107, elkies!!
L= 109, elkies!!
L= 113, elkies!!
L= 127, elkies!!
L= 131, atkin!!
L= 137, elkies!!
L= 139, atkin!!
L= 149, elkies!!
L= 151, elkies!!
L= 157, atkin!!
L= 163, elkies!!
L= 167, atkin!!
L= 173, atkin!!
L= 179, elkies!!
L= 181, atkin!!
L= 191, elkies!!
L= 193, elkies!!
L= 197, elkies!!
L= 199, atkin!!
L= 211, elkies!!
L= 223, atkin!!
L= 227, atkin!!
L= 229, atkin!!
L= 233, elkies!!
L= 239, elkies!!
L= 241, atkin!!
L= 251, atkin!!
L= 257, atkin!!
L= 263, elkies!!
L= 269, atkin!!
L= 271, atkin!!
L= 277, elkies!!
L= 281, elkies!!
L= 283, atkin!!
L= 293, atkin!!
L= 307, atkin!!
L= 311, atkin!!
L= 313, atkin!!
L= 317, elkies!!

```
L= 331, elkies!!
L= 337, elkies!!
L= 347, elkies!!
L= 349,                 atkin!!
L= 353,                 atkin!!
L= 359, elkies!!
L= 367,                 atkin!!
L= 373, elkies!!
L= 379, elkies!!
L= 383,                 atkin!!
L= 389, elkies!!

real    1m9.348s
user    1m8.760s
sys     0m0.588s

b93030@linux7:~/Work/Schoof/miracl/sea2lab2$ time ./sea2
EP[0]= 2
t[0]= 1
NP mod 7 =       4
NP mod 11 =      4
NP mod 23 =      2
NP mod 29 =     28
NP mod 37 =      6
NP mod 43 =     10
NP mod 53 =      2
NP mod 67 =     34
NP mod 71 =     57
NP mod 79 =     39
NP mod 107 =    98
NP mod 109 =     3
NP mod 113 =    82
NP mod 127 =    16
NP mod 137 =    82
NP mod 149 =     9
NP mod 151 =    19
NP mod 163 =   158
NP mod 179 =    18
NP mod 191 =     4
NP mod 193 =   122
NP mod 197 =   123
NP mod 211 =    27
NP mod 233 =   163
NP mod 239 =   161
NP mod 263 =    35
NP mod 277 =   239
NP mod 281 =   149
NP mod 317 =   135
NP mod 331 =    74
NP mod 337 =   317
NP mod 347 =    77
NP mod 359 =   218
NP mod 373 =   324
NP mod 379 =   150
NP mod 389 =     5
p =
772907504603451668939070378186397468859785465941286999731447050290
```

```
303828457912084907238753316384515592492723206300435435473015732208
597531148581734693416149739396162964684 8
order =
996400197857582456187342835750237922856646781353582747097669177619
7661738262
ordermod =
207321071864533349878302816672499785552563184397961155174122021914
84681515202
TR = 0
Releasing 5 Tame and 5 Wild Kangaroos
.....
NP is 4*Prime!
NP=
4*193226876150862917234767594546599367214946366485321749932861762 5
725759571144780212268133978522706711834706712800825351461273674974
066617311929682421617092503555733685276673

real    0m51.395s
user    0m46.179s
sys     0m0.132s
```