SCIS 2002 The 2002 Symposium on Cryptography and Information Security
Shirahama,Japan, Jan.29-Feb.1,2002
The Institute of Electronics,
Information and Communication Engineers

An improved baby step giant step algorithm for point counting of hyperelliptic curves over finite fields

Kazuto Matsuo * Jinhui Chao † Shigeo Tsujii ‡

Abstract— Counting the number of points on the Jacobian varieties of hyperelliptic curves over finite fields is necessary for construction of hyperelliptic curve cryptosystems. Recently Gaudry and Harley proposed a practical algorithm for point counting of hyperelliptic curves. In the Gaudry–Harley algorithm, the residue modulo $m, m \in \mathbb{Z}_{>0}$ of the order of a given Jacobian variety are computed at first, then the order is computed by a search algorithm with square-root complexity, using its modulo m residue. In fact, the parallelized λ–method is applied as the square-root algorithm in their algorithm. However the square-root algorithm part of the Gaudry–Harley algorithm took 50CPU days to compute an order of 127 bits.

This paper proposes an improvement of the baby step giant step algorithm for point counting of hyperelliptic curves and shows that the proposed algorithm carries out faster point counting than the Gaudry–Harley algorithm. Moreover, implementation results of the proposed algorithm is presented. A 135 bits order is computed in 16 hours on Alpha 21264/667MHz by using the proposed algorithm.

Keywords: Hyperelliptic curves, Point counting algorithm, Baby step giant step algorithm, Cartier–Manin operator, Schoof–like algorithm

1 Introduction

Security of a hyperelliptic curve cryptosystem depends in an essential way on the order of the Jacobian variety of the hyperelliptic curve used in the system. In particular, it is believed that for hyperelliptic curve cryptosystems using small genera curves, if the orders of their Jacobian varieties are enough large prime numbers and are coprime to the characteristic of the definition finite fields, then they are secure against any known attacks except maybe the Weil–Descent attacks. Therefore, computation of the orders of the Jacobian varieties for given random hyperelliptic curves is one of most important subject to construct hyperelliptic curve cryptosystems.

Recently several researches have been reported on point counting algorithms. Especially, efficient algorithms [Ked01, Gau01, GG01] have been proposed for curves over small characteristic finite fields and using these algorithms, it is possible to compute orders of Jacobian varieties over such fields, in sizes for cryptographic usage (e.g. 160 bits orders).

On the other hand, the situation is quite different on point counting of curves over finite fields with arbitrary characteristic. Although a number of theoretical results such as [Pil90, Kam91, AH96, HI98] have been known, it is only until very recent that a practical point counting algorithm for curves over large characteristic finite fields is proposed by Gaudry and Haley [GH00, Gau00].

In the Gaudry–Harley algorithm, the residue modulo $m, m \in \mathbb{Z}_{>0}$ of the order of a given Jacobian variety is computed at first, then the order is computed by a search algorithm of square root complexity, using its modulo m residue. (We will misuse the square-root algorithm refering to these kind of search algorithms hereafter.) This is a natural generalization of the point counting algorithm of elliptic curves proposed in [MVZ93] for hyperelliptic curves. However, the Gaudry–Harley algorithm is not yet able to compute orders in sizes for cryptographic usage. Their algorithm took 50CPU days for the square root algorithm part to compute a 127 bits order.

This paper proposes an improvement of the baby step giant step algorithm, (which is also one of the square-root algorithms), for point counting of hyperelliptic curves and then shows that the proposed algorithm carries out faster point counting of hyperelliptic curves than the Gaudry–Harley algorithm. Moreover, implementation results of the proposed algorithm are also presented. A 135 bits order is computed in 16 hours on Alpha 21264/667MHz by using the proposed algorithm.

2 Hyperelliptic curves and the orders of their Jacobian varieties

Let p be an odd prime, $\mathbb{F}_q$ a finite field of order q with $\mathrm{char}(\mathbb{F}_q) = p$. Let g be a positive integer. Then a

* Research and Development Initiative, Chuo University, 42-8 Ichigaya Honmuracho, Shinjuku-ku, Tokyo, 162-8473 Japan
† Depertment of Electrical, Electronic, and Communication Engineering, Faculty of Science and Engineering, Chuo University, 1-13-27 Kasuga, Bunkyo-ku, Tokyo, 112-8551 Japan
‡ Depertment of Information and System Engineering, Faculty of Science and Engineering, Chuo University, 1-13-27 Kasuga, Bunkyo-ku, Tokyo, 112-8551 Japan

Input: genus 2 HEC $C/\mathbb{F}_q$
Output: $\#\mathcal{J}_C(\mathbb{F}_q)$

1: Compute $\#\mathcal{J}_C(\mathbb{F}_q) \bmod 2^e$ by the halving algorithm
2: **for** prime numbers $l = 3, 5, \ldots, l_{max}$ **do**
3: Compute $\chi_q(X) \bmod l$ by a Schoof-like algorithm
4: Compute $\#\mathcal{J}_C(\mathbb{F}_q) \bmod l$ from $\chi_q(X) \bmod l$
5: **end for**
6: Compute $\chi_q(X) \bmod p$ by using the Cartier-Manin operator
7: Compute $\#\mathcal{J}_C(\mathbb{F}_q) \bmod p$ from $\chi_q(X) \bmod p$
8: Compute $\#\mathcal{J}_C(\mathbb{F}_q) \bmod m, m = 2^e \cdot 3 \cdots l_{max} \cdot p$ by CRT
9: Compute $\#\mathcal{J}_C(\mathbb{F}_q)$ by a square root algorithm using $\#\mathcal{J}_C(\mathbb{F}_q) \bmod m$

Algorithm 1: Gaudry-Harley point counting algorithm

genus g hyperelliptic curve $C/\mathbb{F}_q$ is defined as follows:

$$C : Y^2 = F(X),$$
$$F(X) = X^{2g+1} + f_{2g}X^{2g} + \cdots + f_0, \quad (1)$$

where $f_i \in \mathbb{F}_q$, $\mathrm{disc}\,(F) \neq 0$.

We restrict ourselves to $g = 2$ for simplicity.

Let $\mathcal{J}_C$ be the Jacobian variety of C, $\mathcal{J}_C(\mathbb{F}_q)$ its $\mathbb{F}_q$-rational points. It is known that $\mathcal{J}_C(\mathbb{F}_q)$ is a finite Abelian group, so that discrete logarithm based cryptosystems can be construct on it.

The characteristic polynomial $\chi_q(X)$ of the qth power Frobenius endomorphism of $\mathcal{J}_C$ is given as follows [Sti93, Kam91]:

$$\chi_q(X) = X^4 - s_1X^3 + s_2X^2 - s_1qX + q^2, \quad (2)$$

where $s_i \in \mathbb{Z}$ and

$$|s_1| \leq 4\sqrt{q}, \quad (3)$$
$$|s_2| \leq 6q. \quad (4)$$

Then the order $\#\mathcal{J}_C(\mathbb{F}_q)$ of $\mathcal{J}_C(\mathbb{F}_q)$ is obtained as

$$\#\mathcal{J}_C(\mathbb{F}_q) = \chi_q(1) = q^2 + 1 - s_1(q+1) + s_2 \quad (5)$$

from $\chi_q(X)$ [Sti93]. It is known that $\#\mathcal{J}_C(\mathbb{F}_q)$ is bounded within the Hasse–Weil range:

$$L_o = \left\lceil (\sqrt{q}-1)^4 \right\rceil \leq \#\mathcal{J}_C(\mathbb{F}_q) \leq H_o = \left\lfloor (\sqrt{q}+1)^4 \right\rfloor. \quad (6)$$

3 The Gaudry–Harley algorithm

This section gives rough descriptions of the Gaudry–Harley algorithm and its implementation results according to [GH00, Gau00]. (Refer to [GH00, Gau00] for further details).

In the Algorithm 1, we show an outline of the Gaudry–Harley algorithm.

The largest computable values of l_{max} in the Step 2 of the Algorithm 1 depend on many factors such as the size of $\mathbb{F}_q$. However, for construction of secure curves, $l_{max} = 13$ at the present. The computation of the Cartier–Manin operator appeared in the Step 6 costs exponential time in $\log p$, so that the computation is impossible for curves over prime fields with large size such as in cryptographic usage. Therefore the Step 6 and 7 are skipped when the algorithm is applied for curves over prime fields. For the details of the Cartier–Manin operator and its computation are referred to [Man63, Man65, Yui78, GH00, Gau00].

Gaudry and Harley computed the orders of the Jacobian varieties of hyperelliptic curves over a 64 bits prime field and a degree 3 extension of a 16 bits prime field respectively. The orders are 127 bits and 128 bits respectively, and these results seemed to be present records of point counting of hyperelliptic curves over prime field and over large characteristic field [Wen01].

However, it is still impractical to construct secure hyperelliptic curve cryptosystems by using the Gaudry–Harley algorithm. For an example, their algorithm took 50CPU days in the Step 9 when they computed the 127 bits order. In fact, the point counting algorithms have to be repeated before a secure curve is found.

4 A baby step giant step algorithm using $\#\mathcal{J}_C(\mathbb{F}_q) \bmod m$

It is known that comparing with the baby step giant step algorithm, the parallelized λ–method used in the Step 9 in the Algorithm 1 possesses merits such as can be parallelized and and take only constant memory space. While the computational complexities in CPU time of both algorithms remain essentially the same.

In this section, as a preliminary to the following sections, we describe a variation of the standard baby step giant step algorithm which can be used in the Step 9 in the Algorithm 1. This algorithm is an extension of the algorithm that has been applied for point counting of elliptic curves shown in [MVZ93].

We can assume that $N_r \in \mathbb{Z}$ such that

$$\#\mathcal{J}_C(\mathbb{F}_q) = N_r + mN_m, 0 \leq N_r < m \quad (7)$$

is given in the Step 9 of the Algorithm 1. Therefore $\#\mathcal{J}_C(\mathbb{F}_q)$ can be obtained by searching N_m among

$$\lfloor L_o/m \rfloor \leq N_m \leq \lfloor H_o/m \rfloor. \quad (8)$$

Now we set $n \approx \sqrt{R_o}$, where

$$R_o \approx (H_o - L_o)/m = 8q^{3/2}/m + O(q/m). \quad (9)$$

Then $N_m = i + nj$ can be computed by finding (i, j) such that

$$(N_r + mi)\mathcal{D} = -mnj\mathcal{D} \quad (10)$$

for ${}^\forall\mathcal{D} \in \mathcal{J}_C(\mathbb{F}_q)$ by searching a collision between the lhs and the rhs of (10) among

$$0 \leq i < n, \quad (11)$$
$$\left\lfloor \frac{L_o}{mn} \right\rfloor - 1 \leq j \leq \left\lfloor \frac{H_o}{mn} \right\rfloor. \quad (12)$$

Consequently we can compute $\#\mathcal{J}_C(\mathbb{F}_q)$ from the pair (i,j) obtained by the above computation as follows:

$$\#\mathcal{J}_C(\mathbb{F}_q) = N_r + m(i+nj). \tag{13}$$

This algorithm costs $O(q^{3/4}/\sqrt{m})$ due to (9).

Unfortunately more than one $\mathcal{D} \in \mathcal{J}_C(\mathbb{F}_q)$ should be used in order to determine (i,j) uniquely in general [Coh93]. However to apply (10) to a $\mathcal{D} \in \mathcal{J}_C(\mathbb{F}_q)\backslash\{0\}$, we can decide whether $\#\mathcal{J}_C(\mathbb{F}_q)$ is a prime number or not, then compute $\#\mathcal{J}_C(\mathbb{F}_q)$ when $\#\mathcal{J}_C(\mathbb{F}_q)$ is a prime number. This is sufficient for application to cryptosystems. Therefore we hereafter assume such situation for simplicity.

5 An improved baby step giant step algorithm

In the Algorithm 1 in the section 3, the residues modulo m of s_i for $\chi_q(X)$ in (2), as well as $\#\mathcal{J}_C(\mathbb{F}_q)$ mod m, can be obtained by either the Schoof-like algorithm or the Cartier–Manin operator.

Thereupon this section proposes an improved baby step giant step algorithm, which uses effectively the residues s_i mod m and is faster than the baby step giant step algorithm shown in the section 4.

Firstly, we show tighter estimates of the boundaries of s_i in (3), (4) in the following lemma.

Lemma 1. *The s_1 in (2) is bounded by*

$$s_{1l} = -\lfloor 4\sqrt{q} \rfloor \le s_1 \le s_{1u} = \lfloor 4\sqrt{q} \rfloor \tag{14}$$

and the s_2 by

$$s_{2l} = \lceil 2\sqrt{q}|s_1| - 2q \rceil \le s_2 \le s_{2u} = \left\lfloor \frac{1}{4}s_1^2 + 2q \right\rfloor. \tag{15}$$

Remark 1. The upper bound in (15) is due to [Elk95]. The lower bound was pointed out to the authors by Prof. Fumiyuki Momose.

Now we assume that $s_i' \in \mathbb{Z}$ which satisfy the following equations are given.

$$0 \le s_i' < m, \tag{16}$$
$$s_1 = s_1' + mt_1, t_1 \in \mathbb{Z}, \tag{17}$$
$$s_2 = s_2' + mt_2', t_2' \in \mathbb{Z} \tag{18}$$

Then the t_1 in (17) is bounded by

$$L_1 = \left\lfloor \frac{s_{1l}}{m} \right\rfloor \le t_1 \le H_1 = \left\lfloor \frac{s_{1u}}{m} \right\rfloor \tag{19}$$

due to (14) and the t_2' in (18) is bounded by

$$L_2' = \left\lfloor \frac{s_{2l}}{m} \right\rfloor \le t_2' \le H_2' = \left\lfloor \frac{s_{2u}}{m} \right\rfloor \tag{20}$$

due to (15). Moreover let t_2, t_3 be integers which satisfy

$$t_2' = t_2 + nt_3,\ t_2, t_3 \in \mathbb{Z}, \tag{21}$$
$$0 \le t_2 < n \tag{22}$$

for a positive integer n, then t_3 is bounded by

$$L_3 = \left\lfloor \frac{s_{2l}}{mn} \right\rfloor - 1 \le t_3 \le H_3 = \left\lfloor \frac{s_{2u}}{mn} \right\rfloor \tag{23}$$

due to (20).

Consequently we have

$$\#\mathcal{J}_C(\mathbb{F}_q) = q^2+1-s_1'(q+1)+s_2'-m(q+1)t_1+mt_2+mnt_3 \tag{24}$$

by substituting (17), (18), (21) to (5). Hence $\#\mathcal{J}_C(\mathbb{F}_q)$ can be computed by finding (t_1, t_2, t_3) satisfying

$$(q^2+1-s_1'(q+1)+s_2'-m(q+1)t_1+mnt_3)\mathcal{D} = -mt_2\mathcal{D} \tag{25}$$

for $^\forall\mathcal{D} \in \mathcal{J}_C(\mathbb{F}_q)$ among the ranges in (19), (22), (23). These computations are executed by collision searching between the lhs and the rhs of (25).

Next we determine the most effective value of n.

The number of the pairs (s_1, s_2) is roughly $32q^{3/2}/3$ because

$$\int \frac{1}{4}s_1^2 + 2q - (2\sqrt{q}|s_1| - 2q)\mathrm{d}s_1 = s_1(\frac{1}{12}s_1^2 - \sqrt{q}|s_1| + 4q). \tag{26}$$

Therefore the number S of the triples (t_1, t_2, t_3) is

$$S \approx \frac{32q^{3/2}}{3m^2}. \tag{27}$$

Now we set n as

$$n \approx \sqrt{S} = \frac{4\sqrt{6}q^{3/4}}{3m} \tag{28}$$

then the number of point additions for all (t_1, t_2, t_3) is roughly n in both the lhs and the rhs of (26). The algorithm under this setting works the most efficiently. Therefore the computational complexity of the algorithm is $O(q^{3/4}/m)$ due to (28) and the computation of $\#\mathcal{J}_C(\mathbb{F}_q)$ by the algorithm is $O(\sqrt{m})$ times faster than the algorithm shown in the section 4.

To examplify the proposed algorithm, its application to prime order computation is shown in the Algorithm 2.

Remark 2. The s_i which are obtained in the process of the algorithm shown in this section are not always correct values [Elk95]. The algorithm should be slightly modified in order to find correct s_i.

Remark 3. The algorithm proposed in this section costs $O(q^{g(g+1)/8}/m^{g/2})$ time for genus g curves. On the other hand the algorithm shown in the section 4 costs $O(q^{(2g-1)/4}/\sqrt{m})$. Therefore, the proposed algorithm is faster than the previous algorithm in the case of $m > q^{(g-2)/4}$ for genus $g > 2$ curves. In the case of $m \le q^{(g-2)/4}$, in order that the proposed algorithm is faster than the previous algorithm, one needs e.g. take the properties described in the Remark 2 into consideration.

Input: A genus 2 HEC $C/\mathbb{F}_q$, $m, s_1', s_2' \in \mathbb{Z}_{>0}$ such that $s_i \equiv s_i' \bmod m$ and $0 \le s_i' < m$
Output: $\#\mathcal{J}_C(\mathbb{F}_q)$, if it is a prime number

1: $n \leftarrow \lfloor 4\sqrt{6}q^{3/4}/(3m) \rceil$
2: $l \leftarrow q^2 + 1 - s_1'(q+1) + s_2'$
3: Choose a random $\mathcal{D} \in \mathcal{J}_C(\mathbb{F}_q) \backslash \{0\}$
4: $B \leftarrow \{(B_j = -jm\mathcal{D}, j) \mid 0 \le j < n\}$
5: Sort B by B_j
6: $\mathcal{D}_1 \leftarrow l\mathcal{D}$
7: **for** $i = -\lceil \lfloor 4\sqrt{q} \rfloor / m \rceil \ldots \lfloor 4\sqrt{q}/m \rfloor$ **do**
8: $\quad \mathcal{D}_2 \leftarrow \mathcal{D}_1 - im(q+1)\mathcal{D}$
9: $\quad s_1 \leftarrow s_1' + im$
10: $\quad$ **for** $k = \lfloor (\lceil 2\sqrt{q}|s_1| \rceil - 2q)/(mn) \rfloor - 1 \ldots \lfloor (\lfloor s_1^2/4 \rfloor + 2q)/(mn) \rfloor$ **do**
11: $\quad\quad \mathcal{D}_3 \leftarrow \mathcal{D}_2 + kmn\mathcal{D}$
12: $\quad\quad$ **if** $^\exists j$ such that $B_j = \mathcal{D}_3$ **then**
13: $\quad\quad\quad l \leftarrow l + (-i(q+1) + j + kn)m$
14: $\quad\quad\quad$ **if** $l =$ a prime number **then**
15: $\quad\quad\quad\quad$ Output l as $\#\mathcal{J}_C(\mathbb{F}_q)$ and terminate
16: $\quad\quad\quad$ **else**
17: $\quad\quad\quad\quad$ $\#\mathcal{J}_C(\mathbb{F}_q)$ is not a prime number and terminate
18: $\quad\quad\quad$ **end if**
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: **end for**

Algorithm 2: An improved baby step giant step algorithm for finding prime order curves

6 Implementation and construction of prime order curves

The Algorithm 2 is implemented to construct genus 2 prime order hyperelliptic curves. We also implemented computation of both $s_i \bmod 2$ and $s_i \bmod p$ by the Cartier–Manin operator in order to obtain $s_i \bmod m$.

6.1 Computation of $s_i \bmod 2$

The halving algorithm used to compute $\#\mathcal{J}_C(\mathbb{F}_q) \bmod 2^e$ by Gaudry and Harley cannot compute $s_i \bmod 2^e$. However, for construction of prime order curves, the residues modulo 2 are in fact fixed when $2 \nmid \#\mathcal{J}_C(\mathbb{F}_q)$. Below, we selected irreducible F and set $s_i \equiv 1 \bmod 2$ according to the Lemma 2.

Lemma 2.

$$2 \nmid \#\mathcal{J}_C(\mathbb{F}_q) \Leftrightarrow F : \textit{irreducible}/\mathbb{F}_q \Leftrightarrow 2 \nmid s_i \quad (29)$$

Remark 4. Dr. Seigo Arita pointed out to the authors that the residues $s_i \bmod 2$ are completely determined from the factors of F.

6.2 Computation of $s_i \bmod p$

We computed $s_i \bmod p$ by using the Cartier–Manin operator [Man63, Man65, Yui78, GH00, Gau00].

The dominant part of Cartier–Manin operator computation is to compute

$$U = \sum u_i X^i = F^{(p-1)/2} \quad (30)$$

for F in (1), when the characteristic p is large. This computation itself can be efficiently executed by the FFT multiplication. Here we speed up it even further by the following tricks. We firstly compute

$$V = \sum v_i X^i = \begin{cases} F^{(p-1)/4}, & \text{if } 4 \mid p-1 \\ F^{(p-3)/4}, & \text{if } 4 \nmid p-1 \end{cases} \quad (31)$$

by the FFT multiplication. Then it is sufficient to compute only $u_{p-2}, u_{p-1}, u_{2p-2}, u_{2p-1}$ to determine $s_i \bmod p$, from v_i, f_i as

$$U = \begin{cases} V^2, & \text{if } 4 \mid p-1. \\ FV^2, & \text{if } 4 \nmid p-1. \end{cases} \quad (32)$$

This technique can reduce both the computational time and the required memory space needed by the original version using FFT by roughly 1/2 times.

6.3 Implementation of the Algorithm 2

The Algorithm 2 is speeded up by using the following techniques in the implementation.

1. Both the computational time and the required memory space can be reduced by roughly $1/\sqrt{2}$ by use of the property that $-\mathcal{D}$ can be obtained easily for given $\mathcal{D} \in \mathcal{J}_C(\mathbb{F}_q)$. This is done by choice of the value of n to be $\sqrt{2}$ times of the value in the (28), and the boundaries of t_2 to be $-n \le t_2 \le n-1$, also the condition of j in the Step 12 to be $B_j = \pm\mathcal{D}_3$ and so on.

2. Although the Algorithm 2 is designed to minimize the cost of the worst case computation, it is more appropriate to design an algorithm minimizing the average cost for computation of prime order curves. We can minimize the average cost by choice of the value of n as $1/\sqrt{2}$ times of the value in the (28) [Tes01]. This reduced both the average time and the memory space by roughly $1/\sqrt{2}$ times.

3. We use 32 bits hash values of $-jm\mathcal{D}$ in the table B and the precomputation table described in [LMMS94].

4. The practical speed of the Algorithm 2 depends on the addition speed on $\mathcal{J}_C(\mathbb{F}_q)$. So we use an improved Harley addition algorithm shown in [MCT01].

5. The algorithm will terminated once one checked out that $\mathcal{J}_C(\mathbb{F}_q)$ has a non-prime order.

Remark 5. Both the average time and the memory space is reduced by roughly 1/2 times using the techniques of 1. and 2. simultanously. The n in (28) is not modified when both 1. and 2. are used.

6.4 Implementation results

The Algorithm 3 shows an outline of the construction algorithm of prime order curves used in this section.

Input: A finite field $\mathbb{F}_q$ and $p = \mathrm{char}(\mathbb{F}_q)$
Output: A prime order curve C and $\#\mathcal{J}_C(\mathbb{F}_q)$

1: **repeat**
2: Choose a monic irreducible polynomial $F/\mathbb{F}_q, \deg F = 5$ randomly
3: $C : Y^2 = F$
4: Compute $s_{CMi} \equiv s_i \bmod p, 0 \leq s_{CMi} < p$ by using the Cartier–Manin operator
5: $m \leftarrow 2p$, $s'_i \leftarrow s_{CMi}$ if $2 \nmid s_{CMi}$, else $s'_i \leftarrow s_{CMi} + p$
6: Compute $\#\mathcal{J}_C(\mathbb{F}_q)$ by the Algorithm 2
7: **until** $\#\mathcal{J}_C(\mathbb{F}_q) =$ a prime number
8: Output C and $\#\mathcal{J}_C(\mathbb{F}_q)$

Algorithm 3: Construction of a prime order genus 2 hyperelliptic curve

This section shows two examples of genus 2 hyperelliptic curves with prime orders constructed by the Algorithm 3 and also timings to calculate their orders.

These computations are conducted on a Pentium III/866MHz with 1GB RAM and a Alpha 21264/667MHz with 4GB RAM respectively. The NTL [Sho01] is used for finite field and polynomial operations.

Example 1. A 123 bits prime order curve

$$C_1/\mathbb{F}_q : Y^2 = F_1(X),$$
$$F_1 = X^5 + (567033\alpha^2 + 322876\alpha + 957805)X^4 + (1123698\alpha^2 + 933051\alpha + 141410)X^3 + (393269\alpha^2 + 233572\alpha + 708577)X^2 + (692270\alpha^2 + 350968\alpha + 788883)X + 968896\alpha^2 + 895453\alpha + 589750$$

is obtained by the Algorithm 3, where

$$\mathbb{F}_q = \mathbb{F}_p(\alpha),$$
$$\alpha^3 + 1073470\alpha^2 + 34509\alpha + 1223366 = 0,$$
$$p = 1342181.$$

The order of $\mathcal{J}_{C_1}(\mathbb{F}_q)$ is

$$\#\mathcal{J}_{C_1}(\mathbb{F}_q) = 584610376401469447932232931574028593 1.$$

The computation of $\#\mathcal{J}_{C_1}(\mathbb{F}_q)$ costs 197 minutes on Pentium III/866MHz. The Table 1 shows the timing of main parts of the Algorithm 3 and the Algorithm 2 for computing $\#\mathcal{J}_{C_1}(\mathbb{F}_q)$.

Example 2. A 135 bits prime order curve

$$C_2/\mathbb{F}_q : Y^2 = F_2(X),$$
$$F_2 = X^5 + (2817153\alpha^2 + 3200658\alpha + 1440424)X^4 + (3310325\alpha^2 + 481396\alpha + 1822351)X^3 + (108275\alpha^2 + 120315\alpha + 469800)X^2 + (2168383\alpha^2 + 1244383\alpha + 5010679)X + 4682337\alpha^2 + 53865\alpha + 2540378$$

is obtained by the Algorithm 3, where

$$\mathbb{F}_q = \mathbb{F}_p(\alpha),$$
$$\alpha^3 + 4519302\alpha^2 + 3749080\alpha + 607603 = 0,$$
$$p = 5491813.$$

The order of $\mathcal{J}_{C_2}(\mathbb{F}_q)$ is

$$\#\mathcal{J}_{C_2}(\mathbb{F}_q) = 27434335457581234045473311611818187339271.$$

The computation of $\#\mathcal{J}_{C_2}(\mathbb{F}_q)$ costs 16 hours on Alpha 21264/667MHz. The Table 2 shows the timing of main parts of the Algorithm 3 and the Algorithm 2 for computing $\#\mathcal{J}_{C_2}(\mathbb{F}_q)$.

Remark 6. In both the Example 1 and 2, the giant steps (Step 6–21 in Algorithm 2) were slower than the baby steps (Step 4, 5 in Algorithm 2). However the average cost of the giant steps is the same as of the baby step. Moreover, the cost of the baby step is fixed for a fixed definition field.

Algorithm	Step	Time (min.)
Algorithm 3	Step 4	7
Algorithm 2	Step 4	70
	Step 5	1
	Step 6 – Step 21	119
Total		197

Table 1: Timing of computing $\#\mathcal{J}_{C_1}(\mathbb{F}_q)$ on Pentium III/866MHz

Algorithm	Step	Time (min.)
Algorithm 3	Step 4	42
Algorithm 2	Step 4	330
	Step 5	20
	Step 6 – Step 21	557
Total		949

Table 2: Timing of computing $\#\mathcal{J}_{C_2}(\mathbb{F}_q)$ on Alpha 21264/667MHz

7 Conclusion

This paper proposed an improvement of baby step giant step algorithm for point counting of hyperelliptic curves over finite fields. This improved baby step giant step algorithm can construct secure hyperelliptic curves of genus 2 more efficient than the previous algorithms. Moreover an example of a 135 bit prime order curve constructed by the improved baby step giant step algorithm was shown.

Computation of an order of a hyperelliptic curve in the size of cryptographic usage is possible if both the algorithm proposed in this paper and the Gaudry-Harley's Schoof–like algorithm are used simultanously. However further improvement of Schoof–like algorithm is necessary in order to construct secure hyperelliptic curves. This is because that the Gaudry-Harley's Schoof–like algorithm takes too long time for general

curves. This means that it will be difficult at this moment to use these algorithms to find a secure random curve.

Acknowledgement

The authors would like to thank Prof. Fumiyuki Momose for pointing out the Lemma 1. The authors would also like to thank Dr. Seigo Arita and Dr. Koh-ichi Nagao for their interesting discussions.

A part of this research was supported by Telecommunications Advancement Organization of Japan (TAO).

References

[AH96] L. M. Adleman and M. D. Huang, *Counting rational points on curves and Abelian varieties over finite fields*, ANTS-II (H. Cohen, ed.), Lecture Notes in Computer Science, no. 1122, Springer-Verlag, 1996, pp. 1–16.

[Coh93] H. Cohen, *A course in computational algebraic number theory*, Graduate Text in Mathematics, no. 138, Springer-Verlag, 1993.

[Elk95] N. D. Elkies, *Elliptic and modular curves over finite fields and related computational issues*, Computational perspectives on number theory (D. A. Buell and J. T. Teitlbaum, eds.), AMS, 1995, pp. 21–76.

[Gau00] P. Gaudry, *Algorithmique des courbes hyperelliptiques et applications à la cryptologie*, Ph.D. thesis, École polytechnique, 2000.

[Gau01] P. Gaudry, *Algorithms for counting points on curves*, ECC2001, 2001.

[GG01] P. Gaudry and N. Gürel, *An extension of Kedlaya's point–counting algorithm to superelliptic curves*, Advances in Cryptology - ASIACRYPT2001 (C. Boyd, ed.), Lecture Notes in Computer Science, no. 2248, Springer-Verlag, 2001, pp. 480–494.

[GH00] P. Gaudry and R. Harley, *Counting points on hyperelliptic curves over finite fields*, ANTS-IV (W. Bosma, ed.), Lecture Notes in Computer Science, no. 1838, Springer-Verlag, 2000, pp. 297–312.

[HI98] M. D. Huang and D. Ierardi, *Counting rational point on curves over finite fields*, J. Symbolic Computation **25** (1998), 1–21.

[Kam91] W. Kampkötter, *Explizite gleichungen für Jacobische varietäten hyperelliptischer kurven*, Ph.D. thesis, GH Essen, 1991.

[Ked01] K. S. Kedlaya, *Counting points on hyperelliptic curves using Monsky–Washinitzer cohomology*, preprint, 2001.

[LMMS94] F. Lehmann, M. Maurer, V. Müller, and V. Shoup, *Counting the number of points on elliptic curves over finite fields of characteristic greater than three*, ANTS-I (L.M. Adleman and M.D.Huang, eds.), Lecture Notes in Computer Science, no. 877, Springer-Verlag, 1994, pp. 60–70.

[Man63] J. I. Manin, *The theory of commutative formal groups over fields of finite characteristic*, Russian Mathematical Surveys **18** (1963), 1–83.

[Man65] J. I. Manin, *The Hasse–Witt matrix of an algebraic curve*, Trans. AMS **45** (1965), 245–264.

[MCT01] K. Matsuo, J. Chao, and S. Tsujii, *Fast genus two hyperelliptic curve cryptosystems*, Technical Report ISEC2001-31, IEICE Japan, 2001.

[MVZ93] A. Menezes, S. Vanstone, and R. Zuccherato, *Counting points on elliptic curves over* $\mathbb{F}_{2^m}$, Math. Comp. **60** (1993), 407–420.

[Pil90] J. Pila, *Frobenius maps of Abelian varieties and finding roots of unity in finite fields*, Math. Comp. **55** (1990), 745–763.

[Sho01] V. Shoup, *A tour of NTL*, `http://www.shoup.net/ntl/`, 2001.

[Sti93] H. Stichtenoth, *Algebraic function fields and codes*, Universitext, Springer-Verlag, 1993.

[Tes01] E. Teske, *Square–root algorithms for the discrete logarithm probrem (A survey)*, Tech. Report CORR2001-7, CACR, U. Waterloo, 2001.

[Wen01] A. Weng, *The CM–method for hyperelliptic curves*, ECC2001, 2001.

[Yui78] N. Yui, *On the Jacobian varieties of hyperelliptic curves over fields of characteristic* $p > 2$, J. Algebra **52** (1978), 378–410.