Time-Memory Trade-offs for Index Calculus in Genus 3

Kim Laine[1] and **Kristin Lauter**[2]

[1] University of California, Berkeley
[2] Microsoft Research, Redmond

Abstract. In this paper, we present a variant of Diem's $\widetilde{O}(q)$ index calculus algorithm to attack the discrete logarithm problem (DLP) in Jacobians of genus 3 non-hyperelliptic curves over a finite field $\mathbb{F}_q$. We implement this new variant in C++ and study the complexity in both theory and practice, making the logarithmic factors and constants hidden in the $\widetilde{O}$-notation precise. Our variant improves the computational complexity at the cost of a moderate increase in memory consumption, but we also improve the computational complexity even when we limit the memory usage to that of Diem's original algorithm. Finally, we examine how parallelization can help to reduce both the memory cost per computer and the running time for our algorithms.

1 Introduction

The discrete logarithm problem (DLP) in the Jacobian of a genus g curve with $g > 1$ over a finite field $\mathbb{F}_q$ was proposed for use in public key cryptosystems by Koblitz [Kob]. The potential advantage over elliptic curve cryptosystems comes from the fact that $\#\operatorname{Jac}_C(\mathbb{F}_q) = q^g + O(q^{\frac{2g-1}{2}})$. So the underlying finite field can be smaller, while still yielding a group of the same size with potentially the same security level. In genus 2 there are no known attacks faster than the generic square root attacks, and the smaller field size and alternate models for the group make the arithmetic very efficient, even efficient enough to be competitive with elliptic curve cryptography ([BCHL], [BCLS]).

However for curves of very large genus, subexponential index calculus attacks were discovered by Adleman, DeMarrais, and Huang [ADH], and improved by Gaudry, Enge, and others [Gau], [Eng], [EG], [VJS]. For small genus but still with $g > 3$, the expected security was drastically reduced by attacks, which, although exponential, were so much better than square-root attacks so as to render the complexity/security trade-off unacceptable (e.g. [Gau], [The], [GTTD], [Nag]).

The case of genus 3 is less clear. The hyperelliptic locus of genus 3 curves is of codimension 1 so almost all genus 3 curves are non-hyperelliptic. The non-hyperelliptic curves are smooth plane quartics due to the canonical embedding. The rest are hyperelliptic of higher degree. A double large prime index calculus algorithm ([GTTD]) in the hyperelliptic case reduces the complexity of the DLP from $O(q^{3/2})$ using Pollard rho to $\widetilde{O}(q^{4/3})$, where $\widetilde{O}$ hides both constants and logarithmic factors, but this does not seem to be efficient enough to rule out the use of genus 3 hyperelliptic curves in cryptography. On non-hyperelliptic curves the simpler geometry can be exploited to yield more efficient attacks. Diem has been developing index calculus algorithms on Jacobians of low-degree plane curves [Die, Die2], in particular the case of non-hyperelliptic genus 3 [DT]. These more efficient algorithms show that the complexity of solving the DLP on a non-hyperelliptic genus 3 Jacobian is at most $\widetilde{O}(q)$.

But the $\widetilde{O}$-notation is often quite misleading, since the hidden constants and logarithmic factors can be significant for field sizes of practical interest. In this paper, we develop a variant of Diem's algorithm which improves the computational complexity at the cost of a moderate increase in memory consumption. We implemented this new variant in C++ and studied the complexity in both theory and practice, making the logarithmic factors and constants hidden in the $\widetilde{O}$-notation precise. We found clear estimates for the security of non-hyperelliptic genus 3 Jacobians which have to be considered when designing genus 3 hyperelliptic cryptosystems, due to possible isogeny attacks on hyperelliptic Jacobians as given in [Smi].

Perhaps the biggest limitation of all algorithms of this type is that they require massive amounts of memory. We also study our variant in the case where the memory usage is limited to that of Diem's original algorithm, and still find performance improvements. Finally we look at how parallelization can help to reduce both the memory cost per computer and the running time.

1.1 Overview

Diem's algorithm works by building a graph (actually a tree) which can be used to construct relations for the linear algebra stage of the index calculus algorithm. Taking advantage of the

geometry of plane quartic curves, one can pass a line through pairs of points on the curve and this line then intersects the curve (over an algebraically closed field) in another 2 points. This gives a relation which holds in the group (the Jacobian of the curve) and can be used in index calculus. Using a fixed initial factor base of size $\widetilde{O}(q^{1/2})$, each pair of factor base elements yields another two points on the line, and if they are rational over the base field and if one of them is in the factor base or the tree already and the other is not, then the other point may be added to the tree and tagged with this relation.
Once the tree is built and has size $q^{3/4}$, a relation-finding stage commences. The goal of this stage of index calculus is to produce relations involving only factor base elements, called *full relations*. Remaining unused pairs of factor base elements generate additional pairs of points, which, if they are already in the tree, can be used to trace down to obtain full relations, involving only factor base elements. The version of Diem's algorithm from [Die2] is described precisely below in Section 2.

New variant. A rough description of our new algorithm is as follows. A precise description is given in the algorithm presented in Section 3, and total time and storage cost is given in Section 4. We start with an initial factor base of much smaller size, $\widetilde{O}(q^{1/8})$, which we then expand into a full factor base of size $\widetilde{O}(q^{1/2})$ as follows.
At the *base vertices* construction stage, we take all pairs of elements of the initial factor base, find the other two points on the curve which lie on the line intersecting the factor base points, and (if they are rational) add one of them to the factor base and one to the graph and call it a base vertex, connected to the base (special node) of the graph. This stage is very efficient because it simultaneously builds the factor base and the graph, with very high probability at each step because it builds the graph as long as not both are in the graph already, and the graph is very small to begin with.
The next step is to build *triangle relations*. We take all pairs of base vertices, find the other two points on the line and the curve, and add one to the factor base, and one to the graph. The one added to the graph is called a *top triangle vertex*. At the end of this stage, the graph consists of many triangles, the bottom two triangle vertices are connected directly to the factor base (the special node), and the top vertex is connected to the base triangle vertices (with a relation involving one element of the factor base). At this point, the factor base consists of roughly $q^{1/2}$ elements, roughly the same as the size of the graph.
The next stage continues *graph building* while also starting to *collect full relations*. Essentially we will build tree branches on top of the top triangle vertices, while also collecting relations. That way we can make use of relations both when the new points are in the graph and when one of them is not. We only discard a relation if both of the new points are not in the graph. This stage terminates when enough full relations have been generated. In Theorem 2 we show that if the size of the factor base is roughly $4\lambda^4 q^{1/2}$, the graph grows to size $4\lambda^2 q^{3/4}$, where λ satisfies $\lambda \exp(4\lambda^8) = q^{1/8}$, and the algorithm can be expected to terminate successfully.
For the linear algebra step, we don't make any specific changes, but we will show that our average row weight is a constant factor less than that in Diem's algorithm (Lemma 2), which leads to an improvement in running time of the linear algebra stage.
Note that for both our algorithm and Diem's, the entire graph building and relation finding stages do not use the (expensive) group operation on the Jacobian of the plane quartic curve. Instead each step requires only finding the other two points of intersection of a line with the curve. The complexity of this operation is logarithmic in q and in the characteristic 2 case it is particularly simple, as we explain below. The running time of our algorithm presented here refers to the characteristic 2 case.

Performance. To understand the overall performance of this type of attack on the discrete logarithm problem, we must estimate the time and memory costs for both the relation collection stage and the linear algebra stage. To maximize efficiency, these algorithms are generally designed so that the time and memory costs in the two stages are relatively balanced. In Table 1, we estimate the time and memory costs for the relation collection stage of our algorithm for field sizes of interest, $2^{30} \leq q \leq 2^{240}$, based on the theoretical results from Theorem 2. Time complexity is measured in terms of counting the number of field multiplications required, and based on rough estimates of the parameters for our algorithm where q is in the range of $2^{70} \leq q \leq 2^{120}$ (see Remark 1), we estimate the running time to be

$$\approx 1.23123 \cdot \log_2^2(q) \cdot q$$

in the binary case. Table 1 shows for example, that to achieve a 128-bit security level measured in terms of time complexity, q should be at least 115 bits. This ignores, however, the memory requirements for these algorithms, which is the size of the graph, $\widetilde{O}(q^{3/4})$ for both. Table 2 gives

experimental results, for small q, and shows that in practice the performance of our implementation of our algorithm matches very well with the theoretical predictions.

Comparison with Diem's algorithm. Our algorithm gives a constant factor improvement over Diem's algorithm in two different ways. First, the time required for the matrix-building stage is dominated by the graph-building and relation-finding step of our algorithm, and this is proportional to the square of the size of the factor base. In Lemma 2 we show that the squared ratio of the size of Diem's factor base to the size of our factor base approaches 3 (from above) as q increases. In addition, the graph that we build is wider and not as deep as Diem's graph, which leads to an improvement in the row weight of our vectors for the linear algebra stage. In Lemma 2 we show that we improve the linear algebra stage by a factor which approaches 9 asymptotically as q increases, which is demonstrated in Table 4.
We obtain these improvements at the cost of slightly increasing the size of the graph used in relation collection, which increases the memory requirements of the algorithm. Our graph size is roughly $4\lambda^2 q^{3/4}$ as compared to Diem's $q^{3/4}$. Table 1 shows values for λ which are very close to 1 for all q in the range considered. However, we also show that, even if we restrict our graph size to be the same as Diem's, we still get an improvement in the running time. Corollary 1 shows that in the case of restricted graph size our algorithm improves on Diem's by the same ratios asymptotically, but the ratios approach 3 and 9 more slowly. This is demonstrated in Table 5.

2 Diem's Index Calculus

There are several variants of Diem's index calculus for low degree plane curves. We restrict to the case of non-hyperelliptic genus 3 curves over finite fields embedded as smooth plane quartics using the canonical embedding. These are all double large prime index calculus algorithms where elements of the group are decomposed into sums of factor base elements and at most two non-factor base elements (large primes). The large primes are then eliminated using various methods to produce relations consisting only of factor base elements.
Let C be a non-hyperelliptic curve of genus 3 over a finite field $\mathbb{F}_q$. Using the canonical embedding it can be realized as a smooth plane quartic. Let P_0 be an $\mathbb{F}_q$-rational point on $\mathcal{C}$. The algorithms find x in the DLP

$$D_2 - 3[P_0] = x \cdot (D_1 - 3[P_0])$$

provided that a solution exists. Here $\deg(D_1) = \deg(D_2) = 3$ and both D_1 and D_2 are sums of three $\mathbb{F}_q$-rational points: $D_1 = [P_1^1] + [P_2^1] + [P_3^1]$, $D_2 = [P_1^2] + [P_2^2] + [P_3^2]$. For simplicity we assume that both divisors $D_2 - 3[P_0]$ and $D_1 - 3[P_0]$ live in the same subgroup of prime order p. Remark that, in general, any $\mathbb{F}_q$-rational divisor can be written in a unique way in the *along P_0 maximally reduced form* $D - \ell[P_0]$ ([Hes]), where $\ell \leq 3$ and in the generic case $\ell = 3$. If D decomposes into a sum of three $\mathbb{F}_q$-rational points, it is called *completely split*. If the divisor is not completely split, the standard strategy is to multiply both sides of the DLP by constants until they become completely split and then proceed as usual. Once the discrete logarithm has been found, these multipliers must be cancelled. See [Die2] for details. Here is the algorithm as given in [Die2] with one modification. Diem suggests taking the factor base to be slightly larger to ensure that the algorithm terminates successfully. The size used below is an absolute minimum for which we can expect the algorithm to succeed. In practice the size should be slightly bigger.
Choosing the factor base: Choose as set $\mathcal{F} \subseteq \mathcal{C}(\mathbb{F}_q)$ with $\lceil ((3/2)\ln q + 4)^{1/2}\, q^{1/2} \rceil$ elements. Include the points $\{P_j^i\} \cup \{P_0\}$ in $\mathcal{F}$. The set $\mathcal{F}$ is called the *factor base*. Let $\mathcal{L} := \mathcal{C}(\mathbb{F}_q) \setminus \mathcal{F}$ be the set of *large primes*.

Tree building: We constuct a tree $\mathbf{T}$ as follows.

Let $\mathbf{V} = \{*\}$ be the set of vertices of $\mathbf{T}$. Here $*$ denotes a distinguished *special vertex*. Let $\mathbf{E} = \emptyset$ be the set of edges.

for all unordered pairs $(F_1, F_2) \in \mathcal{F} \times \mathcal{F}$ such that $F_1 \neq F_2$ **do**

Draw a line through the points F_1 and F_2. This will intersect the curve $\mathcal{C}$ at two other points L_1 and L_2.

if L_1 and L_2 are not both $\mathbb{F}_q$-rational **then**

Move on to the next pair.

end if

if $L_1 \in \mathbf{V} \cup \mathcal{F}$ and $L_2 \notin \mathbf{V} \cup \mathcal{F}$ **then**

Add a vertex to $\mathbf{T}$ labeled L_2.

If $L_1 \in \mathbf{V}$ draw an edge to $\mathbf{T}$ connecting L_1 and L_2 and label the edge with the linear relation $[F_1] + [F_2] + [L_1] + [L_2] = 0$. If $L_1 \in \mathcal{F}$ draw an edge to $\mathbf{T}$ connecting $*$ and L_2 and label the edge with the linear relation $[F_1] + [F_2] + [L_1] + [L_2] = 0$.

end if
if $\#\mathbf{V} \geq \lceil q^{3/4} \rceil$ **then**
break (tree building succeeded)
end if
end for
if $\#\mathbf{V} < \lceil q^{3/4} \rceil$ **then**
Tree building failed. Restart the algorithm.
end if

Relation search:
for all unordered pairs $(F_1, F_2) \in \mathcal{F} \times \mathcal{F}$, $F_1 \neq F_2$, that were not already used in tree building **do**
Draw a line through the points F_1 and F_2. This will intersect the curve $\mathcal{C}$ at two other points L_1 and L_2.
if L_1 and L_2 are not both $\mathbb{F}_q$-rational **then**
Move on to the next pair.
end if
if $L_1, L_2 \in \mathbf{V} \cup \mathcal{F}$ **then**
If one or both of L_1, L_2 is in $\mathbf{V}$ use the tree and the linear relations labeling the edges to write
$$[F_1] + [F_2] + [L_1] + [L_2] = \sum_{F \in \mathcal{F}} \lambda_F [F] = 0\,.$$
Record this relation. We call it a *full relation*.
end if
if the number of full relations found is $\geq \#\mathcal{F} + 1$ **then**
break (relation search succeeded)
end if
end for
if the number of full relations is $\leq \#\mathcal{F}$ **then**
Restart the algorithm.
end if

Linear algebra step:
Construct a sparse matrix M with $\#\mathcal{F}$ columns, each column labeled by a point of $\mathcal{F}$.
The first row of M is given by the points in the divisor $D_1 - 3[P_0]$.
The second row of M is given by the points in the divisor $D_2 - 3[P_0]$.
for each full relation **do**
Add a row to M corresponding to the left-hand side of the full relation.
end for
Use sparse linear algebra techniques to find a vector $\overline{v}$ such that $M\overline{v} = \overline{0}$ over the ring $\mathbb{Z}/p\mathbb{Z}$ where p is the order of the subgroup of the Jacobian where the DLP was defined. Choose $\overline{v}$ to be such that the second component v_2 is invertible modulo p.

Output: $-v_1/v_2$ modulo p.

Theorem 1 ([Die2]). *Under some heuristic assumptions and when q is large enough, after a constant number of attempts the algorithm terminates successfully and outputs a number $x \in \mathbb{Z}/p\mathbb{Z}$ such that $D_2 - 3[P_0] = x \cdot (D_1 - 3[P_0])$. The complexity of the algorithm is $\widetilde{O}(q)$.*

3 Our Variant

In our version of the algorithm we generate the factor base in a different way. We start with a much smaller initial set for the factor base of size $O(q^{1/8})$, and then build up the factor base and the graph simultaneously at the beginning. This improves the efficiency of the graph-building, and then after this initial step we build the graph and find full relations simultaneously. As a result, both the overall relation collection time and the linear algebra stage are improved. Our algorithm works as follows.

Input:
1) The Jacobian of a smooth plane quartic $\mathcal{C}$ over a finite field $\mathbb{F}_q$;
2) An $\mathbb{F}_q$-rational point P_0 on the curve;

3) A discrete logarithm problem on the Jacobian

$$D_2 - 3[P_0] = x \cdot (D_1 - 3[P_0])$$

where $\deg(D_1) = \deg(D_2) = 3$ and both D_1 and D_2 are sums of three $\mathbb{F}_q$-rational points:

$$D_1 = [P_1^1] + [P_2^1] + [P_3^1]\,, \qquad D_2 = [P_1^2] + [P_2^2] + [P_3^2]\,;$$

4) The size p of a prime order subgroup[3] containing $D_2 - 3[P_0]$ and $D_1 - 3[P_0]$.

Initialization: Let λ be a positive real number satisfying

$$\lambda \exp\left(4\lambda^8\right) = q^{1/8}\,.$$

Choose a set $\mathcal{RP}$ of $\lceil 4\lambda\, q^{1/8} \rceil$ $\mathbb{F}_q$-rational points on the curve. Let $\mathcal{F}$ be another set of points, the *factor base*, and for now let

$$\mathcal{F} := \mathcal{RP} \cup \{P_j^i\} \cup \{P_0\}\,.$$

Construction of the base vertices: We construct a graph $\mathbf{G}$ as follows.

Let $\mathbf{V} := \{*\}$ be the set of vertices of the graph $\mathbf{G}$. Here $*$ denotes a distinguished *special vertex*. Let $\mathbf{E} := \emptyset$ be the set of edges.

for all unordered pairs $(F_1, F_2) \in \mathcal{RP} \times \mathcal{RP}$ such that $F_1 \neq F_2$ **do**

Draw a line through the points F_1 and F_2. This will intersect the curve $\mathcal{C}$ at two other points F and L.

if F and L are $\mathbb{F}_q$-rational, $L \notin \mathbf{V} \cup \mathcal{F}$ and $F \notin \mathbf{V}$ **then**

Let $\mathcal{F} := \{F\} \cup \mathcal{F}$ and add a vertex to $\mathbf{V}$ labeled L.

Add an edge to $\mathbf{E}$ connecting $*$ and L and label the edge with the linear relation $[F_1] + [F_2] + [F] + [L] = 0$. The vertices L constructed here we call *base vertices*.

end if

end for

Let $\mathbf{B}$ denote the set of all base vertices. Note that at this point $\mathbf{V} = \mathbf{B} \cup \{*\}$.

Construction of the triangle relations:

for all unordered pairs $(B_1, B_2) \in \mathbf{B} \times \mathbf{B}$ such that $B_1 \neq B_2$ **do**

Draw a line through the points B_1 and B_2. This will intersect the curve $\mathcal{C}$ at two other points F and L.

if F and L are $\mathbb{F}_q$-rational, $L \notin \mathbf{V} \cup \mathcal{F}$ and $F \notin \mathbf{V}$ **then**

Let $\mathcal{F} := \{F\} \cup \mathcal{F}$ and add a vertex to $\mathbf{V}$ labeled L.

Draw a triangle in the graph with corners B_1, B_2 and L. Label the triangle with the linear relation $[B_1] + [B_2] + [L] + [F] = 0$. The vertices L constructed here we call *top triangle vertices*.

end if

end for

Graph building and relation search:

for all unordered pairs $(F_1, F_2) \in \mathcal{F} \times \mathcal{F}$ such that $F_1 \neq F_2$ **do**

Draw a line through the points F_1 and F_2. This will intersect the curve $\mathcal{C}$ at two other points L_1 and L_2.

if L_1 and L_2 are not both $\mathbb{F}_q$-rational **then**

Move on to the next pair.

end if

if $L_1 \in \mathbf{V} \setminus \mathbf{B}$ and $L_2 \notin \mathbf{V} \cup \mathcal{F}$ **then**

Add a vertex to $\mathbf{V}$ labeled L_2.

Add an edge to $\mathbf{E}$ connecting L_1 and L_2 and label the edge with the linear relation $[F_1] + [F_2] + [L_1] + [L_2] = 0$.

else if $L_1, L_2 \in (\mathbf{V} \setminus \mathbf{B}) \cup \mathcal{F}$ **then**

[3] It is not strictly speaking necessary for the subgroup to have prime order, but this will simplify the linear algebra step and guarantee that the DLP has a solution.

In case $L_1, L_2 \in \mathbf{V} \setminus \mathbf{B}$ use the graph and the linear relations labeling the edges to write

$$[F_1] + [F_2] + [L_1] + [L_2] = \lambda_1[L_1'] + \lambda_2[L_2'] + \sum_{F \in \mathcal{F}} \lambda_F[F] = 0$$

where L_1' and L_2' are top triangle vertices, λ_i are ± 1 and λ_F are integers. Then use the triangle relations to substitute L_1' and L_2' with elements of $\mathcal{F}$. In cases where neither one or exactly one of L_i is in $\mathbf{V} \setminus \mathbf{B}$ we need to perform the above substitution process only to the one that is in $\mathbf{V} \setminus \mathbf{B}$. In any case we obtain a relation involving only elements of $\mathcal{F}$. Record this relation. We call it a *full relation*.

end if

if the number of full relations found is $\geq \#\mathcal{F} + 1$ **then**

break (relation search succeeded)

end if

end for

if the number of full relations is $\leq \#\mathcal{F}$ **then**

Restart the algorithm.

end if

In practice we do not restart the algorithm but instead re-run the **for**-loop. Almost certainly it will **break** very soon after starting and only a few if any duplicate full relations are produced.

Linear algebra step:

Construct a sparse matrix M with $\#\mathcal{F}$ columns, each column labeled by a point of $\mathcal{F}$.

The first row of M is given by the points in the divisor $D_1 - 3[P_0]$.

The second row of M is given by the points in the divisor $D_2 - 3[P_0]$.

for each full relation **do**

Add a row to M corresponding to the left-hand side of the full relation

end for

Use sparse linear algebra techniques to find a vector $\overline{v}$ such that $M\overline{v} = \overline{0}$ over the ring $\mathbb{Z}/p\mathbb{Z}$ where p is the order of the subgroup of the Jacobian where the DLP was defined. Choose $\overline{v}$ to be such that the second component v_2 is invertible modulo p.

Output: $-v_1/v_2$ modulo p.

Theorem 2. *(Heuristic) Under some heuristic assumptions and if q is large enough, after a constant number of attempts the algorithm terminates and outputs a number $x \in \mathbb{Z}/p\mathbb{Z}$ such that $D_2 - 3[P_0] = x \cdot (D_1 - 3[P_0])$. The size of the factor base will be approximately $4\lambda^4\, q^{1/2}$ and the size of the graph will be approximately $4\lambda^2\, q^{3/4}$.*

Proof. Correctness is easy; see for example [Die2]. It remains to prove that the size of the factor base is sufficiently large for the algorithm to terminate. The strategy is the following. Let N be the size of the factor base at the beginning, essentially $N = \#\mathcal{RP}$. First we compute the number of factor base elements and graph vertices produced when constructing the base vertices and the triangle relations, and express these in terms of N. Next we compute the expected number of full relations produced in the graph building step when allowing the graph to grown until it has $N_{\max}$ vertices. Since we know that the number of full relations needed is $\#\mathcal{F} + 1$, we can find an expression for $N_{\max}$ in terms of N. Finally we compute the number of factor base pairs needed to grow the graph to size $N_{\max}$. We set this number equal to the number of factor base pairs available, which yields an equation for N.

Due to the roundabout way of computing the numbers N and $N_{\max}$, we need to have an idea of their sizes beforehand, so that the most significant terms can be computed. For this purpose, we assume $N = O(q^{1/8})$, so $\#\mathcal{F} = O(q^{1/2})$, and $N_{\max} = O(q^{3/4})$, which are in line with Diem's choices in [Die2].

By [DT, Prop. 14], a line through two $\mathbb{F}_q$-rational points on the curve intersects the curve at two other $\mathbb{F}_q$-rational points with probability $1/2 + O(q^{-1/2})$.

Suppose $\mathcal{RP}$ is a set of $N = O(q^{1/8})$ random $\mathbb{F}_q$-rational points on the curve. Construction of the base vertices $\mathbf{B}$ produces

$$\left(\frac{1}{2} + O(q^{-1/2})\right)\binom{N}{2} = \frac{N^2}{4} + O(q^{1/8})$$

base vertices and equally many factor base elements.

Construction of the triangle relations from the set $\mathbf{B}$ produces

$$\left(\frac{1}{2}+O(q^{-1/2})\right)\binom{\#\mathbf{B}}{2}=4\left(\frac{N}{4}\right)^4+O(q^{3/8})$$

triangles and equally many factor base elements. At this point we expect to have

$$\#\mathcal{F}=4\left(\frac{N}{4}\right)^4+O(q^{3/8})=4\left(\frac{N}{4}\right)^4\left(1+O(q^{-1/8})\right).$$

We will need to choose N so that in the graph building step we expect to find $\#\mathcal{F}+1$ full relations. To this end, suppose that when the algorithm terminates successfully the number of vertices in the graph is $N_{\max}=O(q^{3/4})$. When the graph building starts we already have approximately $4(N/4)^4$ vertices in the graph, consisting of the base vertices $\mathbf{B}$ and the top triangle vertices. If the size of the graph at a particular moment is x, the probability of adding a new vertex and edge to the graph with a particular choice of a pair $(F_1,F_2)\in\mathcal{F}\times\mathcal{F}$ is

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

Hence, to add one new vertex and edge we need to try approximately

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

pairs. For each pair we try, the probability of finding a full relation through the process described in the algorithm is

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

so when the size of the graph is increased by one we expect to have found approximately

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

more full relations.
Once the triangle building step has been completed, the size of the graph is roughly equal to the number of triangles, which again is roughly equal to the size of the factor base. We denote this by

$$N_0=4\left(\frac{N}{4}\right)^4+O(q^{3/8}).$$

Note that once the triangles have been constructed, the factor base does not change anymore and only the graph is built using the factor base. The total number of full relations produced in the entire graph building step, when the size of the graph is built from N_0 to $N_{\max}=O(q^{3/4})$, is

$$\left(\frac{1}{2}+O(q^{-1/2})\right)\sum_{k=N_0}^{N_{\max}}\frac{k/q}{1-k/q}\approx\left(\frac{1}{2}+O(q^{-1/2})\right)\int_{N_0}^{N_{\max}}\frac{x/q}{1-x/q}\,dx$$

$$=\left(\frac{q}{2}+O(q^{1/2})\right)\left[-\frac{N_{\max}}{q}-\ln\left(1-\frac{N_{\max}}{q}\right)+\frac{N_0}{q}+\ln\left(1-\frac{N_0}{q}\right)\right].$$

If we expand out the first two terms of the logarithms we see that this is

$$=\left(\frac{q}{2}+O(q^{1/2})\right)\left[\frac{1}{2}\left(\frac{N_{\max}}{q}\right)^2+O(q^{-3/4})\right]=\frac{q}{4}\left(\frac{N_{\max}}{q}\right)^2+O(q^{1/4}).$$

We want this to equal roughly the size of the factor base (or maybe slightly more in practice to ensure that the algorithm terminates successfully) so we must have

$$\frac{q}{4}\left(\frac{N_{\max}}{q}\right)^2=4\left(\frac{N}{4}\right)^4\left(1+O(q^{-1/8})\right).$$

From this we can solve

$$\frac{N_{\max}}{4q}=\left(\frac{N}{4q^{1/4}}\right)^2\left(1+O(q^{-1/8})\right)=\left(\frac{N}{4q^{1/4}}\right)^2+O(q^{-3/8}). \tag{1}$$

With this we are ready to compute the value of N. The number of unordered pairs of factor base elements needed to build the graph from size N_0 to size $N_{\max} = O(q^{3/4})$ should equal the number of pairs available, i.e. $\binom{\#\mathcal{F}}{2} = 8(N/4)^8 + O(q^{7/8})$. Therefore,

$$8\left(\frac{N}{4}\right)^8 + O(q^{7/8}) = \left(1 + O(q^{-1/2})\right) \sum_{k=N_0}^{N_{\max}} \frac{1}{(k/q)(1-k/q)}$$

$$\approx \left(1 + O(q^{-1/2})\right) \int_{N_0}^{N_{\max}} \frac{1}{(x/q)(1-x/q)}\, dx$$

$$= \left(q + O(q^{1/2})\right) \left[\ln\left(\frac{N_{\max}}{4q}\right) - \ln\left(1 - \frac{N_{\max}}{q}\right) - \ln\left(\frac{N_0}{4q}\right) + \ln\left(1 - \frac{N_0}{q}\right)\right]$$

$$= \left(q + O(q^{1/2})\right) \left[\ln\left(\frac{N_{\max}}{4q}\right) - \ln\left(\frac{N_0}{4q}\right) + O(q^{-1/4})\right]$$

$$= -2q \ln\left(\frac{N}{4q^{1/4}}\right) + O(q^{7/8})$$

where we have used (1). Hence we obtain

$$4\left(\frac{N}{4q^{1/8}}\right)^8 + O(q^{-1/8}) = -\ln\left(\frac{N}{4q^{1/4}}\right).$$

Denoting

$$\lambda = \frac{N}{4q^{1/8}}$$

this becomes

$$\lambda \exp\left(4\lambda^8\right) = q^{1/8} + O(1) \approx q^{1/8},$$

where the approximation makes sense when q is large enough. In practice the error term is small, even for small field sizes. The function $\lambda \exp\left(4\lambda^8\right) - q^{1/8}$ is monotonically increasing and has precisely one positive real root. This is the equation stated in the initialization step of the algorithm, so if we take $N = 4\lambda\, q^{1/8}$ the algorithm can be expected to terminate successfully.
If q is large, then the size of the factor base will be

$$\#\mathcal{F} = 4\lambda^4\, q^{1/2} + O(q^{3/8}) \approx 4\lambda^4\, q^{1/2}$$

and the size of the graph when the algorithm terminates will be

$$N_{\max} = 4\lambda^2\, q^{3/4} + O(q^{5/8}) \approx 4\lambda^2\, q^{3/4}.$$

For field sizes of most practical interest (perhaps between 60 and 120 bits) our algorithm has not much worse storage requirements than Diem's algorithm but our factor base remains smaller. In practice the factor base can be taken to be even slightly smaller than $4\lambda^4\, q^{1/2}$ if we run the graph building step twice in a row as was explained earlier. We will look at some precise numbers in the next section.

Specifics of Implementation in Characteristic 2

In our experiments we made an artificial restriction to the case of binary fields in order to be able to count the number of points on the Jacobian easily. In this case it is easy to perform the geometric step of the algorithm where a line is drawn through two points on the curve and the intersection divisor is computed. Indeed, we do this by first constructing an equation for the line, then solving for one of the variables in terms of the other ones and substituting this into the equation of the curve to obtain a quartic polynomial in one variable. It is a simple computation to divide this polynomial by the two linear factors corresponding to the two points we started with. Finally the quadratic equation can be transformed into an Artin-Schreier equation by a linear change of variables and the solutions can be immediately written down using Chen's formulas. The complexity is logarithmic in q.
For odd characteristic fields one has to compute a square-root in $\mathbb{F}_q$ using the Tonelli-Shanks algorithm to solve the quadratic polynomial, the complexity of which is logarithmic in q.

4 Complexity for Realistic Field Sizes

Relation Collection Time and Total Memory

In our naive implementation the number of field multiplications (in the binary field case) needed to process each pair of factor base elements as explained above was

$$M_{\text{pair}} = 7 \log_2 q + 13 \,.$$

This count is an actual count of how many field multiplications are used to process each pair of points in our code, but it is also a theoretical count of the number of field operations required to pass a line through two points and intersect it with the curve. There are field inversions involved, which roughly explains the $\log q$ factors. This number could likely be improved in a more robust implementation. The total number of field multiplications needed was therefore approximately

$$M_{\text{Total}} \approx M_{\text{pair}} \cdot \binom{\#\mathcal{F}}{2} = (7 \log_2 q + 13) \cdot 8\lambda^8 \, q \,.$$

Remark 1. Locally the numbers λ behave roughly logarithmically as a function of q. If we focus on the range $q \in [2^{70}, 2^{120}]$ we can for instance approximate

$$\lambda(q) \approx C \log_2^{\alpha} q \,, \quad \text{where} \quad C = 0.62054 \,, \quad \alpha = 0.12431 \,.$$

Then

$$M_{\text{Total}} \approx 0.17589 \cdot (7 \log_2 q + 13) \cdot \log_2^{0.99448}(q) \cdot q \approx 1.23123 \cdot \log_2^2(q) \cdot q \,.$$

The memory consumption was roughly 370 bytes per graph vertex but our implementation was not optimized towards saving memory so this number can probably be reduced by a significant factor. Moreover, every vertex data must contain the coordinates of some points on the curve or other vertex identifiers, the size of which grows logarithmically in q. Hence the size of a vertex will grow logarithmically in q, but this dependence is weak and for practical field sizes it is not a significant factor. To take this into account, we assume the size of a vertex data is $\lceil (\log_2 q)/64 \rceil \cdot 370$ bytes. The results are shown in Table 1.

Table 1: Results for field sizes of practical interest

Field size	λ	$\log_q \#\mathcal{F}$	$\log_q N_{\max}$	$\log_q M_{\text{Total}}$	$\log_2 M_{\text{Total}}$	Memory
2^{30}	0.94987	0.55677	0.81172	1.34024	40.20729	7.4 GB
2^{40}	0.98285	0.54750	0.79875	1.27488	50.99510	1430 GB
2^{50}	1.00974	0.54112	0.79056	1.23231	61.61568	267 TB
2^{60}	1.03251	0.53641	0.78487	1.20212	72.12744	50490 TB
2^{70}	1.05230	0.53277	0.78067	1.17947	82.56275	$1.90 \cdot 10^7$ TB
2^{80}	1.06983	0.52987	0.77743	1.16177	92.94144	$3.56 \cdot 10^9$ TB
2^{90}	1.08559	0.52749	0.77486	1.14752	103.27649	$6.62 \cdot 10^{11}$ TB
2^{100}	1.09992	0.52550	0.77275	1.13577	113.57690	$1.2 \cdot 10^{14}$ TB
2^{110}	1.11306	0.52380	0.77099	1.12590	123.84916	$2.28 \cdot 10^{16}$ TB
2^{115}	1.11926	0.52304	0.77022	1.12153	128.97628	$3.10 \cdot 10^{17}$ TB
2^{120}	1.12522	0.52234	0.76950	1.11748	134.09806	$4.22 \cdot 10^{18}$ TB
2^{140}	1.14712	0.51994	0.76711	1.10386	154.53975	$2.16 \cdot 10^{23}$ TB
2^{160}	1.16646	0.51805	0.76528	1.09327	174.92298	$7.29 \cdot 10^{27}$ TB
2^{180}	1.18380	0.51652	0.76382	1.08479	195.26141	$8.43 \cdot 10^{31}$ TB
2^{200}	1.19954	0.51525	0.76262	1.07782	215.56441	$1.10 \cdot 10^{37}$ TB
2^{220}	1.21397	0.51418	0.76163	1.07199	235.83869	$3.71 \cdot 10^{41}$ TB
2^{240}	1.22729	0.51326	0.76080	1.06704	256.08921	$1.24 \cdot 10^{46}$ TB

According to these estimates one should use a field of size at least 115 bits to get a security level of 128 bits and a field of size at least 240 bits to get a security level of 256 bits. Of course this is only the relation collection step and is not taking into account the linear algebra step or even more importantly the massive memory consuption. Later we will see how parallelization reduces both time and memory requirements (per computer).

Experimental Results for Small Examples

We ran small experiments and got results corresponding to the theoretical results above. We chose $\mathcal{RP}$ to be slightly bigger than was strictly speaking needed to ensure that the algorithm finishes successfully on the first attempt. More precisely, we chose it so that about 95% of the factor base pairs are used when the algorithm finishes. The results are shown in Table 2. We conclude that the experimental results correspond closely to the theoretical results, even for such small field sizes.

Field size	$\log_q \#\mathcal{F}$ (theory)	$\log_q \#\mathcal{F}$ (practice)	FBPU	$\log_q M_{\text{Total}}$ (theory)	$\log_q M_{\text{Total}}$ (practice)
2^{17}	0.57846	0.58034	96.4	1.51247	1.50780
2^{19}	0.57387	0.57692	95.5	1.47352	1.46740
2^{21}	0.56683	0.57223	95.3	1.44080	1.43510
2^{23}	0.56637	0.56819	95.4	1.41287	1.40769
2^{25}	0.56325	0.56468	96.0	1.38869	1.38418
2^{27}	0.56046	0.56158	96.4	1.36752	1.36351

Table 2: Experimental results. (FBPU = percent of factor base pairs used)

Linear Algebra

We recall that the complexity of sparse linear algebra algorithms is $O\left(w \cdot (\#\mathcal{F})^2\right)$ where w denotes the average row weight. The row weight depends logarithmically on the field size. The expected average depth of a tree on a given number of vertices can be estimated using the method in [GTTD] but we need to take into account that the size of the graph is changing while we are looking for full relations.

Lemma 1. *The average row weight of the matrix is* $w \leq 18 - 8\ln\lambda + \ln q$.

Proof. In [GTTD] it is established that for a tree containing k vertices the expected average depth can be approximated from above by the function $1 + \ln k$. Our graph consists of trees built on top of each of the top triangle vertices. When our graph has a total of k vertices, each one of these $4\lambda^4 q^{1/2} + O(q^{3/8})$ trees has on average $k/(4\lambda^4 q^{1/2}) + O(q^{-1/8})$ vertices. So if we choose a tree at random and a point in it at random, the expected depth (measured from the root of that tree) is

$$\leq 1 + \ln\left(\frac{k}{4\lambda^4 q^{1/2}}\right) + O\left(\frac{q^{3/8}}{k}\right) = 1 + \ln\left(\frac{k}{4\lambda^4 q^{1/2}}\right) + O(q^{-3/8}).$$

We find full relations while building the graph so the sizes of the trees change. Recall that right after the triangles are constructed, the size of the graph is

$$N_0 = 4\lambda^4 q^{1/2} + O(q^{3/8}).$$

When a full relation is produced and the graph tracing step performed, the number of factor base elements we end up with on average is[4]

$$\begin{aligned}
&\leq \frac{2^2}{\#\mathcal{F}} \sum_{k=N_0}^{N_{\max}} \left(1 + \ln\left(\frac{k}{4\lambda^4 q^{1/2}}\right) + O(q^{-3/8})\right) \cdot \left(\frac{1}{2} + O(q^{-1/2})\right) \frac{k/q}{1 - k/q} \\
&\approx \frac{q^{1/2}}{2\lambda^4} \left(1 + O(q^{-1/8})\right) \int_{N_0/q}^{N_{\max}/q} \left(1 + \ln\left(\frac{t\, q^{1/2}}{4\lambda^4}\right) + O(q^{-3/8})\right) \cdot \frac{t}{1-t}\, dt \\
&= \frac{q^{1/2}}{2\lambda^4} \left(1 + O(q^{-1/8})\right) \left(1 + O(q^{-1/4})\right) \int_{N_0/q}^{N_{\max}/q} t \left(1 + \ln\left(\frac{t\, q^{1/2}}{4\lambda^4}\right) + O(q^{-3/8})\right) dt \\
&= 4 \left(1 + O(q^{-1/8})\right) \left(\frac{1}{2} - 2\ln\lambda + \frac{1}{4}\ln q + O(q^{-1/8})\right) + O(q^{-1/2}) \\
&= 2 - 8\ln\lambda + \ln q + O(q^{-1/8}\ln q).
\end{aligned}$$

In addition to the contribution coming from moving in the graph, we have 2 initial factor base elements, 2 coming from using the triangle relations and 12 from the base vertices. The expected average row weight should therefore satisfy $w \leq 18 - 8\ln\lambda + \ln q$.

[4] One factor of 2 is for the two factor base elements that are produced at every step in the graph and the other factor of 2 from tracing back two vertices to their respective roots. Here we are only concerned with the factor base elements that result from moving in the graph. The others are taken into account below.

Note that in practice q is large so $q^{-1/8}\ln q$ is small, and we let $w_{\text{est}} = 18 - 8\ln\lambda + \ln q$. Results for field sizes of practical interest are shown in Table 3. The complexity of the linear algebra is in all cases slightly better than the complexity of matrix building so there might be some room for optimization by choosing the factor base to be slightly larger, which makes relation collection faster and linear algebra slower.

Table 3: Linear algebra results for field sizes of practical interest

Field size	$\log_2 w_{\text{est}}$	$\log_2 \#\mathcal{F}$	Lin. alg.	M_{Total}
2^{30}	5.29300	16.70320	$\approx 2^{39}$	$\approx 2^{40}$
2^{40}	5.51930	21.90017	$\approx 2^{49}$	$\approx 2^{51}$
2^{50}	5.71644	27.05593	$\approx 2^{60}$	$\approx 2^{62}$
2^{60}	5.89076	32.18461	$\approx 2^{70}$	$\approx 2^{72}$
2^{70}	6.04685	37.29417	$\approx 2^{81}$	$\approx 2^{83}$
2^{80}	6.18808	42.38952	$\approx 2^{91}$	$\approx 2^{93}$
2^{90}	6.31698	47.47391	$\approx 2^{101}$	$\approx 2^{103}$
2^{100}	6.43551	52.54957	$\approx 2^{112}$	$\approx 2^{114}$
2^{110}	6.54518	57.61814	$\approx 2^{122}$	$\approx 2^{124}$
2^{115}	6.59709	60.15016	$\approx 2^{127}$	$\approx 2^{129}$
2^{120}	6.64723	62.68083	$\approx 2^{132}$	$\approx 2^{134}$
2^{140}	6.83216	72.79205	$\approx 2^{152}$	$\approx 2^{155}$
2^{160}	6.99630	82.88852	$\approx 2^{173}$	$\approx 2^{175}$
2^{180}	7.14381	92.97370	$\approx 2^{193}$	$\approx 2^{195}$
2^{200}	7.27774	103.04993	$\approx 2^{213}$	$\approx 2^{216}$
2^{220}	7.40038	113.11892	$\approx 2^{234}$	$\approx 2^{236}$
2^{240}	7.51347	123.18192	$\approx 2^{254}$	$\approx 2^{256}$

5 Comparison with Diem's Algorithm

Case 1: Free Graph Size

In Diem's algorithm (see [Die2]) the size of the factor base should be taken to be at least

$$\#\mathcal{F}_{\text{Diem}} = \lceil ((3/2)\ln q + 4)^{1/2}\, q^{1/2} \rceil .$$

We can compare this to the size of our factor base $\#\mathcal{F} \approx 4\lambda^4\, q^{1/2}$. Using the defining equation of λ to write $\ln q = 8\ln\lambda + 32\lambda^8$ we find

$$\rho_{\text{MB}} = \left(\frac{\#\mathcal{F}_{\text{Diem}}}{\#\mathcal{F}}\right)^2 = 3 + \frac{3\ln\lambda + 1}{4\lambda^8} .$$

When q increases λ increases and so this ratio approaches 3.

Lemma 2. *The ratio of the complexity of the matrix building step in our algorithm and the complexity of the matrix building step in Diem's algorithm approaches* 3 *as the size of the field increases.*
The ratio of the complexity of the linear algebra step in our algorithm and the complexity of the linear algebra step in Diem's algorithm approaches 9 *as the size of the field increases.*

Proof. In both algorithms the sizes of the factor bases have been chosen so that the algorithm can just barely terminate. Thus almost all pairs of factor base elements have to be used which means that the ratio of the complexities in the matrix building step is roughly

$$\frac{\binom{\#\mathcal{F}_{\text{Diem}}}{2}}{\binom{\#\mathcal{F}}{2}} \approx \left(\frac{\#\mathcal{F}_{\text{Diem}}}{\#\mathcal{F}}\right)^2$$

which approaches 3 as q grows.
The complexity of the linear algebra step depends quadratically on #FB and linearly on the average row weight. In Diem's algorithm the expected average depth of the tree is approximately $1 + \ln\left(q^{3/4}\right) = 1 + (3/4)\ln q$ and so the average row weight is $2 + 4(1 + (3/4)\ln q) = 6 + 3\ln q$ whereas in our algorithm it is generally smaller, $18 - 8\lambda + \ln q$. Hence the ratio of the complexities in the linear algebra step is roughly

$$\rho_{\text{LA}} := \frac{6 + 3\ln q}{18 - 8\ln\lambda + \ln q}\left(\frac{\#\mathcal{F}_{\text{Diem}}}{\#\mathcal{F}}\right)^2$$

which approaches $3 \cdot 3 = 9$.

Denote the ratios in the above lemma by ρ_{MB} and ρ_{LA}. Let ρ_{MEM} denote the ratio in memory consumption which we measure by the maximum size of the graph. These numbers for field sizes of practical interest are presented in Table 4.

Table 4: Comparison with Diem

Field size q	ρ_{MB}	ρ_{LA}	$1/\rho_{\mathrm{MEM}}$
2^{30}	3.31904	5.78911	3.60900
2^{40}	3.27221	6.36242	3.86398
2^{50}	3.23808	6.77252	4.07829
2^{60}	3.21213	7.07935	4.26429
2^{70}	3.19171	7.31689	4.42932
2^{80}	3.17519	7.50579	4.57814
2^{90}	3.16153	7.65933	4.71400
2^{100}	3.15004	7.78637	4.83926
2^{110}	3.14022	7.89309	4.95564
2^{115}	3.13582	7.94026	5.01093
2^{120}	3.13172	7.98390	5.06448
2^{140}	3.11772	8.12990	5.26351
2^{160}	3.10664	8.24185	5.44247
2^{180}	3.09763	8.33019	5.60552
2^{200}	3.09015	8.40153	5.75560
2^{220}	3.08383	8.46025	5.89487
2^{240}	3.07841	8.50934	6.02500

As we have pointed out earlier, in our algorithm it is possible to take the factor base to be slightly smaller and re-run the graph building/relation searching step once it has failed for the first time. At that point the graph is huge and it should not take long to find the missing full relations. This is not possible in Diem's approach. We also save time because we only need to find roughly $q^{1/8}$ random points on the curve compared to over $q^{1/2}$ which have to be found in Diem's algorithm. Of course we then need to compute more intersection divisors to find the remaining factor base elements.

Case 2: Fixed Graph Size

Another interesting way to compare these algorithms is to see what happens when memory consumption is fixed. Suppose we let our graph grow to size $q^{3/4}$ and after that only search for full relations. More precisely, we start by choosing a set $\mathcal{RP}_{\mathrm{Res}}$ of $4\eta\, q^{1/8}$ random points on the curve where η is a real number and use these to construct a factor base $\mathcal{F}_{\mathrm{Res}}$ of size $4\eta^4\, q^{1/2}$ just as in our original algorithm. We expect η to be somewhere between 1 and 10 for practical field sizes. In the graph building/relation searching step we stop adding new vertices to the graph once it has reached size $q^{3/4}$ and after that only search for full relations.

Theorem 3. *(Heuristic) If q is large enough and we take η to be a root of*

$$2\eta^2 \exp\left(4\eta^8 - 4\eta^4 + 1/4\right) = q^{1/8}$$

we can expect the algorithm to terminate successfully. The average row weight of the matrix will be approximately

$$20 + \frac{1}{8\eta^4} - 4\ln(4\eta^4) + \ln q\,.$$

Proof. This is very similar to the proof of Theorem 2. We start with a set $\mathcal{RP}_{\mathrm{Res}}$ of $4\eta\, q^{1/8} = O(q^{1/8})$ random points. The size of the factor base and the number of vertices in the graph after the triangles have been constructed are

$$\#\mathcal{F}_{\mathrm{Res}} = 4\eta^4\, q^{1/2} + O(q^{3/8})\,, \qquad N_0 = 4\eta^4\, q^{1/2} + O(q^{3/8})\,.$$

We build the graph until it has size $q^{3/4}$. This produces

$$\left(\frac{1}{2} + O(q^{-1/2})\right) \int_{N_0}^{q^{3/4}} \frac{x/q}{1 - x/q}\, dx = \frac{q^{1/2}}{4} + O(q^{-3/4})$$

full relations (see the proof of Theorem 2). The total number of full relations needed is $\#\mathcal{F}_{\text{Res}}+1$ so we are lacking

$$4\eta^4\, q^{1/2} - \frac{q^{1/2}}{4} + O(q^{3/8}) = \left(4\eta^4 - \frac{1}{4}\right) q^{1/2} + O(q^{3/8})$$

of them. We also need to know how many pairs of factor base elements were used in this. Just like in the proof of Theorem 2 we find that this number is

$$\left(1 + O(q^{-1/2})\right) \int_{N_0}^{q^{3/4}} \frac{1}{(x/q)(1-x/q)}\, dx = q \ln\left(\frac{q^{1/4}}{4\eta^4}\right) + O(q^{7/8})\,.$$

Now once the graph has size $q^{3/4}$ we need to find the rest of the full relations. For each pair of factor base elements we try, the probability of finding a full relation is now approximately $1/(2q^{1/2}) + O(q^{-1})$ (see the proof of Theorem 2). Thus we need to try

$$2q^{1/2}\left(1 + O(q^{-1/2})\right) \cdot \left[\left(4\eta^4 - \frac{1}{4}\right) q^{1/2} + O(q^{3/8})\right] = \left(8\eta^4 - \frac{1}{2}\right) q + O(q^{7/8})$$

more factor base pairs. We want to end up using precisely all of the factor base pairs, of which there are $\binom{\#\mathcal{F}_{\text{Res}}}{2} = 8\eta^8\, q + O(q^{7/8})$. We get the equation

$$8\eta^8\, q + O(q^{7/8}) = q \ln\left(\frac{q^{1/4}}{4\eta^4}\right) + \left(8\eta^4 - \frac{1}{2}\right) q + O(q^{7/8})\,.$$

Exponentiating this yields

$$2\eta^2\, \exp(4\eta^8 - 4\eta^4 + 1/4) = q^{1/8} + O(1)\,.$$

When q is big we can approximate this with

$$2\eta^2\, \exp(4\eta^8 - 4\eta^4 + 1/4) = q^{1/8}\,.$$

As in the proof of Lemma 1 we find that the average row weight is approximated from above by

$$\begin{aligned} 16 + \frac{2^2}{\#\mathcal{F}_{\text{Res}}} \Bigg[&\sum_{k=N_0}^{q^{3/4}} \left(1 + \ln\left(\frac{k}{4\eta^4\, q^{1/2}}\right) + O(q^{-3/8})\right) \cdot \left(\frac{1}{2} + O(q^{-1/2})\right) \frac{k/q}{1-k/q} \\ &+ \left(1 + \ln\left(\frac{q^{3/4}}{4\eta^4\, q^{1/2}}\right) + O(q^{-3/8})\right) \left(\left(4\eta^4 - \frac{1}{4}\right) q^{1/2} + O(q^{3/8})\right)\Bigg] \\ = 16 + \frac{\ln q}{16\eta^4} - \frac{\ln(4\eta^4)}{4\eta^4} + \frac{1}{8\eta^4} &+ \left(1 - \frac{1}{16\eta^4}\right)\left(4 - 4\ln(4\eta^4) + \ln q\right) + O(q^{-1/8}\ln q) \\ &= 20 + \frac{1}{8\eta^4} - 4\ln(4\eta^4) + \ln q + O(q^{-1/8}\ln q)\,. \end{aligned}$$

Corollary 1. *Lemma 2 holds even if we restrict the size of the graph to $q^{3/4}$.*

Proof. We find

$$\left(\frac{\#\mathcal{F}_{\text{Diem}}}{\#\mathcal{F}_{\text{Res}}}\right)^2 = 3 + \frac{7 + 12\ln(2\eta^2) - 48\eta^4}{16\eta^8}$$

which approaches 3 and

$$\frac{6 + 3\ln q}{20 + 1/(8\eta^4) - 4\ln(4\eta^4) + \ln q} \left(\frac{\#\mathcal{F}_{\text{Diem}}}{\#\mathcal{F}_{\text{Res}}}\right)^2$$

which approaches $3 \cdot 3 = 9$.

In practice things are not quite that good. The values for the ratios ρ_{MB} (complexities of matrix building steps) and ρ_{LA} (linear algebra step) in this memory restricted setting for field sizes of practical interest are presented in Table 5.

Table 5: Comparison with Diem, Memory restricted

Field size q	ρ_{MB}	ρ_{LA}	$1/\rho_{\mathrm{MEM}}$
2^{30}	1.31190	2.61456	1
2^{40}	1.42304	3.10424	1
2^{50}	1.51028	3.49559	1
2^{60}	1.58162	3.81649	1
2^{70}	1.64164	4.08534	1
2^{80}	1.69323	4.31463	1
2^{90}	1.73829	4.51311	1
2^{100}	1.77818	4.68707	1
2^{110}	1.81388	4.84116	1
2^{115}	1.83038	4.91189	1
2^{120}	1.84610	4.97890	1
2^{140}	1.90224	5.21555	1
2^{160}	1.94980	5.41253	1
2^{180}	1.99087	5.57988	1
2^{200}	2.02685	5.72440	1
2^{220}	2.05876	5.85091	1
2^{240}	2.08734	5.96288	1

The Full Algorithm

There is another variant of Diem's algorithm, namely the *full algorithm* approach of Diem and Thomé [DT] (see [GTTD] for the hyperelliptic case). In this algorithm a very large graph is constructed, of the size $q^{5/6}$. Consequently the factor base can be chosen to be smaller, of the size $2q^{1/2}$. The graph is disconnected and is searched for cycles which are then used to produce full relations. The complexity analysis of the *full algorithm* is difficult and the large graph size makes is impractical for all but the smallest examples so we will not discuss it further here.

6 Parallelization

In this section we study how the work of computing the DLP can be split between K computers. It turns out that with an increase in total computing time and total memory cost we can be split the computation between these K computers so that each needs to pay only a fraction of the original memory cost.
Suppose ϵ is a real number and we start by choosing a set $\mathcal{RP}_{\mathrm{Par}}$ of $4\epsilon\, q^{1/8}$ random points on the curve and as usual use these to generate a factor base $\mathcal{F}_{\mathrm{Par}}$ of size $4\epsilon^4\, q^{1/2}$. As usual at this point the graph will contain $4\epsilon^4\, q^{1/2}$ triangles. Now distribute the entire factor base and the base vertices to each one of the K computers and split the triangles evenly between them so that each computer gets $4\epsilon^4\, q^{1/2}/K$ triangles. Each computer starts building a graph and searching for full relations using their share of the triangles until they have each found approximately $4\epsilon^4\, q^{1/2}$ full relations. Now all full relations are combined into a matrix and the linear algebra step is performed as usual.

Theorem 4. *(Heuristic) If q is large enough and we take ϵ to be a root of*

$$\epsilon \exp\left(4\epsilon^8\right) = K^{1/4}\, q^{1/8}$$

we can expect the algorithm to terminate successfully. Each of the K computers will end up with a graph of size

$$N_{max} = \frac{4\epsilon^2\, q^{3/4}}{\sqrt{K}}\,.$$

The average row weight of the matrix will be approximately $18 + 2\ln K - 8\ln\epsilon + \ln q$.

Proof. The proof is again similar to the proof of Theorem 2. We start with a set $\mathcal{RP}_{\mathrm{Par}}$ of $4\epsilon\, q^{1/8} = O(q^{1/8})$ random points and suppose that $N_{\max} = O(q^{3/4})$. The size of the factor base and the number of vertices in the graph after the triangles have been constructed are

$$\#\mathcal{F}_{\mathrm{Par}} = 4\epsilon^4\, q^{1/2} + O(q^{3/8})\,, \qquad N_0 = 4\epsilon^4\, q^{1/2} + O(q^{3/8})\,.$$

Building the graphs from size N_0/K to $N_{\max}$ on each computer produces

$$\left(\frac{1}{2}+O(q^{-1/2})\right)\sum_{k=N_0/K}^{N_{\max}}\frac{k/q}{1-k/q}=\frac{N_{\max}^2}{4q}+O(q^{1/4})$$

full relations. But each computer should produce $\#\mathcal{F}_{\mathrm{Par}}/K=4\epsilon^4\,q^{1/2}/K+O(q^{3/8})$ full relations, so we get an equation and solve

$$N_{\max}=\frac{4\epsilon^2\,q^{3/4}}{K^{1/2}}\left(1+O(q^{-1/8})\right)=\frac{4\epsilon^2\,q^{3/4}}{\sqrt{K}}+O(q^{5/8})\,.$$

The number of pairs of factor base elements that each computer has is $\binom{\#\mathcal{F}_{\mathrm{Par}}}{2}=8\epsilon^8\,q+O(q^{7/8})$. We want this number to equal the number of pairs needed to build the graphs as explained above. Hence we need

$$8\epsilon^8\,q+O(q^{7/8})=\left(1+O(q^{-1/2})\right)\sum_{k=N_0/K}^{N_{\max}}\frac{1}{(k/q)(1-k/q)}=q\ln\left(\frac{K^{1/2}q^{1/4}}{\epsilon^2}\right)+O(q^{7/8})$$

which gives the equation

$$\epsilon\,\exp\left(4\epsilon^8\right)=K^{1/4}\,q^{1/8}+O(1)\,.$$

Finally, the average row weight is approximated from above by

$$16+\frac{2^2}{\#\mathcal{F}_{\mathrm{Par}}/K}\sum_{k=N_0/K}^{N_{\max}}\left(1+\ln\left(\frac{kK}{4\epsilon^4\,q^{1/2}}\right)+O(q^{-3/8})\right)\cdot\left(\frac{1}{2}+O(q^{-1/2})\right)\frac{k/q}{1-k/q}$$

$$=18+2\ln K-8\ln\epsilon+\ln q+O(q^{-1/8}\ln q)\,.$$

Remark 2. In the above proof we were only concerned with the asymptotic behavior as q grows and in particular some of the O-estimates do not make sense unless $K\ll q^{1/8}$.

Above we assumed that each one of the computers computes the same $8\epsilon^8\,q$ intersection divisors but of course this is completely unnecessary if there is a very efficient way for the computers to communicate with each other. In this case each one computes $\left(1/2+O(q^{-1/2}\right)8\epsilon^8\,q/K$ intersection divisors and shares their results with the other $K-1$ computers. Unfortunately this is a massive amount of data to share. In fact, merely storing all these intersection divisors costs potentially much more memory than storing the graph since

$$\frac{4\epsilon^8\,q}{K}\gg\frac{4\epsilon^2\,q^{3/4}}{\sqrt{K}}$$

for practical field sizes unless K is impossibly large. Hence instead of storing the intersection divisors they should be streamed to all other computers as soon as they are generated and then deleted immediately afterwards. This speeds up the algorithm only if the connections between the computers are so fast that distributing the data over the network is faster than for each computer to generate it separately. We assume this is the case.
Let $M_{\mathrm{Total}}^{\mathrm{Par}}$ be the total number of field multiplications needed per computer. We have

$$M_{\mathrm{Total}}^{\mathrm{Par}}=M_{\mathrm{pair}}\cdot\frac{1}{K}\binom{\#\mathcal{F}_{\mathrm{Par}}}{2}\approx(7\log_2 q+13)\cdot\frac{8\epsilon^8\,q}{K}\,.$$

We denote by ρ_{LA} the ratio of the complexities of the linear algebra stages in the parallelized and unparallelized algorithms. We see that

$$\rho_{\mathrm{LA}}=\frac{18+2\ln K-8\ln\epsilon+\ln q}{18+\ln q}\left(\frac{\epsilon}{\lambda}\right)^8\,.$$

We denote

$$\text{MPC (Memory Per Computer)}=\frac{4\epsilon^2\,q^{3/4}}{\sqrt{K}}\cdot(\lceil(\log_2 q)/64\rceil\cdot 370\text{ bytes})$$

and

$$\text{IDPC (Intersection Divisors Per Computer with }\mathbb{F}_q\text{-rational points)}=\frac{4\epsilon^8\,q}{K},$$

Table 6: Parallelization: $K = 4$

Field size q	$\log_2 M_{\text{Total}}^{\text{Par}}$	ρ_{LA}	MPC	$\log_2$ IDPC
2^{30}	38.37745	1.21411	3.8 GB	29.57655
2^{40}	49.12762	1.16379	731 GB	39.93286
2^{50}	59.72405	1.13164	136 TB	50.22023
2^{60}	70.21903	1.10941	25648 TB	60.46080
2^{70}	80.64203	1.09317	$9.62 \cdot 10^{6}$ TB	70.66761
2^{80}	91.01129	1.08082	$1.80 \cdot 10^{9}$ TB	80.84890
2^{90}	101.3389	1.07114	$3.35 \cdot 10^{11}$ TB	91.01023
2^{100}	111.63331	1.06336	$6.21 \cdot 10^{13}$ TB	101.15555
2^{110}	121.90060	1.05698	$1.15 \cdot 10^{16}$ TB	111.28773
2^{115}	127.02555	1.05420	$1.56 \cdot 10^{17}$ TB	116.34960
2^{120}	132.14535	1.05166	$2.13 \cdot 10^{18}$ TB	121.40894
2^{140}	152.58044	1.04333	$1.09 \cdot 10^{23}$ TB	141.62479
2^{160}	172.95868	1.03711	$3.67 \cdot 10^{27}$ TB	161.81275
2^{180}	193.29321	1.03232	$1.24 \cdot 10^{32}$ TB	181.97919
2^{200}	213.59307	1.02852	$5.56 \cdot 10^{36}$ TB	202.12853
2^{220}	233.86478	1.02544	$1.86 \cdot 10^{41}$ TB	222.26393
2^{240}	254.11314	1.02289	$6.24 \cdot 10^{45}$ TB	242.38778

Table 7: Parallelization: $K = 100$

Field size q	$\log_2 M_{\text{Total}}^{\text{Par}}$	ρ_{LA}	MPC	$\log_2$ IDPC
2^{60}	65.76817	1.39707	5304 TB	56.00995
2^{70}	76.16726	1.34197	$1.98 \cdot 10^{6}$ TB	66.19284
2^{80}	86.51785	1.30004	$3.69 \cdot 10^{8}$ TB	76.35545
2^{90}	96.83049	1.26708	$6.85 \cdot 10^{10}$ TB	86.50181
2^{100}	107.11261	1.24051	$1.27 \cdot 10^{13}$ TB	96.63485
2^{110}	117.36965	1.21863	$2.35 \cdot 10^{15}$ TB	106.75678
2^{115}	122.49008	1.20909	$3.19 \cdot 10^{16}$ TB	111.81413
2^{120}	127.60571	1.20032	$4.33 \cdot 10^{17}$ TB	116.86931
2^{140}	148.02689	1.17141	$2.20 \cdot 10^{22}$ TB	137.07124
2^{160}	168.39449	1.14963	$7.45 \cdot 10^{26}$ TB	157.24856
2^{180}	188.72061	1.13265	$2.51 \cdot 10^{31}$ TB	177.40660
2^{200}	209.01367	1.11905	$1.12 \cdot 10^{36}$ TB	197.54912
2^{220}	229.27975	1.10791	$3.77 \cdot 10^{40}$ TB	217.67891
2^{240}	249.52340	1.09863	$1.26 \cdot 10^{45}$ TB	237.79803

Table 8: Parallelization: $K = 1000$

Field size q	$\log_2 M_{\text{Total}}^{\text{Par}}$	ρ_{LA}	MPC	$\log_2$ IDPC
2^{80}	83.29477	1.46929	$1.19 \cdot 10^{8}$ TB	73.13238
2^{90}	93.59828	1.41737	$2.20 \cdot 10^{10}$ TB	83.26961
2^{100}	103.87281	1.37563	$4.07 \cdot 10^{12}$ TB	93.39505
2^{110}	114.12344	1.34135	$7.52 \cdot 10^{14}$ TB	103.51057
2^{115}	119.24103	1.32642	$1.02 \cdot 10^{16}$ TB	108.56507
2^{120}	124.35401	1.31272	$1.39 \cdot 10^{17}$ TB	113.61761
2^{140}	144.76630	1.26760	$7.05 \cdot 10^{21}$ TB	133.81065
2^{160}	165.12699	1.23369	$2.38 \cdot 10^{26}$ TB	153.98106
2^{180}	185.44760	1.20728	$8.00 \cdot 10^{30}$ TB	174.13358
2^{200}	205.73615	1.18616	$3.58 \cdot 10^{35}$ TB	194.27161
2^{220}	225.99849	1.16887	$1.20 \cdot 10^{40}$ TB	214.39765
2^{240}	246.23898	1.15448	$4.01 \cdot 10^{44}$ TB	234.51361

which measures the amount of data that needs to be shared. These numbers for practical field sizes are shown in Tables 6, 7, 8. Due to Remark 2 we only show results for field sizes such that $K \ll q^{1/8}$.
Unfortunately parallelization makes the complexity of the linear algebra part slightly worse and for large K the difference between the matrix building step and the linear algebra step is very large.
In the above our main concern was with reducing the per computer memory cost. For each one of these computers we can parallelize the local work to any number of cores where each core computes its own share of intersection divisors which are then collected together and used to build the graph. This results in even better running times for the matrix building step but again does not help with linear algebra.

References

[ADH] 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** (1999), no. 1-2, pp. 7–18.

[BCLS] D.J. Bernstein, C. Chuengsatiansup, T. Lange, P. Schwabe, *Kummer strikes back: new DH speed records*, `http://cr.yp.to/papers.html#kummer`, Preprint 2014, 21pp.

[BCHL] J. W. Bos, C. Costello, H. Hisil, K. Lauter, *Fast Cryptography in Genus 2*, Advances in Cryptology - EUROCRYPT 2013, Lecture Notes in Computer Science, May 2013.

[DT] Diem, C., Thomé, E., *Index Calculus in Class Groups of Non-Hyperelliptic Curves of Genus Three*, Journal of Cryptology **21** (2008), no. 4, 591–611.

[Die] Diem, C., *An Index Calculus Algorithm for Plane Curves of Small Degree*, Algorithmic number theory, 543–557, Springer Berlin Heidelberg, 2006.

[Die2] Diem, C., *Index Calculus in Class Groups of Non-Hyperelliptic Curves of Genus* 3 *from a Full Cost Perspective*, `http://www.math.uni-leipzig.de/~diem/preprints/sharcs.pdf`.

[Eng] Enge, A., *Computing discrete logarithms in high-genus hyperelliptic jacobians in provably subexponential time*, Math. Comp. **71** (2002), no. 238, pp. 729–742.

[EG] Enge, A., Gaudry, P., *A general framework for subexponential discrete logarithm algorithms*, Acta Arith. **102** (2002), no. 1, pp. 83–103.

[Gau] Gaudry, P., *An algorithm for solving the discrete log problem on hyperelliptic curves*, Advances in Cryptology - EUROCRYPT 2000, Springer-Verlag, LNCS **1807**, pp. 19–34, 2000.

[GTTD] Gaudry, P., Thomé , E., Thériault, N., Diem, C., *A Double Large Prime Variation for Small Genus Hyperelliptic Index Calculus*, Math. Comp. **76** (2007), no. 257, 475–492.

[Hes] Hess, F., *Computing Riemann-Roch spaces in algebraic function fields and related topics*, J. Symbolic Comput. **33** (2002), no. 4, 425–445.

[Kob] Koblitz, N., *Hyperelliptic Cryptosystems*, Journal of Cryptology **1** (1989), 139–150.

[Nag] Nagao, K., *Index calculus for Jacobian of hyperelliptic curve of small genus using two large primes*, Japan Journal of Industrial and Applied Mathematics, **24**, no.3 (2007).

[Smi] Smith, B., *Isogenies and the Discrete Logarithm Problem in Jacobians of Genus 3 Hyperelliptic Curves*, Advances in Cryptology - EUROCRYPT 2008, 163-180, Springer Berlin Heidelberg, 2008.

[The] Thériault, N., *Index calculus attack for hyperelliptic curves of small genus*, Advances in Cryptology - ASIACRYPT 2003, New York: Springer, 2003.

[VJS] M. D. Velichka, M. J. Jacobson Jr., A. Stein, *Computing discrete logarithms in the Jacobian of high-genus hyperelliptic curves over even characteristic finite fields* Math. Comp. **83** (2014), 935–963.