A double large prime variation for small genus hyperelliptic index calculus

P. Gaudry, E. Thomé, N. Thériault and C. Diem

November 21, 2005

Abstract

In this article, we examine how the index calculus approach for computing discrete logarithms in small genus hyperelliptic curves can be improved by introducing a double large prime variation. Two algorithms are presented. The first algorithm is a rather natural adaptation of the double large prime variation to the intended context. On heuristic and experimental grounds, it seems to perform quite well but lacks a complete and precise analysis. Our second algorithm is a considerably simplified variant, which can be analyzed easily. The resulting complexity improves on the fastest known algorithms. Computer experiments show that for hyperelliptic curves of genus three, our first algorithm surpasses Pollard's Rho method even for rather small field sizes.

1 Introduction

The discrete logarithm problem in the jacobian group of a curve is known to be solvable in subexponential time if the genus is large compared to the base field size [1, 20, 8, 9, 14, 6]. The corresponding index calculus algorithm also works for small fixed genus, and although the running time becomes exponential it can still be better than Pollard's Rho algorithm [11]. Introducing a large prime variation [23], it is possible to obtain an index calculus algorithm that is asymptotically faster than Pollard's Rho algorithm already for genus 3 curves.

In the present work, we go one step further in this direction and introduce a double large prime variation for the small genus index calculus. Our algorithm is a simple extension to the single large prime algorithm of [23]. However, making a rigorous analysis is not that easy: Double large prime variations are commonly used in factorization algorithms and analyzed empirically. In order to obtain a proven complexity result, we introduce a simplified algorithm for the double large prime variation which lends itself much better to a rigorous complexity analysis. The analysis is made for fixed genus and growing field size. Our proof is valid for the restricted context of hyperelliptic curves in imaginary Weierstrass form with cyclic jacobian group, and the complexity result is stated as follows.

Theorem 1. *Let $g \geq 3$ be fixed. Let $\mathcal{C}$ be a hyperelliptic curve of genus g over $\mathbb{F}_q$ given by an imaginary Weierstrass equation, such that the jacobian group $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ is cyclic. Then the discrete logarithm problem in $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ can be solved in expected time*

$$\widetilde{O}\left(q^{2-\frac{2}{g}}\right)$$

as q tends to infinity.

The $\widetilde{O}$-notation captures logarithmic factors. This complexity improves on the previous best bound $\widetilde{O}(q^{2-\frac{2}{g+1/2}})$. The presented algorithm also applies to general curves of genus $g \geq 3$, not

2000 *Mathematics Subject Classification.* Primary 11Y16; Secondary 11T71, 94A60.

necessarily hyperelliptic and not necessarily with cyclic jacobian group (but provided that the Jacobian arithmetic can be performed in polynomial time). Heuristically, the complexity result still holds.

The improvement is negligible for curves of large genus and therefore the case of genus 3 curves is given special consideration. For genus 3 curves, Pollard's Rho method has a running time in $\widetilde{O}(q^{1.5})$, whereas the single large prime algorithm is in $\widetilde{O}(q^{1.428\cdots})$ and our new method is in $\widetilde{O}(q^{1.333\cdots})$. We did practical experiments that demonstrate that even when the jacobian group has relatively small size, our algorithm is much faster than Pollard's Rho algorithm. In these comparisons, we consider only curves whose jacobian group is of almost prime order, so that splitting the discrete problem in smaller problems in subgroups [22] is not possible. This is the case for instances that occur in the context of cryptography. Therefore, when designing a cryptosystem [16] based on a genus 3 curve, it is necessary to take into account our attack, and not only Pollard's Rho attack. The sizes of the parameters should then be enlarged by about 12.5% to maintain the security level.

The article is organized as follows: In Section 2, we fix the general setting and recall previous work. Our double large prime variation is introduced in Section 3, together with our simplified variant. This simplified variant is analyzed in Section 4. In Section 5, we describe our computer experiments that validate our approach and show that it outperforms Pollard's Rho method rather early. Section 6 explores the relationship between our "full" and "simplified" algorithms, as well as the relevance of our algorithm beyond the restricted context of hyperelliptic curves with cyclic jacobian group.

The order of the authors is chronological. The first two authors found the algorithm and gave a heuristic analysis. A complete proof was obtained by the first three authors. The fourth author then gave a much simpler proof, and the proof of Theorem 1 presented in this work follows the ideas of the fourth author.

Acknowledgements

The first three authors thank Antoine Lejay who helped with the probabilistic statements that occurred in the first proof of Theorem 1.

2 Setting and previous work

2.1 Setting

Let $\mathcal{C}$ be a hyperelliptic curve of genus $g \geq 3$ over a finite field $\mathbb{F}_q$ with q elements, given by an imaginary Weierstrass equation. The elements of the jacobian group $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ of $\mathcal{C}$ over $\mathbb{F}_q$ are handled via their Mumford representation [19]: a divisor class contains a unique reduced divisor that is represented by a pair of polynomials $\langle u(x), v(x)\rangle$. The degree of $u(x)$ is called the *weight* of the reduced divisor and a reduced divisor is called *prime* if $u(x)$ is irreducible.

A discrete logarithm problem in $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ is to be solved. Namely, we work in a cyclic subgroup G of $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$. A generating reduced divisor D_1 of G and another reduced divisor $D_2 \in G$ are given. The goal is to compute the integer λ in $[0, \#G-1]$ such that $D_2 = \lambda D_1$ in $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$. The group order is also supposed to be part of the input; in our case, since the genus is fixed, it can be computed in polynomial time [21].

The algorithms we are dealing with have a complexity which is exponential in $\log q$. Since any task that takes a time which is polynomial in $\log q$ is considered easy, we shall often use the $\widetilde{O}()$-notation for complexity estimates: A function in $\widetilde{O}(f(q))$ is a function that is bounded by $f(q)$ times a polynomial in $\log q$ for large enough q.

2.2 Basic index calculus

The general index calculus algorithm proceeds as follows: A factor base $\mathcal{B}$ that consists of prime divisors of low weight is formed. Then random linear combinations of D_1 and D_2 are computed using, for instance, Cantor's algorithm [3]. For each combination, one checks whether it can be written as a sum of elements of the factor base by factoring the u-polynomial of its Mumford representation. If this is the case, then we have a useful relation and the corresponding data is put in a row of a matrix. After enough relations have been found, there exists a non trivial combination of the rows that sums to zero; this is a simple linear algebra problem. Since each row represents a linear combination of D_1 and D_2, any combination of rows also represents a linear combination of D_1 and D_2 that can be computed. We can then find α and β such that $\alpha D_1 + \beta D_2 = 0$. This gives the solution to the discrete logarithm problem as long as β is invertible modulo $\#G$, which will be the case with large probability.

For (small) fixed genus, the prime divisors considered for the factor base are divisors of weight 1. This "basic index calculus" algorithm has therefore complexity $\widetilde{O}(q^2)$, split in $\widetilde{O}(q)$ for the relation search and $\widetilde{O}(q^2)$ for the linear algebra.

An optimized variant of the basic index calculus algorithm, due to Harley, consists in balancing the relation search and linear algebra steps by restricting the factor base size to q^r elements, with $0 < r < 1$. With the best value of $r = 1 - 1/(g+1)$, the complexity becomes $\widetilde{O}(q^{2-\frac{2}{g+1}})$.

We shall not give more details on the basic technique of index calculus and refer the reader to [23] for a complete description.

2.3 Single large prime variation

Extending the idea of the "balanced" index calculus approach, a single large prime variation has been presented in [23]. The factor base is again chosen with size q^r, with $0 < r < 1$. The $\Theta(q)$ reduced divisors of weight one which are outside the factor base are called "large primes".

The algorithm proceeds like any index calculus algorithm with a large prime variation. Random linear combinations of D_1 and D_2 are computed. Only combinations which involve at most one large prime in the sum of their prime reduced divisors are considered.

At the heart of the analysis is the birthday paradox which says that after having collected k relations involving large primes, they can be combined to form an expected number of $\frac{k^2}{2q}$ relations involving only elements of the factor base. Then, estimating the probability of getting a relation with one large prime and balancing everything with the linear algebra step, the optimal value for r is $1 - \frac{1}{g+1/2}$, and the overall complexity is $\widetilde{O}(q^{2-\frac{2}{g+1/2}})$.

From this, the following result is obtained in [23].

Theorem (Thériault)**.** *Let $g \geq 3$ be fixed. Let $\mathcal{C}$ be a hyperelliptic curve of genus g over $\mathbb{F}_q$ given by an imaginary Weierstrass equation, such that the jacobian group $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ is cyclic. Then the discrete logarithm problem in $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ can be solved in expected time*

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

as q tends to infinity.

3 A double large prime variation

We now present the context of our double large prime variant, which comes as the natural extension of the previous single large prime algorithm. As before, we define a factor base and a set of large primes, which are sets of reduced divisors of weight 1. In other words, they can be interpreted as rational points of the curve. Since computing the hyperelliptic involution can be done almost for free with our representation, we use it to reduce the cardinalities of these sets.

Definition 2. *Let r be a constant real number such that $0 < r < 1$.*

The **factor base** $\mathcal{B}$ *is a set of representatives of $\lfloor \frac{1}{2} q^r \rfloor$ arbitrary orbits of $\mathcal{C}(\mathbb{F}_q)$ under the hyperelliptic involution ι.*

The set of **large primes** $\mathcal{L}$ *is a set of representatives of the remaining orbits under ι.*

Due to Weil's theorem, $\#\mathcal{C}(\mathbb{F}_q) = q + O(\sqrt{q})$. Therefore it is possible to construct a suitable factor base $\mathcal{B}$, and we have $\#\mathcal{L} = \frac{q}{2} + O(\sqrt{q})$.

As before, we form random linear combinations of D_1 and D_2, and to each such combination R we apply the following procedure for the smoothness test.

- Compute the Mumford representation $\langle u(x), v(x) \rangle$ of R.
- Discard R if $u(x)$ has a non-linear irreducible factor. Otherwise, write $R = \sum_{i=1}^{r} n_i P_i$, where P_i is a reduced divisor of weight 1 and $n_i \geq 1$.
- For all i such that $P_i \notin \mathcal{B} \cup \mathcal{L}$, replace P_i with $\iota(P_i)$ and negate n_i.

At the end of this procedure, we have obtained an expression of the following form which we call a "relation".

$$\alpha D_1 + \beta D_2 = \sum_{i=1}^{r} n_i P_i. \tag{1}$$

where the equality holds in the jacobian group and the P_i are elements of the factor base or of the set of large primes (we say that the relation *involves* these factor base elements and large primes).

Definition 3. *A relation is said to be* **Full** *if it involves only elements of the factor base $\mathcal{B}$. A relation is said to be* **FP** *if it involves elements of $\mathcal{B}$ and exactly one large prime. A relation is said to be* **PP** *if it involves elements of $\mathcal{B}$ and exactly two large primes.*

Clearly, PP relations can be found much more quickly than FP relations, but the problem is to combine all these relations in order to obtain more Full relations. This can be done by looking for cycles in a graph where vertices are large primes and edges are relations involving them. For this purpose, an adaptation of the *union-find* algorithm is used, making it possible to solve this question in time almost linear in the number of PP relations found.

This *relation collection* terminates when as many as $\#\mathcal{B} + 1$ Full or recombined relations have been obtained. Afterwards, the algorithm proceeds with the linear algebra step as in the classical index calculus situation described in Section 2.2.

3.1 Description of the LP-graph and its evolution

Double large prime variations of all kinds use a *graph of large prime relations.* Within the context of an index calculus algorithm, the relations involve multiplicities, so squares cannot be canceled out as is done in the classical case of integer factorization. For this reason, the description of the graph of large prime relations is more technical.

The **graph of large prime relations** (LP-graph, for short) is an undirected acyclic graph with $1 + \#\mathcal{L}$ vertices, corresponding to the elements of $\mathcal{L}$ and the special vertex 1. All edges of the LP-graph are labeled with a relation.

At the beginning of the algorithm there are no edges in the LP-graph, and a counter C is set to zero. The algorithm stops when C reaches the prescribed value $C_{\max} = \#\mathcal{B} + 1$. Recall that $\#\mathcal{B} \ll \#\mathcal{L}$. The counter C must first be regarded as the number of independent cycles that would appear in the LP-graph in the course of its evolution even though no cycle is actually created.

We start our relation search. Each time we find a relation R, the LP-graph is modified according to the following procedure.

- If R is Full, the LP-graph is unchanged and the counter C is incremented.

- If R involves two large primes or less, we consider a new edge E, labeled by R, for potential inclusion into the LP-graph. If R is FP, the vertices of E are 1 and p_1 (the large prime appearing in R), while if R is PP, the vertices of E are the two large primes p_1 and p_2 appearing in R.

  We consider the following exclusive cases:

  - If adding E would not create any cycle, E is added to the LP-graph.
  - If adding E would create a cycle Γ, we are led to a technical distinction. Let $k = \#\Gamma$ be the number of edges that form Γ, $V(\Gamma)$ their vertices, and $R(\Gamma)$ their attached relations. $V(\Gamma)$ has cardinality k, and depending on whether $1 \in V(\Gamma)$ or not, the relations in $R(\Gamma)$ involve $k-1$ or k large primes, respectively. By linear algebra, we can obtain a linear combination of the relations in $R(\Gamma)$ which has the contribution of at least $k-1$ large primes canceled. Hence:
    * If $1 \in V(\Gamma)$, a Full relation can be obtained. C is increased, and the LP-graph is unchanged (note that a Full relation may also be obtained in lucky cases even when $1 \notin V(\Gamma)$; this "luck" is automatic in the classical case of the factorization of integers by the quadratic or number field sieve, because the linear algebra involved takes place over $\mathbb{F}_2$).
    * Otherwise, an FP relation can be obtained. The counter C is unchanged and the procedure described is now applied to this FP relation.

It is now apparent that the counter C in fact represents the number of independent Full relations that are obtained from the input relations (this is the reason for having chosen $C_{\max} = \#\mathcal{B} + 1$). While this is clearly linked to the number of cycles, the last sub-case states the distinction between the two.

Implementing the LP-graph as described here together with its evolution process is efficiently done with the so-called *union-find* algorithm, presented and analyzed for example in [2]. The processing time obtained is then essentially constant, and tiny (bounded by the inverse Ackermann function), for each relation. As a result, the complexity of the relation collection step is the average time to build a relation times the number of relations to build before the counter C reaches $\#\mathcal{B}+1$.

3.2 A simplified algorithm

We propose a simplified algorithm which will be easier to analyze. The setting is slightly restricted, and the processing of the LP-graph is changed.

3.2.1 Restricted setting

First, we restrict our setting and redefine the large primes and factor base as follows.

- We restrict ourselves to the situation where the jacobian group of the curve is cyclic. Without loss of generality we can them assume that D_1 generates the whole jacobian group.
- $\mathcal{B}$ and $\mathcal{L}$ are restricted to the orbits of size 2 (we avoid ramification points).
- When testing for smoothness, relations involving ramification points and relations that involve less than g distinct weight 1 reduced divisors are discarded.

Recall that at most $2g+1$ orbits under ι have size 1 (these correspond to the ramification points) and that reduced divisors having multiplicities are an order of magnitude less numerous than general reduced divisors, therefore these restrictions have little impact. Note that $\mathcal{B} \cup \mathcal{L}$ and $\iota(\mathcal{B} \cup \mathcal{L})$ now form a partition of the non-ramification points of $\#\mathcal{C}(F_q)$.

This restricted setting is mostly for convenience of the exposition. The hypothesis of the first statement can be replaced by the assumption that the group structure is known (see Section 6.4), and the two other statements are there to simplify the probability estimates.

3.2.2 Simplified LP-graph

The relation search of the simplified algorithm resembles the original one, with the following radical changes. First, no edge is added to the LP-graph that is not connected to 1. Therefore the LP-graph can be seen as a tree with root 1. The technical discussion with cycles yielding or not a Full relation can therefore be skipped: If including some edge would create a cycle, this cycle would be linked to 1 by construction, so it can always be extended to include 1. Second, we do not consider all relations: Full relations are never taken into account, and only one FP relation is ever considered during the construction of the graph. Third, we split the growth of the LP-graph and the production of recombined relation into different phases.

The relation search now operates with the following three phases. In each of them relations are drawn uniformly at random, that is to say, α and β in expression (1) on page 4 are drawn uniformly at random. We will see that the duration of Phase 0 is negligible, and the switch point between Phase 1 and Phase 2 will be discussed later.

> **Phase 0** – Relations are discarded until *one* FP relation involving some large prime p is encountered. The edge $1 — p$ is included in the LP-graph.

During Phase 1, we associate with each incoming PP relation an edge E (candidate for inclusion in the LP-graph) whose vertices are the large primes p_1 and p_2 appearing in the relation.

> **Phase 1** – All relations except PP relations are discarded.
>
> - If E would not be connected to the special vertex 1, do nothing.
> - If E is already present or would create a cycle, do nothing.
> - Otherwise the edge E is added to the LP-graph (thus enlarging the connected component of 1).

> **Phase 2** – Relations with arbitrarily many large primes are considered.
>
> - If the large primes involved all belong to the LP-graph, C is incremented.
> - Otherwise do nothing (thus the LP-graph does not change).

During Phase 1, the simplified algorithm disregards many relations that would have been considered by the full algorithm. Furthermore, the LP-graph no longer changes during Phase 2, while it can always keep on growing in the original algorithm. Phase 2 runs until the counter C reaches the prescribed value $C_{\max} = \#\mathcal{B} + 1$.

In Phase 2 we consider relations with possibly more than 2 large primes. Restricting to 2 large primes as in the full algorithm would yield the same time complexity but a slightly worse space complexity.

4 Complexity analysis of the simplified algorithm

In this section, we intend to give an upper bound for the running time of the simplified algorithm. Let us recall that we analyze the situation where the genus is fixed and q grows to infinity. Throughout the analysis, the quantities we mention depend on q and the particular curve $\mathcal{C}$ under consideration. We shall write $\alpha \sim \beta$ when α and β are functions such that α/β tends to 1 when q tends to infinity (for any family of curves over $\mathbb{F}_q$). Extending this notation, we also write $\alpha \overset{\kappa}{\sim} \beta$ when α/β tends to some non-zero constant. The $o()$, $O()$ and $\widetilde{O}()$ notations also refer to asymptotic behaviors when q tends to infinity.

4.1 Probabilities and uniformity of pairs of large primes

The relation collection forms many random linear combinations. By construction, these linear combinations span the whole subgroup generated by D_1. Since we have assumed in 3.2.1 that $\mathrm{Jac}_{\mathcal{C}}(\mathbb{F}_q)$ is cyclic and that D_1 is a generator, this equals the whole jacobian group. The subset of elements of the jacobian group which meet the restrictions stated in paragraph 3.2.1 has cardinality exactly $2^g\binom{\#(\mathcal{B}\cup\mathcal{L})}{g}$. The random linear combinations which are considered are uniformly distributed within this set.

The uniformity of the large primes is obtained by counting arguments: choose arbitrarily a set of k large primes. The number of relations which involve these large primes and no others is exactly $2^g\binom{\#\mathcal{B}}{g-k}$. This implies in particular that all possible pairs of two distinct large primes are met with equal probability. The probabilities to get a Full, FP or PP relation follow:

Proposition 4. *Let k be an integer in $[1, g-1]$ and let $P_1, P_2, \ldots, P_k$ be k distinct elements of $\mathcal{L}$. The number of reduced divisors in the restricted setting that have exactly $P_1, P_2, \ldots, P_k$ as large primes is $2^g\binom{\#\mathcal{B}}{g-k}$. The probabilities a, b, c for a uniformly random reduced divisor to yield a Full, FP or PP relation are*

$$a \sim q^{g(r-1)}/g! \ , \quad b \sim q^{(g-1)(r-1)}/(g-1)! \ , \quad c \sim q^{(g-2)(r-1)}/(2(g-2)!) \ .$$

Hence, for large enough q, we have $a \ll b \ll c$ and $\#\mathcal{B} \ll \#\mathcal{L}$. To be more precise, we have

$$\frac{\#\mathcal{B}}{\#\mathcal{L}} \overset{\kappa}{\sim} \frac{b}{c} \overset{\kappa}{\sim} \frac{a}{b} \overset{\kappa}{\sim} q^{r-1} = o(1).$$

4.2 Expected running time of relation collection

We arbitrarily set our unit of time for the analysis of the relation collection to be the time required to compute a random relation and factor it. The actual complexity of this unit of time is polynomial in $\log q$ (corresponding to operations in the jacobian group and smoothness tests). Relatively to this time scale, only integer time values are relevant to a given run of the algorithm.

For our analysis, it is important to study the expected time until the graph has reached a certain size. Let $\mathbf{t}(N)$ be the random variable describing the time needed until the number of edges of the LP-graph equals N. We are interested in the expected value of $\mathbf{t}(N)$, denoted $t(N) = \mathrm{E}[\mathbf{t}(N)]$. In general a bold font is used for a random variable and the corresponding italic symbol for its expected value.

Initially, there are no edges in the LP-graph and $t(0) = 0$. Each random trial yields an FP relation with probability b. Hence the expected duration of Phase 0 is given by:

$$t(1) = \frac{1}{b}.$$

For any integer $N \geq 1$, we have:

$$\mathrm{E}[\mathbf{t}(N+1)] = \mathrm{E}[\mathbf{t}(N)] + \mathrm{E}[\mathbf{t}(N+1) - \mathbf{t}(N)],$$

and we now have to analyze the quantity $\mathbf{t}(N+1) - \mathbf{t}(N)$, which is the time before we encounter a PP relation involving exactly one large prime that meets the LP-graph.

By Proposition 4, the conditional probability for having such a PP relation depends only on the size of the LP-graph at this time, and not on its actual composition. Denoting by u the ratio $\frac{N}{\#\mathcal{L}}$, a PP relation has 0, 1, or 2 of its large primes meeting the LP-graph with respective probabilities $(1-u)^2$, $2u(1-u)$, or u^2. Therefore we have:

$$\mathrm{E}[\mathbf{t}(N+1) - \mathbf{t}(N)] = \delta(N), \quad \text{where} \quad \delta(x) \overset{\text{def}}{=\!=} \frac{1}{2c\frac{x}{\#\mathcal{L}}\left(1 - \frac{x}{\#\mathcal{L}}\right)}.$$

Note that δ is a decreasing function on the interval $\left[0, \frac{\#\mathcal{L}}{2}\right]$. We let the integer $N_{\max} \in [1, \frac{\#\mathcal{L}}{2}]$ be the target size of the number of connected large primes in the LP-graph before we switch from Phase 1 to Phase 2. We have:

$$\begin{aligned}
t(N_{\max}) &= \frac{1}{b} + \sum_{N=1}^{N_{\max}-1} \delta(N), \\
&= \frac{1}{b} + \int_1^{N_{\max}} \delta(x)dx + e, \quad \text{where } e \text{ is an error term studied below}, \\
&= \frac{1}{b} + \frac{\#\mathcal{L}}{2c}\left(\log\left(\frac{\frac{N_{\max}}{\#\mathcal{L}}}{1-\frac{N_{\max}}{\#\mathcal{L}}}\right) - \log\left(\frac{\frac{1}{\#\mathcal{L}}}{1-\frac{1}{\#\mathcal{L}}}\right)\right) + e, \\
&= \frac{1}{b} + \frac{\#\mathcal{L}}{2c}\left(\log N_{\max} - \log\left(1 - \frac{N_{\max}-1}{\#\mathcal{L}-1}\right)\right) + e, \\
&= \frac{\#\mathcal{L}}{2c}\left(\log N_{\max} + O(1)\right) + e, \quad \text{because } N_{\max} \leq \tfrac{\#\mathcal{L}}{2}.
\end{aligned}$$

The term $\frac{1}{b}$ is absorbed by the larger quantity $\frac{\#\mathcal{L}}{2c}$. The error term is:

$$\begin{aligned}
e &= \sum_{N=1}^{N_{\max}-1}\left(\delta(N) - \int_N^{N+1}\delta(x)dx\right), \\
0 \leq e &\leq \sum_{N=1}^{N_{\max}-1}\left(\delta(N) - \delta(N+1)\right) \quad \text{since } \delta \text{ is decreasing}, \\
0 \leq e &\leq \delta(1) = \frac{\#\mathcal{L}}{2c}(1+o(1)).
\end{aligned}$$

This bound and the formula above yield the expected duration of Phases 0 and 1:

$$t_{\text{phases } 0,\, 1} = t(N_{\max}) \sim \frac{\#\mathcal{L}}{2c}\log N_{\max}. \tag{2}$$

During Phase 2, all trials are independent, and each entails an increase of the counter C with probability equivalent to $\frac{1}{g!}\left(\frac{\#\mathcal{B}+N_{\max}}{\#\mathcal{B}+\#\mathcal{L}}\right)^g$ (this is verified easily). Therefore the expected duration of Phase 2 is:

$$t_{\text{phase } 2} \sim \#\mathcal{B}g!\left(\frac{\#\mathcal{B}+\#\mathcal{L}}{\#\mathcal{B}+N_{\max}}\right)^g. \tag{3}$$

Calculus yields that the total expected running time of Phases 1 and 2 is minimized by setting:

$$N_{\max} \sim \left(2cg\#\mathcal{B}g!\#\mathcal{L}^{g-1}\right)^{\frac{1}{g}} \overset{\kappa}{\sim} \left(c\#\mathcal{B}\#\mathcal{L}^{g-1}\right)^{\frac{1}{g}} \overset{\kappa}{\sim} q^{\frac{1+r(g-1)}{g}}.$$

We will see in the next two sections that the running time of the linear algebra grows like $\log(N_{\max})$, so that we are not too far from the optimal by tuning $N_{\max}$ only with respect to the relation collection.

Substituting inside formulae (2) and (3) this value for $N_{\max}$ as well as the values for c, $\#\mathcal{B}$, $\#\mathcal{L}$, we obtain the following asymptotic equivalent for the expected running time of the relation collection:

$$\begin{aligned}
t_{\text{rel. collec.}} &\overset{\kappa}{\sim} q^{1-(g-2)(r-1)}\log q, \\
&\in O\big(q^{1-(g-2)(r-1)}\log q\big).
\end{aligned} \tag{4}$$

This represents the expected number of random linear combinations to explore before finding enough relations with the simplified algorithm. Note that this hides the complexity for arithmetic operations in the jacobian group and smoothness tests, since these operations represent a unit of time.

4.3 Linear algebra

The next step of the discrete logarithm computation is a linear algebra problem: Finding a non-trivial vector in the kernel of a sparse matrix of size $\#\mathcal{B}$ using Lanczos or Wiedemann algorithm. This step has to be done modulo the group size. However, if this is not a prime (and especially if this is not a square-free number), some complications arise for which we refer to [9]. This linear algebra step has a complexity proportional to $(\#\mathcal{B})^2$ times the row weight of the matrix. The rows of the matrix correspond either to Full relations or to recombined relations created by the large prime matching process.

The recombined relations are computed during Phase 2 of the algorithm, when all large primes involved belong to the LP-graph. Let us denote by ℓ the expected average depth of the LP-graph during Phase 2 (in this phase the LP-graph does not evolve). The expected weight of a recombined relation involving k large primes is at most $g - k + k\ell g$ factor base elements , therefore $O(\ell)$. This implies that the linear algebra step requires

$$O\left(\ell q^{2r}\right)$$

operations modulo the group order.

4.4 Analysis of the LP-graph depth

For any integer N representing the size of the LP-graph at a given point in time during Phase 1, and for any integer $i \geq 0$, let the random variable $\mathbf{d}_i(N)$ denote the number of vertices (excluding 1) belonging to the graph and linked to the special vertex 1 by a path of length i. We have $\mathbf{d}_0(N) = 0$, and $\mathbf{d}_i(N) = 0$ for all $i > N$. Furthermore we have $\sum_{i=0}^{\infty} \mathbf{d}_i(N) = N$.

Let $\mathbf{w}(N) \stackrel{\text{def}}{=\!=} \sum_{i=0}^{\infty} i\mathbf{d}_i(N)$, so that $\frac{\mathbf{w}(N)}{N}$ is the average depth of the LP-graph. We have:

$$\begin{aligned}
\mathrm{E}\left[\mathbf{w}(N+1) \,\middle|\, (\mathbf{d}_i(N))_i\right] &= \mathbf{w}(N) + \sum_{i=0}^{\infty} (i+1)\frac{\mathbf{d}_i(N)}{N}, \\
&= \mathbf{w}(N) + 1 + \frac{\mathbf{w}(N)}{N}.
\end{aligned}$$

We infer an easy recurrence formula for $w(N) \stackrel{\text{def}}{=\!=} \mathrm{E}\left[\mathbf{w}(N)\right]$:

$$w(N+1) - w(N) = 1 + \frac{w(N)}{N}.$$

The expected average graph depth during Phase 2 is $\frac{w(N_{\max})}{N_{\max}}$. In order to bound this value, we introduce the auxiliary function f defined by $f(x) = x + x\log x$. This function is a solution of the differential equation analogous to the recurrence formula above. Since f' is an increasing function, we have for any integer N:

$$f(N+1) - f(N) \geq f'(N) = 1 + \frac{f(N)}{N}.$$

Since we also have $f(1) = w(1) = 1$, this implies by induction that $w(N) \leq f(N)$, and:

$$\ell = \frac{w(N_{\max})}{N_{\max}} \leq \frac{f(N_{\max})}{N_{\max}} = 1 + \log N_{\max}.$$

Given that $N_{\max} \leq \frac{\#\mathcal{L}}{2}$ and $\frac{\#\mathcal{L}}{2} < q$ for $q \gg 0$, we finally reach a bound for the complexity of the linear algebra step:

$$t_{\text{lin. alg.}} \in O\left(q^{2r}\log q\right). \tag{5}$$

The "unit of time" corresponding to this equation is the time of operations modulo the group order. As for the relation collection case, this hides a complexity involving logarithmic factors in q.

4.5 Computing the discrete logarithm

The dependency obtained from the linear algebra step has the form $AD_1 + BD_2 = 0$, where the coefficients A and B are obtained as the sums of the corresponding terms in the different relations involved in the dependency. We must show that from the dependency, the logarithm of D_2 to base D_1 can be obtained with high probability, i.e. that B is invertible modulo $\#G$.

After the first phase, each connected large prime in the LP-graph corresponds to a weight 1 reduced divisor that can be rewritten as a sum of elements in the factor base plus a linear combination of D_1 and D_2. Let us consider now what happens in the second phase when a linear combination $D = \alpha D_1 + \beta D_2$ produces a row in the matrix. The reduced divisor D can be obtained in as many as $\#G$ different ways from combinations of D_1 and D_2, and all these combinations have equal probability (any value for β is possible, and for each β only one value of α gives a sum equal to D). If D contains some large primes, they have to be rewritten using the LP-graph. Hence the row will correspond to the reduced divisor $D' = D + \alpha' D_1 + \beta' D_2 = (\alpha + \alpha')D_1 + (\beta + \beta')D_2$, where α' and β' depend only on the large primes and the data in the LP-graph. The perturbation due to the use of large primes is therefore independent of the choice of α and β that give D. We have thus obtained that each row correspond to as many as $\#G$ different combinations of D_1 and D_2, and all these combinations have equal probability.

Looking at the final result, we use the same kind of argument: The result of the linear algebra computation does not depend on the particular way the reduced divisor corresponding to each row is represented as a sum of D_1 and D_2. Therefore the resulting linear combination between D_1 and D_2 that annihilates is uniformly random among all the possible choices. The probability that B is not invertible modulo $\#G$ is no more than $1 - \frac{\phi(\#G)}{\#G}$. In that unlucky case, we can add a row to the matrix, thus yielding another dependency to try. Since $\liminf \frac{\phi(n)}{n} = \frac{e^{-\gamma}}{\log\log n}$, this means that in worst cases (namely if $\#G$ is a smooth integer), we have to add an expected number of $O(\log\log \#G)$ rows to the matrix, which does not change the complexity. In these special cases, however, since $\#G$ is smooth, one should also consider the Pohlig-Hellman algorithm [22].

4.6 Total complexity

The times $t_{\text{rel. collec.}}$ and $t_{\text{lin. alg.}}$ are relative to different units, both hiding arithmetic complexities which are polynomial in $\log q$. Ignoring such logarithmic factors which are of negligible importance to the overall complexity, we can balance the relation search (see Formula (4)) with the linear algebra (see Formula (5)) by taking $r = 1 - \frac{1}{g}$. Finally, we obtain Theorem 1 stated in the introduction.

For small genera, we get the following complexities for groups of almost prime orders:

g	3	4	5	6
Pollard's algorithm	$q^{3/2}$	q^2	$q^{5/2}$	q^3
Basic index calculus	q^2	q^2	q^2	q^2
Balanced index calculus	$q^{3/2}$	$q^{8/5}$	$q^{5/3}$	$q^{12/7}$
Single large prime	$q^{10/7}$	$q^{14/9}$	$q^{18/11}$	$q^{22/13}$
Double large prime	$q^{4/3}$	$q^{3/2}$	$q^{8/5}$	$q^{5/3}$

Obviously, when the genus gets large, the improvement is marginal, not to say invisible. On the other hand, for genus 3 curves, the $\widetilde{O}(q^2)$ complexity of the basic index calculus becomes $\widetilde{O}(q^{1.5})$ in its balanced variant and drops to $\widetilde{O}(q^{1.428\cdots})$ with the single large prime algorithm of [23] and to $\widetilde{O}(q^{1.333\cdots})$ with our double large prime variant. The constants involved are small enough so that even for small sizes our algorithm is expected to be faster than Pollard's Rho algorithm. The crossover is examined in Section 5.2.

5 Computer experiments

We have implemented the full algorithm with two goals in mind. First we want to assert that the upper bound obtained for the running time of the simplified algorithm is not too far from the running time of the real algorithm. In particular, we need to check that the cycle length is not too bad, since there is no easy argument to relate it to the cycle length we analyzed in the simplified algorithm. The second objective is to compare our algorithm with Pollard Rho algorithm.

Our implementation is in `C/C++` and covers only hyperelliptic curves of genus 3, since this is the most important case. We programmed the arithmetic in the jacobian group using explicit formulae, based on the work of [27]. On a Pentium-M processor clocked at 1.7 GHz, our implementation performs 200 000 additions or doublings in the jacobian group per second (i.e. 5 microseconds each), for a prime base field of size up to 2^{27}. A step of the algorithm, corresponding to the unit of time chosen in the analysis above, is performed in 20 microseconds. This includes the time for the smoothness test.

5.1 Relation search in the full algorithm

The series of experiments shown in table 1 gives an idea of how the full algorithm performs. The final value of t for several experiment sizes are listed. We recall that t represents the number of trial relations to test for smoothness before sufficiently many recombined relations are obtained. The running time of the relation search is t times a polynomial expression in $\log q$ accounting for arithmetic in the jacobian group and smoothness tests.

q	final t	$t/q^{4/3}$	$\#\mathcal{B}$	$\frac{\text{cyc. len.}}{\log q}$
$\approx 2^{15}$	815473	0.78	512	1.29
$\approx 2^{16}$	1811672	0.69	812	1.19
$\approx 2^{17}$	4705192	0.71	1290	1.41
$\approx 2^{18}$	11253002	0.67	2047	1.42
$\approx 2^{19}$	27776102	0.66	3250	1.44
$\approx 2^{20}$	66834647	0.63	5160	1.47
$\approx 2^{21}$	170327927	0.63	8191	1.59
$\approx 2^{22}$	417044579	0.62	13003	1.70
$\approx 2^{23}$	1036566361	0.61	20642	1.80
$\approx 2^{24}$	2576921045	0.60	32767	1.92
$\approx 2^{25}$	6430349490	0.59	52015	2.02
$\approx 2^{26}$	15899195912	0.58	82570	2.18
$\approx 2^{27}$	39993810485	0.58	131071	2.32

Table 1: Final value of t for the full algorithm

The comparison of t with $q^{4/3}$ is given in the third column. Since the LP-graph has $\#\mathcal{L} \approx q/2$ vertices and that on average every $\approx 2q^{1/3}$ steps one produces either an edge or an increment of C, then the running time of the first phase of the full algorithm is bounded by a constant times $q^{4/3}$. Hence a $\log q$ factor is saved in this phase compared to the analysis of the simplified algorithm.

For the average cycle length, given in the fourth column, it seems to be slightly worse than $\log q$, but it is not possible to make a guess for the real asymptotic behavior from these experimental values.

5.2 Comparison with Pollard Rho

The Pollard Rho algorithm is known to have $\widetilde{O}(\sqrt{\#G}\,)$ complexity. More precisely, in the case of a prime order jacobian group of a hyperelliptic curve of genus three, the number of jacobian operations required is equivalent to $\sqrt{\pi\#J}/2$ (we take advantage of the hyperelliptic involution). Instantiated with the parameters for a genus three curve over $\mathbb{F}_q$, where $q \approx 2^{27}$, this yields $1.37 \cdot 10^{12}$ operations in the jacobian group, or, at the pace quoted above, 79 days of computation on a Pentium-M processor.

In comparison, the index calculus algorithm described here, with the double large prime variation, requires only $4 \cdot 10^{10}$ jacobian group operations and smoothness tests on the same curve as above. This corresponds to 9 days of computation. We performed the corresponding linear algebra computation, using as a linear system solver the block Wiedemann implementation described in [5, 24, 25]. This linear algebra computation required 5.8 days of computation on the same processor. Therefore, the algorithm presented here induces a speed-up of 5.3 compared to Pollard Rho for this problem size. For a curve defined over a field of size 2^{24}, the corresponding speed-up is already of 4.4. Using our implementation, a definition field of size 2^{27} would correspond roughly to the crossover point between Pollard Rho and the single large prime algorithm.

q	Relation search	Linear algebra	Total	Pollard Rho (estim.)
2^{24}	0.6 days	0.2 days	0.8 days	3.5 days
2^{27}	9 days	5.8 days	14.8 days	79 days

Table 2: Total time for our algorithm and Pollard Rho

Note that because of the linear algebra step, the index calculus approach cannot enjoy the same amount of parallelization as Pollard's algorithm and its variants. Partial distribution of the linear algebra is possible through the use of multi-processor machines, and taking advantage of the distribution capabilities of the *block* Wiedemann algorithm. For the largest experiment, we have been able to reduce the linear algebra wall-clock time to 1.9 days this way, with room for further improvement since we have not yet ported the asymptotically fast algorithm presented in [24].

6 Qualitative comments

6.1 Growth of the LP-graph

We digress here to comment briefly on the growth of the LP-graph in the context of the simplified algorithm. Previous works dealing with double large prime variants [17, 18] have coined terms such as "explosive growth" or "phase transition" for describing the growth of this graph. Such behaviour

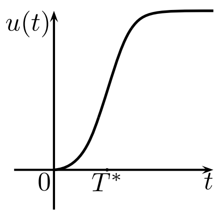

Figure 1: General form of $u(t)$

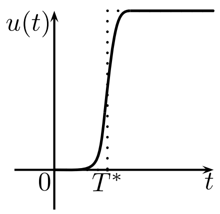

Figure 2: $u(t)$ when $\#\mathcal{L} \gg 1$

is indeed shown by the equations obtained. We can obtain a graphical view of the evolution of the LP-graph as time goes by expressing the size N of the graph as a function of the expected time t. Figures 1 and 2 represent the ratio $u(t) \stackrel{\text{def}}{=\!=} \frac{N}{\#\mathcal{L}}$ for arbitrary values of the relevant constants. On a large scale, i.e. when $\#\mathcal{L} \gg 1$ as in Figure 2, the slope close to $t = 0$ looks horizontal.

The curve has an inflexion point at time $T^* = (T + O(1)) \log \#\mathcal{L}$ with $T = \frac{\#\mathcal{L}}{2c}$. The time T^* can be viewed as a transition point, since it is around this time that $u(t)$ varies the most and jumps from 0 to 1. Indeed, for $\#\mathcal{L} \gg 1$, for $t = T^* - 2T$, we have $u(t) \approx 0.12$, whereas for $t = T^* + 2T$, we have $u(t) \approx 0.88$.

We note however that by the choice of $N_{\max}$, the construction of the LP-graph already terminates at a time of $\sim (1 - \frac{1}{g} + \frac{1}{g^2}) \cdot T^*$, i.e. before the phase transition is reached.

6.2 Full algorithm and random graphs

We used experiments in order to get an idea of the behaviour of the cycle lengths in the full algorithm, but, as we said before, it is hard to guess an asymptotic behaviour from these. Another approach to make conjectures is to consider the LP-graph as a random graph perturbed by the special vertex 1 and the numerous edges attached to it due to FP relations. At the end of the computation we expect as many as $O(b\,q^{2-\frac{2}{g}}) = O(q^{1-\frac{1}{g}})$ such edges, so this perturbation is more important for high genera.

We mention two theoretical results from the literature on random graphs. The first suggests the cycle length might be exponential, but is probably too pessimistic. The second suggests polynomial cycle length, but is probably too optimistic.

In [10], it is proven that the first cycle in a random graph appears once $\#\mathcal{L}/2$ edges are included and the length of the first cycle is of order $\Theta((\#\mathcal{L})^{1/6})$. It is also proven that for any constant k, the k-th cycle has a length of order $\Theta((\#\mathcal{L})^{1/6} \log(\#\mathcal{L})^{k-1})$. These lengths would be too large in our context. This result does not capture our situation for two reasons: First, we need to estimate the length of a large quantity of cycles, and the analysis of [10] is valid only for the first few cycles. In particular, it assumes that the cycles are in disjoint components, which is not the case in the end of the relation search. Second, the presence of the FP relations is enough to make the first cycle appear earlier than would be expected otherwise.

The other theoretical result is taken from [4]: the diameter of a random graph is $O(\log(\#\mathcal{L}))$ if the number of edges is a constant bigger than $1/2 \cdot \#\mathcal{L}$. This estimate for the diameter would be perfect for our analysis. However in the full algorithm the relations are built on the fly as soon as cycles appear, so that we cannot deduce a useful bound for the cycle lengths, except maybe at the very end of the relation search. It is also not clear that the algorithm stops at a point where the graph is dense enough to have the appropriate diameter.

6.3 Memory requirement

For the simplified algorithm, the memory requirement during the linear algebra step is in $\widetilde{O}(q^{1-\frac{1}{g}})$. In the full algorithm this might be larger if the cycle length cannot be bounded by a polynomial in $\log q$.

For the relation collection, the estimate of the memory requirement is in $\widetilde{O}(q)$, if a naïve implementation is used for the LP-graph. Indeed, there are $\#\mathcal{L} = O(q)$ vertices in the graph and since no edge that would create a cycle is ever stored in the LP-graph, there are at most $O(q)$ edges, so we need $O(q)$ memory to store the LP-graph information.

In the case of the full algorithm we cannot hope for a better storage requirement, since the experiments suggests that $\Theta(q)$ edges are indeed needed before having enough relations.

On the other hand, in the simplified algorithm, at the end of Phase 2, the LP-graph has only $N_{\max} = O(q^{1-\frac{1}{g}+\frac{1}{g^2}})$ edges. Therefore most of the vertices (i.e. large primes) are never used. Choosing an appropriate data structure the memory requirement drops to $\widetilde{O}(N_{\max})$ (with a possible loss of a $\log q$ factor in the time complexity, in order to handle the data structure).

Hence the memory requirement of the simplified algorithm can be made smaller than for the full algorithm by an exponential factor. This should be kept in mind for practical applications, when the memory might be problematic. A reasonable approach might be to implement a mixture between the full and the simplified algorithm, where we keep the memory low but still make use of the FP relations.

6.4 Relevance to more general context

We now provide heuristic arguments backing the validity of our approach to a broader context.

We have restricted our proof to the case of hyperelliptic curves of cyclic jacobian group given by an imaginary Weierstrass equation. If the jacobian group is not cyclic, our analysis is not valid. However, as soon as the group structure is known, we can follow the randomizing strategy of Section 7 in [9] to produce uniformly random elements in the whole group.

Furthermore, the results are formulated only for the context of hyperelliptic curves, but on heuristic basis our algorithm should perform equally well for general curves.

General setting

For any fixed genus g, we consider a family of curves over a finite field $\mathbb{F}_q$ with q elements, where q grows to infinity. We assume that the curves are given in the following representation, which is more suitable from the algorithmic point of view. Let $\mathcal{C}$ be a curve in the family. Even if the practical models are affine or singular, we actually consider a complete non-singular curve associated to it, and the notation $\mathcal{C}$ is reserved for this non-singular model. The algorithmic assumptions are the following:

- There exists a rational point $P_\infty \in \mathcal{C}(\mathbb{F}_q)$. This is true for any curve as long as q is large enough.
- The group structure of $\mathrm{Jac}_\mathcal{C}(\mathbb{F}_q)$ is known. Namely we have an explicit set of generators G_1, $\ldots$, G_k, of known orders such that $\mathrm{Jac}_\mathcal{C}(\mathbb{F}_q) = \langle G_1 \rangle \oplus \cdots \oplus \langle G_k \rangle$.
- We have a probabilistic algorithm running in time polynomial in $\log q$ to perform the group operations in $\mathrm{Jac}_\mathcal{C}(\mathbb{F}_q)$. The elements are represented by divisors in the form $E - wP_\infty$, where E is an effective divisor of degree $w \leq g$ and w is minimal. Such an E exists and is unique in each class, and $E - wP_\infty$ is called a reduced divisor.
- We have a probabilistic algorithm running in time polynomial in $\log q$ to decompose an effective divisor as the sum of its prime divisors.

Apart from the known group structure, these assumptions are verified in the classical case of hyperelliptic curves given by an imaginary Weierstrass equation (using Mumford representation), and for $\mathcal{C}_{ab}$ curves. Also, if a curve is given by a plane equation of bounded degree, there are algorithms available to perform the group operations in polynomial time [26, 15, 13].

In this setting, it is possible to define the factor base and the large primes by partitioning the set of effective divisor of degree 1 into sets of appropriate cardinalities. The descriptions of the full and simplified algorithms for hyperelliptic curves extend easily, with Cantor's algorithm and Mumford representation replaced with their generalized equivalent notions. However, the proof of the simplified algorithm does not follow, since it heavily relies on the statistical properties of large primes given in Proposition 4. In the case of hyperelliptic curves, it was simple to estimate this statistics, based on the properties of the Mumford representation with respect to the hyperelliptic involution. Heuristically, there seems to be no reason why the statistics would behave differently for general curves, however a proof in the most general setting is out of the scope of this work, and we keep to a heuristic result for non-hyperelliptic curves.

7 Conclusion

We have described two algorithms for solving discrete logarithms in curves of small genus at least 3. The first one is a traditional double large prime variant of the algorithm of [23] and its complexity is heuristic. The second algorithm is a simplified variant that can be rigorously analyzed in the context of cyclic jacobian groups of hyperelliptic curves. The complexity is better than previously known methods and experiments demonstrated that even for rather small sizes, our method is faster than Pollard Rho algorithm. On the other hand, the space requirement is much larger and can become problematic.

The direct application to cryptography is that the security of a genus 3 cryptosystem is over-estimated if only Pollard's Rho algorithm is taken into account. Indeed, we have shown that the running time for solving a discrete logarithm problem in a genus 3 jacobian group has a complexity similar to a discrete logarithm computation in an elliptic curve for which the logarithm of the group order is 1/9th smaller. We therefore recommend to enlarge the group-size by 12.5%.

The complexity of our attack, as for any index calculus method, depends only on the size of the whole jacobian group. Hence we are in a situation somewhat similar to multiplicative groups of finite fields: It is possible to work in a subgroup and with a private key whose sizes are large enough to counter Pollard Rho and similar attacks, as long as the size of the whole group is large enough to prevent an index calculus attack.

We note that our method also applies to the Weil descent algorithm of [12] that attacks elliptic curves defined over small extension fields. Hence, this asymptotic 12.5% penalty also applies to elliptic curve cryptosystems defined over extension finite fields whose degree is a multiple of 3.

Finally, we would like to point out that an alternative double large prime method that uses smooth functions instead of divisors has recently been proposed [7] for the non-hyperelliptic setting. A heuristic complexity analysis indicates that this algorithm can solve the discrete logarithm problem in jacobian groups of non-hyperelliptic curves of genus 3 over $\mathbb{F}_q$ in time of $\widetilde{O}(q)$.

References

[1] L. M. Adleman, J. DeMarrais, and M.-D. Huang. A subexponential algorithm for discrete logarithms over the rational subgroup of the jacobians of large genus hyperelliptic curves over finite fields. In L. Adleman and M.-D. Huang, eds., *ANTS-I*, vol. 877 of *Lecture Notes in Comput. Sci.*, pp. 28–40. Springer–Verlag, 1994.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms.* Addison–Wesley, Reading, MA, 1974.

[3] D. G. Cantor. Computing in the Jacobian of a hyperelliptic curve. *Math. Comp.*, 48(177):95–101, 1987.

[4] F. Chung and L. Lu. The diameter of random sparse graphs. *Adv. in Appl. Math.*, 26:257–279, 2001.

[5] D. Coppersmith. Solving linear equations over GF(2) via block Wiedemann algorithm. *Math. Comp.*, 62(205):333–350, Jan. 1994.

[6] J.-M. Couveignes. Algebraic groups and discrete logarithm. In *Public-key cryptography and computational number theory*, pp. 17–27. de Gruyter, 2001.

[7] C. Diem. Index calculus in class groups of plane curves of small degree. Cryptology ePrint Archive: Report 2005/119, 2005.

[8] A. Enge. Computing discrete logarithms in high-genus hyperelliptic Jacobians in provably subexponential time. *Math. Comp.*, 71:729–742, 2002.

[9] A. Enge and P. Gaudry. A general framework for subexponential discrete logarithm algorithms. *Acta Arith.*, 102:83–103, 2002.

[10] P. Flajolet, D. Knuth, and B. Pittel. The first cycles in an evolving graph. *Discrete Math.*, 75:167–215, 1989.

[11] P. Gaudry. An algorithm for solving the discrete log problem on hyperelliptic curves. In B. Preneel, ed., *Advances in Cryptology – EUROCRYPT 2000*, vol. 1807 of *Lecture Notes in Comput. Sci.*, pp. 19–34. Springer–Verlag, 2000. Proceedings.

[12] P. Gaudry. Index calculus for abelian varieties and the elliptic curve discrete logarithm problem. Cryptology ePrint Archive: Report 2004/073, 2004.

[13] F. Heß. Computing Riemann-Roch spaces in algebraic function fields and related topics. *J. Symbolic Comput.*, 33:425–445, 2002.

[14] F. Heß. Computing relations in divisor class groups of algebraic curves over finite fields. Preprint, 2004.

[15] M.-D. Huang and D. Ierardi. Counting points on curves over finite fields. *J. Symbolic Comput.*, 25:1–21, 1998.

[16] N. Koblitz. Hyperelliptic cryptosystems. *J. of Cryptology*, 1:139–150, 1989.

[17] A. K. Lenstra and M. S. Manasse. Factoring with two large primes. *Math. Comp.*, 63(208):785–798, Oct. 1994.

[18] P. Leyland, A. K. Lenstra, B. Dodson, A. Muffett, and S. S. Wagstaff, Jr. MPQS with three large primes. In C. Fieker and D. R. Kohel, eds., *ANTS-V*, vol. 2369 of *Lecture Notes in Comput. Sci.*, pp. 448–462. Springer–Verlag, 2002. Proceedings.

[19] A. Menezes, Y.-H. Wu, and R. Zuccherato. An elementary introduction to hyperelliptic curves. Appendix to *Algebraic aspects of cryptography*, by N. Koblitz, pages 155–178, Springer-Verlag, 1998.

[20] V. Müller, A. Stein, and C. Thiel. Computing discrete logarithms in real quadratic congruence function fields of large genus. *Math. Comp.*, 68(226):807–822, 1999.

[21] J. Pila. Frobenius maps of abelian varieties and finding roots of unity in finite fields. *Math. Comp.*, 55(192):745–763, Oct. 1990.

[22] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Trans. Inform. Theory*, IT–24:106–110, 1978.

[23] N. Thériault. Index calculus attack for hyperelliptic curves of small genus. In C. Laih, ed., *Advances in Cryptology – ASIACRYPT 2003*, vol. 2894 of *Lecture Notes in Comput. Sci.*, pp. 75–92. Springer–Verlag, 2003. Proceedings.

[24] E. Thomé. Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm. *J. Symbolic Comput.*, 33(5):757–775, Jul. 2002.

[25] E. Thomé. *Algorithmes de calcul de logarithme discret dans les corps finis.* Thèse, École polytechnique, May 2003.

[26] E. Volcheck. Computing in the jacobian of a plane algebraic curve. In L. Adleman and M.-D. Huang, eds., *ANTS-I*, vol. 877 of *Lecture Notes in Comput. Sci.*, pp. 221–233. Springer–Verlag, 1994.

[27] T. Wollinger, J. Pelzl, and C. Paar. Cantor versus Harley: Optimization and analysis of explicit formulae for hyperelliptic curve cryptosystem. Technical report, Universität Bochum, 2004. Available at `http://www.crypto.rub.de/`.