COMPUTING DISCRETE LOGARITHMS IN THE JACOBIAN OF HIGH-GENUS HYPERELLIPTIC CURVES OVER EVEN CHARACTERISTIC FINITE FIELDS

M. D. VELICHKA, M. J. JACOBSON, JR., AND A. STEIN

Abstract. We describe improved versions of index-calculus algorithms for solving discrete logarithm problems in Jacobians of high-genus hyperelliptic curves defined over even characteristic fields. Our first improvement is to incorporate several ideas for the low-genus case by Gaudry and Theriault, including the large prime variant and using a smaller factor base, into the large-genus algorithm of Enge and Gaudry. We extend the analysis in [24] to our new algorithm, allowing us to predict accurately the number of random walk steps required to find all relations, and to select optimal degree bounds for the factor base. Our second improvement is the adaptation of sieving techniques from Flassenberg and Paulus, and Jacobson to our setting. The new algorithms are applied to concrete problem instances arising from the Weil descent attack methodology for solving the elliptic curve discrete logarithm problem, demonstrating significant improvements in practice.

1. Introduction

Elliptic curves [26, 33] and hyperelliptic curves [27] were proposed for use in public-key cryptographic protocols based on the discrete logarithm problem in the Jacobian of these curves. Elliptic curves have become a popular choice in many protocols. Recent attacks (see e.g. [16, 38, 15]) show that hyperelliptic curves of genus 2 and possibly genus 3 have comparable properties.

Hyperelliptic curves are not only of interest on the constructive side. High genus hyperelliptic curves are of cryptographic interest in the analysis of elliptic curve cryptosystems in the context of the Weil descent attack methodology [12, 14]. Weil descent allows one to reduce the elliptic curve discrete logarithm problem (ECDLP) on some elliptic curves defined over a finite field of composite degree to an instance of the hyperelliptic curve discrete logarithm problem (HCDLP) of high genus defined over a smaller field. Under certain circumstances, the resulting instance of the HCDLP can be solved in subexponential time using index calculus algorithms. In our case, we only consider fields of characteristic 2.

In [24], the authors looked at the cost of using Weil descent to solve some specific instances of the elliptic curve discrete logarithm problem in practice. The main tool was an improved version of the Enge-Gaudry algorithm [9] for solving instances of the hyperelliptic curve discrete logarithm problem over high-genus curves, that includes an enhanced smoothness-testing algorithm and a strategy for selecting

2010 *Mathematics Subject Classification.* Primary 14G50; Secondary 11G20,11Y16,11Y40.
Key words and phrases. hyperelliptic curves, discrete logarithm problem, sieving, Weil descent.
The first author's research was supported by NSERC of Canada.
The second author's research is supported by NSERC of Canada.

an optimal factor base empirically. Four instances of the elliptic curve discrete logarithm problem were considered for which the curves were defined over $\mathbb{F}_{2^{62}}$, $\mathbb{F}_{2^{93}}$, $\mathbb{F}_{2^{124}}$, and $\mathbb{F}_{2^{155}}$. Weil descent reduces these to instances of the hyperelliptic curve discrete logarithm problem where the curves have genus 31 and are defined over $\mathbb{F}_{2^2}$, $\mathbb{F}_{2^3}$, $\mathbb{F}_{2^4}$, and $\mathbb{F}_{2^5}$. The first three discrete logarithm problems were solved using a parallel implementation of the improved Enge-Gaudry algorithm, and estimates for the fourth example were given indicating that it should be solveable, as the time required is comparable to that required to break the data encryption standard block cipher.

In this paper[1], we describe further improvements to the Enge-Gaudry algorithm in the large genus case that allowed us to solve the remaining discrete logarithm problem from [24] (elliptic curve over $\mathbb{F}_{2^{155}}$) in less time than predicted. The algorithm is extended, in terms of both the implementation and the empirical parameter selection strategy, to incorporate the large prime variants described by Thériault [38] for low-genus hyperelliptic curves. In addition, the sieving strategy for relation generation of Flassenberg and Paulus [10] is optimized for characteristic 2 and extended to work with the well-known self-initialization strategy employed in integer factorization and index-calculus in quadratic number fields. The resulting algorithms are applied to the same Weil descent examples as in [24], showing that each of the modifications does indeed lead to improved performance in practice.

This paper is organized as follows. We describe the relevant background material on hyperelliptic curves and the hyperelliptic curve discrete logarithm problem in Section 2. Our adaptation of Thériault's large prime variation [38] to the version of the Enge-Gaudry algorithm from [24], as well as the revised empirical estimates for parameter selection, are described in Section 3. The sieve-based improvements are described in Section 4. Our numerical results are presented in Section 5, followed by a summary of possible future research directions.

2. Hyperelliptic Curves

For details on hyperelliptic curves arithmetic, we refer to [5, 31, 7]. Here, we briefly sketch the basics.

Let $k = \mathbb{F}_q$ denote the finite field with q elements and $\overline{k} = \bigcup_{n \geq 1} \mathbb{F}_{q^n}$ its algebraic closure. A *hyperelliptic curve C of genus g over k* is defined by a non-singular equation

$$v^2 + h(u)v = f(u) \ ,$$

where $h, f \in k[u]$, $\deg f = 2g + 1$, and $\deg h \leq g$. Let L be an extension field of k. The set of *points* on C is $C(\overline{k}) = \{(x, y) : x, y \in \overline{k},\ y^2 + h(x)y = f(x)\} \cup \{\infty\}$. The *opposite* of $P = (x, y) \in C(\overline{k}) \setminus \{\infty\}$ is $\widetilde{P} = (x, -y - h(x))$ and $\widetilde{\infty} = \infty$.

2.1. **Jacobian of a Hyperelliptic Curve.** A *degree zero divisor* D of C is a formal sum $\sum_{P \in C(\overline{k})} m_P P$, where $m_P \in \mathbb{Z}$, only a finite number of the m_P's are non-zero, and $\sum m_P = 0$. The set D^0 of zero divisors is an additive group under formal addition $\sum m_P P + \sum n_P P = \sum (m_P + n_P) P$. The *Frobenius map* $\sigma : \overline{k} \to \overline{k}$, $x \mapsto x^q$, extends naturally to $C(\overline{k})$ by $(x, y) \mapsto (x^\sigma, y^\sigma)$ and $\infty^\sigma \mapsto \infty$, and homomorphically to D^0 by $\sum m_P P \mapsto \sum m_P P^\sigma$. Let $D_k^0 = \{D \in D^0 : D^\sigma = D\}$ be the set of zero divisors defined over k and let $k(C)$ be the *function field* of C over k, i.e., the field of

[1]The results presented in this paper are described in more detail in M. Velichka's M.Sc. thesis [39].

fractions of the integral domain of polynomial functions $k[u,v]/(v^2+h(u)v-f(u))$. For $f \in k(C)$, the *divisor of* f is defined as $\mathrm{div}(f) = \sum_{P \in C(\overline{k})} v_P(f)P$, where $v_P(f)$ denotes the multiplicity of P as a root of f. If we let $\mathrm{Prin}_k = \{\mathrm{div}(f) : f \in k(C)\}$, then Prin_k is a subgroup of D_k^0. Finally, the *Jacobian* of C defined over k is the quotient group $J_C(k) = D_k^0/\mathrm{Prin}_k$.

The Jacobian $J_C(k)$ is a finite abelian group of size $\#J_C(k) \approx q^g$. We write $D_1 \sim D_2$ to denote that D_1 and D_2 lie in the same equivalence class of divisors in $J_C(k)$. In our situation, we know that each equivalence class has a unique *reduced divisor*, i.e., a divisor $\sum_{P \neq \infty} m_P P - (\sum_{P \neq \infty} m_P)\infty$ satisfying (i) $m_P \geq 0$ for all P; (ii) if $m_P \geq 1$ and $P \neq \widetilde{P}$, then $m_{\widetilde{P}} = 0$; (iii) $m_P = 0$ or 1 if $P = \widetilde{P}$; and (iv) $\sum m_P \leq g$. Each reduced divisor can be uniquely represented by a pair of polynomials $a, b \in k[u]$ where (i) $\deg b < \deg a \leq g$; (ii) a is monic; and (iii) $a \mid (b^2+bh-f)$. We write $D = \mathrm{div}(a,b)$ to mean $D = \gcd(\mathrm{div}(a), \mathrm{div}(b-v))$ where the gcd of two divisors $\sum m_P P$ and $\sum n_P P$ is defined to be $\sum \min(m_P, n_P)P$. The identity of $J_C(k)$ is represented by the the divisor $\mathrm{div}(1,0)$.

In order to measure the size of a divisor, we now define the *degree* of $D = \mathrm{div}(a,b)$ as $\deg D = \deg a$. Notice that by the degree of a divisor, we do not mean the degree of $D \in D^0$ as a formal sum (which is of course 0). Similarly, we define the *norm* of D to be $N(D) = a$, and note that $N(D_1 + D_2) = N(D_1)N(D_2)$, where $D_1 + D_2$ denotes the formal sum.

The group law in $J_C(k)$ can be efficiently computed via NUCOMP as described in [25]. For two reduced divisors D_1 and D_2, a reduced divisor $D_3 \sim D_1 + D_2$ can be determined by this algorithm in polynomial time.

2.2. **The Hyperelliptic Curve Discrete Logarithm Problem.** Let C be a genus g hyperelliptic curve over $k = \mathbb{F}_q$. The hyperelliptic curve discrete logarithm problem (HCDLP) is the following: given C, $D_1 \in J_C(k)$, $r = \mathrm{ord}(D_1)$, and $D_2 \in \langle D_1 \rangle$, find the integer $s \in [0, r-1]$ such that $D_2 = sD_1$. We shall assume that r is prime, and $\#J_C(k) \approx r$.

Enge and Gaudry [9] developed an index-calculus algorithm suitable for solving the HCDLP on high-genus hyperelliptic curves. Under the assumptions that $g/\log q \to \infty$ and that $\#J_C(k)$ is known, their algorithm requires $L_{q^g}[\sqrt{2}] = L_{q^{2g+1}}[1]$ bit operations, where $L_N[\alpha] = \exp((\alpha + o(1))\sqrt{\log N \log\log N})$. Our improved algorithms are based on the Enge-Gaudry method; we begin with an overview of this algorithm.

The key to the efficiency of any index-calculus algorithm, including the Enge-Gaudry algorithm, is being able to efficiently and with high probability find and recognize elements that are "smooth" in some sense. In the hyperelliptic curve case, we need to find divisors that are the sum of small-degree prime divisors. A reduced divisor $D = \mathrm{div}(a,b) \in J_C(k)$ is called a *prime divisor* if a is irreducible over k. Throughout the paper, primes and prime divisors are simply identified with reduced divisors $\mathrm{div}(p,b) \in J_C(k)$, denoted as $P = \mathrm{div}(p,b)$, where p is a prime polynomial. Its degree is $\deg P = \deg p$. The size of P is measured by the degree of p as a polynomial (even though its degree as formal divisors is 0). It makes therefore sense to define smooth divisors as follows: A reduced divisor is said to be *t-smooth* if it can be expressed as a sum of prime divisors that all have degee $\leq t$.

The set of all prime divisors of degree $\leq t$ can be found as follows. For each monic irreducible polynomial $p \in k[u]$ of degree $\leq t$, find the roots of $v^2 + h(u)v - f(u)$

modulo $p(u)$. For each root $b(u)$ (there are either 0, 1 or 2 such roots), $\mathrm{div}(p, b)$ is a prime divisor. In order to ensure we always use the same prime divisor for a given polynomial p in the cases where 2 roots exist, we take the root $b(u)$ for which the integer value of $b(u)$ evaluated at q is smaller.

A reduced divisor $D = \mathrm{div}(a, b) \in J_C(k)$ can be efficiently decomposed into prime divisors in expected time polynomial in the degree of a and b. Simply factor a into monic irreducibles over k and obtain $a = \prod_{i=1}^{L} a_i^{e_i}$. Then $D = \sum_{i=1}^{L} e_i \mathrm{div}(a_i, b_i)$, where $b_i = b \bmod a_i$ for $1 \leq i \leq L$.

2.3. **The Enge-Gaudry index-calculus algorithm.** As in any index-calculus algorithm, the Enge-Gaudry index-calculus algorithm requires to choose a factor base S of suitable prime elements of small norm. Select a degree bound t and define the factor base $S = \{P_1, P_2, \ldots, P_w\}$ as the set of prime divisors of degree $\leq t$. The crucial input is a pseudo-random walk (see e.g. [37]) in the set of reduced divisors. This walk produces random linear combinations of the form $\alpha D_1 + \beta D_2$. If the reduced divisor R equivalent to this form is t-smooth, it will be stored. This leads to relations of the form $\alpha_i D_1 + \beta_i D_2 \sim R_i = \sum_j e_{ij} P_j$. Once we have encountered $w + 1$ distinct relations, we apply linear algebra modulo r to obtain a non-trivial linear combination $\sum_{i=1}^{w+1} \gamma_i (e_{i1}, e_{i2}, \ldots, e_{iw}) = (0, 0, \ldots, 0)$. This yields the discrete logarithm $\log_{D_1} D_2 = -(\sum \gamma_i \alpha_i)/(\sum \gamma_i \beta_i) \bmod r$, if $\sum \gamma_i \beta_i$ is invertible modulo r.

The pseudo-random walk is constructed by first computing 20 random reduced divisors $T_i \sim a_i D_1 + b_i D_2$ for $0 \leq i \leq 19$, where a_i, b_i are randomly selected integers from $[0, r-1]$. The starting point of the walk is a reduced divisor $R_0 \sim \alpha_0 D_1 + \beta_0 D_2$, where α_0 and β_0 are randomly selected from $[0, r-1]$. In each step of the pseudo-random walk, the reduced divisor R_i is derived from the previous reduced divisor $R_{i-1} = \mathrm{div}(a, b)$ as $R_i \sim R_{i-1} + T_j$, where j is obtained by taking the integer formed from the 5 least significant bits of the binary representation of a, and reducing it modulo 20. We then obtain a representation $R_i \sim \alpha_i D_1 + \beta_i D_2$ where $\alpha_i = (\alpha_{i-1} + a_j) \bmod r$ and $\beta_i = (\beta_{i-1} + b_j) \bmod r$. Thus the pair (α_i, β_i) can be easily computed from the pair $(\alpha_{i-1}, \beta_{i-1})$. For the high-genus examples we are interested in, NUCOMP as described in [25] is the most efficient choice for divisor arithmetic.

In order to find relations, each divisor produced by the random walk must be tested for smoothness. Suppose that the factor base contains all prime divisors with degree $\leq t$. Given a reduced divisor $D = \mathrm{div}(a, b)$, $a(u)$ is first subjected to a square-free factorization algorithm (e.g., see [3]). The square-free portion $\overline{a}(u)$ is then tested for t-smoothness using the fact that $x^{q^l} - x$ is the product of all monic irreducible polynomials in $\mathbb{F}_q[x]$ of degree dividing l. If $\overline{a}(u)$ is indeed t-smooth, then the factorization can be obtained using, for example, the Cantor-Zassenhaus factoring algorithm [6].

Note that it is straightforward to parallelize this algorithm. Each processor independently executes a different random walk and sends the relations to a master process responsible for coordinating the algorithm.

3. Improvements to the Enge-Gaudry Algorithm

Thériault [38] describes two improvements to Gaudry's index-calculus algorithm [13] for solving the HCDLP on low-genus hyperelliptic curves. In this case, the factor base consists only of degree 1 prime divisors. Thériault shows how the use of

large primes, as previously employed in integer factorization and discrete logarithm computation in finite fields and class groups of number fields, and a novel idea of using only a portion of the degree one prime divisors in the factor base, can be used to obtain algorithms with improved asymptotic run times over [13].

Let r be a parameter such that $2/3 < r < 1$. Then prime divisors are added to the factor base until $|S| = q^r$. In the remainder of this section we will use the term smooth to mean *smooth over* S. We will use $\mathcal{P}$ to denote all prime divisors that have norms with degree equal to one. A *potentially smooth* divisor is one that is smooth over $\mathcal{P}$ (but not necessarily over S). A divisor is *almost smooth* if all of its factors are in S with the exception of one, which is in $\mathcal{P} \setminus S$. That is, a divisor D is almost smooth if $D \sim P + \sum_{i=1}^{|S|} e_i P_i$, where the divisors P_i are again the prime divisors in D, and $P \in \mathcal{P} \setminus S$. Such a prime divisor P is called a *large prime*.

If two almost smooth divisors share the same large prime divisor, or one has the large prime P and the other has its inverse $-P$, then we have an *intersection*, and these two divisors can be combined to form a relation. Let $E_1 \sim P + \sum_{i=1}^{|S|} e_{1,i} P_i \sim a_1 D_1 + b_1 D_2$ and $E_2 \sim P + \sum_{i=1}^{|S|} e_{2,i} P_i \sim a_2 D_1 + b_2 D_2$. Then

$$E_1 - E_2 \sim \sum_{i=1}^{|S|} (e_{1,i} - e_{2,i}) P_i \sim (a_1 - a_2) D_1 + (b_1 - b_2) D_2 \ ,$$

and we have a relation. If E_2 has large prime $-P$, we consider $E_1 + E_2$.

This gives rise to two variations of the random walk strategy as used by Gaudry in [13]. The first is almost exactly the same as the original. Each divisor is checked to see if it is potentially smooth. If so, it is completely factored and if the factors are all in S a relation has been found. Otherwise the divisor is discarded and the next one is tested. The effect of this strategy is that relations will be harder to find as compared with Gaudry's algorithm, but the factor base is smaller, allowing the linear system to be solved faster. By balancing the time for finding relations with the linear algebra, Thériault obtained an improved asypmptotic run time.

The second variation that Thériault analyzes takes advantage of the almost smooth divisors. Again, a divisor is first checked for smoothness over $\mathcal{P}$. If it passes this first test then it is completely factored. If the divisor turns out to be smooth over S, a relation is recorded. If the divisor is almost smooth then the corresponding prime divisor is tested for intersections with any previously found almost smooth divisors. If so, a relation is formed and recorded, with both almost smooth divisors being removed. Otherwise, the almost smooth divisor is saved and the next divisor is checked.

We now describe our method of incorporating these ideas into the Enge-Gaudry algorithm as described in the previous section. Similar to that done by Thériault in [38], we introduce a parameter r such that $0 < r \leq 1$. Then we create a factor base that contains all prime divisors with degree up to and including $t-1$. In addition, if there are A_t prime divisors with degree equal to t, we add rA_t of them to the factor base. Then the size of the factor base is $\sum_{i=1}^{t-1} A_i + rA_t$. Note that the previous definitions of smooth and potentially smooth carry over. We let $\mathcal{P}$ denote not the prime divisors with degree equal to 1, but rather all of the prime divisors with degree up to and including t.

The random walk operates in a similar manner as before. A divisor is tested for potential smoothness over $\mathcal{P}$ using the test described in the previous section. If

the divisor $\mathrm{div}(a, b)$ passes this first test, then a is explicitly factored to see if it is actually smooth over the norms of the prime divisors in the factor base.

Recall that this is essentially how Thériault's algorithm works. The change in what is included in the factor base obviously results in a change in how the large prime variations works. We consider two different ideas for incorporating large primes into the large genus case.

First we describe the obvious generalization of Thériault's work. Suppose we let r be such that $0 < r < 1$ as above. Then there will be some prime divisors with degree equal to t that are not in the factor base. Similar to [38], these are our large primes. These large prime divisors work in the exact same manner as described above. A divisor $\mathrm{div}(a, b)$ is tested for smoothness over the factor base using the test from the previous chapter. If it passes, then a is factored, and if the divisor turns out to be almost smooth we check for intersections. If this results in a relation being formed, then it is added to the collection of relations that have been found. Otherwise the almost smooth divisor is stored. If the tested divisor is neither smooth nor almost smooth, it is discarded.

The second idea is to fix the degree bound for the divisors in the factor base to be t and include in the set $\mathcal{P}$ the prime divisors with norms having degree $t+1$. We allow our r parameter to be in the range $(0, 1]$ and thus the large prime divisors include all of the prime divisors with degree equal to $t + 1$ and possibly prime divisors with degree t, if r is not equal to 1. We proceed in a similar manner. Test a divisor for smoothness over $\mathcal{P}$. If it passes, factor the divisor. If it is almost smooth, check for intersections and form relations or store the almost smooth divisor.

3.1. **Empirical Analysis.** In [24], empirical estimates are provided for the expected number of random walk steps required to find all required relations. We now extend this analysis to incorporate our adaptations of Thériault's improvements. Given empirical data about the amount of time required to generate and test a divisor, these computations can give estimated runtimes for various combinations of t, r and different strategies for applying the large prime variant. Furthermore, if estimated running times for the linear algebra with different sized matrices are available, we can compute estimated times for solving the discrete logarithm problem. We apply these methods to concrete examples in Section 5.

We first recap the analysis developed by Jacobson, Menezes, and Stein [24] for the Enge-Gaudry algorithm without r and large primes. Assume that we are working with a hyperelliptic curve of genus g defined over $\mathbb{F}_q$. Let A_l be the number of irreducible polynomials p of degree l for which there exists a prime divisor $\mathrm{div}(p, b)$, where $1 \leq l \leq t$. From Chapter 14 in [41] we know that there are

$$I(l, q) = \frac{1}{l} \sum_{d|l} \mu\left(\frac{l}{d}\right) q^d$$

monic irreducible polynomials of degree l in $\mathbb{F}_q[X]$, where $\mu(x)$ is the Möbius function. Recall that $\mu(x)$ evaluates to 1 if $n = 1$, $(-1)^k$ if n is the product of k distinct primes and 0 if n is not square free. We expect about half of the irreducible polynomials to split in the field [41] which gives us

$$A_l \approx \frac{1}{2}\left(\frac{1}{l} \sum_{d|l} \mu\left(\frac{l}{d}\right) q^d\right) \quad .$$

Recall that from each irreducible polynomial that splits we obtain two prime divisors that are inverses of each other; one of these becomes an element in the factor base.

As in [24], and generalized in [30] by Maurer, Menezes and Teske, let $M(g,t)$ denote the number of t-smooth reduced divisors with norms having degree at most g. Then we get

$$M(g,t) = \sum_{i=1}^{g} \left([x^i] \prod_{l=1}^{t} \left(\frac{1+x^l}{1-x^l} \right)^{A_l} \right)$$

where we use $[x^i]$ to denote the coefficient of x^i. When A_l is known, $M(g,t)$ can be computed by finding the first $g+1$ terms of the Taylor expansion of $\prod_{l=1}^{t} \left(\frac{1+x^l}{1-x^l}\right)^{A_l}$ about $x=0$ and summing the coefficients of x, x^2, ..., x^g.

We assume that reduced t-smooth divisors are distributed evenly in the Jacobian. With this assumption we can compute the expected number of random walk iterations needed to find a reduced t-smooth divisor. We denote this value by $E(t) = \#J_C(k)/M(g,t)$. Since each splitting polynomial gives rise to a prime divisor in the factor base, let $F(t) = \sum_{l=1}^{t} A_l$ be the size of the factor base. In [24] the authors find $F(t)+5$ relations before performing the linear algebra step, a decision made based on empirical data. We use the same value here. Then we expect to create and test $T(t) = (F(t)+5)E(t)$ divisors to find a sufficient number of relations.

Adding the parameter r. If we introduce the parameter r only and do not use large primes, as in the first variant analyzed by Thériault in [38], the only change is the number of prime divisors with degree t in the factor base. Thus, we repeat the above calculations using rA_t in place of A_t and obtain $M(g,t)$, $E(t)$, and $F(t) = \sum_{l=1}^{t-1} A_l + rA_t$ as before.

Adding Large Primes and setting $r=1$. We now explain how we can estimate the number of random walk steps needed when using the large prime versions of the algorithm. We start with the most basic situation: setting $r=1$ and using the prime divisors of degree $t+1$ as large primes. During the search for relations we encounter both smooth relations and almost smooth relations. We have to consider how many of each are encountered in the search.

The number of t-smooth divisors is still $M(g,t)$ as defined above. We also need to count the number of almost smooth divisors. An almost smooth divisor is one that is smooth over the prime divisors of degree less than or equal to t, with the exception of a single factor which has degree equal to $t+1$. Then the number of almost smooth divisors is the number of t-smooth divisors with degree less than or equal to $(g-(t+1))$, given by $M(g-(t+1),t)$, multiplied by the number of prime divisors with degree equal to $t+1$. A divisor with degree equal to $(g-(t+1))$ added to a divisor with degree equal to $t+1$ has degree less than or equal to g, so the divisors we are counting are reduced, as are the divisors produced by the random walk that are tested for smoothness.

Recall that A_l represents the number of irreducible polynomials of degree l and each irreducible polynomial gives rise to two distinct prime divisors, a divisor and its negation. Then there are $2A_{t+1}$ large prime divisors and a total of $A(g,t) = 2A_{t+1}M(g-(t+1),t)$ almost smooth divisors. We assume that these are also randomly distributed amongst the set of effective degree zero divisors. Then, the number of steps required to find an almost smooth divisor is

$E_{LP}(t) = \#J_C(k)/A(g,t)$ and the number of steps required to find a smooth divisor is $E(t) = \#J_C(k)/M(g,t)$ as described above.

If in the course of our search we find x almost smooth divisors then we expect that in $xE_{LP}(t)$ random walk steps we would also find $x\frac{E_{LP}(t)}{E(t)} = x\frac{M(g,t)}{A(g,t)}$ smooth divisors. We need to compute the number of relations we expect to obtain by combining the x almost smooth divisors. In Section 5.6 of [38], Thériault provides us with the required information. Let $E_{n,s}$ be the expected number of intersections when s samples are drawn with replacement from a set of n elements. Then from Theorem 1 in [38] we know that when $3 \leq s < n/2$, $\frac{2s^2}{3n} \leq E_{n,s} \leq \frac{s^2}{n}$. Here we have $n = 2A_{t+1}$, the number of possible large prime divisors, and $s = x$ is the number of almost smooth divisors found. The number of intersections, $E_{n,s} = E_{2A_{t+1},x}$, is thus the number of relations we expect to find from the almost smooth divisors.

Again, we would like the search to yield a total of $F(t)+5$ relations. Still using x to represent the number of almost smooth divisors found, we need x such that

$$x\frac{M(g,t)}{A(g,t)} + \frac{2}{3}\frac{x^2}{2A_{t+1}} = F(t) + 5 \ .$$

Solving for x and taking the positive root gives

$$x = \frac{\sqrt{9M(g,t)^2(2A_{t+1})^2 + 48F(t)A(g,t)^2A_{t+1} + 120A(g,t)^2A_{t+1}} - 3M(g,t)A_{t+1}}{4A(g,t)} \ .$$

This gives us an expected value of $T(t) = xE_{LP}(t)$ random walk steps needed to generate the relations for the linear system.

Using Large Primes with $r \neq 1$ and degree t polynomials. We now consider the computations in the case where we allow r to vary and use only the prime divisors having norms with degrees equal to t not in the factor base as the large primes. Let $L(t,r) = A_t - rA_t$ denote the number of irreducible polynomials that generate large prime divisors resulting from fixing r. Then as before we have $2L(t,r)$ large prime divisors and $A(g,t) = 2L(t,r)M(g-t,t)$ almost smooth divisors, where rA_t is used in place of A_t in computing $M(g-t,t)$, $M(g,t)$ and $F(t)$. $E_{LP}(t) = N/A(g,t)$ as before. We then find x using the same method as above and represent the number of divisors we expect to be tested by $T(t) = xE_{LP}(t)$.

Using Large Primes with $r \neq 1$ and degree $t+1$ polynomials. Finally, we consider the computations for the case where we allow r to vary and use both the remaining prime divisors with degree equal to t along with the prime divisors with degree equal to $t+1$ as large primes. Then $L(t,r) = A_t - rA_t$ is as above and $A(g,t) = 2L(t,r)M(g-t,t) + 2A_{t+1}M(g-(t+1),t)$ is the number of large prime divisors, where again $M(g-t,t)$, $M(g-(t+1),t)$ and $F(t)$ are all computed using rA_t in place of A_t. Once again, compute $E_{LP}(t) = N/A(g,t)$, x and $T(t) = xE_{LP}(t)$ using the previous method.

4. A Sieve-based Algorithm

In [40], Vollmer introduced a subexponential algorithm for computing discrete logarithms in the class group of an imaginary quadratic number field. This algorithm was of interest because it did not require the class number a priori. In our setting we assume that the class number $\#J_C(k)$ is available, but the interesting aspect of Vollmer's algorithm is that is is amenable to an especially efficient sieve-based relation generation strategy in the number field case. In this section

we describe a version of Vollmer's algorithm for the HCDLP. We also describe an improved version of the sieving methods of Flassenberg and Paulus [10] that is suitable to our setting.

Vollmer's algorithm [40], given divisors D_1 and D_2 such that $D_2 = sD_1$, computes s as follows:

(1) Generate a factor base $S = \{P_1, \ldots, P_k\}$ with $\deg P_i \leq t$ as described above.
(2) Randomly generate $k + 5$ relations $\vec{e}_j = (e_{1,j}, \ldots, e_{k,j})$ such that

$$\sum_{i=1}^{k} e_{i,j} P_i \sim \operatorname{div}(1, 0) \ .$$

Set $A = (e_{i,j})$.
(3) Find vectors $\vec{v}_{-D_1} = (v_{-D_1,1}, \ldots, v_{-D_1,k})$ and $\vec{v}_{D_2} = (v_{D_2,1}, \ldots, v_{D_2,k})$, such that $-D_1 \sim \sum_{i=1}^{k} v_{-D_1,i} P_i$ and $D_2 \sim \sum_{i=1}^{k} v_{D_2,i} P_i$. Note that we have $-D_1 + \sum_{i=1}^{k} -v_(-D_1, i) P_i \sim \operatorname{div}(1, 0)$ and $D_2 + \sum_{i=1}^{k} v_(D_2, i) P_i \sim \operatorname{div}(1, 0)$, so the vectors $(1, 0, -\vec{v_{-D_1}})$ and $(0, 1, \vec{v_{D_2}})$ are relations over the extended factor base $\{-D_1, D_2, P_1, \ldots, P_k\}$.
(4) Set $\bar{A} = \begin{pmatrix} 1 & 0 & \vec{0} \\ 0 & 1 & \vec{0} \\ -\vec{v_{-D_1}}^T & -\vec{D_2}^T & A \end{pmatrix} = \begin{pmatrix} \vec{a} \\ A' \end{pmatrix}$ and solve the linear system $A'\vec{v} = (1, 0, 0, 0, \ldots, 0)^T$ over the integers modulo N
(5) Set $s \equiv \vec{v} \cdot \vec{a} \mod N$.

We show that s is indeed the discrete logarithm. If $A'\vec{v} \equiv (1, 0, 0, 0, \ldots, 0)^T \pmod N$ has a solution then we can compute $\bar{A}\vec{v} \equiv (s, 1, 0, \ldots, 0)^T \pmod N$, and thus $(s, 1, 0, \ldots, 0)$ is a linear combination of the columns of $\bar{A}$. This implies that $-sD_1 + D_2 \sim \operatorname{div}(1, 0)$, since the columns of $\bar{A}$, when taken as coefficients of a linear combination on the extended factor base $\{-D_1, D_2, P_1, \ldots, P_k\}$, also result in $\operatorname{div}(1, 0)$. Finally, since $-sD_1 + D_2 \sim \operatorname{div}(1, 0)$, we know that $D_2 \sim sD_1$, giving us the correct result.

Since we are computing relations randomly, there are circumstances in which this could fail. Consider the homomorphism $\phi : \mathbb{Z}^k \to J_C(k)$ that takes vectors $\vec{v}$ in $\mathbb{Z}^k$ to the linear combination over the factor base $\sum_{i=1}^{k} v_i P_i$. The set of relations, i.e., those vectors $\vec{v}$ for which this linear combination results in $\operatorname{div}(1, 0)$, forms the kernel of ϕ, and we have $\mathbb{Z}^k / \ker \phi \cong J_C(k)$, provided that the factor base generates the entire Jacobian. If the lattice generated by the columns of A is not equal to the above kernel then it is possible that we cannot find a solution vector $\vec{x}$.

In an attempt to avoid this situation we can take some measures to make it more likely that the sublattice generated by the columns of A has full rank. For example, it is straightforward to ensure that multiples of existing relations are discarded. One could also generate relations such that each member of the factor base is used in at least one relation. This can still be done randomly, as discussed later, but slows down computations considerably.

In [40], Vollmer analyses his algorithm in the quadratic number field setting and shows that its run time is subexponential in $\log |\Delta|$, where Δ is the discriminant of the field. In the hyperelliptic curve setting, assuming we know $\#J_C(k)$, we expect an asymptotic result similar to that of Enge and Gaudry, but the work is left for future consideration.

4.1. **Sieving.** An alternative method of relation generation in hyperelliptic curves is motivated by the use of sieving in factoring, such as in [35] and [28]. Sieving is used to generate relations in order to compute class groups in quadratic number fields by Jacobson in [21] and by Jacobson to solve discrete logarithms in quadratic number fields in [23]. In [10], Flassenberg and Paulus provide an algorithm for sieving in hyperelliptic curves with characteristic not equal to 2. In the following we describe how sieving can be employed effectively in the framework of Vollmer's algorithm to solve the HCDLP by generalizing the approach in [10] to the characteristic 2 scenario and by incorporating a number of practical improvements.

The basic idea behind using sieving to generate relations, generalized to our context, is as follows. We use the well-known correspondence between divisors of a hyperelliptic curves C and integral ideals of the corresponding function field (see, for example, [7, Sec.4.4.6]). Recall that we have $v^2 + hv = f$, where f is monic with $\deg(f) = 2g + 1$, and since we are considering the characteristic two case, $h \neq 0$ has degree at most g. Suppose we have a divisor $D = \operatorname{div}(a, b)$ that is known to be smooth over the factor base. The divisor D corresponds to the ideal $\mathfrak{a} = a\mathbb{F}_q[X] + (b + v)\mathbb{F}_q[X]$. Let $\alpha = aS + (b + v)T \in \mathfrak{a}$ with $S, T \in \mathbb{F}_q[X]$, where $a \mid b^2 + bh + f$, be an arbitrary element in $\mathfrak{a}$. Then $N(\alpha)$, the product of α and its conjugate, is:

$$(aS + (b + v)T)\,(aS + (b + h + v)T) = a\left(aS^2 + hST + \frac{b^2 + bh + f}{a}T^2\right) \quad .$$

Since $a = N(\mathfrak{a})$ there exists an ideal $\mathfrak{b}$ with $N(\mathfrak{b}) = F(S, T) = aS^2 + hST + cT^2$, where $c = \frac{b^2+bh+f}{a}$ such that $\mathfrak{a}\mathfrak{b} = (\alpha)$. In terms of divisors, this means that there exists a divisor D' corrsponding to $\mathfrak{b}$ such that $D + D' = \operatorname{div}(\alpha)$ (i.e., $D + D'$ is principal) and $N(D') = F(S, T)$. If $F(S, T)$ is smooth for some $S, T \in \mathbb{F}_q[x]$, then we can factor D' over the factor base and since D is also smooth over the factor base, this yields a relation. Thus, given a smooth divisor D, finding relations reduces to finding smooth values of $F(S, T)$. Sieving allows us to efficiently find such smooth values.

We consider one dimensional sieving, as in [22]. Fix $T = 1$ and let $F(S) = aS^2 + hS + c$ be the *sieving polynomial* with coefficients in $\mathbb{F}_q[X]$. Fix a prime divisor P from the factor base and consider $N(P) = p$. Let r be a root of $F(S)$ modulo p. Then $F(r)$ is divisible by p, as is $F(z)$ for $z = r + ip$ for $i \in \mathbb{F}_q[X]$.

To illustrate the basic functionality of sieving, we describe the case where $F(S) \in \mathbb{Z}[x]$ first. We begin by selecting a sieve interval $(-M, M)$, where M is a positive integer, and initializing the *sieve array* D by setting $D[i] = 0$ for $i = -M$ to M. Next, for each prime p, compute the roots of $F(S)$ modulo p (there could be one or two roots), and for each root r, add $\log p$ to $D[r + ip]$ for i such that $-M \leq r + ip \leq M$. After all primes have been processed, traverse through the sieve array and any c such that $D[c]$ is larger than a given *tolerance value* Y is marked as a candidate. For each candidate c, compute $F(c)$ and test for smoothness over the factor base. Any smooth candidates result in relations.

By using a tolerance value to determine smooth candidates we can allow for prime powers that divide $F(c)$. A lower tolerance value results in more candidates that have to be tested, but through tuning a value can be chosen that allows for a balance between the amount of time spent testing non-smooth values and the time saved by using candidates that have powers of primes as factors.

As Flassenberg and Paulus point out in [10], sieving in the hyperelliptic curve context provides a new challenge: representing and moving through a sieve array where all values involved are polynomials. In the integer case we can represent the sieve array by integer indices of an array, as the map from $(-M, M)$ to these indices is trivial. However, in the hyperelliptic curve context it is not as clear how we can map polynomials in $\mathbb{F}_q[X]$ to these integer indices efficiently. Additionally, moving from $r + ip$ to $r + (i+1)p$ in the integer case is easy as we just move p places in the array. Again, in the hyperelliptic curve context we do not have this luxury because the distance between interesting array entries is not constant.

We now present a generalization of Flassenberg and Paulus' sieving strategy suitable to the characteristic 2 case. As before, we continue to consider one dimensional sieving, and set $F(S) = F(S, 1)$ to be the sieving polynomial. In this case we can set $b_F = aS + b$ and find the signs of the exponents in $\vec{v}$ by finding s_i such that $s_i = 1$ if $b_F \equiv b_{p_i} \pmod{p_i}$ and $s_i = -1$ if $b_F \equiv b_{p_i} + h \pmod{p_i}$ for all prime divisors P_i in the factor base with $N(P_i) = p_i$.

Since we are working over a characteristic two field we cannot use the familiar quadratic formula to find the roots of this polynomial. However, the task is made easier by recalling that we are finding roots modulo p where p is the norm of a prime divisor in the factor base. If $p \mid a$ then

$$F(S) \equiv hS + c \pmod{p}$$

and $r \equiv ch^{-1} \pmod{p}$ is clearly the single root. If $p \mid f$ and $p \nmid a$ then

$$F(S) \equiv aS^2 + hS + b^2(a^{-1}) + bh(a^{-1}) \pmod{p}$$

and the roots are given by $r \equiv ba^{-1} \pmod{p}$ and $r \equiv (b + h)a^{-1} \pmod{p}$, by inspection. Finally, if $p \nmid a$ and $p \nmid f$ then we can solve

$$r^2 + rha^{-1} + ca^{-1} \equiv 0 \pmod{p}$$

using a generalized version of the RESSOL algorithm by Shanks as in [10]. This gives us the roots X and $X + ha^{-1} \pmod{p}$. Thus we have the roots of the sieving polynomial and can proceed with the rest of the algorithm.

Sieve Array Implementation. Efficiently moving through the sieve array given a root is one of the main complications in the hyperelliptic curve case. As originally presented in [10], we assign a sieve array for which each element of the array corresponds to a polynomial in $i \in \mathbb{F}_q[X]$ with $\deg(i) \leq M$ for some degree bound M. Each such polynomial i is mapped to a unique index $\nu(i) \in [0, \ldots q^M - 1]$ as follows. We define a map $\nu_0 : \mathbb{F}_q \to \mathbb{Z}^{\geq 0}$ that takes every element $\gamma \in \mathbb{F}_q$ to a unique integer between 0 and $q - 1$. If the elements of $\mathbb{F}_q$ are represented as polynomials, this is done by evaluating the polynomial at the characteristic of the field. Then the map $\nu : \mathbb{F}_q[X] \to \mathbb{Z}^{\geq 0}$ is defined for $S \in \mathbb{F}_q[X]$ by $\nu(S) = \sum_{i=0}^{\deg S} \nu_0(S_i) q^i$.

Moving through the sieving array from one index to the next requires the computation of $\nu(r + (k+1)p)$ given $\nu(r + kp)$. Our method, which improves on that of [10], is as follows. For a prime divisor P with norm p, start by computing and storing $\nu_0(p_i)$ for all the coefficients of p. To compute $k_{j+1}p$ from $k_j p$ we have to consider what happens when we add 1 to k. We perform a place-wise addition of the coefficients of p, which gives us $\nu_0((kp)_i)$ for $i \leq \deg(kp)$, where $(kp)_i$ denotes the coefficient of x^i in the polynomial kp. Then we compute $\nu(kp) = \sum_{i=0}^{\deg(kp)} \nu_0((kp)_i) q^i$ with $\deg(kp)$ multiplications and additions, assuming that the q^i terms have been

precomputed. This simplifies the computation by avoiding the evaluation of ν_0 every time. Finally, $\nu(r + kp)$ is calculated by summing $\nu(r)$ and $\nu(kp)$.

This idea could be taken even further. One could create a $M+1$ dimensional sieve array, where M is the upper bound on the degree of the sieve radius, as described in the previous chapter, and where each dimension has size q. Then by maintaining a series of pointers representing the various coefficients of the polynomials evaluated at $r + kp$, we can step through the array and mark spots as being divisible or not. This results in the removal of most of the polynomial evaluations, and could possibly result in further improvement.

Low-degree Sieving. The factor base typically consists of a much larger number of prime divisors having norms with large degree than divisors with small degree norms. Considering the norms of these prime divisors, we expect fewer of these higher degree irreducible polynomials to divide the test polynomials. However, a significant amount of time is spent checking for divisibility by these high-degree irreducible polynomials.

This observation leads to the idea of sieving with only the norms with lower degrees. For example, sieve with the norms of prime divisors that have degree up to $t-1$ if the factor base degree bound is t. The tolerance value can then be adjusted accordingly to account of the possibility of a higher degree factor that has not been sieved. How we do this is discussed at the end of the section. Clearly the amount of time spent sieving is reduced, but there is a trade off in that potentially smooth candidates have to be tested for smoothness over the factor base. The hope is that a tolerance value can be chosen such that the extra amount of time spent testing for smoothness is less than the time that would have been spent sieving with the higher degree norms.

Self-Initialization. The sieving process relies greatly on the speed at which sieving polynomials can be created and the time taken to compute the roots of these polynomials modulo primes. For factoring [1, 8] and computing class groups in quadratic number fields [22], a process called self-initialization has been introduced which allows the amount of computation done to be reduced significantly. As there are no known implementations of self-initialization being used in conjunction with sieving in the hyperelliptic curve case, we now describe self-initialization as described in [22], generalized to work in this context in characteristic 2.

Suppose we have a divisor D with norm a that is the sum of j distinct prime divisors Q_i, with norms q_i. Then $D = \sum_{i=1}^{j} v_i Q_i$ where $v_i \in \{-1, 1\}$ and $a = \prod_{i=1}^{j} q_i^{|v_i|}$. Note that $\sum_{i=1}^{j} -v_i Q_i = -D$, so to avoid duplicate relations we fix $v_j = 1$ and only produce 2^{j-1} divisors, denoted by $D_1 = \mathrm{div}(a, b_1)$, $D_2 = \mathrm{div}(a, b_2)$, $\ldots$, $D_2^{j-1} = \mathrm{div}(a, b_{2^j-1})$. The trick is computing these divisors without actually computing the various products of prime ideals. In fact, both this and the computation of most of the roots of the corresponding sieving polynomials modulo the norms of the factor base elements can be computed in an efficient manner.

Let $F_i(S) = aS^2 + hS + c_i$ denote the sieving polynomial corresponding to D_i, where $c_i = (b_i^2 + b_i h + f)/a$ and, as noted above, $a = \prod_{i=1}^{j} q_i$. Similar to Theorem 4.1 in [22], for $1 \leq i \leq j$ we write

$$B_i = (a/q_i) \left((a/q_i)^{-1} t_i \pmod{q_i} \right) \pmod{a}$$

and

$$B_i^{'} = (a/q_i)\left((a/q_i)^{-1}(t_i + h) \pmod{q_i}\right) \pmod{a}$$

where t_i is a solution to $x^2 + xh \equiv f \pmod{q_i}$. Then $b = \pm B_1 \pm B_2 \cdots + B_j$ is a solution to $x^2 + xh \equiv f \pmod{a}$, where we use $-B_i$ to denote $B_i^{'}$. To see this, first recall that the Chinese Remainder Theorem tells us that if $a = \prod_{i=1}^{j} q_i$ then $b^2 + bh + f \equiv 0 \pmod{a} \iff b^2 + bh + f \equiv 0 \pmod{q_i}$ for all $1 \leq i \leq j$. Note that $B_i \equiv t_i \pmod{q_i}$ and $B_i \equiv 0 \pmod{q_k}$ for $1 \leq k \leq j$, $k \neq i$. Similar results hold for $B_i^{'}$. Then $b^2 + bh + f \equiv t_i^2 + t_i h + f \equiv 0 \pmod{q_i}$ or $b^2 + bh + f \equiv (t_i + h)^2 + (t_i + h)h + f \equiv 0 \pmod{q_i}$ for all $1 \leq i \leq j$. Thus b as defined above is a solution to $b^2 + bh \equiv f \pmod{a}$.

Now we can easily change sieving polynomials. Let $\vec{v} = \{v_1, v_2, \ldots, v_j\}$ with $v_i = 1$ for $i = 1, \ldots, j$. Then we can compute the sieving polynomial by finding $a = \prod_{i=1}^{j} q_i$ and $b_1 = \sum_{i=1}^{j} B_i$, giving $F_1(S) = aS^2 + hS + ({b_1}^2 + b_1 h + f)/a$. Note that we fix $v_j = 1$ in order to ensure we do not use both $\vec{v}$ and $-\vec{v}$. Now, consider a $j-1$ bit Gray code, starting with $\vec{v}$. Then iterating through the possible values for $\vec{v}$ in this manner results in each $\vec{v_l}$ differing from $\vec{v}_{l-1}$ in a single place, say k. Suppose $D_{l-1} = (a, b_{l-1})$ with $b_{l-1} = v_1 B_1 + v_2 B_2 + \cdots + v_k B_k + v_{k+1} B_{k+1} + \cdots + B_j$. Then $b_l = v_1 B_1 + v_2 B_2 + \cdots + (-v_k) B_k + v_{k+1} B_{k+1} + \cdots + B_j$ where $-B_k = B_k^{'}$, and we have $b_l = b_{l-1} + B_k + B_k^{'}$. This gives $D_l = (a, b_l)$ and the sieving polynomial $F_l(S) = aS^2 + hS + ({b_l}^2 + b_l h + f)/a$ with two additions, assuming the previously computed B_i and $B_i^{'}$ values are stored.

In addition to computing the next sieving polynomial we can easily compute the roots of the new sieving polynomial modulo the norms of the prime divisors P in the factor base. Let P be an divisor in the factor base such that $P \notin \{Q_1, \ldots, Q_j\}$. First, suppose that $N(P) = p$, $p \nmid a$ and let r be a root of $F_l(S) \pmod{p}$. Then $r + (B_k + B_k^{'})a^{-1} \pmod{p}$ is a root of $F_{l+1}(S) \pmod{p}$, where k is the place where $\vec{v_l}$ and $\vec{v_{l+1}}$ differ. This is easy to confirm with substitution, and easy to compute if $(B_k + B_k^{'})a^{-1}$ is precomputed. To find the roots for the remaining norms of the prime divisors, that is for $p \mid a$, we have to compute $r \equiv c_{l+1} h - 1 \pmod{p_i}$ directly. However, there are only j of these roots that need computing. Thus, we can compute new sieving polynomials and their roots efficiently from previously computed polynomials and roots.

4.2. **Parameter Selection.** Sieving introduces a number of parameters into the algorithms for solving discrete logarithm problems. In addition to the factor base bound t, we have to consider how large the sieve radius M should be. For this discussion, let M be the degree of the maximum polynomial in the sieve array. We also need to determine the degree of polynomials with which to sieve, the tolerance value Y that decides whether or not we should check a candidate for smoothness and j, the number of prime divisors used to create the divisor that gives rise to a sieve polynomial. Note that all these parameters are positive integers. Our approach for finding values for these parameters was a combination of analysis and empirical work. Analytical work helped to determine ranges in which we should search for paramters. Then we ran test programs in which several hundred relations were found using different combinations of parameters. The set of parameters that resulted in the fastest computation was our final choice. In this section we provide

guidance for choosing these values, assuming that self-initalized sieving is being used.

Selecting the sieveing polynomial in such a way that $\deg F(S)$ is as small as possible for all S in the sieving interval is important, as smaller-degree polynomials are more likely to be smooth than those with large degree. With this in mind, we want a, the leading coefficient of $F(S) = aS^2 + hS + c$, to have degree approximately $g - M$, the genus minus the sieve radius. Because $\deg(S) \leq M$, this ensures that $\deg F(S) \leq g + M + 1$, since

$$\deg(F(S)) \leq \max(\deg(aS^2), \deg(hS), \deg(c))$$

and we have

$$\begin{aligned} \deg(aS^2) &= \deg(a) + 2\deg(S) \leq (g - M) + 2M = g + M \ , \\ \deg(hS) &= \deg(h) + \deg(S) \leq g + M \ , \end{aligned}$$

and

$$\begin{aligned} \deg(c) &\leq \deg(b^2 + bh + f) - \deg(a) \\ &= \max(\deg(b^2), \deg(bh), \deg(f)) - \deg(a) \\ &\leq \max(2(g - M), (g - M) + g, 2g + 1) - (g - M) \\ &= \max(g - M, g, g + 1 + M) \\ &= g + M + 1 \ , \end{aligned}$$

since $\deg(b) < \deg(a) = g - M$. In fact, $\deg(b^2 + bh) < \deg(f)$, so no cancellation in the numerator of c is possible and we have $\deg(c) = g + M + 1$ for our sieve polynomials.

Since j irreducible polynomials are used to generate a, each should have degree $(g - M)/j$. In order to ensure that this is possible, this value must be smaller than the maximum degree of the factor base, t, so we must take $j \geq \frac{g-M}{t}$. As a starting point we assume that t is the optimal value determined by the expected random walk values computed using the formulas discussed in Section 3.1. Once we have chosen M and j we can use these to choose which prime divisors are used to generate the sieving polynomials. We simply add any j prime divisors from the factor base with degree equal to $(g - M)/j$ or as close to it as possible.

Using the search method described above we found empirical data that suggests setting $M = t - 2$ and taking j such that $(g - M)/j \approx t - 1$ are reasonable settings to use. This makes sense, as we wish to ensure there are enough prime divisors available to generate enough sieving polynomials to find the relations needed. The relationship between j, M and t was used to dictate the parameter search space.

The last parameter is the tolerance value used for sieving. Recall that we only sieve with prime divisors of degree $\leq t - 1$ and do not explicitly test for repeated factors, so we need to select a tolerance value that accounts for this. In order to ensure that most of the candidates produced really are smooth, we need to account for not only the degrees of the prime divisors in the factor base, but also the degree of $F(S)$ when evaluated at polynomials in the sieve radius and the bound on potential large prime divisors occuring in almost smooth divisors.

We first derive a lower bound on $\deg(F(S))$. For all S with $\deg(S) \leq M$, we have $\deg(aS^2 + hS) < \deg(c)$, so $\deg(F(S)) = \deg(c) = g + M + 1$.

Now consider some $F(S)$ that is smooth over the factor base. Then $F(S) = \prod_{i=1}^{k} p_i^{a_i}$ where $p_i = N(P_i)$ for P_i in the factor base, and $\deg F(S) = \sum_{i=1}^{k} a_i \deg p_i$. If $a_i = 1$ for all i and we sieve with all primes in the factor base, then using $g + M + 1$ as a tolerance value will cause $F(S)$ to be marked as a candidate, as $\deg F(s) \geq g + M + 1$ for all S in the sieve radius. However, if we want to be able to catch factors that are squares we have to make the tolerance value smaller, for example, $g + 1$ would allow for a square factor.

When using large primes, the tolerance value must be smaller to allow for the large prime factor. For example, if we are in the case where the large prime divisors are those with norms of degree $t + 1$ we must reduce the tolerance value by $t + 1$, because an almost smooth candidate has degree that is the sum of the degrees of the prime divisors in the factor base plus the degree of the large prime, $t + 1$ (or t, depending on the variation and value of r). Furthermore, when sieving with only divisors of degree $\leq t - 1$ we have to use an even smaller tolerance value, such as $\min(\deg(F(S)) - it = g + M + 1 - it$, for some integer i, to allow for multiple factors of degree t dividing $F(S)$. This is to account for any missing degree t factors not marked during the sieving process.

These guidelines should be used as a starting point for finding the tolerance value, which can be fine-tuned using the tests mentioned above. We used test values in the range $Y \pm i$ where i depended on how long the tests were expected to take, and the results of previous tests, starting with $Y = g - M + 1 - 2T$.

5. Numerical Results

We have implemented our improvements to the Enge-Gaudry algorithm and our sieve-based hyperelliptic curve version of Vollmer's algorithm in C++. NTL [36] is used to perform the finite field arithmetic An implementation of the Lanczos algorithm from the linbox package [29] was used to solve the resulting linear systems.

We used the Advanced Cryptography Laboratory (ACL) at the University of Calgary as our testing platform. The ACL is a Beowulf cluster consisting of 152 nodes, 139 of which have dual Intel P4 Xeon 2.4 Ghz processors with 512 kb cache. The remaining 13 nodes have dual Intel P4 Xeon 2.8 Ghz processors with 512 kb cache. All nodes have 2 GB of RAM and 40 GB hard drives. These nodes are interconnected with gigabit Ethernet. The nodes all run Red Hat Enterprise Linux 3 and have the GNU Multi-Precision C library (GMP) version 4.2.2 ([18]) installed, along with NTL version 5.4.1 ([36]) and the MPICH Message Passing Interface (MPI) version 1.2.5 ([32]). Additionally, we have installed the Automatically Tuned Linear Algebra Software (ATLAS) version 3.7.31 ([2]) and linbox version 1.1.3 ([29]) to perform linear algebra. The compiler used was GCC version 3.4.4 ([17]).

We now describe some features of our implementations. The relation generation phase is done in parallel in all cases. For the random walk algorithms, each process initializes and performs its own random walk and all smooth and almost smooth relations are reported back to a single "master" process. For the sieve-based algorithm, the master process coordinates which sieve polynomials are used by each process by sending a different set of j prime divisors to be used with self-initialization.

Rather then searching the almost smooth divisors for potential intersections each time we find a new one, we wait until the expected number of intersections plus the number of relations currently found is large enough. We use Theorem 1 in [38]

to determine when this happens. To be exact, we wait until $\frac{x^2}{6A_{t+1}}$ plus the current number of relations found is greater than $F(t)+5$, the total number of relations we wish to find. In this, x is the number of almost smooth divisors found and A_{t+1} is the number of degree $t+1$ irreducible polynomials, which give rise to $2A_{t+1}$ large prime divisors, divisors and their inverses, as discussed earlier. Combining the almost smooth relations is done by the master process at this point.

The special relations corresponding to $-D_1$ and D_2 required for Vollmer's algorithm are produced by sieving with divisors of the form $-D_1 + D'$ or $D_2 + D'$ for randomly-produced smooth divisors D' without using self-initialization. Any smooth value of the corresponding sieving polynomials lead to a factorization of $-D_1$ or D_2. This step is also done in parallel by sending different random divisors D' to each process.

Once we have enough relations we move on to the linear algebra phase. In the Enge-Gaudry algorithm we find a non-zero vector in the kernel of A, i.e., we find $\vec{x}$ such that $A\vec{x} \equiv \vec{0} \pmod{N}$, where N is the provided class number. Note that our implementation assumes N is prime. If not, one can factor N and compute solutions to $A\vec{x} \equiv \vec{0}$ modulo each factor, combining the results with the Chinese Remainder Theorem, as discussed in Section 4 of [9]. We find a solution to this linear system by finding a random vector $\vec{v}$ and using the Lanczos system solver provided by the linbox library to solve $A\vec{x} \equiv A\vec{v} \pmod{N}$. The vector $\vec{x}-\vec{v}$ is, with high probability, a non-zero vector in the kernel of A. We use the function `LinBox::solve` using the Lanczos method with the default preconditioner `FULL_DIAGONAL`, specifying that the matrix is singular, limiting the maximum number of tries to 1, and ignoring the ability to certify a system without a solution. This last decision was made so that rather than spending time confirming that the system is not solvable we generate 5 more relations and try again. Our experiments show that very few additional iterations are required, if any, before a solution is found.

We use the same linbox function and options to solve the linear system $A'\vec{x} = (1, 0, \ldots, 0)$ required for Vollmer's algorithm. In addition to computing 5 new relations in the event that we cannot solve this linear system, we also recompute the special relations corresponding to $\mathfrak{a}^{-1}$ and $\mathfrak{b}$. Our experiments suggest again that very few, if any, iterations of this process are necessary.

Empirical Estimates and Parameter Selection. We tested our algorithms on instances of the HCDLP in the Jacobian of four genus 31 hyperelliptic curves defined over $k = \mathbb{F}_q$ for $q = 4$, 8, 16 and 32. The hyperelliptic curves over these fields, taken from [24], are denoted C62, C93, C124 and C155. They all have $\#J_C(k) = 2r$ where r is prime and as described in [24], were obtained by applying the GHS attack to an instance of the ECDLP on elliptic curves defined over $\mathbb{F}_{2^{62}}$, $\mathbb{F}_{2^{93}}$, $\mathbb{F}_{2^{124}}$, and $\mathbb{F}_{2^{155}}$, respectively. The curve parameters are listed in Table 1. The curve equations are given by $v^2 + h(u)v = f(u)$, where $h, f \in \mathbb{F}_q[u]$, and the prime factorizations of $\#J_C(k)$ are also listed.

In Section 3.1 we generalized the method of [24] for estimating the number of random divisors that must be tested in order to find a total of $F(t)+5$ relations to methods that also make use of a reduced factor base and large primes. We now apply these ideas to the curves C62, C93, C124, and C155 in order to find optimal parameters for our improved versions of Enge-Gaudry. We also estimate the amount of time required to perform the entire algorithm by measuring the time required to create a smaller number of divisors, for example 1000 divisors,

TABLE 1. Hyperelliptic curves C62, C93, C124, and C155 of genus $g = 31$.

C62, $q = 4$, $\mathbb{F}_{2^2} = \mathbb{F}_2[w]/(w^2 + w + 1)$ $f(u) = u^{63} + w^2u^{62} + u^{48} + w^2$ $h(u) = u^{31} + u^{30} + wu^{28} + u^{24} + w^2u^{16} + w^2$ $\#J_{\mathrm{C62}}(\mathbb{F}_{2^2}) = 2 \cdot 2305843007560748609$
C93, $q = 8$, $\mathbb{F}_{2^3} = \mathbb{F}_2[w]/(w^3 + w + 1)$ $f(u) = w^4u^{63} + w^5u^{62} + w^5u^{60} + w^3u^{56} + w^5u^{48} + wu^{32} + w^5$ $h(u) = w^2u^{31} + w^5u^{30} + u^{28} + w^6u^{24} + w^6$ $\#J_{\mathrm{C93}}(\mathbb{F}_{2^3}) = 2 \cdot 4951760157141611728579495009$
C124, $q = 16$, $\mathbb{F}_{2^4} = \mathbb{F}_2[w]/(w^4 + w + 1)$ $f(u) = w^3u^{63} + w^7u^{60} + w^3u^{56} + w^3u^{48} + 1$ $h(u) = w^9u^{31} + w^{12}u^{30} + w^8u^{28} + w^{13}u^{24} + w^6u^{16} + w^6$ $\#J_{\mathrm{C124}}(\mathbb{F}_{2^4}) = 2 \cdot 1063382396627932698548377588868981712 1$
C155, $q = 32$, $\mathbb{F}_{2^5} = \mathbb{F}_2[w]/(w^5 + w^2 + 1)$ $f(u) = w^4u^{63} + w^6u^{62} + w^{15}u^{60} + w^{26}u^{56} + w^{25}u^{48} + w^7u^{32} + w^{13}$ $h(u) = w^2u^{31} + w^7u^{30} + w^{30}u^{28} + w^{22}u^{24} + w^3u^{16} + w^{22}$ $\#J_{\mathrm{C155}}(\mathbb{F}_{2^5}) = 2 \cdot 22835963083295358096932727763065266972881541089$

and test them for smoothness. With this information we can estimate how long the relation generation stage will take, through interpolating and extrapolating the data. We also have estimated linear algebra times based on some trial runs and extrapolation. Finally, the time required for initialization (that is, creating the factor base) has been measured and used in these estimates. With these timings we can compute estimated runtimes for the Enge-Gaudry algorithm with the curves above and different choices for the factor base bound t, the parameter r and what form of almost smooth divisors are considered.

When not using large primes, there are two parameters we can vary: the factor base bound t and the parameter r controlling the number of the largest degree polynomials we use to create prime divisors for our factor base. When using large primes, we consider the case where we use just the prime divisors having norm with degree equal to t as large prime divisors and the case where we also include the prime divisors having norms with degree equal to $t + 1$.

Tables 2 to 5 provide sample points for each variation of the random walk parameters and each curve we are considering. In these tables LP indicates the number of large prime divisors that exist with these settings, $E(t)$ is the expected number of steps required to find a smooth divisor, $E_{LP}(t)$ is the expected number of tests to find an almost smooth divisor, $T(t)$ is the total number of steps expected to find $F(t) + 5$ relations, T represents the estimated time required to test $T(t)$ divisors using 256 processors. The column LA refers to the estimated time required for the linear algebra stage. The total time is the sum of LA, T and the time required for initialization (not shown). Those lines marked with $(*)$ represent the settings recommended in [24].

Note that, when incorporating large primes and $r \neq 1$, the values of t suggested in [24] may not necessarily be the best ones (see C62 and C124). In addition, the use of large primes with appropriate parameter selection results in a significant

TABLE 2. Estimated C62 runtimes in seconds.

t	r	$F(t)$	LP	$E(t)$	$E_{LP}(t)$	$T(t)$	T	LA	total
Not considering almost smooth divisors									
5	1.00	144	-	36296	-	5408075	51.61	0.07	54.14
6	0.50	309	-	7776	-	2441636	24.13	0.34	27.24
6	1.00	474	-	2614	-	1251873	13.20	0.81	16.97
7	0.50	1059	-	921	-	980453	10.37	4.04	19.06
(∗) 7	1.00	1644	-	421	-	694997	7.57	9.73	23.48
Using only degree t norms									
5	0.90	134	10	48446	162924	5050630	48.67	0.07	51.21
6	0.60	342	132	6092	6443	1391785	13.67	0.48	17.02
6	0.80	408	66	3900	9256	1249536	12.37	0.68	16.03
6	0.90	441	33	3177	15884	1223078	12.15	0.79	15.97
7	0.10	591	1053	2058	996	802851	8.44	1.43	13.28
Using degree $t+1$ norms									
5	1.00	144	330	36296	9436	2604401	25.49	0.08	28.04
6	0.50	309	1335	7776	1656	1280375	13.36	0.39	16.52
6	1.00	474	1170	2614	1051	809562	8.78	0.92	12.65
7	0.50	1059	4665	921	304	664803	7.58	4.58	16.81
7	1.00	1644	4080	421	258	542944	6.35	11.04	23.57

TABLE 3. Estimated C93 runtimes in minutes.

t	r	$F(t)$	LP	$E(t)$	$E_{LP}(t)$	$T(t)$	T	LA	total
Not considering almost smooth divisors									
4	1.00	596	-	1830509	-	1.10×10^9	178.34	0.03	178.46
5	0.50	2234	-	145591	-	325977380	56.30	0.45	56.88
(∗) 5	1.00	3872	-	28668	-	111146195	19.26	1.35	20.77
6	0.50	14771	-	6281	-	92810648	17.12	19.64	37.42
Using only degree t norms									
4	0.90	546	50	2834369	6261994	1.04×10^9	168.44	0.03	168.56
5	0.50	2234	1638	145591	75143	171550333	29.47	0.51	30.11
5	0.80	3217	655	51489	82729	115159023	20.11	1.06	21.31
5	0.90	3544	328	38100	130084	108360217	18.93	1.28	20.37
6	0.10	6052	19618	20016	6513	75517246	13.67	3.74	17.67
Using degree $t+1$ norms									
4	1.00	596	3276	1830509	266121	484073278	83.42	0.04	83.54
5	0.50	2234	23436	145591	19132	167421504	30.55	0.51	31.19
5	1.00	3872	21798	28668	7577	71792103	13.03	1.53	14.72
6	0.50	14771	160695	6281	1394	64731785	13.70	22.29	36.65

reduction in the expected number of random walk steps required to generate $F(t)+5$ relations.

Using Tables 2 to 5 we can pick optimal parameters for the Enge-Gaudry algorithm with random walks for relation generation. It would appear that in our situation, with the relation generation being done in parallel on 256 processors, we should use the variation of large primes that considers the set of large prime divisors to include those divisors having norms with degree equal to $t+1$ in all four curves. It also appears that we should set $r = 1$ for all four curves. See Table 6 for a summary of what we used for input for random walks. The large prime bound is the degree of the norms of divisors that we use in our set of large prime divisors. We also list in Table 7 the settings from [24] for reference and comparison.

TABLE 4. Estimated C124 runtimes in hours.

t	r	$F(t)$	LP	$E(t)$	$E_{LP}(t)$	$T(t)$	T	LA	total
Not considering almost smooth divisors									
4	0.50	4808	-	18991245	-	9.14×10^{10}	267.29	0.07	267.37
4	1.00	8872	-	1498799	-	1.33×10^{10}	38.86	0.24	39.11
5	0.32	42426	-	267103	-	1.13×10^{10}	36.39	5.59	42.01
5	0.50	61300	-	127546	-	7.82×10^{9}	25.37	11.67	37.08
(*) 5	1.00	113728	-	25876	-	2.94×10^{9}	10.34	40.15	50.59
Using only degree t norms									
4	0.50	4808	4064	18991245	6040905	3.60×10^{10}	104.42	0.08	104.50
4	0.90	8059	813	2308631	5163911	1.28×10^{10}	37.39	0.23	37.62
5	0.32	42426	71302	267103	88111	5.84×10^{9}	18.82	6.34	25.18
5	0.50	61300	52428	127546	67239	4.38×10^{9}	14.28	13.24	27.56
Using degree $t+1$ norms									
4	0.50	4808	108920	18991245	1336140	3.98×10^{10}	124.06	0.08	124.15
4	1.00	8872	104856	1498799	225558	7.72×10^{9}	23.96	0.28	24.24
5	0.32	42426	1469042	267103	27402	7.18×10^{9}	24.66	6.34	31.03
5	0.50	61300	1450168	127546	17154	5.16×10^{9}	17.80	13.24	31.09
5	1.00	113728	1397740	25876	6964	2.28×10^{9}	9.37	45.57	55.04

TABLE 5. Estimated C155 runtimes in days.

t	r	$F(t)$	LP	$E(t)$	$E_{LP}(t)$	$T(t)$	T	LA	total
Not considering almost smooth divisors									
4	0.80	110365	-	3358791	-	3.71×10^{11}	60.76	1.61	62.37
(*) 4	1.00	136528	-	1378374	-	1.88×10^{11}	33.17	2.47	35.65
5	0.20	807616	-	440408	-	3.56×10^{11}	114.64	86.36	201.10
Using only degree t norms									
4	0.80	110365	26163	3358791	3510031	2.13×10^{11}	35.20	1.83	37.03
4	0.92	126063	10465	1939095	5529341	1.79×10^{11}	30.71	2.39	33.11
5	0.20	807616	2684352	440408	109868	1.91×10^{11}	64.89	98.02	163.00
Using degree $t+1$ norms									
4	0.80	110365	3381603	3358791	382768	2.40×10^{11}	41.55	1.83	43.38
4	1.00	136528	3355440	1378374	210311	1.33×10^{11}	24.76	2.80	27.56
5	0.20	807616	92160024	440408	37543	2.72×10^{11}	99.37	98.02	197.48

TABLE 6. Random Walk Parameters for the Enge-Gaudry Algorithm.

Curve	C62	C93	C124	C155
Factor Base Bound t	6	5	4	4
Parameter r	1	1	1	1
Large Prime Bound	7	6	5	5
Estimated Time	12.65 Seconds	14.72 Minutes	24.24 Hours	27.56 Days

TABLE 7. Random Walk Parameters from [24] for the Enge-Gaudry Algorithm.

Curve	C62	C93	C124	C155
Factor Base Bound t	7	5	5	4
Parameter r	1	1	1	1
Estimated Time	23.48 Seconds	20.77 Minutes	50.59 Hours	35.65 Days

We used the settings in Table 6 as a starting point for selecting the sieve parameters. The remaining parameters were generated by starting with the strategy described in Section 4.2. The final parameters selected are listed in Table 8.

TABLE 8. Parameters for Sieve-Based Algorithm.

Curve	C62	C93	C124	C155
Factor Base Bound t	6	5	4	4
Parameter r	1	1	1	1
Large Prime Bound	7	6	5	5
Divisors used j	5	7	9	9
Radius Degree M	4	3	2	2
Tolerance Value Y	19	21	25	25

Numerical Results. We have used our implementation to solve discrete logarithm problems in C62, C93 and C124 using both the Enge-Gaudry algorithm and our sieve-based adaptation of Vollmer's algorithm with self-initialized sieving and parameters as selected above. Tables 9 to 12 list the results for C62 to C155, respectively. We list estimated and actual runtimes and statistics using the strategy and optimal paramters from [24] (labelled as "JMS"), our optimized version incorporating large primes, and our sieve-based version of Vollmer's algorithm.

TABLE 9. C62 Results

	JMS EG Estimate	JMS EG Result	Opt. EG Estimate	Opt. EG Result	Sieving Result
Initialization Time	6.18s	6.28s	2.96s	2.73s	0.76s
Total Relations	1649	1649	479	479	489
Full Relations	1649	1649	310	302	321
Almost Smooth Relations	-	-	770	780	986
Intersections	-	-	169	198	180
Unique Intersections	-	-	169	198	180
Sieve Polynomials	-	-	-	-	618361
Total Divisors Tested	694997	590394	809562	640471	633201664
Total Search Time	32.31m	27m 25.31s	37.44m	27m 51.86s	21m 40.78s
Real Search Time	7.57s	6.42s	8.78s	6.53s	8.21s
Total Iterations	-	1	-	1	3
Special Divisors Checked	-	-	-	-	1081344
Special Rels Time	-	-	-	-	0.99s
Linear Algebra Time	9.73s	34.45s	0.92s	2.85s	13.03s
Total Time	32.57m	28m 6.66s	37.51m	27m 57.91s	21m 56.45s
Real Total Time	23.48s	47.77s	12.65s	12.58s	22.90s

First notice that the linear algebra times were significantly underestimated in all cases. Using more accurate estimates might result in different choices for t and r as we attempt to choose settings that result in the lowest runtime. However, in most cases it is the search for smooth relations that dominated the computation time, especially when considering the total times as opposed to the real time required with 256 processors.

TABLE 10. C93 Results

	JMS EG Estimate	JMS EG Result	Opt. EG Estimate	Opt. EG Result	Sieving Result
Initialization Time	9.67s	9.76s	9.67s	8.70s	4.55s
Total Relations	3877	3877	3877	3877	3892
Full Relations	3877	3877	2504	2480	2547
Almost Smooth Relations	-	-	9475	9546	13246
Intersections	-	-	1373	1834	5271
Unique Intersections	-	-	1373	1834	1496
Sieve Polynomials	-	-	-	-	8889395
Total Divisors Tested	111146195	101080503	71792103	70951503	36410961920
Total Search Time	3.42d	3d 4h 2m 9.10s	2.32d	2d 5h 41m 16.87s	11h 36m 43.94s
Real Search Time	19.26m	17m 49.25s	13.03m	12m 34.98s	3m 13.40s
Total Iterations	-	1	-	1	3
Special Divisors Checked	-	-	-	-	11501568
Special Rels Time	-	-	-	-	7.92s
Linear Algebra Time	1.35m	6m 5.73s	1.53m	4m 57.28s	22m 7.37s
Total Time	3.43d	3d 4h 8m 26.11s	2.32d	2d 5h 46m 28.02s	11h 58m 59.23s
Real Total Time	20.77m	24m 6.26s	14.72m	17m 46.13s	25m 20.80s

For the C62 case (Table 9) we notice that significantly fewer divisors were tested than we expected in both the version of Enge-Gaudry from [24] and our optimized version. One possible cause of this is if a large number of smooth or almost smooth relations are found early in random walks. This, spread over the 256 random walks occurring, could explain this difference. Further evidence for this idea was provided when we ran the same test using a smaller number of processors. Tests using both 10 processors and 1 processor resulted in the number of divisors tested to be much closer to the estimates. We also point out that in our optimized version of Enge-Gaudry we expected $T(t)/E_{LP}(t) = 809562/1051 \approx 770$ almost smooth relations to be found, and this is almost exactly what we did find. Comparing our optimal results to the results produced using the version from [24] we see that the search times are almost the same, but due to the larger factor base size recommended by JMS, the linear algebra time is significantly larger. Finally, we note that the time spent searching for relations was improved significantly using sieving. Unfortunately, the first relation matrix did not yield a solution to the HCDLP, and in fact the linear algebra had to be repeated three times, negating the impact of the speed-up in relation generation. Even so, the total time spent on all processors was the least when using Vollmer's algorithm, but the fact that

TABLE 11. C124 Results

	JMS EG Estimate	JMS EG Result	Opt. EG Estimate	Opt. EG Result	Sieving Result
Initialization Time	5.56m	5m 44.75s	17.64s	15.59s	10.44s
Total Relations	113733	113733	8879	8879	8870
Full Relations	113733	113733	5154	5176	5084
Almost Smooth Relations	-	-	34232	34106	34520
Intersections	-	-	3725	4953	4510
Unique Intersections	-	-	3725	4953	4510
Sieve Polynomials	-	-	-	-	1581411077
Total Divisors Tested	2942900860	2931742632	7721296965	7557879515	6477459771392
Total Search Time	110.32d	115d 5h 56m 15.77s	255.59d	243d 23h 42m 45.22s	81d 6h 59m 28.12s
Real Search Time	10.34h	10h 48m 15.99s	23.96h	22h 52m 25.95s	7h 37m 14.86s
Total Iterations	-	1	-	1	1
Special Divisors Checked	-	-	-	-	143921152
Special Rels Time	-	-	-	-	3m 34.67s
Linear Algebra Time	40.15h		0.28h	37m 43.56s	1h 44.90s
Total Time	112d		255.60d	244d 21m 2.53s	81d 8h 3m 47.69s
Real Total Time	50.59h		24.24h	23h 30m 43.26s	8h 42m 3.49s

the relation generalization was distributed over 256 processors meant that the real time was not.

For C93 (Table 10) we see that the number of divisors tested in both the JMS and optimal results are much closer to that which we expected. This time the same factor base size was recommended for the version from [24] and our optimized version. Here we see that using large primes does reduce the amount of time needed in the search. Again, when comparing the results to the sieve-based algorithm we see that while relation generation with sieving is significantly faster, taking around a quarter of the time, once again the linear algebra took three iterations. This significantly increases the amount of time this test took to run on the cluster, as the linear algebra implementation we used is not parallelized. If adjusting the algorithm to produce the extra relations results in a reduction on the number of linear algebra iterations then this would result in the sieve-based method being significantly better here. We also note that while there were a significant number of almost smooth relations and intersections in the sieve-based method, a large number of the relations formed by the intersections were discarded as being duplicate relations. This consistently happened, and we can offer no explanation for this behavior.

We now consider the results for C124 in Table 11. Again our results are quite close to the expected values. Relation generation takes significantly longer with our

TABLE 12. C155 Results

	JMS EG Estimate	Opt. EG Estimate	Sieving Result
Initialization Time	5.95m	5.95m	6m 4.87s
Total Relations	136533	136533	136533
Full Relations	136533	96662	99226
Partial Relations	-	633522	612780
Intersections	-	39871	51660
Unique Intersections	-	39871	51660
Sieve Polynomials	-	-	3036124745
Total Divisors Tested	188193560300	133236952000	99484699519415
Total Search Time	8492.67d	6338.01d	1720d 19h 37m 34.28s
Real Search Time	33.17d	24.76d	6d 21h 15m 44.92s
Total Iterations	-	-	1
Special Divisors Checked	-	-	177531606
Special Rels Time	-	-	6m 35.15s
Linear Algebra Time	2.47d	2.80d	14d 5h 51m 26.46s
Total Time	8495.14d	6340.81d	1735d 1h 30m 6.81s
Real Total Time	35.65d	27.56d	21d 7h 39m 17.12s

optimized version of Enge-Gaudry than with the version in [24]. However, we note that due to the larger factor base required in [24], we expect the linear algebra to take significantly longer in that case. Since the difference in the search time can be spread out over a parallel system but the linear algebra cannot, our settings should be better, and increasing the number of processors would only improve that situation. However the most dramatic result that we see is that from the sieve-based algorithm. Here the search time is less than the time using [24], and additionally, while the linear algebra is worse than that in our optimal Enge-Gaudry case, it is significantly better than that using the settings from [24], giving us a very significant improvement.

Finally we examine the results for C155 in Table 12. Once again we see that the use of sieving results in a much faster search for relations when compared to the random walk estimates we have computed. In this case, the search takes approximately a quarter of the time that we expect to take with the Enge-Gaudry algorithm. As mentioned before, the linear algebra estimates are very wrong and require further investigation. Because the linear algebra algorithm used depends both on the size of the matrix and the number of non-zero entries in the matrix, we see the large jump in linear algebra time from the C124 example to the C155 example. This result makes computing linear algebra estimates for the C155 case

difficult. In any case, because our parameter search suggests that we would be using the same size factor base for both the Enge-Gaudry algorithm and the sieve-based algorithm, we would expect the linear algebra time to be close to the same for both. Correcting our optimal Enge-Gaudry estimates with this observation, we see that the sieve-based algorithm takes little more than half the time we expect the Enge-Gaudry algorithm to take in its best case

Our results show that using sieving to compute relations can be much faster than using random walks. In conjunction with our adaptation of Vollmer's algorithm for solving the discrete logarithm problem, we have a more efficient algorithm for computing discrete logarithms in divisor class groups of hyperelliptic curves defined over even characteristic finite fields.

6. Further Work

There are several potential improvements and extensions possible with this work. A number of these, most of which are already work in progress, are listed below.

Although the introduction of large primes and sieving does result in significant performance improvements in practice as opposed to the random walks strategy, there is still further room for improvement. The empirical analysis used to find optimal parameters for our improved versions of the random walks strategy can likely be extended to our sieve-based version of Vollmer's algorithm. While we do some analysis in this case, the final parameter selection is currently done empirically. A careful analysis may yield optimal parameter choices without the experimental trials.

Further algorithmic improvements are also forseable. A double large prime variant, again adapted from integer factorization, is described in [15] for solving the HCDLP in the low genus case. It should be equally applicable to high-genus curves; work on adapting our optimal parameter selection strategy is in progress. Improved sieving and smoothness testing methods, including the multi-dimensional array approach mentioned in Section 4.1 and Bernstein's batch smoothness test [4], may also yield improvements in performance.

The examples we chose for this paper are interesting in that they shed some light on the feasibility of the Weil descent attack on the elliptic curve discrete logarithm problem. However, it would certainly be interesting the explore the utility of our algorithms in additional settings, including lower-genus curves and curves defined over odd characteristic finite fields. An investigation into the performance of the algorithm as g and q vary would also be useful in finding the limits of our methods.

It is well-known [9, 40] that the index calculus methods described in this paper can be adapted easily to compute the order and structure of the Jacobian $J_C(k)$. Roughly, the same relation generation strategies can be employed, including the improvements described in this paper, resulting in a relation matrix A whose columns correspond to principal divisors. Computing the determinant of the lattice generated by the columns of A and the Smith normal form of A yields $\#J_C(k)$ and the elementary divisors of $J_C(k)$, respectively. The required linear algebra is significantly harder than solving $A\vec{x} = \vec{b} \pmod{\#J_C(k)}$ as is required for compting the HCDLP. It would be interesting to extend our empirical analysis to this case taking into account the increased complexity of the linear algebra, and see how efficiently we can compute these invariants with our improved algorithms.

Finally, in [34], Müller, Stein, and Thiel present index calculus algorithms for computing the regulator and solving the infrastructure discrete logarithm problem in a real quadratic function field. Recent work by Hammell [19, 20] shows that the sieving approach discussed here gives similar results in this setting. However, attempts to apply sieving to real quadratic function fields of odd characteristic were not as successful. More work is required to determine whether sieving can be made more efficient in this setting, and in the low genus case.

References

1. W. R. Alford and C. Pomerance, *Implementing the self-initializing quadratic sieve on a distributed network*, Proceedings of International Conference "Number Theoretic and Algebraic Methods in Computer Science" (Moscow, 1993) (A.J. van der Poorten, I. Shparlinski, and H.G Zimmer, eds.), World Scientific, 1995, pp. 163–174.
2. ATLAS, *Automatically tuned linear algebra software*, Software, 2007, See `http://math-atlas.sourceforge.net/`.
3. E. Bach and J. Shallit, *Algorithmic number theory*, MIT Press, Cambridge, Massachusetts and London, England, 1996.
4. D. Bernstein, *How to find small factors of integers*, to appear in Math. Comp.
5. D. Cantor, *Computing in the Jacobian of a hyperelliptic curve*, Mathematics of Computation **48** (1987), 95–101.
6. D. G. Cantor and H. Zassenhaus, *A new algorithm for factoring polynomials over finite fields*, Math. Comp. **36** (1981), no. 154, 587–592.
7. H. Cohen and G. Frey, editors. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, Number 34 in Discrete Mathematics and Its Applications. Chapman& Hall/CRC, 2005.
8. S. Contini, *Factoring integers with the self-initializing quadratic sieve*, Master's thesis, University of Georgia, Athens, Georgia, 1997.
9. A. Enge and P. Gaudry, *A general framework for subexponential discrete logarithm algorithms*, Acta Arithmetica **102** (2002), 83–103.
10. R. Flassenberg and S. Paulus, *Sieving in function fields*, Experimental Mathematics **8** (1999), 339–349.
11. G. Frey, *How to disguise an elliptic curve (Weil descent)*, Talk at ECC '98, Waterloo, 1998. Slides available from `http://www.cacr.math.uwaterloo.ca/conferences/1998/ecc98/slides.html`
12. G. Frey, *Applications of arithmetical geometry to cryptographic constructions*, Proceedings of the Fifth International Conference on Finite Fields and Applications, Springer, 2001, pp. 128–161.
13. P. Gaudry, *An algorithm for solving the discrete log problemon hyperelliptic curves*, Advances in Cryptology - EUROCRYPT '2000, Lecture Notes in Computer Science, vol. 1807, 2000, pp. 19–34.
14. P. Gaudry, F. Hess, and N. Smart, *Constructive and destructive facets of weil descent on elliptic curves*, Journal of Cryptology **15** (2002), 19–46.
15. P. Gaudry, E. Thomé, N. Thériault, and C. Diem, *A double large prime variation for small genus hyperelliptic index calculus*, Math. Comp. **76** (2007), no. 257, 475–492.
16. P. Gaudry, *On breaking the discrete log on hyperelliptic curves*, Advances in Cryptology - Eurocrypt 2000, Lecture Notes in Computer Science, vol. 1807, 2000, pp. 19–34.
17. GCC, *GCC, the GNU compiler collection*, Software, 2007, See `http://gcc.gnu.org/`.
18. GMP, *The GNU multiple precision bignum library*, Software, 2007, See `http://gmplib.org/`.
19. J. F. Hammell, *Index calculus in the infrastructure of real quadratic function fields*, Master's thesis, University of Calgary, Canada, 2008.
20. J. F. Hammell and M. J. Jacobson, Jr., *Index-calculus algorithms in real quadratic function fields*, In preparation, 2011.
21. M. J. Jacobson, Jr., *Applying sieving to the computation of quadratic class groups*, Math. Comp. **68** (1999), no. 226, 859–867.
22. ———, *Subexponential class group computation in quadratic orders*, Ph.D. thesis, Technische Universität Darmstadt, Darmstadt, Germany, 1999.

23. ———, *Computing discrete logarithms in quadratic orders*, Journal of Cryptology **13** (2000), 473–492.
24. M. J. Jacobson, Jr., A. J. Menezes, and A. Stein, *Solving elliptic curve discrete logarithm problems using Weil descent*, J. Ramanujan Math. Soc. **16** (2001), no. 3, 231–260.
25. M. J. Jacobson, Jr., R. Scheidler, and A. Stein, *Fast arithmetic on hyperelliptic curves via continued fraction expansions*, Advances in Coding Theory and Cryptology (T. Shaska, W.C. Huffman, D. Joyner, and V. Ustimenko, eds.), Series on Coding Theory and Cryptology, vol. 3, World Scientific Publishing, 2007, pp. 201–244.
26. N. Koblitz, *Elliptic curve cryptosystems*, Mathematics of Computation **48** (1987), 203–209.
27. N. Koblitz, *Hyperelliptic cryptosystems*, Journal of Cryptology, **1** (1989), 139–150.
28. A. K. Lenstra and H. W. Lenstra, Jr., *The development of the number field sieve*, Lecture Notes in Mathematics, vol. 1554, Springer-Verlag, Berlin, 1993.
29. LinBox, *Project LinBox: Exact computational linear algebra*, Software, 2007, See `http://www.linalg.org/`.
30. M. Maurer, A. J. Menezes, and E. Teske, *Analysis of the GHS Weil descent attack on the ECDLP over characteristic two finite fields of composite degree*, LMS Journal of Computation and Mathematics **5** (2002), 127–174.
31. A. Menezes, Y.-H. Wu, and R. J. Zuccherato, *An elementary introduction to hyperelliptic curves*, Algebraic Aspects of Cryptography, Algorithms and Computation in Mathematics, vol. 3, Springer Verlag, Berlin, 1998, pp. 155–178.
32. MPI, *Message passing interface*, Software, 2007, See `http://www-unix.mcs.anl.gov/mpi/`.
33. V. Miller, *Uses of elliptic curves in cryptography'*, Advances in Cryptology - CRYPTO 85, Lecture Notes in Computer Science, vol. 218, 1986, pp. 417–426.
34. V. Müller, A. Stein, and C. Thiel, *Computing discrete logarithms in real quadratic congruence function fields of large genus*, Math. Comp. **68** (1999), 807–822.
35. C. Pomerance, *The quadratic sieve factoring algorithm*, Advances in Cryptology - EUROCRYPT 84, Lecture Notes in Computer Science, vol. 209, 1985, pp. 169–182.
36. V. Shoup, *NTL: A library for doing number theory*, http://www.shoup.net, 2008.
37. E. Teske, *Speeding up Pollard's rho method for computing discrete logarithms*, Algorithmic Number Theory - ANTS-III (Portland, Oregon), Lecture Notes in Computer Science, vol. 1423, Springer-Verlag, Berlin, 1998, pp. 541–554.
38. N. Thériault, *Index calculus attack for hyperelliptic curves of small genus*, Advances in Cryptology - ASIACRYPT 2003, Lecture Notes in Computer Science, vol. 2894, 2003, pp. 75–92.
39. M. D. Velichka, *Improvements to index calculus algorithms for solving the hyperelliptic curve discrete logarithm problem over characteristic two finite fields*, Master's thesis, University of Calgary, Canada, 2008.
40. U. Vollmer, *Asymptotically fast discrete logarithms in quadratic number fields*, Algorithmic Number Theory — ANTS-IV, Lecture Notes in Computer Science, vol. 1838, 2000, pp. 581–594.
41. J. von zur Gathen and J. Gerhard, *Modern computer algebra*, Cambridge University Press, 1999.

Institute for Security, Privacy and Information Assurance, University of Calgary, 2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4
E-mail address: `mdvelich@ucalgary.ca`

Institute for Security, Privacy and Information Assurance, University of Calgary, 2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4
E-mail address: `jacobs@cpsc.ucalgary.ca`

Institut für Mathematik, Carl-von-Ossietzky Universität Oldenburg, D-26111 Oldenburg, Germany
E-mail address: `andreas.stein1@uni-oldenburg.de`