UNIVERSITY OF CALGARY

Improvements To Index Calculus Algorithms For Solving The Hyperelliptic Curve Discrete Logarithm Problem Over Characteristic Two Finite Fields

by

Mark Velichka

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

APRIL, 2008

UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Improvements To Index Calculus Algorithms For Solving The Hyperelliptic Curve Discrete Logarithm Problem Over Characteristic Two Finite Fields" submitted by Mark Velichka in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE.

Supervisor,
Dr. Michael J. Jacobson, Jr.
Department of Computer Science

Dr. Wayne Eberly
Department of Computer Science

Dr. Bill Sands
Department of Mathematics and Statistics

April 11, 2008
Date

Abstract

Public key cryptosystems are based on mathematical problems that are "hard" to solve, such as the hyperelliptic curve and elliptic curve discrete logarithm problems (HCDLP, ECDLP). Using Weil descent one can, in certain cases, reduce an instance of the ECDLP to an instance of the HCDLP, justifying investigation of efficient algorithms for solving the HCDLP.

Index calculus has been used to solve the HCDLP for some time ([7], [22]), and improvements have been made in the small-genus case [65]. As Weil descent can produce high-genus curves, our interest lies there.

This thesis investigates a variety of practical improvements to the high-genus case, including large primes [65] and sieving [24]. We find that in practice these improvements result in a significant performance increase, allowing us to solve an HCDLP example produced from an instance of the ECDLP on a curve defined over $\mathbb{F}_{2^{155}}$ for the first time.

Acknowledgements

I would like to thank my supervisor, Dr. Michael J. Jacobson, Jr., for providing me with the opportunity to pursue this research. In addition, I would like to thank him for his support, direction, helpful comments, and encouragement during my time at the University of Calgary. I have learned much more than what is contained on the surface of this thesis. I'd also like to thank my committee members for their helpful comments and suggestions.

I would also like to thank the University of Calgary, NSERC and iCORE for the funding I received from them in support of my work. I would also like to thank the Department of Computer Science for providing me the opportunity to TA, an enjoyable experience that all students should have a chance to take part in.

Thanks to everybody who has been through ICT 609. In particular, thanks to Shantha, Karel, Jonathan and Ryan for being there to answer questions, discuss results, and provide quick stress relief. Thanks also to the students in the math department, and everybody else in CISaC for their support.

A big thanks to my family, who encouraged me to do the best that I could, and who provided all the support I needed from them, when I needed it from them. Also, thanks to my friends, specifically Emilia and Paul, who supported me and constantly reminded me that people outside my study area could find my work interesting.

Finally, I would like to thank God for blessing me with the abilities that I have, without which none of this would have been possible.

Table of Contents

List of Tables

List of Symbols and Nomenclature

$\mathbb{Z}$	the integers
$f(n) \in O(g(n))$	if $f(n) \leq cg(n)$ for some constant c and sufficiently large n
$f(n) \in \Theta(g(n))$	if $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for some constants c_1 and c_2, and sufficiently large n
$\vec{v}$	vector with n elements $(v_1, v_2, \cdots, v_n)$
v_i	i^{th} entry in $\vec{v}$
$k = \mathbb{F}_q$	finite field of size q
$\bar{k}$	algebraic closure of k
K	extension field of k
JMS	Jacobson, Menezes and Stein [35]

Hyperelliptic Curves

E	elliptic curve
C	hyperelliptic curve, defined by $v^2 + h(u)v = f(u)$
g	genus
P	point on a curve
$\tilde{P}$	opposite of point P
$\text{ord}_P(G)$	order of polynomial function G at point P
∞	point at infinity
J	Jacobian of C
$\bar{k}[C]$	coordinate ring of C over $\bar{k}$
$\bar{k}(C)$	function field of C over $\bar{k}$

Divisors

D^0	degree 0 divisor
$D = \text{div}(a, b)$	divisor
$\mathbb{P}$	principal divisors
$\text{ord}_P(D)$	order of divisor D at point P
$\text{div}(R)$	divisor of a rational function R

Quadratic Order and Ideals

$\mathcal{O}_K$	quadratic order
$Cl(K)$	ideal class group
$\mathfrak{a}$	$\mathcal{O}_K$-ideal
$N(\mathfrak{a})$	norm of an $\mathcal{O}_K$-ideal
$\bar{\mathfrak{a}}$	conjugate of an $\mathcal{O}_K$-ideal
$\mathfrak{a}^{-1}$	inverse of an $\mathcal{O}_K$-ideal
(α)	ideal generated by α
$\mathcal{I}$	set of fractional ideals
$\mathcal{P}$	set of prime ideals
$\mathcal{P}_t$	prime ideals having norms with degree $\leq t$
$\mathfrak{a} \sim \mathfrak{b}$	ideal equivalence

Algorithm Symbols and Parameters

B	factor base
t	degree bound
r	bound on degree of the norms of the elements in B
M	sieve radius
Y	tolerance
j	number of prime ideals used to form sieve polynomial
$F(S) = aS^2 + bS + c$	sieve polynomial

Chapter 1

Introduction

Elliptic curve cryptography has been receiving more attention as of late. First presented by Koblitz [42], it has evolved to the point where it is used quite frequently in a number of implementations. One of the primary reasons for this interest is that it appears that the smaller key size used in elliptic curve cryptography can provide the same amount of security as conventional public key cryptographic algorithms using a larger key size. For example, in order to obtain equivalent levels of security for signatures with the Digital Signature Algorithm and the elliptic curve variation, ECDSA, one should use a 1024-bit key with DSA, but can use a 160-bit key with ECDSA, as outlined by Koblitz [44]. This makes elliptic curve cryptography very tempting to use in settings where memory and physical size are important to consider, such as cell phones, smart cards or secure RFID chips. One merely needs to look at the Certicom website (`http://www.certicom.com/`) to see how large of a business elliptic curve cryptography has become. An even larger development is the inclusion of two elliptic curve variations of cryptographic algorithms in the NSA Suite B set of algorithms, found on the NSA website [55]: the Elliptic Curve Diffie-Hellman key agreement protocol, as described in a Certicom Research technical report [13], and the Elliptic Curve Digital Signature Algorithm, covered by Koblitz [44] and Certicom Research [13].

The security of both of these algorithms relies on the difficulty of the *discrete logarithm problem* in the group of points on an elliptic curve. That is, given two points, P and Q, where Q is a multiple of P, on an elliptic curve defined over a finite field, find the smallest positive integer x such that $Q = xP$. There are no known subexponential algorithms for solving the elliptic curve discrete logarithm problem in general. However, there are certain instances of the elliptic curve discrete logarithm problem that can be reduced to instances of the hyperelliptic curve discrete logarithm problem that can be solved in subexponential time.

The procedure that converts instances of the elliptic curve discrete logarithm problem to instances of the hyperelliptic curve discrete logarithm problem is called *Weil descent*. It was first presented by Frey [25] and was improved and further studied by Gaudry, Hess and Smart [27]. It should be noted that the instances that Gaudry, Hess and Smart concentrate on are over a field with characteristic two, and for the remainder of the thesis we assume we are dealing with an underlying field with characteristic two. This algorithm was implemented by Jacobson, Menezes and Stein [35]. This method was also further studied by Menezes and Qu [51] and by Maurer, Menezes and Teske [49].

In addition to this reduction, hyperelliptic curves are also of interest for their own use in cryptographic algorithms. Koblitz [43] presented the idea of using hyperelliptic curves defined over finite fields in a variation of the Diffie-Hellman key agreement protocol. This makes use of the discrete logarithm problem for Jacobians of hyperelliptic curves. One should note that hyperelliptic curves extend the idea of elliptic

curves. In fact, an elliptic curve is just a genus one hyperelliptic curve. Expanding on the advantage of elliptic curve cryptosystems, it was once suspected that for small genus, such as genus 2 or 3, an even smaller key size could be used to provide an equivalent security level to conventional or even elliptic curve algorithms, mentioned by Jacobson, Menezes and Stein [36]. However, later work, such as the analysis done by Thériault [65], showed that this does not hold.

Because of both the idea of reducing the elliptic curve discrete logarithm problem to the hyperelliptic curve discrete logarithm problem and the use of hyperelliptic curves in their own cryptosystems, algorithms for solving the hyperelliptic curve discrete logarithm problem have become of great interest. The focus of this thesis is the study of some of these algorithms and improvements to them, primarily in the high genus case.

1.1 Previous Work

The discrete logarithm problem, for a general cyclic group G with group operation written additively, is as follows. Given a generator $a \in G$ and b in the subgroup generated by a, find the least positive integer x such that $xa = b$. There is a very obvious way to find such an x: simply compute a, $2a$, $3a$ and so on until x is reached such that $xa = b$. If x is large, this is not an efficient solution. In fact, one could end up computing every element in the group G.

In 1971, Shanks [61] introduced an algorithm to solve discrete logarithm problems

using a procedure called Baby-Steps, Giant-Steps that runs in time $O(\sqrt{|G|})$. This algorithm was followed in 1978 by a probabilistic algorithm of *Las Vegas* type of Pollard [59] that has an expected run time of $O(\sqrt{|G|})$ in any cyclic group G, and will always return a correct answer. Pollard's algorithm has the advantage of using less memory than that of Shanks. Baby-Steps, Giant-Steps creates a look-up table in the first part of execution and the larger the table, the better the run-time. The balance is thus $O(\sqrt{|G|})$ storage. Pollard's algorithm uses $O(1)$ storage. It is known as Pollard's Rho method because the search space can be presented in a shape similar to the Greek letter ρ. In this method, two sequences in the group G are computed, and when the sequences have the same value for the same index, further computations are performed to compute the discrete logarithm. In 1998, an improvement to Pollard's method was published by Teske [64]. The improvement was made to the random walk used by Pollard, resulting in a reduction of the expected number of elements that had to be computed, and thus improving the overall computation time of the algorithm. A *random walk* is a random traversal from element to element in a group. Teske's random walk idea appears again later.

McCurley [50] and Hafner with McCurley [29], both published in 1989, present Las Vegas type algorithms for performing discrete logarithm computations in, and computing class numbers of, imaginary quadratic number fields. The method they use is called *index calculus*. Index calculus algorithms have two major phases. Let G be the group in which we are attempting to solve the discrete logarithm problem. Then the first major step is to find elements in G that factor completely over a

predefined subset of G that we call the *factor base*. The second phase is a linear algebra step. We focus on the first phase as linear algebra has been studied at great length. For example, see the work done by Wiedemann [69] and the related work by Kaltofen and Saunders [41], LaMacchia and Odlyzko [46], Kaltofen [40], Chen, Eberly, Kaltofen, Saunders, Turner and Villard [14] and Mulders [54] as well as the work by Lanczos [47] and related work by Eberly and Kaltofen [20] and Montgomery [53].

Index calculus was used in a subexponential algorithm for computing discrete logarithms in Jacobians of high-genus hyperelliptic curves by Adleman, DeMarrais and Huang [7] in 1994. This method was improved upon by Gaudry [26] in 2000 and Enge [21] in 2002. Enge and Gaudry [22] then developed a general index calculus framework for solving discrete logarithm problems in a group G, given the size of that group and making assumptions on the smoothness of elements in the group. This framework runs in subexponential time in $|G|$ under the assumption that there are a subexponential proportion of elements in the group that are smooth over a certain subset of group elements.

Thériault [65] analyzes the use of *large primes* and random walks in Gaudry's algorithm for the computation of discrete logarithms in the Jacobian of small genus hyperelliptic curves. Large primes had been used in integer factorization, such as work done by Kurowski [45], Lenstra and Manasse [48] and Boender and te Riele [10]. They have also been used in algorithms for finding the class group of a quadratic number field, such as by Jacobson [33]. However, the analysis done by Thériault

was new, and promising. This work focused on the small genus case, and showed that small genus hyperelliptic curve cryptosystems do not have the level of security once thought.

Independent of this work, Vollmer [66] published a subexponential algorithm for solving the discrete logarithm problem in quadratic number fields in 2000. This algorithm, while not requiring the class number, which is often unknown and difficult to compute, can be modified to make use of it. However, the interesting part of this algorithm for us is that it allows us to consider a different method for finding relations. The Enge-Gaudry algorithm is well-suited for the use of random walks, but it is not very flexible. Vollmer's algorithm allows us to use an idea called *sieving* to find relations. Note that Vollmer's original presentation does not make use of sieving for finding relations.

Sieving has been used extensively in the context of factoring by Pomerance [60], Silverman [63], Boender and te Riele [10] and Kurowski [45], for example. It has also been used in the computation of class groups of quadratic number fields by Jacobson [32, 33], in conjunction with large primes. Jacobson [34] has also used sieving to compute discrete logarithms in the class group of a quadratic order in. In 1999, Flassenberg and Paulus [24] published a paper on sieving in the context of quadratic function fields of odd characteristic. This is currently the only known use of sieving in this context, and thus hyperelliptic curve context.

In addition to using large primes with sieving, a method for efficiently generat-

ing sieving polynomials has been developed, called *self-initialization*. It has been used in the factoring context by Alford and Pomerance [8] and Contini [16], and by Jacobson [33] in his sieving algorithm for computing class groups in quadratic number fields. It is suggested by Flassenberg and Paulus that it may be applicable in algorithms for performing computations in quadratic function fields as well.

The work of Enge, Gaudry, Thériault, Jacobson, Menezes and Stein making use of random walks and the work of Jacobson on sieving has directed the work done in this thesis.

1.2 Contribution

When Thériault [65] analyzed large primes in the Jacobian of hyperelliptic curves he did so in the small genus case. In particular, he assumes that q, the size of the underlying field, is large when compared to the genus. In these cases, there are a large number of degree one prime divisors, that is, prime divisors corresponding to irreducible linear polynomials over the finite field. By varying the number of them which are included in the factor base, Thériault reduced the size of the factor base and thus the size of the matrix used in the resulting linear algebra problem. Additionally, by considering large primes (the prime divisors not in the factor base) he was able to reduce the asymptotic runtime of Gaudry's algorithm [26].

In this thesis we take cues from Thériault [65] and attempt to mirror his work in the large genus case. The Jacobian of an imaginary hyperelliptic curve is isomorphic to

the ideal class group of an imaginary quadratic function field. We focus our work in the quadratic function field context, and discuss ideals rather than divisors. Let t be the bound on the degree of the norms of the elements in our factor base. That is, the prime ideals in the factor base correspond to the irreducible polynomials over the underlying finite field that have degree less than t. Then we can consider four new variations of the Enge-Gaudry algorithm. First, we consider only reducing the number of prime ideals having norms with degree equal to t in the factor base. While this does not provide the same improvement that Thériault saw, we are not finished. Because our factor base contains ideals with norms of different degrees, the idea to vary what sort of prime ideals become our large primes is obvious. One could use only the prime ideals with norm of degree equal to t that are not in the factor base as large primes, as in Thériault's work. One could use all of the prime ideals having norms with degree equal to t in the factor base and then use only the prime ideals having norms with degree equal to $t+1$ as large primes. Or, one could combine the two, reducing the size of the factor base and increasing the number of large primes. We consider all three variations of large primes, giving a total of five variations of the Enge-Gaudry algorithm.

Jacobson, Menezes and Stein [35], present formulas for estimating the number of random walk steps required to find enough relations to compute the discrete logarithm problem using the Enge-Gaudry algorithm. Part of the work in this thesis involves generalizing these formulas to compute estimated number of steps required in our four new variations of the Enge-Gaudry algorithm. Jacobson, Menezes and Stein also provide four instances of the discrete logarithm problem in four different

hyperelliptic curves in which to compute. We use these same instances in our tests. In the context of these examples, we proceed to evaluate the estimating formulas in order to compare the different variations. This allows us to determine the optimal strategy and factor base size empirically. According to these computations, we expect the variation in which the factor base contains all prime ideals having norms with degree less than or equal to t in the factor base and using only the prime ideals having norms with degree equal to $t+1$ as large primes to perform the best for the examples we consider.

We noted above that the Flassenberg and Paulus presentation of sieving [24] is described for odd characteristic underlying finite fields. In this thesis we generalize sieving to the even characteristic case. We also present self-initialized sieving in the even characteristic function field setting. In addition to this improvement, we discuss different methods of implementing the sieve array and discuss using only a subset of the factor base for sieving in order to increase the speed of the procedure. Finally, since this is, as far as we know, the first place where sieving is investigated in this setting, we try to provide some guidance for choosing parameters for sieving.

In addition to studying and presenting the above algorithms and our proposed changes, we have also implemented the algorithms discussed as part of a larger C++ package called ANTL: Algebraic Number Theory Library. This is built on top of Shoup's NTL, a library for doing number theory [62]. The goal ANTL is to provide a general library for performing discrete logarithm and class group computations in both quadratic number fields and quadratic function fields. This library

is currently in a very preliminary stage. We focus our discussion on the implementation for quadratic function fields. In this library we have implemented the Enge-Gaudry algorithm, including the studied variations, and Vollmer's algorithm with self-initialized sieving. With this implementation we attempted to solve the discrete logarithm problems investigated by Jacobson, Menezes and Stein (JMS) [35], examples over curves they call C62, C93, C124 and C155. Note that the C155 example has not been computed previously. These results verify the estimates we computed earlier, showing that using large primes does improve the performance of the Enge-Gaudry algorithm. For example, while there is little difference in the search time for C62, our settings also result in using a smaller factor base, which improves the linear algebra stage, and so the overall algorithm requires only one quarter the time with our settings versus those provided by JMS [35]. In the C93 case, both our parameters and those suggested by JMS [35] use the same factor base. However, using large primes results in a search time that is only two-thirds the time required without large primes. For C124, we use a smaller factor base, which doubles the search time, but decreases the linear algebra time from over three days to under an hour. This does not decrease the total time of the algorithm, but it does reduce the real time when the search is done in parallel over several processors. We only compute expected results for the C155 example, but work suggests that using our parameters will result in a search time that is three-quarters the time required by the parameters suggested by JMS [35].

An even more substantial improvement is the performance of Vollmer's algorithm with sieving, proving to find relations much faster than our fastest Enge-Gaudry

variations. In the C62 example the search time required by sieving was only slightly improved over the Enge-Gaudry time, and the linear algebra stage had to be repeated after finding a few more relations before a solution to the linear system could be found, resulting in a total sequential time that was faster than both the Enge-Gaudry implementations if we suppose the work was done on a single processor, but slower real time when we consider the result of performing the relation search in parallel. The C93 example begins to really support the case for sieving. Here, the total search time was one quarter that required by the fastest Enge-Gaudry variation. Again, the linear algebra had to be repeated, and while the total sequential time was better, doing the search in parallel results in a slower real time. In the C124 example using sieving in the search again speeds up the stage by a factor of four. This time, the linear algebra did not have to be repeated, and so both the total sequential time and real time used are approximately four times faster using Vollmer's algorithm with sieving than using our fastest Enge-Gaudry parameters. Finally, due to these improvements, we were able to compute the C155 example for the first time. Using sieving resulted in a search that was once again approximately four times faster than the search time expected for our optimal Enge-Gaudry parameters. Unfortunately our results show that we grossly underestimated the amount of time required for the linear algebra stage for the C155 example, but correcting for this demonstrates again the success of sieving in this example. These results clearly indicate that sieving can significantly improve performance in practice, and that sieving should be studied more.

1.3 Outline

Chapter 2 of this thesis provides the background on hyperelliptic curves and quadratic function fields necessary for the remainder of the thesis. This includes describing the discrete logarithm problem in the Jacobian of a hyperelliptic curve, and the ideal class group of a quadratic function field, as well as discussing the equivalence of the two problems. We also mention how this ties in with elliptic curves. While results are stated, proofs are left to the included references.

In Chapter 3 we cover the algorithmic background for this thesis. This includes both the Enge-Gaudry algorithm [22] and Vollmer's algorithm [66] for solving the discrete logarithm problem in the ideal class group of a quadratic function field. This starts with a general overview of index calculus algorithms, and includes thorough descriptions of two methods for finding relations: random walks, as suggested by Teske [64] and used by Jacobson, Menezes and Stein [35], and sieving, presented by Flassenberg and Paulus [24] for fields of odd characteristic. Thériault [65] studies the uses of large primes in solving the hyperelliptic curve discrete logarithm problem in low genus cases using the Enge-Gaudry algorithm with random walks. We also present that in this chapter. There is also an improvement to sieving, called self-initialization, that has been used in similar contexts, such as computing the class group of a quadratic number field, as done by Jacobson [33]. This work is also presented here. Finally, we briefly discuss the linear algebra that is used in these index calculus algorithms.

Chapter 4 presents our improvements and novel work in this area. We show how to apply the idea of large primes to the high genus cases. This applies both to the use of random walks and sieving. We also present the work done by Jacobson, Menezes and Stein [35] to compute an estimated number of random walk steps necessary to compute sufficient relations for the Enge-Gaudry algorithm to run successfully. We extend these computations to include the large prime ideas, providing a number of variations of this algorithm that can be tested. We also generalize the Flassenberg and Paulus method of sieving [24] to even characteristic fields, and incorporate self-initialization. We also discuss implementation of sieving and attempt to provide guidance for selecting appropriate sieve parameters.

Finally, Chapter 5 contains an outline of our implementation, including a description of the platform on which we are testing our program, the details of the implementation that may not have been covered before, and precisely what examples we are testing on. These examples are taken from Jacobson, Menezes and Stein [35] and in addition to running them with the same algorithm as done in that paper, we use our formulas for estimating random walk steps from Chapter 4 to derive settings that result in a faster runtime than those from [35]. We also perform computations using our implementation of sieving with self-initialization and large primes, demonstrating its efficiency for finding relations. Finally, we compare our results in all test cases with the estimated results and provide interpretation of the data.

Chapter 2

Quadratic Function Fields

One current major area of interest for cryptographers is elliptic curves. The group of points on an elliptic curve is a particularly useful setting for cryptographic algorithms. Two algorithms that take advantage of this group are the Elliptic Curve Diffie-Hellman key agreement protocol, described by Certicom Research [13], and the Elliptic Curve Digital Signature Algorithm, presented by Koblitz [44] and Certicom Research [13], both of which are part of the NSA Suite B set of cryptographic algorithms, described on the NSA website [55].

Elliptic curve cryptosystems are of interest because frequently the same level of security can be provided by a much smaller key. For example, in order to obtain an equivalent of 80 bits of security, the DSA key used must be 1024-bits long where ECDSA uses a 160-bit key, as described by Koblitz [44]. That is, the size of the field over which the elliptic curve is defined should be approximately 2^{160}. For more information about elliptic curve cryptosystems, see the initial presentation by Koblitz [42].

The security of many of the elliptic curve cryptosystems relies on the difficulty of the discrete logarithm problem in the group of points on an elliptic curve. The *discrete logarithm problem*, written for a multiplicative cyclic group G is: given a

generator $a \in G$ and $b = a^x$, find the least positive integer x satisfying $a^x = b$. For an additive group, an example of which is the group of points on an elliptic curve, this is written as $b = xa$. There are no known subexponential algorithms for solving general instances of the elliptic curve discrete logarithm problem (ECDLP). However certain instances of the ECDLP can be solved through a process known as the *GHS attack*, introduced by Gaudry, Hess and Smart [27]. This attack uses a procedure called Weil descent, that reduces certain instances of the ECDLP to instances of the hyperelliptic curve discrete logarithm problem for which there are in some cases subexponential algorithms known.

Hyperelliptic curves are also of interest in their own right because, although the points on a curve of genus greater than 1 do not form a group, hyperelliptic curves can still be used for cryptography. The Jacobian of a hyperelliptic curve forms a group in which the discrete logarithm problem is not trivial. For an example of an algorithm that makes use of this, see the variation of the Diffie-Hellman key agreement protocol proposed by Koblitz [43] and presented in the Handbook of Elliptic and Hyperelliptic Curve Cryptography [15].

It turns out that hyperelliptic curves are related to quadratic function fields. In fact, the Jacobian of a hyperelliptic curve is isomorphic to the class group of a quadratic function field. In this chapter we briefly discuss hyperelliptic curves, introduce quadratic function fields, and describe the link between them. Note that we do not cover Weil descent in much depth, but we do discuss it a little more in Chapter 5. We also discuss the discrete logarithm problem and its importance.

2.1 Hyperelliptic Curves

This section provides a brief introduction to hyperelliptic curves. For more information, the interested reader can consult the appendix by Menezes, Wu and Zuccherato [44] and the Handbook of Elliptic and Hyperelliptic Curve Cryptography [15].

Let $k = \mathbb{F}_q$ be a finite field with $|k| = q$ and let $\bar{k}$ be the algebraic closure of k. Recall that this means every polynomial with coefficients in $\bar{k}$ has a root in $\bar{k}$. Let $f(u) \in k[u]$ be a monic polynomial with degree $2g+1$ or $2g+2$ and $h(u) \in k[u]$ be a polynomial with degree less than or equal to g for some $g \geq 1$. Further, suppose there are no points $(u, v) \in \bar{k} \times \bar{k}$ that satisfy both $v^2 + h(u)v = f(u)$ and the equations $2v + h(u) = 0$ and $\frac{\partial h}{\partial u}(u)v - \frac{\partial f}{\partial u}(u) = 0$. Note that the last two equations are partial derivatives with respect to v and u, respectively.

Definition 2.1 *(Jacobson, Scheidler and Stein [38])*
We call

$$C : v^2 + h(u)v = f(u)$$

satisfying the above conditions with $\deg(f(u)) = 2g + 1$ *an* imaginary hyperelliptic curve of genus g over k.

Just as there are imaginary and real quadratic number fields, there are imaginary and *real* hyperelliptic curves. It is possible to convert curves from the imaginary form to the real form, and in certain cases, from real to imaginary, stated by Jacobson, Scheidler and Stein [37]. For the remainder of this thesis we will only consider the imaginary case. See Jacobson, Scheidler and Stein [37] for more information on

real curves.

Note that when $g = 1$ we have an elliptic curve. Also note that if the characteristic of the field is odd, we can write the curve as $C : v^2 = f(u)$ with a change of variables. That is, when the characteristic is not 2, we can assume $h(u) = 0$.

Let K be an extension field of k. That is, K is a subfield of $\bar{k}$ containing k. Then we consider the following set.

Definition 2.2 *(Menezes, Wu and Zuccherato [44, page 157])*
The set of K-rational points on C *is defined to be*
$C(K) = \{(u, v) \in K \times K \mid v^2 + h(u)v = f(u)\} \cup \{\infty\}$. *It is usually denoted simply as* C *and called the points on* C. *We call* ∞ *the* point at infinity.

Definition 2.3 *(Menezes, Wu and Zuccherato [44, page 157])*
Let $P = (x, y) \in C$. *Then the* opposite *of* P *is* $\tilde{P} = (x, -y - h(x))$. *The opposite of* ∞ *is* ∞.

If $P \in C$ then $\tilde{P} \in C$ as well.

When $g \neq 1$ we do not have a natural group law on the points on C. However, we do have a group if we consider the set of reduced divisors in the Jacobian of C over k. The remainder of this section leads us to that. First we consider the properties of polynomials when viewed as functions on a hyperelliptic curve.

Consider the ideal in $\bar{k}[u, v]$ generated by $v^2 + h(u)v - f(u)$.

Definition 2.4 *(Menezes, Wu and Zuccherato [44, page 159])*
We call the quotient $\bar{k}[C] = \bar{k}[u,v]/(v^2 + h(u)v - f(u))$ *the* coordinate ring *of* C *over* $\bar{k}$. *An element* $G(u,v) \in \bar{k}[C]$ *is called a* polynomial function *on* C.

Menezes, Wu and Zuccherato [44, page 159] state that the polynomial $v^2 + h(u)v - f(u)$ is irreducible in $\bar{k}$. In order to see this, suppose it is not for the purpose of arriving at a contradiction. Then $v^2 + h(u)v - f(u) = (a(u) - v)(b(u) - v) = a(u)b(u) - (a(u) + b(u))v + v^2$ for some $a(u)$, $b(u) \in \bar{k}[u]$. Then the degree of $a(u)b(u)$ equals the degree of $f(u)$, $2g+1$, and the degree of $a(u) + b(u)$ equals the degree of $h(u)$, which is less than or equal to g. Clearly this is impossible. Since $v^2 + h(u)v - f(u)$ is irreducible, $\bar{k}[C]$ is an integral domain.

For any polynomial $G(u,v)$ in any equivalence class in $\bar{k}[C]$ we can replace instances of v^2 with $f(u) - h(u)v$ and write it as $G(u,v) = a(u) + b(u)v$. This final form is unique, and thus this provides representatives for the equivalence classes of $\bar{k}[C]$, stated by Menezes, Wu and Zuccherato [44, page 159].

We have the following three functions on polynomials that will be useful later.

Definition 2.5 *(Menezes, Wu and Zuccherato [44, page 159])*
The conjugate *of a polynomial* $G(u,v) = a(u) + b(u)v$ *in* $\bar{k}[C]$ *is*

$$\bar{G}(u,v) = a(u) - b(u)(v + h(u)).$$

Definition 2.6 *(Menezes, Wu and Zuccherato [44, page 159])*
The norm *of a polynomial* G *in* $\bar{k}[C]$ *is defined to be* $N(G) = G\bar{G}$.

Note that the norm is multiplicative; that is, $N(GH) = N(G)N(H)$.

Definition 2.7 *(Menezes, Wu and Zuccherato [44, page 165])*
Let $G(u,v) = a(u) + b(u)v$ be a nonzero polynomial function and let $P \in C$. Then the order of G at P, *written $ord_P(G)$ is defined as follows:*

1. *If $P = (x,y)$ is not the point at infinity, let r be the highest power of $(u+x)$ that divides $a(u)$ and $b(u)$. Then $G = (u+x)^r(a_0(u)+b_0(u)v)$. If $a_0(x)+b_0(x)y \neq 0$, set $s = 0$. Otherwise let s be the highest degree of $(u+x)$ that divides $N(a_0(u)+b_0(u)v)$. If $P \neq \tilde{(P)}$, $ord_P(G) = r+s$, otherwise $ord_P(G) = 2r+s$.*
2. *If $P = \infty$, $ord_P(G) - \max\{2\deg_u(a), 2g+1+2\deg_u(b)\}$.*

Since $\bar{k}[C]$ is an integral domain, it is natural to consider fractions consisting of elements in $\bar{k}[C]$. This gives rise to the following definition.

Definition 2.8 *(Menezes, Wu and Zuccherato [44, page 160])*
The function field of C over $\bar{k}$ *is $\bar{k}(C) = \{\frac{G}{H} | G, H \in \bar{k}[C], H(u,v) \neq 0\}$, also called the field of fractions of $\bar{k}[C]$. An element $R \in \bar{k}[C]$ is called a* rational function *on C.*

We will now describe the set of divisors, allowing us to define the Jacobian of a hyperelliptic curve.

Definition 2.9 *(Menezes, Wu and Zuccherato [44, page 167])*
A divisor *is a formal sum of points on C, $D = \sum_{P \in C} m_P P$ where $m_P \in \mathbb{Z}$ and only a finite number of the coefficients m_P are non-zero. The* degree *of a divisor D is*

$\deg(D) = \sum_{P \in C} m_P$ *and the* order *of* D *at* P *is* $ord_P(D) = m_P$. *We denote the set of degree* 0 *divisors by* D^0.

The set of divisors forms an additive group with addition defined by $\sum_{P \in C} m_P P + \sum_{P \in C} n_P P = \sum_{P \in C} (m_P + n_P) P$ and the set D^0 is a subgroup of this group.

Definition 2.10 *(Menezes, Wu and Zuccherato [44, page 167])*
Let $R \in \bar{k}(C)$ *be a nonzero rational function. Then we can write* $R = G/H$ *for polynomial functions* G *and* H. *Then the* divisor of R *is given by* $div(R) = \sum_{P \in C} ord_P(R) P$, *where* $ord_P(R) = ord_P(G) - ord_P(H)$.

Thus every element in $\bar{k}(C)$ gives rise to a divisor of C. We gives these divisors a special name.

Definition 2.11 *(Menezes, Wu and Zuccherato [44, page 168])*
A divisor $D \in D^0$ *is called a* principal divisor *if* $D = div(R)$ *for some nonzero* $R \in K(C)$. *We denote the set of all principal divisors by* $\mathbb{P}$, *and this is a subgroup of* D^0.

We now have enough background to present the most important part of this section.

Definition 2.12 *(Menezes, Wu and Zuccherato [44, page 168])*
The quotient group $J = D^0/\mathbb{P}$ *is called the* Jacobian of C.

If $D_1, D_2 \in D^0$ are such that $D_1 - D_2 \in \mathbb{P}$ then we say D_1 and D_2 are *equivalent* divisors, written $D_1 \sim D_2$, and they appear in the same equivalence class in the Jacobian of C. In addition, the following definition provides us with a way of representing each equivalence class in the Jacobian.

Definition 2.13 *(Menezes, Wu and Zuccherato [44, pages 168,171])*
A degree 0 divisor $D = \sum_{P \neq \infty} m_P P - \left(\sum_{P \neq \infty} m_P\right) \infty$ *is* reduced *if* $\deg(P) \leq g$, $m_P \geq 0$ *for all* $P \neq \infty$, *and for* $m_P \geq 1$,

$$P \neq \tilde{P} \implies m_{\tilde{P}} = 0, \text{ and}$$

$$P = \tilde{P} \implies m_P = 0 \text{ or } m_P = 1.$$

The following theorem completes the puzzle for hyperelliptic curves.

Theorem 2.1 *(Jacobson, Menezes and Stein [36])*
Each equivalence class of the Jacobian of an imaginary hyperelliptic curve is represented by a unique reduced divisor.

Therefore the set of reduced divisors forms a group, where the group operation consists of computing the reduced sum of two divisors. Briefly, this is done as follows:

1. In order to add divisors $D_1 = \text{div}(a_1, b_1)$ and $D_2 = \text{div}(a_2, b_2)$, use the Euclidean algorithm to find polynomials d_1, e_1, e_2, d, c_1, $c_2 \in k[u]$ such that $\gcd(a_1, a_2) = d_1 = e_1 a_1 + e_2 a_2$ and $gcd(d_1, b_1 + b_2 + h) = d = c_1 d_1 + c_2(b_1 + b_2 + h)$. Let $s_1 = c_1 e_1$, $s_2 = c_1 e_2$ and $s_3 = c_2$, and set $a = a_1 a_2 / d^2$ and $b = \frac{s_1 a_1 b_2 + s_2 a_2 b_1 + s_3(b_1 b_2 + f)}{d} \pmod{a}$. Then the output from the addition algorithm is $D = \text{div}(a, b)$.

2. In order to reduce a divisor $D = \text{div}(a, b)$, set $a' = (f - bh - b^2)/a$ and $b' = (-h - b) \pmod{a'}$. If $\deg(a') > g$, set $a = a'$, $b = b'$ and repeat. Otherwise, make a' monic and output $D' = \text{div}(a', b')$.

For more details, the interested reader can consult works by Koblitz [43], Menezes, Wu and Zuccherato [44, Section 7 of the Appendix], Jacobson, Scheidler and Stein [38] and the Handbook of Elliptic and Hyperelliptic Curve Cryptography [15, pages 552–553].

The important thing is that algorithms exist for efficiently computing in the Jacobian. It is the group of reduced divisors that is used for hyperelliptic curve cryptography. Also, the points on an elliptic curve to which we can apply Weil descent are mapped to the group of reduced divisors. This group is isomorphic to the ideal class group of a quadratic function field, seen below.

2.2 Introduction to Quadratic Function Fields

Again we let $k = \mathbb{F}_q$ be a finite field with q elements and $\bar{k}$ be its algebraic closure. Then $k[u]$ is a polynomial ring and $k(u)$ is the smallest field containing $k[u]$, the field of fractions of $k[u]$. That is, $k(u) = \{\frac{G}{H} | G, H \in k[u], H \neq 0\}$. Consider the equation $v^2 + h(u)v = f(u)$ where $f, h \in k[u]$, f is monic with degree $2g+1$ or $2g+2$ and $\deg(h) \leq g$. Note that for ease of notation we frequently write f and h in place of $f(u)$ and $h(u)$. Also note that as above the polynomial $v^2 + hv - f$ is irreducible in $k[u, v]$. We now define a quadratic function field.

Definition 2.14 *(Paulus and Stein [58], Paulus and Rück [57], Jacobson, Scheidler and Stein [38])*
Let $K(C) = k(u)(v)$ where v is a root of $v^2 + h(u)v = f(u)$, $\deg(f(u)) = 2g+1$, and in the even characteristic case $\deg(h(u)) \leq g$. Then we call $K(C)$ an imaginary

quadratic function field *of C over k.*

Again, there are *real* quadratic function fields, but we will consider the imaginary case. Note that this definition is similar to that of an imaginary hyperelliptic curve.

If the characteristic of the field k is odd then it is possible, using a change of variables, to write the equation $v^2 + h(u)v = f(u)$ from the above definition as $v^2 = f(u)$, stated by Jacobson, Menezes and Stein [36]. Then $v = \sqrt{f(u)}$ and we see that $K(C)$ can be viewed as the result of adjoining v to the rational function field $k(u)$ in an analogous manner to the construction of a quadratic number field.

Definition 2.15 *(Paulus and Rück [57])*
Let $\mathcal{O}_K = k[u][v]$ be the integral closure of $k[u]$ in $K(C)$. That is, $\mathcal{O}_K$ contains all the integers in $K(C)$, and $k[u]$ is a subring of $\mathcal{O}_K$. We call $\mathcal{O}_K$ a(n) (imaginary) quadratic order.

The integral domain $\mathcal{O}_K$ is a Dedekind domain, stated by Paulus and Rück [57], so we consider its ideals.

Theorem 2.2 *(Paulus and Rück [57])*
Let $\mathfrak{a}$ be an integral ideal in $\mathcal{O}_K$. Then we can write $\mathfrak{a} = s(ak[u]+(b+v)k[u])$, where $s, a, b \in k[u]$ and $a \mid b^2 + bh - f$. Furthermore, if $\deg(b) < \deg(a)$ and the leading coefficients of a and s are 1 then (s, a, b) is the unique representation of the ideal $\mathfrak{a}$.

We also have a concept analogous to fractions in number fields.

Definition 2.16 *(Dummit and Foote [18, page 726])*
A fractional $\mathcal{O}_K$*-ideal is a subset* $\mathfrak{b}$ *of* $K(C)$ *for which there exists some* $d \in k[u]$ *such that* $d\mathfrak{b}$ *is an integral ideal.*

The operation on ideals that we are interested in is multiplication.

Definition 2.17 *(Dummit and Foote [18, page 248])*
Ideal multiplication is defined by $\mathfrak{ab} = \{\sum_{(a,b)\in U} ab \mid U \subset \mathfrak{a} \times \mathfrak{b},$ *a finite subset*$\}$, *where by* $\mathfrak{a} \times \mathfrak{b}$ *we mean all possible pairs of generators of* $\mathfrak{a}$ *and* $\mathfrak{b}$.

In practice this is done in a more efficient manner, using an algorithm such as that presented by Cantor [12]. Jacobson, Menezes and Stein [36] state that the multiplication algorithm is almost exactly the same as that used to add divisors in the hyperelliptic curve case, and thus we do not restate it here.

It is also natural to define a multiplicative inverse.

Definition 2.18 *(Dummit and Foote [18, page 726])*
A fractional ideal $\mathfrak{a}$ *is* invertible *if there exists some ideal* $\mathfrak{b}$ *such that* $\mathfrak{ab} = \mathcal{O}_K$. *We write* $\mathfrak{b} = \mathfrak{a}^{-1}$, *and call* $\mathfrak{b}$ *the* inverse *of* $\mathfrak{a}$.

We also have the concepts of conjugates and norms of ideals.

Definition 2.19 *(Jacobson and van der Poorten [39])*
The conjugate *of an* $\mathcal{O}_K$*-ideal* $\mathfrak{a} = s(ak[u] + (b + v)k[u])$ *is*

$$\bar{\mathfrak{a}} = s(ak[u] - (b + h - v)k[u]).$$

That is, it is the ideal that contains the conjugates of the elements in $\mathfrak{a}$.

Definition 2.20 *(Mollin [52, page 148])*

The norm *of an $\mathcal{O}_K$-ideal $\mathfrak{a}$ is $N(\mathfrak{a}) = |\mathcal{O}_K/\mathfrak{a}|$, the cardinality of $\mathcal{O}_K/\mathfrak{a}$.*

In our case, if our ideal is represented by $\mathfrak{a} = (s, a, b)$, we compute the norm $N(\mathfrak{a}) = as^2$ as Jacobson, Menezes and Stein [36] do. This norm is multiplicative: if $\mathfrak{a}$ and $\mathfrak{b}$ are two $\mathcal{O}_K$-ideals then $N(\mathfrak{ab}) = N(\mathfrak{a})N(\mathfrak{b})$.

There are several special types of ideals we are interested in.

Definition 2.21 *(Dummit and Foote [18, page 252])*

Let $\mathfrak{a}$ be a subset of $K(C)$ and let $(\mathfrak{a})$ be the smallest ideal containing $\mathfrak{a}$. Then $(\mathfrak{a})$ is the ideal generated by $\mathfrak{a}$. *An ideal is* principal *if it can be generated by a single element of $K(C)$.*

Definition 2.22 *(Enge and Stein [23])*

If we can write the ideal $\mathfrak{a} = s(ak[u] - (b + h - v)k[u])$, represented by (s, a, b), such that $s = 1$ then $\mathfrak{a}$ is called primitive. *A primitive ideal is also called* semireduced. *A primitive ideal is called* reduced *if* $\deg(a) < g$.

Cantor [12] also provides an algorithm for reducing ideals. Again we do not present it here due to its similarity to the hyperelliptic curve algorithm and simply assume that efficient implementations exist for us to use.

There are also ideals that are analogous to prime numbers in the integers.

Definition 2.23 *(Dummit and Foote [18, page 256])*

We call an $\mathcal{O}_K$-ideal $\mathfrak{p} \neq \mathcal{O}_K$ prime *if for any a, $b \in \mathcal{O}_K$, if $ab \in \mathfrak{p}$ then at least one*

of a and b is in $\mathfrak{p}$. Equivalently, $\mathfrak{p}$ is prime if $\mathfrak{ab} \subseteq \mathfrak{p}$ implies at least one of $\mathfrak{a} \subseteq \mathfrak{p}$ or $\mathfrak{b} \subseteq \mathfrak{p}$ holds, where $\mathfrak{a}$ and $\mathfrak{b}$ are $\mathcal{O}_K$-ideals.

Enge and Stein [23] state that the prime ideals $\mathfrak{q}$ of $k[u]$ are the ideals generated by irreducible polynomials q. We have the following cases for a prime ideal $\mathfrak{p}$ of $\mathcal{O}_K$ above $\mathfrak{q}$.

Definition 2.24 *(Enge and Gaudry [22])*

1. *If $x^2 + h(u)x - f(u) \equiv 0 \pmod{q}$ has two solutions then there are two prime ideals $\mathfrak{p}$ and $\bar{\mathfrak{p}}$ in $\mathcal{O}_K$ above $\mathfrak{q}$, and we call q, $\mathfrak{q}$, $\mathfrak{p}$ and $\bar{\mathfrak{p}}$* splitting.
2. *If $x^2 + h(u)x - f(u) \equiv 0 \pmod{q}$ has one solution then there is a single prime ideal $\mathfrak{p}$ in $\mathcal{O}_K$ above $\mathfrak{q}$ and we say q, $\mathfrak{q}$ and $\mathfrak{p}$ are* ramified.
3. *If $x^2 + h(u)x - f(u) \equiv 0 \pmod{q}$ has no solutions then there is a single prime ideal $\mathfrak{p}$ in $\mathcal{O}_K$ above $\mathfrak{q}$ and we say q, $\mathfrak{q}$ and $\mathfrak{p}$ are* inert.

The norm of a prime $\mathcal{O}_K$-ideal $\mathfrak{p}$ generated by p is $N(\mathfrak{p}) = p$.

Since $\mathcal{O}_K$ is a Dedekind domain, we also have the concept of unique factorization for ideals, presented by Enge and Stein [23]. Consider a primitive ideal $\mathfrak{a}$. Its norm is $N(\mathfrak{a}) = a \in k[u]$, and the norm can be written as a power-product of irreducible polynomials. Then since ideal norms are multiplicative, we can write $\mathfrak{a} = \prod_{i=1}^{k} \mathfrak{p}_i^{e_i}$ for prime ideals $\mathfrak{p}_i$ and exponents e_i if and only if $N(\mathfrak{a}) = \prod_{i=1}^{k} N(\mathfrak{p}_i)^{|e_i|}$. This is an important result that is further expanded in Chapter 3, as the algorithms for solving the discrete logarithm problem presented there rely on being able to factor ideals.

Now we define the ideal class group.

Definition 2.25 *(Jacobson, Menezes and Stein [36])*
Let $\mathcal{I}$ denote the set of fractional ideals of $\mathcal{O}_K$, and $\mathcal{P}$ denote the subgroup of principal ideals. Then the ideal class group *is defined to be the quotient group $Cl(K) = \mathcal{I}/\mathcal{P}$ and the size of the class group, $h = |Cl(K)|$, is called the* class number.

We write elements of $Cl(K)$ as $[\mathfrak{a}]$. We now consider the elements of the classes in the class group.

Definition 2.26 *(Jacobson [33])*
Two ideals $\mathfrak{a}$ and $\mathfrak{b}$ are called equivalent *if there exists principal ideals (α) and (β) for $\alpha, \beta \in \mathcal{O}_K$ such that $(\alpha)\mathfrak{a} = (\beta)\mathfrak{b}$. This is written as $\mathfrak{a} \sim \mathfrak{b}$, and $\mathfrak{a}$ and $\mathfrak{b}$ are in the same class in $Cl(K)$.*

We saw before that ideals can be reduced. More importantly, each $\mathcal{O}_K$-ideal is equivalent to a unique reduced ideal, a consequence of the following theorem.

Theorem 2.3 *(Jacobson, Menezes and Stein [36])*
Each ideal class $[\mathfrak{a}]$ in $Cl(K)$ contains a unique reduced ideal.

This means each ideal class can be represented by a unique reduced ideal. We now describe the group operation for the elements in $Cl(K)$. For any equivalence classes $[\mathfrak{a}]$ and $[\mathfrak{b}]$ in $Cl(K)$ with reduced ideal representatives $\mathfrak{a}$ and $\mathfrak{b}$, first compute $\mathfrak{a}\mathfrak{b} = \mathfrak{c}$. Then reduce $\mathfrak{c}$ to get $\mathfrak{c}'$, the reduced ideal that represents the ideal class $[\mathfrak{c}'] = [\mathfrak{a}][\mathfrak{b}]$. In practice, this is done in a single step using a variation of the NUCOMP algorithm, such as that presented by Jacobson and van der Poorten [39].

Finally, we have the following theorem due to Paulus and Rück [57] that connects

the ideal class group of an imaginary quadratic function field to the Jacobian of the imaginary hyperelliptic curve defined by the same parameters $f(u)$ and $h(u)$ over k.

Theorem 2.4 *(Paulus and Rück [57])*
There is a canonical bijection between the reduced divisors of a hyperelliptic curve seen above and the set of reduced ideals in $\mathcal{O}_K$.

Similar theorems are stated by Paulus and Stein [58] and Jacobson, Menezes and Stein [36] that imply that this is an isomorphic map between the two groups. This tells us that the ideal class group is the same size as the Jacobian of the hyperelliptic curve.

Now that we have seen how we can relate hyperelliptic curves to quadratic function fields, we investigate the hyperelliptic discrete logarithm problem, and describe its interpretation in the quadratic function field context.

2.3 The Discrete Logarithm Problem

As mentioned before, the discrete logarithm problem plays a major role in public key cryptography. Several systems, such as the Diffie-Hellman key agreement protocol, make use of it. In the elliptic curve setting, the discrete logarithm problem is: Given points P, $Q = xP$ on an elliptic curve E, find the least positive integer x such that $Q = xP$.

With this knowledge, we now describe the Elliptic Curve Diffie-Hellman protocol, outlined by Certicom Research [13]. Assume the two parties involved have agreed

on the parameters, such as the curve C and a base point or generator G with order $|G| = n$, that are being used in this communication, and each party has a private, public key pair (d_i, Q_i), where d_i, a random integer between 1 and $n-1$, is the private key and $Q_i = d_i G$ is the public key. Then the two parties exchange public keys. The first participant computes $K = d_1 Q_2 = d_1 d_2 G$ and the second participant computes $K = d_2 Q_1 = d_2 d_1 G$, and thus K is used as the key for further communication. Due to the difficulty of the ECDLP, an eavesdropper can know Q_1, Q_2 and G, but cannot compute d_1, d_2 or K.

The hyperelliptic curve discrete logarithm problem (HCDLP) is as follows: given reduced divisors D_1, $D_2 \sim xD_1$ in the Jacobian of the curve C, find the least positive integer x that satisfies that relation. There exists a method that takes certain instances of the ECDLP problem to instances of the HCDLP called *Weil descent*. Basically, if E is an elliptic curve defined over $K = \mathbb{F}_{2^{nm}}$ with dr points, for small d and large prime r, Weil descent can, in some cases, be used to reduce the ECDLP problem in the subgroup of points on the elliptic curve with order r to an instance of the HCDLP in the subgroup of reduced divisors that has order r, where the hyperelliptic curve is defined over $\mathbb{F}_{2^n}$ and has genus $g \approx m$. If $m > \log 2^n$ then the resulting HCDLP instance can be solved in subexponential time. For more details, the interested reader can consult works by Frey [25], Gaudry, Hess and Smart [27], and Jacobson, Menezes and Stein [35].

An equivalent statement of the HCDLP using the language of quadratic function fields is: given reduced ideals $\mathfrak{a}$, $\mathfrak{b} \sim \mathfrak{a}^x$, find the least positive integer x satisfying

$\mathfrak{a}^x \sim \mathfrak{b}$. We will use this formulation for the remainder of the thesis.

In this chapter we introduced hyperelliptic curves and quadratic function fields. More importantly, we have seen how hyperelliptic curves and quadratic function fields are related. We have also seen the discrete logarithm problem in both settings, and described how they are related. In the remainder of the thesis we study algorithms for solving the discrete logarithm problem in the ideal class group of a quadratic function field. In doing this we see algorithms that can solve the HCDLP. We see that there are instances of the ECDLP that we can thus solve in subexponential time. Finally, we present results that demonstrate the success of these algorithms in practice.

Chapter 3

Index Calculus Algorithms For Solving Discrete Logarithm Problems

There have been several algorithms proposed for solving discrete logarithm problems in quadratic number fields and quadratic function fields. In general, the discrete logarithm problem can be described as follows. Let G be a cyclic group for which the group operation is written multiplicatively. Then given a generator $a \in G$, $b = a^x$, find the least (positive) integer x such that a^x is indeed b. The most obvious method is to simply perform a "brute force" computation by finding a, a^2, a^3, ..., a^k, until $a^k = b$. Clearly this is not an efficient process if x is large, and it could result in traversing through the entire group G.

Pollard [59] introduced his "Rho" method (called such because the search space can be visualized as a shape similar to the Greek letter ρ) in 1978. Here, two sequences of exponentiations are computed and when the values with the same index in the sequences are the same, further computations can be made to compute the discrete logarithm. This method has a runtime of $O(\sqrt{|G|})$ and is applicable in any cyclic group G including the group of points on an elliptic curve and the divisor class group of a hyperelliptic curve.

In 1994, Adleman, DeMarrais and Huang [7] introduced a subexponential algorithm

for computing discrete logarithms in Jacobians of large genus hyperelliptic curves. They do this using a procedure called *index calculus*. This algorithm is similar to the ideas presented by Hafner and McCurley [29], where they use index calculus to compute class groups of imaginary quadratic number fields and by McCurley [50], where index calculus is used to compute class numbers of imaginary quadratic number fields and perform discrete logarithm computations in the class group. Using index calculus for discrete logarithm computations is also investigated by Gaudry [26] and Enge [21]. These ideas are further built upon and improved by Enge and Gaudry [22], where, given $|G|$, a general framework is presented for solving discrete logarithm problems in subexponential time under certain assumptions on the smoothness of elements in the group, where the condition varies based on the group being considered. Vollmer [66] develops an alternate index calculus algorithm that runs in subexponential time in quadratic number fields and can be easily adapted to work in quadratic function fields.

The focus of this thesis is improving index calculus algorithms for solving the discrete logarithm problem in imaginary quadratic function fields. In this chapter we will give a high level description of the index calculus method, splitting it into three main parts: creating a factor base, finding relations and solving a linear algebra problem. We then describe two algorithms that use index calculus in order to solve discrete logarithm problems. These algorithms perform the first step the same way and algorithms for solving linear algebra problems have been studied at length, so we have focused on procedures for generating relations. As such, we conclude this chapter by presenting several different algorithms that have been proposed for gen-

erating relations. Finally, we discuss the linear algebra phase of the index calculus algorithms we present.

3.1 Index Calculus and the Discrete Logarithm Problem

Index calculus algorithms have three main steps. The first is to compute a finite set $B = \{\mathfrak{p}_1, \mathfrak{p}_2, \mathfrak{p}_3, \ldots, \mathfrak{p}_k\}$, called the *factor base*, consisting of prime ideals whose norms have degree less than a given bound. In the next chapter we describe how to pick this bound. Until then, we call it t. We assume without verification that the value t is chosen such that the factor base contains enough elements to generate the entire class group. These prime ideals are found by enumerating the monic polynomials with degree less than or equal to t, the *degree bound* for the factor base, and checking for irreducibility. If a polynomial p is irreducible then one attempts to find a solution to $x^2 + hx - f \equiv 0 \pmod{p}$. If a solution x exists then $\mathfrak{p} = (p, x)$ is a prime ideal of $\mathcal{O}_K$ and we add it to the factor base. We assume that the procedure used to solve the quadratic equivalence above always returns the same solution x when given input p, a condition that can be enforced easily in practice.

The key to the efficiency of index calculus algorithms is being able to rapidly identify "smooth" elements in the group. In our case we say that an $\mathcal{O}_K$-ideal y is *smooth over the factor base*, or simply *smooth* if y is equivalent to a power-product of the elements in the factor base. We say that an element is *t-smooth* if it is smooth with respect to the bound t. A *relation* is a vector $\vec{v_e} = (e_1, e_2, e_3, \ldots, e_k)$ representing the factorization of the element y over the factor base. That is, $\prod_{i=1}^{k} \mathfrak{p}_i^{e_i} \sim y$.

The second step is generating A, a $(k \times l)$ matrix whose columns are relations, with $l > k$, called the *relation matrix.* It should be noted that this second step is, in general, easily adaptable to be done in parallel on many machines. Exactly how this step is performed is discussed later in this section, and the method for picking a value for l is described in the next chapter.

Finally, the third major step in an index calculus algorithm is performing a linear algebra computation. This is usually solving a linear system. The result of the linear algebra computation is used in a final computation to compute the solution to the problem at hand, in our case, a discrete logarithm. Again, what linear algebra problems are solved and how it is done is be discussed later in this chapter.

We now discuss the general algorithm for subexponential discrete logarithm computations outlined and analyzed by Enge and Gaudry [22], presented in the setting of quadratic function fields, and the method described by Vollmer [66] for quadratic number fields, which we generalize to work in quadratic function fields. While both algorithms make use of index calculus, and both are probabilistic algorithms of the Las Vegas variety, the way in which use index calculus is different.

For the remainder of the chapter we consider ideals contained in the maximal order $\mathcal{O}_K$ of the imaginary quadratic function field K.

3.1.1 The Enge-Gaudry Algorithm

The general algorithm presented by Enge and Gaudry [22] evolved from the algorithm presented by Adleman, DeMarrais and Huang [7], the work done by Enge [21] and more directly the work done by Gaudry [26]. While the original algorithm by Adleman, DeMarrais and Huang first requires computation of the group structure (also done with an index calculus algorithm), the algorithm we discuss here, published by Enge and Gaudry, requires only the order of the group, N, and assumes that this value is precomputed and provided as input.

The Enge-Gaudry algorithm [22], given reduced ideals $\mathfrak{a}$, $\mathfrak{b} \sim \mathfrak{a}^x$ in the order $\mathcal{O}_K$, class number N, and factor base bound t, works as follows:

1. Construct the factor base $B = \{\mathfrak{p}_1, \mathfrak{p}_2, \mathfrak{p}_3, \ldots, \mathfrak{p}_k\}$, consisting of prime ideals such that $\deg(N(\mathfrak{p}_i)) \leq t$. This is done in the manner described above.

2. Generate the relation matrix A, consisting of $l > k$ relations. The j^{th} relation is found by randomly generating non-zero α_j and β_j in $\mathbb{Z}_N$ until $\mathfrak{a}^{\alpha_j}\mathfrak{b}^{\beta_j}$ is smooth over the factor base. This gives us a relation of the form

$$\mathfrak{a}^{\alpha_j}\mathfrak{b}^{\beta_j} \sim \prod_{i=1}^{k} \mathfrak{p}_i{}^{e_{i,j}}.$$

   Exactly how we find relations is described later. Store the $e_{i,j}$ values in the relation matrix and α_j and β_j in two additional vectors.

3. Find a non-zero vector $\vec{v} = (v_1, v_2, v_3, \ldots, v_l)$ such that $A\vec{v} = \vec{0} \pmod{N}$. That is, $\vec{v} \in \ker A$. This can be done, for example, using a variation of Wiedemann's algorithm for solving sparse linear equations [69], or Lanczos'

algorithm [47]. The methods we use in practice are discussed in Chapter 5. If such a vector cannot be found, find a small number of additional relations and repeat this step.

4. Compute $\sum_{j=1}^{l} \beta_j v_j \pmod{N}$. If this is invertible modulo N then output the discrete logarithm

$$x = -\left(\sum_{j=1}^{l} \beta_j v_j\right)^{-1} \left(\sum_{j=1}^{l} \alpha_j v_j\right) \pmod{N}.$$

Otherwise, return to step 2.

The output x is indeed the discrete logarithm. Since $A\vec{v} = \vec{0}$, we have

$$\sum_{j=1}^{l} e_{i,j} v_j = 0$$

for all $i = 1, 2, \ldots, k$. Then we have

$$\begin{aligned}
\mathcal{O}_K &\sim \prod_{i=1}^{k} \left(\mathfrak{p}_i^{\sum_{j=1}^{l} e_{i,j} v_j}\right) \\
&\sim \prod_{j=1}^{l} \left(\prod_{i=1}^{k} \mathfrak{p}_i^{e_{i,j}}\right)^{v_j} \\
&\sim \prod_{j=1}^{l} \left(\mathfrak{a}^{\alpha_j} \mathfrak{b}^{\beta_j}\right)^{v_j} \\
&\sim \mathfrak{a}^{\left(\sum_{j=1}^{l} \alpha_j v_j\right)} \mathfrak{b}^{\left(\sum_{j=1}^{l} \beta_j v_j\right)}.
\end{aligned}$$

Thus, $\mathfrak{a}^{-\left(\sum_{j=1}^{l} \alpha_j v_j\right)} \sim \mathfrak{b}^{\left(\sum_{j=1}^{l} \beta_j v_j\right)}$, so the result holds.

Enge and Gaudry [22] provide an analysis of their algorithm. They show, assuming that the relations produced are random and using the work done by Enge and Stein

[23] on the number of t-smooth reduced ideals of a fixed degree, that the expected runtime of this algorithm in imaginary quadratic function fields is

$$O\left(L\left(\sqrt{2}\left(\sqrt{\frac{1}{2\theta}}+\sqrt{1+\frac{1}{2\theta}}\right)+o(1)\right)\right)$$

where $L(c) = e^{c(g\log q)^{\alpha}(\log(g\log q))^{1-\alpha}}$ is the standard subexponential function with $\alpha = \frac{1}{2}$ and $\theta > 0$ such that $g \geq \theta \log q$. Thus, this algorithm is subexponential when g is sufficiently large compared to $\log q$.

3.1.2 Vollmer's Method

The above algorithm relies on knowledge of the order of the group in order to work. Vollmer [66] developed a subexponential algorithm that does not require this. In quadratic fields this is an important development as the order of the ideal class group is typically not known, and is not easy to compute. However, we assume that we do have the class number, and present a version of the algorithm that makes use of it. The other major interest for studying this algorithm is that it allows us to use a different method for generating relations, called sieving. As we discuss later, sieving does not work well in the Enge-Gaudry algorithm, but it fits quite well in Vollmer's algorithm. It should be noted that Vollmer presented this algorithm for discrete logarithm computations in quadratic number fields but it translates directly over to the quadratic function field setting, as presented here.

Vollmer's algorithm [66], given reduced ideals $\mathfrak{a}$ and $\mathfrak{b} \sim \mathfrak{a}^x$ in the order $\mathcal{O}_K$, the class number N, and the factor base bound t, operates as follows:

1. Generate a factor base $B = \{\mathfrak{p}_1, \mathfrak{p}_2, \mathfrak{p}_3, \ldots, \mathfrak{p}_k\}$, again consisting of prime

ideals with $\deg(N(\mathfrak{p}_k)) \leq t$, as described above.

2. Randomly generate $l > k$ relations for the relation matrix A. Again, how we pick l is discussed in the next chapter. Compute relations of the form

$$\prod_{i=1}^{k} \mathfrak{p}_i^{e_{i,j}} \sim \mathcal{O}_K.$$

Store the $e_{i,j}$ values in the relation matrix.

3. Find vectors $\vec{v_{a^{-1}}} = (v_{a,1}, v_{a,2}, \ldots, v_{a,k})$ and $\vec{v_b} = (v_{b,1}, v_{b,2}, \ldots, v_{b,k})$, such that $\mathfrak{a}^{-1} \sim \prod_{i=1}^{k} \mathfrak{p}_i^{v_{a,i}}$ and $\mathfrak{b} \sim \prod_{i=1}^{k} \mathfrak{p}_i^{v_{b,i}}$. That is, find relations of the form

$$\prod_{i=1}^{k} \mathfrak{p}_i^{v_{y,i}} \sim y$$

for both $y = \mathfrak{a}^{-1}$ and $y = \mathfrak{b}$. Note that we have $\mathfrak{a}^{-1} \prod_{i=1}^{k} \mathfrak{p}_i^{-v_{a,i}} \sim \mathcal{O}_K$ and $\mathfrak{b} \prod_{i=1}^{k} \mathfrak{p}_i^{-v_{b,i}} \sim \mathcal{O}_K$, so $(1, 0, -v_{a^{-1}})$ and $(0, 1, -v_b)$ are relations over the extended factor base $\{\mathfrak{a}^{-1}, \mathfrak{b}, \mathfrak{p}_1, \mathfrak{p}_2, \ldots, \mathfrak{p}_k\}$.

4. Set $\bar{A} = \begin{pmatrix} 1 & 0 & \vec{0} \\ 0 & 1 & \vec{0} \\ -\vec{v_{a^{-1}}}^T & -\vec{v_b}^T & A \end{pmatrix} = \begin{pmatrix} \vec{a} \\ A' \end{pmatrix}$ and solve the linear system $A'\vec{v} = (1, 0, 0, 0, \ldots, 0)^T$ over the integers modulo N. We can, for example, use a variant of Wiedemann's algorithm for sparse, singular matrices, described by Wiedemann [69] and Mulders [54], or Lanczos' algorithm [47]. The method used in practice is discussed in Chapter 5. If a solution does not exist, generate a small number of additional relations and return to step 3.

5. Set $x \equiv \vec{v} \cdot \vec{a} = v_1 \mod N$. Then x is returned as the discrete logarithm. Note that this value is not necessarily minimal.

Again, x 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 $\bar{A}\vec{v} \equiv (z, 1, 0, \ldots, 0)^T \pmod{N}$, and thus $(z, 1, 0,$

$\ldots, 0)$ is a linear combination of the columns of $\bar{A}$. This implies that $(\mathfrak{a}^{-1})^z\,\mathfrak{b} \sim \mathcal{O}_K$, or is principal, since the columns of $\bar{A}$ are relations over the extended factor base $\{\mathfrak{a}^{-1}, \mathfrak{b}, \mathfrak{p}_1, \mathfrak{p}_2, \ldots, \mathfrak{p}_k\}$. Finally, since $(\mathfrak{a}^{-1})^z\,\mathfrak{b} \sim \mathcal{O}_K$, we know that $\mathfrak{b} \sim \mathfrak{a}^z$ and $z \equiv \vec{v} \cdot \vec{a} \pmod{N}$, giving us the correct result.

Since we are computing relations randomly, there are circumstances in which this could fail. Consider the homomorphism $\phi : \mathbb{Z}^k \rightarrow Cl(K)$ that takes vectors $\vec{v}$ in $\mathbb{Z}^k$ to the power-product over the factor base $\prod_{i=1}^{k} \mathfrak{p}_i^{v_i}$. Since we are using relations that are equivalent to $\mathcal{O}_K$, the set $\{\vec{v} \in \mathbb{Z}^k \mid \prod_{i=1}^{k} \mathfrak{p}_i^{v_i} \sim \mathcal{O}_K\}$ forms the kernel of ϕ, and we get $\mathbb{Z}^k / \ker\phi \cong Cl(K)$ from the First Isomorphism Theorem, provided that ϕ is surjective, which will be the case if the factor base generates the entire class group. If the lattice formed by the columns of A is not equal to $\ker\phi$ then it is possible that we cannot find a solution vector $\vec{x}$ even if one does exist. In order to avoid this situation we can test to ensure multiples of existing relations are not added, but testing for linear combinations is more difficult. One could also, for example, 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 will slow down computations considerably.

Vollmer [66] analyses his algorithm in the quadratic number field setting. His result gives a subexponential algorithm with an expected running time of $O\left(L_d\left(\frac{3\sqrt{2}}{4}\right)\right)$, where $L_d(c) = e^{c\sqrt{\log d \log\log d}}$, and $-d$ is a discriminant of an imaginary quadratic number field. In the quadratic function field setting, assuming we have the group order, one could use the same algorithms to create the factor base and solve the

linear algebra as used in the Enge-Gaudry algorithm. We assume that we have to generate the same number of relations whether Enge-Gaudry or Vollmer's method is used, as discussed in the next chapter. Also, the distribution of smooth reduced ideals is the same if the same factor base is used for both algorithms. In order to mirror the analysis done by Enge and Gaudry [22] we require the amount of time expected to compute an ideal for testing. We expect an asymptotic result similar to that of Enge and Gaudry, but the work is left for future consideration.

There are several different methods of performing the relation generation stage of the above two algorithms. In the next section we discuss some of the relation generation methods that have been proposed along with some improvements that have been made. Following that we briefly discuss solving the linear algebra problems we have seen.

3.2 Relation Generation

The two most computationally intensive steps in an index calculus algorithm are the relation generation phase and the linear algebra phase. As such, an improvement to either one significantly improves the performance of the algorithm. Much work has gone into trying to improve the relation generation phase of these types of algorithms and this section covers some of this work. In addition, since these algorithms involve finding ideals that are smooth over the factor base, we discuss a method for efficiently checking this condition.

Finding relations can be done efficiently in a random manner. Enge and Stein [23] compute a lower bound on the number of smooth ideals in characteristic two function fields. Theorem 6 of [23] states that if $t = \lceil \log_q L(\rho) \rceil$ for a constant $\rho > 0$ then the number of t-smooth reduced ideals of degree g is greater than or equal to $L\left(-\frac{1}{2\rho}\right) q^g$. What ρ should be becomes evident when a complete analysis is done, as that by Enge and Gaudry [22]. There are fewer than q^g reduced ideals in $\mathcal{O}_K$. Then the probability of a given reduced ideal being smooth is at least $\frac{L\left(-\frac{1}{2\rho}\right)q^g}{q^g} = L\left(-\frac{1}{2\rho}\right)$. This is what gives us a subexponential run time for the algorithms listed in the previous section. Since this is the most computationally intensive part of index calculus algorithms, this also explains why the majority of the work done to improve the relation generation stage of index calculus algorithms has been in speeding up the creation of ideals and the testing of them for smoothness.

3.2.1 Factoring and Smoothness Testing

In order to find relations we must be able to efficiently determine whether various ideals are smooth with respect to the factor base. We first describe how we can factor a smooth ideal over the factor base. Consider an ideal $\mathfrak{a}$ with $N(\mathfrak{a}) = a$ in the order $\mathcal{O}_K$ where K is an imaginary quadratic function field with characteristic two. Suppose $\mathfrak{a} = (a, b)$ is smooth over the factor base and for each $\mathfrak{p}_i$ in the factor base we have $\mathfrak{p}_i = (p_i, b_{p_i})$. Then, similar to Theorem 2.4 presented by Jacobson [33], there is a vector $\vec{e}$ such that $a = \prod_{i=1}^{k} {p_i}^{e_i}$ and $\mathfrak{a} \sim \prod_{i=1}^{k} \mathfrak{p}_i^{s_i e_i}$ where $s_i = 1$ if $b_{p_i} \equiv b \pmod{N(\mathfrak{p}_i)}$ and $s_i = -1$ if $b_{p_i} \equiv b + h \pmod{N(\mathfrak{p}_i)}$. Thus, in order to factor an ideal over the factor base, we only need to factor its norm over the norms of the elements of the factor base, and determine the signs of the exponents using

the above congruence.

We now discuss testing the smoothness of polynomials, since $N(\mathfrak{a}) \in \mathbb{F}_q[X]$. As argued above, with this information we can obtain a factorization of $\mathfrak{a}$. One way to determine smoothness is to trial divide the polynomial we would like to test. However this is not very efficient, especially as the size of the factor base grows. Gerhard and von zur Gathen [68, Chapter 14] provide a generalization of Fermat's little theorem that states that for $d \geq 1$, $x^{q^d} - x \in \mathbb{F}_q[X]$ is the product of all monic irreducible polynomials in $\mathbb{F}_q[X]$ with degree dividing d. Since our factor base contains all monic irreducible ideals with norms up to a given degree t, this theorem gives rise to a smoothness test.

A naïve way to implement a smoothness test using the above theorem would be as follows. We are testing $a \in \mathbb{F}_q[X]$ for smoothness. First set $\gamma = 1$. Then for $i = 1, 2, \ldots, t$, compute iteratively $\gamma \equiv \gamma(x^{q^i} - x) \pmod{a}$, where t is the above mentioned bound on the norms of the factor base elements. If $\gamma = 0$, a divides evenly into the product $(x^{q^1} - x)(x^{q^2} - x) \cdots (x^{q^t} - x)$, and thus a must be smooth over the factor base.

However, one can be slightly more clever, as Jacobson, Menezes and Stein [36] were. Remember that the theorem tells us that $x^{q^d} - x$ is the product of all monic irreducibles of degree *dividing* d. Then starting with $i = \lfloor \frac{t}{2} \rfloor + 1$ and going up to t is sufficient. All values between 2 and $\lfloor \frac{t}{2} \rfloor$ divide at least one of the values $\lfloor \frac{t}{2} \rfloor + 1$, $\lfloor \frac{t}{2} \rfloor + 2, \cdots, t$, and so are represented in the product.

The remaining complication is computing $x^{q^i} - x \pmod{a}$ for the various i values. This is done using the *Frobenius map*. As Gerhard and von zur Gathen [68, Chapter 14] do, let $f \in \mathbb{F}_q[X]$ be monic and squarefree, and consider $R = \mathbb{F}_q[X]/\langle f \rangle$. Then the Frobenius map is $\sigma : R \to R$ such that $\sigma(\alpha) = \alpha^q$. Then the following properties hold for $\alpha, \beta \in R$.

$$\sigma(\alpha + \beta) = \sigma(\alpha) + \sigma(\beta),\ \sigma(\alpha\beta) = \sigma(\alpha)\sigma(\beta),\ \sigma(\alpha) = \alpha \iff \alpha \in \mathbb{F}_q$$

These properties imply that $g(\alpha^q) = g(\alpha)^q$ for all polynomials $g \in \mathbb{F}_q[X]$ and all $\alpha \in R$. We use these properties to compute $x^{q^i} \pmod{a}$ as follows. First compute $x^q,\ x^{2q},\ \cdots,\ x^{\deg(a)q}$ all modulo a. This is done using repeated squaring to find x^q, and then computing each power of that. Then in order to find $x^{q^{i+1}} \pmod{a}$, evaluate $x^{q^i} \pmod{a}$ at x^q using the precomputed values. Note that $x^{q^i} \pmod{a}$ is not necessarily as trivial as it looks, due to the reduction modulo a. However, using the properties of the Frobenius map makes this computation simple.

Smoothness testing and factoring ideals is essential for the relation generation stage of index calculus algorithms. We now describe several algorithms for generating relations. One of the first methods used was computing random exponent vectors. This method can easily be used with both the Enge-Gaudry algorithm and with Vollmer's method. We also discuss the random walk method, which works well when used in the Enge-Gaudry algorithm, and sieving, which is better suited for Vollmer's algorithm.

3.2.2 Random Exponents

The random exponents method of finding relations was one of the first proposed. It was outlined by Hafner and McCurley [29] in a very basic way. We present it here to motivate the improved methods. We tailor this description to the quadratic function field setting. One simply generates a random vector $\vec{w} = (w_1, w_2, w_3, \ldots, w_k)$, with w_i in a given interval. Then one computes $\prod_{i=1}^{k} \mathfrak{p}_i{}^{w_i} = \mathfrak{W}$ where the $\mathfrak{p}_i$ are once again elements of the factor base. The resulting ideal $\mathfrak{W}$ is reduced to $\mathfrak{W}'$ and the result is checked for smoothness over the factor base. If the reduced ideal is indeed smooth then it is factored over the factor base as discussed above, giving the vector w' such that $\mathfrak{W}' = \prod_{i=1}^{k} \mathfrak{p}_i{}^{w'_i}$. Then there are two sets of exponents that produce equivalent ideals. Thus, $\prod_{i=1}^{k} \mathfrak{p}_i^{w_i - w'_i} \sim \mathcal{O}_K$, and $\vec{w} - \vec{w'}$ is a relation.

Düllmann [17] suggests some improvements to the algorithm. These are outlined in English by Jacobson [33]. The main contribution by Düllmann is an adjustment in how the non-zero exponents are chosen, resulting in a relation matrix that has a small, dense part, and is mostly sparse. There are linear algebra algorithms that can take advantage of these properties. Additionally, sparse matrices can require less storage, depending on the implementation. The other contribution of Düllmann is suggesting precomputations that result in faster computation of ideals at the cost of increased storage.

It is clear that this algorithm can be applied directly to computing relations for Vollmer's method. In order to use it with the Enge-Gaudry algorithm, instead of

computing $\mathfrak{W}$ as above, we compute $\mathfrak{a}^{-\alpha}\mathfrak{b}^{-\beta}\mathfrak{W}$. We then reduce this to get $\mathfrak{W}'$. When $\mathfrak{W}'$ is smooth we have $\mathfrak{a}^{-\alpha}\mathfrak{b}^{-\beta}\prod_{i=1}^{k}\mathfrak{p}_i^{w_i-w'_i} \sim (1)$. Thus, $\mathfrak{a}^{\alpha}\mathfrak{b}^{\beta} \sim \prod_{i=1}^{k}\mathfrak{p}_i^{w_i-w'_i}$, as required. Now we can use random exponents to generate relations suitable for use in the Enge-Gaudry algorithm.

3.2.3 Random Walks

Teske [64] suggests a method for improving Pollard's Rho method that makes use of a random walk through a group in order to compute discrete logarithms. The phrase *random walk* refers to the random way in which we traverse through the group, from one element to the next. Teske compares Pollard's random walk to various alternates, and finds walks that perform better in practice. Teske's random walk can be applied to finding relations and is used in conjunction with the Enge-Gaudry algorithm by Gaudry [26] and Jacobson, Menezes and Stein [36].

To find relations using this procedure, start by randomly choosing integers a_0, a_1, ..., a_j, b_0, b_1, ..., b_j such that $0 \leq a_i, b_i \leq N-1$ for $i = 0, \ldots, j$, where N again is the class number. Then compute and store reduced ideals $\mathfrak{T}_i \sim \mathfrak{a}^{a_i}\mathfrak{b}^{b_i}$, along with a_i and b_i. We start the random walk with $\mathfrak{R}_0 \sim \mathfrak{a}^{\alpha_0}\mathfrak{b}^{\beta_0}$, where α_0 and β_0 are selected randomly as before. To find the next ideal compute $\mathfrak{R}_{i+1} \sim \mathfrak{R}_i\mathfrak{T}_l$, where $l = H(\mathfrak{R}_i)$, for some hash function $H : \mathfrak{O} \to \{0, 1, \ldots, j\}$ where $\mathfrak{O}$ represents the set of reduced $\mathcal{O}_K$-ideals. Then $\mathfrak{R}_{i+1} \sim \mathfrak{a}^{\alpha_{i+1}}\mathfrak{b}^{\beta_{i+1}}$ where $\alpha_{i+1} \equiv \alpha_i + a_l \pmod{N}$ and $\beta_{i+1} \equiv \beta_i + b_l \pmod{N}$. If an ideal $\mathfrak{R}_i$ is smooth then we store α_i, β_i and the vector $\vec{w}$ such that $\mathfrak{R}_i \sim \mathfrak{a}^{\alpha_i}\mathfrak{b}^{\beta_i} \sim \prod_{i=1}^{k}\mathfrak{p}_i^{w_i}$. This method results in only one ideal multiplication for each ideal to be tested.

Gaudry [26] performs an asymptotic analysis of the number of operations needed in order to run the Enge-Gaudry algorithm using this as the relation generation method. Here he assumes that the smoothness bound is fixed at 1, so the factor base contains all the prime ideals with norms that have degree equal to one. The result is an estimated runtime of $O(q^2 + g!q)$. Gaudry also points out that if instead you require $g > \log q$ and allow t to vary you get the result presented by Enge and Gaudry [22] and above in the Enge-Gaudry section.

3.2.4 Large Primes

Thériault [65] analyzes the use of large primes, an idea originally proposed for use in factoring algorithms, as used by Lenstra and Manasse [48], Kurowski [45] and Boender and te Riele [10]. This idea was adapted by Jacobson [33] for generating relations to find the class number in the quadratic number field setting. This idea results in a decrease in the number of operations needed in the Enge-Gaudry algorithm. Note that Thériault's analysis is for the case when the genus is small relative to q. For the first variation of random walks that we present, Thériault [65] requires $q > (g-1)!$. For the second variation, he requires $q > (g-1)!/g$ to be satisfied to get the asymptotic results he presents. In these cases, there are a large number of degree one primes. As such, like Gaudry [26], the degree bound for the factor base is $t = 1$. The ideas Thériault studies lead to a reduction in the size of the factor base. This has the added effect of improving the linear algebra stage, as a smaller factor base means a smaller relation matrix.

Thériault [65] also analyzes the use of only a fraction of the degree one prime ideals in the factor base B. He introduces a parameter, r, such that $2/3 < r < 1$. Then prime ideals are added to the factor base until $|B| = q^r$. In the remainder of this section we use the term smooth to mean *smooth over* B. As in the above reference, we use $\mathcal{P}$ to denote all prime ideals that have norms with degree equal to one. A *potentially smooth* ideal is one that is smooth over $\mathcal{P}$ (but not necessarily over B). An ideal is *almost smooth* if all of its factors are in B with the exception of one, which is in $\mathcal{P} \setminus B$. That is, an ideal $\mathfrak{y}$ is almost smooth if $\mathfrak{y} \sim \mathfrak{p} \prod_{i=1}^{|B|} \mathfrak{p}_i^{e_i}$, where the ideals $\mathfrak{p}_i$ are again the prime ideals in B, and $\mathfrak{p} \in \mathcal{P} \setminus B$. Such a prime ideal $\mathfrak{p}$ is called a *large prime*. A *partial relation* is a vector $\vec{v_e} = (e_1, e_2, e_3, \ldots, e_k)$ representing the factorization of the smooth part of the element y over the factor base. That is, $\mathfrak{p} \prod_{i=1}^{k} {\mathfrak{p}_i}^{e_i} \sim y$. When clarity is important, we will call our original relations *full relations* to further differentiate between the two.

If two almost smooth ideals share the same large prime, or one has the prime $\mathfrak{p}$ and the other has the inverse, then we have what Thériault [65] calls an *intersection*, and these two ideals can be combined to form a relation. Let $\mathfrak{y}_1 \sim \mathfrak{p} \prod_{i=1}^{|B|} {\mathfrak{p}_i}^{e_{1,i}} \sim \mathfrak{a}^{\alpha_1} \mathfrak{b}^{\beta_1}$ and $\mathfrak{y}_2 \sim \mathfrak{p} \prod_{i=1}^{|B|} {\mathfrak{p}_i}^{e_{2,i}} \sim \mathfrak{a}^{\alpha_2} \mathfrak{b}^{\beta_2}$. Then

$$\mathfrak{y}_1 {\mathfrak{y}_2}^{-1} \sim \prod_{i=1}^{|B|} {\mathfrak{p}_i}^{e_{1,i}-e_{2,i}} \sim \mathfrak{a}^{\alpha_1-\alpha_2} \mathfrak{b}^{\beta_1-\beta_2}$$

and we have a relation with vector $\vec{v} = (e_{1,1}-e_{2,1}, e_{1,2}-e_{2,2}, e_{1,3}-e_{2,3}, \ldots, e_{1,|B|}-e_{2,|B|})$ and $\alpha = \alpha_1 - \alpha_2$ and $\beta = \beta_1 - \beta_2$. The case when one ideal has the inverse prime of the other is similar. Note that we find the sign of the exponent on the large prime $\mathfrak{p}$ with the same equivalence used to find the signs of the exponents for the vectors e_i.

This gives rise to two variations of the random walk strategy as used by Gaudry [26]. The first is almost exactly the same as the original. Each ideal is checked to see if it is potentially smooth using the method described earlier. If so, it is completely factored and if the factors are all in B a relation has been found. Otherwise the ideal is discarded and the next one is tested. Thériault [65] analyzes this algorithm and after balancing the amount of time required for the linear algebra step with that for the relation generation, he gets a runtime of $O\left(g^5 q^{2-\frac{2}{g+1}+\epsilon}\right)$. This uses the assumption that $q > (g-1)!$. Compare this to the runtime of Gaudry above, for a fixed, small $g > 2$, and one can see that there has been an improvement.

The second variation that Thériault analyzes takes advantage of the almost smooth ideals. Again, an ideal is first tested for smoothness over $\mathcal{P}$. If it passes this first test then it is completely factored. If the ideal turns out to be smooth over B, a relation is recorded. If the ideal is almost smooth then the prime is checked to see if there is an intersection with any previously found almost smooth ideals. If so, a relation is formed and recorded, with both almost smooth ideals being removed. Otherwise, the almost smooth ideal is saved and the next ideal is checked. Again, a thorough analysis is performed by Thériault [65], and the resulting runtime, using the assumption that $q > (g-1)!/g$, is $O\left(g^5 q^{2-\frac{4}{2g+1}+\epsilon}\right)$. This is an improvement over both Gaudry's runtime and the runtime of Thériault's first variation. The drawback of this method is the increased time required for storage and searching of the almost smooth ideals.

The idea of using large primes can be adapted to work in the large genus case, and the effect of them is discussed in the next chapter.

The random walk method is obviously well suited for use with the Enge-Gaudry algorithm. In the next section we see another method for generating relations that is better suited for use with Vollmer's method.

3.2.5 Sieving

Another method of relation generation is motivated by the use of sieving in factoring, such as by Pomerance [60], Silverman [63], Boender and te Riele [10] and Kurowski [45]. Sieving is used to generate relations in order to compute class groups in quadratic number fields by Jacobson [32, 33], and by Jacobson [34] to solve discrete logarithms in quadratic number fields. Flassenberg and Paulus [24] provide an algorithm for sieving in quadratic function fields, but with characteristic not equal to 2. However, it is easy to modify this to work in the characteristic 2 case, as we see in the next chapter. Additionally, since sieving has not been studied as much in the quadratic function field setting, there are a variety of improvements that can be investigated. In the next chapter we look at some of these possible improvements.

The general idea in odd characteristic function fields is as follows. Generate a random ideal $\mathcal{J}$ by creating a random exponent vector $\vec{e} = (e_1, \ldots, e_k)$ over the factor base, where $e_i \in \{-1, 0, 1\}$, and computing $\mathcal{J} = \prod_{i=1}^{k} \mathfrak{p}_i^{e_i}$ as before. Suppose this ideal is generated by $(a, b + v)$, where $v^2 + vh = f$ and $a \mid b^2 + bh - f$, and consider an element $\alpha = aS + (b + v)T \in \mathcal{J}$, where $S, T \in \mathbb{F}_q[X]$. Then we can

compute

$$\begin{aligned} N(\alpha) &= \alpha\bar{\alpha} \\ &= (aS + (b+v)T)\,(aS + (b+h-v)T) \\ &= a^2S^2 + a(2b+h)ST + (b^2 + bh - vh - v^2)T^2. \end{aligned}$$

Since $v^2+vh = f$, this becomes $a\left(aS^2 + (2b+h)ST + \frac{b^2+bh-f}{a}T^2\right)$. Since $a = N(\mathcal{J})$ and ideal norms are multiplicative, there exists an ideal $\mathcal{K}$ such that $\mathcal{K}\mathcal{J} = (\alpha)$ with $N(\mathcal{K}) = aS^2 + (2b+h)ST + cT^2 = F(S,T)$, where $c = (b^2 + bh - f)/a$. If $F(S,T)$ is smooth for some S and T with $\gcd(S,T) = 1 = uS + vT$ then we obtain a vector $\vec{v}$ by factoring $F(S,T)$ over the norms of the elements in the factor base, as discussed above. In order to determine the correct signs for these exponents, compute $b_F = 2aSv + (2b+h)(Su - Tv) + 2cTu$ as done by Flassenberg and Paulus [24]. Then set s_i such that $b_F \equiv s_i b_{p_i} \pmod{2p_i}$ and let $\vec{w} = (s_1v_1, s_2v_2, \cdots, s_kv_k)$. A relation is then formed by $\vec{e} + \vec{w}$. Sieving allows for checking of several possible (S, T) pairs in roughly the same time required to trial divide one.

We consider one dimensional sieving, as Jacobson [33] does. Fix $T = 1$ and let $F(S) = I_1S^2 + I_2S + I_3$ be the *sieving polynomial* with coefficients in $\mathbb{F}_q[X]$, for example, obtained as in the previous paragraph. Fix a prime ideal $\mathfrak{p}$ from the factor base and consider $N(\mathfrak{p}) = p$. 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 in-

teger, 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 mark any c such that $D[c]$ is larger than a given *tolerance value* Y as a candidate. We expect that $D[c] \approx \log(F(c))$ if $F(c)$ is in fact smooth, so Y should be chosen accordingly. 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 [24] point out, sieving in the function field 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 an array of integers, as the map from $[-M, M]$ to this array is trivial. However, in the function field context it is not as clear how we can map polynomials in $\mathbb{F}_q[X]$ to an integer array 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 function field context we do not have this luxury because the distance between interesting array entries is not constant. We now cover Flassenberg and Paulus' presentation

of sieving in finite fields of odd characteristic.

First we address the problem of indexing a sieve array. Every element in the finite field $\mathbb{F}_q$ can be represented uniquely by a positive integer. As Flassenberg and Paulus [24] do, we define a map $\nu_0 : \mathbb{F}_q \rightarrow \mathbb{Z}^{\geq 0}$ 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 define a second map, $\nu : \mathbb{F}_q[X] \rightarrow \mathbb{Z}^{\geq 0}$, that takes elements of the polynomial ring to a unique integer such that for $g = g_0 + g_1 x + \cdots + g_{\deg(g)} x^{\deg(g)} \in \mathbb{F}_q[X]$, $\nu(g) = \sum_{i=0}^{\deg g} \nu_0(g_i) q^i$. Using ν, we can represent the sieve array as an integer array with bounds from 0 to a chosen upper sieving bound M, where each spot represents an evaluation of $\nu(x)$.

While Flassenberg and Paulus [24] describe two dimensional sieving, we continue to only consider one dimensional sieving. The results of Flassenberg and Paulus [24] suggest that using a small value for one sieving dimension and a large value for the other dimension works best. They also make the suggestion that while not proven optimal in the context of function fields, experiences from factoring suggest that setting one dimension to 1 works well in practice. Finally, sieving is often presented in one dimension, such as by Pomerance [60], Silverman [63], Alford and Pomerance [8], Jacobson [33] and many others. For ease of implementation we follow this lead and use one dimensional sieving.

Suppose we wish to find all smooth values of a given sieve polynomial $F(S)$ with coefficients in $\mathbb{F}_q[X]$ over the set of polynomials with degree less than or equal to some positive integer M. Initialize the sieve array D by setting $D[\nu(i)] = \deg(F(i))$ for $0 \leq \nu(i) \leq \nu(X^M)$. This is done in a clever manner by Flassenberg and Paulus [24]. We do not describe this or anything else related to this idea here as our implementation, described in the next chapter, makes use of a generalization of the tolerance value method described above. Then for each prime ideal $\mathfrak{p}$ in the factor base, compute the roots of $F(K)$ modulo $N(\mathfrak{p}) = p$. The roots are computed using the quadratic formula. Note that this requires the characteristic of the field to be not equal to two.

For each $\mathfrak{p}$ in the factor base we find all polynomials z with degree less than or equal to M such that $N(\mathfrak{p})$ divides $F(z)$ as follows. For each root r, add $\deg p$, where $p = N(\mathfrak{p})$ to $D[\nu(r)]$. Now we must jump through the sieve array, marking $D[\nu(r + ip)]$ as being divisible by p for $i \in \mathbb{F}_q[X]$, $0 \leq \nu(r + ip) \leq \nu(X^M)$. Do this by setting $D[\nu(r + ip)] = D[\nu(r + ip)] + \deg(p)$ for each i. After this process has been completed for all prime ideals in the factor base, if $D[\nu(z)]$ is greater than a given tolerance value then $F(z)$ is a smooth candidate, and should be tested using the previously described method of smoothness testing.

In order to move from $\nu(r + i_j p)$ to $\nu(r + i_{j+1} p)$, Flassenberg and Paulus [24] give the following procedure. Set z to be the current possible input for $F(X)$ that is being tested for smoothness. When adding 1 to the constant term of i_j

does not equal q, set $z = z + p$ and $i_{j+1} = i_j + 1$. Otherwise, suppose $i_j = (i_j)_0 + (i_j)_1 X + \cdots + (i_j)_{\deg i_j} X^{\deg i_j}$ and let $l = \max\{m \mid (i_j)_m + 1 = q\}$. Then compute $z = z + X^l p$, set $i_{j+1} = i_j$, $(i_{j+1})_{l+1} = (i_{j+1})_{l+1} + 1$ and $(i_{j+1})_m = 0$ for $0 \leq m \leq l$. Finally, compute $\nu(z)$ and mark $D[\nu(z)]$ as being divisible by p.

The above tells us how we can use sieving to compute relations of the form required for use in Vollmer's algorithm. In fact, sieving can also be used in the Enge-Gaudry algorithm. In that case, multiply the ideal $\mathcal{J}$ used above to find a sieving polynomial by $\mathfrak{a}^\alpha \mathfrak{b}^\beta$ and use the reduced result to generate a sieving polynomial. When a relation is found, we proceed as in a manner similar to that done in order to use the random exponents method with the Enge-Gaudry algorithm.

Sieving is an efficient way to test $F(z)$ for smoothness for several possible z values. One already discussed improvement that can be considered is the use of large primes, as done by Lenstra and Manasse [48], Boender and te Riele [10] and Jacobson [33]. In order to do this, use a smaller tolerance value and use the same large prime procedure as used in the random walk method.

3.2.6 Self Initialized Sieving

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, as done by Alford and Pomerance [8] and Contini [16], and computing class groups in quadratic number fields, as done by Jacobson [33], 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 quadratic function field case we are interested in, we describe the idea behind self-initialization in this section and in the next chapter we describe the details for our context.

Suppose we have an ideal $\mathfrak{a}$ with norm a that is the product of j distinct prime ideals, $\mathfrak{q}_i$ with norms q_i. Then $\mathfrak{a} = \prod_{i=1}^{j} \mathfrak{q}_i^{v_i}$ where $v_i \in \{-1, 1\}$ and $a = \prod_{i=1}^{j} q_i^{|v_i|}$. By varying the signs of v_i, different ideals with the same norm can be formed. In fact, there are 2^j ideals generated by changing the v_i values. Note that if a relation is formed by $\vec{v} + \vec{w}$ then $-\vec{v} - \vec{w}$ is also a relation. However, these two relations essentially "cancel" each other out since one is simply the negation of the other. Thus, if both relations are used we will have a relation and a multiple of that relation in the relation matrix, something we are trying to avoid for the reasons discussed in Section 3.1.2. In order to avoid these situations we fix v_j and only generate 2^{j-1} unique ideals. The trick is computing these ideals without actually computing the various products of prime ideals. In fact, both this and the computation of most of the roots can be done in an efficient manner.

Note that self-initialized sieving does not work when looking for relations of the form $\prod_{i=1}^{k} \mathfrak{p}_i^{e_i} \sim \mathfrak{a}^{\alpha}\mathfrak{b}^{\beta}$, as in the Enge-Gaudry algorithm. In particular, multiplying the sieving polynomial by a new element requires one to recompute the roots of the sieving polynomial, negating that benefit.

Jacobson [33] had significant improvement when using self-initialization to create

relations in his algorithm for class group computations in quadratic number fields. We can adapt the above algorithm to work in characteristic two quadratic function fields, and this is covered in the next chapter.

As mentioned before, all of these relation generation methods are easily parallelized. For the random exponents method, each slave process can compute relations individually, returning relations back to the master process. In the random walks method, each slave process can be set upon its own random walk, and again relations found can be sent to the master process. In the sieve method, the master process can distribute sieving polynomials to the slave processes and relations are sent back to the master.

We have now covered the various relation generation strategies, some improvements that have been made to them in various mathematical settings and how we can efficiently test for smoothness over the factor base. The next section briefly describes the linear algebra algorithms that can be used to solve the problems we have seen.

3.3 Linear Algebra

Every index calculus algorithm has two major computationally intense phases. The first is the relation generation stage, discussed above. The second is the linear algebra stage. Both index calculus algorithms require a solution to a linear system to be found. Let A be the relation matrix. Then the Enge-Gaudry algorithm requires a solution to $A\vec{x} = \vec{0} \pmod{N}$, where $\vec{x}$ is non-zero, that is, a random, non-zero

element in the kernel of A. Vollmer's algorithm requires a solution to the system $A'\vec{x} = (1, 0, \ldots, 0)^T$, where A' is as in the above description of Vollmer's algorithm. If we continue with the assumption that we have the group order N then we can complete Vollmer's algorithm by finding a solution to the above system modulo N. Both of these problems can then be solved using variations of Wiedemann's algorithm [69] or Lanczos' algorithm [47], because the nature of the index calculus algorithms we use results in sparse relation matrices. Both algorithms can take advantage of this as they rely on computing a number of matrix-vector products, which are especially efficient when the matrix is sparse.

It should be noted that there are versions of both Wiedemann's algorithm and Lanczos' algorithm that can be parallelized. For example, see the work done by Kaltofen [40] on Coppersmith's block version of Wiedemann's algorithm and the work done by Montgomery [53] on a block version of Lanczos' algorithm.

Linear algebra was not the focus of this thesis. While the linear algebra algorithm used does have an effect on the overall run time, the relation generation strategies will perform the same no matter which linear algebra strategy is chosen. A change in the linear algebra strategy would most likely require a change in the parameters used in the relation generation strategies, such as the size of the factor base, and this is addressed in Chapter 5. We used an existing implementation in order to solve our linear algebra problems. Since no known, publicly available implementation of the block algorithms exists at this time, we used the C++ library called LinBox [5], designed for efficient, exact linear algebra over a variety of types of matrices,

including sparse matrices. We discuss exactly what functions we used, and why we made these decisions, in Chapter 5.

In this chapter we have covered index calculus algorithms, and a variety of algorithms used in the process of running the index calculus algorithms. Both the Enge-Gaudry [22] and Vollmer [66] algorithms work in the quadratic function field case without any further discussion. However, we have yet to explain how Thériault's improvements [65] to random walks can be adapted to the large genus case. We also have to discuss how Flassenberg and Paulus's sieving method [24] can work in a characteristic two function field and how self-initialized sieving can be adapted to work in the function field case. These issues, along with further improvements and a method for estimating results are all covered in the next chapter.

Chapter 4

Algorithmic Improvements And Analysis

In the previous chapter we looked at index calculus algorithms for solving the discrete logarithm problem. These algorithms have two computationally intense stages. The first is the relation generation stage, and the second is the linear algebra stage. The focus of this research was the improvement of the relation generation stage of the index calculus algorithms used to perform discrete logarithm computations in quadratic function fields. In this chapter we look at how improvements suggested in other settings can be applied to quadratic function fields. In particular, we discuss using large primes in the high genus case and computing estimates for the number of ideals we expect to check before finding enough relations using the random walk method, allowing an optimal configuration of parameters to be derived empirically. We also generalize sieving and self-initialized sieving to the characteristic two case, discuss different methods of implementing the sieve array, and look at the idea of sieving with only part of the factor base. We provide some guidance to picking the various parameters needed in these algorithms as well.

We studied two different methods of finding relations in the previous chapter. The first is the random walk method. We saw how using large primes can improve the random walk algorithm in small genus hyperelliptic curves, as analyzed by Thériault [65]. This can be applied to the large genus case as well and we explain how this

can be done in two different ways in this chapter. We also discuss the work done by Jacobson, Menezes and Stein [35] to develop a method of computing estimates for the number of steps needed in order to find a sufficient number of relations in order to form the linear system required to solve the discrete logarithm problem. In this chapter we extend these estimates to incorporate the large prime idea, allowing us to compare various combinations of parameters in the random walk method of generating relations. In particular, this provides us with an optimal value t for the factor base bound and allows us to select a large prime strategy that should offer the best performance in practice.

We also looked at sieving in the previous chapter. In this chapter we show how self-initialized sieving can be applied in the even characteristic quadratic function field setting and how we can implement the sieve array to improve performance. We also discuss how the actual amount of time spent sieving can be reduced at the possible cost of testing more smooth candidates. We conclude with some discussion about the selection of sieving parameters.

4.1 Random Walk Improvements

Recall the description of random walks from Chapter 3. We saw that each new reduced ideal in the random walk is computed with a single ideal multiplication. This results in an efficient computation, but there is room for improvement. As we saw in the previous chapter, using large primes in the small genus hyperelliptic curve case results in an asymptotic improvement, as does reducing the size of the

factor base through the use of the parameter r. We generalize these ideas to the high genus case in the following.

4.1.1 The Parameter r

Thériault [65] focused on the small genus case, working under the assumption that $q > (g-1)!/g$. In these cases there are a large number of prime ideals $\mathfrak{p}$ such that $\deg(N(\mathfrak{p})) = 1$. In fact, from the work presented by Gerhard and von zur Gathen [68], there are $\Theta(q)$ such prime ideals. As the genus grows, this too grows quickly. Taking all of these prime ideals for the factor base can result in a rather large factor base. In turn, this means that more relations must be found, resulting in a larger linear algebra problem. Ideally it would be helpful to make the factor base smaller. Thus, the introduction of a parameter r into the analysis. Thériault lets r be such that $2/3 < r < 1$ and considers a factor base with q^r prime ideals with norms having degree equal to 1.

We now generalize this to the large genus case. When the genus is relatively large compared to q we require more than just the prime ideals having norm equal to 1 in our factor base in order to ensure the factor base is large enough to generate the entire class group. We use the bound t to limit the degrees of the norms of the prime ideals in our factor base. From the work presented by Gerhard and von zur Gathen [68, Chapter 14] we know that there are $\Theta\left(\frac{q^t}{t}\right)$ monic irreducible polynomials of degree t in $\mathbb{F}_q[x]$. This function increases significantly as t increases. In turn, the number of polynomials that generate prime ideals for the factor base also increases. Thus, a factor base with bound t contains an overwhelming number of prime ideals

with norms having degree t when compared to the number of prime ideals with norms having degree less than t. Here is where we apply the idea of reducing the size of the factor base.

Similar to that done by Thériault [65], we introduce a parameter r such that $0 < r \leq 1$. Then we create a factor base that contains all prime ideals with norms having degree up to and including $t-1$. In addition, if there are A_t prime ideals with norms having 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 will let $\mathcal{P}_t$ denote the set of prime ideals with norms having degree up to and including t.

The random walk operates in a similar manner as before. An ideal is tested for potential smoothness over $\mathcal{P}_t$ using the test described in the previous chapter. If the ideal passes this first test, the norm is then explicitly factored to see if it is actually smooth over the norms of the prime ideals 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 work. In the next section we investigate this.

4.1.2 Large Primes

When Thériault [65] studied large primes, he used the prime ideals having norms with degree equal to 1 that are not in the factor base as large primes. 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 $0 < r < 1$ and that we form the factor base as in the previous section. Then there are some prime ideals having norms with degree equal to t that are not in the factor base. Similar to Thériault [65], these are our large primes. These large prime ideals work in the exact same manner as before. An ideal is tested for smoothness over $\mathcal{P}_t$ using the test from the previous chapter. If the ideal is indeed smooth over $\mathcal{P}_t$, we factor it using the method discussed in Chapter 3. If all the factors are in the factor base, we know the ideal is smooth. If there is only one factor not in the factor base, the ideal is almost smooth and so we check to see if it intersects with any previous almost smooth ideals. If there is an intersection we have a relation as discussed previously, and the relation is recorded. Otherwise the almost smooth ideal is stored. If the tested ideal is neither smooth nor almost smooth, it is discarded.

The second idea is to fix the degree bound for the norms of the ideals in the factor base to be t and perform our smoothness test over the set $\mathcal{P}_{t+1}$, the prime ideals with norms having degree less than or equal to $t + 1$. We allow the parameter r to be in the range $(0, 1]$ and thus the large prime ideals include all of the prime ideals with norms having degree equal to $t+1$ and possibly prime ideals with norms having degree t, if r is not equal to 1. We proceed in a similar manner. Test an ideal for smoothness over $\mathcal{P}_{t+1}$. If this ideal is smooth over $\mathcal{P}_{t+1}$, we completely factor the ideal using the method from the previous chapter. As above, if all the factors are in the factor base we have a smooth ideal and record the resulting relation. If

there is only one factor that is not in the factor base, we check for intersections, recording either the resulting relation if one exists or the almost smooth ideal for future consideration. If there is more than one factor that is not in the factor base, we discard the ideal.

We expect an improvement in computational speed when using these methods as we expect fewer random walk steps to be necessary in order to find a sufficient number of relations. However, note that the intersection search takes time, and may have to be considered. We reduce the number of times we have to search by holding off until the end to perform it, and this appears to reduce the time needed to perform this search, making it almost negligible. Specifically, the search is done in two stages. First, the list of almost smooth ideals is sorted by the large prime factors. Then the almost smooth ideals with the same large prime factors are combined, resulting in a smooth ideal, and thus a relation. Performing the sort a single time at the end of the relation generation phase as opposed to repeating it for every new almost smooth ideal seems to result in a more efficient implementation. In the next section we look at how we can compare the number of steps needed to find enough relations to compute the solution to the discrete logarithm problem.

4.1.3 Estimated Results

Jacobson, Menezes and Stein [35] calculate accurate estimates for the number of random walk steps (that is, the number of ideals checked for smoothness) needed to find the relations necessary to generate the appropriate linear system for solving the discrete logarithm problem. These are based on counting the number of smooth

ideals expected and the total number of reduced ideals representing the classes in the ideal class group. We first cover the calculations done by Jacobson, Menezes and Stein [35] and then present the modifications needed to compute similar results for the large prime variations. Finally, we present an example of these formulas in use.

Original Enge-Gaudry Algorithm

We use the notation developed by Jacobson, Menezes and Stein [35]. Let A_l be the number of irreducible polynomials of degree l that split in the ring $\mathbb{F}_q[X]$, where $1 \leq l \leq t$. From the work presented by Gerhard and von zur Gathen [68, Chapter 14] 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, that is, $\mu(x)$ 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, stated by Jacobson, Menezes and Stein [35], which gives us

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

Recall that from each irreducible polynomial that splits we obtain two prime ideals, one of which becomes an element in the factor base, the other is the inverse of this ideal.

As done by Jacobson, Menezes and Stein [35], and generalized by Maurer, Menezes

and Teske [49], let $M(g,t)$ denote the number of t-smooth reduced ideals with norms having degree at most g. We have

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

where we use $[x^i](f(x))$ to denote the coefficient of x^i in $f(x)$. 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, as Jacobson, Menezes and Stein [35] do, that reduced t-smooth ideals are distributed evenly in the class group. With this assumption we can compute the expected number of random walk iterations needed to find a reduced t-smooth ideal. Let $E(t,g) = \lceil N/M(g,t) \rceil$, where N is the class number and g is the genus, represent this value.

Since each splitting polynomial gives rise to a prime ideal in the factor base, let $F(t) = \sum_{l=1}^{t} A_l$ be the size of the factor base. Jacobson, Menezes and Stein [35] 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 $T(t,g) = (F(t) + 5)E(t,g)$ ideals 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 [65], the only change to the above work is the number of prime ideals with norms having degree t in the factor base. Thus, we simply repeat the above calculations, using $F(t) = \sum_{l=1}^{t-1} A_l + rA_t$. We also recompute $M(g,t)$ using

rA_t in place of A_t to find the first $g+1$ terms of Taylor expansion of $\prod_{l=1}^{t}\left(\frac{1+x^l}{1-x^l}\right)^{A_l}$ about $x=0$. We then compute $E(t,g)=N/M(g,t)$ and $T(t,g)=(F(t)+5)E(t,g)$ as in the previous case.

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 simple situation: setting $r=1$ and using the prime ideals having norms equal to degree $t+1$ as large primes. Then during the search for relations we encounter both smooth ideals and almost smooth ideals. We have to consider how many of each are encountered in the search.

The number of t-smooth ideals remains $M(g,t)$ as defined above. We also need to count the number of almost smooth ideals. An almost smooth ideal is one that is smooth over the prime ideals having norms with degree less than or equal to t, with the exception of a single factor which has a norm with degree equal to $t+1$. Then the number of almost smooth ideals is $M(g-(t+1),t)$, the number of t-smooth ideals having norms with degree less than or equal to $(g-(t+1))$, multiplied by the number of prime ideals having norms with degree equal to $t+1$ available to "complete" the t-smooth ideals having norms with degree less than or equal to $(g-(t+1))$ to almost smooth ideals. An ideal having norm with degree equal to $(g-(t+1))$ multiplied by an ideal having norm with degree equal to $t+1$ has a norm with degree equal to g, so the ideals we are counting are reduced.

Recall that A_l represents the number of irreducible polynomials of degree l and each irreducible polynomial actually gives rise to two prime ideals, an ideal and its inverse. Then there are $2A_{t+1}$ large prime ideals and a total of $A(g,t) = 2A_{t+1}M(g-(t+1),t)$ almost smooth ideals. We assume that these are also evenly distributed. Then the number of steps required to find an almost smooth ideal is $E_{LP}(t,g) = \lceil N/A(g,t) \rceil$ and the number of steps required to find a smooth ideal is $E(t,g) = \lceil N/M(g,t) \rceil$ as described before.

If in the course of our search we find x almost smooth ideals, then we expect to take $xE_{LP}(t,g)$ steps, and, in the process, to find $x\frac{E_{LP}(t,g)}{E(t,g)} = x\frac{M(g,t)}{A(g,t)}$ smooth ideals. We would like to compute the number of relations we expect to obtain from x almost smooth ideals. Thériault [65, Section 5.6] provides us with the information to be able to compute this. 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 Thériault [65, Theorem 1] 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 ideals, and $s = x$ is the number of almost smooth ideals 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 ideals.

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 ideals 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

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

This gives us an expected value of $T(t,g) = xE_{LP}(t,g)$ 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 ideals 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 ideals resulting from fixing r. Then as before we have $2L(t,r)$ large prime ideals and $A(g,t) = 2L(t,r)M(g-t,t)$ almost smooth ideals, where rA_t is used in place of A_t in computing $M(g-t,t)$, $M(g,t)$ and $F(t)$, and $E_{LP}(t,g) = \lceil N/A(g,t) \rceil$ as before. We then find x using the same method as above and denote the number of ideals we expect to be tested by $T(t,g) = xE_{LP}(t,g)$.

Using large primes with $r \neq 1$ and degree $t+1$ polynomials

Finally, we consider the case where we allow r to vary and use both the remaining prime ideals having norms with degree equal to t along with the prime ideals having norms with degree equal to $t+1$ as large primes. Then $L(t,r) = A_t - rA_t$ 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 ideals, 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 $E_{LP}(t,g) = \lceil N/A(g,t) \rceil$, x and $T(t,g) = xE_{LP}(t,g)$ using the previous method.

Example

In this section we will demonstrate the use of the above formulas, comparing the expected results when the different implementations of large primes are used. We consider the function field associated with the hyperelliptic curve called C155 by Jacobson, Menenzes and Stein [35] which is defined by $y^2 + h(u)y = f(u)$ over $\mathbb{F}_{32}$ where

$$f(u) = w^4u^{63} + w^6u^{62} + w^{15}u^{60} + w^{26}u^{56} + w^{25}u^{48} + w^7u^{32} + w^{13},$$

$$h(u) = w^2u^{31} + w^7u^{30} + w^{30}u^{28} + w^{22}u^{24} + w^3u^{16} + w^{22},$$

and $\mathbb{F}_{32} = \mathbb{F}_2[w]/(w^5 + w + 1)$. It is known that for this function field the class number $N = 2 \cdot 228359630832953580969327277630652669728815410 89$. Also note that the genus is 31. The number of irreducible polynomials of degree l that split in this field, A_l, is known (or in the case of $l = 5$, approximated using the formula stated above) and provided in Table 4.1. Throughout the course of this example we use $t = 4$.

l	A_l
1	16
2	240
3	5456
4	130816
5	3355440

Table 4.1: Approximate number of degree l splitting irreducible polynomials

We begin by considering the original Enge-Gaudry algorithm as presented by Ja-

cobson, Menezes and Stein [35]. Since $t = 4$, we have

$$M(g,t) = M(31,4) = \sum_{i=1}^{31} \left([x^i] \prod_{l=1}^{4} \left(\frac{1+x^l}{1-x^l} \right)^{A_l} \right).$$

The first 32 terms of the Taylor expansion of

$$\prod_{l=1}^{4} \left(\frac{1+x^l}{1-x^l} \right)^{A_l} = \left(\frac{1+x}{1-x} \right)^{16} \left(\frac{1+x^2}{1-x^2} \right)^{240} \left(\frac{1+x^3}{1-x^3} \right)^{5456} \left(\frac{1+x^4}{1-x^4} \right)^{130816}$$

about $x = 0$ are

$$1 + 32x + 992x^2 + 31744x^3 + 1015808x^4 + \cdots +$$

$$3101208034903881649366582297355475750396 8x^{31}.$$

Summing the coefficients of x to x^{31} gives us

$$M(g,t) = M(31,4) = 331346358928723812266221500025586417104 96.$$

Then we have

$$\begin{aligned} E(t,g) &= E(4,31) \\ &= \lceil N/M(31,4) \rceil \\ &= \left\lceil \left(\frac{2 \cdot 228359630832953580969327277630652669728815410 89}{331346358928723812266221500025586417104 96} \right) \right\rceil \\ &= 1378374. \end{aligned}$$

In this case we get $F(t) = F(4) = 16+240+5456+130816 = 136528$. Thus, we have $T(t,g) = T(4,31) = (F(4)+5)E(4,31) = (136528+5)1378374 = 188{,}193{,}537{,}342$, and we expect to test that many ideals before finding enough relations using the Enge-Gaudry algorithm.

We now consider adjusting the parameter r. We consider $r = 0.75$. Then we obtain a factor base of size $F(t) = A_1 + A_2 + A_3 + rA_4 = 16 + 240 + 5456 + 98112 = 103824$. Recomputing $M(g,t)$ with rA_4 in place of A_4 means we must compute the first 32 terms of the Taylor expansion of

$$\prod_{l=1}^{4}\left(\frac{1+x^l}{1-x^l}\right)^{A_l} = \left(\frac{1+x}{1-x}\right)^{16}\left(\frac{1+x^2}{1-x^2}\right)^{240}\left(\frac{1+x^3}{1-x^3}\right)^{5456}\left(\frac{1+x^4}{1-x^4}\right)^{98112}$$

about $x = 0$. In fact, the result is

$$1 + 32x + 992x^2 + 31744x^3 + 950400x^4 + \cdots$$
$$+ 99399169371169796855882428475249022432 96x^{31}.$$

Summing the coefficients of x to x^{31} gives us

$$M(g,t) = M(31,4) = 106541984168350643906900841145625477564 48$$

and we find that $E(4,31) = \lceil N/M(31,4)\rceil = 4286754$. Then we expect to test $T(t,g) = T(4,31) = (F(4)+5)E(4,31) = 445{,}089{,}381{,}066$ ideals before finding $F(t)+5$ relations.

For our third case we will use $r = 1$ and add large primes. Then we have

$$M(g,t) = M(31,4) = 331346358928723812266221500025586417104 96$$

t-smooth ideals. We must now compute the number of almost smooth ideals. The number of t-smooth ideals having norms with degree less than or equal to $(g-(t+1))$ is

$$M(g-(t+1),t) = M(26,4) = 323598773613568409940771598841925 12,$$

so the number of almost smooth ideals is $A(g,t) = A(31,4) = 2A_5M(26,4) = 2 \cdot 3355440 \cdot M(26,4)$. Then $E_{LP}(t,g) = E_{LP}(4,31) = \lceil N/A(31,4) \rceil = 210311$. As before, we find that $E(t,g) = E(4,31) = 1378374$ and $F(t) = F(4) = 136528$. We read $A_5 = 3355440$ from Table 4.1. We now have everything we need to solve

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

for x, yielding $x = 633522$. So the number of ideals we expect to test in this case is $T(t,g) = T(4,31) = xE_{LP}(4,31) = (633522) \cdot (210311) = 133{,}236{,}645{,}342$.

Our fourth case again makes use of the parameter r, which we will set to 0.75, and we will consider only the remaining prime ideals having norms with degree t as large primes. Then again our factor base has size $F(t) = F(4) = A_1 + A_2 + A_3 + rA_4 = 103824$. Since $rA_4 = 98112$, $L(4, 0.75) = A_4 - (0.75)A_4 = 32704$. We also have $M(31,4) = 1065419841683506439069008411456254775644 8$ as in the previous case when $r = 0.75$. We find that we have $M(g-t,t) = M(27,4) = 2026212150411721816944777806289560 32$ t-smooth ideals having norms with degree less than or equal to $g-t$, and $A(g,t) = A(31,4) = 2 \cdot 32704 \cdot M(27,4)$ almost smooth ideals exist. This gives us $E_{LP}(4,31) = \lceil N/A(g,t) \rceil = 3446145$ expected steps to find an almost smooth ideal. Now we solve

$$x\frac{M(31,4)}{A(31,4)} + \frac{2}{3}\frac{x^2}{2L(4,0.75)} = F(4) + 5$$

for x, getting $x \approx 68925$. Finally, $T(4,31) = xE_{LP}(4,31) \approx (68925) \cdot (3446145) = 237{,}525{,}544{,}125$ steps are expected to be taken in order to complete our system.

Finally, this fifth case will use $r = 0.75$ and use all prime ideals having norms with degree less than or equal to $t + 1$ that are not in our factor base as large primes. Then once again we have $F(4) = 103824$ and $L(t, r) = L(4, 0.75) = 32704$ as before. We require the value of $A(g, t) = 2L(t, r)M(g - t, t) + 2A_{t+1}M(g - (t + 1), t)$, the number of large prime ideals. We again compute

$$M(31, 4) = 10654198416835064390690084114562547756448.$$

The value $M(27, 4) = 20262121504117218169447778062895603 2$ is computed above in the previous example and $M(26, 4)$ must be recomputed for $r = 0.75$, giving

$$M(26, 4) = 13033029177835826311694870811268992.$$

This gives us $A(31, 4) = 2 \cdot 32704 \cdot M(27, 4) + 2 \cdot 3355440 \cdot M(26, 4)$ and proceed to see that $E_{LP}(4, 31) = 453472$. Now we must solve

$$x\frac{M(31, 4)}{A(31, 4)} + \frac{2}{3}\frac{x^2}{2(L(4, 0.75) + A_5)} = F(t) + 5.$$

We get $x = 621862$ and so $T(4, 31) = (621862) \cdot (453472) = 281{,}997{,}004{,}864$ is the expected number of ideals that are tested in order to find 103829 relations for our linear system.

In order to summarize the results of our example, we present the factor base sizes and the number of steps we expect to take for each of the above cases in Table 4.2. From this table we can see that Case 3, the case with large primes consisting of the prime ideals having norm with degree equal to $t + 1$ and $r = 1$ requires the fewest expected number of steps in order to find the required $F(t) + 5$ relations.

Note, however, that in this case we expect the linear algebra stage to take longer because the larger factor base results in a larger linear system. Thus it is important to consider both stages of the algorithm, and what methods are being used for each stage, when determining optimal settings.

Case	$F(t)$	$T(t,g)$
1	136528	188193537342
2	103824	445089381066
3	136528	133236645342
4	103824	237525544125
5	103824	281997004864

Table 4.2: Summary of the results in the example

Using the formulas presented in this section we can estimate the number of steps required to find a sufficient number of relations to complete the linear system. Given empirical data about the amount of time required to generate and test an ideal, these computations can give estimated runtimes for various combinations of t, r and what large prime variant is being used. 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. This is explored further in the next chapter.

In this section we have seen how the large prime improvement analyzed by Thériault [65] for the small genus case can be applied to the large genus case. We have also seen how we can calculate the number of ideals we expect to test for smoothness

before we find a given number of relations. The next section looks deeper into the implementation of sieving, and various improvements that can be made to the sieving algorithm as it operates in the quadratic function field setting.

4.2 Improvements to Sieving

Sieving in characteristic two fields does introduce some challenges. Considering computations modulo 2 means we cannot compute the roots as we normally would. 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. As in the previous chapter, if we have an ideal $\mathcal{J} = (a, b + v)$, then for any $\alpha = aS + (b + v)T \in \mathcal{J}$ with $S, T \in \mathbb{F}_q[X]$ and $a \mid b^2 + bh + f$, we have

$$\begin{aligned} N(\alpha) &= (aS + (b + v)T)\,(aS + (b + h + v)T) \\ &= a\left(aS^2 + hST + \frac{b^2 + bh + f}{a}T^2\right). \end{aligned}$$

Since $a = N(\mathcal{J})$ there exists an ideal $\mathcal{K}$ with $N(\mathcal{K}) = F(S,T) = aS^2 + hST + cT^2$, where $c = \frac{b^2+bh+f}{a}$ such that $\mathcal{K}\mathcal{J} = (\alpha)$. If $F(S,T)$ is smooth over the factor base for some $S, T \in \mathbb{F}_q[X]$ then we can factor $\mathcal{K}$ over the factor base as described before and, if $\mathcal{J}$ was selected to be smooth, we obtain a relation. We therefore sieve $F(S) = F(S, 1)$ with the norms of the prime ideals in the factor base in order to find values of S for which $F(S)$ is smooth.

Recall that the first step in sieving is to find the roots of $F(S)$ modulo the norms of the prime ideals in the factor base. Since we are working over a characteristic two field we cannot use the familiar quadratic formula to find the roots of this

polynomial. Therefore we proceed as follows. If $p \mid a$ then

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

and $r \equiv ch^{-1} \pmod{p}$ is clearly the single root of $F(S)$ modulo p. 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 of $F(S)$ modulo p 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, presented by Buchmann and Paulus [11], just as Flassenberg and Paulus [24] do. Then if $r \pmod{p}$ is a root, so is $r + ha^{-1} \pmod{p}$. Thus we can compute the roots of the sieving polynomial and proceed with the rest of the algorithm.

Sieving has not seen as much attention in the quadratic function field setting as the random walk strategy has. As such, it is probable that there are a variety of improvements to be made. In this section we describe some possible changes to sieving and why they could result in an improvement to the method of Flassenberg and Paulus [24]. We also look at what is involved in using self-initialized sieving in characteristic two fields. Finally, we provide some direction for choosing good sieve parameters.

4.2.1 Sieve Array Implementation Methods

One area where sieving could be improved is in the implementation of the sieve array. As originally presented by Flassenberg and Paulus [24], moving through the sieving array from root index to root index requires the computation of

$$\nu(r + kp) = \sum_{i=0}^{\deg(r+kp)} \nu_0((r + kp)_i)q^i$$

for $k \in \mathbb{F}_q[X]$ by first computing $r + kp = \sum_{i=0}^{\deg(r+kp)}(r + kp)_i x^i$, then using the map ν_0, and finally the map ν. We improve on this by working directly with the images of the function ν_0.

Note that this description holds for the even characteristic case. First, suppose $r = \sum_{i=0}^{\deg(r)} r_i x^i$, and compute and store $\nu_0(r_i)$ for all i. Then for each prime ideal $\mathfrak{p}$ with norm $p = \sum_{i=0}^{\deg(p)} p_i x^i$, compute and store $\nu_0(p_i)$ for all the coefficients of p. Recall that each evaluation of $\nu_0(y)$ is an evaluation of y at 2. Instead of computing kp we compute $\sum_{i=0}^{\deg(kp)} \nu_0((kp)_i)x^i$ using the same method for finding $r + kp$ described in Chapter 3, where in this case the precomputed coefficients $\nu_0(p_i)$ are be combined using exclusive-or and $r = 0$. Clearly $\nu_0(a + b) = \nu_0(a) \oplus \nu_0(b)$, where $\oplus$ denotes exclusive-or. Then we have $\sum_{i=0}^{\deg(r)} \nu_0((r + kp)_i)x^i = \sum_{i=0}^{\deg(r)}(\nu_0(r) \oplus \nu_0((kp)_i))x^i$ and we can compute $\nu(r + kp)$ by evaluating at q. By precomputing $\nu_0(p_i)$ we avoid having to compute the coefficients later for each location of the sieve array we have to mark as being divisible by p. That is, we remove the function ν_0 from inside the sieving loop.

The goal of this alternate sieve array implementation is to cut down on the amount

of time spent actually sieving by reducing the amount of computation done for each potentially smooth candidate. The next suggestion tries to reduce the time spent sieving by reducing the number of prime ideals we sieve with.

4.2.2 Low-degree Sieving

As discussed earlier, the factor base consists of a much larger number of prime ideals having norms with large degree than ideals with small degree norms. In addition, considering the norms of these prime ideals, we expect fewer of the higher degree irreducible polynomials to divide the test polynomials generated as described at the beginning of Section 4.2, because it is more likely that small degree irreducible polynomials will divide a polynomial of a given degree than a larger degree irreducible polynomial would. 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, sieving with the norms of prime ideals that have degree up to $t-1$ when the factor base degree bound is t. The tolerance value can then be adjusted accordingly to account for the possibility of a higher degree factor that has not been sieved. How we do this is discussed at the end of the chapter. Clearly the amount of time spent sieving is reduced, but there is a tradeoff 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.

In the next section we look at another improvement to sieving. Here we hope to have an improvement by reducing the amount of time needed to change sieve polynomials.

4.2.3 Self-Initialization

In the previous chapter we introduced the idea of self-initialized sieving. Here we generalize the idea to even characteristic function fields. Suppose we have a set $Q = \{\mathfrak{q}_1, \mathfrak{q}_2, \ldots, \mathfrak{q}_j\}$ of j distinct prime ideals from the factor base. Then the ideals $\mathfrak{a} = \prod_{i=1}^{j} \mathfrak{q}_i^{v_i}$ where $v_i \in \{-1, 1\}$ are used to form the sieve polynomials. By varying the sign of v_i we can compute 2^j ideals with norm $a = \prod_{i=1}^{j} q_i$, where $q_i = N(\mathfrak{q}_i)$ for all i. As explained in the previous chapter, we only use 2^{j-1} of these ideals, $\mathfrak{a}_1 = (a, b_1)$, $\mathfrak{a}_2 = (a, b_2)$, $\cdots$, $\mathfrak{a}_{2^{j-1}} = (a, b_{2^{j-1}})$, giving rise to 2^{j-1} unique sieving polynomials of the form $F_i(S) = aS^2 + hS + c$, where $c = \frac{b_i^2 + b_i h + f}{a}$. Self-initialized sieving gives us an efficient way to change sieving polynomials.

Similar to the work by Jacobson [33, Theorem 4.1], let $a = \prod_{i=1}^{j} q_i$, and for $1 \leq i \leq j$,

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

and

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

where t_i and $t_i + h$ are solutions to $x^2 + xh \equiv f \pmod{q_i}$. Then $b = \pm B_1 \pm B_2 \pm \cdots + B_j$ (we fix the sign in front of B_j, just as we fix v_j below) is a solution to $x^2 + xh \equiv f \pmod{a}$, where we define $-B_i$ to be 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 $b^2+bh+f \equiv t_i^2+t_ih+f \equiv 0 \pmod{q_i}$ 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 $b_1 = \sum_{i=1}^{j} v_i B_i$, giving

$$F_1(S) = aS^2 + hS + ({b_1}^2 + b_1 h + f)/a.$$

Note that we keep $v_j = 1$ in order to ensure we do not use both $\vec{v}$ and $-\vec{v}$. Now, consider a $j-1$ bit Gray code, starting with $\vec{v} = (1, 1, \ldots, 1)$. Recall that with a Gray code ordering only one bit changes as we move from one element to the next. For example, a 3-bit Gray code ordering is 000, 001, 011, 010, 110, 111, 101, 100. 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. We determine k by finding the value such that $2^k \mid\mid 2(l-1)$, like Jacobson [33].

Suppose $\mathfrak{a}_{l-1} = (a, b_{l-1})$ with $b_{l-1} = v_1B_1 + v_2B_2 + \cdots + v_kB_k + v_{k+1}B_{k+1} + \cdots + B_j$. Then $b_l = v_1B_1 + v_2B_2 + \cdots + (-v_k)B_k + v_{k+1}B_{k+1} + \cdots + B_j$ where $-B_k$ is defined as above, and we have $b_l = b_{l-1} + B_k + B_k'$. This gives $\mathfrak{a}_l = (a, b_l)$ 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 easily computing the next sieving polynomial we can easily compute the roots of the new sieving polynomial modulo the norms of the prime ideals $\mathfrak{p}$ in the factor base. Let $\mathfrak{p}$ be an ideal in the factor base such that $\mathfrak{p} \notin Q$, so $N(\mathfrak{p}) = p \nmid a$. 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 ideals, that is for $p \mid a$, we must compute them as before, but this is not expensive, as 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.

To summarize, we compute a new sieving polynomial and roots using self-initialization as follows:

1. If $l = 0$, we have to start from the beginning:
   (a) Select j unique ideals from the factor base to form $Q = \{\mathfrak{q}_1, \ldots, \mathfrak{q}_j\}$.
   (b) Set $\vec{v} = (1, 1, \ldots, 1)$.
   (c) Compute $\mathfrak{a}_0 = \prod_{i=1}^{j} \mathfrak{q}_i = (a, b_0)$.
   (d) Compute $c_0 = \frac{b_0^2 + b_0 h + f}{a}$. This give the sieving polynomial $F(S) = aS^2 + hS + c_0$.
   (e) For each prime ideal $\mathfrak{q}_i$ in Q with $N(\mathfrak{q}_i) = q_i$:

i. Find t_i such that $t_i^2 + t_i h \equiv f \pmod{q_i}$.

ii. Compute $B_i = (a/q_i)\left((a/q_i)^{-1} t_i \pmod{q_i}\right) \pmod{a}$, $B_i' = (a/q_i)\left((a/q_i)^{-1}(t_i + h) \pmod{q_i}\right) \pmod{a}$ and $B_{a_i} \equiv (B_i + B_i')a^{-1} \pmod{q_i}$.

(f) For each prime ideal $\mathfrak{p}_i$ in the factor base with $N(\mathfrak{p}_i) = p_i$:

i. If $p_i \mid a$, $r_{i,1} \equiv c_0 h^{-1} \pmod{p_i}$ is stored as a root.

ii. Else If $p_i \mid f$ and $p_i \nmid a$ then $r_{i,1} \equiv b_0 a^{-1} \pmod{p_i}$ and $r_{i,2} \equiv (b_0 + h)a^{-1} \pmod{p_i}$ are stored as roots.

iii. Else If $p_i \nmid a$ and $p_i \nmid f$ solve $r^2 + rha^{-1} + c_0 a^{-1} \equiv 0 \pmod{p_i}$ and store the solutions $r_{i,1}$ and $r_{i,2}$ as roots.

(g) Set $l = l + 1$.

(h) Return $F(S)$, $\vec{v}$ and the set of roots. Store the B_i, B_i', B_{a_i} and l values along with $\mathfrak{a}_0$ and the set of roots for future computations.

2. Else, we compute a new polynomial using the stored information:

(a) Find k such that $2^k \parallel 2(l-1)$.

(b) Set $v_k = -v_k$.

(c) Compute $b_l = b_{l-1} + B_k + B_k'$ using the stored B_i and B_i' values. Set $\mathfrak{a}_l = (a, b_l)$.

(d) Compute $c_l = \frac{b_l^2 + b_l h + f}{a}$. This gives the sieving polynomial $F(S) = aS^2 + hS + c_l$.

(e) For each prime ideal $\mathfrak{p}_i$ in the factor base with $N(\mathfrak{p}_i) = p_i$:

i. If $p_i \mid a$, $r_{i,1} \equiv c_l h^{-1} \pmod{p_i}$ is stored as a root.

ii. Else (that is, $p_i \nmid a$), set $r_{i,n} \equiv r_{i,n} + B_{a_k} \pmod{p_i}$ for $n = 1, 2$.

(f) Set $l = l + 1 \pmod{2^{j-1}}$.

(g) Return $F(S)$, $\vec{v}$ and the set of roots. Store l, a_l and the roots for future computations.

Now sieving can use the above algorithm for finding new sieve polynomials.

There is one remaining piece of the sieving method that we have not discussed. There are a number of parameters that are required in order for these computations to run. In the next section we discuss these parameters.

4.2.4 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 our sieve radius M is. 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 we sieve, the tolerance value Y that decides whether or not we should check a candidate for smoothness and j, the number of prime ideals we use to create the ideal 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 be searching for parameters. Then we ran test programs in which several hundred relations were found using different combi-

nations of parameters. The exact number of relations we searched for depended on the instance we were considering. 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-initialized sieving is being used.

Mimicking the choice of the sieving polynomial done by Jacobson [33], we want a, the leading coefficient of $F(S)$, to have degree approximately $g - M$, the genus minus the sieve radius. This keeps the degree of $F(S)$ evaluated at polynomials in the sieve range small. In fact, this degree is at most $g + M + 1$, since

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

and we have

$$\deg(aS^2) = \deg(a) + 2\deg(S) \leq g + M,$$
$$\deg(hS) = \deg(h) + \deg(S) \leq g + M,$$

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 $\deg(b) < \deg(a) = g - M$, as is the case for reduced ideals. In fact, since $\deg(aS^2 + hS) < \deg(c)$ for all S such that $\deg(S) \leq M$, we have $\deg(F(S)) =$

$\deg(c) = g + M + 1$.

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, and we derive that 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 earlier in this chapter. Once we have chosen M and j we can use these to choose which prime ideals are used to generate the sieving polynomials, namely those ideals having norms with degree greater than or equal to $(g - M)/j$, with ideals with lower-degree norms being used first.

Using the search method described above we derived 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 ideals 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. We will only sieve with prime ideals that have norms with degree less than or equal to $t - 1$. We also do not explicitly test for repeated factors. The choice of tolerance value must account for this. We must also account for both the degrees of the polynomials we are sieving with and the degree of $F(S)$ when evaluated at polynomials in the sieve radius. This will help to ensure that most of the candidates produced really are smooth.

Now consider some $F(S)$ that is smooth over the factor base. As seen above, $\deg(F(S)) = g + M + 1$. Then $F(S) = \prod_{i=1}^{k} p_i^{a_i}$ where $p_i = N(\mathfrak{p}_i)$ for k unique prime ideals from the factor base. Then $\deg(F(S)) = \sum_{i=1}^{k} a_i \deg(p_i)$. If $a_i = 1$ for all i then using $g + M$ as a tolerance value will cause $F(S)$ to be marked a candidate as $\deg(F(S)) \geq g + M$ 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 would allow for a square factor for some p_i with $\deg p_i = M$.

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 ideals are those with norms of degree $t+1$ we should reduce the tolerance value by $t+1$. This is because an almost smooth candidate has degree that is the sum of the degrees of the norms of the ideals in the factor base plus the degree of the norm of the large prime ideal, $t + 1$ (or t, depending on the variation and value of r). Furthermore, when sieving with only the smaller degree irreducible polynomials 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 accounts for any missing degree t factors that are not marked in the sieving process.

Finally, when choosing a tolerance value one should consider that with a larger tolerance value fewer ideals will be tested for smoothness, but more sieving polynomials will have to be created as you are testing fewer candidates per polynomial. Similarly, a smaller tolerance value will result in testing more candidates for smooth-

ness per polynomial, but fewer sieving polynomials will have to be created.

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, and Y was initially $\deg(F(S)) - 2M - 2t$ for our first test case and made smaller for the later test cases.

This should provide the reader with some guidance for choosing sieving parameters. However, we acknowledge that there is more work that could be done in this area to make the choices more concrete, including a more detailed analysis of sieving, which may provide a method of selecting appropriate parameters without having to perform any tests.

In this chapter we have seen how the suggestions for improvement to the random walk method in low genus cases can be applied to the high genus cases. We have also seen how we can compute estimated number of steps required for completing the random walk part of the discrete logarithm algorithm. Additionally, we have introduced a number of possible improvements to the sieving algorithm, including describing self-initialized sieving for this setting. In the next chapter we discuss our implementation and present results found using various combinations of the above algorithms and settings.

Chapter 5

Experimental Results

In Chapter 3 we presented two algorithms used to solve the discrete logarithm problem in the ideal class group of quadratic function fields. The first algorithm we looked at is due to Enge and Gaudry [22], which makes use of the random walk idea of Teske [64]. We also presented large primes, which Thériault [65] analyzes in the case where the genus is small in relation to q. In the fourth chapter we demonstrated how large primes could be used in the larger genus case. We also presented formulas originally presented by Jacobson, Menezes and Stein [35] and Maurer, Menezes and Teske [49] for computing the number of ideals we expect to test for smoothness in order to find $F(t) + 5$ relations when using the random walk method. Finally, we generalized these formulas to take large primes into account.

The second algorithm we looked at in the third chapter was one designed by Vollmer [66]. With this algorithm we combine sieving, used in a similar context by Jacobson [32, 33]. We also introduced the idea of self-initialized sieving that Jacobson [33] used. In chapter four we generalized sieving, with self-initialization, to the even characteristic case. We also presented the idea of small degree sieving and a variation on the implementation of the sieve array. Finally, we provided some guidance for choosing sieve parameters.

We have implemented these algorithms and suggested improvements as part of a larger C++ library we call Algebraic Number Theory Library (ANTL), a library built on Shoup's NTL library [62]. In this chapter we discuss our implementation. The platform available to us for testing this implementation is also described in this chapter. Using the examples provided by Jacobson, Menezes and Stein [35] as motivation we have selected four quadratic function fields with which to test the code. These fields are motivated by the concept of Weil descent, described by Frey [25] and further studied by Gaudry, Hess and Smart [27]. Using these fields we apply the formulas for calculating the expected number of ideals checked for smoothness to determine settings to use with the Enge-Gaudry algorithm. Finally, we apply the discussion regarding sieving parameters to select settings for the sieve. Armed with all this, we present results of our implementation compared to the results found by Jacobson, Menezes and Stein [35], and compare using the Enge-Gaudry algorithm [22] with random walks as the relation generation strategy to Vollmer's method [67] using sieving as the relation generation strategy.

5.1 Testing Platform

Available for us to use at the University of Calgary is the Centre for Information Security and Cryptography (CISaC) Advanced Cryptography Lab [2]. This lab consists 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 are running Red Hat Enterprise Linux 3 and have the GNU Multi-Precision C library (GMP) version 4.2.2 [6] installed, along with NTL version 5.4.1 [62] and the MPICH Message Passing Interface (MPI) version 1.2.5 [4]. Additionally, we have installed the Automatically Tuned Linear Algebra Software (ATLAS) version 3.7.31 [1] and LinBox version 1.1.4 [5] to perform linear algebra. The compiler used was GCC version 3.4.4 [3]. This hardware and software directed the implementation of the algorithms.

5.1.1 Implementation

The algorithms implemented for this thesis exist as part of a larger plan to produce a generalized library for performing computations in quadratic number fields and quadratic function fields. This library is the above mentioned ANTL. It is implemented in C++ and is largely class based. ANTL will eventually provide the ability to perform class group computations and solve discrete logarithm problems in quadratic number fields and quadratic function fields. As such, an attempt has been made to generalize as much as possible, making extensive use of C++ templates. At the time of writing, only the implementation for imaginary quadratic function fields defined over even characteristic finite fields has been tested extensively. Source code is available upon request but, since the library is still in a very preliminary stage, support for some of the functions, especially those not relevant to this thesis, will not be guaranteed.

We now provide details of the implementation of each algorithm. Note that in

this discussion we assume that we have ideals $\mathfrak{a}$ and $\mathfrak{b} \sim \mathfrak{a}^x$ in the order $\mathcal{O}_K$ as in the third chapter. We also assume we are provided the class number N and the factor base bound t, resulting in a factor base of size $F(t)$. Our implementation also takes in the number of irreducible polynomials that we consider for our large prime ideals. Finally, we also determine sieve parameters based on the discussion in the fourth chapter.

Implementation of the Enge-Gaudry Algorithm

We first discuss the important details of our implementation of the Enge-Gaudry algorithm. In order to generate the factor base we order the polynomials in $\mathbb{F}_q[X]$ based on the evaluation of ν for each polynomial, as described in Chapter 3. For each polynomial p (taken in order) we check for irreducibility using the deterministic test in NTL. If it is indeed irreducible, we then proceed with solving $x^2 + hx - f \equiv 0 \pmod{p}$, as discussed in the third chapter. If the solution exists, we add the prime ideal $\mathfrak{p} = (p, x)$ to the factor base. We continue until we have reached either the given maximum elements in the factor base, or we have looked at all polynomials up to the given factor base bound, t. In order to ensure we always use the same prime ideal for a given polynomial p, we take whichever of x and $x + h$ results in the smaller evaluation of ν. After creating the factor base we move on to building the relation matrix. Here we use the random walks method to generate relations.

We do the relation generation phase in parallel, as discussed in the third chapter. Each process performs its own random walk and results are reported back to a designated master node that coordinates the overall algorithm. The random walk

for each processor is initialized by forming 20 random ideals, as done by Jacobson, Menezes and Stein [35] and discussed in the previous two chapters. Our hash function takes the last five bits of the representation of the constant term of the norm of the input ideal and reduces it modulo 20, also as done by Jacobson, Menezes and Stein [35]. We test ideals for smoothness using the test described in the third chapter. If we have an ideal that is almost smooth we store α, β, the large prime and the vector representing the factorization of the ideal. Rather than searching the almost smooth ideals 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 Thériault's result [65, Theorem 1] to determine when this happens. Specifically, we wait until $\frac{x^2}{6A_{t+1}}$ plus the current number of found relations is greater than $F(t)+5$, the total number of relations we wish to find, where x is the number of almost smooth ideals found and A_{t+1} is the number of degree $t+1$ irreducible polynomials, which give rise to $2A_{t+1}$ large prime ideals, ideals and their inverses, as discussed in the previous chapter. The combining of the almost smooth ideals is done by the master process at this point. Our tests show that this is a reasonable strategy to use, typically resulting in approximately 10% more relations than necessary.

Once we have enough relations we move on to the linear algebra problem. In the Enge-Gaudry algorithm we find a non-zero vector in the kernel of A. That is, we solve $A\vec{x} \equiv \vec{0} \pmod{N}$, 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 by Enge and Gaudry [22, Section 4]. 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 $A\vec{x} \equiv A\vec{v} \pmod{N}$. Then $\vec{x} - \vec{v}$ is, with high probability, a non-zero vector in the kernel of A. If a solution to the linear system $A\vec{x} \equiv A\vec{v} \pmod{N}$ does not exist we compute five more random relations and try again. Our experiments show that very few additional iterations are required, if any, before a solution is found.

Specifically, the LinBox function we use is `LinBox::solve(input matrix, solution vector, input vector, Field, Method)` where the input matrix is A, the input vector is $A\vec{v}$, the field is integers modulo N and the method is Lanczos. Additionally, we specify that the matrix is singular (`Method.singular(Specifier::SINGULAR)`), limit the number of attempts to 1 (`Method.maxTries(1)`), and ignore the ability to certify a system without a solution (`Method.certificate(false)`). This decision was made so that rather than spend time confirming that the system is not solvable we generate more relations and try again. The default preconditioner for Lanczos in LinBox is `FULL_DIAGONAL`.

Lanczos' algorithm is a probabilistic algorithm of the Monte Carlo type and is easily made a Las Vegas type of random algorithm by including a check to see that the system is actually solved and repeating if not. Because of the random nature of the algorithm, there has been some interest in the reliability of Lanczos' method. Work done by Eberly and Kaltofen [20] and Eberly [19] suggests that Lanczos' algorithm works well when the computation is done over a large field. This is the exact case

we are in, performing computations modulo large primes. There has also been work done in analyzing the reliability of the algorithm over small finite fields for finding the rank of matrices by Eberly [19], and sampling the nullspace of a matrix by Hovinen and Eberly [30]. It should be noted that this is an evolving area of research with new results being found quite frequently.

As mentioned previously, there are parallel linear algebra algorithms that could be used, such as the block Wiedemann strategy described by Kaltofen [40] or the block Lànczos variant, described by Montgomery [53]. However, implementing these algorithms is well beyond the scope of this thesis, and since no known implementation of these exists, use of them in conjunction with Enge-Gaudry will have to be left for future work. Thus, it should be stressed that this implementation was chosen primarily because it worked, and we acknowledge that there is room for improvement in this area of our work.

Once we have an appropriate vector $\vec{x}$ we compute the discrete logarithm as discussed in Chapter 3.

Implementation of Vollmer's Algorithm

In the implementation of Vollmer's algorithm we use a significantly different method of generating relations. However, before we get there we must first create our factor base. We do so in the same manner as described for the Enge-Gaudry algorithm. Now we use sieving as the method to generate relations.

Again, we perform relation generation in parallel. In the non-self-initialized case, each process is provided a sieving polynomial from the master process, and returns relations (smooth or almost smooth) to the master node. The slave processes also request new sieve polynomials from the master node. In the self-initialized case, each slave process receives a set Q of ideals used to generate the sieving polynomial from the master process. Then each slave process computes the $2^{|Q|-1}$ sieving polynomials possible from this set of ideals, after which it requests a new set of ideals from the master node. In either case, each sieve uses the same parameters, provided upon execution of the program. The sieve array is implemented as discussed in Section 4.2.1, precomputing the values of ν_0. Comparing sample runs using both possibilities, we find that this method is more efficient. Also, while both regular and self-initialized sieving are implemented, we only consider use of the self-initialized version. We use the same smoothness test as in the random walks method to factor candidates produced by the sieve, and wait until we expect to have enough intersections to complete our relation matrix before we combine the almost smooth ideals on the master node to form relations. In this case, our tests show that we end up requiring less than 10% more relations to complete the linear system in two of our test cases, and have less than 10% too many relations in the other two cases.

After creating the relation matrix we have to find the special relations for $\mathfrak{a}^{-1}$ and $\mathfrak{b}$ as discussed in the third chapter. We do this using a method similar to that mentioned in Chapter 3 for the use of sieving in the Enge-Gaudry algorithm. An ideal $\mathcal{J}$ is formed randomly for the purpose of forming a sieving polynomial as described in Chapter 3. We then multiply this ideal by the ideal we wish to factor, and sieve

to find the special relation as described in the third chapter. Again, this is done in parallel. Once we have these special relations we move on to the linear algebra stage.

In Vollmer's algorithm we have to find the solution to a linear system of equations modulo the class number N. Again we use the Lanczos system solver provided by the LinBox library, with the same settings as before, for the same reasoning as in the Enge-Gaudry case above. We use the function `LinBox::solve(input matrix, solution vector, input vector, Field, Method)` where the input matrix is A' from Section 3.1.2, the input vector is $(1, 0, \cdots, 0)^T$, the field is integers modulo N and the method is Lanczos. Again, we specify that the matrix is singular (`Method.singular(Specifier::SINGULAR)`), limit the number of attempts to 1 (`Method.maxTries(1)`), and ignore the ability to certify a system without a solution (`Method.certificate(false)`). Numerous trials with the test data done during the debugging process provided evidence to support not trying multiple times, but rather stopping and finding additional relations. If a solution does not exist then we generate five more relations and recompute the special relations for $\mathfrak{a}^{-1}$ and $\mathfrak{b}$. Our experiments suggest again that very few, if any, iterations of this process are necessary. Once we have a solution to the linear system the solution to the discrete logarithm problem is computed and returned.

5.1.2 Test Data

Jacobson, Menezes and Stein [35] study the Weil descent attack on elliptic curve discrete logarithm problems of Frey [25]. In particular, they focus on the work done by Gaudry, Hess and Smart (GHS) [27]. GHS provide an algorithm that extends

Frey's work. Their algorithm reduces an instance of the ECDLP over a characteristic two finite field $\mathbb{F}_{q^n}$, where the number of points on the elliptic curve is a "small" number (2 or 4) times a prime, to an instance of the discrete logarithm problem in the Jacobian of a hyperelliptic curve over $\mathbb{F}_q$ with genus at most 2^{n-1}. As we have shown, there are subexponential algorithms for solving this second problem when the genus is large relative to q. Exactly how this works is left to the provided references as it is not the focus of this thesis.

Jacobson, Menezes and Stein [35] choose four instances of the ECDLP to apply GHS to and from this obtain four instances of the HCDLP. They call the resulting hyperelliptic curves C62, C93, C124 and C155 since the original elliptic curves are defined over $\mathbb{F}_{2^{62}}$, $\mathbb{F}_{2^{93}}$, $\mathbb{F}_{2^{124}}$ and $\mathbb{F}_{2^{155}}$, respectively. C155 is of particular interest as the Internet Engineering Task Force recommended use of a specific curve over $\mathbb{F}_{2^{155}}$ for a key establishment algorithm [31]. These curves are presented in Table 5.1, where each curve is defined by $v^2 + h(u)v = f(u)$ and defined over $\mathbb{F}_{2^2}$, $\mathbb{F}_{2^3}$, $\mathbb{F}_{2^4}$, and $\mathbb{F}_{2^5}$, respectively. This table also contains N, the prime factor portion of the number of points in the elliptic curve. In fact, the number of points on the elliptic curve is $2N$, and is equal to the number of elements in the Jacobian associated with each curve. All these curves have genus 31. As we saw in Chapter 2, these curves relate directly to quadratic function fields, and thus we can apply our techniques here. For our tests we use the same predetermined ideals $\mathfrak{a}$ and $\mathfrak{b}$ that Jacobson, Menezes and Stein [35] use in order to be able to compare both methods on the same inputs. These values are in Tables 5.2 and 5.3.

C62: $q = 4$, $\mathbb{F}_q = \mathbb{F}_2[w]/(w^2 + w + 1)$ $f(u) = u^{63} + w^2u^{62} + u^{48} + w^2$ $h(u) = u^{31} + u^{30} + wu^{28} + u^{24} + w^2u^{16} + w^2$ $N = 2305843007560748609$
C93: $q = 8$, $\mathbb{F}_q = \mathbb{F}_2[w]/(w^3 + w + 1)$ $f(u) = w^4u^{63} + w^5u^{62} + w^5u^{60}w^3u^{56} + w^5u^{48} + wu^{32} + w^5$ $h(u) = w^2u^{31} + w^5u^{30} + u^{28} + w^6u^{24} + w^6$ $N = 4951760157141611728579495009$
C124: $q = 16$, $\mathbb{F}_q = \mathbb{F}_2[w]/(w^4 + w + 1)$ $f(u) = w^6u^{63} + w^{14}u^{60} + w^6u^{56} + w^6u^{48} + 1$ $h(u) = w^3u^{31} + w^9u^{30} + wu^{28} + w^{11}u^{24} + w^{12}u^{16} + w^{12}$ $N = 10633823966279326985483775888689817121$
C155: $q = 32$, $\mathbb{F}_q = \mathbb{F}_2[w]/(w^5 + w^2 + 1)$ $f(u) = w^4u^{63} + w^6u^{62} + w^{15}u^{60} + w^{26}u^{56} + w^{25}u^{48} + w^7u^{32} + w^{13}$ $h(u) = w^2u^{31} + w^7u^{30} + w^{30}u^{28} + w^{22}u^{24} + w^3u^{16} + w^{22}$ $N = 22835963083295358096932727763065266972881541089$

Table 5.1: Hyperelliptic Curves C62, C93, C124 and C155

C62:

$\mathfrak{a} = (1 + u^2,\ w + wu)$

$\mathfrak{b} = \mathfrak{a}^x = (w + u + w^2u^2 + wu^4 + wu^5 + wu^6 + w^2u^7 + u^8 + wu^9 + wu^{10}$
$+u^{12} + w^2u^{13} + w^2u^{14} + w^2u^{15} + wu^{16} + u^{17} + u^{18} + wu^{19} + w^2u^{20} + u^{21}$
$+u^{24} + 1u^{26} + wu^{27} + w^2u^{28} + u^{29},$
$w + u + u^2 + wu^5 + u^6 + w^2u^9 + u^{10} + w^2u^{12} + u^{13} + u^{14} + w^2u^{21}$
$+u^{22} + u^{24} + u^{25} + w^2u^{26} + w^2u^{27} + wu^{28})$

$x = 123456789$

C93:

$\mathfrak{a} = (1 + wu + w^3u^2 + w^3u^3 + u^4 + w^2u^6 + w^4u^7 + w^2u^8 + w^2u^{10} + w^5u^{11}$
$+w^6u^{12} + u^{13} + wu^{15} + w^5u^{16} + w^5u^{17} + w^5u^{18} + w^4u^{19} + u^{20} + wu^{21}$
$+wu^{22} + w^3u^{24} + w^3u^{25} + w^4u^{26} + w^4u^{28} + u^{29} + w^2u^{30} + u^{31},$
$wu + w^5u + w^6u^2 + w^5u^3 + w^6u^4 + w^3u^5 + wu^6 + w^4u^7 + w^6u^8 + w^6u^{11}$
$+w^2u^{12} + w^2u^{13} + wu^{14} + w^4u^{15} + u^{16} + w^4u^{17} + w^2u^{18} + wu^{19} + u^{20}$
$+w^3u^{21} + u^{22} + w^4u^{23} + w^5u^{24} + u^{25} + w^4u^{27} + w^2u^{30})$

$\mathfrak{b} = \mathfrak{a}^x = (w^4 + w^6u + w^2u^2 + w^3u^3 + w^3u^4 + u^5 + u^6 + w^3u^7 + w^2u^8$
$+w^4u^9 + w^4u^{10} + w^4u^{11} + w^3u^{12} + w^3u^{14} + w^2u^{15} + w^2u^{17} + w^2u^{18}$
$+wu^{19} + w^4u^{20} + w^6u^{21} + w^4u^{22} + w^2u^{23} + w^6u^{24} + w^3u^{27} + w^3u^{28}$
$+w^3u^{29} + u^{31},$
$w + w^2u + w^5u^2 + wu^4 + wu^5 + w^5u^6 + w^6u^7 + w^6u^8 + w^2u^9 + w^4u^{10}$
$+w^5u^{11} + w^6u^{12} + w^3u^{13} + w^4u^{14} + u^{15} + w^5u^{16} + w^5u^{17} + w^3u^{18}$
$+w^3u^{20} + w^2u^{21} + w^6u^{22} + w^2u^{24} + w^2u^{25} + w^2u^{26} + w^2u^{27} + wu^{29} + w^3u^{30})$

$x = 12345678901234567890$

Table 5.2: Test Discrete Logarithm Problems over C62 and C93

C124:

$\mathfrak{a} = (w^{11} + w^4u + wu^3 + w^9u^4 + w^8u^5 + w^{11}u^6 + w^8u^7 + w^9u^8 + w^4u^9 + w^2u^{10}$
$+w^{11}u^{11} + w^{14}u^{12} + w^{10}u^{13} + w^5u^{14} + w^{11}u^{17} + w^7u^{18} + u^{19} + wu^{20} + w^2u^{21}$
$+w^7u^{23} + w^3u^{24} + w^{12}u^{25} + w^3u^{26} + w^4u^{27} + w^{10}u^{30} + u^{31},$
$w^7 + wu^2 + w^7u^3 + w^3u^4 + w^{12}u^5 + w^{12}u^6 + w^{13}u^7 + w^{12}u^8 + w^3u^9 + w^3u^{10}$
$+u^{11} + w^6u^{12} + u^{13} + w^{14}u^{14} + w^4u^{15} + w^9u^{16} + w^{10}u^{17} + w^{14}u^{19} + w^{11}u^{20}$
$+w^4u^{21} + w^6u^{22} + w^{13}u^{23} + w^4u^{24} + w^{11}u^{26} + w^{13}u^{27} + wu^{28} + w^{13}u^{29} + u^{30})$
$\mathfrak{b} = \mathfrak{a}^x = (w^{12} + u^1 + w^7u^2 + w^{11}u^3 + w^6u^4 + w^2u^5 + w^5u^6 + w^4u^7 + wu^8$
$+w^2u^9 + w^8u^{10} + w^6u^{11} + w^{12}u^{12} + w^{12}u^{13} + w^5u^{14} + w^4u^{15} + w^{12}u^{16} + wu^{17}$
$+w^6u^{18} + wu^{19} + w^{10}u^{20} + w^6u^{22} + w^{10}u^{23} + w^5u^{24} + w^2u^{25} + w^2u^{26} + w^6u^{27}$
$+u^{28} + w^3u^{29} + w^2u^{30} + u^{31},$
$w^3 + w^8u + w^8u^2 + w^8u^3 + w^2u^4 + w^{10}u^6 + w^{12}u^7 + w^4u^8 + w^{13}u^9 + w^{10}u^{10}$
$+w^{10}u^{11} + w^9u^{12} + wu^{13} + w^{13}u^{14} + u^{15} + w^2u^{16} + u^{18} + w^{12}u^{19} + w^8u^{20}$
$+wu^{21} + w^8u^{22} + w^8u^{23} + w^{11}u^{24} + wu^{25} + w^8u^{26} + w^{10}u^{27} + w^9u^{28} + w^2u^{29}$
$+w^{14}u^{30})$
$x = 289697194482016303350776099807354482$

C155:

$\mathfrak{a} = (w^{11} + w^{10}u + w^{21}u^2 + w^3u^3 + w^{28}u^4 + w^{12}u^5 + w^5u^6 + w^{29}u^7 + u^8$
$+w^4u^9 + w^{17}u^{10} + w^9u^{11} + w^5u^{12} + w^2u^{13} + w^{29}u^{14} + w^{11}u^{15} + w^{28}u^{16}$
$+w^{30}u^{17} + wu^{18} + w^3u^{19} + w^{13}u^{20} + w^{10}u^{21} + w^7u^{22} + w^{20}u^{23} + w^{21}u^{24}$
$+w^{23}u^{25} + w^3u^{26} + w^{30}u^{27} + w^{10}u^{28} + w^{17}u^{29} + w^{19}u^{30} + u^{31},$
$w^{29} + w^{21}u + w^{27}u^2 + w^{21}u^3 + w^{10}u^4 + w^{12}u^5 + w^{27}u^6 + w^{23}u^7 + w^4u^8$
$+w^{26}u^9 + w^{12}u^{10} + w^{18}u^{11} + w^{10}u^{12} + w^{16}u^{14} + w^{10}u^{15} + w^{24}u^{16}$
$+w^{12}u^{17} + w^{25}u^{18} + w^9u^{19} + w^{28}u^{20} + w^{22}u^{21} + w^{19}u^{22} + w^{16}u^{23}$
$+w^{20}u^{24} + w^7u^{25} + w^6u^{26} + w^9u^{27} + w^4u^{28} + w^4u^{29} + w^{20}u^{30})$
$\mathfrak{b} = \mathfrak{a}^x = (w^{14} + w^{10}u + w^{29}u^2 + w^{27}u^4 + w^{22}u^5 + w^{29}u^6 + w^9u^7 + u^8 + w^2u^9$
$+w^9u^{10} + w^{11}u^{11} + w^{28}u^{12} + w^{23}u^{13} + w^{13}u^{14} + w^9u^{15} + w^{11}u^{16} + w^{28}u^{17}$
$+w^4u^{18} + w^8u^{19} + w^{17}u^{20} + w^{15}u^{21} + w^{19}u^{22} + w^{16}u^{23} + w^{12}u^{24} + w^4u^{25}$
$+w^7u^{26} + wu^{27} + w^{17}u^{28} + w^{10}u^{29} + w^{14}u^{30} + u^{31},$
$w^{20} + w^{24}u + wu^3 + w^6u^4 + w^3u^5 + w^6u^6 + w^{12}u^7 + w^{21}u^8 + w^{17}u^9$
$+w^3u^{10} + w^5u^{11} + w^{24}u^{12} + w^{30}u^{13} + w^4u^{14} + w^4u^{15} + w^{29}u^{16} + w^{17}u^{17}$
$+w^{30}u^{18} + w^6u^{19} + w^{23}u^{20} + w^{12}u^{21} + w^{17}u^{22} + w^{19}u^{23} + w^{22}u^{24} + w^{15}u^{25}$
$+w^7u^{26} + w^9u^{27} + w^{16}u^{28} + w^{13}u^{29} + w^{19}u^{30})$
$x = 204240218234519186099803027514725655587535093 70$

Table 5.3: Test Discrete Logarithm Problems over C124 and C155

5.2 Empirical Estimates

In Chapter 4 we presented formulas presented by Jacobson, Menezes and Stein [35] that provide an expected number of random ideals that need to be tested in order for a total of $F(t)+5$ relations to be found. We also presented formulas that incorporate the idea of almost smooth ideals. Now that we have presented the curves in which we solve the discrete logarithm problem we can apply these formulas and, using the results, we can justify the parameters we used for the Enge-Gaudry algorithm using random walks for relation generation.

After evaluating these formulas we estimate the amount of time required to perform the entire algorithm. We do this by measuring the time required to create a smaller number of ideals, for example 1000 ideals for the C62, and test them for smoothness for selected parameter sets (for example, for each curve, for each t considered, run with $r = 0.25$, $r = 0.50$, $r = 0.75$ and $r = 1$). We then use the data to compute line of best fit and use this line to estimate the time required to compute relations for any r value. Similarly, we have estimated the linear algebra times by running the entire discrete logarithm problem solving algorithm for several values of r for each value of t considered, extracting the linear algebra time, computing a quadratic curve of best fit and using this to estimate the linear algebra time. Note that for C155, since the only result we computed is our final one, we used the formula for the C124 linear algebra times as a basis, and scaled up slightly. However, as our computations show, this scaling was nowhere near sufficient. It should also be noted that if the linear algebra algorithm changes, one would have to recompute

these estimates. In particular if the change was made to an parallel linear algebra algorithm. Finally, time required for initialization (that is, creating the factor base) has been measured in the same manner 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 ideals are considered.

When we do not consider the almost smooth ideals we have 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 ideals for our factor base. We consider t in the range $[1, 9]$ for C62 and $[1, 7]$ for C93, C124 and C155, following the lead from Jacobson, Menezes and Stein [35]. Also, we perform the computations for $0 < r \leq 1$ in steps of 0.01. When we do consider the almost smooth ideals we consider the case where we use just the prime ideals having norm with degree equal to t as large prime ideals and the case where we also include the prime ideals having norms with degree equal to $t + 1$.

Tables 5.4 to 5.7 provide some interesting data points for each variation of the random walk parameters and each curve we are considering. These points include the estimations for the recommended settings from the work by Jacobson, Menezes and Stein [35], marked with $*$. We call these the JMS settings for the remainder of the thesis. We have also included our best settings for the case in which we do not consider the large prime ideals and the case in which we use the degree $t+1$ norms.

t	r	$F(t)$	LP	$E(t)$	$E_{LP}(t)$	$T(t)$	T	LA	total
Not considering almost smooth ideals									
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 5.4: Estimated C62 runtimes in seconds

Finally, the remaining estimates are shown to provide some context and give an idea as to how changing the parameters can result in a change in the expected runtimes. In these tables $F(t)$ is the size of the factor base, LP indicates the number of large prime ideals that result from these settings, $E(t)$ is the expected number of steps required to find a smooth ideal, $E_{LP}(t)$ is the expected number of tests to find an almost smooth ideal, $T(t)$ is the total number of steps expected to find $F(t)+5$ relations, T represents the estimated time required to test $T(t)$ ideals using 256 processors. We are performing the relation generation stage in parallel and as such we present that estimated time. The column LA refers to the estimated time required for the linear algebra stage. Again, note that the linear algebra time could

t	r	$F(t)$	LP	$E(t)$	$E_{LP}(t)$	$T(t)$	T	LA	total
Not considering almost smooth ideals									
4	1.0	596	-	1830509	-	1.1×10^9	178.3	0.03	178.5
5	0.5	2234	-	145591	-	325977380	56.3	0.45	56.9
$*$ 5	1.0	3872	-	28668	-	111146195	19.3	1.35	20.8
6	0.5	14771	-	6281	-	92810648	17.12	19.6	37.4
Using only degree t norms									
4	0.9	546	50	2834369	6261994	1.0×10^9	168.4	0.03	168.6
5	0.5	2234	1638	145591	75143	171550333	29.5	0.51	30.1
5	0.8	3217	655	51489	82729	115159023	20.1	1.06	21.3
5	0.9	3544	328	38100	130084	108360217	18.9	1.28	20.4
6	0.1	6052	19618	20016	6513	75517246	13.7	3.74	17.7
Using degree $t+1$ norms									
4	1.0	596	3276	1830509	266121	484073278	83.4	0.04	83.5
5	0.5	2234	23436	145591	19132	167421504	30.6	0.51	31.2
5	1.0	3872	21798	28668	7577	71792103	13.0	1.53	14.7
6	0.5	14771	160695	6281	1394	64731785	13.7	22.29	36.7

Table 5.5: Estimated C93 runtimes in minutes

be quite different if a parallel algorithm was used, and this could change the choice of optimal parameters. The total time is the sum of LA, T and the time required for initialization. Initialization time is not shown because it is generally small compared to the overall runtime and has little effect on these estimates, particularly for C93, C124 and C155.

There are a few important things about the estimates in Tables 5.4 to 5.7 that should be pointed out. First, one might be puzzled by the entries in which $E_{LP}(t) > E(t)$. However, recall that $E_{LP}(t)$ depends on the number of almost smooth ideals that result from the settings being used, and this in turn depends on the number of large prime ideals that arise from the given settings. When the number of large prime

t	r	$F(t)$	LP	$E(t)$	$E_{LP}(t)$	$T(t)$	T	LA	total
Not considering almost smooth ideals									
4	0.50	4808	-	18991245	-	9.1×10^{10}	267.3	0.07	267.4
4	1.00	8872	-	1498799	-	1.3×10^{10}	38.9	0.24	39.1
5	0.32	42426	-	267103	-	1.1×10^{10}	36.4	5.59	42.0
5	0.50	61300	-	127546	-	7.8×10^{9}	25.4	11.67	37.1
$*$ 5	1.00	113728	-	25876	-	2.9×10^{9}	10.3	40.15	50.6
Using only degree t norms									
4	0.50	4808	4064	18991245	6040905	3.6×10^{10}	104.4	0.08	104.5
4	0.90	8059	813	2308631	5163911	1.3×10^{10}	37.4	0.23	37.6
5	0.32	42426	71302	267103	88111	5.8×10^{9}	18.8	6.34	25.2
5	0.50	61300	52428	127546	67239	4.4×10^{9}	14.3	13.24	27.6
Using degree $t+1$ norms									
4	0.50	4808	108920	18991245	1336140	4.0×10^{10}	124.1	0.08	124.2
4	1.00	8872	104856	1498799	225558	7.7×10^{9}	24.0	0.28	24.2
5	0.32	42426	1469042	267103	27402	7.2×10^{9}	24.7	6.34	31.0
5	0.50	61300	1450168	127546	17154	5.2×10^{9}	17.8	13.24	31.1
5	1.00	113728	1397740	25876	6964	2.3×10^{9}	9.4	45.57	55.0

Table 5.6: Estimated C124 runtimes in hours

ideals is low, there are fewer almost smooth ideals, and so we would expect it to take longer to find one.

Another observation to make is that the values of t suggested by Jacobson, Menezes and Stein [35] may not necessarily be the best ones. For example, compare $t = 6$ and $t = 7$ for C62. In [35] it is suggested that $t = 7$ be used, but our estimates suggest using $t = 6$ requires almost twice as much time to compute the relations, and only one tenth the time to perform the linear algebra computation. Overall this means that $t = 6$ should result in a faster computation. Also consider $t = 4$ and $t = 5$ for C124. Again, in [35] they suggest using $t = 5$, whereas our estimates show

t	r	$F(t)$	LP	$E(t)$	$E_{LP}(t)$	$T(t)$	T	LA	total
Not considering almost smooth ideals									
4	0.80	110365	-	3358791	-	3.7×10^{11}	60.8	1.61	62.4
$*$ 4	1.00	136528	-	1378374	-	1.9×10^{11}	33.2	2.47	35.7
5	0.20	807616	-	440408	-	3.6×10^{11}	114.6	86.36	201.1
Using only degree t norms									
4	0.80	110365	26163	3358791	3510031	2.1×10^{11}	35.2	1.83	37.0
4	0.92	126063	10465	1939095	5529341	1.8×10^{11}	30.7	2.39	33.1
5	0.20	807616	2684352	440408	109868	1.9×10^{11}	64.9	98.02	163.0
Using degree $t+1$ norms									
4	0.80	110365	3381603	3358791	382768	2.4×10^{11}	41.6	1.83	43.4
4	1.00	136528	3355440	1378374	210311	1.3×10^{11}	24.8	2.80	27.6
5	0.20	807616	92160024	440408	37543	2.7×10^{11}	99.4	98.02	197.5

Table 5.7: Estimated C155 runtimes in days

that using $t = 4$ results in an increase in relation generation time that is countered by a significantly larger decrease in the linear algebra time required, resulting again in a faster overall computation. Also important is the effect of using large primes. For example, compare $t = 4$, $r = 1$ in the first and third cases of C124. Use of large primes here results in a reduction in the number of expected steps by almost half. Assuming these calculations are accurate, we expect to see improvements by using large primes.

Using Tables 5.4 to 5.7 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 ideals to include those ideals 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 5.8 for a summary of what we use for input for random walks. The large prime bound is the degree of the norms of ideals that we use in our set of large prime ideals. We also list in Table 5.9 the JMS settings from [35] for reference. Finally, Table 5.10 lists sample input for the case when $r \neq 1$ that we will run in order to help demonstrate that our estimates are correct for that case as well.

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 5.8: Random Walk Parameters 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

Table 5.9: JMS Random Walk Parameters for the Enge-Gaudry Algorithm

We use the settings in Table 5.8 as a starting point for selecting sieve parameters, as we mentioned in Chapter 4. Recall that j is the number of prime ideals used to create the ideal that gives rise to the sieving polynomial, M is the sieve radius, and Y is the tolerance value, which controls what we test for smoothness. In our implementation we take the input Y and subtract it from the degree of $F(S)$, where $\deg(S) = M$.

Curve	C62	C93	C124
Factor Base Bound t	6	5	5
Parameter r	0.80	0.80	0.50
Large Prime Bound	6	5	4
Estimated Time	16.03 Seconds	21.31 Minutes	27.56 Hours

Table 5.10: Sample $r \neq 1$ Random Walk Parameters for Enge-Gaudry Algorithm

So the tolerance value used by our sieve is $\max(\deg(F(S))) - Y = g + M + 1 - Y$. We use B to denote the degree bound on the elements with which we sieve.

We use $t = 6$, $t = 5$, $t = 4$, and $t = 4$ for C62, C93, and C124 and C155, respectively. We also set $r = 1$ and use large primes having norms with degree equal to $t + 1$ in order to keep these settings close to the random walk choices. We now describe how we obtained the remainder of these parameters for C62. The other cases are derived in a similar manner.

With $t = 6$ we want $j \geq \frac{31-M}{6}$. If $M = 1$, we need $j \geq 5$, and as M grows, j can shrink. However, since the number of polynomials tested for smoothness by the sieve grows as M grows, we do not want it to grow too large. Therefore we performed the test mentioned in Chapter 4 with j ranging from 3 to 6 and M ranging from the lower bound provided by $g - tj \leq M$ to $M = 14$ when $j = 3$, $M = 10$ when $j = 4$ and $M = 6$ when $j = 5$ and $j = 6$. In addition, for each (j, M)-pair we tested using $B = 5$ and $B = 6$ for a sieve bound, recalling that we wanted to test both sieving with all polynomials arising from elements in the factor base and sieving with only those polynomials with degree less than t. We also set $Y = 17$ through

$Y = 21$ as tolerance values, an interval centered around $Y = 19$, a value that was fortunately chosen during an earlier stage of testing that seemed to be resulting in relations being found at a reasonable rate.

For C93, C124 and C155, we adjusted the tolerance value interval based on test results and the work presented in Chapter 4 in order to try and find the minimum computation time. Recall that our test had each set of parameters be used to find a small number of relations. Specifically, 200 for C62 and C93, 50 for C124 and 10 for C155. These values were chosen to ensure that our trials completed in a reasonable amount of time, but still allowed us to estimate how long the entire computation should take. The group of parameters that did this the fastest for each of C62, C93, C124 and C155 are presented in Table 5.11.

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
Ideals used j	5	7	9	9
Radius Degree M	4	3	2	2
Tolerance Value Y	19	14	11	11
Sieve bound B	5	4	3	3

Table 5.11: Sieve Parameters for Vollmer's Algorithm

5.3 Results

We have used our implementation to solve discrete logarithm problems in C62, C93 and C124 using both the Enge-Gaudry algorithm and Vollmer's algorithm, and in C155 using Vollmer's algorithm. We now present our results.

Tables 5.12 to 5.18 outline the results for C62, C93, C124, and C155, respectively. 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 total time required for the search for smooth relations that dominated the computation time. This is what we expect based on our estimates.

For the C62 case, results in Table 5.12, we notice that significantly fewer ideals were tested than we expected in both the JMS and optimal Enge-Gaudry cases. One possible cause of this is if a large number of smooth or almost smooth ideals 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. Test runs on both 10 processors and 1 processor resulted in the number of ideals tested to be much closer to the estimates. This situation also occurs in the $r \neq 1$ case. We also point out that in our optimal case we expected $T(t)/E_{LP}(t) = 809562/1051 = 770$ almost smooth ideals to be found. This is almost exactly what we did find. Comparing our optimal results to the results produced from the JMS settings we see that the search times

	JMS EG Estimate	JMS EG Result	Opt. EG Estimate	Opt. EG Result	Vollmer 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
Partial Relations	-	-	770	780	986
Intersections	-	-	169	198	180
Unique Intersections	-	-	169	198	180
Sieve Polynomials	-	-	-	-	618361
Total Ideals 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 Ideals 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

Table 5.12: C62 Results

	Estimate	Result
Initialization Time	2.98s	3.12s
Total Relations	413	413
Full Relations	324	335
Partial Relations	130	144
Intersections	89	80
Unique Intersections	89	79
Total Ideals Tested	1249536	744194
Total Search Time	52.78m	30m 33.37s
Real Search Time	12.37s	7.16s
Linear Algebra Time	0.68s	2.42s
Total Time	52.99m	30m 39.13s
Real Total Time	16.03s	12.92s

Table 5.13: C62 $r \neq 1$ Results

are very close to the same, but due to the larger factor base size recommended by JMS, the linear algebra time is significantly larger. Finally, comparing this to the results from Vollmer's algorithm, we have a faster search for relations, but due to performing the linear algebra three times, this benefit is lost when we perform the search in parallel. When increasing the number of relations we are searching for to $F(t) + 15$, we see the number of linear algebra iterations required reduced to one, but the real time is still one to two seconds slower than that from our optimal Enge-Gaudry settings, due to the increased overhead involved with sieving. One final thing to point out is how many ideals were tested for smoothness in the same time using sieving. The sieve covered much more ground in the same amount of time.

Moving now to the C93 results in Table 5.14 we see that the number of ideals tested

	JMS EG Estimate	JMS EG Result	Opt. EG Estimate	Opt. EG Result	Vollmer 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
Partial Relations	-	-	9475	9546	13246
Intersections	-	-	1373	1834	5271
Unique Intersections	-	-	1373	1834	1496
Sieve Polynomials	-	-	-	-	8889395
Total Ideals 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 Ideals 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

Table 5.14: C93 Results

	Estimate	Result
Initialization Time	8.78s	9.01s
Total Relations	3222	3222
Full Relations	1830	2376
Partial Relations	1392	1536
Intersections	986	880
Unique Intersections	986	879
Total Ideals Tested	115159023	118168104
Total Search Time	3.57d	3d 12h 25m 50.27s
Real Search Time	20.11m	10m 47.30s
Linear Algebra Time	1.06m	3m 59.95s
Total Time	3.58d	3d 12h 30m 1.37s
Real Total Time	21.31m	23m 58.40s

Table 5.15: C93 $r \neq 1$ Results

in both the JMS and optimal results are much closer to that which we expected, as well as our $r \neq 1$ test. This time both JMS and our settings recommended using factor bases of the same size. Here we see that using large primes does reduce the amount of time needed in the search. We also note that for our settings the linear algebra time is lower than the linear algebra time for the JMS settings. This is most likely related to what processor the computation was taking place on, as subsequent test runs show that the actual time required for the linear algebra stage can vary. We would expect the tests without large primes to have better linear algebra times as Lanczos' algorithm performs better on lower density matrices, and using large primes results in more non-zero entries in the matrix. Again when comparing the results to those from Vollmer's method we see that while the search time for sieving is significantly lower, taking around a quarter 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 is not parallelized in our implementation. We had hoped that adjusting the algorithm to produce the extra relations results in a reduction on the number of linear algebra iterations, which would result in Vollmer's method being significantly better here. However, when we performed this test we found that computing $F(t) + 15$ or $F(t) + 20$ relations did not change the situation, and we were still required to repeat the linear algebra computation multiple times. In fact, the computation usually required four or five linear algebra iterations. We did have one trial where linear algebra was not repeated, and in this case the real runtime was 11 minutes, which supports our theory that we would have a faster running algorithm if not for the issue with the linear algebra stage. We also note that while there were a significant number of almost smooth ideals and intersections in Vollmer's 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.

Looking at the results for C124 in Table 5.16, we can see our results are quite close to the expected values. Our settings result in a search that takes significantly longer than the search from JMS. However we note that due to the larger matrix from the JMS settings we expect the linear algebra to take significantly longer in that case. Since the difference in the search time is spread out over a parallel system but the linear algebra is not, our settings should still be better, and increasing the number of processors would only improve that situation. However the most exciting result that we see here is that from Vollmer's algorithm. Here the search time is less than the JMS search time, and additionally, while the linear algebra is worse

	JMS EG Estimate	JMS EG Result	Opt. EG Estimate	Opt. EG Result	Vollmer Result
Initialization Time	5.56m	5m 44.75s	17.64s	15.59s	10.44s
Total Relations	113733	113733	8879	8879	8879
Full Relations	113733	113733	5154	5176	5084
Partial Relations	-	-	34232	34106	34520
Intersections	-	-	3725	4953	4510
Unique Intersections	-	-	3725	4953	4510
Sieve Polynomials	-	-	-	-	1.58×10^{9}
Total Ideals Tested	2942900860	2931742632	7721296965	7557879515	6.48×10^{14}
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 Ideals Checked	-	-	-	-	143921152
Special Rels Time	-	-	-	-	3m 34.67s
Linear Algebra Time	40.15h	3d 11h 10m 14.40s	0.28h	37m 43.56s	1h 44.90s
Total Time	112d	118d 23h 50m 31.65s	255.60d	244d 21m 2.53s	81d 8h 3m 47.69s
Real Total Time	50.59h	3d 22h 5m 22.89s	24.24h	23h 30m 43.26s	8h 42m 3.49s

Table 5.16: C124 Results

	Estimate	Result
Initialization Time	2.49m	2m 32.74s
Total Relations	61305	61305
Full Relations	34335	34157
Partial Relations	65130	65340
Intersections	26970	28004
Unique Intersections	26970	28004
Total Ideals Tested	4379281805	4356289431
Total Search Time	152.31d	146d 16h 48m 56.69s
Real Search Time	14.28h	13h 45m 11.47s
Linear Algebra Time	13.24h	1d 9h 31m 16.33s
Total Time	152.87d	148d 2h 10m 13.03s
Real Total Time	27.56h	1d 23h 6m 27.81s

Table 5.17: C124 $r \neq 1$ Results

than that in our optimal Enge-Gaudry case, it is significantly better than the JMS settings, giving us a very significant improvement over the Enge-Gaudry algorithm. Table 5.17 shows that our estimate is very close to what we experienced, as far as the search time required is concerned. However, just as in Table 5.16, our linear algebra estimate is off.

Finally we examine the results for C155 in Table 5.18. 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

	JMS EG Estimate	Opt. EG Estimate	Vollmer 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 Ideals 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 Ideals 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

Table 5.18: C155 Results

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 Vollmer's 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 Vollmer's 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 particular, we have shown that with sieving, our modifications to the sieving algorithm, and the use of large primes, we have a more efficient algorithm for finding the relations necessary for performing discrete logarithm computations in the ideal class groups of high genus imaginary quadratic function fields with even characteristic that result from the use of Weil descent on elliptic curve discrete logarithm problems as presented by Jacobson, Menezes and Stein [35]. Again, we stress that the purpose of this thesis was improving the relation generation stage. There is no reason to believe that we would not see the same sort of benefit from the use of sieving if we replaced the linear algebra algorithm used above with a parallel version. However, the optimal settings would most likely be changed in order to reflect the shorter linear algebra stage.

Chapter 6

Conclusion

In this thesis we studied a specific context of the discrete logarithm problem: instances in the ideal class group of an imaginary quadratic function field defined over an even characteristic finite field. Recall that this is equivalent to solving the discrete logarithm problem in the Jacobian of a hyperelliptic curve, also called the hyperelliptic curve discrete logarithm problem (HCDLP).

We have presented an implementation of Vollmer's algorithm [66] that uses self-initialized sieving as the relation finding mechanism for this sort of discrete logarithm problem. This is not the first time sieving has been used to solve discrete logarithm problems. For example, see the work done by Jacobson [33]. This is, however, the first time it has been used to solve the discrete logarithm problem in the described context. The use of sieving was motivated by the work done by Flassenberg and Paulus [24] on sieving in quadratic function fields, and using self-initialization was motivated by the work done by Jacobson [33] in quadratic number fields.

Our implementation also makes use of large primes, similar to their use in factoring algorithms by Lenstra and Manasse [48], Boender and te Riele [10] and Kurowski [45]. The idea was also adapted by Jacobson [33] and used in conjunction with sieving to generate relations in his algorithm for computing the class number and

class group of quadratic number fields.

We have also presented a variation of the Enge-Gaudry algorithm [22] that makes use of large primes. Here the description of large primes given by Thériault [65] in his analysis of a variation of Gaudry's algorithm [26] that uses large primes was generalized and used to extend the algorithm presented by Jacobson, Menezes and Stein [35] for solving the HCDLP over the hyperelliptic curves arising from Weil descent.

By generalizing the formulas from Jacobson, Menezes and Stein [35] that can be used for computing the expected number of random walk steps required to find sufficient relations for the Enge-Gaudry algorithm to successfully complete and solve the discrete logarithm problem to take into account large primes, we find that using large primes in this algorithm should result in an algorithm that requires less time to solve the discrete logarithm problems that are presented by Jacobson, Menezes and Stein [35]. Our results support this, particularly for the curve known as C93, where our settings and those provided by Jacobson, Menezes and Stein [35] both suggest using the same size factor base. However, for C62 and C124, our suggested parameters also result in a lower computation time than the parameters given by Jacobson, Menezes and Stein [35]. Based on our results in the other three cases, there is no reason to believe our estimates for the C155 example are not accurate.

Even more exciting are the results of our implementation of Vollmer's algorithm. In all four examples, using sieving resulted in a search time that was less than that

required for the random walk algorithms. This resulted in a total runtime that was lower than the runtime of our implementation of the Enge-Gaudry algorithm in all three examples in which we tested new Enge-Gaudry parameters for a variation of the algorithm that uses large primes. Also, for C124, the real time required by Vollmer's algorithm was one third the time required by Enge-Gaudry's algorithm using our parameters, which in turn was almost four times faster than the execution of Enge-Gaudry's algorithm using the parameters suggested Jacobson, Menezes and Stein [35]. Finally, the use of sieving was used to solve the discrete logarithm over the curve known as C155, a genus 31 curve defined over $\mathbb{F}_{32}$, for the first time. This curve is the result of the Weil descent process on an elliptic curve defined over $\mathbb{F}_{2^{155}}$. In this example, the search time was over three times faster than what we estimate would be required for our optimal Enge-Gaudry parameters, and over four times faster than what we estimate would be required by the parameters found in [35]. These results are all very promising, but there is still a lot of work that can be done.

6.1 Future Work

While we see significant improvements using sieving over the use of random walks for finding relations, there are algorithmic improvements that could be investigated further. For example, the implementation of the sieve array could possibly be improved in order to reduce the amount of computation done in order to "jump" through the array. There is also the possibility of making use of double large primes in the high genus case, as described for the low genus case by Gaudry, Thomé, Thériault and Diem [28]. This idea has been seen before, used for factoring by Lenstra and

Manasse [48]. Finally, the integer smoothness test developed by Bernstein [9] could be adapted to work for polynomials. This may provide further improvement to the algorithms discussed in this thesis.

The work we did focused on even characteristic imaginary quadratic function fields. More precisely, we looked at the high genus case that is arrived at through the use of Weil descent. It would certainly be interesting to investigate the use of the sieving-based implementation of Vollmer's algorithm to generate relations in the lower genus imaginary quadratic function fields that Thériault [65] focused his analysis on, resulting in a comparison of sieving to random walks in this setting, which would have implications in hyperelliptic curve cryptography. In addition, it would be interesting to compare the performance of these algorithms in odd characteristic quadratic function fields. Finally, these algorithms for solving the discrete logarithm problem in the ideal class group of quadratic function fields can be adapted to work in real quadratic function fields.

The Enge-Gaudry algorithm [22] has been analyzed extensively, specifically for low genus cases and the addition of large primes by Thériault [65]. The work done in this thesis, both with the Enge-Gaudry algorithm and Vollmer's algorithm, could be analyzed further. This includes a more in-depth analysis of the methods used to find the parameters used for sieving. This would hopefully lead to a method of choosing these parameters without experimentation.

The index calculus algorithms used in this thesis can be adapted to compute class

groups, as done in quadratic number fields by Hafner and McCurley [29], Paulus [56] and Jacobson [33]. In particular, the improvements to sieving investigated have not been used for this task in quadratic function fields, and the work developed in this thesis could be used as a starting point for an implementation of an algorithm for computing class groups of imaginary and real quadratic function fields.

The above suggestions, like the rest of the thesis, mostly focus on the relation generation stage of index calculus algorithms. In addition to these ideas, one could investigate the linear algebra stage in more detail, and incorporate new linear algebra algorithms into the discrete logarithm algorithms. For example, one could incorporate one of the parallel linear algebra algorithms that have been studied, such as the block Wiedemann strategy as described by Kaltofen [40] or the block Lanczos variant, described by Montgomery [53]. It should be noted that the inclusion of a new linear algebra algorithm would require analyzing the parameters for both Enge-Gaudry and Vollmer's algorithm again in order to ensure the proper balance between the relation generation and linear algebra stages of the algorithm. We would still expect sieving to perform better than random walks in the relation generation stage, but the overall run time would be improved due to the projected improvement in the linear algebra stage of the computation.

This is just a small sample of the work that can be developed based on the work done in this thesis. As this area is both relatively new and relatively unstudied, there are many interesting directions that could be taken from this point. The results of this thesis suggest that the use of sieving in index calculus algorithms can result in an

improvement in a number of areas and settings.

Bibliography

[1] Automatically Tuned Linear Algebra Software (ATLAS). http://math-atlas.sourceforge.net/, 2007.

[2] Centre for Information Security and Cryptography Advanced Cryptographic Laboratory. http://cisac.ucalgary.ca/advanced-cryptographic-laboratory, 2007.

[3] GCC, the GNU Compiler Collection. http://gcc.gnu.org/, 2007.

[4] Message Passing Interface. http://www-unix.mcs.anl.gov/mpi/, 2007.

[5] Project LinBox: Exact computational linear algebra. http://www.linalg.org/, 2007.

[6] The GNU Multiple Precision Bignum Library. http://gmplib.org/, 2007.

[7] L. M. Adleman, J. DeMarrais, and M.-D. Huang. A subexponential algorithm for discrete logarithms over the rational subgroup of the Jacobians of large genus hyperelliptic curves over finite fields. In *Proceedings of the First International Symposium on Algorithmic Number Theory*, volume 877 of *Lecture Notes in Computer Science*, pages 28–40, 1994.

[8] W. Alford and C. Pomerance. Implementing the self-initializing quadratic sieve on a distributed network. In *Number Theoretic and Algebraic Methods in Computer Science, Moscow 1993*, pages 163–174, 1995.

[9] D. J. Bernstein. How to find small factors of integers. manuscript, 2000.

[10] H. Boender and H. J. J. te Riele. Factoring integers with large-prime variations of the quadratic sieve. *Experimental Mathematics*, 5(4):257–273, 1996.

[11] J. Buchmann and S. Paulus. Algorithms for finite abelian groups. In *Number-theoretic and algebraic methods in computer science*, 1995.

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

[13] Certicom Research. SEC1: Elliptic curve cryptography. Technical report, Standards for Efficient Cryptography, 2000.

[14] L. Chen, W. Eberly, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. Efficient matrix preconditioners for black box linear algebra. *Linear Algebra and its Applications*, 343–344:119–146, 2002.

[15] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lane, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Chapman & Hall/CRC, Taylor & Francis Group, 2006.

[16] S. P. Contini. Factoring integers with the self-initializing quadratic sieve. Master's thesis, University of Georgia, 1997.

[17] S. Düllmann. *Ein Algorithmus zur Bestimmung der Klassengruppe positiv definiter binärer quadratischer Formen.* PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 1991.

[18] D. S. Dummit and R. M. Foote. *Abstract Algebra.* John Wiley & Sons, 2nd edition, 1999.

[19] W. Eberly. Reliable Krylov-based algorithms for matrix null space and rank. In *ISSAC '04: Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*, pages 127–134, New York, NY, USA, 2004. ACM.

[20] W. Eberly and E. Kaltofen. On randomized Lanczos algorithms. In *Proceedings, 1997 International Symposium on Symbolic and Algebraic Computation*, pages 176–183, 1997.

[21] A. Enge. Computing discrete logarithms in high-genus hyperelliptic Jacobians in provably subexponential time. *Mathematics of Computation*, 71(238):729–742, 2002.

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

[23] A. Enge and A. Stein. Smooth ideals in hyperelliptic function fields. *Mathematics of Computation*, 71(239):1219–1230, 2002.

[24] R. Flassenberg and S. Paulus. Sieving in function fields. *Experimental Mathematics*, 8(4):339–349, 1999.

[25] G. Frey. How to disguise an elliptic curve (Weil descent). Talk at ECC '98, slides available at `http://www.cacr.math.uwaterloo.ca/conferences/1998/ecc98/frey.ps`, September 1998.

[26] P. Gaudry. An algorithm for solving the discrete log problem on hyperelliptic curves. In *EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 19–34, 2000.

[27] P. Gaudry, F. Hess, and N. P. Smart. Constructive and destructive facets of Weil descent on elliptic curves. Technical report, University of Bristol, October 2000.

[28] P. Gaudry, E. Thomé, N. Thériault, and C. Diem. A double large prime variation for small genus hyperelliptic calculus. *Mathematics of Computation*, 76(257):475–492, 2007.

[29] J. L. Hafner and K. S. McCurley. A rigorous subexponential algorithm for computation of class groups. *Journal of the American Mathematical Society*, 2(4):837–850, 1989.

[30] B. Hovinen and W. Eberly. A reliable block Lanczos algorithm over small finite fields. In *ISSAC '05: Proceedings of the 2005 international symposium on Symbolic and algebraic computation*, pages 117–184, 2005.

[31] Internet Engineering Task Force. The OAKLEY key determination protocol (RFC 2412). `http://www.ietf.org/rfc/rfc2412.txt?number=2412`, November 1998.

[32] M. J. Jacobson, Jr. Applying sieving to the computation of quadratic class groups. *Mathematics of Computation*, 68(226):859–867, 1999.

[33] M. J. Jacobson, Jr. *Subexponential Class Group Computation in Quadratic Orders*. PhD thesis, Technische Universität Darmstadt, Darmstadt, Germany, 1999.

[34] M. J. Jacobson, Jr. Computing discrete logarithms in quadratic orders. *Journal of Cryptology*, 13(4):473–492, 2000.

[35] M. J. Jacobson, Jr., A. Menezes, and A. Stein. Solving elliptic curve discrete logarithm problems using Weil descent. *Journal of the Ramanujan Mathematical Society*, 16(3):231–260, 2001.

[36] M. J. Jacobson, Jr., A. Menezes, and A. Stein. Hyperelliptic curves and cryptography. *Fields Institute Communications*, 41:255–282, 2004.

[37] M. J. Jacobson, Jr., R. Scheidler, and A. Stein. Cryptographic protocols on real hyperelliptic curves. *Advances in Mathematics of Communications*, 1(2):197–221, 2007.

[38] M. J. Jacobson, Jr., R. Scheidler, and A. Stein. Fast arithmetic on hyperelliptic curves via continued fraction expansions. In *Advances In Coding Theory And Cryptography*, 2007.

[39] M. J. Jacobson, Jr. and A. J. van der Poorten. Computational aspects of NUCOMP. In *Algorithmic Number Theory - ANTS-V*, volume 2369 of *Lecture Notes in Computer Science*, pages 120–133, 2002.

[40] E. Kaltofen. Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems. *Mathematics of Computation*, 64(210):777–806, 1995.

[41] E. Kaltofen and B. D. Saunders. On Wiedemann's method of solving sparse linear systems. In *Applied Algebra, Algebraic Algorithms and Error-Correcting*

Codes, volume 539 of *Lecture Notes in Computer Science*, pages 29–38, 1991.

[42] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987.

[43] N. Koblitz. Hyperelliptic cryptosystems. *Journal of Cryptology*, 1(3):139–150, 1989.

[44] N. Koblitz. *Algebraic Aspects of Cryptography*, volume 3 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, 2nd edition, 1999.

[45] B. Kurowski. The multiple polynomial quadratic sieve. paper, April 1998.

[46] B. LaMacchia and A. M. Odlyzko. Solving large sparse linear systems over finite fields. In *Advances in Cryptology - CRYPTO '90: Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 109–133, 1991.

[47] C. Lanczos. Solutions of systems of linear equations by minimized iterations. *Journal of Research of the National Bureau of Standards*, 49:33–53, 1952.

[48] A. K. Lenstra and M. S. Manasse. Factoring with two large primes. *Mathematics of Computation*, 63(208):785–798, October 1994.

[49] M. Maurer, A. 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*, 5:127–174, 2002.

[50] K. S. McCurley. Cryptographic key distribution and computation in class groups. In R. A. Mollin, editor, *Number Theory and Applications*, pages 459–479, 1989.

[51] A. Menezes and M. Qu. Analysis of the Weil descent attack of Gaudry, Hess and Smart. In *Progress in Cryptology - CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 308–318, 1999.

[52] R. A. Mollin. *Algebraic Number Theory.* Chapman & Hall/CRC, 1999.

[53] P. L. Montgomery. A block Lanczos algorithm for finding dependencies in $GF(2)$. In *EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 106–120, 1995.

[54] T. Mulders. Certified sparse linear system solving. *Journal of Symbolic Computation*, 38(5):1343–1373, 2004.

[55] NSA. Fact sheet: NSA Suite B Cryptography. `http://www.nsa.gov/ia/industry/crypto_suite_b.cfm`, 2007.

[56] S. Paulus. An algorithm of subexponential type computing the class group of quadratic orders over principal ideal domains. In *Second International Symposium, ANTS-II Talence, France, May 18–23, 1996 Proceedings*, volume 1122 of *Lecture Notes in Computer Science*, pages 243–257, 1996.

[57] S. Paulus and H.-G. Rück. Real and imaginary quadratic representations of hyperelliptic function fields. *Mathematics of Computation*, 68(227):1233–1241, 1999.

[58] S. Paulus and A. Stein. Comparing real and imaginary arithmetics for divisor class groups of hyperelliptic curves. In *ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*, pages 576–591, 1998.

[59] J. M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.

[60] C. Pomerance. The quadratic sieve factoring algorithm. In *Advances in Cryptology - Proceedings of EUROCRYPT 84*, volume 209 of *Lecture Notes in Computer Science*, pages 169–182, 1985.

[61] D. Shanks. Class number, a theory of factorization and genera. In *Proc. Symp. Pure Math. 20*, pages 415–440, 1981.

[62] V. Shoup. NTL: A library for doing number theory. `http://www.shoup.net/ntl/`, 2007.

[63] R. D. Silverman. The multiple polynomial quadratic sieve. *Mathematics of Computation*, 48(177):329–339, January 1987.

[64] E. Teske. Speeding up Pollard's rho method for computing discrete logarithms. In *Algorithmic Number Theory*, volume 1423 of *Lecture Notes in Computer Science*, pages 541–554, 1998.

[65] N. Thériault. Index calculus attack for hyperelliptic curves of small genus. In *ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 75–92, 2003.

[66] U. Vollmer. Asymptotically fast discrete logarithms in quadratic number fields. In *ANTS-IV*, volume 1838 of *Lecture Notes in Computer Science*, pages 581–594, 2000.

[67] U. Vollmer. A note on the Hermite basis computation of large integer matrices. In J. R. Sendra, editor, *Proc. Int'l. Symp. on Symbolic and Algebraic Computation: ISSAC '03*, pages 242–249. ACM Press, 2003.

[68] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, 2nd edition, 2003.

[69] D. H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, IT-32(1):54–62, 1986.

Index