Index Calculus Attack for Hyperelliptic Curves of Small Genus

Nicolas Thériault

University of Toronto

Abstract. We present a variation of the index calculus attack by Gaudry which can be used to solve the discrete logarithm problem in the Jacobian of hyperelliptic curves. The new algorithm has a running time which is better than the original index calculus attack and the Rho method (and other square-root algorithms) for curves of genus ≥ 3. We also describe another improvement for curves of genus ≥ 4 (slightly slower, but less dependent on memory space) initially mentioned by Harley and used in a number of papers, but never analyzed in details.

1 Introduction

Koblitz [10] first introduced the use of hyperelliptic curves for discrete log based public-key cryptography in 1989. For the first ten years, the best known generic attacks against these cryptosystems were the "square-root" algorithms (Shank's Baby Step-Giant Step, Pollard's ρ method, Pollard's λ method). Pierrick Gaudry's index calculus attack for hyperelliptic curves [8] was the first example of a generic attack that could solve the discrete log problem on the Jacobian of an hyperelliptic curve of small genus over a finite field in a time smaller than the square-root of the group order (assuming the genus of the curve is greater than 4) (an attack for curves of high genus was introduced the year before in [1] by Adleman, DeMarrais and Huang).

In this paper, we analyse in detail a variation of the original index calculus attack which was first introduced by Robert Harley and implemented for a number of papers, but never analyzed in detail. This algorithm works in time $O\left(g^5 q^{2-\frac{2}{g+1}+\epsilon}\right)$ and gives an improvement on previous attacks for curves of genus greater than 3. We also describe how the algorithm can be improved further by using the large prime method of the number field sieve. For this variation, we get a running time of $O\left(g^5 q^{2-\frac{4}{2g+1}+\epsilon}\right)$ and an improvement for all curves of genus greater than 2. Comparing the running times for curves of genus 3, 4 and 5, we get

g	3	4	5
square-root algorithms	$q^{3/2}$	q^2	$q^{5/2}$
original index calculus	q^2	q^2	q^2
reduced factor base	$q^{3/2}$	$q^{8/5}$	$q^{5/3}$
with large primes	$q^{10/7}$	$q^{14/9}$	$q^{18/11}$

This paper is divided as follows: The main ideas and concepts used in the index calculus attack are described in Sect. 2. We then present the two algorithms in Sects. 3 and 4. The running times of both algorithms are analyzed together in Sect. 5 and the memory space required to run the algorithms is discussed in Sect. 6.

2 The Index Calculus Attack

2.1 The Discrete Log Problem

Let C be an imaginary quadratic curve over $\mathbb{F}_q$, i.e. a smooth hyperelliptic curve of genus g over $\mathbb{F}_q$ with a single point at infinity and whose finite part can be written in the form $y^2 + h(x)y = f(x)$ with $\deg(f) = 2g+1$ and $\deg(h) \leq g$.

Note 1. Throughout this paper, we will use J_q for $Jac(C)(\mathbb{F}_q)$.

Definition 1. *Given D_1, D_2, two elements of J_q such that $D_2 \in \langle D_1 \rangle$, the* hyperelliptic discrete log problem *for the pair (D_1, D_2) on J_q consists in computing the smallest integer $\lambda \in \mathbb{N}$ such that $D_2 = \lambda D_1$.*

In practice, we can assume that D_1 has large prime order in J_q (if not, we can bring the problem down to subgroups of $\langle D_1 \rangle$ of prime order, solving the corresponding discrete log problem on each subgroup independently).

2.2 Jacobian Arithmetic

Note 2. Throughout the paper, we will assume only basic arithmetic for multiplication. In practice, faster algorithms (Karatsuba, FFT) should be used, but they will reduce the overall running time by a factor of less than $g\log(q)$.

Points of $Jac(C)$ can be represented uniquely by reduced divisors, i.e. divisors of the form

$$\sum_{i=1}^{k} P_i - k\infty$$

where the P_i's are points in $C(\overline{\mathbb{F}_q})$ with $P_i \neq -P_j$ for $i \neq j$ and with $k \leq g$ and ∞ is the unique point at infinity of C. ¿From now onward, we identify $Jac(C)$ with the collection of reduced divisors.

We use the following result of Cantor [2]:

Proposition 1. *For every reduced divisor $D = \sum_{i=1}^{k} P_i - k\infty$ (with $P_i = (x_i, y_i)$), there is a unique representation by a pair of polynomials $[a(x), b(x)]$, $a(x), b(x) \in \overline{\mathbb{F}}_q[x]$, with*

$$a(x) = \prod_{i=1}^{k} (x - x_i)$$

and $b(x_i) = y_i$ satisfying $\deg(b) < \deg(a) \leq g$ and $b(x)^2 + h(x)b(x) - f(x)$ divisible by $a(x)$. The sum (as a reduced divisor) of two reduced divisors in J_q can be computed in $O(g^2(\log(q))^2)$ bit operations.

The reduced divisor $D = [a(x), b(x)]$ is associated to a point in J_q if and only if both $a(x)$ and $b(x)$ are in $\mathbb{F}_q[x]$.

2.3 Smooth Divisors

Let $\mathcal{P}$ be the collection of $\mathbb{F}_q$-rational points of C, i.e. $\mathcal{P} = C(\mathbb{F}_q)$. For every $P \in C(\overline{\mathbb{F}_q})$, we let $D(P) = P - \infty$.

Definition 2. *Let B be a subset of $\mathcal{P}$. A divisor D is said to be* smooth relative to B *if it is reduced and $D = \sum_{i=1}^{k} D(P_i)$ with all the P_i's in B.*

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

Definition 4. *A divisor will be said to be* potentially smooth *if it is smooth relative to $\mathcal{P}$.*

Definition 5. *A point P in $\mathcal{P}$ will be called a* large prime *relative to a factor base B if $P \notin B$.*

Definition 6. *A reduced divisor $D = \sum_{i=1}^{k} D(P_i)$ will be said to be* almost-smooth *if all but one of the P_i's are in B and the remaining P_i is a large prime.*

2.4 Random Walk

The index calculus algorithm relies in a large part on using a pseudo-random walk to search for smooth divisors. We set up a pseudo-random walk by specifying a hash hunction $\mathcal{H}$ and a state function $\mathcal{R}$. A hash function $\mathcal{H}$ is a function $\mathcal{H} : J_q \to \{1, 2, \ldots, n\}$. A state function is a map $\mathcal{R} : J_q \times \{1, 2, \ldots, n\} \to J_q$. Given an initial point $T_0 \in J_q$, our interest is in computing the sequence (the "random walk") (T_i) with $T_{i+1} = \mathcal{R}(T_i, \mathcal{H}(T_i))$.

To have an effective index calculus attack for the discrete log problem for a given pair $(D_1, D_2) \in J_q \times J_q$, the pair $(\mathcal{R}, \mathcal{H})$ should be chosen to satisfy certain statistical and computational constraints. The function $\mathcal{R}$ should be chosen so that given $T_i = \alpha_i D_1 + \beta_i D_2$, it is easy to compute T_{i+1} as well as α_{i+1} and β_{i+1} such that $T_{i+1} = \alpha_{i+1} D_1 + \beta_{i+1} D_2$. A simple method is to set

$$\mathcal{R}(T, j) = T + T^{(j)}$$

where $T^{(j)} = \alpha^{(j)} D_1 + \beta^{(j)} D_2$ for some randomly chosen $\alpha^{(j)}$ and $\beta^{(j)}$.

At each step of the random walk, we compute T_{i+1} as well as α_{i+1} and β_{i+1} modulo the order of J_q. The values of T_i, α_i and β_i need to be recorded only if T_i is a smooth divisor (or an almost-smooth divisor in the second algorithm).

2.5 Index Calculus

¿From the sequence (T_i) of divisors obtained in the random walk, we extract a subsequence of smooth divisors (R_i). Then each R_i can be written both as $R_i = \alpha_i D_1 + \beta_i D_2$ and $R_i = \sum_{j=1}^{k_i} D(P_j)$ with the P_j's in the factor base and $k_i \leq g$. The goal of the index caculus attack is to use the R_i's to obtain an equation of the form $\alpha D_1 + \beta D_2 = 0$.

To do this, we order the elements of B as $P_1, P_2, \ldots P_{|B|}$. To each smooth divisor

$$T_i = \sum_{j=1}^{k_i} D(P_{i,j}) = \sum_{l=1}^{|B|} a_{i,l} D(P_l)$$

we can associate a vector

$$\mathbf{v_i} = (\ a_{i,1}\ ,\ a_{i,2}\ , \ldots ,\ a_{i,|B|}\ )$$

We then use the vectors $\mathbf{v_i}$ to build the matrix $M = (a_{i,j})_{i,j}$ where each row, corresponding to a smooth divisor, has weight at most g. When the size of M is large enough (i.e. when M is overdetermined), we use linear algebra to find a nonzero vector in the kernel of M. Note that all operations are done modulo $|J_q|$. Once a nonzero solution of the system is found, we can write

$$\sum_{i=0}^{m} \gamma_i \mathbf{v_i} = \mathbf{0}$$

and (in terms of divisors)

$$\sum_{i=0}^{m} \gamma_i R_i = 0.$$

Substituting $R_i = \alpha_i D_1 + \beta_i D_2$, we get

$$\left(\sum_{i=0}^{m} \gamma_i \alpha_i \right) D_1 + \left(\sum_{i=0}^{m} \gamma_i \beta_i \right) D_2 = \alpha D_1 + \beta D_2 = 0,$$

from which we obtain the solution $D_2 = \lambda D_1$ ($\lambda = -\alpha/\beta$). The algorithm fails only if $\beta = 0$, in which case we must go through the algorithm again starting from the initialization of the random walk. This is very unlikely however (the algorithm fails with probability $|\langle D_1 \rangle|^{-1}$ if D_1 has prime order), hence we expect to have to go through the algorithm only once.

In practice, once a point $P_i \neq -P_i$ is included in the factor base, we take $-P_i$ as being in the factor base but replace $D(-P_i)$ by $-D(P_i)$ in the construction of the linear algebra system (since the divisor $D(P_i) + D(-P_i)$ reduces to 0). This makes it possible to reduce the number of smooth divisors we must find in the random walk by a factor of close to 2.

3 First Algorithm

3.1 Factor Base

In the original version of the index calculus attack for hyperelliptic curve, the factor base is $\mathcal{P} = C(\mathbb{F}_q)$. This gives a running time of

$$O\left(g^2 g! q (\log(q))^2\right) + O\left(g^3 q^2 (\log(q))^2\right)$$

where the first part is due to the search for smooth divisors, while the second part is the cost of solving the linear algebra system.

If q is large enough relative to g, i.e. if $q > (g-1)!$, then most of the cost of the index calculus attack comes from the linear algebra. The first approach to reduce the overall running time consists in reducing the size of the factor base, which reduces the time required to solve the linear algebra system on the one hand, but increases the search time on the other hand (since reducing the size of the factor base also reduces the number of smooth divisors). We do this until both parts of the running time are equal, i.e. up to the point where any further reduction of the factor base would make the search too costly.

Given that $q > (g-1)!$, the factor base can be chosen as a subset B of $\mathcal{P}$ such that the running time becomes

$$O\left(g^5 q^{2-\frac{2}{g+1}+\epsilon}\right).$$

For the analysis, we assume that $q > (g-1)!$ and we set $|B| = q^r$, with $\frac{2}{3} < r < 1$ and compute the value of r which gives the best running time.

3.2 Algorithm

The first algorithm can be summarized as follows:

1. Search for the elements of the factor base
   Compute the x and y coordinates of points in $C(\mathbb{F}_q)$ until $|B| = q^r$.
2. Initialization of the random walk
   Choose the $\alpha^{(j)}$'s and $\beta^{(j)}$'s randomly and compute the $T^{(j)}$'s. Also choose α_0 and β_0 randomly and compute $T_0 = \alpha_0 D_1 + \beta_0 D_2$.
3. Search for smooth divisors (random walk)
   The following steps are repeated until the linear system is large enough:
   a) Search for potentially smooth divisors
      Compute $T_{i+1} = [a(x), b(x)]$ and check if $a(x)$ splits over $\mathbb{F}_q$.
   b) Factorization of the potentially smooth divisors
      If $a(x)$ splits over $\mathbb{F}_q$, compute the points in $C(\mathbb{F}_q)$ corresponding to T_{i+1}. T_{i+1} is smooth if and only if all the points are in B.
   c) Construction of the linear algebra system
      Compute α_{i+1} and β_{i+1}. If T_{i+1} is smooth, record α_{i+1}, β_{i+1} and the factors of T_{i+1}.

4. Solution of the linear algebra system
   Compute a nonzero vector in the kernel of the matrix obtained at step 3.
5. Final solution
   Compute λ (if $\beta = 0$, return to step 2).

Note that in step 3, the factorization of $a(x)$ is done in two parts: we first check if $a(x)$ splits over $\mathbb{F}_q$ by breaking down $a(x)$ into squarefree factors and checking that the factors divide $x^q - x$. If $a(x)$ splits in $\mathbb{F}_q$, we can then completely factor $a(x)$ using Cantor-Zassenhaus. The second part, which is probabilistic, is obviously skipped if $a(x)$ does not split over $\mathbb{F}_q$ (in that case, the divisor is not potentially smooth and obviously cannot be smooth).

4 Second Algorithm

The new improvement to the index calculus mimics the use of large primes in the number field sieve. We again reduce the size of the factor base as much as possible to reduce the time required to solve the linear algebra system without making the search for smooth divisors too costly. This time however, we make use of the points in $\mathcal{P}$ which are not part of the factor base.

If $q > (g-1)!/g$, we can play the almost-smooth divisors against each other to cancel the large primes to bring the running time down to

$$O\left(g^5 q^{2-\frac{4}{2g+1}+\epsilon}\right).$$

For the analysis, we once again assume that $q > (g-1)!/g$ and that the factor base has size $|B| = q^r$ with $\frac{2}{3} < r < 1$.

4.1 Large Primes

To make use of the almost-smooth divisors, we consider them in the order in which they appear during the search.

Definition 7. *Let T_i be an almost-smooth divisor with the large prime P. T_i will be called an* intersection *if one of the previous T_j $(j < i)$ has large prime $\pm P$.*

If two almost-smooth divisors T_1, T_2 have large prime P, i.e. if they can be written in the form

$$T_1 = D(P) + \sum_{i=1}^{k_1-1} D(P_{1,i}) \quad \text{and} \quad T_2 = D(P) + \sum_{i=1}^{k_2-1} D(P_{2,i})$$

with $P_{1,i}, P_{2,i} \in B$, we consider

$$T_1 - T_2 = \sum_{i=1}^{k_1-1} D(P_{1,i}) - \sum_{i=1}^{k_2-1} D(P_{2,i})$$

and set $T' = T_1 - T_2$ (after doing all the extra cancellations that may be necessary if $P_{1,i} = P_{2,j}$ for some pair i, j).

If T_1, T_2 are almost-smooth divisors such that T_1 has large prime P and T_2 has large prime $-P$, i.e. if they can be written in the form

$$T_1 = D(P) + \sum_{i=1}^{k_1-1} D(P_{1,i}) \quad \text{and} \quad T_2 = D(-P) + \sum_{i=1}^{k_2-1} D(P_{2,i})$$

with $P_{1,i}, P_{2,i} \in B$, we consider

$$T_1 + T_2 = \sum_{i=1}^{k_1-1} D(P_{1,i}) + \sum_{i=1}^{k_2-1} D(P_{2,i})$$

and set $T' = T_1 + T_2$ (after doing all the extra cancellations that may be necessary if $P_{1,i} = -P_{2,j}$ for some pair i, j).

In both cases, T' factors over the factor base even though it may not be smooth (T' need not be reduced). For the linear algebra, the vector associated with T' will work in exactly the same way as if it was the difference of two smooth divisors with a common $P_i \in B$ and will have weight $< 2g$.

Proposition 2. *Each intersection is counted only once no matter how many times the large prime (or its negative) appeared before.*

Proof. Let P be a large prime. Suppose that $k > 1$ almost-smooth divisors with large prime P or $-P$ occurred during the random walk, say $T_{j_1}, T_{j_2}, \ldots, T_{j_k}$, $k-1$ of which are intersections. Using the same idea as described in Sect. 2.5, we associate the T_{j_i}'s to vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ with an extra coordinate for $D(P)$. In order to use these to add information to the linear algebra system, we must cancel out the coordinate associated to $D(P)$. If we use $\mathbf{v}_1$ to do the cancellation in the other $\mathbf{v}_i$'s, we obtain $k-1$ vectors $\mathbf{v}'_2, \ldots, \mathbf{v}'_k$ which are then used to construct M (after removing the coordinate associated to $D(P)$). Since

$$\mathit{span}\,\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\} = \mathit{span}\,\{\mathbf{v}_1, \mathbf{v}'_2, \ldots, \mathbf{v}'_k\},$$

once T_{j_1} has been used to cancel the large prime in T_{j_i}, using another T_{j_l} to do the cancellation again does not produce any supplementary information for the linaer algebra system. Q.E.D.

We therefore look for intersections of almost-smooth divisors and use these to obtain extra equations in our linear algebra system.

The advantage of this method is that the number of almost-smooth divisors is greater than the number of smooth divisors by a factor of $O(gq^{1-r})$ and the search should produce more intersections of almost-smooth divisors than smooth divisors.

For the analysis, we will assume that any point P_i in $\mathcal{P}$ such that $P_i = -P_i$ is in the factor base and that a point is in the factor base if and only if its negative is also in the factor base. This has no efffect on the running time, but it simplifies the analysis (in particular for Theorem 1).

4.2 Algorithm

The second algorithm can be summarized has follows (all steps, except 3c, work in the same way as in the first algorithm).

1. Search for the elements of the factor base
2. Initialization of the random walk
3. Search for smooth divisors (random walk)
   a) Search for potentially smooth divisors
   b) Factorization of the potentially smooth divisors
   c) Cancellation of the Large Primes
      If the divisor is almost-smooth, check whether or not it is an intersection. If not, add it to the list of non-intersections. If it is an intersection, cancel its large prime and use the result as if it were a smooth divisor.
   d) Construction of the linear algebra system
4. Solution of the linear algebra system
5. Final solution

5 Running-time analysis

5.1 Factor Base

In order to choose our factor base, we look at the x-coordinates of the points in $C(\mathbb{F}_q)$.

We go through the values of $x_i \in \mathbb{F}_q$ starting from 0 and following a chosen order on $\mathbb{F}_q$. We first evaluate $y^2 + h(X)y - f(X)$ at $X = x_i$ (this can be done in $O(g^2(\log(q))^2)$ bit operations). We then factor the quadratic polynomial in $\mathbb{F}_q[y]$ obtained which takes $O((\log(q))^2)$ bit operations. If the polynomial has roots $y_{i,1}, y_{i,2}$ in $\mathbb{F}_q$ (y_i if we have a double root), we include (x_i, y_{i_1}) and (x_i, y_{i_2}) in B. We then go on to the next $x_i \in \mathbb{F}_q$ until $|B| = q^r$.

This method will require $O(q)$ tries for the possible x-coordinates, each taking $O(g^2(\log(q))^2)$ bit operations, for a total of

$$O(g^2 q(\log(q))^2)$$

bit operations to build the factor base.

5.2 Initialization

To initialize the Random walk, we need to precompute the divisors $T^{(i)}$ used in the state function $\mathcal{R} : J_q \times \{0, 1, \ldots, n\} \rightarrow J_q$ as well as T_0.

For each $T^{(i)}$ (and for T_0), we choose both $\alpha^{(i)}$ and $\beta^{(i)}$ randomly in $\{1, 2, \ldots, (|J_q| - 1)\}$ and set $T^{(i)} = \alpha^{(i)} D_1 + \beta^{(i)} D_2$. We then need $O(g \log(q))$ Jacobian operations to compute each of the $T^{(i)}$'s, each Jacobian operation taking $O(g^2(\log(q))^2)$ bit operations. In practice, we can take $n = O(\log(|J_q|)) = O(g \log(q))$, which gives a total of

$$O(g^4 \log(q)^4)$$

bit operations to initialize the random walk.

5.3 Smooth Divisors

Proposition 3. *For $\frac{2}{3} < r < 1$, there are $\frac{q^{rg}}{g!} + O\left(\frac{g^2 q^{r(g-1)}}{g!}\right)$ smooth divisors in J_q.*

Proof. All smooth divisors relative to B can be written in the form $\sum_{i=1}^{k} D(P_i)$ with the P_i's in B and $k \leq g$. To count to number of smooth divisors, we need to consider the number of distinct P_i's in the representation of the divisors. The number of smooth divisors with g distinct P_i's is:

$$\frac{1}{g!}\prod_{i=0}^{g-1}(q^r - i) = \frac{q^{rg}}{g!} - \frac{q^{r(g-1)}}{2(g-2)!} + O\left(q^{r(g-2)}\right).$$

The number of smooth divisors with $g-1$ distinct P_i's, one of which is repeated is:

$$\frac{g-1}{(g-1)!}\prod_{i=0}^{g-2}(q^r - i) = \frac{q^{r(g-1)}}{(g-2)!} + O\left(q^{r(g-2)}\right).$$

The number of smooth divisors with $g-1$ distinct P_i's, none of which are repeated is:

$$\frac{1}{(g-1)!}\prod_{i=0}^{g-2}(q^r - i) = \frac{q^{r(g-1)}}{(g-1)!} + O\left(q^{r(g-2)}\right)$$

Finally, the number of smooth divisors with less than $g-1$ distinct P_i's is $O\left(q^{r(g-2)}\right)$. This gives a total of

$$\frac{q^{rg}}{g!} + O\left(\frac{g^2 q^{r(g-1)}}{g!}\right)$$

smooth divisors relative to B. Q.E.D.

The proportion of smooth divisors in J_q is then

$$\frac{\frac{q^{rg}}{g!} + O\left(\frac{g^2 q^{r(g-1)}}{g!}\right)}{q^g + O\left(g q^{g-\frac{1}{2}}\right)} = \frac{q^{-(1-r)g}}{g!} + O\left(\frac{g^2 q^{-(1-r)g-r}}{g!}\right) + O\left(\frac{g q^{-(1-r)g-\frac{1}{2}}}{g!}\right),$$

so we expect to have to look at

$$O\left(g! q^{(1-r)g}\right)$$

divsiors for each smooth divisor found in the search.

5.4 Potentially Smooth Divisors

Proposition 4. *For $\frac{2}{3} < r < 1$, there are $\frac{q^g}{g!} + O\left(\frac{g}{g!} q^{g-\frac{1}{2}}\right)$ potentially smooth divisors in J_q.*

Proof. All smooth divisors relative to $\mathcal{P}$ can be written in the form $\sum_{i=1}^{k} D(P_i)$ with the P_i's in $\mathcal{P}$ and $k \leq g$. To count to number of smooth divisors, we need to consider the number of distinct P_i's in the representation of the divisors. Since $|\mathcal{P}| = q + O\left(\sqrt{q}\right)$ (from Hasse's bound), the number of potentially smooth divisors with g distinct P_i's is:

$$\frac{1}{g!}\prod_{i=0}^{g-1}(|\mathcal{P}| - i) = \frac{q^g}{g!} + O\left(\frac{g}{g!}q^{g-\frac{1}{2}}\right).$$

The number of potentially smooth divisors with less than g distinct P_i's is $O\left(q^{g-1}\right)$, which gives a total of

$$\frac{q^g}{g!} + O\left(\frac{g}{g!}q^{g-\frac{1}{2}}\right)$$

potentially smooth divisors. Q.E.D.

The proportion of potentially smooth divisors in J_q is then

$$\frac{\frac{q^g}{g!} + O\left(\frac{gq^{g-\frac{1}{2}}}{g!}\right)}{q^g + O\left(gq^{g-\frac{1}{2}}\right)} = \frac{1}{g!} + O\left(\frac{g}{g!\sqrt{q}}\right)$$

and we expect to have a potentially smooth divisor for every $O(g!)$ divisors computed in the search.

5.5 Almost-Smooth Divisors

Proposition 5. *For $\frac{2}{3} < r < 1$, there are $\frac{q^{rg+1-r}}{(g-1)!} + O\left(\frac{q^{rg}}{(g-1)!}\right)$ almost-smooth divisors in J_q.*

Proof. Each almost-smooth divisor can be written in the form $D(P)+\sum_{i=1}^{k-1} D(P_i)$ with $P \in \mathcal{P} \setminus B$, the P_i's in B and $k \leq g$, so each almost-smooth divisor can be associated to a large prime and at most $g-1$ P_i's in B. Using an argument similar to the one in the proof of Proposition 3, we get

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

possible distinct choices for the P_i's in B. There are $|\mathcal{P}| - |B| = q - q^r + O(\sqrt{q})$ choices for the large prime, so we have

$$\frac{q^{rg+1-r}}{(g-1)!} - \frac{q^{rg}}{(g-1)!} + O\left(\frac{(g-1)q^{rg+1-2r}}{(g-2)!}\right) + O\left(\frac{q^{rg+\frac{1}{2}-r}}{(g-1)!}\right)$$

almost-smooth divisors relative to B. Since $\frac{2}{3} < r < 1$ and $q > g!$, we get

$$\frac{q^{rg+1-r}}{(g-1)!} + O\left(\frac{q^{rg}}{(g-1)!}\right).$$

Q.E.D.

The proportion of almost-smooth divisors in J_q is

$$\frac{\frac{q^{rg+1-r}}{(g-1)!} + O\left(\frac{q^{rg}}{(g-1)!}\right)}{q^g + O\left(gq^{g-\frac{1}{2}}\right)} = \frac{q^{-(1-r)(g-1)}}{(g-1)!} + O\left(\frac{q^{-(1-r)g}}{(g-1)!}\right) + O\left(g\frac{q^{-(1-r)(g-1)-\frac{1}{2}}}{(g-1)!}\right).$$

During the search, we can expect to look at

$$O\left((g-1)!q^{(1-r)(g-1)}\right)$$

divisors for each almost-smooth divisor found.

5.6 Intersections

We now consider the effect on the search of using almost-smooth divisors to get the equations required for the linear algebra more quickly.

In order to know how many equations can be obtained from the almost-smooth divisors, we need an estimate of the expected number of intersections out of a set of s almost-smooth divisors. For this, we consider only the large prime of each almost-smooth divisor.

Let $Q(n, s, i)$ be the probability of having i intersections out of a sample of size s drawn with replacement from a set of n elements and let $E_{n,s}$ be the expected number of intersections, i.e.

$$E_{n,s} = \sum_{i=0}^{s-1} iQ(n, s, i).$$

Theorem 1. *If* $3 \le s < n/2$*, then* $E_{n,s}$ *is between* $\frac{2s^2}{3n}$ *and* $\frac{s^2}{n}$*.*

Proof. If we consider the probability of having i intersections after $s+1$ draws, we have

$$Q(n, s+1, i) = \frac{n - 2(s-i)}{n} Q(n, s, i) + \frac{2(s-i+1)}{n} Q(n, s, i-1)$$

since if T_{s+1} contains the large prime P_{s+1}, then T_{s+1} is an intersection if and only if $\pm P_{s+1}$ appears in one of the $s-i$ or $s-i+1$ non-intersections in the first i almost-smooth divisors. Then

$$\begin{aligned}
E_{n,s+1} &= \sum_{i=0}^{s} iQ(n, s+1, i) \\
&= \sum_{i=0}^{s} i\left(\frac{n-2(s-i)}{n} Q(n, s, i) + \frac{2(s-i+1)}{n} Q(n, s, i-1)\right) \\
&= \sum_{i=0}^{s-1} i\frac{n-2(s-i)}{n} Q(n, s, i) + \sum_{i=1}^{s} i\frac{2(s-i+1)}{n} Q(n, s, i-1)
\end{aligned}$$

$$
\begin{aligned}
&= \frac{n-2s}{n}\sum_{i=0}^{s-1} iQ(n,s,i) + \frac{2}{n}\sum_{i=0}^{s-1} i^2 Q(n,s,i) + \frac{2s}{n}\sum_{i=1}^{s} Q(n,s,i-1) \\
&\quad + \frac{2s-2}{n}\sum_{i=1}^{s}(i-1)Q(n,s,i-1) - \frac{2}{n}\sum_{i=1}^{s}(i-1)^2 Q(n,s,i-1) \\
&= \frac{n-2}{n}\sum_{i=0}^{s-1} iQ(n,s,i) + \frac{2s}{n}\sum_{i=0}^{s-1} Q(n,s,i) \\
&= \frac{n-2}{n}E_{n,s} + \frac{2s}{n}.
\end{aligned}
$$

Solving for $E_{n,s}$ (using $E_{n,1} = 0$), we get

$$E_{n,s} = \frac{n}{2}\left(1-\frac{2}{n}\right)^s + s - \frac{n}{2} = \frac{n}{2}\sum_{i=2}^{s}\binom{s}{i}\left(\frac{-2}{n}\right)^i$$

Since $\frac{2(s-i)}{in} < 1$, the terms in the sum are strictly decreasing in absolute values, hence

$$E_{n,s} < \frac{s(s-1)}{n} < \frac{s^2}{n}$$

and

$$E_{n,s} > \frac{s(s-1)}{n} - \frac{2s(s-1)(s-2)}{3n^2} > \frac{s^2}{n} - \frac{2s^3}{3n^2} = \frac{s^2}{n}\left(1 - \frac{2}{3}\frac{s}{n}\right) > \frac{2}{3}\frac{s^2}{n}.$$

Q.E.D.

5.7 Search (First Algorithm)

In order to insure the existence of a nonzero vector in the kernel of the linear algebra system in step 4, we need to find $O(|B|) = O(q^r)$ smooth divisors. Since we expect to look at $O\left(g!q^{g(1-r)}\right)$ divisors for each smooth divisor found, the search will take an expected

$$O\left(g!q^{g(1-r)+r}\right)$$

random walk steps.

At each step of the random walk, we first have to compute T_i which requires $O(g^2(\log(q))^2)$ bit operations for the arithmetic in J_q. ¿From the representation of T_i as $[a(x), b(x)]$, we can test whether or not T_i is potentially smooth by checking if $a(x)$ factors into linear factors over $\mathbb{F}_q$, which can be done in $O(g^2\log(q)^2)$ bit operations. We must also compute α_i and β_i modulo $|J_q|$, which requires $O(g^2(\log(q))^2)$ bit operations. Since this must be done for all $O\left(g!q^{g(1-r)+r}\right)$ divisors generated, this gives

$$O\left(g^2 g! q^{g(1-r)+r}\log(q)^2\right)$$

bit operations.

We now consider the cost of completely factoring the potentially smooth divisors. Since there are $O(g!)$ divisors for each potentially smooth divisor, we expect to find $O\left(q^{g(1-r)+r}\right)$ potentially smooth divisors during the search. Since computing the points of $\mathcal{P}$ in the representation of a divisor $[a(x), b(x)]$ requires to completely factor $a(x)$ over $\mathbb{F}_q$ (to get the x-coordinates and multiplicities) and then evaluating $b(x)$ at the roots of $a(x)$ (to obtain the y-coordinates), which takes $O(g^2 \log(q)^2)$ bit operations (since $a(x)$ has degree $O(g)$), determining which potentially smooth divisors are really smooth and representing them in terms of the factor base takes

$$O\left(g^2 q^{g(1-r)+r} \log(q)^2\right)$$

bit operations.

The search is then expected to take

$$O\left(g^2 g! q^{g(1-r)+r} \log(q)^2\right)$$

bit operations for the first algorithm.

Note that it may be possible to reduce the number of divisors to consider for factorization by giving conditions on the coefficients of $a(x)$ for the divisor to be considered for smoothness. For exemple, if q is prime and the x-coordinates of the points in the factor base are between 0 and cq^r, then if the divisor is smooth, $a(x)$ must be of the form $x^k - a_{k-1}x^{k-1} + \ldots$ with $0 < a_{k-1} < kcq^r$. Even though this reduces the cost of testing for potentially smooth divisors and complete factorization, the arithmetic in J_q is unaffected, and so the effect on the running time will be at most a constant factor. This method will not work for the second algorithm since there are no restrictions on the x-coordinate of the large prime.

5.8 Search (Second Algorithm)

If we let n be the number of large primes (i.e. $n = q - q^r + O(\sqrt{q})$) and ask that $E_{n,s} = O(q^r)$ (i.e. so that we expect the search to yield enough intersections to build the linear algebra system), then we need

$$s = O\left(q^{\frac{r+1}{2}}\right).$$

It will then take

$$O\left(s(g-1)! q^{(g-1)(1-r)}\right) = O\left((g-1)! q^{(g-1)(1-r)+\frac{r+1}{2}}\right)$$

steps of random walk to build the linear algebra system.

Note that we expect the search to also produce

$$\frac{O\left((g-1)! q^{g-rg+r-1+\frac{r+1}{2}}\right)}{O\left(g! q^{g-rg}\right)} = O\left(\frac{1}{g} q^{r-\frac{1-r}{2}}\right)$$

smooth divisors, which are obviously used to get the linear algebra system but are not enough to have an important effect on the running time.

As in the first algorithm, computing $T_i = [a(x), b(x)]$, α_i and β_i and testing whether or not T_i is potentially smooth takes $O(g^2 \log(q)^2)$, for a total of

$$O\left(gg!q^{(g-1)(1-r)+\frac{r+1}{2}}(\log(q))^2\right)$$

bit operations over the whole search.

Since one in every $O(g!)$ divisors is potentially smooth, we expect to find $O\left(q^{(g-1)(1-r)+\frac{r+1}{2}}/g\right)$ potentially smooth divisors during the search. For each potentially smooth divisor, we compute the points in $\mathcal{P}$ in its representation (which takes $O(g^2 \log(q)^2)$ bit operations) and check if it is smooth or almost-smooth. If the divisor is smooth, it is used to produce the linear algebra system; if it is almost-smooth we look at the previous almost smooth divisors to see if it is an intersection, which takes $O(\frac{1+r}{2}\log(q))$ bit operations (there are $O(q^{\frac{1+r}{2}})$ non-intersections and only the large prime is considered doing this search). If we have an intersection, we cancel the large prime and use the resulting divisor to increase the size of the linear system, otherwise we add the divisor to the list of non-intersections. This process is expected to take

$$O\left(gq^{(g-1)(1-r)+\frac{r+1}{2}}(\log(q))^2\right)$$

bit operations for all the potentially smooth divisors encountered during the search.

The search is then expected to take

$$O\left(gg!q^{(g-1)(1-r)+\frac{r+1}{2}}(\log(q))^2\right)$$

bit operations for the second algorithm.

5.9 Linear Algebra

As said before, we continue with the search until we have an overdetermined system. This gives us a matrix M of size $O(q^r) \times O(q^r)$, hence there exists a nonzero vector in the kernel of M. Since each row has weight $O(g)$ ($\leq g$ for the first algorithm and $< 2g$ for the second), the system is sparse with weight $O(gq^r)$.

Since M is sparse, we can use the algorithms by Lanczos [11] and Wiedemann [13]. We can then find a vector in the kernel of this matrix in $O(gq^{2r})$ operations modulo $|J_q|$. Since $|J_q| = q^g + O(gq^{g-1/2})$, finding a solution will take

$$O\left(g^3 q^{2r}(\log(q))^2\right)$$

bit operations.

5.10 Final Solution

From the vector in the kernel of M, we have

$$\sum_i \gamma_i \mathbf{v_i} = \mathbf{0}.$$

We obtain the final solution by computing

$$\alpha = \sum_i \gamma_i \alpha_i \qquad \text{and} \qquad \beta = \sum_i \gamma_i \beta_i$$

modulo $|J_q|$, where α_i, β_i come from the representation as $T_i = \alpha_i D_1 + \beta_i D_2$ of the i^{th} divisor used to build the linear algebra system. If $\beta \neq 0$, the final solution of the discrete log problem for the pair (D_1, D_2) is

$$\lambda \equiv -\frac{\alpha}{\beta} \bmod |J_q|.$$

Computing α and β requires $O(q^r)$ operations modulo $|J_q|$, each of these operations taking $O(g^2(\log(q))^2)$ bit operations. This gives a total of

$$O(g^2 q^r (\log(q))^2)$$

bit operations for the final step.

5.11 Optimization (First Algorithm)

Theorem 2. *The factor base can be chosen such that the running time of the first algorithm becomes*

$$O\left(g^5 q^{2-\frac{2}{g+1}+\epsilon}\right).$$

Proof. From the previous sections, the steps of the first algorithm have the following running times:

1. $O\left(g^2 q (\log(q))^2\right)$
2. $O\left(g^4 (\log(q))^4\right)$
3. $O\left(g^2 g! q^{g-(g-1)r} (\log(q))^2\right)$
4. $O\left(g^3 q^{2r} (\log(q))^2\right)$
5. $O\left(g^2 q^r (\log(q))^2\right)$

Since the running times for parts 1, 2 and 5 are all much smaller than those for parts 3 and 4 when $\frac{2}{3} < r < 1$, the overall running time is:

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

In order to minimize this, we choose r such that both parts have the same asymptotic form, i.e. such that

$$(g-1)! q^{g-(g-1)r} = q^{2r}.$$

Solving for r, we get

$$r = \frac{g + \log_q((g-1)!)}{g+1},$$

and since r is indeed between $\frac{2}{3}$ and 1 for genus ≥ 3, this gives a running time of

$$O\left(g^3((g-1)!)^{\frac{2}{g+1}} q^{\frac{2g}{g+1}} (\log(q))^2\right).$$

Finally, since $(g/4)^{g+1} < (g-1)! < g^{g+1}$ for $g \geq 3$, this is

$$O\left(g^5 q^{2-\frac{2}{g+1}+\epsilon}\right).$$

Q.E.D.

5.12 Optimization (Second Algorithm)

Theorem 3. *The factor base can be chosen such that the running time of the second algorithm becomes*

$$O\left(g^5 q^{2-\frac{4}{2g+1}+\epsilon}\right).$$

Proof. For the second algorithm, the steps have running times:

1. $O\left(g^2 q (\log(q))^2\right)$
2. $O\left(g^4 (\log(q))^4\right)$
3. $O\left(g g! q^{(g-1)(1-r)+\frac{r+1}{2}} (\log(q))^2\right)$
4. $O\left(g^3 q^{2r} (\log(q))^2\right)$
5. $O\left(g^2 q^r (\log(q))^2\right)$

Once again, steps 3 and 4 are more costly than the others, so the overall running time is:

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

Forcing both parts to have the same asymptotic form requires

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

which gives

$$r = \frac{g - \frac{1}{2} + \log_q((g-1)!/g)}{g + \frac{1}{2}},$$

and since r is indeed between $\frac{2}{3}$ and 1 for genus ≥ 3, this gives a running time of

$$O\left(g^3((g-1)!/g)^{\frac{4}{2g+1}} q^{\frac{4g-2}{2g+1}} (\log(q))^2\right).$$

Finally, since $(g/4)^{g+\frac{1}{2}} < (g-1)!/g < g^{g+\frac{1}{2}}$ for $g \geq 3$, we get

$$O\left(g^5 q^{2-\frac{4}{2g+1}+\epsilon}\right).$$

Q.E.D.

6 Memory Space

For both algorithms, storing the linear algebra system requires $O\left(gq^r\log(q)\right)$ bits ($O(q^r)$ equations and for each equation, the factored divisor, α_i and β_i each take $O(g\log(q))$ bits). For the second algorithm, we must also store all the non-intersections almost-smooth divisors, which requires $O\left(gq^{\frac{1+r}{2}}\log(q)\right)$ bits (we expect to need $O(q^{\frac{1+r}{2}})$ almost-smooth divisors, each taking $O(g\log(q))$ bits to store the factorization, α_i and β_i), which will take more space than the linear algebra system.

Substituting r with the values found in the proofs of Theorems 2 and 3, we get $O\left(g^2q^{\frac{g}{g+1}+\epsilon}\right)$ bits for the first algorithm and $O\left(g^2q^{\frac{2g}{2g+1}+\epsilon}\right)$ bits for the second algorithm.

7 Conclusion

We have described two algorithms for the hyperelliptic curves discrete log problem which improve on previously published attacks. If we compare the running time of these two algorithms with those of the original index calculus and the various "square-root" algorithms (Baby Step-Giant Step, Pollard ρ, etc.), for small genus, we have:

g	3	4	5	6	7	8	9
square-root algorithms	$q^{3/2}$	q^2	$q^{5/2}$	q^3	$q^{7/2}$	q^4	$q^{9/2}$
original index calculus	q^2	q^2	q^2	q^2	q^2	q^2	q^2
reduced factor base	$q^{3/2}$	$q^{8/5}$	$q^{5/3}$	$q^{12/7}$	$q^{7/4}$	$q^{16/9}$	$q^{9/5}$
with large primes	$q^{10/7}$	$q^{14/9}$	$q^{18/11}$	$q^{22/13}$	$q^{26/15}$	$q^{30/17}$	$q^{34/19}$

Since the running times using large primes are slightly lower than previously published attacks, the large primes algorithm should be taken into account when designing any cryptosystem based on hyperelliptic curves of genus greater than 2. In particular, for curves of genus 3, the field of definition requires approximatively 5% more bits of memory space for the curves to give the same level of security as they did when the best known attacks were the square root algorithms (obviously, the cost of the group operation will also increase in consequence). The 5% increase is due to the ratio $\log(q')/\log(q) \approx 21/20$ required for the index calculus attack for a genus 3 curve defined over $\mathbb{F}_{q'}$ to require the same expected running time as Pollard's ρ algorithm for a genus 3 curve defined over the field $\mathbb{F}_q$.

Note that for genus 2 curves, Gaudry showed that the linear algebra system can be solved in linear time (see [7]). The best possible running time for the index calculus (using all the points over $\mathbb{F}_q$ as the factor base) is then $O(q)$, which is the same as Pollard's ρ method and the other square roots algorithms.

References

1. L. M. Adleman, J. DeMarrais, M.-D. Huang, A subexponential algorithm for discrete logarithms over hyperelliptic curves of large genus over GF(q), *Theoret. Comput. Sci.*, **226**, no. 1-2, pp. 7-18, 1999.
2. D. G. Cantor, Computing in the Jacobian of an hyperelliptic curve, *Math. Comp.*, **48**(177), pp. 95-101, 1987.
3. A. Enge, Computing discrete logarithms in high-genus hyperelliptic jacobians in provably subexponential time, *Math. Comp.*, **71**, no. 238, pp. 729-742, 2002.
4. A. Enge, P. Gaudry, A general framework for subexponential discrete logarithm algorithms, *Acta Arith.*, **102**, no. 1, pp. 83-103, 2002.
5. A. Enge, A. Stein, Smooth ideals in hyperelliptic function fields, *Math. Comp.*, **71**, no. 239, pp. 1219-1230, 2002.
6. T. Garefalakis, D. Panario, The index calculus method using non-smooth polynomials, *Math. Comp.*, **70**, no 235, pp. 1253-1264, 2001.
7. P. Gaudry, *Algorithmique des courbes hyperelliptiques et applications à la cryptologie*, Thèse de doctorat de l'École polytechnique, 2000
8. P. Gaudry, An algorithm for solving the discrete log problem on hyperelliptic curves, *Advances in cryptology - EUROCRYPT 2000*, Springer-Verlag, LNCS 1807, pp. 19-34, 2000.
9. M. Girault, R. Cohen, M. Campana, A generalized birthday attack, *Advances in Cryptology - EUROCRYPT '88*, Springer-Verlag, LNCS 330, pp. 129-156, 1988.
10. N. Koblitz, Hyperelliptic cryptosystems, *J. of Cryptology*, **1**, pp. 139-150, 1989.
11. B. A. LaMacchia, A. M. Odlyzko, Solving large sparse linear systems over finite fields, *Advances in Cryptology - CRYPTO '90*, Springer-Verlag, LNCS 537, pp. 109-133, 1990.
12. V. Müller, A. Stein, C. Thiel, Computing discrete logarithms in real quadratic congruence function fields of large genus, *Math. Comp.*, **68**, no. 226, pp. 807-822, 1999.
13. D. H. Wiedemann, Solving sparse linear equations over finite fields, *IEEE Trans. Inform. Theory*, **IT-32**, no. 1, pp.54-62, 1986.