On the Index Calculus for Jacobian of Hyperelliptic Curves of Small Genus

Koh-ichi Nagao, nagao@kanto-gakuin.ac.jp,

Dept. of Engineering, Kanto-Gakuin Univ.

Abstract. Gaudry present a variation of index calculus attack for solving the DLP in the Jacobian of hyperelliptic curves, which is a basic tool for solving elliptic curve cryptosystem using Weil descent. Gaudry and Harley improve this algorithm by the use of the restriction of smooth divisors. Later, Thérialut improve these kind of algorithm by the use of the one large prime. Here, we will present a variation of these kind of algorithm, which use two large primes (Thérialut use one large prime) and is faster than previous ones.
Keywords Index calculus attack, Jacobian, Hyperelliptic curve, DLP,

1 Introduction

Gaudry [3] first present a variation of index calculus attack for hyperelliptic curves that could solve the DLP on the Jacobian of an hyperelliptic curve of small genus by the use of smooth divisors. It requires $O(q)$ storage and at now, its implementation is difficult. However, in the near future, many computers linked by high speed network will appear and the possibility of this attack is believed to be realized. Gaudry and Harley(cf. [2]) improve this algorithm by the use of the restriction of smooth divisors. Later, Thérialut[1] improve these kind of algorithm by the use of the one large prime. In [1], these algorithms work in time $O(q^{2-\frac{2}{g+1}+\epsilon})$ and $O(q^{2-\frac{4}{2g+1}+\epsilon})$ respectively. Thériault's algorithm uses the almost-smooth divisor $D = \sum D(P_i)$ that all but one of the P_i's are in the set B called factor base. This technique was often used in the number field sieve factorization algorithm, which uses the almost smooth integer $n = \prod p_i$, that all but one of the p_i's are in the factor base B, which is the set of small primes. $P_i' s$ ($p_i' s$ in case of factorization), not including in B, are called large primes. In factorization algorithm, the technique using large prime (including the cases using 2 or 3 large primes) contribute speeding up factorization, but does not seems to contribute decreasing its complexity. The distribution of large prime is very one-sided. This is the reason that the complexity does not decrease. For the index calculus for the Jacobian of curves, we first compute the point of Jacobian and later consult whether it is almost smooth or not and the distribution of the large primes is unique. So that, the new algorithm that use the 2-almost smooth divisors, that all but 2 of the P_i's are in the set B, is useful and contribute to decrease its complexity. For my sense, similar algorithm using 3 large primes, only contribute speeding up to solve the discrete log, but does not contribute to decrease its complexity. For example, the almost smooth divisor of the form $v_1 = \sum$ terms of $B + D(P_1)$, and the 2-almost smooth divisors of the form $v_2 = \sum$ terms of $B + D(P_1) + D(P_2)$, $v_3 = \sum$ terms of $B + D(P_2) + D(P_3)$ are given, $v_1 - v_2 = \sum$ terms of $B - D(P_2)$,

$v_1 - v_2 + v_3 = \sum$ terms of $B + D(P_3)$ are other almost smooth divisors. So, we can get much more almost smooth divisors from gathering 2-almost smooth divisors. From this improvement, we get an attack of a running time of $O(q^{2-\frac{2}{g}+\epsilon})$. In case of small genus case, this improvement is especially useful.

genus	Thériault algorithm	Our Algorithm
3	$O(q^{10/7})$	$O(q^{4/3})$
4	$O(q^{14/9})$	$O(q^{3/2})$

For example, in the case of genus being 3 and q being a 60-bit prime power, theoretically, about $1600 \doteq 2^{60(10/7-4/3)}$ times improvement is expected.

2 Jacobian arithmetic

Let C be a hyperelliptic curve of genus g over $\mathbb{F}_q$ of the form $y^2 + h(x)y = f(x)$ with $\deg f = 2g + 1$ and $\deg h \le g$. Further, Use the notation J_q for $\mathbf{Jac}_C(\mathbb{F}_q)$.
Further, we will assume that $|J_q|$ is odd prime number, for simplicity.

Definition 1 *Given D_1,$D_2 \in J_q$ such that $D_2 \in < D_1 >$, DLP for (D_1, D_2) on J_q is computing λ such that $D_2 = \lambda D_1$.*

For an element $P = (x, y)$ in $C(\bar{\mathbb{F}}_q)$, put $-P := (x, -h(x) - y)$.

Lemma 1 *$C(\mathbb{F}_q)$ is written by the union of disjoint sets $\mathcal{P} \cup -\mathcal{P} \cup \{\infty\}$, where $-\mathcal{P} := \{-P | P \in \mathcal{P}\}$.*

Proof. Since $|J_q|$ is odd prime, we have $2 \not| |J_q|$ and there are no point $P \in C(\mathbb{F}_q)$ such that $P = -P$.

Further, we will fix $\mathcal{P}$.
Point of $\mathbf{Jac}_C$ can be represented uniquely by the reduced divisor of the form

$$\sum_{i=1}^{k} n_i P_i - \sum_{i=1}^{k} n_i \infty, \quad P_i \in C(\bar{\mathbb{F}}_q), \quad P_i \ne -P_j \text{ for } i \ne j$$

with $n_i \ge 0$ and $\sum n_i \le g$.

Definition 2 *The reduced divisor of a point of Jacobian J_q is written by the elements of $C(\mathbb{F}_q)$ i.e.*

$$\sum_{i=1}^{k} n_i P_i - \sum_{i=1}^{k} n_i \infty, \quad P_i \in C(\mathbb{F}_q).$$

Then the point is said to be potentially smooth point.

Let $D(P) := P - \infty$. Note that $P + (-P) \sim 2\infty$. From lemma 1, potentially smooth point v of J_q can be represented of the form

$$\sum_{P \in \mathcal{P}} n_P^{(v)} D(P)$$

with $n_P^{(v)} \in \mathbb{Z}$ and $\sum_{P \in \mathcal{P}} |n_P^{(v)}| \leq g$. Further, we will use this representation to potentially smooth points.

Definition 3 *A subset B of $\mathcal{P}$ used to define smoothness is called factor base.*

Definition 4 *A point $P \in \mathcal{P} \backslash B$ is called large prime.*

Definition 5 *A divisor v of the form*

$$\sum_{P \in B} n_P^{(v)} D(P)$$

is called smooth divisor.

Definition 6 *A divisor v of the form*

$$n_{P'}^{(v)} P' + \sum_{P \in B} n_P^{(v)} D(P),$$

where P' is a large prime, is called almost smooth divisor.

Definition 7 *A divisor v of the form*

$$n_{P'}^{(v)} P' + n_{P''}^{(v)} P'' + \sum_{P \in B} n_P^{(v)} D(P),$$

where P', P'' are large primes, is called 2-almost smooth divisor.

Definition 8 *An element $J \in J_q$ is called (resp. almost smooth, resp. 2-almost smooth) point, if the reduced divisor representing J is smooth (resp. almost smooth, resp. 2-almost smooth) divisor.*

Further, we will consider the coefficients n_P of a smooth (resp. almost smooth, resp. 2-almost smooth) divisor modulo $|J_q|$. For a smooth (resp. almost smooth, resp. 2-almost smooth) divisor v, put

$$l(v) := \#\{P \in B | n_P^{(v)} \neq 0\}.$$

Lemma 2 *Let v_1, v_2 be smooth (resp. almost smooth, resp. 2-almost smooth) divisors and let r_1, r_2 be integers modulo $|J_q|$. Then the cost for computing $r_1 v_1 + r_2 v_2$ is $O(g^2 (\log q)^2 (l(v_1) + l(v_2)))$.*

Proof. It requires $l(v_1) + l(v_2)$-time products and additions modulo $|J_q|$. Note that $|J_q| \doteq q^g$. Since the cost of one elementary operation modulo $|J_q|$ is $\log |J_q| = (g \log q)^2$, we have this estimation.

3 Outline of algorithm

In this section, we present the outline of the proposed algorithm. Let k be a real number satisfying $0 < k < 1/2g$. Further in this paper, we will use k as a parameter of this algorithm. Put

$$r := r(k) = \frac{g-1+k}{g}.$$

We will fix a set of factor base B with $|B| = q^r$.

Lemma 3

$$2r > 1+k > 1 > \frac{1+r}{2} = \frac{2g+k-1}{2g} > \frac{(g-1)+(g+1)k}{g}.$$

Proof. trivial.

The whole algorithm consists of the following 7 parts.

Algorithm 1 Whole Algorithm

```
Input: $C/\mathbb{F}_q$ hyper elliptic curve of small genus $g$, $D_1, D_2 \in J_q$ such that $D_2 \in < D_1 >$ .
Output: Integer $\lambda$ modulo $|J_q|$ such that $D_2 = \lambda D_1$.
1: Part 1 Computing all points of $C(\mathbb{F}_q)$ and making $\mathcal{P}$ and fix $B \subset \mathcal{P}$ with $|B| = q^r$.
2: Part 2 Gathering 2-almost smooth divisors and almost smooth divisors
   Computing a set $V_2$ of 2-almost smooth points and a set $V_1$ of almost smooth points
   of $J_q$, of the form $\alpha D_1 + \beta D_2$ with $|V_1| > q^{\frac{(g-1)+(g+1)k}{g}}$ and $|V_2| > q^{1+k}$.
3: Part 3 Computing a set of almost smooth divisor $H_m$ with $|H_m| > q^{(1+r)/2}$.
4: Part 4 Computing a set of smooth divisor $H$ with $|H| > q^r$.
5: Part 5 Solving linear algebra of the size $q^r \times q^r$
   Computing integers $\{r_h\}_{h \in H}$ modulo $|J_q|$, satisfying $\sum_{h \in H} r_h h \equiv 0 \bmod |J_q|$.
6: Part 6 Computing integers $\{s_v\}_{v \in V_1 \cup V_2}$ modulo $|J_q|$, satisfying $\sum_{v \in V_1 \cup V_2} s_v v \equiv 0 \bmod |J_q|$.
7: Part 7 Computing $\lambda$.
```

4 Gathering 2-almost smooth points and almost smooth points

Lemma 4 *The probability that a point in J_q is almost smooth is*

$$\frac{1}{(g-1)!} q^{(-1+r)(g-1)}$$

and the probability that a point is 2-almost smooth is

$$\frac{1}{2(g-2)!} q^{(-1+r)(g-2)}.$$

Algorithm 2 Gathering the 2-almost smooth pts and almost smooth pts

```
Input: $C/\mathbb{F}_q$ curve of genus $g$, $D_1, D_2 \in \mathbf{Jac}_C(\mathbb{F}_q)$
Output: $V_1$ a set of almost smooth divisors, $V_2$ a set of 2-almost smooth divisors
    such that $|V_2| > q^{1+k}$, $|V_1| > q^{\frac{(g-1)+(g+1)k}{g}}$, Integers $\{(\alpha_v, \beta_v)\}_{v \in V_1 \cup V_2}$ such that
    $v = \alpha_v D_1 + \beta_v D_2$
 1: $V_1 \leftarrow \{\}$, $V_2 \leftarrow \{\}$
 2: repeat
 3:     Let $\alpha, \beta$ be random numbers modulo $|J_q|$
 4:     Compute $v = \alpha D_1 + \beta D_2$
 5:     if $v$ is almost smooth then
 6:         $V_1 \leftarrow V_1 \cup \{v\}$
 7:         $(\alpha_v, \beta_v) \leftarrow (\alpha, \beta)$
 8:     end if
 9:     if $v$ is 2-almost smooth then
10:         $V_2 \leftarrow V_2 \cup \{v\}$
11:         $(\alpha_v, \beta_v) \leftarrow (\alpha, \beta)$
12:     end if
13: until $|V_2| > q^{1+k}$ and $|V_1| > q^{\frac{(g-1)+(g+1)k}{g}}$
14: return $V_1$,$V_2$,$\{(\alpha_v, \beta_v)\}_{v \in V_1 \cup V_2}$
```

Proof. We can get above lemma similarly from proposition 3,4,5 in [1]. For example, the probability of 2-almost smooth points is roughly estimated by

$$\frac{(2|B|)^{g-2}\,(2|\mathcal{P}\backslash B|)^2}{2!(g-2)!} \div |J_q| \doteq \frac{(q^r)^{g-2}\,q^2}{2!(g-2)!q^g} = \frac{1}{2(g-2)!}q^{(-1+r)(g-2)}.$$

From this lemma, the number of the loops that $|V_2| > q^{1+k}$ is estimated by

$$q^{(1+k)} \cdot 2(g-2)!q^{(1-r)(g-2)} = 2(g-2)!q^{2r},$$

and the number of the loops that $|V_1| > q^{\frac{(g-1)+(g+1)k}{g}}$ is estimated by

$$q^{\frac{(g-1)+(g+1)k}{g}} \cdot (g-1)!q^{(1-r)(g-1)} = (g-1)!q^{2r}.$$

Since the cost of computing Jacobian $v = \alpha D_1 + \beta D_2$ is $O(g^2(\log q)^2)$ and the cost of judging whether v is potentially smooth or not is $O(g^2(\log q)^3)$, the total cost of this part is estimated by

$$O(g^2(g-1)!(\log q)^3 q^{2r}).$$

Here, we will estimate the required storage. Note that the bit-length of one relative smooth point is $2g\,\log q$. So, the storage for V_1, the set of almost smooth divisors, is $O(g\,q^{\frac{(g-1)+(g+1)k}{g}}\,\log q)$ and the storage for V_2, the set of 2-almost smooth divisors, is $O(g\,q^{(1+k)}\,\log q)$. From lemma 3, we have $g\,q^{(1+k)}\,\log q >> g\,q^{\frac{(g-1)+(g+1)k}{g}}\,\log q$. So the total required storage can be estimated by

$$O(g\,q^{(1+k)}\,\log q).$$

5 Elimination of large prime (Framework)

Let E be a set of almost smooth divisors, and let F be a set of 2-almost smooth divisors or a set of almost smooth divisors. Note that element $e \in E$ and $f \in F$ are written by

$$e = n_{P_1}^{(e)} P_1 + \sum_{P \in B} n_P^{(e)} P,$$

$$f = n_{P_2}^{(f)} P_2 (+ n_{P_3}^{(f)} P_3) + \sum_{P \in B} n_P^{(f)} P.$$

Put $\mathrm{sup}(e) := \{P_1\}$ and $\mathrm{sup}(f) := \{P_2, (P_3)\}$. When $P \in \mathrm{sup}(e) \cap \mathrm{sup}(f)$, put

$$\phi(e, f, P) := n_p^{(f)} e - n_p^{(e)} f.$$

Trivially, $\phi(e, f, P)$ is almost smooth divisor, if F is a set of 2-almost smooth divisors and $\phi(e, f, P)$ is smooth divisor, if F is a set of almost smooth divisors and e is not of the form constant times f.
Let E be a set of almost smooth divisors and F be a a set of 2-almost smooth divisors. So, we construct the set of almost smooth divisors E' and a set of 2-almost smooth divisors F'. Roughly speaking, E' is a maximal subset of

$$\cup \phi(e, f, P), \quad (e \in E, f \in F,\ P \in \mathbf{sup}(e) \cap \mathbf{sup}(f) \text{ s.t. } e \neq \text{Const} \times f),$$

satisfying that E' does not contains both of the elements of the form $\phi(e_1, f, P_1)$ and $\phi(e_2, f, P_2)$, and

$$F' = F \setminus \cup f,$$

where $f \in F$ satisfies $\mathbf{sup}(e) \cap \mathbf{sup}(f) \neq \emptyset$ for some $e \in E$. In our aim(which is needed to Lemma 9 in order to get rid of the possibility of trivial relations), E' must not contains both of the elements of the form $\phi(e_1, f, P_1)$ and $\phi(e_2, f, P_2)$. E' is constructed as a set that such elements are omitted.

Definition 9 *Further, put*

$$E \odot F := (E', F'), \quad (E \odot F)[1] := E', \quad (E \odot F)[2] := F'.$$

We will estimate the size of $E \odot F$.

Lemma 5 *Let E be a set of randomly choosen almost smooth divisors and F be a set of randomly choosen 2-almost smooth divisors. Assume $|E| << q < |F|$. The size of $(E \odot F)[1]$ is estimated by*

$$|(E \odot F)[1]| \doteq \frac{2|E||F|}{|\mathcal{P} \setminus B|} \doteq \frac{4|E||F|}{q}.$$

Further, $|(E \odot F)[2]| = |F| \setminus |(E \odot F)[1]|$.

Proof. Let $e \in E$, $f \in F$ be randomly chosen elements. Put $P := \mathrm{sup}(e)$. Since F is a set of 2-almost smooth divisors, the probability that $P \in \mathrm{sup}(f)$ is $\frac{2}{|\mathcal{P} \setminus B|} \doteq \frac{4}{q}$ and the size is estimated by $\frac{2}{|\mathcal{P} \setminus B|}|E||F| = \frac{4}{q} \times |E||F|$. 2nd formula is trivial.

Algorithm 3 Elimination of Large primes

```
Input: $E$ almost smooth divisors, $F$ 2-almost smooth divisors
Output: $E'$ almost smooth divisors, $F'$ 2-almostsmooth divisors
 1: set $\mathcal{P}\backslash B = \{R_1, R_2, .., R_{|\mathcal{P}\backslash B|}\}$ (pre-computation)
 2: for $i = 1, 2, .., |\mathcal{P}\backslash B|$ do
 3:   $st[i] \leftarrow \{\}$
 4: od
 5: for all $e \in E$ do
 6:   $P = \sup(e)$
 7:   Compute $i$ s.t. $P = R_i$
 8:   $st[i] \leftarrow st[i] \cup \{e\}$
 9: od
10: $E' \leftarrow \{\}$, $F' \longleftarrow F$
11: for all $f \in F$ do
12:   for all $P \in \sup(f)$ do
13:     Compute $i$ s.t. $P = R_i$
14:     if $st[i] \neq \emptyset$ then
15:       for all $e \in st[i]$ s.t. $e \neq Const \times f$ do
16:         $E' \leftarrow E' \cup \{\phi(e, f, P)\}$, $F' \leftarrow F' \setminus \{f\}$
17:         break
18:         break (return to the loop of next $f \in F$)
19:       od
20:     end if
21:   od
22: od
23: return $E', F'$
```

We will estimate the cost and the storage for computing $E \odot F$.

Lemma 6 *Put* $c_1 := \max\{l(e)|e \in E\}$ *and* $c_2 := \max\{l(f)|f \in F\}$. *Assume that* $|E| << q$. *Then the cost of computing* $E \odot F$ *is*

$$O(c_1(\log q)^2|E|) + O((\log q)^2|F|) + O((c_1 + c_2)(\log q)^2|E||F|/q)$$

. and the required storage is

$$O(c_1 \log q|E|) + O((c_1 + c_2) \log q|E||F|/q).$$

Proof. The required storage for $st[i]$ is $O(c_1 \log q\,|E|)$ and the required storage for E' is $O((c_1 + c_2) \log q\,|E||F|/q)$, since $|E'| \doteq |E||F|/q$ and $\max\{l(v)|v \in E'\} = c_1 + c_2$ from lemma 2.
Note that the cost of the routine "Computing index i" is $\log q \log |\mathcal{P}\backslash B| = O((\log q)^2)$. Also note that $|E \odot F| = O(|E||F|/q)$ and remark that the probability of $st[i] \neq \emptyset$ is very small, since$|E| << q$. Thus, we see that the cost of the 1st loop is $O(c_1(\log q)^2|E|)$, the cost of the part "Computing index i" of the 2nd loop is $O((\log q)^2|F|)$, and the cost of the part "Computing the elements of E' and F'" of the 2nd loop is $O((c_1 + c_2)(\log q)^2|E||F|/q)$ from lemma 2.

Now, let E be a set of almost smooth divisors. We will construct E' a set of smooth divisors from E. Roughly speaking, E' is a maximal subset of

$$\cup\phi(e_1, e_2, P)$$

where $e_1, e_2 \in E$, $e_1 \neq$ Const $\times e_2$, and $P = \mathbf{sup}(e_1) \cap \mathbf{sup}(e_2)$ such that it does not contains any 2 elements of the form $\phi(e, f_1, P_1)$, $\phi(e, f_2, P_2)$, $\phi(f_3, e, P_3)$ and $\phi(f_4, e, P_4)$. Note that if $e_1, e_2 \in E$ are used once, e_1, e_2 are never used to the construction of E'.

Algorithm 4 Elimination of Large primes

```
Input: $E$ almost smooth divisors
Output: $E'$ smooth divisors
 1: set $\mathcal{P}\backslash B = \{R_1, R_2, .., R_{|\mathcal{P}\backslash B|}\}$ (pre-computation)
 2: for $i = 1, 2, .., |\mathcal{P}\backslash B|$ do
 3:    $st[i] \leftarrow \{\}$
 4: od
 5: for all $e \in E$ do
 6:    $P = \sup(e)$
 7:    Compute $i$ s.t. $P = R_i$
 8:    $st[i] \leftarrow st[i] \cup \{e\}$
 9: od
10: $E' \leftarrow \{\}$
11: for all $f \in E$ do
12:    P:=sup(f)
13:    Compute $i$ s.t. $P = R_i$
14:    if $st[i] \neq \emptyset$ then
15:       for all $e \in st[i]$ s.t. $e \neq Const \times f$ do
16:          $E' \leftarrow E' \cup \{\phi(e, f, P)\}$, $st[i] \leftarrow st[i] \setminus \{e, f\}$
17:          break (return to the loop of next $f \in E$)
18:       od
19:    end if
20:    od
21: od
22: return $E'$
```

Definition 10 *Further, put*

$$(E \odot E)[1] := E'.$$

Lemma 7 *Let E be a set of randomly choosen almost smooth divisors. Assume $|E| << q$. The size of $(E \odot E)[1]$ is estimated by*

$$|(E \odot E)[1]| \doteq \frac{|E|^2}{2|\mathcal{P} \setminus B|} \doteq \frac{|E|^2}{q}.$$

Further, put $c_1 := \max\{l(e)|e \in E\}$, then the cost of computing $(E \odot E)[1]$ is

$$O(c_1 (\log q)^2 |E|) + O(c_1 (\log q)^2 |E|^2/q),$$

and the required storage is

$$O(c_1 \log q |E|) + O(c_1 \log q |E|^2/q).$$

Proof. Let $e_1, e_2 \in E$ be randomly chosen elements. Put $P := \sup(e_1)$. The probability that $P \in \sup(e_2)$ is $\frac{1}{|\mathcal{P}\backslash B|} \doteq \frac{2}{q}$ and the size is estimated by $\binom{|E|}{2} \times \text{prob.} = \frac{1}{2|\mathcal{P}\setminus B|}|E|^2 = \frac{1}{q} \times |E|^2$. Cost estimations are similarly done by the previous case.

6 Computing H_m

In this section, we will construct H_m a set of almost smooth divisors $|H_m| > q^{(1+r)/2}$.

Algorithm 5 Computing H_m

```
Input: $V_1$ a set of almost smooth divisors s.t. $|V_1| > q^{\frac{(g-1)+(g+1)k}{g}}$, $V_2$ a set of 2-almost
    smooth divisors s.t. $|V_2| > q^{(1+k)}$
Output: Integer $m > 0$ and $H_1$, $H_2$,...,$H_m$ sets of almost smooth divisors s.t. $|H_m| >$
    $q^{(1+r)/2}$
1: $H_1 \leftarrow V_1$, $V_{2,1} \leftarrow V_2$
2: $i \leftarrow 1$
3: repeat
4:    $i++$
5:    $(H_i, V_{2,i}) \leftarrow H_{i-1} \odot V_{2,i-1}$,
6: until $|H_i| > q^{(1+r)/2}$
7: $m \leftarrow i$
8: return $m$,$H_1$, $H_2$, ..., $H_m$
```

Since H_i and $V_{2,i}$ are not randomly choosen sets. By this mean, Lemma 5 can not be used for the estimation of $|H_i|$ and $|V_{2,i}|$. However, in the appendix, we will show that the size estimation of Lemma 5 is valid by a computer experiment. So, we will use this size estimation of $|H_i|$ and $|V_{2,i}|$ as a heuristics. From lemma 5, the size of H_i is estimated by

$$|H_i| \doteq |H_1| \times (q^k)^{i-1} = q^{\frac{(g-1)+(g\,i+1)k}{g}}.$$

So, solving the equation $\frac{(g-1)+(g\,i+1)k}{g} = (1+r(k))/2$ for i, we have the following.

Lemma 8 *m is estimated by*

$$\frac{1-k}{2gk}.$$

Further, we will assume $m = O(\frac{1}{gk})$. Note that $\{l(v)|v \in \cup_{i\leq m}H_i\} \leq mg$. From lemma 6, the cost for computing H_m is

$$m \times (O((\log q)^2 q^{(1+k)}) + O(mg(\log q)^2 q^{(1+r)/2})))$$

and the required storage is

$$O(mg\, q^{(1+r)/2}\, \log q).$$

7 Computing H

In this section, we compute H a set of smooth divisors for $|H| > q^r$.

Algorithm 6 Computing H

```
Input: $H_1$,$H_2$,...,$H_m$ sets of almost smooth divisors s.t. $|H_m| > q^{(1+r)/2}$
Output: $H$ a set of smooth divisors s.t. $|H| > q^r$.
1: Put $H' := \cup_{i=1}^m H_i$
2: $H \leftarrow (H' \odot H')[1]$
3: return $H$
```

From this construction, note that $|H'| > q^{(1+r)/2}$. Since H' is not a randomly choosen set. By this mean, the size estimation using Lemma 5 is invalid. However, by a computer experiment in appendix, this estimation turns to be valid and we use this estimation as heuristics. From lemma 7, the size of H is estimated by

$$|H| = |H'|^2/q \geq q^r.$$

Note that $\{l(v)|v \in \cup_{i\leq m} H\} \leq 2mg$. From lemma 7, the cost for computing H is

$$O((\log q)^2 q^{(1+r)/2}) + O(mg(\log q)^2 q^r)$$

and the required storage is

$$O(mg \log q \, q^{(1+r)/2}).$$

8 Two ways representation of $h \in H$

An element $h \in H$ is written by the form

$$h = \sum_{P \in B} a_P^{(h)} D(P),$$

since it is a smooth divisor. Moreover, from its construction, we see easily that

$$l(h) = \#\{P \in B \mid a_P^{(h)} \neq 0\} \leq 2mg.$$

Set $B = \{R_1, R_2, ..., R_{|B|}\}$.

Definition 11

$$\textit{Put} \qquad \mathbf{vec}(h) := (a_{R_1}^{(h)}, a_{R_2}^{(h)}, ..., a_{R_{|B|}}^{(h)}).$$

The computation of $h(= \mathbf{vec}(h))$ means the set of pairs $\{(a_{R_i}^{(h)}, R_i)\}$ for non-zero $a_{R_i}^{(h)}$. Note that the required storage for one h is $O(m\,g\,\log q)$. On the other hands, from its construction, h is written by linear sum of at most $2m$ elements of $V_1 \cup V_2$. i.e.

$$h = \sum_{v \in V_1 \cup V_2} b_v^{(h)} v, \qquad \#\{v \mid b_v^{(h)} \neq 0\} \leq 2m.$$

Definition 12

$$\textit{Put} \qquad \mathbf{v}(h) := \{(b_v^{(h)}, v) \mid b_v^{(h)} \neq 0\}.$$

Note that the required storage for one $\mathbf{v}(h)$ is $O(m \log q)$.
Remark By little modifying the algorithm, we can obtain both representations of h of the forms $\mathbf{vec}(h)$ and $\mathbf{v}(h)$. (The order of the cost and the order of the storage for computing H is essentially the same.)
Further, we will assume that the computations of $\mathbf{vec}(h)$ and $\mathbf{v}(h)$ for $h \in H$ are done.

9 Linear algebra

In this section, we will solve the linear algebra and finding a linear relation of H.

Algorithm 7 Linear algebra

```
Input: $H$ a set of smooth divisors such that $|H| > q^r$
Output: Integers $\{\gamma_h\}_{h \in H}$ modulo $|J_q|$ s.t. $\sum_{h \in H} \gamma_h h \equiv 0 \bmod |J_q|$
1: Set $H = \{h_1, h_2, ..., h_{|H|}\}$
2: Set matrix $M = ({}^t\mathbf{vec}(h_1), {}^t\mathbf{vec}(h_2), ..., {}^t\mathbf{vec}(h_{|H|}))$
3: Solve linear algebra of $M$ and compute $(\gamma_1, \gamma_2, ..., \gamma_{|H|})$ such that $\sum_{i=1}^{|H|} \gamma_i \mathbf{vec}(h_i) = \mathbf{0}$
4: return $\{\gamma_i\}$
```

Note that the elements of matrix is integers modulo $|J_q| \doteq q^g$. So the cost of an elementary operation modulo J_q is $O(g^2 (\log q)^2)$.
M is a sparse matrix of the size $q^r \times q^r$. Note that the number of non-zero elements in one column is $2mg$. So, using [4] [5], we can compute $\{ \gamma_i\}$. Its cost is

$$O(g^2 (\log q)^2 \cdot 2mg \cdot q^r q^r) = O(mg^3 (\log q)^2 q^{2r})$$

and the required storage is

$$O(\log(q^g)\, m\, g \cdot q^r) = O(m\, g^2\, q^r\, \log q).$$

(The required storage for sparse linear algebra is essentially the storage for non-zero data. Note that the bit length of integer modulo $|J_q|$ is $\log(q^g)$, the number of nonzero elements of one row is mg.)

10 Computing s_v

Remember that each element $h \in H$ is of the form $h = \sum_{v \in V_1 \cup V_2} b_v^{(h)} v$. In the previous section, we found $\{\gamma_h\}$ such that $\sum_{h \in H} \gamma_h h \equiv 0 \bmod |J_q|$. So, put

$$s_v := \sum_{h \in H} \gamma_h b_v^{(h)} \bmod |J_q| \qquad \text{for all } v \in V_1 \cup V_2$$

and we have
$$\sum_{v \in V_1 \cup V_2} s_v v \equiv 0 \bmod |J_q|.$$

Algorithm 8 Computing s_v

```
Input: $V_1$,$V_2$,$H$,$\{\gamma_h\}_{h \in H}$ s.t. $\sum_{h \in H} \gamma_h h \equiv 0$
Output: $\{s_v\}_{v \in V_1 \cup V_2}$
1: for all $v \in V_1 \cup V_2$ do
2:     $s_v \leftarrow 0$
3: od
4: for all $h \in H$ do
5:     for all $v \in V_1 \cup V_2$ s.t $b_v^{(h)} \neq 0$ do
6:         $s_v \leftarrow s_v + \gamma_h b_v^{(h)}$
7:     od
8: od
9: return $\{s_v\}$
```

The cost of this part is
$$O(g\, q^{1+k} \log q) + O(m\ g^2\ (\log q)^2 q^{(1+r)/2})$$
and the storage is
$$O(g\ q^{1+k} \log q).$$
Let $h \in H$ such that $r_h \neq 0$, and $v \in \mathbf{v_2}(h)$.

Lemma 9 *$\{s_v\}_{v \in V_1 \cup V_2}$ contains at least one non-zero element.*

Proof. Put $\mathbf{v_2}(h) := \{v \in V_2 \,|\, b_v^{(h)} \neq 0\}$ for $h \in H$, and $G := \{h \in H \,|\, r_h \neq 0\}$. So, we easily have $s_v = \sum_{h \in G} r_h b_v^{(h)}$. $h \in H$ is written by the form $\phi(h_1, h_2, P)$ where $h_1 \in H_{i_1}$ and $h_2 \in H_{i_2}$. So, put $d(h) := \max(i_1, i_2)$. Take h' be an element of G whose $d = d(h')$ is maximal. Then there are some $v' \in V_{2,d-1} \setminus V_{2,d}$ such that $b_{v'}^{h'} \neq 0$. Note that v' is first used to construct H_d and v' is not used to construct any elements $h \in G \setminus \{h'\}$. So, for any $h \in G \setminus \{h'\}$, $v' \notin \mathbf{v_2}(h)$. Then we have $s_{v'} = \sum_{h \in G} r_h b_v^{(h)} = r_{h'} b_{v'}^{(h')} \neq 0$.

11 Finding discrete log

In the previous section, we found $\{s_v\}$ such that $\sum s_v v \equiv 0 \bmod |J_q|$. In the part 2 of the algorithm, we have computed (α_v, β_v) such that
$$v = \alpha_v D_1 + \beta_v D_2.$$
So, we have
$$\sum_{v \in V_1 \cup V_2} s_v(\alpha_v D_1 + \beta_v D_2) = (\sum_{v \in V_1 \cup V_2} s_v \alpha_v) D_1 + (\sum_{v \in V_1 \cup V_2} s_v \beta_v) D_2 \equiv 0. \bmod |J_q|$$

So, $-(\sum_{v \in V_1 \cup V_2} s_v \alpha_v)/(\sum_{v \in V_1 \cup V_2} s_v \beta_v) \bmod |J_q|$ is required discrete log.
Since $\{s_v\}$ contains non-zero elements (Lemma 9), the probability $\sum_{v \in V_1 \cup V_2} s_v \beta_v) = 0 \bmod |J_q|$ is $1/q$ and can be omitted.

Algorithm 9 Computing λ

```
Input: $V_1$,$V_2$,$\{\alpha_v, \beta_v\}$, $\{s_v\}$
Output: Integer $\lambda \bmod |J_q|$ s.t. $D_1 = \lambda D_2$
 1: return $-(\sum_{v \in V_1 \cup V_2} s_v \alpha_v)/(\sum_{v \in V_1 \cup V_2} s_v \beta_v) \bmod |J_q|$
```

Note that the cost of this part is $O(g^2\ q^{1+k}\ (\log q)^2)$.

12 Cost estimation

In this section, we will estimate the cost and the required storage of whole algorithm under the assumption of

$$k = \frac{1}{\log q}.$$

First, remember that $m = O(\frac{1}{gk}) = O(\frac{\log q}{g})$. By a direct computation, we have

$$r = r(k) = \frac{g - 1 + k}{g} = 1 - \frac{1}{g} + \frac{1}{g \log q},$$

and

$$q^{2r} = q^{2-\frac{2}{g}} \times \exp(\frac{2}{g}) = O(q^{2-\frac{2}{g}}).$$

From our cost estimation, the cost of the routine except part 2 and part 5 is written by the form

$$O(g^a\ (\log q)^b\ q^c) \quad a, b \leq 4,\ c \leq 1 + k.$$

On the other hands, the cost of the routine part 2 and part 5 is written by

$$O(g^2 (g-1)! (\log q)^3 q^{2r}) \text{ and } O(mg^3\ (\log q)^2 q^{2r}).$$

From lemma 3, we see $1 + k < 2r$ and the cost of the whole parts can be estimated by

$$O(g^2 (g-1)! (\log q)^3\ q^{2r}) = O(g^2 (g-1)! (\log q)^3 q^{2-\frac{2}{g}}).$$

Similarly, we see that the required storage (dominant part is part 2 and part 7, since $1 + k > 1 > (1 + r)/2$ from lemma 3) is

$$O(g\, q^{1+k}\ \log q) = O(g\, q^{1+k}\ \log q) = O(g\, q\ \exp(1)\ \log q) = O(g\, q\ \log q).$$

13 Conclusion

In ASIACRYPT2003, Thériault presented a variant of index calculus for the Jacobian of hyperelliptic curve of small genus, using almost smooth divisors. Here, we improve Thériault's result, using 2-almost divisors and propose an attack for DLP of the Jacobian of hyperelliptic curves of small genus, which works $O(q^{2-\frac{2}{g}+\epsilon})$ running time.

14 Appendix

In the previous version of this paper is rejected for some conference because of a negative comment of some referee, which says *" The result itself is not surprising, given Theriault's approach and analysis. What is surprising is the rather limited combinations that are created among the 2 almost smooth divisors: as far as I can tell longer chains of 2 almost smooth divisors are not created. This makes the analysis much easier. But I imagine that in actual implementation a huge speedup can be obtained by approaching the combinations as was done in QS and NFS. The approach also begs the question what number of large primes is optimal, a question that is not raised in the present paper".* Algorithm for using so called large prime is applied to factorization. In the factorization case, this technique contribute to speeding up of computation, but does not contribute to decreasing its complexity. The distribution of large primes being very one-sided may be the reason that its complexity does not decreasing (longer chains of 2-almost smooth divisors are not created.). In our case, the distribution of large primes is trivially unique and the situation is different from factorization case. Further, we will show that longer chains of 2-almost smooth divisors are created and that the size estimation of Lemma 5 is valid by a computer experiment.
For simplicity, we will do the computer simulation as follows.
1. Using the set $\{0, 1, 2, ..., Q-1\}$ in stead of the set of large primes $\mathcal{P} \setminus B$.
2. Using the randomly choosen N_1 elements $\{v1[i] \,|\, 0 \leq i < N_1\}$ of $\{0, 1, 2, ..., Q-1\}$ in stead of $V_1(= H_1)$.
3. Using the pairs of randomly choosen N elements $\{(v21[i], v22[i]) \,|\, 0 \leq i < N\}$ of $\{0, 1, 2, ..., Q-1\}$ in stead of $V_2(= V_{2,1})$.
So, starting from parameter $N = 30000000$, $Q = 25000000$, $N1 = 3000$, we have the following estimation. So, we confirm that longer chains of 2-almost smooth divisors are created and the size estimation of lemma 5 is valid. Starting from another parameter, we can also confirm this validity.

Simulation Program

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
/* N  maximam data size, number of V2 (ie. 2LP) */
#define N (30000000)
/* Q  number of LP */
#define Q (25000000)
/* N1 number of V1 (ie. 1LP) */
#define N1 3000

int cmp(const void *a, const void *b){
int aa,bb;
aa=*(int *)a;
bb=*(int *)b;
  if(aa>bb) return 1; else if (aa<bb) return -1; else return 0;
  }

void arraycopy(int *h1, int *h2,int n, int m){
int i;
for(i=0;i<n;i++) h1[i+m]=h2[i];
```

```
return;
}

main(){
int *v21,*v22,*h,*htmp,*v1,*hd,*p;
int i,j,n,n1,nd,nold,n1old,ndold,m;
printf("size of int %ld \n",sizeof(int));
if (sizeof(int)<4) {printf("sorry size of int must be >=4");exit(0);}
v1=(int *)malloc(N1*sizeof(int));  //  V1={ v1[i] | i=0..N1-1 }
v21=(int *)malloc(N*sizeof(int));  //  V2={(v21[i],v22[i])| i=0..N-1}
v22=(int *)malloc(N*sizeof(int));
h=(int *)malloc(N*sizeof(int));   // for the stack for Hj
htmp=(int *)malloc(N*sizeof(int)); // temporary stack cp from h
hd=(int *)malloc(N*sizeof(int));  // for the stack for \cup Hj

srand(87945); // for debug; use  srand((unsigned)time(NULL)); */
printf("Check  RAND_MAX=%ld > Q=%ld \n",RAND_MAX,Q);

// make data of V2={(v21[i],v22[i])| i=0..N-1}
for(i=0;i<N;i++){v21[i]=rand()%Q; v22[i]=rand()%Q;}

// make data of V1={ v1[i] | i=0..N1-1 }
for(i=0;i<N1;i++){ v1[i]=rand()%Q;}

n=N; n1=N1; nd=N1; // n=|Hj|, n1=|V_{2,j}|, nd=\sum_{i=1}^j |Hi|
arraycopy(h,v1,n1,0);
printf("j=1 |Hj|=%ld |V2j|=%ld \n", n1,n);
for(j=2;j<=10;j++){
    nold=n; n1old=n1; ndold=nd;
    qsort(h,n1,sizeof(int),cmp);
    arraycopy(htmp,h,n1,0);
    m=-1;
    for(i=0;i<N;i++){
      if(v22[i]==Q || v21[i]==Q) continue;
      p=(int *)bsearch(&(v21[i]),htmp,n1,sizeof(int),cmp);
      if(p!=NULL){ m++; h[m]=v22[i]; v22[i]=v21[i]=Q;continue;}
      p=(int *)bsearch(&(v22[i]),htmp,n1,sizeof(int),cmp);
      if(p!=NULL){ m++; h[m]=v21[i]; v22[i]=v21[i]=Q;}
      }
   n1=m; nd+=m; n-=m;
   arraycopy(hd,h,m,ndold);
   printf("j=%ld |Hj|=%ld (expected value)=%lf ", j,n1,((double)nold/Q*2*n1old));
   printf("|V2j|=%ld sum_{k=1}^j Hk=%ld \n", n, nd);
   if (nd>Q/80) break; // we have surfficient smooth elements
   }
qsort(hd,nd,sizeof(int),cmp);
m=0;
for(i=0;i<=nd-2;i++){
   j=0;
   while(hd[i]==hd[i+1] && i<=nd-2){i++;j++;}
   m=m+(j+1)/2;
   }
// Note that if nd/n is large (for example >0.05),
// |H| must be smaller than expected value from construction.
printf("|H|=%ld (expected value)=%lf \n",m,((double)nd*nd/2/Q) );
free(v1);free(v21); free(v22); free(h); free(htmp);free(hd);
```

}

Estimation of $|H_j|$

j	$\|H_j\|$	expcted value of $\|H_j\|$	$\|V_{2,j}\|$	$\| \cup_{j=1}^{k} H_k\|$
1	3000	———-	30000000	3000
2	7037	7200.000000	29992963	10037
3	16982	16884.838450	29975981	27019
4	40874	40724.168747	29935107	67893
5	98037	97885.405081	29837070	165930
6	233289	234010.946527	29603781	399219

Estimation of $|H|$
$|H| = 3362$, expected value of $|H| = 3187.516199$.

References

1. N. Thériault, Index calculus attack for hyperelliptic curves of small genus, ASIACRYPT2003, LNCS 2894, Springer-Verlag, 2003, pp. 75–92.
2. A. Enge, P. Gaudry, A general framework for subexponential discrete logarithm algorithms, *Acta Arith.,* **102**, no. 1, pp. 83–103,2002.
3. P.Gaudry, An algorithm for solving the discrete log problem on hyperelliptic curves, *Eurocrypt 2000*, LNCS 1807, Springer-Verlag, 2000, pp. 19–34.
4. B. A. LaMacchia, A. M. Odlyzko, Solving large sparse linear systems over finite fields, *Crypto '90*, LNCS 537, Springer-Verlag, 1990, pp. 109–133.
5. D. H. Wiedemann, Solving sparse linear equations over finite fields, *IEEE Trans. Inform. Theory*, **IT-32**, no.1, pp.54–62, 1986.

Postscript: Gaudry, and Thomé, [1] propose the similar method using graph theory independently.

Diem [2] propose the index culculas to the Jacobian of general plane curves.

[1] A double large prime variation for small genus hyperelliptic index calculus, preprint, `http://eprint.iacr.org/2004/153/` ,2004.

[2] Index Calculus in Class Groups of Plane Curves of Small Degree, preprint, `http://eprint.iacr.org/2005/119` ,2005.