CARL VON OSSIETZKY universität OLDENBURG

IfM INSTITUT FÜR MATHEMATIK

Explicit Methods for Solving the Discrete Logarithm Problem on Algebraic Curves

Der Fakultät für Mathematik und Naturwissenschaften der *Carl von Ossietzky Universität Oldenburg* zur Erlangung des Grades und Titels eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

vorgelegte Dissertation von

Herrn Wilke Idäus Trei

geboren am 11. September 1985 in Norden.

Betreuender Gutachter:	**Prof. Dr. Andreas Stein**
Zweitgutachter:	**Prof. Michael J. Jacobson, Jr.**
Tag der Disputation:	**31.03.2025**

Zusammenfassung

Diskrete Logarithmusprobleme (DLPs) sind ein Grundpfeiler der modernen Public-Key-Kryptographie und spielen eine zentrale Rolle für die Informationssicherheit und Authentifizierung. Insbesondere DLPs in der Jacobischen Gruppe einer elliptischen Kurve über einem endlichen Körper haben sich als effizient erwiesen, da sie eine gute Balance zwischen der Komplexität arithmetischer Operationen, Schlüsselgröße und erwarteter Sicherheit bieten. Für DLPs auf allgemeinen elliptischen Kurven existieren nur generische Algorithmen, die typischerweise Laufzeiten proportional zur Quadratwurzel der Gruppengröße haben. Hyperelliptische und allgemeinere algebraische Kurven bieten jedoch zusätzliche Strukturen, die für die Lösung von DLPs genutzt werden können.

In dieser Dissertation präsentieren wir eine umfassende Analyse von DLPs auf allen algebraischen Kurven, die mindestens einen voll verzweigten Punkt bieten. Alle diese Kurven lassen sich in der Form einer $C_{n,d}$ Kurve darstellen, wobei der voll verzweigte Punkt im Unendlichen liegt und n und d die höchsten Potenzen der auftretenden Koeffizienten der Kurvengleichung angeben. Diese große Unterklasse von algebraischen Kurven bietet die Möglichkeit, Elemente der Gruppe surjektiv mit affin repräsentierten Idealklassen des Funktionenkörpers der Kurve zu identifizieren. Dies führt zu einer natürlichen Darstellung der Elemente der Jacobischen Gruppe, erlaubt die Entwicklung expliziter Formeln für die Gruppenarithmetik und macht diese Klasse für kryptographische Anwendungen attraktiv. Zudem erlaubt sie es, Erkenntnisse über die Arithmetik der Ideale auf die Jacobische Gruppe anzuwenden. Die Klasse der $C_{n,d}$-Kurven umfasst sowohl imaginäre hyperelliptische und superelliptische Kurven als auch weitere Beispiele, für die bereits fortschrittliche Lösungsmethoden mit niedriger Komplexität untersucht wurden.

Der Schwerpunkt dieser Arbeit liegt in der Einführung eines einheitlichen Rahmens zur Analyse und Lösung von DLPs auf dieser Kurvenklasse, einschließlich der Entwicklung neuer Algorithmen, die auf dem Prinzip des Index Calculus und einer Siebmethode basieren. Wir zeigen, dass diese verbesserten Siebmethoden in jeder Situation den bekannten Ergebnissen für alle Unterklassen von $C_{n,d}$-Kurven überlegen sind, da sie beweisbar weniger arithmetische Operationen zur Lösung der DLPs erfordern.

Ein zentraler Aspekt der Komplexität eines jeden Index-Calculus-Verfahrens ist die erwartete Glattheitswahrscheinlichkeit, mit der von dem Algorithmus erzeugte Elemente über einer zuvor festgelegten Faktorbasis zerfallen. Im Falle von algebraischen Kurven handelt es sich dabei um die Wahrscheinlichkeit von Idealen des Ganzheitsring des Funktionenkörpers der Kurve über einer Faktorbasis aus kleingradigen Primidealen. Wir präsentieren eine detaillierte Analyse dieser Wahrscheinlichkeit in Abhängigkeit von den Kurveninvarianten, die bestehende Ergebnisse bestätigt und für bisher nicht untersuchte Kurven

in der Klasse der $C_{n,d}$ Kurven erweitert. Damit bieten wir für jedes DLP auf Kurven dieser Klasse umfassende Methoden zur Analyse und Lösung des Problems.

Abschließend demonstrieren wir die Leistungsfähigkeit der in dieser Arbeit neu entwickelten Algorithmen mithilfe von zwei Implementierungen, die für unterschiedliche Szenarien optimiert sind. Im direkten Vergleich mit bestehenden Ansätzen zeigt sich die Effektivität und Effizienz der neuen Algorithmen, die es erlauben, auch schwere Instanzen des DLPs mit deutlich verringertem Zeit- und Energieaufwand zu lösen. Zudem zeigen wir, dass die entwickelten Algorithmen für den Einsatz auf parallel arbeitenden Beschleunigerchips, wie z.B. Grafikkarten, geeignet sind und die erzielte Geschwindigkeit signifikant vom Einsatz dieser modernen Hardware profitiert.

Abstract

Discrete logarithm problems (DLPs) are a cornerstone of modern public-key cryptography and play a central role in information security and authentication. In particular, DLPs in the Jacobian group of an elliptic curve over a finite field have proven to be efficient, as they offer a good balance between the complexity of arithmetic operations, key size, and expected security. For DLPs on general elliptic curves, only generic algorithms exist that typically have runtimes proportional to the square root of the group size. However, hyperelliptic and more general algebraic curves provide additional structures that can be utilized for solving DLPs.

In this dissertation, we present a comprehensive analysis of DLPs on all algebraic curves having at least a single fully ramified point. All these curves can be represented as $C_{n,d}$ curves where the fully ramified point extends the rational place at infinity. Here, n and d denote the curve degrees in its two coordinates, making them the most relevant invariants of the curve. This form of a curve leads to a natural representation of the elements of its Jacobian group, which allows the development of explicit formulas for group arithmetic and makes this class attractive for cryptographic applications. Additionally, the representation allows insights from the arithmetic of ideals to be applied to the Jacobian group. The class of $C_{n,d}$ curves includes both imaginary hyperelliptic curves and other curves for which advanced low-complexity solution methods have already been studied.

The focus of this work is to introduce a unified framework for analyzing and solving DLPs on this curve class, including new algorithms based on the principle of index calculus and a sieving method. We demonstrate that these improved sieving methods are consistently superior to known results for subclasses of these curves, as they demonstrably require fewer arithmetic operations to solve the DLP.

The smoothness probability of ideals over a factor base, which is required for the index calculus procedure, is crucial for the complexity class of the overall solution procedure. We present a detailed analysis of this probability depending on the curve parameters, which confirms existing results and extends them to previously unstudied parameterizations. Thus, for each DLP on curves of this class with a certain minimum difficulty, we provide comprehensive methods for analyzing and solving the problem.

Finally, we demonstrate the performance of our algorithms using two optimized implementations. Compared to existing approaches, the new algorithms are highly efficient, enabling the solution of previously intractable DLPs using today's resources.

Contents

Notations 3

Introduction 7

1. Preliminaries 15
- 1.1. Discrete Logarithms . . . 15
  - 1.1.1. Definitions and Applications . . . 15
  - 1.1.2. Generic Solution Strategies for Discrete Logarithm Problems . . . 18
  - 1.1.3. Index Calculus Solution Strategies for Discrete Logarithm Problems 22
- 1.2. Algebraic Curves . . . 27
  - 1.2.1. Algebraic Function Fields . . . 28
  - 1.2.2. The Divisor Class Group of Function Fields . . . 32
  - 1.2.3. $C_{n,d}$ Curves and their Properties . . . 36
  - 1.2.4. Number Theoretic Aspects of Algebraic Curves . . . 37
  - 1.2.5. Divisor Representation and Decomposition . . . 42
  - 1.2.6. Elliptic Curve Discrete Logarithm Problems and Weil Descent . . 50

2. Smoothness Estimates for Divisors in Function Fields 55
- 2.1. Existing Estimates and Approaches . . . 56
- 2.2. Counting Prime Divisors of a Fixed Degree . . . 59
- 2.3. A Refined Estimate for Low Degree Divisor Decomposition . . . 62
- 2.4. A Refined Estimate for High Degree Divisor Decomposition . . . 66
- 2.5. Discussion of Results . . . 78

3. Explicit Methods for Computing Discrete Logarithms over $C_{n,d}$ Curves 81
- 3.1. Existing Approaches for Solving Discrete Logarithms in Jacobians . . . 82
  - 3.1.1. The Sieve of Velichka, Jacobson, and Stein . . . 82
  - 3.1.2. The Sieve of Sarkar and Singh . . . 85
  - 3.1.3. $L_{q^g}\left[\frac{1}{3}, O(1)\right]$ Discrete Logarithms of Gaudry, Enge, and Thomé . . 85
  - 3.1.4. Large Prime Methods for Low Degree Curves . . . 87
- 3.2. Sieving for General Relations over a Factor Base . . . 94
  - 3.2.1. Construction of Sieve Polynomials . . . 94
  - 3.2.2. An Efficient Sieve Initialization . . . 98
  - 3.2.3. A linear Time Sieve Iteration Algorithm . . . 105
  - 3.2.4. Discussion of Results . . . 114
- 3.3. A Large Prime Graph Scheme for Low Degree Curves . . . 118
  - 3.3.1. Construction of a Large Prime Graph . . . 118
  - 3.3.2. Parameter Balancing for Complete DLP Computations over Low Degree Curves . . . 123

3.3.3. Discussion of Results . . . 127
3.4. Auxiliary Routines for DLP Computations . . . 131
3.4.1. Building the Factor Base . . . 131
3.4.2. Special Divisor Decomposition . . . 134
3.5. Summary . . . 141

4. Implementation and Numerical Results **143**
4.1. Description of Implementations . . . 144
4.1.1. Implementation for Large Genus Hyperelliptic Curves over Even Characteristic Fields . . . 144
4.1.1.1. Overview and Experiments Setup . . . 144
4.1.1.2. Implementation Details . . . 145
4.1.2. Implementation for Small Genus Hyperelliptic Curves over Odd Characteristic Fields on Graphics Cards . . . 149
4.1.2.1. An Introduction to OpenCL . . . 150
4.1.2.2. Implementation Details . . . 153
4.2. Experimental Results . . . 158
4.2.1. A Family of Curves from Weil Descent . . . 158
4.2.2. 184 Bit Hyperelliptic DLP from a Trapdoor Curve . . . 163
4.2.3. Investigation of the Smoothness Probability and Algorithm Runtime for Varying Genus . . . 166
4.2.4. GPU Experiments with Small Genus . . . 174
4.3. Discussion of Results . . . 180

Outlook **183**

A. Complexity Theory and Examples **187**
A.1. Subexponential Complexity Theory . . . 187
A.2. Results from Probability Theory . . . 190
A.3. Polynomial Factoring and Root Finding . . . 192
A.3.1. Computing Roots of Polynomials of Arbitrary Degree . . . 193
A.3.2. Finding Roots of Quadratic Polynomials in Odd Characteristic . . . 194
A.3.3. Finding Roots of Quadratic Polynomials in Characteristic 2 . . . 197
A.4. Solving Linear Systems . . . 201
A.5. Gray Codes . . . 212

B. Bibliography **217**

C. Experiment Curves and Additional Numeric Results **225**
C.1. Weil Descent Curves from Sections 4.2.1 and 4.2.2 . . . 225
C.2. Scaling Test Curves and Additional Results for Section 4.2.3 . . . 227
C.3. Test Curves and Detailed Results for Section 4.2.4 . . . 241

Notations

General Mathematical Notation and Complexity Analysis

Symbol	Meaning
$\mathbb{N}$	The set of natural numbers
$\mathbb{N}_0$	The set of non-negative integers ($\mathbb{N} \cup \{0\}$)
$\mathbb{Z}$	The ring of integers
$\lceil x \rceil$	Ceiling function (smallest integer $\geq x$)
$\lfloor x \rfloor$	Floor function (largest integer $\leq x$)
$\log_b x$	Logarithm of x to the base b
$\ln x$	Natural logarithm of x
$O(\cdot)$	Asymptotic upper bound
$\widetilde{O}(\cdot)$	Asymptotic upper bound without logarithmic factors
$o(1)$	Term approaching 0 for a growing variable
$L_n[\alpha, \beta]$	Sub-exponential complexity function defined as $e^{\beta(\ln n)^\alpha (\ln \ln n)^{1-\alpha}}$
$L_n[\alpha]$	Sub-exponential complexity function with $\beta = O(1)$

Discrete Logarithm Problem and Algorithms

Symbol	Meaning
G	A finite group
$\langle \gamma \rangle$	A cyclic subgroup generated by a single element γ
$\log_\gamma \alpha$	Discrete logarithm of α to the base γ
$\mathcal{F}$	Factor base
η	Degree of divisors being tested for smoothness
m	Degree bound for factor base elements
θ	Parameter relating degree η to the field size, $\theta = \eta / \ln q$

Symbol	Meaning
μ	An ordering map $\mathbb{N} \to \mathbb{F}_q[x]$
ν	Index function mapping polynomials to sieve array indices
T	Tolerance value for sieving
M	Sieve bound (max degree of the polynomial $s(x)$ in the sieve)

Algebraic Curves and Function Fields

Symbol	Meaning
C	An algebraic curve
$C_{n,d}$	Class of curves defined by $y^n + x^d + \cdots = 0$, satisfying additional constraints as presented in Definition 1.2.28
n	Degree of the curve equation in y, which equals the extension degree $[\mathbb{F}_q(x,y) : \mathbb{F}_q(x)]$
d	Degree of the curve equation in x
g	Genus of the curve
F	Function field of the curve, often equal to $\mathbb{F}_q(x,y)$
K	Exact field of constants, often equal to $\mathbb{F}_q$
$\mathcal{O}$	Valuation ring within $\mathbb{F}_q[x,y]$
$\mathbb{P}_F$	Set of places of the function field F
P_∞	The unique place at infinity for a rational function field or a $C_{n,d}$ curve
Div	Group of divisors of a given function field
Div_0	Group of divisors of degree 0
$\mathrm{Jac}(C)$	Jacobian group of the curve (Div_0/principal divisors)
$[D]$	Divisor class of a divisor modulo the set of principal divisors
$v_P(\alpha)$	Discrete valuation of an element α at the place P
$\mathrm{Cl}(F)$	Ideal class group of F

Finite Fields and Polynomials

Symbol	Meaning
$\mathbb{F}_p$	Finite field with p elements, where p is prime
$\mathbb{F}_q$	Finite field with $q = p^k$ elements
$\mathbb{F}_q[x]$	Ring of polynomials in the variable x over $\mathbb{F}_q$
$\mathbb{F}_q(x)$	Rational function field in the variable x over $\mathbb{F}_q$
$\mathcal{N}(I)$	Norm of an ideal $I \subset \mathbb{F}_q[x, y]$ over $\mathbb{F}_q[x]$
$\mathcal{N}_{\mathbb{F}_q(x)\subset F}(\alpha)$	Norm of an element $\alpha \in F$ defined by the field norm over $\mathbb{F}_q(x)$
$\mathrm{Res}_y(f, g)$	Resultant of polynomials f and g with respect to the variable y
$\mathrm{Tr}(x)$	Trace of an element x over $\mathbb{F}_2$

Introduction

The presumed difficulty of solving discrete logarithm problems (DLPs) is foundational to modern public-key cryptography and crucial for information security and authentication. Specifically, DLPs in the Jacobian group of an elliptic curve over a finite field have gained prominence due to the excellent trade-off between the computational effort of curve arithmetic, key size, and expected security. Traditional algorithms based on factorization problems or discrete logarithms in the multiplicative group of a finite field have increasingly been replaced by those using elliptic curves.

For example, this is reflected in NIST FIPS 186-5 [46], where the classic DSA algorithm based on finite field discrete logarithm problems is considered deprecated and allowed only for verifying existing signatures. With the advent of quantum computing, newer publications of NIST, e.g., FIPS 203, 204, and 205 [47], introduce lattice-based or hash-based cryptography as potential successors to curve-based methods. Since curve-based methods are the current workhorse of public-key cryptography with accepted security assumptions, these methods will remain relevant for the foreseeable future. Additionally, there are examples of recommendations for hybrid algorithms combining post-quantum cryptography with curve-based approaches to build on the accepted properties of elliptic curve cryptography [20].

For the discrete logarithm problem on a generic elliptic curve, only generic solution algorithms are available to date, typically with a complexity scaling with the square root of the group size. Examples include Shanks' algorithm [53] and Pollard's Rho algorithm [49].

However, the complexity changes when we extend our focus to non-elliptic algebraic curves or consider elliptic curves with specific vulnerabilities. Key examples include supersingular curves, which possess large endomorphism rings exploitable for cryptanalysis [43], and the Weil descent method, where the Jacobian group of a curve and the DLP instance of interest are mapped into the Jacobian group of a higher-genus curve, enabling the DLP solution search to be performed in this new group [23].

These are just a few reasons why DLPs on general algebraic curves over finite fields are

of practical interest, besides being interesting algorithmic problems in themselves.

When analyzing the complexity of DLPs beyond elliptic curves, many studies focus on larger subfields of algebraic curves and analyze specific curve structures. In most cases, the algorithms chosen for this analysis are based on the idea of an index calculus method. The complexity of these algorithms hinges on a trade-off between the difficulty of finding sufficient relations over a factor base and the computational cost of solving the resulting linear system.

For hyperelliptic curves of relatively small genus over large finite fields, cryptanalysis often involves fixing the genus and exploring the complexity over different base fields. The factor base is constructed from a subset of divisors of degree one, and relations are generated through trial division [24, 60]. This process is commonly optimized with variants of large-prime methods [25]. Despite these improvements, solving the DLP in this context remains exponential relative to the size of the Jacobian group.

Conversely, for hyperelliptic curves over a fixed finite field with sufficiently large genus, the complexity of the hyperelliptic curve discrete logarithm problem (HCDLP) becomes subexponential [19]. Specifically, this class of problems can be solved in subexponential time $L_{q^g}\left[\frac{1}{2}, \sqrt{2}\right]$, where L denotes the subexponential function as defined in Definition A.1.1. As discussed later, the genus of the curve must be quite large to ensure that the required factor base exists and that the divisors exhibit a sufficiently high probability of smoothness.

Beyond hyperelliptic curves, additional research focuses on identifying classes of curves that have a special structure or backdoors, allowing specialized faster solution techniques. For instance, in [18], a family of curves is presented where the DLP can be solved in subexponential time $L_{q^g}\left[\frac{1}{3} + \epsilon\right]$. These curves currently have the lowest known complexity for solving their DLPs when generated randomly within their family.

From an algorithmic perspective, higher-genus curves provide advantages in generating elements that are smooth over the factor base compared to lower-genus cases. Building on a framework from [21], Velichka, Jacobson, and Stein [62] developed a sieving algorithm for the HCDLP over large-genus, characteristic-2 finite fields. This sieving approach bypasses the need for trial division, offering a logarithmic speedup relative to the size of the Jacobian of the hyperelliptic curve. However, a drawback of this method is that the divisors generated by the sieve tend to have higher degrees compared to randomly generated ones, and thus the sieve-generated divisors are less likely to be smooth over the factor base. This makes the method less effective for small-genus hyperelliptic curves and requires careful

control of the search space in higher-genus cases.

In particular, our contributions include a DLP solution strategy and a careful analysis that not only unifies existing results but also extends them to a broader class of algebraic curves. The primary research object of this dissertation is the DLP on $C_{n,d}$ curves as defined in Definition 1.2.28. These are general algebraic curves, with the sole restriction that they possess a fully ramified point at infinity. This class encompasses both imaginary (hyper)elliptic curves and the family of curves from [18], which allowed solving the DLP in $L_{q^g}\left[\frac{1}{3}+\epsilon\right]$. By applying additional techniques, such as those in [48], we assume that the methods provided and analyzed in this thesis are not limited to $C_{n,d}$ curves, yet their structure simplifies the general analysis.

$C_{n,d}$ curves allow us to surjectively map affine-represented elements from the ideal class group of the function field of the curve to elements of the curve's Jacobian group. Therefore, the generation of relations for the DLP computation, as well as smoothness estimates and factor base properties, can be carried out using ideal arithmetic and computations without requiring additional steps to resolve divisors supported at multiple points that extend the infinite rational place.

We present an extensive analysis of the smoothness probability for this class of curves, drawing upon ideas from [16, 19, 60]. The perspective from [16] enables us to relate the size of the underlying finite field to the degree of the divisors to be factored, allowing us to conduct a smoothness analysis based on this relationship. This method allows us to uniformly address both small- and large-genus curves without needing to fix the genus, curve model, or base field for our conclusions.

Additionally, the analysis is flexible concerning the degree of the generated elements, making the results applicable to a wide range of algorithms and different curve models. We also show that the spectrum of possible DLP complexities can be continuously extended as the size of the Jacobian group increases, whether through modifications in the genus, curve representation, or finite field size. One of the central results of these considerations is summarized in Theorem 2.5.1. This reads:

Result 1. *Let C be a $C_{n,d}$ curve of genus g represented by the function field $\mathbb{F}_q(x,y)$ of degree n over $\mathbb{F}_q(x)$. Fix a degree $\eta \in \mathbb{N}$ such that we have an algorithm to create random function field elements with norm-degree less than or equal to η, and let $\theta := \frac{\eta}{\ln q}$.*

Assume

$$\max\left\{0.5\left\lfloor\sqrt{\frac{\theta}{2}}\sqrt{\ln\ln q^{\eta}}\right\rfloor, 1\right\} \geq 2\log_q\left(3+4g+4n\right).$$

Let $\mathcal{F}$ be a factor base of size $L_{q^\eta}\left[\frac{1}{2}, \frac{\beta}{2}\right]$ consisting of those split prime ideals of $\mathbb{F}_q[x, y]$ of lowest norm over $\mathbb{F}_q(x)$, and let β be selected as

$$\beta = \begin{cases} \frac{\sqrt{2}}{\sqrt{\frac{\theta}{2}}\sqrt{\ln \ln q^\eta}} & \text{for } \sqrt{\frac{\theta}{2}}\sqrt{\ln \ln q^\eta} < 1 \\ \sqrt{2} & \text{for } \sqrt{\frac{\theta}{2}}\sqrt{\ln \ln q^\eta} \geq 1. \end{cases}$$

Then the probability that the created function field elements of degree less than η are $\mathcal{F}$-smooth is bounded below by $L_{q^\eta}\left[\frac{1}{2}, -\frac{\beta}{2} - o(1)\right]$.

Note that the technical assumption of the theorem is satisfied whenever η or q is sufficiently large. For example, if we fix a curve model, so that the genus g, n, and η remain constant, then for variable field sizes q, the condition is met whenever $1 \geq 2\log_q(3 + 4g + 4n)$. If q is fixed instead, the condition implies a minimal value of η for the theorem to be valid. In general, the technical assumptions are met provided there is a minimum problem size for the DLP.

The theorem implies that for sufficiently large η, there is a factor base whose size is subexponential in q^η, and which offers a subexponential probability of the function field elements being smooth over the factor base. Conversely, if η is small relative to q, then for the choice of β, both the factor base size and the smoothness probability are exponential in q^η. The achievement of this theorem is that it unifies both cases and depicts a clear transition between them. Furthermore, we observe that the final complexity of collecting sufficiently many smooth elements over the factor base does not depend on the actual size of the Jacobian, but instead on the achievable value of η.

In Chapter 3, we develop a general algorithm designed to solve the DLP for the broad class of $C_{n,d}$ curves, achieving provably low values of η. We build upon the concepts from [21] and [62] and extend them to work for this more general class of curves. Moreover, we present optimized methods for both the sieving step and the generation of the sieving polynomials and their roots.

Compared to existing approaches for high-genus hyperelliptic curves, these improvements yield a significant speedup that is logarithmic in q^g for divisor decomposition. The following result is a reformulated version of Theorem 3.2.24.

Result 2. *Let the factor base be configured as in Result 1, with B bounding the degree of its elements. Suppose $M \geq B - 1$ is a sieve bound allowing q^M elements to be tested for smoothness in one pass, within the framework of Section 3.2.1.*

Then the expected time for the test for $\mathcal{F}$-smoothness of each element in the search space is in the complexity class

$$O\left(g\left(\ln q\right)^2\right).$$

We observe that, because of the scaling effects of sieving, the unit cost per element tested for smoothness is even lower than that of the best-known algorithms for performing the group law on general $C_{n,d}$ curves as given in [26]. Also, the provided complexity is lower than that of the group law on high-genus hyperelliptic curves or that of trial-dividing a polynomial of degree g. Both algorithms are known to be in the complexity class $O\left(g^2\left(\ln q\right)^2\right)$.

When comparing the sieve approach examined in this thesis with the methods from [24] for low-genus hyperelliptic curves, our approach has the disadvantage of generating elements that have degree $g+1$ instead of g. While this is not a significant problem for curves with a subexponential relationship between factor base size and smoothness probability, this factor potentially has a negative impact on the smoothness probability and thus the overall algorithm complexity, as pointed out in Result 1.

Guided by the detailed analysis leading to Result 2, we develop a variant that is computationally more efficient per element, effectively offsetting the drawback in smoothness probability. Consequently, by employing the new algorithms, the DLP on small-degree curves can be solved within the same complexity class as previous algorithms using trial division, with even lower logarithmic factors.

Furthermore, we introduce an alternative double-large-prime approach for low-degree curves that requires finding fewer relations compared to the method developed in [25]. Combining the results of the new large-prime method with the sieving technique, we achieve the following result for hyperelliptic curves of small genus.

Result 3. *Let C be an imaginary hyperelliptic curve of genus g over the finite field $\mathbb{F}_q$, and let $g < \frac{\ln q}{\ln \ln q^g}$. Then there is an algorithm that solves any instance of the discrete logarithm problem on C in expected time:*

$$O\left(q^{2-\frac{2}{g}}(g-1)^2 g^{2-\frac{2}{g}}\left(\ln q\right)^{3-\frac{2}{g}}\right) < q^{2-\frac{2}{g}} g^4 O\left(\left(\ln q\right)^3\right).$$

We note that the condition on the curve is met for any fixed genus g and large enough q. Therefore, Result 3 improves the result of [25] as analyzed in Theorem 3.1.12 by at least a factor of $g^{1+\frac{2}{g}}\left(\ln q\right)^{\frac{2}{g}}$.

The sieve method is particularly advantageous for non-hyperelliptic $C_{n,d}$ curves and thus curves with $n > 2$. When initialized optimally, it generates elements with a provable degree

of $\left\lceil \frac{n-1}{n} d \right\rceil + M \approx d$, rather than the genus $g = \frac{(n-1)(d-1)}{2}$. Consequently, the sieve yields elements of a significantly smaller degree than standard divisor reduction using the curve's group law, which typically produces elements of degree g.

Therefore, if we consider a $C_{n,d}$ curve where $n \approx d$, we can solve the discrete logarithm problem on the curve much more quickly than on a hyperelliptic curve with a Jacobian group of similar size. These results are consistent with the findings of [18] and demonstrate that our sieving technique is also applicable to these curves. In contrast to the findings of Enge and Gaudry, our methods also apply to low-degree or low-genus curves, and therefore we can provide the following result as an example regarding Picard curves, which are non-hyperelliptic $C_{n,d}$ curves of genus 3.

Result 4. *Let C be a non-hyperelliptic $C_{n,d}$ curve of genus 3 over $\mathbb{F}_q$. Then any instance of the discrete logarithm problem can be solved in expected time:*

$$O\left(q(\ln q)^2\right).$$

Following the theoretical construction of the sieve, Chapter 4 presents two implementations of the algorithms discussed in the previous chapters and tests them in various scenarios to analyze their performance. First, we enhanced the implementation of [62], which is used for hyperelliptic curves with a high genus in even characteristic. Our results show that the theoretical logarithmic optimization factor can, in practice, lead to speedups of up to two orders of magnitude, as the new methods can be implemented in a highly streamlined manner.

We also tested our enhanced implementations on a hyperelliptic curve with a Jacobian of size $\log_2 q^g = 184$ and genus 8 over $\mathbb{F}_{2^{23}}$, which was first presented in [59]. We show that problems of this size are feasible within a few days using the sieving method and a modern high-performance computer cluster.

Finally, we adapted the algorithm for small-degree curves in odd characteristic. The implementation was tailored for parallel execution on graphics cards, and it demonstrates that the new algorithms are well-suited for use with these modern computing accelerators. We also conducted various tests using large-prime methods with this implementation, providing further insights into the practical performance of the described algorithms.

In conclusion, this dissertation presents a unified and expansive framework for solving the discrete logarithm problem for a large group of algebraic curves. By building on existing methods and introducing novel optimizations, we demonstrate significant improvements

in both theoretical complexity and practical performance. Our contributions range from high-degree, high-genus curves to low-degree cases, ensuring that our algorithms are versatile and applicable in diverse cryptographic settings. With practical implementations that leverage modern computing architectures, such as GPUs, we provide strong evidence of the real-world applicability of these methods.

1

Preliminaries

In this chapter, we cover the mathematical foundations necessary to consider discrete logarithm problems on $C_{n,d}$-curves.

This chapter includes the definitions, notations, and conventions used throughout the thesis, organized into two main areas: discrete logarithm problems and their general solution approaches; and algebraic curves, along with their Jacobian groups. As both areas are well established, we will limit this chapter to essential definitions and theorems necessary for understanding the thesis, referencing them where appropriate.

Fundamental knowledge of complexity theory and the complexity of certain basic algorithms is not covered here and has instead been summarized in Appendix A.

1.1. Discrete Logarithms

1.1.1. Definitions and Applications

Definition 1.1.1 (Discrete Logarithm)**.** Let $G = \langle \gamma \rangle$ be a finite, cyclic group where the group law is denoted multiplicatively. Let γ be the generator of the group and let

$\alpha \in G$ be an arbitrarily chosen element. We define $\log_\gamma \alpha < |G|$ to be the least positive integer satisfying $\gamma^{\log_\gamma \alpha} = \alpha$ in G. We call $\log_\gamma \alpha$ the *discrete logarithm* of α on G with respect to the generator γ.

Remark 1.1.2. *For additively written groups, the discrete logarithm is defined analogously. Although the notation* $\left(\log_\gamma \alpha\right) \cdot \gamma = \alpha$ *may be confusing when considered outside its context, the term* $\log_\gamma \alpha$ *will also be used in this case.*

For a given group G with generator γ and given element a, finding the discrete logarithm $\log_\gamma a$ is one of the most important problems in computational number theory with respect to classical cryptography. The main reason is that both digital signature schemes and the Diffie-Hellman scheme for key exchange over an insecure communication channel rely on the presumed difficulty of solving an instance of the DLP.

Algorithm 1.1.3 (Diffie-Hellman Key Exchange Scheme)**.**

Input: Two parties, A and B, connected via an insecure connection and a cyclic group $G = \langle\gamma\rangle$ offering a hard-to-solve discrete logarithm problem.

Output: A shared secret element α of the group.

1: Share the group generating element γ through the insecure connection.
2: A chooses a random secret element $x \in \{0, \ldots, \mathrm{ord}(\gamma) - 1\}$
3: A sends $x\gamma$ to B via the connection.
4: B chooses a random secret element $y \in \{0, \ldots, \mathrm{ord}(\gamma) - 1\}$
5: B sends $y\gamma$ to A through the connection.
6: A computes the shared secret element $\alpha := xy\gamma = x(y\gamma)$.
7: B computes the shared secret element $\alpha := xy\gamma = y(x\gamma)$.

Given an eavesdropper listening to exchanged parameters, the security of the encryption relies on the difficulty of reconstructing x or y from G, γ, $x\gamma$ and $y\gamma$, i.e., solving the discrete logarithm of $x\gamma$ or $y\gamma$ in G. The following table gives an overview of the complexity classes for solving the DLP in certain groups. For some of the instances we will use the subexponential complexity notation. This will be briefly explained in Appendix A.1.

Example 1.1.4. *This table gives an overview of the difficulty of discrete logarithm problems that are either used in cryptography or are under recent scientific investigation.*

Group	***Parameter Restrictions***	***Complexity of Solution***
$\mathbb{Z}/n\mathbb{Z}, +$	*n is a positive integer*	$O\left((\ln n)^2\right)$ *by the extended Euclidean algorithm.*
$\mathbb{F}_p^*, \cdot$	*p prime*	$L_p\left[\frac{1}{3}, \left(\frac{64}{9}\right)^{\frac{1}{3}}\right]$ *by a variant of the number field sieve [12].*
$\mathbb{F}_{p^k}^*, \cdot$	*p prime, k even and* $p \leq L_{p^k}(\alpha)$	$L_{p^k}[\alpha + o(1))]$, *subexponential to quasi polynomial depending on p and k [3]*
$(\mathbb{Z}/n\mathbb{Z})^*$	$n = p \cdot q$ *with* $p < q$ *prime numbers*	*Equivalent to factoring n;* $L_p\left[\frac{1}{2}, \sqrt{2}\right]$ *with the ECM algorithm [40] or* $L_n\left[\frac{1}{3}, \left(\frac{64}{9}\right)^{\frac{1}{3}}\right]$ *by the number field sieve [39]*
$\mathrm{Jac}(C)$	*C is a* $C_{n,d}$ *curve*[a] *over* F_q *with* $g^{\frac{1}{3}} \leq n, d \leq g^{\frac{2}{3}}$	$L_{q^g}\left[\frac{1}{3} + o(1), O(1)\right]$ *by a variant of the index calculus algorithm*[b] *[18].*
$\mathrm{Jac}(C)$	*C is a* $C_{n,d}$ *curve.*	*The complexity depends on the parameters* p, n *and d. As of 2025, there is no closed formula in the literature*[c].
$\mathrm{Jac}(C)$	*C is a hyperelliptic curve of genus g over* $\mathbb{F}_q$.	$O\left(g^5 (\ln q)^3 q^{2-\frac{2}{g}}\right)$ *for* $g < \ln q$ *[25]* $L_{q^g}\left[\frac{1}{2}, \sqrt{2} + o(1)\right]$ *for large enough genus* g^d *[17], [19]*
$\mathrm{Jac}(E)$	*E is a generic elliptic curve over* $\mathbb{F}_q$ *without known trapdoors.*	$O\left(q^{\frac{1}{2}}\right)$ *by Algorithm 1.1.5 or Algorithm 1.1.7*

[a] A definition of these curves will be given in 1.2.28
[b] The algorithms will be described in the subsequent sections
[c] Proper algorithms and their complexity for these discrete logarithm problems are subject of this thesis
[d] The exact conditions are given in [19] and will be discussed later in chapter 2

The complexity class for solving discrete logarithm problems depends on the underlying group. Jacobians of elliptic curves are among the remaining practical examples of hard-to-solve discrete logarithm problems with respect to the bit length of the group representation.

Therefore, these curves and the related hyperelliptic curves have been of particular scientific interest since the discovery of the quadratic sieve algorithm in 1984 [50]. Since most known exceptions to the complexity of the elliptic curve discrete logarithm problem are due to homomorphisms to other algebraic curves, which offer simpler solutions to the problem, these problems have also become more relevant for judging the security of the Diffie-Hellman scheme. The scope of this thesis is to investigate the complexity of the discrete logarithm problems for the wide class of $C_{n,d}$ curves. In doing so, we will also fill the gap between the two cited analyses for hyperelliptic curves.

In the next section, we will present some basic algorithms for solving the discrete logarithm problem over different groups. We will also provide insight into the reason for the different complexity classes.

1.1.2. Generic Solution Strategies for Discrete Logarithm Problems

In generic groups lacking additional structure to solve the DLP more efficiently, two classes of algorithms are employed. The first of these classes consists of algorithms that perform a direct search by forming multiples of the group generator, adding them to the target element, and checking for collisions.

The best known algorithm to solve the DLP in a generic group is the algorithm of Shanks [53], which is also known as the *Baby-Step, Giant-Step algorithm*.

Algorithm 1.1.5 (Shanks Baby-Step, Giant-Step [53])**.**
Input: Given a cyclic group $G = \langle\gamma\rangle$ of known order n and $\alpha \in G$.
Output: The discrete logarithm x such that $\alpha = x\gamma$.

1: Compute and store all pairs $(\alpha_i,\ i)$ with $\alpha_i \leftarrow \alpha + i \cdot \gamma$ and $0 \leq i \leq \lceil\sqrt{n}\rceil$.
2: Initialize $\gamma' \leftarrow \lceil\sqrt{n}\rceil\gamma$.
3: **for** $j \leftarrow 0$ *to* $\lceil\sqrt{n}\rceil$ **do**
4: Compute $\beta = j \cdot \gamma'$.
5: **if** $\exists$ stored $(\alpha_i,\ i)$ with $\alpha_i = \beta$ **then**
6: **return** $j\,\lceil\sqrt{n}\rceil - i$

7: **end if**

8: **end for**

Algorithm 1.1.5 is correct because the discrete logarithm has a representation of the form $j \cdot \lceil \sqrt{n} \rceil - i$ for some $0 < i,j \leq \lceil \sqrt{n} \rceil$ by using modular arithmetic. Unlike many other approaches, this makes the algorithm deterministic.

Theorem 1.1.6. *Algorithm 1.1.5 terminates after at most* $2\lceil \sqrt{n} \rceil$ *group operations and consumes* $\lceil \sqrt{n} \rceil$ *memory cells storing one group element each.*

The primary drawback of Algorithm 1.1.5 is its substantial memory consumption. Consequently, Pollard's Rho algorithm [49] is often preferred in practice due to its more memory-efficient implementation. While its average time complexity is identical to that of Algorithm 1.1.5, it has a higher worst-case complexity. This is a result of its operating principle, which is based on the birthday paradox and is therefore probabilistic.

Algorithm 1.1.7 (Pollard's ρ Algorithm [49])**.**

Input: Given a cyclic group $G = \langle \gamma \rangle$ of known order n and $\alpha \in G$.
Furthermore, let ν be a mapping $\nu : G \to \{1,\ 2,\ 3\}$

Output: The discrete logarithm x such that $\alpha = x\gamma$.

1: Select random $t \in_R \mathbb{N}$ and compute $\beta \leftarrow \alpha + t\gamma$.

2: Initialize $s \leftarrow 1$.

3: Store (β, s, t)

4: $u \leftarrow \nu(\beta)$

5: $$\beta \leftarrow \begin{cases} \beta + \gamma & u = 1 \\ 2\beta & u = 2 \\ \beta + \alpha & u = 3 \end{cases} \quad s \leftarrow \begin{cases} s & u = 1 \\ 2s & u = 2 \\ s + 1 & u = 3 \end{cases} \quad t \leftarrow \begin{cases} t + 1 & u = 1 \\ 2t & u = 2 \\ t & u = 3 \end{cases}$$

6: **if** $\exists$ stored $(\tilde{\beta},\ \tilde{s},\ \tilde{t})$ with $\beta = \tilde{\beta}$ **then**

7: **if** $(s - \tilde{s}), \in \mathbb{Z}_n^*$ **then**

8: **return** $(s - \tilde{s})^{-1}(\tilde{t} - t) \pmod{n}$

9: **else**

10: **Goto** 1.
11: **end if**
12: **else**
13: **Goto** 2.
14: **end if**

In its original variant, Pollard's ρ algorithm is neither superior to Shanks's algorithm in terms of memory consumption nor in its expected time complexity as shown in the following analysis.

Theorem 1.1.8. *In 50% of the cases, Algorithm 1.1.7 terminates in at most $\sqrt{n \log 2}$ iterations. Furthermore its expected memory consumption is approximately $\sqrt{n \log 2}$ memory cells with each storing an element of G and two integers less than n.*

Proof. The correctness of the result follows from a direct calculation. One central assumption is that the walk through the group introduced by ν is pseudo-random. Therefore, we expect the set of stored values of b to behave like a randomly chosen set. Under this assumption and Theorem A.2.1 the algorithm terminates before reaching $\sqrt{n \log 2}$ steps in at least 50% of all cases.

All intermediate results of the random walk are stored to detect a match. Therefore, $\sqrt{n \log 2}$ also provides an approximation of the required memory cells. For the sake of simplicity we assume that each cell holds one group element. □

To achieve the aforementioned advantage in memory consumption, a variant known as *Floyd's cycle-finding algorithm* [37, exercise 7] is used. The idea is that once the first collision occurs, every subsequent iteration also causes a collision due to the deterministic random walk.

Algorithm 1.1.9 (Floyd's algorithm)**.** Start two random walks $(b, s, t)_i$ and $(\tilde{b}, \tilde{s}, \tilde{t})_i$ at the same position. The stepping rule of the first walk will be chosen identically to Algorithm 1.1.7. In contrast, for the second walk we perform two iteration steps instead of only one. Store only those elements of the walk which are required for performing the next iteration.

Then there will be an $i > 0$ such that $b = \tilde{b}$. Then we can calculate the desired discrete logarithm by $(s - \tilde{s})^{-1}(\tilde{t} - t) \pmod{n}$.

Lemma 1.1.10. *Algorithm 1.1.9 is correct and terminates in fewer steps than Algorithm 1.1.7 requires.*

Proof. Assume that the random walk of Algorithm 1.1.7 will have required k steps. Then there was a pre-period of length l and a period of the walk of periodicity m, such that $l+m = k$, since Algorithm 1.1.7 terminated exactly after the completion of the first period.

In Algorithm 1.1.9, the first walk enters the period after l steps. At this point, the second walk is already further into the period and may have completed it multiple times. With each subsequent iteration, the distance between the second walk and the first decreases by one until they coincide. This happens after at most m steps. □

Floyd's cycle-finding variant requires at most three times the number of group operations compared to Pollard's original algorithm. This increased computational cost is a trade-off for eliminating the table searches required to detect a match, resulting in a significantly smaller memory footprint. Consequently, Floyd's variant is the most commonly used algorithm for discrete logarithms in generic groups.

Both Pollard's ρ and the Baby-Step, Giant-Step algorithm have been refined in various ways. For instance, Terr's algorithm [58] adapts the Baby-Step, Giant-Step method to groups of unknown order, while other modifications allow Pollard's ρ to be parallelized on distributed systems.

Although these improvements are highly relevant for practical implementations, they do not fundamentally alter the expected asymptotic complexity of solving the discrete logarithm problem in generic groups. To achieve subexponential performance in more specific groups, we instead focus on index calculus methods, which are introduced in the next section.

1.1.3. Index Calculus Solution Strategies for Discrete Logarithm Problems

Index calculus methods differ from generic algorithms by exploiting the specific algebraic structure of the underlying group. The formulation presented here follows the original approach by Kraitchik [42], which was later adapted by Adleman [1] into the modern framework used currently.

Algorithm 1.1.11 (Brief Function Principle of Index Calculus)**.** Let $G = \langle\gamma\rangle$ be a finite, additively-written cyclic group of known order n and $\alpha \in G$. Consider a subset $\mathcal{F} \subset G$ of size k that allows simple testing of whether a certain element $x \in G$ can be decomposed into elements of $\mathcal{F}$, i.e., there are $f_1, \ldots, f_k \in \mathcal{F}$ and integers $e_1, \ldots, e_k \in \mathbb{Z}$ such that

$$x = \sum_{i=1}^{k} e_i f_i.$$

We will refer to this decomposition as factoring x over the *factor base* $\mathcal{F}$.

Consider a random sequence of group elements (x_i) starting with γ. Further elements of the sequence are generated randomly by adding elements of $\mathcal{F}$ and powers of γ to the previous element. For each iteration, we attempt to factor x_i over the factor base. As soon as more than $\#\mathcal{F}$ factorizations are successful, one has at least $\#\mathcal{F}$ relations of form

$$z \cdot \gamma = \sum_{i=1}^{k} e_i f_i$$

for $z \in \mathbb{Z}$. Since $z = \sum_{i=1}^{k} e_i \log_\gamma f_i \pmod{n}$, finding sufficient relations permits solving the linear system by using one of the algorithms presented in Section A.4 to obtain all individual $\log_\gamma f_i$.

The second phase of the algorithm starts by initiating a second random sequence at α. Upon finding a sequence element that decomposes as

$$z'\alpha + z\gamma = \sum_{i=1}^{k} e_i' f_i,$$

one directly gets the discrete logarithm $\log_\gamma \alpha = z'^{-1} \left(\sum_{i=1}^{k} e'_i \log_\gamma f_i - z \right) \pmod{n}$.

We will cite three variants of Algorithm 1.1.11 that differ in the form of the linear equation to be solved. The first alternative uses a kernel finding algorithm instead of solving a general linear equation. We adopt the notation of Enge and Gaudry [17]. In the cited paper, the authors introduce and analyze this variant for multiple groups.

Algorithm 1.1.12 (Index Calculus Alternative I)**.** In this variant, the sequence is replaced by searching for solutions of the form

$$z'_j \, \alpha + z_j \, \gamma = \sum_{i=1}^{k} e_{i,j} \, f_i$$

for multiple randomly chosen $z_j, z'_j \in \mathbb{Z}_n$. Then it is sufficient to find a single kernel vector v satisfying $(e_{i,j})\, v = 0$. The coefficients of v can then be used to obtain the equation

$$\bar{z}' \alpha + \bar{z} \gamma = 0$$

for some $\bar{z}, \bar{z}' \in \mathbb{Z}$ by combining the found relations with weights given by v. If $\bar{z}' \in \mathbb{Z}_n^*$, we have succeeded in finding the discrete logarithm analogous to the original approach.

While searching for a kernel vector can reduce the arithmetic cost of solving the linear system, it has a significant drawback compared to Algorithm 1.1.11: it does not support solving multiple DLPs without rerunning the entire relation search.

In contrast, the variant proposed by Vollmer [63] overcomes this limitation. It enables the solution of multiple instances by allowing both the generator and the target element to be exchanged without recalculating the relations.

Algorithm 1.1.13 (Index Calculus Alternative II)**.** In this variant the randomly generated relations are of structure

$$\sum_{i=1}^{k} e_{i,j} \, f_i = 0.$$

This means we search for elements in the factor base that combine in a non-trivial

way to the group's neutral element.

Elements of finite groups are frequently represented by pre-images in an infinite group, particularly when working with residue class groups. This representation offers two distinct strategies for finding the relations required by this index calculus variant.

The first strategy involves trial division of randomly generated elements known to lie in the same residue class as the group's neutral element (zero). The second strategy seeks elements that admit two distinct decompositions over the factor base. Typically, one starts with a trivial decomposition over $\mathcal{F}$ – often constructed by summing factor base elements – and then switches to a different representative from the same residue class. Performing trial division on this new representative yields a second, distinct decomposition of the same group element.

Once these general relations are established, it suffices to find two special relations of the form

$$z' \, \alpha = \sum_{i=1}^{m} e_{i,\alpha} \, f_i \text{ and}$$
$$z \, \gamma = \sum_{i=1}^{m} e_{i,\gamma} \, f_i$$

Now let $A = (e_{i,j})$ and A' be the extended relation matrix

$$A' := \begin{pmatrix} 0 & 1 & \overrightarrow{0} \\ \overrightarrow{e_{i,\gamma}} & \overrightarrow{e_{i,\alpha}} & A \end{pmatrix}.$$

Assume that $x = (x_i)$ is the solution of the linear system $Ax = v$ with $v = (1, 0, \ldots, 0)'$. Then the discrete logarithm $\log_\gamma \alpha$ equals $x_1 = v^t x \pmod{n}$.

Algorithm 1.1.13 is the basis for one of the implementations in this thesis. Therefore, we sketch a proof of the correctness of this algorithm.

Theorem 1.1.14. *Algorithm 1.1.13 is correct.*

Proof. Assume that a solution x for the linear system $A'x = v$ is given and define $s := v^t x$

(mod n). Then for the matrix

$$A'' := \begin{pmatrix} 1 & 0 & \overrightarrow{0} \\ & A' & \end{pmatrix},$$

$A''x = (s, 1, 0 \ldots, 0)^t \pmod{n}$ holds. Each column of A'' represents a relation over the extended factor base $\mathcal{F} \cup \{\alpha, \gamma\}$. Therefore, we obtain $-s\gamma + \alpha = 0$ in G due to the structure of the first two columns of A''. Consequently, $\alpha = s \cdot \gamma$ holds in the group, verifying the correctness of the returned value. □

A primary advantage of Algorithm 1.1.13 is that — with the exception of the two specific relations — it places no constraints on the number of factors or their coefficient values, and the relations contain no divisors beyond the factor base elements. This flexibility allows the factor base elements to be strategically selected, enabling the use of optimized generation methods that significantly reduce computational effort. We will present these specific methods for our target finite groups in Chapter 3.

One further improvement over the previous method can be achieved when we calculate the individual logarithms of the factor base elements. Then we are able to compute multiple discrete logarithms without solving a new linear system. Therefore, certain implementations of discrete logarithm linear system solving, e.g., [57], use the following variation.

Algorithm 1.1.15 (Index Calculus Alternative III)**.** As in Algorithm 1.1.13 let the general relations be of the form

$$\sum_{j=1}^{m} e_{i,j} f_j = 0.$$

Also, we require two special relations

$$z'\alpha = \sum_{j=1}^{m} e_{\alpha,j} f_j \text{ and}$$
$$z\gamma = \sum_{j=1}^{m} e_{\gamma,j} f_j.$$

Then each solution (x_j) to the linear system

$$\begin{pmatrix} \overrightarrow{e_{\gamma,j}} \\ (e_{i,j}) \end{pmatrix} \cdot (x_j) = \begin{pmatrix} z \\ \overrightarrow{0} \end{pmatrix}$$

with $z \neq 0 \pmod{n}$ consists of the individual discrete logarithms $x_j = \log_\gamma f_j$.

Therefore, we can compute the discrete logarithm of α by the formula

$$\log_\gamma \alpha = \frac{1}{z'} \sum_{j=1}^{m} e_{\alpha,j} \log_\gamma f_j \pmod{n},$$

where n is the known order of the group.

All variants of index calculus have in common that each of these algorithms essentially consists of two phases. First, the search for relations over a factor base and the need to solve a linear system, which is often referred to as *linear algebra phase*. In Lemma 1.1.16, we balance these two phases for optimal performance.

Lemma 1.1.16. *Assume the group order n is known. Let $\mathcal{F}$ be a factor base of size $X \geq 1$ and $T(\mathcal{F})$ the average effort to find a single smooth element over the factor base. This effort is the product of the arithmetic cost of creating and trial dividing group elements over $\mathcal{F}$ with the expected probability of an element created by the method being $\mathcal{F}$-smooth.*

Then the runtime of Algorithms 1.1.11 to 1.1.13 is optimal with respect to the factor base size as a parameter, when $X = O(T(\mathcal{F}))$.

Proof. The runtime of the linear algebra phase is given by $O(X^2)$ by Theorem A.4.14, when carried out over the finite ring $\mathbb{Z}/n\mathbb{Z}$. For the rest of the proof we drop the O operator for readability.

In the context of A.4.14, we assume there is a constant $\Delta > 0$ giving the average number of non-zero entries per matrix column. We acknowledge that Δ may depend on the specific algorithm used to create the group elements that are then decomposed into factor base elements, but we assume that a change in X has no significant impact on Δ.

Because of increasing smoothness probability we can assume that $T(\mathcal{F})$ is increasing for

decreasing X and vice versa. Therefore, the two phases are balanced if $X \cdot T(\mathcal{F}) = X^2$ and thus $X = T(\mathcal{F})$. This is the minimum of the combined complexity function. □

Remark 1.1.17. *To solve the linear system over $\mathbb{Z}_n$, the variants listed in this section depend on the exact group order. If only an approximation is known, the system must instead be treated as a $\mathbb{Z}$-module and solved using its Smith Normal Form. This shift significantly impacts the complexity of the linear algebra phase, as it makes quasi-quadratic sparse matrix algorithms, such as the Wiedemann algorithm as defined in Appendix A.4 and used in the proof of Lemma 1.1.16, unsuitable.*

We have not yet addressed the *degree* of group elements, as the specific groups utilized in this thesis have not been introduced. Nevertheless, it is clear that different strategies for constructing elements to decompose over $\mathcal{F}$ impact both the smoothness probability and the number of factors in each relation. This, in turn, changes the parameter Δ in the linear algebra phase's complexity analysis.

Consequently, $X = O(T(\mathcal{F}))$ serves as an *approximation* for the optimal parametrization. A rigorous parameter balancing for our selected algorithms and the finite groups considered in this thesis is provided in Chapter 3.

1.2. Algebraic Curves

This section provides a brief introduction to function fields of algebraic curves and their Jacobian groups, which serve as the setting for the DLP algorithms discussed in the previous section. In Sections 1.2.1 and 1.2.2, we adhere to the notation and definitions established by Stichtenoth [55], while adapting the structure and certain theorems to suit our requirements. Proofs are included only when they are directly relevant to the algorithms developed subsequently.

1.2.1. Algebraic Function Fields

Definition 1.2.1. Let $K \subset F$ be a transcendental field extension. F is a *function field* over K if there is an $x \in F$ transcendental over K, such that F is a finite algebraic extension of the rational function field $K(x)$.

Remark 1.2.2. *([55, I.1.2]) The element x defining the underlying function field is not unique. Let F be a function field over K. Then any $x \in F$ transcendental over K satisfies $[F : K(x)] < \infty$.*

As the definition of function fields allows a variety of field constructions, the theory of function fields is comprehensive. We will focus on function fields arising from plane affine algebraic curves.

Definition 1.2.3. [55, B6] Let $f \in K[X, Y]$ be a bivariate polynomial. A *plane affine algebraic curve* is the set of points $P \in \overline{K}^2$ satisfying $f(P) = 0$.
If f is irreducible, $F := K[X, Y]/(f)$ is a field and a finite extension of $K(X)$ by identifying X with its residue class $(\mathrm{mod}\ F)$, thus F is a function field by Definition 1.2.1.

Remark 1.2.4. *Let $x := X \pmod f$ and $y := Y \pmod f$ be the corresponding residue classes of X and Y in F. Then $F = K(x, y)$ and $K(x) \simeq K(X)$.*

The field extension $K(x) \subseteq F$ is not necessarily generated by additional elements transcendental over K. Let $\tilde{K}$ be the algebraic closure of K in F. Then there is a tower of fields $K \subseteq \tilde{K} \subset F$. The field $\tilde{K}$ is called the *exact field of constants* of F. For this thesis we will focus on those function fields and base fields satisfying $K = \tilde{K}$, so K is the exact field of constants.

Central objects of interest are the subrings of F.

Definition 1.2.5. A *valuation ring* $\mathcal{O}$ of $K \subset F$ is a ring satisfying $K \subsetneq \mathcal{O} \subsetneq F$ such that for all $x \in F^*$

$$\{x, x^{-1}\} \cap \mathcal{O} \neq \emptyset.$$

The most important properties of valuation rings are listed in the following lemma.

Lemma 1.2.6 ([55], I.1.5 and I.1.6)**.** *Let $\mathcal{O}$ be a valuation ring of $K \subset F$, then*

1. *$\mathcal{O}$ is a local ring with unique prime ideal $P = \mathcal{O} \setminus \mathcal{O}^*$.*
2. *$\mathcal{O}$ is a principal ideal domain. In particular, there is a prime element $p \in P$ such that $(p) = P$.*
3. *For all $z \in F^* : z \in P \Leftrightarrow z^{-1} \notin \mathcal{O}$.*

Definition 1.2.7. A *place* of the function field $K \subset F$ is the maximal ideal of a valuation subring of K. The set of all places of $K \subset F$ is denoted by $\mathbb{P}_F$.

Remark 1.2.8. *For a fixed place P with ideal generator p, each element a of its valuation ring $\mathcal{O}$ has a unique representation $a = \epsilon \cdot p^\delta$ with $\epsilon \in \mathcal{O}^*$. This induces a map $v_P : \mathcal{O} \mapsto \mathbb{N}_0$, $a \to \delta$. We will refer to the map as the valuation given by P.*

The purpose of valuation rings and places is not self-evident from their definitions alone. In fact, these algebraic structures represent the geometric points of the curve underlying our function field. We state this correspondence in the following lemma.

Lemma 1.2.9. *[55, I.1.15] Let $\mathcal{O}$ be a valuation ring of $K \subset F$ and P the corresponding place. Then $\mathcal{O}/P$ is a field and a finite extension of K.*

Because x and y satisfy $f(x, y) = 0$, any valuation ring $\mathcal{O}$ with $x, y \in \mathcal{O}$ gives rise to a point $Q = (x', y')$ on the curve with its coordinates being in the algebraic closure $\overline{K}$ of K. The points are derived by reducing $x' = x \pmod P$ and $y' = y \pmod P$.

Conversely, for every $Q = (x', y') \in \overline{K}^2$ with $f(Q) = 0$ there is a place $P \in \mathbb{P}_F$ with corresponding valuation ring $\mathcal{O}$ satisfying $x' = x \pmod P$, $y' = y \pmod P$.

For later use, we define $\deg(P) := [\mathcal{O}/P : K]$ to be the *degree* of P.
The following two theorems state that the set of all places is the union of the set of places arising from the points on the algebraic curve plus the set of places with $x^{-1} \in P$.

Theorem 1.2.10 ([55], I.2.2). *Let $K(x)$ be a rational function field. Then*

1. *For each monic, irreducible polynomial $p \in K[x]$ the set*

$$\mathcal{O}_p := \left\{ \frac{f}{g} \;\middle|\; f,g \in K[x], p \nmid g \right\}$$

is a valuation ring with corresponding place

$$P_p := \left\{ \frac{f}{g} \;\middle|\; f,g \in K[x], p \nmid g,\ p \mid f \right\}.$$

2. *Another valuation ring is*

$$\mathcal{O}_\infty := \left\{ \frac{f}{g} \;\middle|\; f,g \in K[x], \deg(f) \leq \deg(g) \right\}$$

with corresponding place

$$P_\infty := \left\{ \frac{f}{g} \;\middle|\; f,g \in K[x], \deg(f) < \deg(g) \right\}.$$

3. *The set of all places of the rational function field is given by*

$$\mathbb{P}_{K(x)} = \{ P_p \mid p \in K[x] \text{ prime} \} \cup \{P_\infty\}$$

The above-mentioned composition of the set of places is due to an extension theorem that is comparable to the theory of prime elements splitting in number fields. The similarity of function fields over finite fields and number fields play an important role for our algorithms. We will discuss these properties in sections 1.2.4 and 1.2.2.

Definition 1.2.11. Let F be a function field with constant field K and $F \subset F'$ be a finite algebraic extension. We define a place $P' \subset F'$ to lie over a place $P \subset F$ if $P \subseteq P'$. A common notation for this situation is $P' \mid P$.

Theorem 1.2.12 ([55], III.1.7). *Let F be a function field with exact field of constants K and $F \subset F'$ be a finite algebraic extension. Then*

1. *For each place $P' \subset F'$ there is exactly one place $P \subset F$ with $P' \mid P$.*

2. *For each place $P \subset F$ there are finitely many places $P' \subset F'$ lying over P. Furthermore, the set of all places $P' \subset F'$, $P' \mid P$ is non-empty.*

It is important to note that the distinction between finite places — those arising from curve points defined over the algebraic closure of K — and those at infinity is an artifact of the selected affine model of the curve. If we instead consider the projective model of the curve, the 'places at infinity' correspond simply to the points lacking a direct affine representation.

Furthermore, the theory of function field extensions parallels algebraic number theory; specifically, the decomposition of places behaves analogously to the decomposition of prime ideals in number fields.

Definition 1.2.13 ([55], III.1.5)**.** Let F be a function field with constant field K and $F' \mid F$ be a finite algebraic extension.
For $P' \in \mathbb{P}_{F'}$, $P \in \mathbb{P}_F$ with $P' \mid P$ we define

1. $f(P', P) := [\mathcal{O}_{P'}/P' : \mathcal{O}_P/P]$ to be the *relative degree* of P' over P
2. the integer $e(P', P)$ satisfying $v_{P'}(x) = v_P(x)$ for all $x \in F$ to be the *ramification index* of P' over P.

Both integers are well-defined and exist, but their proof is beyond the scope of this thesis. Using these terms, the following theorem holds.

Theorem 1.2.14 ([55], III.1.11)**.** *Let F be a function field with constant field K and $F \subset F'$ be a finite algebraic extension. Then for each $P \in \mathbb{P}_F$:*

$$\sum_{P'|P} e(P', P) \cdot f(P', P) = [F' : F].$$

1.2.2. The Divisor Class Group of Function Fields

Definition 1.2.15. Let F be a function field with exact field of constants K and set of places $\mathbb{P}_F$. A *divisor* D is a formal sum

$$D = \sum_{P \in \mathbb{P}_F} n_P P$$

with $n_P \in \mathbb{Z}$ and $n_P = 0$ for all but finitely many places $P \in \mathbb{P}_F$.

The set of all divisors in a given function field is denoted by Div.

We define addition of divisors via component-wise addition of their coefficients, which gives the divisor group the structure of a $\mathbb{Z}$-module. The degree of a divisor is defined analogously by summing the degrees of the places appearing in the support, weighted by their coefficients.

Definition 1.2.16. The *degree* of a divisor $D = \sum_{P \in \mathbb{P}_F} n_P P$ is defined by

$$\deg D := \sum_{P \in \mathbb{P}_F} n_P \deg(P).$$

The degree of a divisor is always finite by Definition 1.2.15 and Lemma 1.2.9.

We define divisors corresponding to elements of the function field F by using the valuation given for each place.

Theorem 1.2.17. *[55, I.4.3] Let F be a function field and let $x \in F^*$. Then*

1. *the formal sum $(x) := \sum_{P \in \mathbb{P}_F} v_P(x) P$ is a divisor.*
2. *$\deg(x) = 0$.*
3. *$(x) = 0 \Leftrightarrow x \in K^*$, where K is the exact field of constants of F.*

Definition 1.2.18. Let F be a function field. The set $\{(x) \mid x \in F^*\}$ is called the set of *principal divisors*.

This set forms a subgroup of the group of degree-zero divisors, denoted by Div_0,

because the map $x \mapsto (x)$ is a homomorphism: multiplying elements in the function field corresponds to adding their associated divisors.

Definition 1.2.19. [55, B.10] Let F be a function field with group of degree zero divisors Div_0. The factor group

$$Jac(F) := \mathrm{Div}_0 / \{(x) \mid x \in F^*\}$$

is called the *Jacobian* of F.

The Jacobian group of an algebraic curve serves as the setting for the discrete logarithm problems discussed in this thesis. The following definition and the subsequent theorems are central to understanding the structure and size of this group.

Definition 1.2.20. We define a divisor $D = \sum_{P \in \mathbb{P}_F} n_P P$ to be *effective* if $n_P \geq 0$ for all $P \in \mathbb{P}_F$.

Theorem 1.2.21. *[55, Proposition I.4.9 and Proposition I.4.14] Let D be a divisor over a function field F. The set*

$$\mathcal{L}(D) := \{x \in F \mid D + (x) \text{ is effective}\} \cup \{0\}$$

is a finite-dimensional K-vector space. Moreover, there is a $\gamma \in \mathbb{Z}$ depending on F such that

$$deg\, D - \gamma \leq \dim_K \mathcal{L}(D) \leq \deg\, D + 1.$$

Theorem 1.2.21 leads to the most important invariant of algebraic curves.

Definition 1.2.22. We define the *genus* of a function field $K \subset F$ to be the integer

$$g := \max_{D \in \mathrm{Div}} \deg D - \dim_K \mathcal{L}(D) + 1.$$

If the base field K is finite, then the Jacobian group is also finite. Furthermore, the actual size of the Jacobian group can be estimated using the Hasse bound.

Theorem 1.2.23. *[55, Theorem V.2.3] Let F be a function field of genus g with exact field of constants $\mathbb{F}_q$. Then*

$$| \#Jac(F) - q^g + 1 | \leq 2g\sqrt{q}.$$

This range for the size of the Jacobian is known as the Hasse bound.

The case $g = 1$ corresponds to *elliptic curves*, which form the basis of standard cryptographic schemes like Diffie-Hellman 1.1.3. While generic elliptic curves are considered secure, this assumption does not hold for curves of larger genus, where the structure allows for more efficient methods to solve the DLP.

To rigorously analyze these index calculus algorithms and their complexity, we must first be able to compute explicitly within the Jacobian. Since the Jacobian is an abstract quotient group, efficient computation requires a concrete representation of its elements. To establish a suitable representation for computations, we first investigate methods for mapping the full divisor class group $\mathrm{Div}/\{(x) \mid x \in F^*\}$ into the Jacobian.

Definition 1.2.24. Let F be a function field with exact field of constants K.

Let $P \in \mathbb{P}_F$. Then we define $[P]$ to be the *divisor class* of P in $\mathrm{Div}/\{(x) \mid x \in F^*\}$.

Let P' be a fixed divisor of degree one. Then the map

$$P \mapsto [P - \deg(P)P']$$

induces a homomorphism

$$\mathrm{Div}/\{(x) \mid x \in F^*\} \to Jac(F).$$

To implement this mapping, the function field must possess a divisor of degree one. For computational efficiency, it is advantageous if this place lies at infinity, allowing for a rational representation of the remaining places. As we can map any degree-one place to infinity by a suitable change of coordinates, we will assume the existence of such a rational place in the remainder of this work.

With this assumption in place, we now examine the kernel and image of the mapping. The subsequent theorem is crucial, as it guarantees that divisors of degree g are sufficient to represent every class in the Jacobian.

Theorem 1.2.25. *Let D' be an effective divisor of degree g. Then for each $D_0 \in \mathrm{Div}_0$ there is an effective divisor D of degree g satisfying $[D - D'] = [D_0]$ in the Jacobian of F.*

Proof. Let $D_0 \in \mathrm{Div}_0$. Then by Theorem 1.2.21 and the definition of g 1.2.22 with $\gamma := g-1$ we know that

$$\dim_K \mathcal{L}(D_0 + D') \geq g - g + 1 = 1.$$

Therefore, there is a $z \in \mathcal{L}(D_0 + D')$ such that $D := D_0 + D' + (z)$ is an effective divisor. This completes the proof, because $[D - D'] = [D_0]$ since (z) is a principal divisor. □

Remark 1.2.26. *The definition of the genus ensures that for each element in the Jacobian there is an effective divisor of degree g in the preimage of the map along any D'.*

If the function field has a rational place P, i.e., a place of degree one, we may specialize this representation by setting $D' = kP$ with $k \leq g$, allowing us to represent the class using an effective divisor D of degree k. A key advantage of minimizing k is that it ensures the supports of D and D' are disjoint.

In this setting — which we assume throughout this work — D' has support only in those places extending the place P_∞ of the rational function field.

Definition 1.2.27. The *affine component* of a divisor $D = \sum_{P \in \mathbb{P}_F} n_P P$ is the divisor

$$D' = \sum_{P \in \mathbb{P}_F} n'_P P$$

with $n'_P = 0$ for all $P \mid P_\infty$ and $n'_P = n_P$ otherwise.

In the next section, we introduce one class of curves that satisfies the condition of having a unique place extending the place at infinity. Therefore, each divisor class is uniquely represented by the affine component of a divisor of degree less than or equal to g.

1.2.3. $C_{n,d}$ Curves and their Properties

One important subclass of algebraic curves is $C_{n,d}$ curves. We use these curves later for our algorithms and for a detailed complexity analysis. In this section, we will briefly introduce these kinds of curves and some of their properties.

Definition 1.2.28. A $C_{n,d}$ curve is an algebraic curve defined by

$$y^n + x^d + \sum_{jn+di<nd} c_{i,j} y^i x^j$$

with $c_{i,j} \in \mathbb{F}_q$ and $\gcd(n,d) = 1$.

Remark 1.2.29. *Without restriction, we assume $d > n$ throughout this thesis. For curves that do not satisfy this condition, we achieve it by swapping x and y in the curve equation. The swapping affects neither the genus nor other characteristics of the curves.*

Prominent subclasses of $C_{n,d}$ curves include imaginary model hyperelliptic curves — with $n = 2$ and odd d — and superelliptic curves with $n > 2$ and $gcd(n,d) = 1$. These curves serve as a natural testing ground for our algorithms, as they share many structural properties with the elliptic case while offering higher genus environments.

The class of $C_{n,d}$ curves allows a simple determination of the genus and offers a single ramified divisor above P_∞.

Lemma 1.2.30 (Gaudry, Enge and Thomé; Section 2 of [18])**.** *Given a $C_{n,d}$ curve of genus g, then $g = \frac{(d-1)(n-1)}{2}$. Furthermore, P_∞ is totally ramified, i.e., there is a unique divisor above infinity.*

The utility of the $C_{n,d}$ model is underscored by the fact that it effectively covers all curves with a totally ramified point. As the following theorem demonstrates, any algebraic curve featuring such a place can be transformed into a $C_{n,d}$ representation.

Theorem 1.2.31 (Appendix B and lemma on page 1416 [44]). *Let C be an absolutely irreducible algebraic curve defined over a field K, where K is the exact field of constants. If C has a rational place P that is totally ramified over the rational function field $K(x)$, then C is birationally equivalent to a $C_{n,d}$ curve.*

Furthermore, if P corresponds to a finite point in a given model of C, there exists a birational coordinate transformation — specifically, a fractional linear transformation of x — that maps P to the unique place at infinity, yielding the standard $C_{n,d}$ equation form.

We note that Theorem 1.2.31 is the exact equivalent to the condition allowing one to transfer certain real model elliptic- and hyperelliptic curves into their imaginary model representation.

1.2.4. Number Theoretic Aspects of Algebraic Curves

Besides the divisor-related point of view, we will investigate function fields over a finite field by applying methods from algebraic number theory. The algebraic properties are useful for factoring elements of the Jacobian over a previously defined factor base. Also, we exploit the algebraic properties described in this section for building efficient versions of the algorithms presented in Section 1.1.3.

As before, we assume that a function field $K \subset F$ arises from a smooth algebraic curve with K being the field of constants of F.

Definition 1.2.32. Let $K \subset F = K(x, y)$ be a function field arising from an algebraic curve. We define the ring $K[x, y] \subset F$ to be the *coordinate ring* of F.

Lemma 1.2.33. *Given a function field $F = K(x, y)$ where K is the exact field of constants. Then $\mathcal{O} := K[x, y]$ is the integral closure of $K[x]$ in F.*

The proof of Lemma 1.2.33 is trivial, because y is integral by the definition of the algebraic curve and thus every linear combination is. Also, there are no further integral elements in F since K is the field of constants of F.

The coordinate ring has similar properties to the ring of integers of algebraic number fields.

Theorem 1.2.34. *Let K be a finite field and $K \subset F$ be a function field with exact field of constants K. Then the coordinate ring $\mathcal{O}$ is a Dedekind domain. Thus, any ideal in $\mathcal{O}$ has a unique factorization into prime ideals.*

Proof. We know that $K[x]$ is a principal ideal domain and F is a finite-dimensional extension of the quotient field $K(x) = \mathrm{Quot}(K[x])$. Thus, the integral closure $\mathcal{O}$ of $K[x]$ in F is a Dedekind domain by the definition of Dedekind domains and the Krull–Akizuki theorem [29, theorem 4.9.2]. □

The set of places $\mathbb{P}_F$ not extending the infinite place is closely related to the ideals of $\mathcal{O}$. In fact, each affine place directly corresponds to a prime ideal of $\mathcal{O}$.

Theorem 1.2.35. *Let K be a finite field and $K \subset F$ be a function field with exact field of constants K. Let $P \in \mathbb{P}_F$ with $P \nmid P_\infty$. Then there is a prime ideal $I \subset \mathcal{O}$ such that $\mathcal{O}_P$ is the localization $\mathcal{O}_I$. Each prime ideal of $\mathcal{O}$ gives rise to a discrete valuation over F.*

Proof. We will start with the second statement. Let $z \in F$, then the fractional principal ideal (z) has a unique prime ideal factorization with exponents taken from $\mathbb{Z}$, because $\mathcal{O}$ is a Dedekind domain.

Therefore, for every prime ideal $I \subset \mathcal{O}$, we define the valuation $v_I(z)$ to be the exponent of I in the ideal factorization of (z). With this definition

$$\mathcal{O}_I := \{z \in F \mid \ v_I(z) \geq 0\}$$

satisfies all conditions to be a valuation ring within F. We conclude that the corresponding place is affine because of Theorem 1.2.12 and since $\mathcal{O}$ is the ring of integers in F.

Additionally, we rewrite $\mathcal{O}_I$ to be

$$\mathcal{O}_I = \{r \cdot s^{-1} \mid \ r, s \in \mathcal{O}, s \notin I\},$$

so it is equal to the localization of $\mathcal{O}$ corresponding to I.
Conversely, if $P \in \mathbb{P}_F$ is affine, then $P \cap \mathcal{O}$ is a non-empty ideal of $\mathcal{O}$. Due to our construction, $I := P \cap \mathcal{O}$ is a prime ideal. The valuation corresponding to I now satisfies our requirement. □

Remark 1.2.36. *Mapping prime ideals to places induces a map from the set of all ideals to the set of affine divisors, since $\mathcal{O}$ is a Dedekind domain. We will later use this map to establish an effective way to generate semi-reduced divisors with predefined properties.*

The embedding of principal ideals into the Jacobian is trivial.

Lemma 1.2.37. *Let K be a finite field and $K \subset F$ be a function field with exact field of constants K. Let $P \in \mathbb{P}_F$. Then there is an element $z \in F$ satisfying*

$$v_P(z) = 1, \text{ and}$$
$$v_{P'}(z) = 0, \ \forall P' \in \mathbb{P}_F, P' \neq P, P' \nmid P_\infty$$

if and only if the ideal $P \cap \mathcal{O}$ is principal.
Furthermore, if the condition is met, the affine components of the principal divisor of z equal the affine component of P.

Proof. Assume that $I = P \cap \mathcal{O}$ is principal. Then there is an element $z \in \mathcal{O}$ satisfying $(z) = I$. Furthermore, $\mathcal{O}_P$ is equal to the localization of $\mathcal{O}$ to the prime ideal I. Therefore, we know that the image of z generates P in $\mathcal{O}_P$ and thus $v_P(z) = 1$.

For any other place $P' \nmid P_\infty$, we know that the ideal $J := P' \cap \mathcal{O}$ is comaximal to I. Therefore, z is a unit in $\mathcal{O}_{P'}$; thus $v_{P'}(z) = 0$.

On the other hand, if we assume there is a $z \in \mathcal{O}$ satisfying $v_P(z) = 1$ for a place $P \nmid P_\infty$ and $v_{P'}(z) = 0, \ P' \in \mathbb{P}_F, P' \neq P, P' \nmid P_\infty$. Since all prime ideals of $\mathcal{O}$ correspond to affine places, we know that $(z) = P \cap \mathcal{O}$, because the principal ideal generated by z has a unique prime ideal factorization. All other prime ideals J do not divide (z) since z is a unit for all affine places except for P. This proves that $(z) = P \cap \mathcal{O}$ is principal. □

Lemma 1.2.37 proves that only affine divisors, which are not divisible by any principal factor, are necessary to describe the elements of the Jacobian up to the places above P_∞. We will define a homomorphism from the *ideal class group* — which is a well-investigated object in algebraic number theory — into the Jacobian.

Definition 1.2.38. Let K be a finite field and $K \subset F$ be a function field with exact field of constants K. Let $J(F)$ be the multiplicative group of fractional ideals of F, i.e., the set of submodules M such that there is an $r \in K[x,y]$ such that rM is an ideal of $K[x,y]$. Further, let $P(F)$ be the group of all fractional principal ideals. Then we define the quotient group

$$Cl(F) := J(F)/P(F).$$

to be the *ideal class group* of $K[x,y]$.

By Lemma 1.2.37, the divisors corresponding to principal ideals are equal to the affine component of a principal divisor. Therefore, we use reduced representatives of the ideal classes for mapping ideals to divisors. Each ideal is equivalent to a primitive or a semi-reduced ideal in $Cl(F)$.

Definition 1.2.39. An ideal I is *semi-reduced* if it has no principal factor.

We identify semi-reduced ideals by their decomposition into prime ideals.

Lemma 1.2.40. *Let K be a finite field and $K \subset F = K(x,y)$ be a function field with exact field of constants K. Let I be a semi-reduced ideal of $K[x,y]$ factored into a product of prime ideals*

$$I = \prod P^{e_P}$$

with $e_P \in \mathbb{N}_0$, $e_P \neq 0$ for only finitely many prime ideals. Then the following two conditions hold:

1. *For a prime ideal $P \subset \mathcal{O}$, let $\widetilde{P} \subset K[x]$ be the unique prime ideal with $\widetilde{P} \subset P$. Define $S_P = \{P' \subset \mathcal{O}$ prime ideal satisfying $\widetilde{P} \subset P'\}$. Assume $n_P > 0$ for a prime P. Then there is at least one $P' \in S_P$ satisfying $n_{P'} = 0$.*

2. *If P^k is principal for an ideal P and integer $k > 0$, then $n_P < k$.*

Proof. Let I be a semi-reduced ideal. Then (2) obviously holds, since any higher power directly implies a principal factor divides I. Note that the condition also implies that $e_P = 0$ if P is itself principal.

Condition (1) holds, because otherwise I would be divisible by the product of all ideals in S_P. This product equals the principal ideal $\widetilde{P}K[x,y]$. □

Semi-reduced ideals allow a convenient construction of divisor class representatives with an effective affine component.

Theorem 1.2.41. *Let K be a finite field and F be a function field with exact field of constants K and group of divisors* Div. *Let $D \in$ Div of positive degree and with effective affine component. Then there is a divisor D' with effective affine component satisfying $[D'] = [D]$ and the affine components of D' correspond directly to a semi-reduced ideal.*

Proof. Given an effective divisor D, we can factor the ideal I corresponding to the affine component of D into a product of a semi-reduced ideal J and a principal ideal H. Note that all powers of factors are positive, because D is effective. Therefore, $J, H \subset \mathcal{O}$. By applying Lemma 1.2.37 to the prime factorization of H, it is immediate that there is an element $z \in F$ such that the affine component of the divisor (z) corresponds to H.

We conclude that $[D] = [D]-[(z)] = [D-(z)]$. With $D' := D-(z)$, the affine component of D' corresponds to J. Since $J \subset \mathcal{O}$, we obtain that D' is effective. This completes the proof. □

For later use in our algorithms, we also have to introduce the relative norm of ideals in the function field over the rational function field in x.

Definition 1.2.42. Let J be a prime ideal of $\mathcal{O}_{K(x,y)}$ lying over the prime ideal $I \subset \mathcal{O}_{K(x)} = K[x]$. Then we define the *relative norm* of I by

$$N_{K(x)\subset K(x,y)}(J) := I^{\left[\mathcal{O}_{K(x,y)}/J \,:\, \mathcal{O}_{K(x)}/I\right]}.$$

We know that $\mathcal{O}_{K(x,y)}$ is a Dedekind domain by Theorem 1.2.34. Therefore, the defini-

tion extends to products of ideals in a way that N is a multiplicative morphism on the set of ideals.

Remark 1.2.43. *Since $K[x]$ is a principal ideal domain, this gives an easy-to-use criteria for divisibility of ideals in $\mathcal{O}_{K(x,y)}$. Given two ideals $I, J \subset \mathcal{O}_{K(x,y)}$ then I divides J implies that the polynomial generating $N(I)$ divides the polynomial generating $N(J)$.*
For prime polynomials, even more is true. Let J be an ideal of $\mathcal{O}_{K(x,y)}$ such that there is a prime polynomial $p \in K[x]$ dividing $N(J)$. Then there is an ideal $I \subset \mathcal{O}_{K(x,y)}$ such that

$$I \mid p\mathcal{O}_{K(x,y)} \text{ and } I \mid J.$$

This equivalence between polynomial division and divisor decomposition will be of algorithmic use for the discrete logarithm solving algorithms discussed later in this thesis. Finally, we require that the generator of the relative norm of a principal ideal equals the field norm of the ideal generator. Therefore, field and ideal norm are compatible.

Lemma 1.2.44 (Chapter I, Proposition 14, [52])**.** *Given an element $z \in \mathcal{O}_{K(x,y)}$, the norm of the principal ideal $z\mathcal{O}_{K(x,y)}$ is equal to $\mathcal{N}(z)K[x]$ where $\mathcal{N}(z)$ is the field norm of z.*

1.2.5. Divisor Representation and Decomposition

By embedding the ideals of the ideal class group of the function field into the Jacobian of the algebraic curve, we obtain a divisor representation that is well-suited for arithmetic operations. First, we focus on divisors whose norm is a prime polynomial. Since the principal ideals are only mapped to the neutral element, those prime polynomials are particularly important when they split into non-principal ideals upon extension to the function field, i.e., when their associated principal ideals are divided by new prime norm ideals.

Theorem 1.2.45. *Let $\mathbb{F}_q(x) \subset F$ be a function field extension of the rational function field $\mathbb{F}_q(x)$ with exact field of constants $\mathbb{F}_q$. Assume that the extension arises from an algebraic curve defined by an irreducible equation $f(x, y) = 0$. Let $p \in \mathbb{F}_q[x]$ be a prime polynomial and $b \in \mathbb{F}_q[x]$, $\deg b < \deg p$ such that $f(x, b) \equiv 0 \pmod{p}$.*

Then the ideal generated by p and $(y-b)$ divides the ideal $p\mathbb{F}_q[x,y]$.

Proof. We adopted this theorem from a well-known result in algebraic number theory. Specifically, let $R \subset S$ be an extension of Dedekind domains, and let y be an element of the conductor with minimal polynomial f, where $f = \prod f_i \pmod p$ for a prime $p \in R$. Additionally, assume that the smallest ideal containing both pS and y is S itself. From this, we can conclude that the ideal pS is the product of all prime ideals $\langle p, f_i \rangle$.

In our case, the conditions for the conductor and y are trivially satisfied, as $\mathbb{F}_q$ is the exact field of constants and $\mathbb{F}_q[x,y]$ is the ring of integers of the function field F. Furthermore, it is clear that f is the minimal polynomial of y. Therefore, the existence of the pair (p,b) implies that one of the factors f_i is a linear coefficient of the form $y-b \in \left(\mathbb{F}_q[x]/p\right)[y]$. This completes the proof. □

This divisibility property allows us to represent every prime place of the function field with such a pair of polynomials.

Corollary 1.2.46. *Let P be an affine place in a function field of an algebraic curve defined in the variables x and y over $\mathbb{F}_q$. Assume the following conditions hold:*

1. *$x, y \in \mathcal{O}_P$, i.e., the image of x, y in the residue class field defined by P exists, and*
2. *$\deg P = \left[\mathbb{F}_q(\bar{x}) : \mathbb{F}_q\right]$ for $\bar{x}$ denoting the residue of x in the class field defined by P.*

Then there are $p, b \in \mathbb{F}_q[x]$ with $\deg(b) < \deg(p)$ and $f(x,b) \equiv 0 \pmod p$ such that the ideal generated by p and $y-b$ represents P.

Proof. The ideal corresponding to a place P is a prime ideal dividing a principal ideal $p\mathbb{F}_q[x,y]$ where p is a prime polynomial in $\mathbb{F}_q[x]$. By Theorem 1.2.45 we conclude that we can represent it as a pair of polynomials p and b with p a prime polynomial. The assertion of the degrees is true, because we assumed that the field extension of the factor group $\mathbb{F}_q[x,y]/P$ is of the same degree as p. Therefore, the root of the defining polynomial is already included in this field extension and can thus be represented by a polynomial degree less than $\deg(p)$. □

From the results for prime polynomials we can derive a similar assertion for ideals of composite norm.

Lemma 1.2.47. *Let F be a function field extension of the rational function field $\mathbb{F}_q(x)$ with exact fields of constants $\mathbb{F}_q$. Assume that the extension rises from a smooth algebraic curve defined by an irreducible equation $f(x,y) = 0$. Let $a \in \mathbb{F}_q[x]$ be a polynomial that splits in $\mathbb{F}_q[x,y]$ and whose prime factors are distinct in $\mathbb{F}_q[x]$. Further, let $b \in \mathbb{F}_q[x]$, $\deg b < \deg a$ such that $f(x,b) \equiv 0 \pmod{b}$.*

Then a divides the norm of the ideal $I = \langle a, (y-b)\rangle$ and $I \mid a\mathbb{F}_q[x]$. Thus I is semi-reduced.

Proof. Let $p \in \mathbb{F}_q[x]$ be a prime divisor of a. Then b is equivalent to a root $b' \pmod{p}$ by modular reduction. Therefore, $\exists\ c \in \mathbb{F}_q[x]$ such that $b = c \cdot p + b'$ and $p' \in \mathbb{F}_q[x]$ with $pp' = a$. Then

$$p\,(p' - c \cdot t)\,s + (y - b')t = as + (y - b)t$$

for any $s, t \in \mathbb{F}_q[x]$. We conclude that the ideal $\langle p, (y-b')\rangle$ divides the ideal $I := \langle a, (y-b)\rangle$. By Theorem 1.2.45, p divides the norm of I. Therefore the ideal itself is the product of the prime ideals of form $\langle p, (y-b)\rangle$, because a splits and we did not fix p from the set of primes dividing a. □

Remark 1.2.48. *Since the coordinate ring is a Dedekind domain, every ideal can be expressed as a product of prime ideals. Moreover, any prime polynomial p for which $f(x,b) \equiv 0 \pmod{p}$ has no solution b generates a principal ideal. Consequently, all ideals involved in the description of the Jacobian group can be represented in the two-element form defined in Lemma 1.2.47.*

It should be noted that it is not necessary for general divisors to factor into distinct prime divisors. In the case that the divisors are pairwise distinct, though, the appropriate value of the polynomial b can be calculated using the Chinese remainder theorem.

To formulate a discrete logarithm problem in the Jacobian group, we not only need a suitable representation of the group elements, but we also need to perform the arithmetic of these elements.

From the homomorphism defined in 1.2.24, we know that the group operation in the Jacobian group equals the addition of effective divisors and the multiplication of the associated ideals. We also know from Theorem 1.2.25 that we can choose a representative of degree less than or equal to g for each ideal class. However, finding this representative is not necessarily trivial.

For elliptic curves, many papers have been published on this topic, examining the group law to allow for the fastest possible arithmetic calculations depending on the application and the equation of the curve. Due to its relevance in practical applications, Bernstein and Lange set up an online database [5] as described in [4] to cover recent findings on formulas for the group law for the different forms in which elliptic curves can be represented.

While for almost all models of elliptic curves an explicit formula for the group law is known, this is no longer necessarily the case if we broaden our view to hyperelliptic curves or even $C_{n,d}$ curves. Nevertheless, with Cantor's algorithm [8], there is a long-known and relatively easy-to-use algorithm for hyperelliptic curves in their usual representation.

For arbitrary $C_{n,d}$ curves, the problem becomes more complex. In a paper by Harasawa and Suzuki [26], an arithmetic algorithm was examined that solves this problem for any curve of this very general type.

Theorem 1.2.49 (Theorem 3 and Algorithm 7 of [26]). *Let $\mathbb{F}_q[x,y]$ be the ring of integers corresponding to a $C_{n,d}$ curve and let I be an ideal of $\mathbb{F}_q[x,y]$.*

Then we say I is in its Hermite normal form if it is generated by elements $\beta_0 \dots \beta_{n-1}$, $\beta_i = \sum_{j=0}^{n-1} \beta_{i,j}(x)y^j$ and the matrix $(\beta_{i,j})$ is in Hermite normal form.
Let I_1 and I_2 be two ideals in their Hermite normal form.

Then there is an algorithm computing the minimal ideal I_3 equivalent to $I_1 I_2$ in the ideal class group and I_3 in Hermite normal form in an expected runtime of $\mathcal{O}\left(n^8 g^2 (\ln q)^2\right)$.

It is evident that the computational cost of the group operation increases rapidly with the curve degree n. While the optimal asymptotic complexity with respect to n remains an open research question, significant progress has been made regarding upper bounds. Notably, Khuri-Makdisi [35] presented a geometric algorithm that performs group operations in the Jacobian of general algebraic curves using $O\left(g^4\right)$ finite field operations.

By combining group arithmetic with the decomposition properties of non-prime divisors, we can now formulate a simplified algorithm for solving the DLP on algebraic curves. This procedure essentially specializes Algorithm 1.1.12 to the specific structure of the Jacobian.

Algorithm 1.2.50.

Input: Given a $C_{n,d}$ curve C of genus $g = \frac{(d-1)(n-1)}{2}$ over $\mathbb{F}_q$ as well as two divisors A and G, such that $A \in \langle G \rangle$.

Output: The discrete logarithm $\log_G A$ on C.

1: Select a factor base $\mathcal{F}$ by selecting small degree divisors until the desired size M is reached.
2: Set $f = 0$.
3: Compute a semi-reduced divisor $D := zA + z'G$ with z, z' being random numbers from $\mathbb{Z}$.
4: Reduce D to a divisor D' of degree less than or equal g.
5: Try to factor D' over $\mathcal{F}$. If this is successful, increase f by one, because a relation in the sense of Algorithm 1.1.12 is found. Store this relation.
6: If $f < M$ go back to step 3. Else continue.
7: Solve a linear system to obtain the discrete logarithm as described in 1.1.12.

Remark 1.2.51. *In Chapter 3, we establish a modification of Algorithm 1.2.50 that fits within the relation generation scheme of Algorithm 1.1.13. In order to do so, we only have to replace the generation of D by adding sufficiently many randomly picked divisors from $\mathbb{F}$, such that D is semi-reduced, but not reduced. Then the obtained relation has exactly the properties required for Algorithm 1.1.13.*

Algorithm 1.2.50 improves upon generic DLP solvers by exploiting the arithmetic of the underlying function field alongside the Jacobian group structure. This allows for the efficient factorization of high-degree divisors into smaller components, providing computational advantages unavailable in generic groups.

However, this algorithm does not efficiently apply for elliptic curves, as all reduced divisors in such curves inherently have degree one. Consequently, the factorization approach is trivial in this case, and no further optimization can be derived from the ideal class group structure. This limitation is specific to elliptic curves, as they lack the richer divisor structure found in higher-genus curves.

The complexity of Algorithm 1.2.50 depends primarily on balancing the size of the factor base, which determines the cost of the linear algebra phase, against the probability of finding a smooth divisor in step five. Given the critical nature of this trade-off, Chapter 2 is dedicated entirely to analyzing smoothness probabilities on curves of varying ratio between their degree and the size of the underlying base field.

For the trial divisions in Algorithm 1.2.50, it is sufficient to trial factor the polynomial a over all primes that give rise to the elements of the factor base for a reduced ideal $I = \langle a, y - b \rangle$. Depending on the degree of a, this can be a costly procedure. In Chapter 3, we will construct explicit schemes that allow us to bypass this step and create smooth elements more directly. To do this, we look at the factorization behavior of principal ideals.

Theorem 1.2.52. *Let $\mathbb{F}_q[x, y]$ be the ring of integers of a function field $F = \mathbb{F}_q(x, y)$ and let $\alpha \in \mathbb{F}_q[x, y]$ be a function on the curve. Denote the ideal norm of F over $\mathbb{F}_q[x]$ by $\mathcal{N}$.*

Let $\mathcal{F}$ be a factor base built of low degree prime ideals of distinct ideal classes in $Cl(\mathbb{F})$. Further, assert that for all prime ideals in $\mathcal{F}$, those other prime ideals of $\mathbb{F}_q[x, y]$ with identical norm are also a part of $\mathcal{F}$.

Then the principal ideal (α) factors over $\mathcal{F}$ if and only if $\mathcal{N}_{\mathbb{F}_q(x) \subset F}(\alpha)$ factors over $\mathcal{F}' := \{\mathcal{N}(x) \mid x \in \mathcal{F}\}$ in $\mathbb{F}_q[x]$. Here $\mathcal{N}_{\mathbb{F}_q(x) \subset F}$ denotes the field norm of F over $\mathbb{F}_q(x)$.

Proof. Let $p_1, \ldots, p_k \in \mathcal{F}$ such that $\prod_{i=1}^{k} p_i = (\alpha)$, i.e., the principal ideal generated by α decomposes over $\mathcal{F}$. Then

$$\mathcal{N}_{\mathbb{F}_q(x) \subset F}(\alpha) = \mathcal{N}((\alpha)) = \mathcal{N}\left(\prod_{i=1}^{k} p_i\right),$$

and therefore $\mathcal{N}_{\mathbb{F}_q(x) \subset F}$ decomposes over $\mathcal{F}'$. Note that the first equality is the known result

about the equivalence of field and ideal norms for principal ideals.

Conversely, let $\mathcal{N}_{\mathbb{F}_q(x)\subset F}$ decompose over $\mathcal{F}'$. We know that for each norm in $\mathcal{F}'$ all prime divisors with this norm are in $\mathcal{F}$. Also every prime ideal of $\mathbb{F}_q[x]$ extends to one or many prime ideals in $\mathbb{F}_q[x,y]$. This suffices to prove that (α) decomposes over $\mathcal{F}$. □

The equivalence of decomposition of ideals in $\mathbb{F}_q[x,y]$ and of polynomials in $\mathbb{F}_q[x]$ has arithmetic advantages. The method for generating relations between factor basis elements was examined for hyperelliptic curves by Velichka, Jacobson and Stein [62]. The following is a direct generalization of this work.

Theorem 1.2.53. *Given the settings of Theorem 1.2.52 and let D be a divisor represented by the ideal $I = \langle a, y-b\rangle$ with $a, b \in \mathbb{F}_q[x]$ as described in 1.2.48.*

Let n be the dimension of I as an $\mathbb{F}_q$-module and $x_0, \ldots, x_{n-1}$ be transcendental elements over $\mathbb{F}_q$. Define

$$\alpha := x_0\, a + \sum_{i=1}^{n-1} x_i\ (y-b)^i\,.$$

Then $f_{(a,b)} := \frac{\mathcal{N}(\alpha)}{\mathcal{N}(I)}$ is a polynomial in $x_0 \ldots x_{n-1}$ over $\mathbb{F}_q[x]$. If there are any $s_i \in \mathbb{F}_q[x]$, such that $f_{(a,b)}\,(s_1, \ldots, s_n)$ decomposes over $\mathcal{F}'$, then an ideal representing $-D$ decomposes over $\mathcal{F}$.

Proof. We start with the assertion that $f_{(a,b)}$ is a polynomial with coefficients in $\mathbb{F}_q[x]$. As described in [45] – which we will cite later in Theorem 3.2.2 – one can compute the norm by evaluating a resultant of two polynomials. This is equivalent to computing the determinant of a certain matrix that is built from the y-coefficients of the input bivariate polynomials. For computing the norm of α, the two involved polynomials in this calculation are given by α as polynomial in the variable y over $\mathbb{F}_q(x)$ and the function field defining polynomial over $\mathbb{F}_q(x)$.

Both polynomials have coefficients that are integral over $\mathbb{F}_q[x]$ multiplied with the transcendental elements. Thus the entries of the matrix are polynomials in $x_0, \ldots x_{n-1}$ with coefficients in $\mathbb{F}_q[x]$ as is the norm, i.e., the determinant of the matrix.

Since the principal ideal generated by α is a subset of I we also know that the norm of I divides the norm of α and therefore, $f_{(a,b)}$ is a polynomial.

Now let $s_i \in \mathbb{F}_q[x]$, such that $f_{(a,b)}(s_1, \ldots s_n)$ decomposes over $\mathcal{F}'$. Then by Theorem 1.2.52 we know that the principal ideal α decomposes over $\mathcal{F} \cup \{I\}$. Therefore, $(\alpha)I^{-1}$ is a $\mathcal{F}$-smooth ideal representing the divisor class of $-D$, because principal ideals are mapped to the neutral element of the Jacobian. □

The theorem allows one to limit the number of trial factorizations drastically by applying a *sieving* technique. We will introduce this technique in the following algorithm.

Algorithm 1.2.54.

Input: A divisor D represented by the ideal $I := \langle a, b - y \rangle$ of rank n, two parameters M and T, an injective function $\nu : \mathbb{F}_q[x] \to \mathbb{N}_0$ and constants $s_2, \ldots s_n$ not all equal to zero.

Output: A list of candidates for $s_1 \in \mathbb{F}_q[x]$ with $\deg(s_1) \leq M$ such that the principal ideal $\langle v \rangle$ is $\mathcal{F}$-smooth for $v = s_1 a + \sum_{i=2}^{n} s_i (y - b)^i \in I$ and thus the decomposition of $\langle v \rangle$ gives a relation over $\mathcal{F}$.

1: Initialize an array $S[\nu(s_1)] = 0$, $s_1 \in \mathbb{F}_q[x],\ \deg(s_1) \leq M$.
2: **for** $p \in \mathcal{F}$ **do**
3: **for** $b \in \mathbb{F}_q[x]^n$ such that $f_I(b', s_2, \ldots, s_n) \equiv 0 \pmod{p}$ **do**
4: Compute all $c := b' + p \cdot v$ for vectors $v = (v_i) \in \mathbb{F}_q[x]^n$ that satisfy $max\left(\deg_x(b_i + pv_i)\right) \leq M$ and increase $S[\nu(c)] \leftarrow S[\nu(c)] + \deg(p)$.
5: **end for**
6: **end for**
7: For any $s_1 \in \mathbb{F}_q[x]$ with $\deg_x s_1 \leq M$ such that $S[\nu(s_1)] \geq T$ trial factor $s_1 a + \sum_{i=1}^{n-1} s_{i+1}(y - b)^i$ over $\mathcal{F}$.
Any successful factorization gives a representation of $-D$ over $\mathcal{F}$.

The algorithm introduces new variables we will often use throughout this thesis; therefore, we establish their notation.

Definition 1.2.55. In the above algorithm we name the integer M the sieve bound and the integer T the tolerance value. Furthermore S is commonly called the sieve

array. The computation of b is the *initialization phase* of the sieve algorithm, while the incrementation of the array elements is called the *sieving phase* or *stepping through the sieve array*.

Remark 1.2.56. *It is worth mentioning that the tolerance value serves two distinct purposes. The first occurs if a large-prime method like in [60] or [25] is to be used in order to allow elements of the search space to pass the criteria, which have some few factors outside of the factor base. The second is to cover cases where a prime divisor might divide a search space element with double or higher multiplicity, because the sieving algorithm will inherently only add* $\deg p$ *once for each root.*

As for the injective function ν in Algorithm 1.2.54, a sufficient choice is to use the evaluation homomorphism by evaluating the vectors c component wise at q. Due to the definition of M, each entry will be naturally smaller than q^M. Then one can weight the different components of the resulting $\mathbb{F}_q^n$ vector by powers of q^M to map each potential outcome to a unique integer value.

In the above algorithm we avoid the situation $s_2 = s_3 = \ldots = s_n = 0$, because $\mathcal{N}(s_1 a) = \mathcal{N}(s_1)\mathcal{N}(a)$. Therefore, if a direct decomposition of D is already known the resulting relation will only contain the conjugates of the already known factors and will only give a trivial relation. On the other hand, to simplify the algorithm, it is common to restrict to the subset of the ideal, which is indexed by $s_2 = 1, s_3 = \ldots = s_n = 0$.

In order to use this early precursor of the algorithm efficiently, some steps have to be concretized and refined. We will address this in detail in chapter 3 after investigating the topic of ideal factor base sizes and smoothness probabilities in chapter 2.

1.2.6. Elliptic Curve Discrete Logarithm Problems and Weil Descent

By [17] we know that the discrete logarithm problem in algebraic curves of higher degree or higher genus is efficiently solvable with non-generic methods. In contrast, the elliptic curve discrete logarithm problem — i.e., the discrete logarithm problem on an algebraic curve of genus one — is not generally vulnerable to non-generic attacks. Therefore, elliptic curves

are often recommended [30] for high security purposes of the protocol given in Algorithm 1.1.3, whereas hyperelliptic curves of higher genus or non-hyperelliptic algebraic curves are rarely used.

A central motivation for solving discrete logarithm problems in higher genus Jacobians is the reducibility of an elliptic curve discrete logarithm problem (*ECDLP*) to an algebraic curve discrete logarithm problem under certain conditions. This vulnerability of the ECDLP was introduced by Gaudry, Hess and Smart in [23]. Later it was investigated by Stein in [32] and it is known as the *Weil descent approach*. We will give a short introduction into this important motivation. For more elaborate details, there is a good synopsis of the topic by Hess [28].

Definition 1.2.57 (Weil restriction)**.** Let $K \subset L$ be an extension of finite fields and X an abelian variety over L. Also let $\times_K$ denote the tensor product as K-vector spaces. Then

$$Res_{K\subset L}X(S) := X(S \times_K L)$$

is the *Weil-restriction* of X with respect to K.

Remark 1.2.58. *Note that $Res_{K\subset L}X$ is defined over K instead of L and that for example the set of $K-$ rational points of $Res_{K\subset L}X$ is isomorphic to the set of K-rational points of X. Compare [64, section 1.3]*

For an elliptic curve E the set of $L-$ rational projective points is isomorphic to the Jacobian. Therefore, the Weil restriction can be used to build varieties over fields of the same characteristic – but lower extension degree over the prime field – that offers an isomorphism from the set of $K-$rational points into the Jacobian of the elliptic curve. On the other hand degree and genus of $Res_{K\subset L}(E)$ will probably rise. This technique also translates the discrete logarithm problem on the elliptic curve into a discrete logarithm problem on a curve of higher degree and genus over a subfield of L.

Abstractly the Weil-descent attack on the elliptic curve discrete logarithm problem is just a generalization of this concept.

Algorithm 1.2.59 (Brief Working Principle of Weil Descent)**.** Let E be an elliptic function field defined over $\mathbb{F}_q$, $q = p^n$, $p \in \mathbb{P}$, and $C \mid E$ an extension of function fields such that the exact field of constants of C equals $\mathbb{F}_q$.

Assume there is a field automorphism σ of C of order l extending the Frobenius automorphism of $\mathbb{F}_q \mid \mathbb{F}_{q'}$ with $q' = p^{n'}, n' \mid n$. Then for the function field C^0 of elements fixed by σ, we obtain $[C : C^0] = l$ and the exact field of constants is $\mathbb{F}_{q'}$.

Using the norm map of $C^0 \subset C$, we can map the discrete logarithm problem from $Jac(E)$ into $Jac(C^0)$ if the elements involved are not in the kernel of σ.

The construction of C, C^0 and σ is not straightforward. One concrete instance for $p = 2$ was found by Gaudry, Hess and Smart, and is thus known as the GHS attack on the discrete logarithm problem on an elliptic curve. For the following theorem, we need the definition of the *trace* of elements in a tower of fields in characteristic 2. This definition can be found in Appendix A.3.11.

Theorem 1.2.60 (Theorem 1 of [28])**.** *Let E be an elliptic function field defined by the equation*

$$Y^2 + XY = X^3 + aX^2 + b.$$

over a finite field $\mathbb{F}_q$, where $q = 2^k$ and $b \neq 0$. In every isomorphism class of elliptic curves, there is at least one curve of this form. Further, let $\mathbb{F}'_q$ be the subfield of $\mathbb{F}_q$ containing a. Consider the Artin-Schreier equation (see Appendix A.3.10)

$$y^2 + y = \frac{\gamma}{x} + a + \beta x$$

that is birationally equivalent to the initial equation by the transformation

$$\tilde{x} = \frac{x}{\gamma} \text{ for some } \gamma \in \mathbb{F}_q^*$$

$$Y = \frac{y}{\tilde{x}} + b^{\frac{1}{2}}, \qquad X = \frac{1}{\tilde{x}}, \qquad \beta = \frac{b^{\frac{1}{2}}}{\gamma}.$$

Now let C be the splitting field of γ, a, β over $\mathbb{F}_{q'}(x)$. Then the Frobenius automorphism of $\mathbb{F}_q(x)$ with respect to $\mathbb{F}_{q'} \subset \mathbb{F}_q$ extends to C if and only if one of the following conditions holds:

1. $Tr_{\mathbb{F}_2 \subset \mathbb{F}_q}(a) = 0$
2. $Tr_{\mathbb{F}_{q'} \subset \mathbb{F}_q}(\beta) \neq 0$
3. $Tr_{\mathbb{F}_{q'} \subset \mathbb{F}_q}(\gamma) \neq 0$.

Then the Weil descent method described in Algorithm 1.2.59 is applicable to the fixing field C^0 of the Frobenius on C.

For the resulting curve, the genus and degree of the resulting function field C^0 are of interest. Theorem 1.2.61 proves that it depends on the choice of γ.

Theorem 1.2.61. *In the situation of Theorem 1.2.60, let d_γ and d_β be the degrees of the minimal polynomials of γ and β over k. Further, let $d_{\gamma,\beta}$ be the degree of the least common multiple of the minimal polynomials of γ and β. Then the genus g of C^0 satisfies*

$$g = 2^{d_{\gamma,b'}} - 2^{d_{\gamma,\beta}-d_\gamma} - 2^{d_{\gamma,\beta}-d_\beta} + 1.$$

Furthermore, there is a rational function subfield $\mathbb{F}_{q'}(z)$ of C satisfying

$$[C : \mathbb{F}_{q'}(z)] = \min\left\{2^{d_\gamma}, 2^{d_\beta}\right\}.$$

Corollary 1.2.62. *In the situation of Theorem 1.2.61, we conclude that C^0 is a hyperelliptic function field if $\gamma \in \mathbb{F}_{q'}$ or $\beta \in \mathbb{F}_{q'}$.*

A high-degree curve, as well as the size of the resulting Jacobian will hinder the attempt to solve the discrete logarithm problem in $Jac(E)$ by attacking the structure of $Jac(C^0)$. The same applies if the map from $Jac(E)$ to $Jac(C^0)$ has a kernel that is too large. Therefore, the choice of γ has a high impact on the practicality of the approach.

Hess [28] gives an example with an elliptic curve defined over $\mathbb{F}_{2^{155}}$ whose Jacobian is mapped into Jacobians of hyperelliptic curves of genus 31 and 32767, both defined over

$\mathbb{F}_{2^5}$, with the only difference caused by the choice of γ. Later, we will investigate solving discrete logarithms in Jacobians of hyperelliptic curves obtained via Weil descent.

A primary limitation of Weil descent is the sparsity of vulnerable elliptic curves. For instance, among the approximately 2^{156} isomorphism classes of elliptic curves over $\mathbb{F}_{2^{155}}$, only about 2^{32} are directly vulnerable to the GHS attack, i.e., the constructed curves over the subfield $\mathbb{F}_{32}$ have genus 31 or 32 [32, comments to Theorem 1].

Theorem 1.2.63 (Gaudry, Hess and Smart [23])**.** *Let E be an elliptic function field over $\mathbb{F}_q$ and E' a second function field such that $\#Jac(E) = \#Jac(E')$. Then the curves E and E' are isogenous — i.e., there is a surjective morphism from the variety associated with E to that of E'. Consequently, if E' is vulnerable to the Weil descent method, one can also attack the DLP on E by exploiting the isogeny and its dual. Note that it is not necessary for this approach that E and E' share the same isomorphism class.*

Following the arguments of Galbraith, Hess and Smart in [23], this increases the number of vulnerable isomorphism classes of elliptic curves over $\mathbb{F}_{2^{155}}$ to about 2^{104}. Furthermore, they prove that almost every curve over a field $\mathbb{F}_{q^7}$ with q is a power of two is vulnerable to Weil descent attacks using isogenies.

2

Smoothness Estimates for Divisors in Function Fields

The runtime of index calculus algorithms is primarily determined by the size of the factor base. Generally, a larger factor base allows for faster discovery of relations over the factor base. However, since the runtime for solving the linear system increases quadratically with the size of the factor base, the overall complexity rises if the base becomes too large.

Thus, obtaining an accurate estimate of the smoothness of elements over a factor base with a given structure is essential for both complexity analysis and practical parameterization of a DLP solver. This chapter aims to provide smoothness probability estimates based on the factor base size, curve model, and method used to select ideals for decomposition, independent of any specific curve. In line with Lemma 1.1.16, we focus primarily on the range of factor base sizes where the size is balanced against the estimated number of trials.

This chapter is organized as follows: We first review relevant literature, highlighting the work that inspired our developments. We then derive an essential formula for estimating the number of relevant primes of a given degree for solving the DLP. This estimate supports our own calculations for the smoothness probability for ideals of function fields.

We distinguish between function fields of small and large degree, as the factors influencing smoothness differ in each case. While considering these cases separately, we emphasize that both estimates align at the transition between the two regimes.

A note on nomenclature: As established in the previous chapter, we can identify semi-reduced effective divisors with ideals in $\mathbb{F}_q[x, y]$, where each entry in the divisor reflects the multiplicities of prime ideals in the factorization of an ideal. Key properties, such as the degree of a divisor, directly correspond to the degree of the associated ideal. This allows us to use the divisor and ideal perspectives interchangeably. Both representations appear in the literature.

The results in this chapter are valid for semi-reduced effective divisors in general function fields of algebraic curves. For the class of $C_{n,d}$ curves, which we have chosen as the curves of interest in this thesis, the only advantage is that the fully ramified prime ideal at infinity results in a one-to-one correspondence between the semi-reduced effective divisors and the elements of the Jacobian group of the curve. This allows the direct translation of the properties of ideals and their decomposition into the Jacobian.

2.1. Existing Estimates and Approaches

Most results in the literature concern hyperelliptic curves. For these curves and the classic index calculus approach as given in Algorithm 1.2.50, it is most relevant how many of the approximately q^g divisors of degree g are smooth over a previously selected factor base. We will first cite a result of Gaudry and Theriault for hyperelliptic curves of constant and low genus.

Theorem 2.1.1 (Gaudry [24], Theriault [60]). *Let C be a hyperelliptic curve of genus g with constant field $\mathbb{F}_q$, and let $\mathcal{F}$ be a factor base consisting of q^r divisors of degree one. Furthermore, assume that* $\log_q(g!) < 1$.
Then the factor base size and the smoothness probability are equal for $r = \frac{g}{g+1} + \frac{\log_q(g!)}{g+1} < 1$.
In particular, the total run time of Algorithm 1.2.50 is in $\widetilde{O}\left(q^{2r}\right) = \widetilde{O}\left(q^2\right)$.

To understand the complexity analysis of Theorem 2.1.1, it is worth recalling that $\widetilde{O}$

denotes an asymptotic complexity that omits logarithmic factors.

We omit a dedicated proof here, as we will later formulate an analogous theorem for non-hyperelliptic curves that follows the same approach. We note that the term $g!$ is negligible for the overall complexity only when g is small relative to q. For larger values of g, restricting to primes of degree one for creating the factor base is not sufficient, as the factorial will become dominant. We will cite a result of Enge [16], taking increasing values of g into account and proving a subexponential complexity for higher genera while also covering cases where g and $\ln q$ are close.

Theorem 2.1.2 (Theorem 1 of [16]). *Let $g \geq \theta \log q$ for some constant $\theta > 0$.*

Then there is an algorithm that solves any instance of the discrete logarithm problem over a hyperelliptic curve of genus g over $\mathbb{F}_q$ in subexponential time

$$L_{q^g}\left[\frac{1}{2}, \frac{5}{\sqrt{6}}\left(\sqrt{1+\frac{3}{2\theta}}+\sqrt{\frac{3}{2\theta}}+o(1)\right)\right].$$

Remark 2.1.3. *In the original paper, the notation for the complexity class is given as $L_{q^g}(\beta)$, which is a common short version for the complexity class $L_{q^g}\left[\frac{1}{2}, \beta\right]$ that we define and discuss in Appendix A.1. For better consistency, we have translated all occurrences of the short version into its long form throughout this chapter.*

The algorithm referenced in Theorem 2.1.2 is a variant of Algorithm 1.2.50, utilizing randomly generated divisors and trial factorization for generating relations. As determined in [16], the algorithm's complexity is dominated by the product of the smoothness probability and the factor base size.

For curves of higher genus, stricter estimates for the smoothness probability are available. Notably, Enge and Stein [19] provide sharper tuning for this ratio. Their work is one of several approaches to estimating smoothness, chosen here because its proof relies solely on combinatorial techniques and avoids heuristic assumptions.

Theorem 2.1.4. *[19, Theorems 5 and 6] Consider the ring of integers of a function field corresponding to a genus g hyperelliptic curve over $\mathbb{F}_q$. Let $N(\eta, m)$ denote the number of all semi-reduced ideals of degree η. Further, let $m = \lceil \log_q L_{q^\eta}(\rho) \rceil$ for a constant $\rho > 0$ satisfy*

$$\eta \geq m \geq 4 \max\left(4 \log_q\left(2g + 6 + \sqrt{2}\right), 1 + \log_q\left(\left(6 + \frac{10}{3}\sqrt{2}\right) n\right)\right).$$

Then there is a function $\beta(g)$ in $o(1)$ for $g \to \infty$ such that

$$N(\eta, m) \geq L_{q^\eta}\left[\frac{1}{2}, -\frac{1}{2\rho} - \beta(g)\right] q^g.$$

We note that we changed the notation of [19], naming the degree of the elements to factor η instead of n in order to avoid confusion with the degree in y of a $C_{n,d}$ curve.

Corollary 2.1.5. *Balancing the parameter ρ in Theorem 2.1.4 and using $\eta = g$ as in Algorithm 1.2.50, we gain an asymptotic complexity of the hyperelliptic curve discrete logarithm problem of*

$$L_{q^g}\left[\frac{1}{2}, \sqrt{2} + o(1)\right]$$

with the overhead of the relation search as well as the β function of 2.1.4 covered by the $o(1)$ term.

The result of Enge and Stein is sharper than Theorem 2.1.2 for hyperelliptic curves of higher genus, because the achieved complexity class is lower. The result also has the advantage that the degree of the ideals to be factored is kept flexible. Therefore, Theorem 2.1.4 can also be used to examine alternative algorithms that produce elements of degree unequal to g to be factored.

However, because 2.1.4 does not apply to small-genus curves, a gap remains in the complexity estimates for curves in the transition range. The scope of this chapter is to create sharp estimates for the smoothness probability for general $C_{n,d}$ curves that can be applied for both high and low genus and degree and align at the critical transition. To do so, we

will first introduce some important results on the number of prime divisors we can expect our function fields to have for a fixed degree.

2.2. Counting Prime Divisors of a Fixed Degree

For the following smoothness analysis, the count of low-degree prime divisors is critical, as these form the building blocks of the factor base. To ensure the factor base can be constructed as proposed, we first establish bounds for the number of prime divisors, which depend on the genus of the curve and the target degree.

The approach of the following estimate is similar to that used for Theorem 2 of [19] with the new scope of extending the theorem to non-hyperelliptic curves. We restrict our attention to a subset of all prime divisors which we will require for the algorithm we introduce in the next chapter.

Definition 2.2.1. Let P be a place in a function field of an algebraic curve defined in the variables x and y over $\mathbb{F}_q$. Then we will use P as a component of the factor base if

1. $x, y \in \mathcal{O}_P$, i.e., the image of x, y in the residue class field defined by P exists, and
2. $\deg P = \left[\mathbb{F}_q(\bar{x}) : \mathbb{F}_q\right]$ for $\bar{x}$ denoting the residue of x in the residue class field defined by P.

We refer to prime divisors satisfying these conditions as *convenient*.

Interpreted via ideals, Condition 2 implies the underlying prime polynomial $p \in \mathbb{F}_q[x]$ cannot be inert, as this would result in a higher extension degree. Furthermore, because $F_q[x]$ is a principal ideal domain, any inert prime in the extension $F_q[x, y]$ would itself be principal. Consequently, we can filter these cases out.

The following theorem will provide an estimate of the number of convenient places of each degree.

Theorem 2.2.2. *Let C be an algebraic curve over the finite field $\mathbb{F}_q$ with function field $\mathbb{F}_q(x, y)$. Further define $n := \left[\mathbb{F}_q(x, y) : \mathbb{F}_q(x)\right]$ to be the extension degree of the function*

field over the rational function field in x and let g denote the genus of C.

Finally, let $\Pi(k)$ *denote the number of convenient prime divisors of degree k. Then*

$$\frac{1}{k}\left(q^k - q^{\frac{k}{2}}(1+2g+2n)\right) - n \leq \Pi(k) \leq \frac{1}{k}\left(q^k + q^{\frac{k}{2}}(1+2g)\right).$$

Proof. First, we have to estimate the total number of places of a fixed degree k. Let B_k denote the set of all prime divisors of degree k. Then, by Corollary 5.2.10 of [55] we have

$$\left|B_k - \frac{q^k}{k}\right| \leq \left(\frac{q-1}{q} + 2g\frac{q^{\frac{1}{2}}}{q^{\frac{1}{2}}-1}\right)\frac{q^{\frac{k}{2}}-1}{k}.$$

We subtract the primes not matching the condition to be convenient. Let the function field be defined by $\mathbb{F}_q(x,y)$. Then any finite place corresponds to a prime ideal of the ring of integers $\mathbb{F}_q[x,y]$ that extends a prime ideal of $\mathbb{F}_q[x]$. The same is valid for the non-finite places extending the single infinite place of $\mathbb{F}_q(x)$.

The number of non-finite places of all degrees can be at most n, because the added degrees must not exceed the degree of the field extension. Therefore, the only remaining prime divisors to count are those that do not fulfill the second condition of Definition 2.2.1.

Any affine place corresponds to a prime ideal I of $\mathbb{F}_q[x,y]$. Therefore, we conclude that there is a prime polynomial $p \in \mathbb{F}_q[x]$ such that the residue class field $\mathbb{F}_q(\bar{x},\bar{y}) \simeq \mathbb{F}_q[x,y]/I$ extends $\mathbb{F}_q[x]/(p)$ and the ideal I extends (p). Let $k' = \deg p$. It is immediate that $k' \mid k$ and that the extension $I \mid (p)$ corresponds to an irreducible factor of the function field's defining polynomial $\pmod p$ of degree $\frac{k}{k'}$. Therefore, for each such p of degree k' there are at most $\frac{nk'}{k}$ places of degree k such that the corresponding ideal extends (p).

Estimating the exact number of inert prime polynomials is more complex, but it is plainly bounded by the number $N(k')$ of all prime polynomials of degree k'. By some basic combinatorial considerations, $N(k') \leq \frac{1}{k'}q^{k'}$ holds.

All occurring degrees are integers, and thus we can conclude that there are at most

$$\sum_{k\neq k'\mid k} \frac{nk'}{k}\frac{1}{k'}q^{k'} = \frac{n}{k}\sum_{k\neq k'\mid k} q^{k'} \leq \frac{n}{k}\sum_{i=1}^{\frac{k}{2}} q^i < \frac{n}{k}\frac{q^{\frac{k}{2}+1}}{q-1} < 2\frac{n}{k}q^{\frac{k}{2}}$$

non-convenient but finite primes.

Combining these with the estimate for B_k and the maximal possible number of primes extending the non-finite place of $\mathbb{F}_q(x)$, we can conclude

$$\frac{1}{k}\left(q^k - q^{\frac{k}{2}}(1+2g+2n)\right) - n \leq \Pi(k) \leq \frac{1}{k}\left(q^k + q^{\frac{k}{2}}(1+2g)\right).$$

□

Corollary 2.2.3. *Let $k \geq 2\log_q(2+4g+4n)$ in the situation of Theorem 2.2.2. Then*

$$\frac{1}{2k}q^k - n \leq \Pi(k) \leq \frac{2}{k}q^k.$$

Proof. By the condition $k \geq 2\log_q(2+4g+4n)$, we can conclude that

$$0.5q^k \leq q^{\frac{k}{2}}(1+2g+2n).$$

Inserting this into the result of 2.2.2 directly gives the desired estimate. □

Remark 2.2.4. *For later use, the "$-n$"-term in the lower bound will be of no effect. For $C_{n,d}$-curves, the divisor extending the infinite place of the rational function field is the unique representative of the neutral element of the Jacobian. For other curves, we can include the other places extending infinity to the factor base and allow a special treatment of them. For hyperelliptic curves that do not have an imaginary quadratic representation, this special treatment is known as infrastructure computation. Jacobson, Scheidler, and Stein [33] gave an elaborate introduction and analysis of this topic.*

Given these estimates, we are able to prove that enough convenient divisors of low degree exist to create a factor base in the first place. We will do so for very low degree curves relative to q to ensure we have sufficient prime divisors to create the factor base needed for the assumptions in the next section.

Lemma 2.2.5. *Consider a $C_{n,d}$ curve as defined in Definition 1.2.28 of genus $0 < g = \frac{(n-1)(d-1)}{2} < \log q$, and assume $q \geq 5640$. Then there are at least $\frac{1}{2}q$ convenient prime*

divisors of degree one.

Proof. We know that $n > 1$ in order to get a curve of genus greater than 0. Moreover, $d > n$, so we can calculate $(n-1)^2 < (n-1)(d-1) < 2g$ and consequently $n < \sqrt{2g}+1$. Because $n = 2$ for $g \in \{1,2\}$ and $n < g+1$ otherwise, we even get $2+4g+4n \leq 6+8g \leq 6+8\log q$. Now, with $\log_q(6+8\log q) < 0.5$ for any integer $q > 5640$, we have verified the assumptions of Corollary 2.2.3 and Remark 2.2.4. This completes the proof. □

This lemma establishes a lower bound on q for low-degree $C_{n,d}$ curves, guaranteeing that sufficiently many degree-one prime divisors exist to build a factor base.

2.3. A Refined Estimate for Low Degree Divisor Decomposition

For large q or low curve degree, the divisors targeted during the relation search phase have small degrees. Consequently, the optimal factor base size allows it to consist exclusively of split prime divisors of degree one. We provide a smoothness estimate for this case in the theorem below. These conditions are specifically tailored to fulfill Lemma 2.2.5, guaranteeing that sufficient primes are available for the factor base construction.

Theorem 2.3.1. *Given an algebraic curve with exact field of constants $\mathbb{F}_q$, let $r < 1$. Let $\mathcal{F}$ be a factor base containing exactly q^r degree-one divisors, and let η be the expected degree of divisors to be factored. Then the probability that a randomly chosen non-inert divisor is $\mathcal{F}$-smooth is bounded by $O\left(\frac{1}{\eta!}q^{\eta(r-1)}\right)$.*

Proof. By following the Hasse-Weil theorem, the total number of divisors of degree η is in the interval $q^\eta \pm (1+2g+4d)q^{\frac{\eta}{2}}$. Since we consider only those divisors with inertia degree 1, we can use q^η as an upper bound of the total number of divisors in the search range. Of these divisors, at least $\frac{1}{\eta!}(q^r)^\eta = \frac{1}{\eta!}q^{\eta r}$ can be composed of η distinct primes of the factor base. Therefore, the probability of a randomly chosen non-inert divisor to be smooth over the factor base is bounded below by

$$\frac{1}{\eta!}\frac{q^{\eta r}}{q^\eta} = \frac{1}{\eta!}q^{\eta(r-1)}.$$

□

To represent the complexity with respect to the size of the problem, we apply a technique similar to that of Enge [16] to the smoothness probability. By doing so, we introduce a new variable θ measuring the size of η with respect to q.

Definition 2.3.2. Let C be an algebraic curve with exact field of constants $\mathbb{F}_q$, and let η be the degree of divisors generated for factoring them over the factor base.

Then define $0 < \theta \in \mathbb{R}$ such that $\eta = \theta \ln q$. Further define $\gamma := \ln \ln (q^\eta)$.

Unlike Enge's original approach [16], which assumed divisors were generated via Cantor's reduction algorithm, thereby fixing $\eta = g$, we allow η to vary based on the generation method.

Until now, we have the smoothness estimate of Theorem 2.3.1 as well as the factor base size given as powers of q. For a better comparison with results for curves of greater degree, we now translate these terms into the subexponential notation. To do so, we use the definition of θ and γ as in Definition 2.3.2.

Theorem 2.3.3. *Let q, η, θ, and γ be as in Definition 2.3.2 with $\theta \leq 1$. Then*

$$q = L_{q^\eta}\left[\frac{1}{2}, \frac{1}{\sqrt{\gamma\theta}}\right], \text{ and}$$
$$\eta! \leq \eta^\eta = L_{q^\eta}\left[\frac{1}{2}, \frac{\sqrt{\gamma\theta}}{2}\right].$$

Proof. The first assertion is given by a direct computation:

$$\begin{aligned} L_{q^\eta}\left[\frac{1}{2}, \frac{1}{\sqrt{\gamma\theta}}\right] &= e^{\frac{1}{\sqrt{\gamma\theta}}\sqrt{\eta \ln q}\sqrt{\ln \ln q^\eta}} \\ &= e^{\frac{1}{\sqrt{\theta}\sqrt{\frac{\eta}{\ln q}}}\sqrt{\eta \ln q}\sqrt{\theta}} \\ &= e^{\ln q} = q. \end{aligned}$$

Similarly, we can prove the second assertion directly. First of all, it is immediate that

$$\sqrt{\theta}\sqrt{\ln q^\eta} := \sqrt{\frac{\eta}{\ln q}}\sqrt{\eta \ln q} = \eta.$$

Therefore,

$$\ln \eta = \ln \sqrt{\theta}\sqrt{\ln q^\eta} = \frac{1}{2}\left(\ln \theta + \ln \ln q^\eta\right) \leq \frac{1}{2} \ln \ln q^\eta,$$

since $\theta \leq 1$ and thus $\ln \theta \leq 0$. From the definition of γ it follows that $\ln \eta \leq \frac{\sqrt{\gamma}}{2}\sqrt{\ln \ln q^\eta}$. Finally, we get

$$\eta \ln \eta \leq \frac{\sqrt{\gamma\theta}}{2}\sqrt{\ln q^\eta}\sqrt{\ln \ln q^\eta}$$

and therefore

$$\eta! \leq \eta^\eta = e^{\eta \ln \eta} \leq L_{q^\eta}\left[\frac{1}{2}, \frac{\sqrt{\gamma\theta}}{2}\right].$$

□

With these estimates at hand, we can now state a replacement theorem for Theorem 2.1.2. The replacement is only valid for $\theta < 1$, but offers a sharper estimate and more flexibility with respect to the degree of elements that we attempt to factor.

Theorem 2.3.4. *Let C be an algebraic curve with exact field of constants $\mathbb{F}_q$, and assume there is a way to generate divisors with known non-trivial decomposition of degree η. Furthermore, let $\mathcal{C}$ be the maximum of the individual cost functions for the divisor generation, trial factoring and the logarithmic terms of the linear algebra phase. Finally, let $\theta < \frac{2}{\gamma} \leq 1$ and thus $\ln \ln q^\eta \geq 2$ or $q^g \geq 1619$ respectively.*

Then there is an algorithm for solving any instance of the discrete logarithm problem over C in expected time

$$O(\mathcal{C})\, L_{q^\eta}\left[\frac{1}{2}, 2 \cdot \frac{2d + \gamma\theta}{2d+2}\frac{1}{\sqrt{\gamma\theta}}\right] \leq O(\mathcal{C})\, L_{q^\eta}\left[\frac{1}{2}, 2 \cdot \frac{1}{\sqrt{\gamma\theta}}\right]$$
$$\xrightarrow[\theta \to \frac{2}{\gamma}]{} O(\mathcal{C})\, L_{q^\eta}\left[\frac{1}{2}, \sqrt{2}\right]$$

Proof. For solving the discrete logarithm problem, we use Algorithm 1.2.50 and apply Theorem 2.3.1 for estimating the smoothness probability for a factor base with about q^r elements for $r \leq 1$ yet to be optimized.

Consequently, we have to search for about $q^r = L_{q^\eta}\left[\frac{1}{2}, r\frac{1}{\sqrt{\gamma\theta}}\right]$ relations by applying

Theorem 2.3.3. The expected number of divisors to be tested for smoothness is given by

$$O\left(\eta! q^{(1-r)\eta}\right) = L_{q^\eta}\left[\frac{1}{2}, \frac{\sqrt{\gamma\theta}}{2} + (1-r)\eta\frac{1}{\sqrt{\gamma\theta}} + o(1)\right]$$

for each of the searched relations.

We compute the expected time complexity of the relation collection phase to be

$$O(\mathcal{C})\, L_{q^\eta}\left[\frac{1}{2}, r\frac{1}{\sqrt{\gamma\theta}} + \frac{\sqrt{\gamma\theta}}{2} + (1-r)\eta\frac{1}{\sqrt{\gamma\theta}}\right].$$

By using the Wiedemann algorithm A.4.11, the expected time complexity of the linear algebra phase of Algorithm 1.1.15 is the square of the factor base size times some logarithmic terms. Therefore, we can estimate the complexity to be

$$O(\mathcal{C})\, L_{q^\eta}\left[\frac{1}{2}, 2r\frac{1}{\sqrt{\gamma\theta}}\right].$$

Since the parameter r is not yet set, we can balance it to minimize the overall time spent in the two consecutive phases. Therefore, we have to compute a value for r such that

$$2r\frac{1}{\sqrt{\gamma\theta}} = r\frac{1}{\sqrt{\gamma\theta}} + \frac{\sqrt{\gamma\theta}}{2} + (1-r)\eta\frac{1}{\sqrt{\gamma\theta}},$$

which is given for $r = \frac{\eta+\gamma\theta}{\eta+2}$. Note that the condition $\theta < \frac{2}{\gamma}$ ensures $r < 1$. This is important to ensure that there are enough divisors of degree one to fill the factor base.

Consequently, we estimate the entire complexity to be less than or equal to

$$O(\mathcal{C})\, L_{q^\eta}\left[\frac{1}{2}, 2\frac{1}{\sqrt{\gamma\theta}}\right],$$

which completes the proof. □

Remark 2.3.5. *For the sake of simplicity, we dropped the complexity for generating the two special relations that are required by Algorithm 1.1.15. We can fix this issue by the following consideration. Assume we start random walks at the elements α and the group generator γ by*

adding elements from the factor base. By using the algorithm of Harasawa and Suzuki [26], we can ensure that all elements from the two walks are of degree at most g, at a cost of $O\left((\ln q^g)^2\right)$. *Therefore, on average, we have to do approximately* g! *steps until we find representations that we can decompose to divisors of degree one. If we add these elements of degree one to the factor base beforehand we ensure that we found the two special relations.*

The only thing left to prove is that $g! \leq L_{q^\eta}\left[\frac{1}{2}, 2\frac{1}{\sqrt{\gamma\theta}}\right]$. *This obviously depends on the values of* η *generated by the chosen method.*

Alternatively, other algorithms can be used to decompose the two special divisors into primes of low degree. In Section 3.4.2, we will construct an algorithm that succeeds in doing so within the desired time bound if η *is at least about* $2\sqrt{g}$.

A central difference between Theorems 2.1.2 and 2.3.4 is that the latter result depends on γ, which represents the square logarithm of the size of the Jacobian. Nevertheless, the result of Theorem 2.3.4 is sharper than the reference.

Corollary 2.3.6. *Given a hyperelliptic curve C of genus* $g = \theta \log q$ *such that* $\theta < \frac{2}{\gamma}$ *for* $\gamma = \ln\ln q^g \geq 1$. *Then there is an algorithm solving any instance of the hyperelliptic curve discrete logarithm problem on C in expected time*

$$O\left(g^2 (\ln q)^2\right) L_{q^g}\left[\frac{1}{2}, 2\sqrt{\frac{1}{\theta}}\right] < O\left(L_{q^g}\left[\frac{1}{2}, \frac{5}{\sqrt{6}}\left(\sqrt{1+\frac{3}{2\theta}} + \sqrt{\frac{3}{2\theta}} + o(1)\right)\right]\right).$$

Note that in this case, as $\eta = g$, the issue regarding the creation of the two special relations does not occur, as any randomly created linear combination of the two divisors has the exact same degree as those used for the general relations.

2.4. A Refined Estimate for High Degree Divisor Decomposition

The estimates from the previous section fail for larger η, as higher degrees force $r \geq 1$ in the proof of Theorem 2.3.4. This would be required to compensate for the larger $d!$ term in the smoothness estimates done in Theorem 2.3.1. Therefore, prime divisors of higher degree have to be included when building the factor base.

As for the estimates of lower degree, we required our factor base to be built from prime divisors satisfying Definition 2.2.1 only. Because we will now need to include primes of higher degree as well, we will require it to contain all convenient prime divisors of degree less than the maximum occurring degree.

Definition 2.4.1. Let $\mathcal{F}$ be a finite factor base of function field divisors containing only prime divisors that are convenient in the function field extension $\mathbb{F}_q(x) \subset F$. Let $B := \max\{\deg(D) \mid D \in \mathcal{F}\}$.
Then we define $\mathcal{F}$ to be *dense* if

$$\{D \text{ divisor of } F \mid \deg D < B \text{ and } D \text{ is convenient}\} \subseteq \mathcal{F}.$$

For a dense factor base $\mathcal{F}$ we define $N(\eta, \mathcal{F})$ to be the number of $\mathcal{F}$-smooth divisors of degree η. To prove certain smoothness probabilities we extend this definition to $N(\eta, m)$, which denotes the number of divisors of degree η that are smooth for a factor base including all convenient prime divisors up to and including degree m.

The following theorem is inspired by Theorem 3 of [19]. The primary difference is a reduction of the requirements towards m. This will help to find sharper estimates in the forthcoming assertions.

Theorem 2.4.2. *Let $m \in \mathbb{N}$ satisfy $m \geq 4^{\frac{\ln(3+4g+4n)}{\ln q}}$, let further $u := \frac{\eta}{m} > 1$ and $\eta \leq 2g + 2n$. Then*

$$N(\eta, m) \geq \frac{q^\eta}{4\eta^{\lceil u \rceil}}.$$

Proof. The proof constructs divisors of degree η using the fewest possible prime factors, thereby maximizing the degree of each factor towards m. A lower bound for the number of required prime factors is $\lceil u \rceil$.

Let $m_0 := max\{1 \leq x \leq m \mid x \in \mathbb{N},\ x\lceil u \rceil \leq \eta\}$. Note that $m_0 \geq \lfloor \frac{u-1}{u} \rfloor m \geq (\frac{u-1}{u} - \frac{1}{m})m$, because

$$\frac{u-1}{u} m \lceil u \rceil \leq \frac{u-1}{u} m \frac{u+1}{u} u \leq \frac{u^2-1}{u^2} \eta.$$

Furthermore, we have to rule out some specific edge cases. For $m = 1$, we also have $m_0 = 1$ and $u = \eta$ by construction. For $m = 2$, we have $\lfloor \frac{\eta}{2} \rfloor$ factors of degree two and at most one of degree one. For $m > 2$ and $\lceil u \rceil = 2$, we know that a divisor of degree $\eta > m$ decomposes either into two divisors of degree $\frac{\eta}{2} > \frac{m}{2}$ or into one of degree $\lfloor \frac{\eta}{2} \rfloor \geq \frac{m}{2}$ and one of degree $\lceil \frac{\eta}{2} \rceil$. If $m > 2$ and $u > 2$, we know by the formula above that $m_0 \geq (\frac{u-1}{u} - \frac{1}{m})m \geq \frac{m}{2}$.

Therefore, we know $m_0 \geq 2^{\frac{\ln(3+4g+4n)}{\ln q}}$ and we apply Corollary 2.2.3 for estimating the number of prime divisors of degree greater than or equal to m_0.

Now let $m_1 := m_0 + 1$. Then $\lceil u \rceil m_0 \leq \eta < \lceil u \rceil m_1 = \lceil u \rceil m_0 + \lceil u \rceil$ holds by construction. For the smoothness estimates, we will count the divisors of degree η that are built from $\lceil u \rceil$ prime divisors of degree m_0 and m_1 only. We set $r_1 := n - \lceil u \rceil m_0$ and $r_0 := \lceil u \rceil - r_1$. Then it is immediate that $r_0 + r_1 = \lceil u \rceil$ and $r_0 m_0 + r_1 m_1 = \eta$.

Consequently, every divisor factoring to r_0 prime factors of degree m_0 and r_1 prime factors of degree m_1 will be of degree η. Each of these divisors is m-smooth because, in the case where $m_1 > m$, this implies $m_0 \mid \eta$ and thus $r_0 = u = \lceil u \rceil$ and $r_1 = 0$.

We estimate the number of divisors that arise from the given construction by applying Theorem 2.2.2 and Remark 2.2.4:

$$\begin{aligned} N(\eta, m) &\geq \binom{\Pi(m_0)}{r_0} \binom{\Pi(m_1)}{r_1} \geq \frac{(\Pi(m_0) - r_0)^{r_0}}{r_0!} \frac{(\Pi(m_1) - r_1)^{r_1}}{r_1!} \\ &\geq \frac{\left(\frac{q^{m_0} - q^{\frac{m_0}{2}}(1+2g+2n)}{m_0} - r_0 \right)^{r_0}}{r_0!} \frac{\left(\frac{q^{m_1} - q^{\frac{m_1}{2}}(1+2g+2n)}{m_1} - r_1 \right)^{r_1}}{r_1!}. \end{aligned}$$

This intermediate result requires more refinement to achieve the desired result. We know $m_i r_i \leq \eta \leq 2g + 2n$ for $i \in \{0, 1\}$ by the assumptions of the theorem. Therefore,

$$0.5 q^{\frac{m_i}{2}} \geq 0.5 q^{\frac{2 \log_q 3+4g+4n}{2}} = 1.5 + 2g + 2n > m_i r_i$$

holds due to the lower bound on m.

With this estimate the lower bound for $N(\eta, m)$ modifies to

$$N(\eta, m) \geq \frac{\left(\frac{q^{m_0} - q^{\frac{m_0}{2}}(1.5+2g+2n)}{m_0}\right)^{r_0}}{r_0!} \frac{\left(\frac{q^{m_1} - q^{\frac{m_1}{2}}(1.5+2g+2n)}{m_1}\right)^{r_1}}{r_1!}.$$

Similar to the above estimate, we know $m_i \geq \frac{\ln(1.5+4g+4n)}{\ln q}$ and thus

$$0.5q^{\frac{m_i}{2}} \geq 0.5q^{\frac{2\log_q 3+4g+4n}{2}} = 1.5 + 2g + 2n.$$

Inserting this into the last estimate simplifies the lower bound for smoothness to

$$N(\eta, m) \geq \frac{\left(\frac{q^{m_0}}{2m_0}\right)^{r_0}}{r_0!} \frac{\left(\frac{q^{m_1}}{2m_1}\right)^{r_1}}{r_1!} = \frac{q^{m_0 r_0 + m_1 r_1}}{2^{r_0+r_1} \cdot m_0^{r_0} r_0! \cdot m_1^{r_1} r_1!} \geq \frac{q^{\eta}}{4\left(m_0 r_0\right)^{r_0}\left(m_1 r_1\right)^{r_1}},$$

because $2^{r_i-1} r_i! \leq r_i^{r_i}$ for all $r_i \in \mathbb{N}$.

Finally, we can estimate $r_i m_i \leq \eta$ and $r_0 + r_1 = \lceil u \rceil$ to get $N(\eta, m) \geq \frac{q^{\eta}}{4\eta^{\lceil u \rceil}}$.

□

Remark 2.4.3. *For later use, we also require the special case $m = 1$. We see that for this special case, the requirement of Theorem 2.4.2 regarding m can be reduced to*

$$1 = m \geq 2\frac{\ln(3 + 4g + 4n)}{\ln q}.$$

This reduced requirement is sufficient because for $m = 1$ we also have $m_0 = m = 1$ and this lower bound for m_0 is sufficient throughout the proof.

Remark 2.4.4. *Similar to the previous remark, we can rephrase the conditions of the theorem to*

$$m_0 \geq 2\frac{\ln(3 + 4g + 4n)}{\ln q}.$$

We will later see that under certain conditions, we gain lower bounds for m and u. This will

allow us to improve the lower bound $m_0 \geq \frac{m}{2}$ that we used in the theorem.

Theorem 2.4.2 does not always guarantee a subexponential complexity in terms of q^η for the relation search. In order to achieve this, we refine the result by allowing more flexibility in the number of prime factors used, as well as their degrees.

The following theorem and its proof are adapted from Theorem 5 of [19], with modifications designed to extend its validity to non-hyperelliptic curves. Additionally, we adjust the approach to permit factor base elements of lower degree than in the original formulation. This latter adjustment is necessary to align our results with those of Theorem 2.3.4 for curves of low degree.

Theorem 2.4.5. *Let $\mathcal{F}$ be a dense factor base of size q^m divisors and let $u = \frac{\eta}{m} > \ln \eta$ for some integer $\eta > n$. Further, let $\frac{1}{2} \leq c \leq 1 - \frac{1}{m}$ be a parameter such that m and c satisfy*

$$\lfloor cm \rfloor \geq max \left\{ 4 \log_q (3 + 4g + 4n), 1 \right\}.$$

Then the number $N(\eta, \mathcal{F})$ of $\mathcal{F}$-smooth divisors of degree η is bounded below by

$$N(\eta, \mathcal{F}) \geq \frac{q^n}{u^{u(1+g(\eta,u))}}, \text{ where}$$
$$g(\eta, u) := \frac{2\frac{1-c}{c} \ln \eta + \ln \frac{2}{1-c} + 5 - \frac{\ln u}{u}}{\ln u}.$$

Proof. We will estimate the degree of elements in the factor base. Concretely, we know that the number of prime divisors of degree less than or equal to $\lfloor m \rfloor$ is less than $\frac{2}{\lfloor m \rfloor} q^m < q^m$ by Corollary 2.2.3. Since $\lfloor cm \rfloor \geq 1$, it implies that $m \geq 2$ by the allowed range for c. Consequently, $\mathcal{F}$ contains all convenient prime divisors of degree up to $\lfloor m \rfloor$ and $N(\eta, \mathcal{F}) \geq N(\eta, \lfloor m \rfloor)$.

We will now apply a variant of the approach of Theorem 5 of [19]. We define

$$\mathcal{F}_1 := \{ p \in \mathcal{F} \mid \deg p \leq \lfloor cm \rfloor \}, \text{ and}$$
$$\mathcal{F}_2 := \{ p \in \mathcal{F} \mid \lfloor u \rfloor \deg p \leq \eta \} \setminus \mathcal{F}_1.$$

The idea of this proof is to construct divisors that have $\lfloor u - 1 \rfloor$ prime factors from $\mathcal{F}_2$

and pad the resulting divisors with elements of smaller degree, which are $\mathcal{F}_1$-smooth, in order to reach the desired degree η. Note that $\mathcal{F}_2$ contains at least all primes with degree between $\lfloor cm \rfloor + 1 \leq \lfloor m \rfloor$ and $\lfloor m \rfloor$, because $c \leq 1 - \frac{1}{m}$ and $\lfloor u \rfloor \lfloor m \rfloor \leq \eta$.

Let I be the set of all divisors that are built by using $\lfloor u - 1 \rfloor$ prime factors taken from $\mathcal{F}_2$. Since $\mathcal{F}_1$ contains all prime divisors up to degree $\lfloor cm \rfloor$ and the two sets are disjoint, we conclude that

$$N(\eta, \mathcal{F}) \geq \sum_{D \in I} N(\eta - \deg D, \lfloor cm \rfloor) .$$

By the condition for m and c, we have $\lfloor cm \rfloor \geq 4 \log_q (3 + 4g + 4n)$ and the sub-factor base $\mathcal{F}_1$ satisfies the conditions of Theorem 2.4.2. Moreover, we have $\eta - \deg D \geq m > \lfloor cm \rfloor$, because all elements of I are built from at most $\lfloor u - 1 \rfloor$ factors. Therefore, all conditions of Theorem 2.4.2 are satisfied, and we get

$$N(\eta, \mathcal{F}) \geq \sum_{D \in I} \frac{q^{\eta - \deg D}}{4 \left(\eta - \deg D\right)^{\left\lceil \frac{\eta - \deg D}{\lfloor cm \rfloor} \right\rceil}} = \frac{1}{4} q^{\eta} \sum_{D \in I} \frac{q^{-\deg D}}{\left(\eta - \deg D\right)^{\left\lceil \frac{\eta - \deg D}{\lfloor cm \rfloor} \right\rceil}} .$$

To simplify the sum, we will first bound the denominator. We have

$$\left\lceil \frac{\eta - \deg D}{\lfloor cm \rfloor} \right\rceil \leq \left\lceil \frac{\eta - (\lfloor cm \rfloor + 1)(\lfloor u \rfloor - 1)}{\lfloor cm \rfloor} \right\rceil \leq \left\lceil \frac{\eta}{\lfloor cm \rfloor} - \lfloor u \rfloor + 1 - \frac{u - 2}{\lfloor cm \rfloor} \right\rceil ,$$

because the least degree of elements in I is $(\lfloor cm \rfloor + 1)(\lfloor u \rfloor - 1) \geq \lfloor cm \rfloor (\lfloor u \rfloor + 1)$.

By using $\frac{\eta}{m} = u$, we can rephrase the lower bound to

$$\begin{aligned} \left\lceil \frac{\eta}{\lfloor cm \rfloor} - \lfloor u \rfloor + 1 - \frac{u - 2}{\lfloor cm \rfloor} \right\rceil &= \left\lceil \frac{cm}{\lfloor cm \rfloor} \frac{\eta}{cm} - \lfloor u \rfloor + 1 - \frac{u - 2}{\lfloor cm \rfloor} \right\rceil \\ &= \left\lceil \frac{cm}{\lfloor cm \rfloor} \frac{1}{c} u - \lfloor u \rfloor + 1 - \frac{u - 2}{\lfloor cm \rfloor} \right\rceil . \end{aligned}$$

By inserting $\lfloor u \rfloor \geq u - 1$, $cm \leq \lfloor cm \rfloor + 1$, and $\frac{1}{c} - 1 = \frac{1 - c}{c}$ we get

$$\begin{aligned} \left\lceil \frac{cm}{\lfloor cm \rfloor} \frac{1}{c} u - \lfloor u \rfloor + 1 - \frac{u - 2}{\lfloor cm \rfloor} \right\rceil &\leq \left\lceil \left(1 + \frac{1}{\lfloor cm \rfloor}\right) \frac{1}{c} u - \left(1 + \frac{1}{\lfloor cm \rfloor}\right) u + \left(2 + \frac{2}{\lfloor cm \rfloor}\right) \right\rceil \\ &= \left\lceil \left(1 + \frac{1}{\lfloor cm \rfloor}\right) \frac{1 - c}{c} u + \left(2 + \frac{2}{\lfloor cm \rfloor}\right) \right\rceil . \end{aligned}$$

Thereby, the exponent in the denominator is bounded above by

$$\left\lceil \frac{\eta - \deg D}{\lfloor cm \rfloor} \right\rceil \leq u \left(\left(1 + \frac{1}{\lfloor cm \rfloor} \right) \frac{1-c}{c} + \frac{3}{u} + \frac{2}{u \lfloor cm \rfloor} \right) \leq u \left(2 \frac{1-c}{c} + \frac{5}{u} \right).$$

With the condition $u \geq \log \eta \geq \log \eta - \deg D$ for all $D \in I$, we change the basis of the denominator from $\eta - \deg D$ to u and achieve:

$$N(\eta, \mathcal{F}) \geq \frac{q^{\eta}}{\eta^{u\left(2\frac{1-c}{c}+\frac{5}{u}\right)}} \sum_{D \in I} q^{-\deg D} \geq \frac{q^{\eta}}{u^{u\left(\frac{2\frac{1-c}{c}\ln \eta + 5}{\ln u}\right)}} \sum_{D \in I} q^{-\deg D}.$$

To complete the proof, we have to find a lower bound for the sum. I holds all combinations of at most $\lfloor u - 1 \rfloor$ — not necessarily distinct — prime divisors from $\mathcal{F}_2$. Therefore,

$$\sum_{D \in I} q^{-\deg D} \geq \frac{1}{\lfloor u-1 \rfloor !} \left(\sum_{D \in \mathcal{F}_2} q^{-\deg D} \right)^{\lfloor u-1 \rfloor} \geq \frac{1}{\lfloor u-1 \rfloor !} \left(\sum_{k=\lfloor cm \rfloor}^{\lfloor m \rfloor} \Pi(k) q^{-k} \right)^{\lfloor u-1 \rfloor},$$

by sorting the divisors by degree.

By applying Corollary 2.2.3, this refines to

$$\sum_{D \in I} q^{-\deg D} \geq \frac{1}{\lfloor u-1 \rfloor !} \left(\sum_{k=\lfloor cm \rfloor}^{\lfloor m \rfloor} \frac{q^k}{2k} q^{-k} \right)^{\lfloor u-1 \rfloor} = \frac{1}{\lfloor u-1 \rfloor !} \left(\frac{1}{2} \sum_{k=\lfloor cm \rfloor + 1}^{\lfloor m \rfloor} \frac{1}{k} \right)^{\lfloor u-1 \rfloor}.$$

There are certain opportunities to bound the latter sum. For our purpose, the rough estimate

$$\sum_{k=\lfloor cm \rfloor + 1}^{\lfloor m \rfloor} \frac{1}{k} \geq \frac{\lfloor m \rfloor - (\lfloor cm \rfloor + 1) + 1}{\lfloor m \rfloor} \geq \frac{\lfloor m \rfloor - cm}{\lfloor m \rfloor}$$

is sufficient. We know $c \leq 1 - \frac{1}{m}$, and therefore $\sum_{k=\lfloor cm \rfloor + 1}^{\lfloor m \rfloor} \frac{1}{k} \geq \frac{1}{\lfloor m \rfloor}$ holds. With this lower bound, we get

$$2 \cdot \sum_{k=\lfloor cm \rfloor + 1}^{\lfloor m \rfloor} \frac{1}{k} \geq \frac{\lfloor m \rfloor - cm + 1}{\lfloor m \rfloor} \geq m \frac{1-c}{\lfloor m \rfloor} \geq 1 - c.$$

Therefore, we estimate the sum to be

$$\sum_{D\in I} q^{-\deg D} \geq \frac{1}{\lfloor u-1\rfloor!}\left(\frac{1}{4}(1-c)\right)^{\lfloor u-1\rfloor} \geq \frac{1}{2\left(\frac{1}{2}\right)^{u-1}(u-1)^{u-1}}\left(\frac{1}{4}(1-c)\right)^{\lfloor u-1\rfloor},$$

by applying the Stirling formula. Then we get

$$\sum_{D\in I} q^{-\deg D} \geq \frac{1}{2(u-1)^{u-1}}\frac{1}{\left(\frac{2}{1-c}\right)^{u-1}} \geq \frac{1}{u^{u-1}}\frac{1}{\left(\frac{2}{1-c}\right)^{u-1}} = \frac{1}{u^{(u-1)\left(1+\frac{\ln\frac{2}{1-c}}{\ln u}\right)}},$$

since $\left(\frac{x}{x+1}\right)^x \leq \frac{1}{2}$ for all $x \geq 1$. The combination of this result with the calculation of the smoothness for $\mathcal{F}_1$ finally gives

$$\begin{aligned} N(\eta,\mathcal{F}) &\geq \frac{q^n}{u^{u\left(\frac{2\frac{1-c}{c}\ln\eta+5}{\ln u}\right)}}\frac{1}{u^{(u-1)\left(1+\frac{\ln\frac{2}{1-c}}{\ln u}\right)}} \\ &\geq \frac{q^n}{u^{u\left(1+\frac{2\frac{1-c}{c}\ln\eta+\ln\frac{2}{1-c}+5-\frac{\ln u}{u}}{\ln u}\right)}}, \end{aligned}$$

which completes the proof. □

The value c introduced in Theorem 2.4.5 is a technical parameter included to simplify the estimation process. While c does not influence the actual smoothness probability of the group, it significantly affects the tightness of our bound. Consequently, we can select a value for c close to the optimum to maximize the derived lower bound on the number of smooth divisors of degree η.

Corollary 2.4.6. *Let $\mathcal{F}$ be a dense factor base of size q^m, and let $u = \frac{\eta}{m}$ for some integer η. Further, let $3 \leq \eta$ and let $m+1 \geq \ln\eta$ satisfy*

$$\left\lfloor \frac{\ln\eta}{\ln\eta+1} m \right\rfloor \geq \max\left\{4\log_q(3+4g+4n), 1\right\}.$$

Then the number $N(\eta, \mathcal{F})$ of $\mathcal{F}$-smooth divisors of degree η is bounded below by

$$N(\eta, \mathcal{F}) \geq \frac{q^n}{u^{u(1+g(\eta,u))}}, \text{ where}$$
$$g(\eta, u) := \frac{8.5 + \ln\ln\eta}{\ln u} - \frac{1}{u}.$$

Proof. By the bounds for η, we get $\ln\eta > 1$ and so $\frac{\ln\eta}{\ln\eta+1} > \frac{1}{2}$. Additionally, we have $m + 1 \geq \ln\eta$, and therefore $\frac{\ln\eta}{\ln\eta+1} \leq 1 - \frac{1}{m}$.

Then the corollary is a direct consequence of Theorem 2.4.5 with $c = \frac{\ln\eta}{\ln\eta+1}$. For this choice of c, we have $\frac{1-c}{c}\ln\eta = 1$ and $\ln\frac{2}{1-c} = \ln(2(1+\ln\eta)) < 1.5 + \ln\ln\eta$ for all $\eta \geq 3$. Inserting these bounds gives the desired result. □

To get a more condensed result, such as in Theorem 2.3.4 for small degrees, we will balance the factor base size with the smoothness probability. This gives a proper estimate for m.

Theorem 2.4.7. *Let C be an algebraic curve of genus g represented by the function field $\mathbb{F}_q(x, y)$ of degree n over $\mathbb{F}_q(x)$. Assume there is a way to generate divisors with known non-trivial decomposition of degree $\eta \leq 2g + 2n$. Moreover, assume that we can generate such divisors with pre-defined factors including the generator of the Jacobian.*

Furthermore, let $\mathcal{C}$ be the maximum of the individual cost functions for this divisor generation, trial factoring, and the logarithmic terms of the linear algebra phase.

Let $\theta := \frac{\eta}{\ln q}$ and assume $\sqrt{\frac{\theta}{2}}\sqrt{\ln\ln q^\eta} > 1$ and

$$\max\left\{0.5\left\lfloor\sqrt{\frac{\theta}{2}}\sqrt{\ln\ln q^\eta}\right\rfloor, 1\right\} \geq 2\log_q(3 + 4g + 4n).$$

Then there is an algorithm for solving any instance of the discrete logarithm problem over C in expected time

$$\mathcal{C}L_{q^\eta}\left[\frac{1}{2}, \sqrt{2} + o(1)\right].$$

Proof. We will use Algorithm 1.1.13 for solving the discrete logarithm problem. First, we will pick a dense factor base $\mathcal{F}$ of size $q^m = L_{q^\eta}\left[\frac{1}{2}, \frac{1}{\sqrt{2}}\right]$. For this choice m satisfies

$$m = \log_q L_{q^\eta}\left[\frac{1}{2}, \frac{1}{\sqrt{2}}\right] = \frac{\frac{1}{\sqrt{2}}\sqrt{\eta \ln q}\sqrt{\ln\ln q^\eta}}{\ln q} = \sqrt{\frac{\theta}{2}}\sqrt{\ln\ln q^\eta}.$$

Let $u := \frac{\eta}{m}$. Then we can also express u in terms of q and η:

$$u = \frac{\eta}{m} = \frac{\sqrt{2}\eta}{\sqrt{\theta}\sqrt{\ln\ln q^\eta}} = \sqrt{2}\sqrt{\eta}\frac{\sqrt{\ln q}}{\sqrt{\ln\ln q^\eta}} = \sqrt{2}\frac{\sqrt{\ln q^\eta}}{\sqrt{\ln\ln q^\eta}}.$$

To prove the desired time complexity while using Algorithm 1.1.13, we will prove that the smoothness probability is high enough for the selected factor base size. We will distinguish two cases.

First, we assume $m \leq 8\frac{\ln 3+4g+4n}{\ln q}$. Then the approximation

$$\eta^{\lceil u\rceil} = u^{\lceil u\rceil} \cdot m^{\lceil u\rceil} \leq u^{u(1+\frac{1}{u})\left(1+\frac{\ln(8\ln(3+4g+4n))}{\ln u \ln q}\right)} \in u^{u\cdot(1+o(1))},$$

holds. The final asymptotic expression is valid for $u \to \infty$. Note that the numerator of the fraction will also increase for increasing u due to $u < \eta \leq 2g + 2n$. Due to the double logarithm the assertion remains valid even when considering this dependency between the variables.

Since $u \to \infty \Leftrightarrow q^\eta \to \infty$, we conclude that the asymptotic term is $o(1)$ for increasing search space sizes in general.

We assumed $\max\left\{0.5\left\lfloor\sqrt{\frac{\theta}{2}}\sqrt{\ln\ln q^\eta}\right\rfloor, 1\right\} \geq 2\log_q 3 + 4g + 4n$. Therefore, we have either satisfied the conditions of Theorem 2.4.2 – because $\lfloor m\rfloor \geq 4\log_q 3 + 4g + 4n$ – or the conditions of Remark 2.4.3.

Therefore, we get

$$N(\eta, \mathcal{F}) \geq \frac{q^\eta}{u^{u(1+o(1))}}.$$

In the opposite case, where $m > 8\frac{\ln 3+4g+4n}{\ln q}$, all requirements of Corollary 2.4.6 are satisfied, and we obtain $N(\eta, \mathcal{F}) \geq \frac{q^\eta}{u^{u(1+o(1))}}$ as well. Again, replacing the function g by $o(1)$ is valid because m increases with the square root of η and therefore the double logarithm of η increases slower than the logarithm of u. Again, the $o(1)$ term is valid for $q^\eta \to \infty$ as long as q remains small enough compared to η to ensure the other assumptions of the

theorem still hold.

We see that $N(\eta, \mathcal{F}) \geq \frac{q^\eta}{u^{u(1+o(1))}}$ holds for both cases. By the representation of u we can estimate

$$u = \sqrt{2}\frac{\sqrt{\ln q^\eta}}{\sqrt{\ln \ln q^\eta}} \leq \sqrt{2}\sqrt{\ln q^\eta}$$

and following $\ln u \leq \frac{1}{2}\ln \ln q^\eta + \ln \sqrt{2}$.

The combination of the estimate with the previous representation for u gives

$$\begin{aligned}
\ln u^u = u \ln u &\leq \left(\frac{1}{2}\ln \ln q^\eta + \ln \sqrt{2}\right)\sqrt{2}\frac{\sqrt{\ln q^\eta}}{\sqrt{\ln \ln q^\eta}} \\
&\leq \frac{\sqrt{2}}{2}\sqrt{\ln q^\eta}\sqrt{\ln \ln q^\eta}\left(1 + \frac{0.35}{\ln \ln q^\eta}\right) \\
&\in \frac{\sqrt{2}}{2}\sqrt{\ln q^\eta}\sqrt{\ln \ln q^\eta}\,(1 + o(1))\,, \text{ for } q^\eta \to \infty.
\end{aligned}$$

By Definition A.1.1, we see that the probability for a randomly picked divisor of degree η to be $\mathcal{F}$-smooth is bounded below by $L_{q^\eta}\left[\frac{1}{2}, -\frac{\sqrt{2}}{2} + o(1)\right]$.

Therefore, we expect to generate $L_{q^\eta}\left[\frac{1}{2}, \frac{\sqrt{2}}{2} + o(1)\right]$ random divisors for each smooth divisor. Since we require $L_{q^\eta}\left[\frac{1}{2}, \frac{1}{\sqrt{2}}\right]$ smooth divisors, we can estimate the overall runtime to be

$$\mathcal{C}L_{q^\eta}\left[\frac{1}{2}, \frac{1}{\sqrt{2}} + \frac{\sqrt{2}}{2} + o(1)\right] = \mathcal{C}L_{q^\eta}\left[\frac{1}{2}, \sqrt{2} + o(1)\right].$$

□

One interesting aspect of the proof of Theorem 2.4.7 is that the optimal value of u depends on the quantity q^η only. Therefore, we can conclude that once q^η is large enough, we can relax the requirements according to Remark 2.4.4.

Lemma 2.4.8. *Given the setting of Theorem 2.4.7, and let $q^\eta > 2^{114}$. Assume that*

$$\begin{aligned}
\theta &\geq \frac{3\sqrt{2}}{5} + \frac{6\sqrt{2}}{5}\log_q(3 + 4g + 4n) \\
&< 0.85 + 2\log_q(3 + 4g + 4n)\,.
\end{aligned}$$

Then the search for general relations in Algorithm 1.1.13 can be completed in expected time

$$\mathcal{C}L_{q^\eta}\left[\frac{1}{2}, \sqrt{2} + o(1)\right].$$

Proof. For $q^\eta > 2^{114}$, we get that $u > 6$ by the formula found in the proof of Theorem 2.4.7. Similarly, we calculate $\sqrt{\ln \ln q^g} > 2$.

Then we can compute

$$m_0 \geq \lfloor \frac{u-1}{u} \rfloor m \geq (\frac{u-1}{u} - \frac{1}{m})m \geq \left(\frac{5}{6} - \frac{1}{m}\right) m$$

in Theorem 2.4.2.

Following Remark 2.4.4, we therefore can relax the conditions of Theorem 2.4.2 to

$$\left(\frac{5}{6} - \frac{1}{m}\right) m \geq 2 \log_q (3 + 4g + 4n).$$

We achieve the refined condition on θ by inserting the estimate for $\sqrt{\ln \ln q^g}$.

With this relaxed version of Theorem 2.4.2, we can reproduce the proof of Theorem 2.4.7. In this context, the conditional and its second case are not affected by the changed assumptions. □

We will give a version of Theorem 2.4.7 for hyperelliptic curves and divisor generation by creating and decomposing randomized divisors. In this case, the reduced divisors are of degree $\eta = g$ and $n = 2$.

Corollary 2.4.9. *Let C be a hyperelliptic curve of genus g with exact field of constants $\mathbb{F}_q$. Define $\theta := \frac{g}{\ln q}$ and assume*

$$\max\left\{0.5\sqrt{\frac{\theta}{2}}\sqrt{\ln \ln q^g}, 1\right\} \geq 2 \log_q (11 + 4g), \text{ and}$$
$$\sqrt{\frac{\theta}{2}}\sqrt{\ln \ln q^g} > 1.$$

Then there is a choice of factor base such that Algorithm 1.2.50 solves any instance of the

discrete logarithm problem over C with expected time complexity

$$L_{q^g}\left[\frac{1}{2}, \sqrt{2} + o(1)\right].$$

2.5. Discussion of Results

The two independent estimates for divisor smoothness offer a distinct advantage over existing results, particularly as they asymptotically coincide when $\sqrt{\frac{\theta}{2}}\sqrt{\ln\ln q^\eta} \to 1$. By narrowing the analysis of the factor base size to a tight interval around the presumed optimum, we achieve sharper estimates for small η and extend the range of validity for larger values compared to [19]. Furthermore, since our arguments rely on general combinatorial formulas for counting convenient prime divisors, these estimates remain valid for a broad class of curves. We synthesize these partial results into the following theorem, which provides robust and generalized smoothness bounds.

Theorem 2.5.1. *Let C be an algebraic curve of genus g represented by the function field $\mathbb{F}_q(x,y)$ of degree n over $\mathbb{F}_q(x)$. Fix a degree η such that we can create random divisors of degree η and let $\theta := \frac{\eta}{\ln q}$ as above.*

Assume

$$\max\left\{0.5\left\lfloor\sqrt{\frac{\theta}{2}}\sqrt{\ln\ln q^\eta}\right\rfloor, 1\right\} \geq 2\log_q(3 + 4g + 4n).$$

Then, for each dense factor base $\mathcal{F}$ of size $L_{q^\eta}\left[\frac{1}{2}, \frac{\beta}{2}\right]$ the probability for a randomly picked divisor of degree η to be $\mathcal{F}$-smooth is bounded below by $L_{q^\eta}\left[\frac{1}{2}, -\frac{\beta}{2} - o(1)\right]$, where

$$\beta = \begin{cases} \frac{\sqrt{2}}{\sqrt{\frac{\theta}{2}}\sqrt{\ln\ln q^\eta}} & \sqrt{\frac{\theta}{2}}\sqrt{\ln\ln q^\eta} < 1 \\ \sqrt{2} & \sqrt{\frac{\theta}{2}}\sqrt{\ln\ln q^\eta} \geq 1. \end{cases}$$

The theorem does not require a dedicated proof, as it follows directly from the combination of Theorems 2.3.4 and 2.4.7. Note that the technical assumption of the theorem is satisfied whenever η or q is sufficiently large. For example, if we fix a curve model such that the genus g, degree n, and η remain constant, then for variable field sizes q, the condition

is met whenever $1 \geq 2\log_q(3+4g+4n)$. Conversely, if q is fixed, the condition implies a lower bound on η for the theorem to hold. In general, these technical assumptions are satisfied provided the DLP instance meets a minimum size requirement.

The figure below compares this theorem with results from the literature, specifically those of [16] (light gray line) and [19] (black dashed line), for hyperelliptic curves. In this specific example, we assume $\eta = g$, as most existing implementations use a random sequence of divisors reduced by Cantor's algorithm [8] to effective divisors of degree g prior to trial division. Furthermore, to incorporate the results of [16], we assume $q^g \approx 2^{80}$, since the formulas in the original work are parametrized solely by θ and do not depend on the size of the Jacobian.

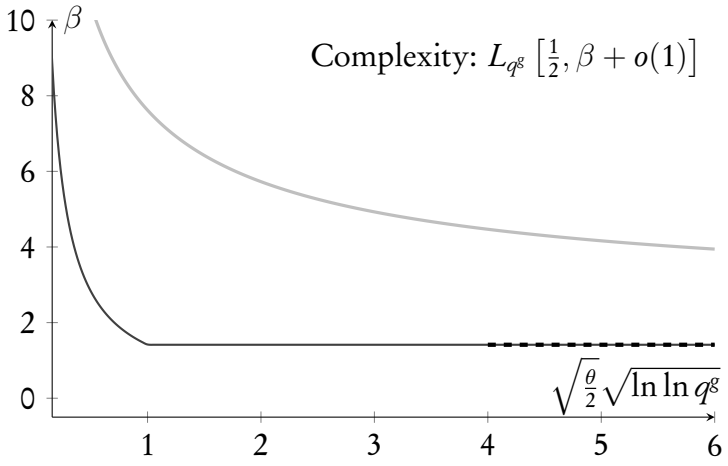

Figure 2.1.: Comparison of Theorem 2.5.1 with literature results.

We emphasize that, while we have not introduced an algorithm to generate smoother elements, we have successfully sharpened the theoretical estimates to close the gap between previous approximations. This improvement was achieved in both cases — for relatively high and low values of η — by focusing on factor bases near the presumed optimum and incorporating the double-logarithmic term in our analysis more thoroughly than was previously done. For hyperelliptic curves and rather low values of η, this refinement allows us to state that the complexity reaches $L_{q^g}\left[\frac{1}{2}, \sqrt{2} + o(1)\right]$ for $\sqrt{\frac{\theta}{2}}\sqrt{\ln\ln q^\eta} \to 1$.

Additionally, we have reduced the genus requirements compared to Theorems 3 and 5 of [19]. In the original work, a factor base degree m satisfying $m \geq 2 + 8\log_q(2g + 6\sqrt{2})$ is

required. For curves with an 80-bit Jacobian, this implies

$$\sqrt{\frac{\theta}{2}}\sqrt{\ln \ln q^g} = m > 2 + 8\frac{\ln 2g + 6 + \sqrt{2}}{80 \ln 2}.$$

This formula yields $m > 4$ for every curve of genus $g \geq 5$. Therefore, we marked this as the lower bound of the validity domain in Figure 2.1. In contrast, our estimates for higher values of η only require $m = \sqrt{\frac{g}{2 \ln q} \ln \ln q^g} > 1$.

We believe that these sharper results are also useful in contexts other than hyperelliptic curves to obtain sharper bounds or relaxed requirements. For example, in [18], Enge, Gaudry, and Thomé introduced a family of non-hyperelliptic $C_{n,d}$ curves with $n \approx g^{\frac{1}{3}}$. They provided an algorithm to generate divisors of degree around $g^{\frac{2}{3}}$ and used a smoothness estimate by Hess [27] requiring $m \geq 3 \log_q(4 + 14g)$.

The result of Theorem 2.5.1 applies to the setting in [18] and serves as an alternative smoothness estimate. In doing so, we can relax the requirement to $m \geq 3 \log_q 3 + 4g + 4n$, where $g^{\frac{1}{3}} \leq n \leq g^{\frac{1}{2}}$, by following the arguments in Lemma 2.4.8 and Remark 2.4.4 for the original curves.

Given the structure of Theorem 2.5.1, we claim that there is also a family of low-genus curves with $n \approx g^{\frac{1}{3}}$, such that the discrete logarithm problem on this family of curves has an estimated complexity of $L_{q^g}\left[\frac{1}{3}, \beta\right]$ with β increasing for lower genera.

In the following chapter, we introduce algorithms for solving discrete logarithms that allow an explicit computation of the degree of divisors to be trial-divided. We will demonstrate that, using this new analysis, there exists a finite class of $C_{n,d}$ curves on which the discrete logarithm problem can be solved within an expected time of less than $L_{q^g}\left[\frac{1}{3}, O(1)\right]$. In general, this technique, which is based on the findings of [62], offers a guaranteed upper bound for η for all $C_{n,d}$ curves.

3

Explicit Methods for Computing Discrete Logarithms over $C_{n,d}$ Curves

In this chapter, we provide a comprehensive complexity analysis of the Discrete Logarithm Problem (DLP) on $C_{n,d}$ curves by detailing the sub-algorithms required for Algorithms 1.1.13 and 1.1.15. Specifically, we introduce novel methods for constructing divisors of a prescribed degree η with a known non-trivial decomposition over the factor base, as well as techniques for decomposing them a second time. The combination of these two distinct decompositions is sufficient for the index calculus algorithms mentioned above. Our approach leverages the divisor representation defined in Section 1.2.5 and builds upon the preliminary sieving scheme outlined in Algorithm 1.2.54.

The chapter is structured as follows:

First, we briefly review the literature on comparable sieving methods and other relevant sub-algorithms, focusing primarily on the theorems and concepts essential for our subsequent analysis.

We next address the core components of Algorithm 1.1.15 — specifically, the sieve initialization and the sieve stepping– whose performance is heavily dependent on the factor base.

These steps are critical due to a dual dependency: the structure of the factor base elements determines the arithmetic complexity of each operation, while the factor base size governs both the smoothness probability and the required execution frequency.

As demonstrated in the previous chapter, the relationship between the factor base size and the problem size becomes most challenging when the curve degree is small relative to the finite field size. To address this, Section 3.3 introduces and analyzes a new large-prime scheme tailored to this scenario.

Finally, we analyze the auxiliary sub-algorithms required for Algorithm 1.1.15, such as the factor base construction and the computation of the two special relations. Although these routines are typically executed only once, their asymptotic complexity requires investigation, particularly in cases where generating general relations over a small factor base is significantly cheaper than forcing specific factors into the two special relations.

3.1. Existing Approaches for Solving Discrete Logarithms in Jacobians

Before presenting our general framework, we summarize the state of the art for specific subclasses of $C_{n,d}$ curves. The vast majority of prior complexity analyses have been limited to hyperelliptic curves, and thus they form the bulk of this review. Nevertheless, we also highlight the existing works that extend beyond this case, focusing on the specific results that inspired the generalizations proposed in the following sections.

3.1.1. The Sieve of Velichka, Jacobson, and Stein

In their paper, Velichka, Jacobson, and Stein [62] construct a sieving algorithm for solving the discrete logarithm problem on imaginary model hyperelliptic curves of large genus in even characteristic. The general construction of the sieve follows our description provided in Algorithm 1.2.54.

The authors carry out the norm computation of Algorithm 1.2.54 explicitly by using the fact that function fields associated with hyperelliptic curves are of degree $n = 2$ over $\mathbb{F}_q(x)$.

Theorem 3.1.1 (Section 4.1 of [62]). *Let q be a power of 2. Let the hyperelliptic curve be described by the equation $y^2 + hxy + f(x)$ with $h, f \in \mathbb{F}_q[X]$ of degree $2g+1$ and let $I = \langle a, y - b\rangle$ be an ideal as given in Algorithm 1.2.54.*

Then I has rank 2 as an $\mathbb{F}_q[x]$-module and for transcendental S, T the sieve polynomial as defined in Theorem 1.2.53 has the form

$$f_{(a,b)}(S,T) = \frac{\mathcal{N}(aS + (b-y)T)}{a} = aS^2 + hST + \frac{b^2 - h - f}{a}T^2 \in \mathbb{F}_q[x,S,T].$$

We note that for hyperelliptic function fields, the computation of the norm in the formula can be done quickly by multiplying the element by its conjugate. By using this explicit formula, the authors compute the degree of divisors associated with the sieve polynomial explicitly. In their analysis, they restrict themselves to evaluating T to 1 and also give an optimal choice for the degree of a for this setup.

Theorem 3.1.2 (Section 4.2 of [62]). *Given the setting of Theorem 3.1.1 and let $s \in \mathbb{F}_q[x]$ of degree bounded by M. Then if $\deg a = g - M$ the sieve polynomial evaluated at $(s,1)$ satisfies*

$$\deg f_{(a,b)}(s,1) \leq g + 1 + M,$$

which is the optimum possible for the selected size of M.

We intentionally omit the proofs for these theorems, as they inspire a similar construction that will be examined in a broader context later in this chapter. Using the construction described in the theorems, the authors follow the approach in Algorithm 1.1.13, combined with 1.2.54, to solve the discrete logarithm problem on hyperelliptic curves of sufficiently large genus.

In their specific setup for the sieving process, the authors of [62] compute a by multiplying distinct low-degree factor base elements until reaching the target degree of $g - M$. Next, they compute the corresponding b using the Chinese remainder theorem. For initializing the sieve — i.e., calculating the roots of the sieve polynomial modulo the remaining factor base elements as outlined in 1.2.55 — they apply a generalized version of the Shanks-Tonelli

algorithm [53]. This algorithm has a quadratic runtime relative to the representation of the factor base elements. Additionally, they employ a quadratic runtime algorithm for stepping through the sieve array using standard polynomial arithmetic.

The authors do not conduct an explicit complexity analysis of the logarithmic terms in their work. Instead, they offer a runtime comparison between their approach and an implementation of Algorithm 1.2.50, as presented in [32]. The comparison suggests that the computational costs for initialization and sieving are lower than those for trial dividing the same number of randomized divisors created by the group law on a high-genus hyperelliptic curve.

A key factor in their performance gain is the use of a concept referred to as *self-initialization*, which minimizes the time required to initialize the sieving process.

Theorem 3.1.3 (End of section 4.1 in [62]). *Let a hyperelliptic curve be defined by $y^2 + hxy + f(x) = 0$ over $\mathbb{F}_q$ with even q and let $q_1, \ldots q_r \in \mathbb{F}_q[x]$ be prime polynomials that split in $\mathbb{F}_q[x, y]$.*

Then there are 2^{r-1} ideals $I_i = \langle a, y - b_i \rangle$ and thus sieve polynomials that may yield different relations sharing the same $a := \prod_{i=1}^{r} q_i$.

Now fix $1 \leq k \leq r$ and let t_k, t_k' be the two solutions of $y^2 + hxy + f(x) = 0 \pmod{q_k}$. Then define

$$B_k := \left(\frac{a}{q_k}\right)\left(\left(\frac{a}{q_k}\right)^{-1} t_k \pmod{q_k}\right) \pmod{a_i}, \text{ and}$$

$$B_k{}' := \left(\frac{a}{q_k}\right)\left(\left(\frac{a}{q_k}\right)^{-1} t_k' \pmod{q_k}\right) \pmod{a_i}.$$

Now let $p \in \mathcal{F}' \setminus \{q_i \mid 1 \leq i \leq r\}$ and assume s is a root of any $F_{(a,b_i)}(S, 1) \pmod{p}$. Then

$$s + a^{-1}(B_k + B_k{}') \pmod{p}$$

is the root of one $f_{(a,b_j)}$, $j \neq i$. More precisely, I_j corresponds to the ideal that is equal to I_i up to the root for the k-th prime polynomial in the decomposition of a.

We will later derive a similar construction that is not limited to hyperelliptic curves or

fields of characteristic 2. The theorem allows us to compute the roots of all 2^{r-1} sieve polynomials for all factor base elements at the cost of computing one root and all the $a^{-1}(B_k + B_k{}')$, which are at most r elements for each factor base element. On average, this gives a low initialization cost, which was one of the key components of the sieve of [62] for outperforming prior algorithms.

3.1.2. The Sieve of Sarkar and Singh

One variant of the work of Velichka, Jacobson, and Stein [62] is due to Sarkar and Singh [51]. The variant uses a similar construction of sieve polynomials for hyperelliptic curves of low genus in odd characteristic. The different setting requires the explicit formulas to be changed, because the generic equation for the norm on such hyperelliptic curves differs. Additionally, the factor base consists exclusively of divisors of degree one, as in the scenario covered in Section 2.3.

For the relation search, the main difference is that they do not sieve for relations directly. Instead, they search the list of sieve polynomial roots mod factor base elements for duplicates. It turns out that this is equivalent to sieving with $M = 0$. To reduce the cost for computing the sieve polynomial roots, they use a pre-computed table of square roots in $\mathbb{F}_q$. The pre-computation causes additional costs for the computations themselves and additional memory consumption, but the number of sieve polynomials that profit from these computations is high enough to compensate for the extra costs. Finally, they use a double-large-prime variant as proposed by Gaudry [25]. We will discuss this variant later in Section 3.1.4. The search for double-large-prime relations causes their tolerance value T as described in Algorithm 1.2.54 to be $g - 1$.

3.1.3. $L_{q^g}\left[\frac{1}{3}, O(1)\right]$ Discrete Logarithms of Gaudry, Enge, and Thomé

Gaudry, Enge, and Thomé [18] were the first to present examples of discrete logarithm problems in Jacobians of $C_{n,d}$ curves, which are of subexponential time complexity of $L_{q^g}\left[\frac{1}{3}, O(1)\right]$.

The authors prove that for $C_{n,d}$ curves with specific limits on n and d, it is possible to

find divisors on the curve which have a norm significantly lower than g. More precisely, they state the following theorem.

Theorem 3.1.4 (Section 2.1 of [18]). *Let F be the function field of a $C_{n,d}$ curve, and let $\alpha \in F$. Then*

$$\deg_x \mathcal{N}_{\mathbb{F}_q(x)\subset F}(\alpha) \leq n \deg_x \alpha + d \deg_y \alpha.$$

For curves where d is close to $g^{\frac{2}{3}}$ and thus $n \approx g^{\frac{1}{3}}$, the resulting norm-degrees can be as low as $g^{\frac{2}{3}}$. As highlighted in Theorem 2.4.7, this leads to an increased smoothness probability, even with small factor bases, if the degree of the curve is sufficiently large.

The approach of Gaudry, Enge, and Thomé involves generating suitable α values and trial-factoring them over a factor base until enough general relations have been found. While this is a straightforward approach for the general relations, the main challenge in performing Algorithm 1.1.13 lies in producing the two special relations that decompose the group generator and the target element over the factor base. This task is more complex than in the hyperelliptic case because the degree difference between the largest factor base elements and the natural representation of divisors (after applying the group law) is greater. Consequently, the smoothness probability of decomposing a linear combination of these special elements over the factor base is significantly lower.

To address this challenge in Algorithm 1.1.13, they use a method known as *Special-q Descent*, described in the following algorithm.

Algorithm 3.1.5 (Special-q Descent for Algebraic Curves, section 2.2 of [18]).

Input: A divisor represented by (a, b) with $a, b \in \mathbb{F}_q[x]$, $\deg b < \deg a$ satisfying $\deg a = g^{\frac{1}{3}+\tau}$ for $0 \leq \tau \leq \frac{2}{3}$ over a $C_{n,d}$ curve with $g^{\frac{1}{3}} < d < g^{\frac{2}{3}}$.

Output: A list of divisors of degree at most $g^{\frac{1}{3}+\frac{\tau}{2}}$ such that the sum of divisors in the list equals the input divisor in the Jacobian of the curve.

1: Compute $k = g^{(\log_g n)-\frac{1}{3}+\frac{\tau}{2}}$.
2: **for** $\alpha \in as_0 + \sum_{i=1}^{k} s_i (y-b)^i$, $s_0, \ldots, s_{n-1} \in \mathbb{F}_q[x], \deg_x \alpha < g^{2/3-k+\frac{\tau}{2}}$ **do**
3: Factor $\mathcal{N}(\alpha) \in \mathbb{F}_q[x]$. Let L be the set of factors found.

4: **if** $\max\{\deg p \mid p \in L\} \leq g^{\frac{1}{3}+\frac{\tau}{2}}$ **then**
5: Reduce $b \mod p$ to compute a list L' of divisors $(p, b \pmod p)$ with $p \in L$.
6: Return the list L' and terminate.
7: **end if**
8: **end for**

We can obtain a full factorization of a given divisor by applying the algorithm iteratively on the elements of the found list. The authors show that, for the given parameters, the elements within the search space for factoring the ideal have a norm degree of approximately $g^{\frac{2}{3}+\frac{\tau}{2}}$. Furthermore, they provide arguments that the search space is large enough and that one can expect to find a decomposition in the first $q^{g^{\frac{1}{3}}}$ trials.

The results presented in [18] assume very large genera, meaning the associated degree bounds and smoothness estimates hold asymptotically. In this work, we adapt the underlying concept of Algorithm 3.1.5 to formulate a variant applicable to a significantly wider range of curves. Furthermore, we leverage the examples provided in [18] to demonstrate that a sieving-based approach is indeed suitable for non-hyperelliptic curves.

3.1.4. Large Prime Methods for Low Degree Curves

While we have already laid the foundations for the literature discussed above in Chapter 1, this has not yet been done for the so-called large-prime methods. Therefore, we will briefly motivate the approach.

The idea behind the mentioned algorithms is to allow a limited number of factors that are not part of the factor base when factoring the ideals. These missing factors must then be resolved at a later stage using another partial relation with the same factor. This approach increases the weight of the relation matrix, but with the advantage of obtaining a higher smoothness probability.

The methods described in this section were originally introduced in [60] and [25] for imaginary hyperelliptic curves with a fixed, low genus. As concluded by Diem in [14], these algorithms are also applicable to curves in a generality even exceeding that of $C_{n,d}$ curves by combining the algorithms with those of Heß [27]. However, this generality comes at the expense of the provable complexity class of these algorithms. In our later analysis, we will

prove that the sieving approach of Velichka, Jacobson, and Stein [62] can be combined with the large-prime methods for $C_{n,d}$ curves and that the combination improves the provable complexity for non-hyperelliptic $C_{n,d}$ curves.

Definition 3.1.6. In the context of Algorithm 1.1.13, we refer to a relation with a complete factor base decomposition as a *full relation*. In contrast to these relations, we will allow relations with factors outside of the factor base for the rest of this section.

We will refer to relations as *single-large-prime* or *double-large-prime* if they incorporate one or two elements outside of the factor base, respectively.

Allowing prime factors outside of the factor base significantly increases the smoothness probability, particularly for curves of low degree.

Lemma 3.1.7. *Consider the setup of Theorem 2.3.1. Further, assume that a large-prime relation consists of primes of degree one only. Then the probability of gaining a $\mathcal{F}$-smooth relation incorporating $k \leq \eta$ large-prime factors equals*

$$\frac{1}{k!(\eta-k)!} q^{(1-r)(\eta-k)} \kappa^k,$$

with $\kappa := \frac{q-\#\mathcal{F}}{q}$.

Proof. The proof is purely combinatorial. There are $\frac{1}{(\eta-k)!} q^{(\eta-k)\cdot r}$ divisors of degree $(\eta-k)$ that are $\mathcal{F}$-smooth. Furthermore, there are about $\frac{1}{k!}(q-\#\mathcal{F})^k$ divisors of degree k that decompose to degree one prime divisors but none of the k prime divisors are in $\mathcal{F}$. This assertion is valid due to the estimate that there are about q prime divisors of degree one over the function field of interest.

The statement now follows from multiplying both estimates and dividing by the total number of divisors of degree η, which is approximately q^η. □

In scenarios where the lemma approximates the ideal factor base size, $\#F$ is often negligible compared to q, implying $\kappa \approx 1$. For low-degree curves, this results in a probability of generating large-prime relations that is higher by a factor of $\eta q^{1-r}\kappa$ compared to full relations. Because this factor is exponential in q^η, the rate of discovering large-prime relations

significantly outpaces that of discovering ordinary relations, making these methods highly effective for low-degree curves.

Large-prime relations cannot be used to solve DLPs directly. Instead, they need to be resolved to full relations beforehand. The following single-large-prime algorithm was formalized by Thériault [60] and can be read as a direct extension using single-large-prime relations in Algorithm 1.2.50.

Algorithm 3.1.8 (Thériault [60])**.**

Input: Two divisors α and γ on an algebraic curve such that α is an element of the cyclic subgroup generated by γ. Furthermore, a factor base $\mathcal{F}$ such that $\mathcal{F}$ contains only a subset of the prime divisors of degree one.

Output: The discrete logarithm $\log_\gamma \alpha$ in the Jacobian of the curve.

Initialize empty arrays R for relations and L for partial relations.
Initialize a random walk starting with $D = z'a + zg$ for some $z, z' \in \mathbb{Z}$.
while $\#R < \#\mathcal{F}$ **do**
  Compute $D \leftarrow D + s'a + sz$ with $s, s' \in \mathbb{N}$ chosen pseudo-randomly.
  Reduce D to be of degree less than g.
  Trial-factor D over $\mathcal{F}$.
  if D decomposes to a single-large-prime relation **then**
    Search L for the large-prime factor included in the relation
    if L contains the large-prime factor **then**
      Create a full relation by inserting the decomposition of the large-prime factor by D into the list element.
      Store the newly created full relation in R.
    else
      Store the found large-prime relation in L.
    end if
  else if D decomposes to a full relation **then**
    Store the found full relation in R.

end if

end while

Solve a linear system as described in Algorithm 1.1.12.

The algorithm executes the relation search more efficiently than a naive search for full relations over $\mathcal{F}$. Therefore, the algorithm allows us to balance the ratio r to a lower value than in the full relation search approach, because the relation search time decreases with the factor base size, while the linear algebra complexity increases with it.

Theorem 3.1.9 (Theorem 3 of [60]). *Let C be a hyperelliptic curve of genus g over $\mathbb{F}_q$ satisfying $g! < \frac{q}{2}$. Then Algorithm 3.1.8 with C as an input curve has an expected runtime of*

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

bit operations when r is of optimal value $\frac{g-\frac{1}{2}+\log_q\left(\frac{(g-1)!}{g}\right)}{g+1}$.

We will provide only a sketch of the proof because we will provide a full proof of a similar result later in Section 3.3.2.

Sketch of proof. The low impact of g on the logarithmic components of the complexity bound is due to the different sensitivity of the algorithm phases to the factor base size. While the relation collection phase becomes faster for growing $\mathcal{F}$ with the $(g-1)$-th power, the linear algebra phase increases the runtime only quadratically by Theorem A.4.14. By Proposition 1 of [60], the relation search phase has an expected runtime of

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

where the initial $g^2 (\ln q)^2$ describes the cost for divisor arithmetic in the random walk and the trial factorization for every tested divisor. The proof provided by [60] already includes an estimate of the required number of single-large-prime relations to be found to combine sufficiently many full relations.

Each row in the resulting linear system has an average weight $O(g)$. Furthermore, all operations of the Wiedemann algorithm (see Algorithm A.4.11) are in a ring with $g\left(\log_2 q\right)$

bits in its representation. Therefore, the expected runtime for the linear algebra phase is given by

$$O\left(g^3 \cdot (\ln q)^2 \cdot q^{2r}\right).$$

Balancing these two runtimes by adjusting r gives an optimal value of

$$r = \frac{g - \frac{1}{2} + \log_q\left(\frac{(g-1)!}{g}\right)}{g+1} \leq \frac{g-\frac{1}{2}}{g+\frac{1}{2}} + \frac{g \log_q g}{g+\frac{1}{2}} < \frac{g-\frac{1}{2}}{g+\frac{1}{2}} + \log_q g.$$

Inserting this estimate into the formula for the linear algebra phase runtime directly gives the desired result. Note that the balancing will only give a value of $r < 1$ if $(g-1)! < q$. This condition is mandatory for the analysis of the algorithm. □

The single-large-prime variant of Thériault was the first algorithm that allowed a faster computation of discrete logarithms on hyperelliptic curves of genus 3 than any generic algorithm. In a later work, Diem, Gaudry, Thériault, and Thomé [25] presented two algorithms that allow the use of double-large-prime relations for further improvements of the complexity of the relation search.

We present the second of these two algorithms, which allows a rigorous analysis.

Algorithm 3.1.10 (Section 3.2 of [25])**.**

Input: A factor base $\mathcal{F}$ such that $\mathcal{F}$ contains only a subset of the prime divisors of degree one.

Output: A list R containing $\#\mathcal{F}$ relations over $\mathcal{F}$

1: Initialize a list $L = \emptyset$ for large-prime factors and set $R = \emptyset$.
2: Create random divisors and trial-factor them until we find one divisor D_0 that is single-large-prime over $\mathcal{F}$.
3: Let p_0 denote the large-prime factor of D_0. Then add the pair (p_0, D_0) to L.
4: **while** $\#R < \#\mathcal{F}$ **do**
5: Create a random divisor D and trial-factor it over $\mathcal{F}$.
6: **if** D is a double-large-prime relation over $\mathcal{F}$ **then**
7: Check whether one of the large-prime factors of D is an index in L. If not,

continue with the next D.

8: **if** Both large-prime factors of D are indexes in L **then**

9: Resolve both large-prime factors of D by replacing them with the relations stored in the corresponding second component in L. Repeat this process until the tree is traversed backward to the root (p_0, D_0).

10: Store the obtained full relation in R.

11: **else**

12: Let p be the large-prime factor of D that is not an index in L. Then add the pair (p, D) to L.

13: **end if**

14: **end if**

15: **end while**

The algorithm has the advantage that one only needs to find a single-large-prime relation, which is less time-consuming than the full search in Algorithm 3.1.8. Once found, the algorithm considers double-large-primes that contain at least one large-prime factor that can be decomposed over the factor base. This creates a tree-structured graph of quasi-single-large-prime relations gained from restricted double-large-prime relations. Full relations are generated whenever leaves in the tree connect.

Theorem 3.1.11 (Equations (4) and (5) of [25]). *Given a hyperelliptic curve of genus g over $\mathbb{F}_q$, let $\#\mathcal{F}$ be a factor base of size q^r for some $0 \leq r < 1$. Assume that $\#\mathcal{F}$ is negligible with regard to q. Finally, let C be the combined cost of creating randomized divisors of degree g and trial-factoring them over $\mathcal{F}$. Then Algorithm 3.1.10 has an expected runtime of*

$$C \cdot (g-2)! q^{1+(1-r)(g-2)} \ln q$$

binary operations.

Furthermore, the depth of the tree-structured graph is approximately $1 + \ln q$.

We see that compared to the single-large-prime algorithm, we add an extra complexity factor of $\ln q$ to both the linear algebra phase and the relation search phase. This is due to

the extra effort required to create the graph for the relation search and the increased density of the matrix columns when solving the linear system.

This additional algorithmic effort is compensated for by a smaller exponent of q compared to the single-large-prime algorithm, since $1 + (g-2)(1-r) < \frac{1+r}{2} + (g-1)(1-r)$ for any $r < 1$. Thus, the relation collection phase becomes more efficient. As shown in [25], this efficiency permits balancing towards a smaller r, yielding a smaller factor base and improved overall complexity versus the single-large-prime variant, despite the extra effort to create the large-prime graph.

Theorem 3.1.12 (Theorem 21.22 of [11]). *Consider the definitions and assumptions of Theorem 3.1.11. Further, assume that a multiplication algorithm of complexity $O\left((\ln q)^2\right)$ is used in $\mathbb{F}_q$. Then the complexity of solving any instance of the discrete logarithm problem over the hyperelliptic curve is bounded by*

$$O\left(g^5 (\ln q)^3 q^{2-\frac{2}{g}}\right)$$

bit operations by using Algorithm 3.1.10 as a general relation search routine for Algorithm 1.1.13.

We omit the proofs of Theorems 3.1.11 and 3.1.10 to focus on the new large-prime variant we presented in Section 3.3. Section 3.3.2 provides a detailed analysis of this new algorithm, employing the same proof techniques required for the omitted proofs.

As mentioned previously, Diem [14] generalized this strategy for non-hyperelliptic curves. Note that Diem used $\widetilde{O}$ notation to describe the complexity, thereby omitting the explicit calculation of logarithmic factors.

Theorem 3.1.13 (Theorem I of [14]). *Let some natural number $g \geq 2$ be fixed. Then the discrete logarithm problem in the degree 0 class groups of curves of genus g over finite fields can be solved in an expected time of*

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

where $\mathbb{F}_q$ is the ground field of the curve.

We will prove that, given the methods of the remainder of the chapter, we can improve on this complexity in the case of non-hyperelliptic $C_{n,d}$ curves.

3.2. Sieving for General Relations over a Factor Base

3.2.1. Construction of Sieve Polynomials

The strategy we choose for creating sieve polynomials is similar to that of Velichka, Jacobson, and Stein [62] that we described in Theorem 3.1.1. The idea is to create ideals from given divisors that allow a direct computation of their norms and give rise to sieve polynomials with strict bounds on the degree, in order to keep the smoothness probability high and predictable.

As described in Section 1.2.2, we have to compute the relative norm of ideals in $\mathbb{F}_q[x,y]$ over $\mathbb{F}_q[x]$ for generating our sieving polynomials. Because the ideals we decompose over the factor base are principal elements extending ideals with a known decomposition, their ideal norm equals the field norm of the ideal generators. Therefore, throughout this section, we restrict ourselves to denoting the field norm by $\mathcal{N}$ to use it interchangeably with the ideal norm.

For hyperelliptic curves, the relative degree of the field extension is two. Therefore, we can compute the norm of a function field element by multiplying it with its conjugate. However, for curves of higher degree, we require a different approach, using the *resultant* of two polynomials to perform the norm computation.

Definition 3.2.1. Let R be an integral domain and let $s,t \in R[y]$ be given by the equations

$$s = \sum_{i=0}^{m} s_i y^i \text{ and } t = \sum_{i=0}^{n} t_i y^i.$$

Then the *resultant* of f and g is defined by

$$Res_y(s,t) := \left| \begin{pmatrix} s_m & \dots & s_0 & & \\ & \ddots & \ddots & \ddots & \\ & & s_m & \dots & t_0 \\ t_n & \dots & t_0 & & \\ & \ddots & \ddots & \ddots & \\ & & t_n & \dots & t_0 \end{pmatrix} \right| .$$

The square matrix with $m+n$ rows and columns belonging to the resultant is called the *Sylvester matrix* of s and t.

The resultant can be used to compute the norm for arbitrary degree function field extensions.

Theorem 3.2.2 (Myerson [45])**.** *Let $f \in \mathbb{F}_q[x,y]$ be a monic polynomial in terms of y that defines a smooth algebraic curve $f(x,y) = 0$. Further, let $\alpha \in \mathbb{F}_q[x,y]$.*
Then the norm of $\alpha \in \mathbb{F}_q(x,y)$ over $\mathbb{F}_q(x)$ equals $Res_y(\alpha, f)$.

This gives an algorithm that allows generating sieve polynomials for arbitrary $C_{n,d}$ curves.

Algorithm 3.2.3.

Input: A $C_{n,d}$ curve over $\mathbb{F}_q$ with a factor base $\mathcal{F}$ of convenient divisors. Furthermore, an $\mathcal{F}$-smooth divisor D represented by (a,b) with $a, b, \in \mathbb{F}_q[x]$ and $\deg b < \deg a$.

Output: A polynomial $f_{(a,b)} \in \mathbb{F}_q[x][S]$ representing $\frac{\mathcal{N}(aS+(y-b))}{a}$.

1: Compute the trivariate polynomial $\alpha = aS + (y-b)$ with S, y transcendental elements and $a, b \in \mathbb{F}_q[x]$.
2: Let f be the function field defining polynomial, i.e., $f(x,y) = 0$. Then return the quotient $Res_y(\alpha, f)/a$.

For efficient sieving and a high smoothness probability, it is crucial to bound the degree of the sieve polynomials when inserting $s \in \mathbb{F}_q[x]$ of known degree. In Section 4.2 of [62],

Velichka, Jacobson, and Stein made an analysis of the degree of the resulting insertion that we cited in Theorem 3.1.2. Similarly, we will estimate the degree of principal divisors arising from our sieve array construction.

Theorem 3.2.4. *Consider a $C_{n,d}$ curve in the variables x and y defined by the irreducible equation $f(x,y) = 0$ and with exact field of constants $\mathbb{F}_q$.*

Let $a \in \mathbb{F}_q[x]$ be a polynomial that splits completely in $\mathbb{F}_q[x,y]$ and whose prime factors are distinct in $\mathbb{F}_q[x]$. Further, let $b \in \mathbb{F}_q[x]$, $\deg b < \deg a$ such that $f(x,b) \equiv 0 \pmod{b}$.

Assume that f satisfies $f = \sum_{i=0}^{\deg_y f} f_i y^i$ with $f_i \in \mathbb{F}_q[x]$ and $f_{\deg_y f} = 1$.

Now let $\alpha = as + (y-b)\cdot 1$ with $0 \neq s \in \mathbb{F}_q[x]$. This is an element of the previously discussed ideal. Then the degree of the norm of α satisfies

$$\deg_x \mathcal{N}(\alpha) \leq \max_{0 \leq i \leq \deg_y f} \left\{ \deg_x f_i + i \cdot \deg_x (a \cdot s) \right\}.$$

Proof. First, we sort the coefficients of α with respect to y and get $\alpha = y + (as - b)$. Therefore, the Sylvester matrix has the form

$$\begin{pmatrix} 1 & (as-b) & & \\ & \ddots & \ddots & \\ & & 1 & (as-b) \\ 1 & f_{\deg_y f - 1} & \cdots & f_0 \end{pmatrix}.$$

By some straightforward row operations, we transform the matrix to the form

$$\begin{pmatrix} 1 & 0 & & (as-b)^{\deg_y f} \\ & \ddots & \ddots & \vdots \\ & & 1 & (as-b)^1 \\ 1 & f_{\deg_y f - 1} & \cdots & f_0 \end{pmatrix}.$$

Then we obtain the desired result by transforming the matrix into upper echelon form. The determinant finally equals the entry in the bottom-left corner of the upper echelon form. □

Remark 3.2.5. *The proof of Theorem 3.2.4 also proves that the norm can be computed by using at most* $2\deg_y(f)$ *row operations. Furthermore, we see that it is possible to compute the norm before evaluating S by only adding and multiplying elements of* $\mathbb{F}_q[x]$*. These operations have quadratic complexity in the size of the polynomial representation.*

Recall from Lemma 1.2.47 that it is possible to divide the sieve polynomial by a regardless of the choice of s. By the division, we remove the known prime factors from the sieve array elements that we included for the initialization. With this step done, we can analyze the degree very precisely for the class of $C_{n,d}$ curves. Because the result is a direct formula that includes the desired sieve bound M, we can also optimize the degree of a to obtain low degree norm.

Theorem 3.2.6. *Consider a* $C_{n,d}$ *curve represented by* $f(x,y) = 0$ *with exact field of constants* $\mathbb{F}_q$*. Let M be a sieve bound as described in Algorithm 3.2.3, and let* $(a,b) \in \mathbb{F}_q[x]^2$ *satisfying* $f(x,b) \equiv 0 \pmod{a}$.

Then the sieve array as constructed with Algorithms 1.2.54 and 3.2.3 corresponds to divisors of degree at most $\max_{0\le i\le n}\left\{\frac{(n-i)d}{n} + i\cdot M + (i-1)\deg_x a\right\}$.
This degree can be bounded above by $\left\lceil\frac{n-1}{n}d\right\rceil + M$ *for the optimal choice of* $\deg a$.

Proof. Given the notation of Theorem 3.2.4, we recall that the sieve polynomial associated with (a,b) is the norm of elements divided by the norm of a. As a can be built up from distinct primes, we can reduce the degree results by $\deg a$ when comparing the sieve polynomial output in contrast to the direct norms of the principal divisors $sa + (y-b)$.

Considering the definition of $C_{n,d}$ curves, we get $\deg f_i \le \frac{(n-i)d}{n}$. By the structure of the transformed Sylvester matrix in the proof of Theorem 3.2.4, the first assertion follows directly.

Note that the only term in the set independent of M corresponds to $i = 0$. This term has degree $d - \deg_x a$, because the coefficient f_0 belongs to x^d. We balance this term with the term that is growing most quickly with M and $\deg_x a$, i.e., $i = n$. Then we get an equation

for the optimal degree of a:

$$\begin{aligned} nM + (n-1)\deg_x a &= d - \deg_x a \\ \Rightarrow \deg_x a &= \frac{d}{n} - M. \end{aligned}$$

Inserting this theoretically optimal value of $\deg_x a$ gives

$$\frac{(n-i)\ d}{n} + i \cdot M + (i-1)\deg_x a = \frac{(n-i)\ d}{n} + M + (i-1)\frac{d}{n} = \frac{n-1}{n}d + M.$$

Note that by the definition of a $C_{n,d}$ curve, the ratio $\frac{d}{n}$ is strictly greater than 1, and it is not an integer. Consequently, we adopt the floor of the theoretically optimal value for the degree of a. While this truncation slightly reduces the beneficial contribution of the x_d term, it simultaneously limits the negative impact of the remaining terms, thereby ensuring that the upper bound asserted in the theorem remains valid. □

Remark 3.2.7. *The construction of sieve polynomials with optimal degree depends on the condition* $\frac{d}{n} - M \geq 1$. *Furthermore, the cardinality of the search space must be sufficient to sustain the entire relation collection phase. This requirement becomes restrictive if* $n \approx d$. *In such cases, it may be necessary to relax the optimality constraint and initialize with a polynomial* $a \in \mathbb{F}_q[x]$ *of higher degree. While this induces a degree growth linear in both* n *and* $\deg_x a$, *the complexity with respect to* g *remains controlled if* $\deg_x a$ *is kept minimal.*

Remark 3.2.8. *The given analysis also covers the scenario for imaginary-model hyperelliptic curves. For those hyperelliptic curves, we have* $n = 2$, $d = 2g+1$, *and thus the degree bound for the sieve array elements is given by* $\lceil \frac{1}{2}(2g+1) \rceil + M = g + 1 + M$ *as already cited in Theorem 3.1.2.*

3.2.2. An Efficient Sieve Initialization

The process of computing the roots of the sieve polynomials for each factor base element is referred to as *sieve initialization*. Efficient initialization of each sieve in a multi-polynomial

sieve is critical to the overall algorithm's complexity and runtime. This is particularly important when factor base elements are close in size to the search range, allowing only a few sieve steps per computed root. In these cases, sieve initialization significantly impacts the total complexity, as emphasized by Kleinjung for quadratic sieves [36].

Our improvements for the sieve initialization leverage the fact that we can freely select $f_{(a,b)}$ with the only limitation that a needs to be $\mathcal{F}'$-smooth in $\mathbb{F}_q[x]$ and of the right degree to create optimal degree evaluations as in Theorem 3.2.6. We will use both the structure of the sieve polynomials in general and the freedom to select (a,b) pairs to create sieve polynomials that allow an initialization, i.e., root computation for the factor base elements with norm not dividing a, with minimal effort.

Theorem 3.2.9. *Let the function field be given by the equation $f(x,y) = 0$ over a finite field $\mathbb{F}_q$ and with f being monic in y. Let $P = (p, b')$ with $p, b' \in \mathbb{F}_q[x]$ and b' being a root of $f \pmod p$ denote a prime divisor of the factor base.*
Further, let $f_{(a,b)}$ be a sieve polynomial as constructed in Algorithm 3.2.3 such that $p \nmid a$.
Then a root of $f_{(a,b)} \pmod p$ is given by

$$a^{-1}(b - b') \pmod p.$$

This root can be used for uniquely representing P in the sieving process.

Proof. First of all, we note that reduction of the sieve polynomial modulo p induces a commutative diagram

$$\begin{array}{ccc} \mathcal{O}_P \cap \mathbb{F}_q[x,y] & \xrightarrow{\mathcal{N}} & \mathbb{F}_q[x] \\ \downarrow{\scriptstyle Red.} & & \downarrow{\scriptstyle Red} \\ \mathcal{O}_P/P & \xrightarrow{\mathcal{N}} & \mathbb{F}_q[x]/p\mathbb{F}_q[x] \end{array}.$$

We can apply this diagram to any element of the function field. Recall that we defined $f_{(a,b)} \in \mathbb{F}_q[x][S]$ for given $a, b \in \mathbb{F}_q[x]$ to be the sieve polynomial given by

$$f_{(a,b)} = \frac{\mathcal{N}(aS + (y - b))}{a} = \frac{\mathrm{Res}_y(aS + (y - b), f)}{a}.$$

Instead of computing the root for the full fraction, we can do so for $\mathcal{N}(aS + (y - b))$, because a is a unit $\pmod{p}$. Because both inputs are monic, the resultant vanishes $\pmod{p}$ if and only if the two input polynomials share a root $\pmod{p}$ (see [7, Korollar 9 of Section 4.4]). Let b' be the previously computed root of $f \pmod{p}$, and thus $y - b'$ a linear factor. Then it also is a root of $a \cdot s + (y - b) \pmod{p}$ if and only if $s \equiv a^{-1}(b - b')$, which completes the proof. □

Remark 3.2.10. *We can extend the theorem to any sieve polynomials constructed by elements monic in y. These have the structure of* $as_0 + \sum_{i=1}^{n-1} (y-b)^i s_i$, *where n is the rank of the ideal as* $\mathbb{F}_q[x]$*-module. Then inserting* b' *for y gives a one line homogeneous linear system for* $s_0, \ldots, s_{n-1}$. *Each solution for this system gives a root of the n dimensional sieve polynomial that we can use as a starting point for sieving.*

Remark 3.2.11. *Theorem 3.2.9 allows us to sieve for relations without explicitly performing Algorithm 3.2.3. By precomputing* $a^{-1} \pmod{p}$ *for each prime norm p of the factor base elements, we can compute the roots of* $f_{(a,b)} \pmod{p}$ *efficiently and sieve in* $\mathbb{F}_q[x]$ *without the requirement of an explicit norm computation.*

While the computation of each new $a^{-1} \pmod{p}$ *for all factor base elements not dividing a can be costly, we can use the following optimization to reduce this effort.*

Let $p_1, \ldots, p_l$ *be the distinct prime norms of a subset of divisors taken from the factor base* $\mathcal{F}$. *Then, we can precompute*

$$p_i^{-1} \pmod{p} \text{ for all } p \in \{\mathcal{N}(p) \mid p \in \mathcal{F}\} \setminus \{p_1, \ldots, p_l\}.$$

If we then choose (a, b) *such that a factors only into primes from the list* $p_1, \ldots, p_l$, *we can compute* a^{-1} *for all other elements of the factor base quickly by multiplying the corresponding inverses of the selected prime factors.*

Theorem 3.2.12 (Complexity of the Sieve Initialization)**.** *Let* $\mathcal{F}$ *be a dense factor base containing prime divisors of maximal degree B. Then, by combining Theorem 3.2.9 with*

the tricks described in Remark 3.2.11, we can compute the roots of a sieve polynomial $f_{(a,b)}$ for all $p \in \mathcal{F}'$ with $p \nmid a$ using at most

$$(2 \deg a + 3)\#\mathcal{F}$$

finite field operations in a subfield of $\mathbb{F}_{q^B}$.
This gives a complexity bound of

$$(2 \deg a + 3)\#\mathcal{F}\ B^2\ O\left((\ln q)^2\right)$$

bit operations.

Proof. Assume that the method of Remark 3.2.11 is used. Then the current polynomial a differs from a previously used a' only by one prime factor, while all others will be reused. Let q be this new prime factor which replaces the prime factor q'. Then, for all prime elements $p \in \mathcal{F}'$ such that $p \nmid a$ and $p \nmid a'$ we have $a^{-1} = q^{-1}q'(a')^{-1} \pmod{p}$. Therefore, we can compute a^{-1} in 3 finite field operations in $\mathbb{F}_q[x]/p\mathbb{F}_q[x]$, which is of size at most q^B.

To compute $b \pmod{p}$, we can first compute the unreduced b by using the Chinese remainder theorem and reduce the result $\pmod{p}$. By treating the residue of $x \pmod{p}$ as an element of $\mathbb{F}_q[x]/p\mathbb{F}_q[x]$, the reduction has a cost of $\deg b \leq \deg a - 1$ finite field operations.

The initialization concludes with the subtraction and multiplication steps from Theorem 3.2.9. These final operations incur a cost of $O\left(B^2 (\ln q)^2\right)$ bit operations, provided that standard quadratic arithmetic is employed. This completes the proof. □

We see that the most operations for initializing are spent reducing b for each factor base element. Because we use only split prime polynomials for the factor base, we know that for each involved root, its conjugates are also forming factor base divisors. This property can be leveraged to reduce computation costs by creating multiple sieve polynomials $f_{(a,b)}$ sharing the same a. This method is exactly the generalization of Theorem 3.1.3, which was borrowed from Velichka, Jacobson, and Stein [62].

Theorem 3.2.13. *Consider a function field defined by $f \in \mathbb{F}_q[x,y]$. Let $a = \prod_{i=1}^{k} q_i$ for factor base elements (q_i, t_i) with pairwise distinct prime polynomials q_i. Let b be the corresponding root such that $f(b) = 0 \pmod a$. We assume that b is constructed from the selected t_i using the Chinese remainder theorem. Further, we assume that for a specific j, there is a second root t_j' of the function field defining polynomial. Define*

$$B_j := \left(aq_j^{-1}\right)\left(\left(aq_j^{-1}\right)^{-1} t_j \pmod{q_j}\right), \text{ and}$$
$$B_j' := \left(aq_j^{-1}\right)\left(\left(aq_j^{-1}\right)^{-1} t_j' \pmod{q_j}\right).$$

Let b' be the root of the function field defining polynomial $\pmod a$ that differs from b only because t was replaced by t' in the build-up process.
Then $b' = b + (B_j - B_j')$.

Proof. The composition of any b from the roots of the function field defining polynomial is unique $\pmod a$ by the Chinese remainder theorem. Therefore, we only have to prove that the created b' has all the desired properties. Consider q_i with $i \neq j$. Then $b \equiv t_i \pmod{q_i}$, and since $q_i \mid aq_j^{-1}$, we conclude

$$b' = b + (B_j - B_j') \equiv b + (0 - 0) = b \equiv t_i \pmod{q_i}.$$

Conversely, we know that aq_j^{-1} is invertible $\pmod{q_i}$ and get

$$\begin{aligned} b' = b + (B_j - B_j') &\equiv t_j + \left(\left(aq_j^{-1}\right)\left(\left(aq_j^{-1}\right)^{-1} t_j'\right) - \left(aq_j^{-1}\right)\left(\left(aq_j^{-1}\right)^{-1} t_j'\right)\right) \\ &\equiv t_j + (t_j' - t_j) \equiv t_j' \pmod{q_i}. \end{aligned}$$

Therefore, the constructed element satisfies all the desired properties. □

The theorem allows us to switch between multiple roots of a prime dividing a and thus to re-initialize the sieve quickly if a remains unchanged.

Theorem 3.2.14. *Given the setting of Remark 3.2.11. Assume that we use only factor base elements for building a sieve polynomial that are completely split, i.e., for each q_i there are n roots of the function field defining polynomial $\pmod{q_i}$ with $n = \left[\mathbb{F}_q(x,y) : \mathbb{F}_q(x)\right]$. Further assume that k factor base divisors are required to build a sieve polynomial.*

Then for each set $q_1, \ldots, q_k$ of pairwise disjoint factor base norms that satisfy the above condition, there are at least n^{k-1} independent sieve polynomials sharing the same $a = \prod_{i=1}^{k} q_i$. As a result, we can initialize all these sieves by using

$$\#\mathcal{F} \cdot \left(k^2 + nk\right) \cdot O\left((B \ln q)^2\right) + \#\mathcal{F} \cdot n^{k-1} \cdot (k+1) \cdot O\left(B \ln q\right).$$

bit operations.

Proof. First, we note that we get n^{k-1} independent sieve polynomials by fixing one of the roots of q_1 throughout the entire process. This is to avoid the situation where all conjugates of the divisor (q_1, t_1) are present in the relations generated by the sieve polynomials. Otherwise, we introduce double rows that reduce the matrix rank in the linear algebra step.

Assume that the $q_i^{-1} \pmod{p}$ for all $p \in \mathcal{F}' \setminus \{q_1, \ldots, q_k\}$ are already precomputed, as described in Remark 3.2.11. Then the k elements $\left(aq_i^{-1}\right)^{-1} \pmod{q_i}$ can be computed by $k-1$ modular multiplications and one further modular inversion. Therefore, we compute all possibilities for $\left(aq_i^{-1}\right)^{-1} t \pmod{q_i}$ using $k^2 + nk$ finite field operations in an extension field of $\mathbb{F}_q$ of degree at most B.

Moreover, we can compute $\left(aq_j^{-1}\right) \pmod{p}$ for any j and any $p \in \mathcal{F} \setminus \{q_1, \ldots, q_k\}$ in at most k finite field operations in an extension field of $\mathbb{F}_q$ of degree at most B. Consequently all possible combinations can be built in $(\#\mathcal{F} - k)\, k^2$ operations.

To build all $B_t \pmod{p}$ as described in Theorem 3.2.13 for any $p \in \mathcal{F} \setminus \{q_1, \ldots, q_k\}$, we require an additional $k \cdot n$ operations in the field $\mathbb{F}_q[x]/p\mathbb{F}_q[x]$ and the reduction of

$$\left(\left(aq_i^{-1}\right)^{-1} \pmod{q_i}\right) \pmod{p}.$$

Since the latter elements are of degree at most B, this reduction requires at most $B - \deg p$ operations of cost $O\left((\ln q)^2\right)$ bit operations. For those primes in $\mathcal{F}'$ of lower degree, the

cost for the degree B elements serves as an upper bound. In total, by estimating that all other operations modulo p are of cost $BO\left((\ln q)^2\right)$, we already cover the costs for the reduction, and thus can leave it out for the sake of simplicity.

In total, the pre-computation costs aggregate to less than $\#\mathcal{F}\left(k^2+nk\right)$ finite field operations in an extension field of $\mathbb{F}_q$ of degree at most B. By following the arguments of Theorem 3.2.13 we can initialize each of the n^{k-1} sieves by at most $k+1$ additions for each factor base element.

Therefore, the total initialization cost for all n^{k-1} sieves is bounded by

$$\#\mathcal{F}\left(k^2+nk\right)\cdot O\left((B\ln q)^2\right)+\#\mathcal{F}\cdot n^{k-1}\cdot(k+1)\cdot O\left(B\ln q\right).$$

□

The result can be improved further by reducing the factor of $n^{k-1}(k+1)$ in the right sum of the initialization cost term. This is achieved by ordering the n^{k-1} sieve polynomials in a way that switching between two polynomials becomes cheaper than recombining all roots.

Corollary 3.2.15. *By using a gray code scheme for the order of sieve polynomials, we can even achieve a total complexity of*

$$\#\mathcal{F}\left(k^2+nk\right)\cdot O\left((B\ln q)^2\right)+\#\mathcal{F}\cdot 2n^{k-1}\cdot O\left(B\ln q\right)+\#\mathcal{F}\cdot(k+1)\cdot O\left(B\ln q\right)$$

bit operations.

The idea is to select the next polynomial such that we only replace one of the t_j with exactly one t_j'. Then by Theorem 3.1.3 we require only two addition operations for each factor base element to initialize the sieve. The exact details of how the desired order can be established will be described in appendix A.5.

Besides the mentioned complexity improvements, using self-initialization has further advantages. It is immediate that we can save the inversion operation for a by computing inverses of the self initialization polynomials beforehand. Since only a small set of these

polynomials is necessary for setting up all sieve polynomials, this will further reduce the practical effort for initializing the sieve. Unfortunately, there is no additional gain in the asymptotic complexity, because the multiplication of the small degree polynomials and the omitted modular inversion belong to the same asymptotic complexity class.

In the next section we will investigate how stepping through the sieve array can be done efficiently. Together with the last two results, this will allow a precise statement regarding the average costs for constructing and decomposing a divisor over the factor base by using sieving.

3.2.3. A linear Time Sieve Iteration Algorithm

A central challenge in polynomial sieving is efficiently iterating through the sieve array. Specifically, computing the index function ν for all $k \cdot p + b$ with $k \in \mathbb{F}_q[x]$, $\deg(k) + \deg(p) < M$, and ν as defined in Algorithm 1.2.54, is a performance-critical task. As noted in [21] and [62], the main difficulty arises from the non-linearity of all common choices for ν. Consequently, knowing $\nu(k_i \cdot p + b)$ provides little advantage when calculating $\nu(k_{i+1} \cdot p + b)$ for a given ordering k_i of all factors required to populate the sieve array. In this section, we develop a scheme to compute the target values efficiently, minimizing the necessary arithmetic operations.

This section is organized into two parts. First, we present an approach for iterating through the sieve array with nearly constant time per step, specifically how to compute $k_{i+1} \cdot p + b$ from previous steps independently of the degree of k_{i+1}. The second part introduces ideas to address the non-linearity issues of ν.

For even q, both algorithms can be combined into a quasi-linear stepping routine that we implemented for our experiments. Details on combining these strategies in characteristic two are discussed later in Section 4.1.1.2.

For the iteration, we use a straightforward ordering of elements in $\mathbb{F}_q[x]$, defined as follows:

Definition 3.2.16. Let $\omega \in \mathbb{F}_q$ be a generating element of the field extension $\mathbb{F}_p \subset$

$\mathbb{F}_q = \mathbb{F}_p(\omega)$ and let $q = p^l$ for a prime $p \in \mathbb{N}$. Consider the map $\mu : [0, \ldots, q-1] \to \mathbb{F}_q$,

$$\sum_{i=0}^{l-1} c_i p^i \mapsto \sum_{i=0}^{l-1} c_i \, \omega^i.$$

Then, we can map every $n \in \mathbb{N}$ into $\mathbb{F}_q[x]$ using its q-adic representation:

$$n = \sum_{i=0}^{\lceil log_q n \rceil - 1} a_i q^i \mapsto \sum_{i=0}^{\lceil log_q n \rceil - 1} \mu(a_i) t^i \in \mathbb{F}_q[t].$$

We will denote this map by Φ. Furthermore, for the rest of this section, we will define k_i to be the image $\Phi(i) \in \mathbb{F}_q[x]$ for $i \in \mathbb{N}$.

Remark 3.2.17. *A natural choice for the function ν in Algorithm 1.2.54 is the inverse function Φ^{-1}. Let $f \in \mathbb{F}_q[x]$. Then we can treat f as a bivariate polynomial with variables ω and x over $\mathbb{F}_p$. If we further identify $a \in \mathbb{F}_p$ with the smallest positive integer reducing to a then ν equals the evaluation of ω to p and x to q.*

In the following lemma, we discuss several immediate yet essential properties of the natural ordering and our construction of ν.

Lemma 3.2.18. *Let $i \in \mathbb{N}$ with $i < q^l$ and let $p, b' \in \mathbb{F}_q[x]$ with $\deg b' < \deg p$. Furthermore, let $k_i = \Phi(i)$ and k_{ij} be the jth coefficient of k_i. Then*

a) $\nu(k_i \cdot p + b') < q^{l+\deg p}$ holds.

b) $k_{ij} = 0$ for all $j < s$, where s is defined as the least positive integer such that $k_{ij} = k_{i-1j}$ for all $j > s$.

c) $\exists\, t < i$ such that $k_t = k_i - k_{is}x^s$ with s defined as in statement b).

Proof.

a) Since $\nu(k_i) = i < q^l$ we conclude that the degree of k_i is at most $l-1$. Therefore, $\deg(k_i \cdot p + b') \leq l + \deg(p) - 1$ and due to the q-adic evaluation of the polynomials $\nu(k_i \cdot p + b') < q^{l+\deg(p)}$.

b) We know that $k_{i-1j} = k_{ij}$ for all $j > s$, and that s is minimal with this property. We conclude $k_{i-1s} + 1 = k_{is}$ due to the construction of the ordering of k_i. This gives $k_{i-1j} = q - 1$ for all $j < s$ so the addition of 1 is overflowing all coefficients up to the sth and thus gives $k_{ij} = 0$ for all $j < s$.

c) Due to the construction of s and statement b), k_{is} is the least significant term contributing a value to the evaluation of k_i. More precisely, we see that

$$\nu(k_i - k_{is}x^s) = \lfloor \frac{\nu(k_i)}{q^s} \rfloor q^s = \nu(k_t), \text{ for } t := \lfloor \frac{\nu(k_i)}{q^s} \rfloor q^s.$$

This completes the proof.

□

Statement a) of Lemma 3.2.18 ensures that the memory addresses of polynomials generated during the iteration is bounded. In fact, the bound is sharp in terms of powers of q, i.e., when the set of all $pk_i + b'$ contains all polynomials of this form up to degree M, the sieve array has an overall bound of q^{M+1}.

Statement b) is merely a formalization that when counting up, there is a highest coefficient that changes. Letting this index be s, it follows that all coefficients with an index strictly less than s are equal to 0.

We can also restate the proof of statement b) in a constructive way. By some direct calculations we see that t is given by $\lfloor \frac{i}{q^{s+1}} \rfloor q^{s+1}$. Note that $a_{uj} = a_{ij}$ for all $i \leq u < i + q^s$ and $j \geq s$. So, by storing $k_i \cdot p + b$ at a memory address determined by s, we can compute succeeding iterations by adding only a shifted scalar multiple of p. This leads to the following algorithm.

Algorithm 3.2.19.

Input: Polynomials p, b' with $\deg b < \deg p$ over $\mathbb{F}_q$ and a sieve-radius $M \geq \deg p - 1$. Let p be given as an array $P[]$ of integer representations of its coefficients

Output: An array $R[]$ containing all polynomials $k \cdot p + b'$ of degree at most M for $k \in \mathbb{F}_q[x]$.

Set $l \leftarrow q^{M-\deg p+1}$.
Set $S[j] = b'$ for all $0 \leq j \leq M - \deg p + 1$.
Set $i = 0$ and $K[j] = 0$ for all $0 \leq j \leq M - \deg p$.
$R[0] \leftarrow b'$
5: **while** $(i < l)$ **do**
$i \leftarrow i + 1$
$s \leftarrow 0$
while $K[s] = q - 1$ **do**
$K[s] \leftarrow 0$
10: $s \leftarrow s + 1$
end while
$K[s] \leftarrow K[s] + 1$
$R[i] \leftarrow S[s] + \mu(K[s]) \cdot p \cdot x^s$
while $s > 1$ **do**
15: $s \leftarrow s - 1$
$S[s] \leftarrow R[i]$
end while
end while

The concept behind Algorithm 3.2.19 is as follows.

We construct two arrays, K and S, where K tracks the polynomial coefficients of k when computing $k_i \cdot p + b'$, and S stores intermediate computed polynomials to optimize later calculations.

The invariant of our approach is that $S[j]$ always holds the most recent polynomial $k \cdot p + b'$ where the coefficients at position j and below are zero. We begin with $k_0 = 0$. Therefore, the array S is initialized with the polynomial b' in each entry.

In the loop from line 8 to 11, we identify the most significant coefficient s of k_i that changed compared to the previous step k_{i-1}. We know that all coefficients of k_i of degree less than s are zero, and all coefficients with degree higher than s are equal to a k_j computed earlier during the process. Thus, the assignment $R[i] \leftarrow S[s] + \mu(K[s]) \cdot p$ in line 13 correctly computes the current value of $R[i]$.

Finally, in lines 14 to 17, we restore the invariant for S by copying the recently computed polynomial into the entries of S with indices lower than s.

Similar to the construction of $R[i]$, we also can ensure only about $\deg p$ operations are required to evaluate the resulting polynomial to a memory address: We know that for those coefficients of $k \cdot p + b'$ with index less than s, only the coefficients of b' are set. Therefore, it is sufficient to create a list L of precomputed partial evaluations of b' to cover the low-degree terms of $k \cdot p + b'$.

On the other hand, we know that the coefficients associated with the degree above $s + \deg p$ are identical to those of a previously calculated polynomial $k_t \cdot p + b'$. The idea behind the following algorithm is to maintain a list H that contains partial evaluations using a Horner scheme, thereby working from the most to the least significant coefficients. When computing $\nu(k_i \cdot p + b')$, we look up the computation of $\nu(k_t \cdot p)$ that covered the unchanged, most significant coefficients in order to recompute only the central $\deg p + 1$ ones. Note that the creation of this second list does not need to involve any coefficients of b', because $\deg b' \leq \deg p - 1$, meaning these coefficients are never among the unchanged, most significant coefficients.

Algorithm 3.2.20. Let $b' = \sum_{i=0}^{\deg(p)-1} b_i x^i$ and $L \in \mathbb{N}^M$ with $L_0 = 0$ and

$$L_k := \nu\left(\sum_{i=0}^{k-1} b_i\, x^i\right) \text{ for } 0 < k < M.$$

Furthermore, let $k_i p + b'$ and s be as we described in Lemma 3.2.18 and create a second list $H \in \mathbb{N}^{M+2}$ initialized with zeros.

Now, for each step i, let $k_i p + b' = \sum_{j=0}^{M} a_{ij}' x^j$, perform the following two actions:

1. For k ranging from $k = s + \deg(p)$ down to s, update H by the Horner Scheme

$$H_k := H_{k+1} \cdot q + \nu\left(a_{ik}\right).$$

2. Return the memory address

$$\nu\left(k_i p + b'\right) = H_s{}' q^s + L_s.$$

The algorithm is correct because, by Lemma 3.2.18, we know $a_{i,j} = a_{(i-1),j}$ for all $j > s + \deg p$, so we can reuse the partially top-to-bottom evaluated $\nu\left(k_{i-1} p + b'\right)$ to compute $\nu\left(k_i p + b'\right)$ by updating only the $\deg p + 1$ changed coefficients plus adding the missing low degree partial computation of $\nu(b')$.

Combining the two algorithms and summing their complexity over all potential prime polynomials and roots involved gives us a tight estimate of the overall sieving complexity.

Theorem 3.2.21. *Let C be a $C_{n,d}$ curve with exact field of constants $\mathbb{F}_q$. Let $\mathcal{F}$ be a factor base containing all split divisors of C of degree less than or equal to B. Further, assume the sieve is already initialized with the methods described in Section 3.2.2 and let $M \geq B - 1$.*

Then, testing q^{M+1} divisors in the search space to be $\mathcal{F}$-smooth by the combination of Algorithm 3.2.19 and Algorithm 3.2.20 takes on average less than

$$2q^{M+1}\left(B + 2\ln B + 2\right)$$

integer operations with integers ranging up to q^{M+1} and

$$q^{M+1}\left(B + 3\ln B + 3\right)$$

finite field operations over $\mathbb{F}_q$.

Proof. We sum over the different types of operations and progress from the inside of the inner loop to the outer loops. Let $\deg p = l$.

We start with lines 12 and 13 of Algorithm 3.2.19. Updating the s'th coefficient of k takes only a single addition in $\mathbb{F}_q$. Subsequently, computing the coefficient array $R[]$ of $k \cdot p + b'$ takes exactly l multiplications and additions in $\mathbb{F}_q$, because p is monic and the operation consists of multiplying p by $\nu(K[s]) \in \mathbb{F}_q$ and adding it to a stored polynomial at the right powers of x.

Evaluating $R[i]$ to a memory address with Step 1 of Algorithm 3.2.20 now consists of $l+1$ additions and multiplications in $\mathbb{N}$ by using a Horner scheme, where the starting point is the incomplete Horner evaluation H_{s+l} of the coefficients $k_{i,j}$ for $j > s+l$. Finally, the result is multiplied by q^s and added to B_s. The size of all integer numbers involved is less than q^M. Additionally, all q^s can be cheaply pre-computed, so the powering of q can be saved at this point.

While completing the Horner scheme for $k \cdot p$ in Step 1 of Algorithm 3.2.20, restoring H_{s+l-1} down to H_s can be done on the fly, adding $l+1$ storing operations to the complexity.

Additional memory operations as well as simple adding operations arise from the loop between Lines 8 and 11 of Algorithm 3.2.19. Because the loop is determining s, it takes a different number of operations with each new i. The average value $\bar{s}$ of s is bound above by

$$\bar{s} = \sum_{i=0}^{M-i} i\frac{q-1}{q^{i+1}} \leq \sum_{i=1}^{M-i} \frac{i}{q^i} < 2.$$

Therefore, we add fewer than two additional memory accesses, additions, and comparisons in $\mathbb{F}_q$ to our overall complexity.

For the overall runtime, we now need to multiply by the total number of iterations done per polynomial. For $\deg p = l$, the maximum value of i is q^{M-l+1}. Naturally, prime polynomials of smaller degree require significantly more sieving steps compared to larger-degree primes in $\mathcal{F}'$. For a total complexity, we will sum over all split prime divisors of a fixed degree. By [55, Corollary 5.2.10], one can estimate the number of prime divisors of degree l to be approximately $\frac{1}{l}q^l$ by subtracting an upper bound of all ramified and inert primes from the range in the corollary. Note that the absolute number of different prime polynomials p will be significantly fewer than the estimate, but primes having multiple roots b' of the function field defining polynomial will compensate for the difference.

Therefore, the total number of finite field operations required to test all q^{M+1} divisors in

the search range is bounded above by

$$\sum_{l=1}^{B} \frac{q^l}{l} q^{(M-l+1)} \ (l+3) = \sum_{l=1}^{B} q^{M+1} \cdot \frac{l+3}{l}$$
$$\leq q^{M+1} \ (B + 3\ln B + 3)$$

additions and multiplications carried out over $\mathbb{F}_q$.

By a similar calculation, we get the additional cost for the integer and storing operations claimed in the theorem. □

The most important consequence of Theorem 3.2.21 is that the average time to test a single divisor for smoothness depends only on the degree of the factor base elements and not on the sieve radius M itself. This gives a certain flexibility for the choice of M, since it allows us to adjust M to the machine-given optimum without negative algorithmic consequences.

Remark 3.2.22. *At the time of writing, the recommended security level for asymmetric cryptosystems in commercial and government security systems is at least 128 bits [20], [46]. This is reflected in the choice of recommended elliptic curves, e.g., NIST P-256, Curve25519 and brainpoolP256r1 with 256-bit key sizes, assuming only generic solution algorithms are available.*

Translating the key size that equals the size of the Jacobian of 256 bits to a hyperelliptic curve or, more generally, any $C_{n,d}$ curve with genus at least 4, all integer operations mentioned in Theorem 3.2.21 are then within one single 64-bit machine word, which is common in today's computer architecture. Furthermore, for very small field sizes, the finite field operations can be replaced by lookup table accesses. As a result, for small q, the sieving step is computationally cheap compared to the number of divisors tested.

A potential theoretical bottleneck of this method arises when the cost of scanning the sieve array outweighs the sieving step itself. Although checking an individual entry is computationally cheap, the cumulative effort can become significant, particularly for low-degree curves. In the extreme case where the factor base bound B is 1 and $M = 0$, the sieve process degenerates into simply comparing the degree-zero roots of the sieve polynomial.

Here, checking q candidates for smoothness without the benefit of iterative sieving is unnecessarily expensive. To address this, we discuss two strategies for implementing the sieve efficiently under these specific conditions.

Lemma 3.2.23. *Let $\mathcal{F}$ be a factor base containing about q^r prime divisors with $0 < r \leq 1$. Further, assume that for a specific sieve polynomial the q^r roots modulo the factor base elements have already been calculated. Then, we can perform the smoothness testing of q divisor representations using either*

1. *$O(q^r)$ arithmetic and storage operations and $O(q)$ memory cells or*
2. *$O(q^r r \log_2 q)$ arithmetic and storing operations and $O(q^r)$ memory cells.*

Both methods are also suitable for obtaining relations that include a single-large-prime directly. If more large-prime relations are required, we can gain a small candidate list with both approaches.

Proof. The first method follows the direct approach of setting up a sieve array using q memory cells. For each of the about q^r roots we have to increase one entry of the sieve array. For any sieve array entry exceeding the threshold limits, the sieve array entry is stored into a corresponding list. Note that, unlike before, we do not search the entire sieve array for values exceeding the threshold limit. Instead, on each of the q^r additions, we check whether the newly increased value is high enough. Therefore, we can bound the number of arithmetic and storing operations by $O(q^r)$.

The second approach uses the Quicksort algorithm and thus takes $O(q^r r \log_2 q)$ operations to sort the list of all roots. It then takes only an additional $O(q^r)$ operations to search the list for all entries occurring sufficiently often. This approach uses slightly more operations while being more memory-efficient. □

The surprising consequence of this approach is that the average cost for testing one divisor representation for smoothness is $O(q^{1-r})$, which decreases for low $r < 1$ and $q \to \infty$. We will discuss this benefit and its consequence to parameter selection later in Section 3.3.2.

For the general case, we state an aggregated theorem on the complexity of testing a fixed number of divisors for smoothness in Theorem 3.2.24 of Section 3.2.4.

3.2.4. Discussion of Results

When evaluating the total cost of a DLP computation using an index calculus approach, the cost of testing a divisor for smoothness scales quadratically with the size of the factor base, since the number of required tests and the factor base size itself are balanced. This makes it crucial to keep the cost of smoothness testing as low as possible. We will first examine the scenario where the degree of the curve is relatively high compared to the size of the field of constants, i.e., where the calculations for the optimal factor base size follow the arguments in Section 2.4. Based on the results of the previous two subsections, we can establish the following asymptotic cost for testing a divisor for smoothness over an optimally sized factor base.

Theorem 3.2.24. *Given a $C_{n,d}$ curve of genus $g = \frac{(n-1)(d-1)}{2}$ over $\mathbb{F}_q$ and a dense factor base $\mathcal{F}$ as defined in 2.4.1 of optimal size with respect to the degree of sieving elements $\eta = \lceil \frac{n-1}{n} d \rceil + M$ for a constant M.*
Then, the expected duration of the test for $\mathcal{F}$-smoothness of each constructed divisor by using the proposed sieving methods is in the complexity class

$$O\left(g\,(\ln q)^2\right)$$

bit operations.

Proof. By the proof of Theorem 2.4.7, the optimal factor base size is q^m with $m \in O\left(\sqrt{\frac{n-1}{n} d}\right)$. This also implies $B \in O\left(\sqrt{g}\right)$. We will use this estimate to compute the complexity of the different algorithm phases.

We start by calculating the roots of the sieve polynomial over the elements of the factor base. We can distinguish two cases. If the sieve polynomial is generated by a single factor base element, then the possibility of using the self-initialization methods from Section 3.2.2 is precluded.

In this case, we can use the formula from Theorem 3.2.9. Because, in this context, a is the norm of a factor base element, it is already reduced for all other factor base elements

or can be reduced with a few operations. This reduction, as well as the further operations for calculating $a^{-1}(b-b')$, is then in $O(B^2(\ln q)^2)$ for each factor base element.

Otherwise, for $\deg a > B$, we get $k \geq 2$ in the context of Theorem 3.2.14 and thus $k^2 + nk \in O\left(n^{k-1}\right)$. Because we can generate n^{k-1} independent different sieve polynomials from our methods, we again get an estimated cost of $\#\mathcal{F}\ B^2\ O\left((\ln q)^2\right)$ per sieve polynomial.

By the result of Theorem 3.2.21, the sieving process requires $O\left(Bq^{M+1}\right)$ finite field operations in $\mathbb{F}_q$.

Therefore, the average complexity for testing one divisor of the q^{M+1} elements sized search space is bounded by

$$O\left((\ln q)^2 \cdot \max\left\{\frac{B^2\#\mathcal{F}}{q^{M+1}}, \frac{\#\mathcal{F}}{q^{M+1}}, \frac{B}{q}\right\}\right).$$

Since $\#\mathcal{F} < q^B \leq q^{M+1}$, this reduces to $B^2\ O\left((\ln q)^2\right)$ per tested divisor. This proves the theorem. □

Remark 3.2.25. *If we restrict our attention to a family of hyperelliptic curves with increasing genus instead of arbitrary $C_{n,d}$ curves, we get an even sharper result, because the optimal choice for* $\deg a$ *is $g - M$ and so $c \in O\left(\sqrt{g}\right)$. Because $n = 2$ is fixed and the optimal B is in $O\left(\sqrt{g}\right)$, the left summand of the complexity estimate of Theorem 3.2.14 is $O\left(g^2\right)$, while the number of initialized sieves is $O\left(2^g\right)$ and, therefore, the right summand of the formula will become dominant.*

Therefore, the power of B in the equation proving Theorem 3.2.24 reduces by one, and we get an overall complexity estimate of $O(\sqrt{g}\ (\ln q)^2)$ per tested divisor.

For non-hyperelliptic $C_{n,d}$ curves, the sieve method offers a significant advantage: for computing the general relations, the group law does not need to be performed, allowing us to rely solely on polynomial arithmetic. As a result, Theorem 3.2.24 not only demonstrates a computational improvement by a factor of g, but also eliminates the n^8 term associated with the complexity of the curve's group law, as presented in Theorem 1.2.49.

It is worth noting that for certain curves, such as superelliptic curves, the impact of n on the complexity of group arithmetic is less pronounced. Harasawa and Suzuki [26]

have shown that for these cases, the influence of n can be reduced to n^4. However, it remains an open question whether a broader subfamily of non-hyperelliptic $C_{n,d}$ curves exists where relation generation by composing randomized divisors with the group law and trial division could outperform the sieve method, especially considering the relatively small constants involved in the sieve method.

In addition to the low computational cost of the smoothness test for the generated sieve array elements, the sieving method also has the advantage of generating elements of relatively small degree if the sieve is initialized optimally. Combining this theorem with the statements about smoothness probabilities from Section 2.4 yields the following theorem.

Theorem 3.2.26. *Given a $C_{n,d}$ curve of genus g over $\mathbb{F}_q$, such that in the sense of Theorem 2.4.7 we have*

$$\max\{0.5m, 1\} \geq \log_q 2 + 4g + 4d$$

for a solution of the system of equations

$$\eta = \left\lceil \frac{n-1}{n} d \right\rceil + \lceil m \rceil \text{ and } m = \sqrt{\frac{\eta \ln \ln q^\eta}{2 \ln q}}.$$

Furthermore, let $\alpha \in [0,1]$ such that $\eta = g^\alpha \cdot \left(\frac{\ln \ln q^g}{\ln q}\right)^{1-\alpha}$.

Then there is a factor base $\mathcal{F}$ of size q^m over the curve such that we can generate $\#\mathcal{F}$ relations over the curve in expected complexity of

$$O\left(g\,(\ln q)^2\right) L_{q^g}\left[\frac{1}{2}\alpha, \sqrt{2}\right].$$

Conversely, for every $\alpha \in \left[\frac{1}{2}, 1\right]$ there are infinitely many $C_{n,d}$ curves with the given value of α.

Proof. The choice of η in the theorem is compatible with Theorem 3.2.6. Therefore, we know that for $C_{n,d}$ curves of this choice we can generate divisors for setting up sieve arrays with sieve bound $M = B - 1 = \lceil m \rceil - 1$. Note that the definition of m and η is inherently recursive. Nevertheless, because m only grows with the square root of η and η grows with

m, we see that there is a solution for the given equation. In particular since η is required to be an integer, not every slight adjustment to m will have an effect on it.

For $\max\{0.5m, 1\} \geq \log_q 2 + 4g + 4d$, we are in the situation of Theorem 2.4.7 that ensures the existence of the factor base and which states that the value chosen for m is optimal with respect to η. In this situation, the theorem gives that the duration for finding sufficiently many relations for the factor base is $L_{q^\eta}\left[\frac{1}{2}, \sqrt{2} + o(1)\right]$.

The construction of α is equivalent to $q^\eta = L_{q^g}[\alpha, 1]$. Therefore, applying Lemma A.1.4 gives the claimed complexity result.

As for the infiniteness of the family of curves, we must ensure that there are enough choices for g and q that satisfy the conditions. For a fixed α and $2\eta > d \geq \eta$ as well as $d^2 > 2g$, we get the condition

$$\left(g^\alpha \cdot \left(\frac{\ln \ln q^g}{\ln q}\right)^{1-\alpha}\right)^2 > \frac{g}{2}$$

This condition is trivially satisfied for $\alpha \geq 0.5$ and large enough g. As a result, there are no additional limitations on the curves with respect to q, and g, and the conditions of the theorem are satisfied once g is large enough for satisfying the $\max\{0.5m, 1\} \geq \log_q 2 + 4g + 4d$ condition. This inequality holds for infinitely many pairs of q and g.

Conversely, for $\alpha < 0.5$, we require that $\ln \ln q^g > \ln q$. This is satisfied for only finitely many q when fixing the genus. Furthermore, $\log_g \frac{\ln \ln q^g}{\ln q}$ is greater than zero only for finitely many pairs of q and g. For a fixed $\alpha < 0.5$, we conclude that the number of curves satisfying all conditions of the theorem must be finite. □

It is important to note that this theorem only considers the generation of a general set of relations and not the complete resolution of a DLP, because for this, the group generator and the element from which the DLP is to be generated must also be factored over the factor base. For very small factor bases, this can be more expensive than generating the remaining relations. We will address this topic later in Section 3.4.2.

Before that, however, we examine curves for which the size of the factor base is exponential in relation to the size of the Jacobian.

3.3. A Large Prime Graph Scheme for Low Degree Curves

In the following, we propose an algorithm that can be used as an alternative to Algorithm 3.1.10 to collect relations over the factor base more quickly by allowing partial relations with up to two factors outside the factor base. This class of algorithms is particularly suitable when the size of the factor base is exponential in relation to q^g, i.e., a further increase would offer significant disadvantages for the overall performance.

After presenting this new approach in the next subsection, we will also balance the algorithm parameters and discuss the impact of sieving as smooth divisor search algorithm when dealing with small degree curves. Throughout this chapter, we assume the notation and definitions of Section 3.1.4 to be given.

3.3.1. Construction of a Large Prime Graph

The algorithm we propose is a variant of Algorithm 3.1.10 with the following modification. Instead of using only one single-large-prime relation, we search until we find a parametrized number of single-large-prime relations while also storing the double-large-prime relations found during this process. Afterwards, we build a kind of large-prime graph immediately, without searching for further relations, using the found single-large-prime relations as seeds to add the stored double-large-prime relations once connected to the graph.

Algorithm 3.3.1.

Input: A factor base $\mathcal{F}$ such that $\mathcal{F}$ contains $q^r < q$ of the prime divisors of degree one. Furthermore, a parameter $c \in \mathbb{R}$ to be optimized later.

Output: A list containing $\#\mathcal{F}$ relations over $\mathcal{F}$

1: Search for relations and store all occurring double-large-prime relations until $c \cdot q^r$ pairwise distinct **single-large-prime** relations are found.

2: Consider a graph with the nodes identified by a subset of the allowed large-prime factors and edges given by double-large-prime relations. We initialize the graph with all large-prime factors appearing in the single-large-prime relations found. For each node store a weight that equals the number of factor base elements for

representing the large-prime-factor over the factor base. Similarly, for each node, store a boolean that indicates if a node was *touched* initialized to *false*.

3: Initialize k with the number of found full relations and those relations that can be directly created by combining single-large-prime relations found during Step 1.

4: **while** $k < q^r$ **do**

5: From the set of untouched nodes select one node N among those with lowest weight and set N as touched.

6: Select all double-large-prime relations that have N as one of the two large-prime factors. Let N' be the second large-prime factor. If N' is not a connected node of the large-prime graph, add N' to the large-prime graph with the corresponding node of weight equal to the weight of N plus the number of factor base elements in the corresponding double-large-prime relation connecting N and N'.

7: If, instead, N' is a graph element in the previous step, we have found two representations of N' over the factor base. Combine these representations to a full relation and increase k.

8: **end while**

The relations generated in Algorithm 3.3.1 are full relations by construction. The only thing we need to prove is that the loop in lines 4 to 8 terminates. To prove this assertion, we need only assume that the large primes within the discovered double-large-prime relations are uniformly distributed across the set of allowed large-prime factors. By this assumption we can find an upper bound to the weight of the full relations generated by the algorithm.

Theorem 3.3.2 (Graph Depth and Relation Weight in 3.3.1)**.** *Given the setting of Algorithm 3.3.1 and assume that the double-large-prime relations found in step 1 have drawn their large-prime factors from a uniform distribution over the set of all potential large-prime factors. Further let η be the number of prime factors in each full relation found during step 1.*

Then, on average, in each iteration there are about $\frac{c\cdot(\eta-1)}{2} - 1$ new elements added to the large-prime graph.

Furthermore, the relations gained in step 7 have an average weight bound by

$$2(\eta-1)+(\eta-2)\cdot\frac{(1-r)\ln q}{\ln c^{\frac{\eta-1}{2}}}.$$

Proof. Let $q^{r'}$ denote the cardinality of the set of potential large-prime factors. Then, the expected number of tested divisors to find one large-prime relation is

$$(\eta-1)!q^{(1-r)(\eta-1)}1!q^{(1-r')}$$

by following the arguments of Theorem 2.3.1. By the same arguments, the average number of tested divisors to find a double-large-prime relation is approximately

$$(\eta-2)!q^{(1-r)(\eta-2)}2!q^{(1-r')2}.$$

Therefore, upon finding cq^r single-large-prime relations, we expect that we have also found

$$cq^r\frac{(\eta-1)}{2}\frac{q^{1-r}}{q^{1-r'}}=cq^r\frac{(\eta-1)}{2}\frac{q^{r'}}{q^r}=c\frac{(\eta-1)}{2}q^{r'}$$

double-large-prime relations during the process.

Therefore, for each large-prime factor in the set of all potential large-prime factors, there are on average $\frac{c(\eta-1)}{2}$ double-large-prime relations connected to the corresponding large-prime factor. In the first iterations — where the selected large-prime factors correspond to single-large-prime relations — all these double-large-prime relations can be used to extend the graph. In later iterations, it will be one fewer, because we have to subtract the incoming relation to the node in the graph. This proves the first assertion.

For the second assertion, we first note that we expect to require $q^{\frac{r+r'}{2}}$ elements in the large-prime graph until we find q^r relations by the arguments of Theorem A.2.2. Hereby we ignore the full relations as well as the relations gained by combining single-large-prime relations in the initial phase of the algorithm. We build the large-prime graph in a way that lower weight elements are included first. Therefore, we can expect that there are at least $c^2q^{r\frac{\eta-1}{2}}$ nodes in the graph that are connected to a single-large-prime relation with at most

one double-large-prime relation edge in between. In the same way, there are $c^{n-1}q^r\left(\frac{(\eta-1)}{2}\right)^n$ nodes in the graph that are connected to one of the initial graph nodes with at most n double-large-prime relations.

We can conclude that we require a graph depth of about

$$\log_{c^{\frac{\eta-1}{2}}}\frac{q^{\frac{r+r'}{2}}}{cq^r}=\frac{\ln\frac{q^{\frac{r'-r}{2}}}{c}}{\ln c^{\frac{\eta-1}{2}}}=\frac{\frac{r'-r}{2}\ln q-\ln(c)}{\ln c^{\frac{\eta-1}{2}}}<\frac{\frac{r'-r}{2}\ln q}{\ln c^{\frac{\eta-1}{2}}}$$

to construct sufficiently many graph elements that combine into full relations.

Therefore, the final average weight of graph elements is bounded by

$$(\eta-1)+(\eta-2)\cdot\frac{\frac{r'-r}{2}\ln q}{\ln c^{\frac{\eta-1}{2}}}$$

and thus the algorithm produces relations of weight bound by

$$2(\eta-1)+(\eta-2)\cdot\frac{(r'-r)\ln q}{\ln c^{\frac{\eta-1}{2}}}-2<2(\eta-1)+(\eta-2)\cdot\frac{(1-r)\ln q}{\ln c^{\frac{\eta-1}{2}}}.$$

□

The additional routines required to build the new large-prime graph do not significantly increase the algorithm complexity over those of the relation search.

Theorem 3.3.3. *Let C be the expected costs of creating a randomized divisor and testing it for smoothness over the factor base. Then Algorithm 3.3.1 has a complexity of*

$$C(\eta-1)!cq^rq^{(\eta-1)(1-r)}+q^{\frac{1-r}{2}}O\left(\log_2 q\right).$$

Proof. The first phase of Algorithm 3.3.1 is dominant with regard to the run time. Following the arguments of Thériault as described in Theorem 3.1.9, the probability for finding a single-large-prime relation is given by

$$\frac{1}{(\eta-1)!}q^{(\eta-1)(r-1)}.$$

As a result, we can assume that we expect to test

$$(\eta - 1)!cq^r q^{(\eta-1)(1-r)}$$

randomly picked divisors until the first phase of the algorithm finishes. This gives the first summand of the estimated complexity.

By following Theorem A.2.2 we expect that our tree built in the loop from lines 4 to 8 has a total size of $q^{\frac{1-r}{2}}$ elements. The computational cost of building the tree is negligible, because the double-large-prime relations involved can be organized into hashtables indexed by the large-prime factors involved. In order to do so, each double-large-prime relation will be stored twice in this list, and the lists can be indexed by a number in $\mathbb{F}_q$. Therefore, the lookup cost for these lists, and thus the cost of extending a newly found leaf, is $O\left(\log_2 q\right)$.

Since the remaining operations are only additions in $\mathbb{F}_q$ for combining the partial relations to full ones, this completes the proof. □

The efficiency of the double-large-prime algorithm depends on the availability of sufficiently many large primes. An edge case for Algorithm 3.3.1 occurs as $r \to 1$, i.e., when the η is large enough that the optimal value of r is close to one. Then it may happen that the search for q^r single-large-prime relations is nearly as, or even more, complex than the search for full relations over the factor base. We modify the algorithm to better fit curves for which an $O(q)$ sized factor base is optimal.

Algorithm 3.3.4. Instead of forcing both regular and large prime factors to be of degree one, we allow the use of all primes of degree one as factor base elements and the single-large-prime relations to have one large-prime factor of degree 2 and the double-large-prime factors to have two such large-prime factors.

Then the complexity for computing enough double-large-prime relations to ensure the graph is fully connected is given by $(g-4)!q^2$, which may be below $g!q^{r+(1-r)(g-1)}$ for any $r \leq 1$ when $g! > q$.

Besides the extension towards larger-degree factor base elements, we also can extend the algorithm to the use of large-prime relations with more than two large-prime factors.

Theorem 3.3.5. *Given the settings of Algorithm 3.3.1, fix an integer $k < \eta$. Assume that we create cq^r single-large-prime relations in step one of the algorithm. During this process, we can expect to find approximately*

$$c\frac{(\eta-1)!}{k!(\eta-k)!}q^{k(1-r)}\kappa^{k-1}$$

relations that incorporate k prime divisors of degree one beside those of $\mathcal{F}$.

Therefore, we can expect that for each single-large-prime relation, there are at least

$$\frac{(\eta-1)!}{k!(\eta-k)!}q^{(k-1)(1-r)}\kappa^{k-1}$$

large-prime relations with k large primes that share the single-large-prime factor of the relation. These large-prime relations can then be reduced to relations with $k-1$ large-prime factors. Repetition and the assumption that the large-prime factors are equally distributed can be used to generate more double-large-prime relations for Algorithm 3.3.1 from many large-prime relations.

Theorem 3.3.5 allows us to generate a significant number of double-large-prime relations from multiple-factor large-prime relations. The new double-large-prime relations are of higher weight. On the other hand, a smaller graph depth is required in order to increase the graph size sufficiently. We therefore expect that this approach does not increase the average row weight in the relation matrix.

We will not investigate these variants of the algorithm in detail, because the advantages over the simple approach are rather small. Instead, we will comprehensively analyze the overall complexity of the large-prime graph method when combined with sieving.

3.3.2. Parameter Balancing for Complete DLP Computations over Low Degree Curves

Balancing the parameters for Algorithm 3.3.1 is performed similarly to the algorithms for high degree curves, i.e., we have to balance the total cost for building the relations over the

factor base against the cost of the linear algebra phase. We will first summarize the cost of relations sieving as introduced in Sections 3.2.1 to 3.2.3. Then, we will investigate the cost of the linear algebra phase when using relations as created in Algorithm 3.3.1.

Lemma 3.3.6. *Let C be a $C_{n,d}$ curve over $\mathbb{F}_q$ and let $\mathcal{F}$ be a factor base consisting of degree-one divisors of C with a total size of q^r with $r < 1$. Further, let*

$$\kappa := \frac{\#\mathcal{F}}{\#\{\, D \mid \ D \text{ divisor of degree } 1\}}.$$

Then there is an algorithm that generates the required cq^r single-large-prime relations over $\mathcal{F}$ in expected time

$$3\,(\eta-1)!\kappa c q^r q^{(\eta-2)(1-r)} O\left((\ln q)^2\right).$$

Proof. We assume the use of self-initialization as described in Theorem 3.2.13 for initializing the sieves. All divisors in the factor base are of degree one. Therefore, we have to set the parameter k of Theorem 3.2.13 to $\deg a$. Due to the relatively large q, we can select $M = 0$ and thus $k = \deg a = \eta$ is the ideal choice for the self-initialization parameter by applying Theorem 3.2.6.

Following the argument of Theorem 3.2.13, we thus can initialize $n^{\eta-1}$ sieves by using

$$q^r\left(\eta^2 + n\eta\right) O\left((\ln q)^2\right) + q^r n^{\eta-1}(\eta+1)\, O\left((\ln q)\right)$$

bit operations, where the O terms represent the costs for multiplying and adding in $\mathbb{F}_q$.

Sieving over $n^{\eta-1}$ sieve polynomials results in a total number of $qn^{\eta-1}$ tested divisors. Therefore, the average initialization costs for each divisor are

$$\begin{aligned} &q^{r-1}\frac{\left(\eta^2+n\eta\right)}{n^{\eta-1}} O\left((\ln q)^2\right) + q^{r-1}(\eta+1)\, O\left((\ln q)\right) \\ &\quad \le q^{r-1}\, 4\, O\left((\ln q)^2\right) + q^{r-1}(\eta+1)\, O\left((\ln q)\right), \text{ since } \eta \ge n \text{ and } \eta \ge 2 \\ &\quad \le q^{r-1}\, 5\, O\left((\ln q)^2\right) \text{ since } \eta \le \ln q. \end{aligned}$$

By Lemma 3.2.23, the expected time to sieve each sieve array is $q^r O(\ln q)$. Because each iteration tests q divisors for smoothness, we conclude that the total average cost for testing

a single divisor is bound by

$$q^{r-1}\, O\left((\ln q)^2\right).$$

Note that this time we included the constants into the O-term.

By inserting the result of Theorem 3.3.3, we can estimate the overall complexity for the relation search to be less than or equal to

$$3\,(\eta-1)!\kappa c q^r q^{(\eta-2)(1-r)} O\left((\ln q)^2\right).$$

Note that we can also include the additional factor of 3 into the $O()$-term. □

We will now compare the cost of the relation collection computed in Lemma 3.3.6 against the cost of the linear algebra phase to balance the parameter r between the two phases. Note that up to this point, we have not chosen an exact value for the parameter c in Algorithm 3.3.1 yet. We note that $c \geq 2$ is required to ensure the growth of the large prime graph and thus ensure termination of the algorithm. Besides this lower bound, we use the freedom to choose c to simplify the arising terms.

Theorem 3.3.7. *Given the notation and definitions of Lemma 3.3.6, assume* $\log_q(\eta-1) < \frac{1}{\eta-1}$ *and* $g! < q^{2-\frac{2}{\eta-1}}$.

Then there is an algorithm for solving the discrete logarithm problem on the curve that requires an expected time of

$$O\left(q^{2-\frac{2}{\eta-1}} \cdot (\ln q)^{3-\frac{2}{\eta-1}} \cdot (\eta-2)^2 \cdot g^{2-\frac{2}{\eta-1}}\right)$$

where $g = \frac{(n-1)(d-1)}{2}$ *is the genus of the curve.*

Proof. For the algorithm we use the sieve setup of the previous sections. As this does not give a complete solution of the DLP yet, we first take care of the missing decomposition of the group generator and of the group element of interest. Let one of those two elements be represented by (a, b) with $\deg a \leq g$. Then the elements $s \cdot a + (b - y)$ are of degree at most g for $s \in \mathbb{F}_q$. Therefore, we expect to require $g!$ factorizations of those elements until we find one that breaks down into linear factors.

Each factorization can be done in time $O(g^2(\ln q)^2)$ by using the Berlekamp algorithm. With the assumption on $g!$ we get an expected runtime of less than the proposed term for this step of the algorithm. After both factorizations are given, we can just include the resulting prime divisors of degree 1 into our factor base.

For the remaining relations we use Algorithm 3.3.1. The found relations have an average weight of

$$2(\eta-1)+(\eta-2)\cdot\frac{(1-r)\ln q}{\ln c^{\frac{\eta-1}{2}}}$$

by Theorem 3.3.2.

We choose $c=2e$ to simplify the terms. The arithmetic operations carried out during the linear algebra phase are in a field of size q^g. Therefore, we estimate the entire complexity of the linear algebra phase to be

$$\left(2(\eta-1)+(\eta-2)\cdot(1-r)\frac{\ln q}{1+\ln\frac{\eta-1}{2}}\right)q^{2r}g^2O\left((\ln q)^2\right)=(\ln q)q^{2r}g^2O\left((\ln q)^2\right).$$

The simplified estimate on the right-hand side is valid for $r=\frac{\eta-2}{\eta-1}+o(1)$ and $\eta-1<\ln q$. By this simplification, we avoid the appearance of the parameter r in the exponent as well as a factor, and we are able to state an explicit formula for the balanced r.

With this choice of c, the runtime of the relation search equals

$$(\eta-1)!q^rq^{(\eta-2)(1-r)}O\left((\ln q)^2\right)$$

by Lemma 3.3.6. Balancing the two sides omits one factor of q^r on both sides as well as the $O\left((\ln q)^2\right)$ term which is a placeholder for the cost of multiplying elements of length $O(\ln q)$. For the relation search, this is the multiplication of elements in $\mathbb{F}_q$ multiplied by a small hidden constant of $6e\kappa$ for small enough κ. For the linear algebra phase, it is instead the cost for the multiplication of integers close to q. To ease the analysis, we stay with the estimate of a quadratic multiplication algorithm although for both applications more advanced techniques like the Karatsuba multiplication or the Schönhage-Strassen algorithm are available.

Taking the logarithm to base q of the remaining terms then gives the equality

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

which is solved by

$$\begin{aligned} r &= \frac{\eta-2}{\eta-1}+\frac{\ln(\eta-1)!}{(\eta-1)\ln q}-\frac{2\ln g}{(\eta-1)\ln q}-\frac{\ln\ln q}{(\eta-1)\ln q} \\ &\leq 1-\frac{1}{\eta-1}+\frac{\ln(\eta-1)}{\ln q}-\frac{2\ln g}{(\eta-1)\ln q}-\frac{\ln\ln q}{(\eta-1)\ln q}. \end{aligned}$$

Note that the chosen upper bound for r also ensures that $\kappa \in O(1)$, because we assumed $\log_q(\eta-1) < \frac{1}{\eta-1}$. Inserting the upper bound for r into the estimate for the linear algebra runtime gives an overall runtime estimate of

$$O\left(q^{2-\frac{2}{\eta-1}}\cdot(\ln q)^{3-\frac{2}{\eta-1}}\cdot(\eta-2)^2\cdot g^{2-\frac{2}{\eta-1}}\right),$$

which completes the proof. □

3.3.3. Discussion of Results

Compared with previous results for small degree curves, the new algorithms presented in this thesis reduce the logarithmic terms in the complexity for solving the discrete logarithm problem. The two changes — the sieving algorithm and the new large-prime graph algorithm — have different contributions to the improvements. Similar to large-degree curves, the sieving process helps us to reduce the influence of the genus on the runtime. The most crucial point is that the increased efficiency of sieving over random divisor decomposition is large enough to compensate for the worse smoothness probability for small degree curves.

On the other hand, the new large-prime scheme allows a straightforward implementation, as we will present later in Section 4.1.2. Additionally, it helps us to save a factor of $g\ln q$ in the relation collection phase of the algorithms. Due to the sensitivity of the phase to the factor base size this result is only partially visible, because the extra factor of $\log q$ is still present for the linear algebra phase.

For hyperelliptic curves of low genus, the complexity estimates finally reduce to the following corollary. DLPs on these curves are generally considered more difficult to solve than DLPs on other types of curves.

Corollary 3.3.8. *Let C be an imaginary model hyperelliptic curve of genus g and let $g < \frac{\ln q}{\ln \ln q^g}$. Then there is an algorithm that solves any instance of the discrete logarithm problem on C in expected time.*

$$O\left(q^{2-\frac{2}{g}}(g-1)^2 g^{2-\frac{2}{g}} (\ln q)^{3-\frac{2}{g}}\right) < q^{2-\frac{2}{g}} g^4 O\left((\ln q)^3\right).$$

Proof. The proof is given by inserting $\eta = g + 1$ in the formulas of Theorem 3.3.7. Furthermore, we assume that $\kappa > 0.5$ due to the bound on g and therefore the assumptions for the theorem are satisfied. □

We see that for larger g, at least a factor of g is saved over existing approaches as described in Theorem 3.1.12. On the other hand, for the considered curves, the genus is bound by the logarithm of q. More concretely, for $2^{100} \leq q^g \leq 2^{200}$ — which is the size of the Jacobian of current cryptographic interest — g is less than or equal to 6 due to $g < \frac{\ln q}{\ln \ln q^g}$. Therefore, the saved cost is in the range $g^2 (\ln q)^{\frac{1}{3}}$.

However, also for non-hyperelliptic curves, there are certain advantages of the relation search via sieving over the established methods. This is particularly important for the curves of genus three.

Lemma 3.3.9. *Let C be a non-hyperelliptic curve of genus 3 over $\mathbb{F}_q$ with at least a single fully ramified point on C. Then any instance of the discrete logarithm problem can be solved in expected time*

$$O\left(q(\ln q)^2\right).$$

Proof. By Theorem 1.2.31 the fully ramified point guarantees the curve can be birationally transformed into a $C_{n,d}$ curve. All non-hyperelliptic $C_{n,d}$ curves of genus 3 are $C_{3,4}$ curves. Therefore, for $M = 0$, one can compute that the divisors associated with the possible sieve array have degree

$$\eta = \left\lceil \frac{n-1}{n} d \right\rceil = \left\lceil \frac{3-1}{3} 4 \right\rceil = \left\lceil \frac{8}{3} \right\rceil = 3.$$

Applying this to Theorem 3.3.7 gives the desired result. □

We see that for non-hyperelliptic curves of genus 3 we can solve the discrete logarithm problem in quasi-linear time with respect to q. This is an algorithm with cubic-root complexity with respect to the size of the entire Jacobian. Similar assertions are true for the non-hyperelliptic genus 4 curves. Table 3.1 summarizes the average complexity for certain small degree curves. We can derive all values from the result of Theorem 3.3.7.

n	d	Genus	η	Complexity	Remark
2	5	2	3	$O\left(q(\ln q)^2\right)$	On par with generic algorithms
2	7	3	4	$O\left(q^{\frac{4}{3}}(\ln q)^{\frac{7}{3}}\right)$	Hyperelliptic
2	9	4	5	$O\left(q^{\frac{3}{2}}(\ln q)^{\frac{5}{2}}\right)$	Hyperelliptic
2	$2g+1$	$g \geq 4$	$g+1$	$O\left(q^{2-\frac{2}{g}}g^{4-\frac{2}{g}}(\ln q)^{3-\frac{2}{g}}\right)$	Hyperelliptic
3	4	3	3	$O\left(q(\ln q)^2\right)$	All non-hyperelliptic of genus 3
3	5	4	4	$O\left(q^{\frac{4}{3}}(\ln q)^{\frac{7}{3}}\right)$	3rd root time algorithm
3	$3n+1$	$3n$	$2n+1$	$O\left(q^{2-\frac{1}{n}}g^{4-\frac{1}{n}}(\ln q)^{3-\frac{1}{n}}\right)$	
3	$3n+2$	$3n+1$	$2n+2$	$O\left(q^{2-\frac{2}{2n+1}}g^{4-\frac{2}{2n+1}}(\ln q)^{3-\frac{2}{2n+1}}\right)$	
4	5	6	4	$O\left(q^{\frac{3}{2}}(\ln q)^{\frac{7}{3}}\right)$	4th root time algorithm

Table 3.1.: Achieved complexity classes for selected $C_{n,d}$ curves of low degree

In addition to explicit results, the new algorithms also prove a conjecture of Diem and Kochinke [15] when restricting it to $C_{n,d}$ curves.

Conjecture 3.3.10 (Short Formulation of [15])**.** *For nearly all non-hyperelliptic curves of a fixed genus g at least 5, the discrete logarithm problem can be solved in an expected time of*

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

Here the authors use the $\widetilde{O}$ notation to hide logarithmic terms. In particular, due to the fixed g, every power of g is treated as a constant and the terms in $\log q$ are hidden due to the $\widetilde{O}$ notation. For the set of all non-hyperelliptic $C_{n,d}$ curves we can prove that the conjecture is true. One key is that the assumption of a fixed genus also ensures that $(\eta-1)!$ is a constant and is absorbed by the $\widetilde{O}$ notation.

Theorem 3.3.11. *For every non-hyperelliptic $C_{n,d}$ curve C of a fixed genus g at least 5 such that $\#\{D \in Div(C) \mid \deg D = 1\} \geq 0.5q$, the discrete logarithm problem can be solved in an expected time of*

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

Proof. The assertion is a direct consequence of Theorem 3.3.7. The extra condition in the corollary ensures that there are enough elements of degree one to build a factor base. To follow the desired complexity, we are left to prove that η is always less than or equal to $g-1$ for the curves of interest. We distinguish two cases.

First, assume that $n = 3$. Then the genus of a $C_{3,d}$ curve is $g = \frac{(n-1)(d-1)}{2} = d-1$ by Lemma 1.2.30. With Theorem 3.2.6 we see that $\eta = \left\lceil \frac{n-1}{n} d \right\rceil = \left\lceil \frac{2}{3} d \right\rceil$ for $M = 0$. When d is at least 6, the genus of the curve is 5, and we see that $\eta \leq d-2 = g-1$.

In case $n \geq 4$, we have $g > d$ but $\eta \leq d-1$. Therefore, for all of these curves the assertion holds.

Note that the assumption of a fixed genus g ensures that the factorial terms in the complexity estimates are constants. Therefore, the assertion also holds for curves where we usually would switch to larger factor base methods. However, the constants involved are high in this setting and the application of a solver with a subexponential factor base size will yield a better performance in practice. □

By the results of Theorem 3.3.7 we can improve the corollary further, because for curves with higher n the ratio between η and g will further improve. Moreover, we see that hyperelliptic curves from genus three onwards have a reduced DLP complexity due to the new algorithms.

However, it is yet unclear if the new algorithmic improvements are valuable for hyperelliptic curves of genus 2. From the complexity analysis point of view the sieving based algorithm is on par with generic algorithms like Pollard's Rho Algorithm 1.1.7. Due to the large constants in the O-notation in the cost estimates of divisor operations in the Jacobian, it is possible that sieving based methods outperform the generic algorithms for $g = 2$ curves when enough memory is available to hold the sieve array. A detailed analysis of this

application domain is left as a subject for future research.

3.4. Auxiliary Routines for DLP Computations

In this section, we will complete the investigation of algorithms that are required for a complete DLP computation. Specifically, this involves generating the factor base and the two special relations in order to execute Algorithm 1.1.13.

These two algorithms have in common that they only need to be executed once during the entire solution process. Nevertheless, their complexity can be dominant in the entire process, making a closer examination of these steps necessary.

3.4.1. Building the Factor Base

For certain types of curves, generating the factor base can be a computational bottleneck when solving discrete logarithms. In fact, as demonstrated in [16], the complexity of generating the factor base can dominate the entire algorithm. Therefore, it is critical to ensure that factor base construction does not become a bottleneck and can match advancements in relation-finding algorithms to maintain efficient overall complexity.

We begin by outlining an approach to finding all monic prime polynomials up to a given degree. This method is based on the classical sieve of Eratosthenes, adapted for polynomials. To optimize performance, we incorporate the sieve stepping technique that we described in the previous section to ensure efficient complexity for the sieving process.

Lemma 3.4.1. *Let $\mathbb{F}_q$ be a finite field and let $1 \leq B \in \mathbb{Z}$. Then we can find the set of all monic prime polynomials up to degree B by using $O\left(\lceil \frac{B}{2} \rceil q^B\right)$ finite field operations in $\mathbb{F}_q$ via a sieving based approach.*

Proof. We will use a sieving setup with a sieve array up to degree B. Then, we start sieving the array using the prime polynomials of degree one and $b = 0$ as the root. By using Algorithms 3.2.19 and 3.2.20 the total sieving time is $O\left(q^B\right)$ operations following the analysis in Theorem 3.2.21. Note that in this special situation, the unmarked elements in the sieve array are those of interest.

After this process we know the prime polynomials of degree two. Applying the process iteratively until we used all primes up to degree $\lceil \frac{B}{2} \rceil$ completes the process because each composite polynomial of degree less than or equal to B has at least one prime factor of degree less than $\lceil \frac{B}{2} \rceil$.

Following the arguments of Theorem 3.2.21, this entire procedure requires $O\left(\lceil \frac{B}{2} \rceil q^B\right)$ operations in $\mathbb{F}_q$. This completes the proof. □

Let f be the bivariate polynomial defining the function field. Then computing a dense factor base is equivalent to filtering out all those prime candidates p such that f has a linear factor $\pmod p$. This gives rise to the following algorithm for computing a factor base.

Algorithm 3.4.2.

Input: A smooth curve defined by a bivariate polynomial f over $\mathbb{F}_q$ and a bound $B \in \mathbb{N}$.

Output: A list of all (p, b'), such that p is a prime polynomial of degree at most B and t is a root of f modulo p.

1: Compute a list L of prime polynomials up to degree B by the approach of Lemma 3.4.1.
2: **for** $p \in L$ **do**
3: Reduce $f' \equiv f \pmod p$ and $s \equiv y^q - y \pmod p$ with $f', s \in \left(\mathbb{F}_q[x]/p\right)[y]$.
4: Compute $f'' = gcd\,(s, f')$
5: Use the Cantor-Zassenhaus algorithm [9], which is also described in appendix A, Algorithm A.3.2, for factoring f''. Each found factor is a factor of the form $(y - b')$ and gives rise to a factor base element represented by p and b'.
6: **end for**

All sub-algorithms of Algorithm 3.4.2 are of well-known complexity. Therefore, we can estimate the entire complexity for generating the factor base.

Theorem 3.4.3. *Let f be a bivariate smooth polynomial in variables x and y. Then we can compute all pairs $p, b' \in \mathbb{F}_q[x],\ \deg b' < \deg p \leq B$ such that b' is a root of $f \pmod p$*

with a time complexity of

$$O\left(B\left(\log_2 \deg_y F\right) q^B \left(\log q\right)^3\right).$$

Proof. Step 1 of Algorithm 3.4.2 requires no time for the sieve initialization because, for the search of prime polynomials, we will initialize the sieve with zero as the root modulo all inserted primes.

Following the analysis of Theorem 3.2.21, the cost for this first phase is less than Bq^{B-1} finite field operations in $\mathbb{F}_q$, since $M = B - 1$ for this particular case.

By the same assumptions that we used in Theorem 2.2.2, we can estimate that L contains approximately $\frac{1}{k}q^k$ prime polynomials of degree k for all $1 \leq k \leq B$.

The reduction of f and $y^q - y$ is cheap because we do not need to reduce any coefficients of $y^q - y$. Furthermore, the degree of coefficients of f in x is bounded above by the genus of the curve. In contrast, the gcd computation in step 4 has a complexity of $O(q)$ due to the high degree of $y^q - y$. To mitigate this high complexity, we compute $y^q - y \pmod{f'}$ by a square and add approach. This is of cost $O\left(\log_2 q \deg_y f'\right)$ finite field operations in $\left(\mathbb{F}_q[x]/p\right)$. Subsequently, the complexity of the gcd computation reduces to $O\left(\left(\deg_y f\right)^2\right)$ finite field operations in $\left(\mathbb{F}_q[x]/p\right)$.

The Cantor-Zassenhaus algorithm in step 5 finds the roots of f'' in expected time of $O\left(\log_2 q \left(\log_2 \deg_y f''\right) \deg_y f''\right)$ finite field operations in $\left(\mathbb{F}_q[x]/p\right)$. Note that $\deg_y f'' < \deg_y f$ and that the sum of all $\deg_y f''$ must be less than $2\frac{1}{B}q^B$ due to the estimates done in Theorem 2.2.2.

Therefore, we can assume that steps 2 to 6 of Algorithm 3.4.2 have a maximum complexity of

$$O\left(\log_2(q) \cdot \log_2\left(\deg_y f\right) \cdot \frac{2}{B} \cdot q^B\right)$$

finite field operations in finite fields of size at most q^B. Multiplying with the cost of finite field operations in a field of size q^B completes the proof. □

Assume that the desired size of the factor base is balanced with the effort for finding enough smooth candidates, as described in Lemma 1.1.16. Then Algorithm 3.4.2 for building the factor base is of lower complexity than the relation search, because the estimated

size of the factor base is $\frac{1}{B}q^B$. Nevertheless, the factor base computation can take a significant amount of time when not implemented properly.

3.4.2. Special Divisor Decomposition

One challenge in performing Algorithms 1.1.13 and 1.1.15 is finding the two special decompositions of α and γ over the factor base. In this context, α represents an element of the Jacobian in the cyclic subgroup generated by the element γ.

The complexity of this operation depends heavily on the curve parameters. For those curves with relatively large q and thus a factor base of size q^r with $r < 1$ an efficient method is to trial-factor randomized combinations of α and γ into elements of degree 1 without restricting to factor base elements. Afterwards, the primes involved in a successful decomposition can be used to either get part of the factor base or to initialize a sieve — and so get resolved via the relations found in the sieve iteration.

For curves of higher degree, expanding the factor base to include the decomposition is not always an efficient way of creating the decompositions. Still, for curves with high enough d and small n, such as hyperelliptic curves, this problem is relatively minor, as the degrees of α and γ are similar to those of the functions we typically decompose for general relations, i.e., elements of the sieve array. In such cases, we can initiate a random sequence starting from either α or γ by incrementally adding factor base elements and reducing the resulting divisor. Once reduced, we can trial factor the divisor; if unsuccessful, we continue by adding more random factor base elements.

Given the potential cost of the group law, alternatives that leverage the ideal decomposition property are also useful. For example, in [62] as well as in [18], an approach was used that is similar to sieving, but without the benefits of self-initialization.

Lemma 3.4.4. *Let $\langle a, b\rangle$ represent an ideal on a $C_{n,d}$ curve with $a, b \in \mathbb{F}_q[x],\ \deg a < g$ and a is irreducible in $\mathbb{F}_q[x]$. Let M be a search bound.*

Then, for all $\alpha = s \cdot a + (y - b)$ *with* $s \in \mathbb{F}_q[x], \deg s \leq M$, *we have*

$$a \mid \mathcal{N}(\alpha) \in \mathbb{F}_q[x] \text{ and}$$
$$\deg \frac{\mathcal{N}(\alpha)}{a} \leq \max\{d, n \cdot (M + \deg a)\} - \deg a.$$

Proof. The lemma is a direct consequence of Theorem 3.2.4 and the same calculations as carried out in the proof of 3.2.6 and, therefore, does not require a detailed proof. □

The strategy for decomposing an ideal associated with $\langle a, b \rangle$ involves searching for non-trivial values of α and attempting to factorize the norm's fraction into lower-degree prime polynomials. In the case of hyperelliptic curves, where $n = 2$ and the resulting degree is sufficiently close to the values in Theorem 3.2.6, it is feasible to sieve for such decompositions using the original factor base, although without the benefit of the efficient initialization scheme. Since the degree of a is slightly higher than optimal, the degree of the resulting norms of α is also slightly elevated. However, because the degree change is minimal and this step is performed only once for α and γ, it does not impact the overall complexity of the algorithm.

For curves with higher n, the algorithm can be executed iteratively. If $\frac{\mathcal{N}(\alpha)}{a}$ factors into prime polynomials of degree less than or equal to $\deg a - 1$, the process can be repeated starting with these primes. It is not necessary to continue the process down to the factor base bound B. Instead, it can be halted upon reaching $\approx \frac{d}{n} - M$, which is the optimal size for initializing a sieve for smooth divisors over the factor base. Hence, there are two potential stopping points, depending on which is reached first. We note that for a of an already small degree, the minimal degree of α is limited by d. This is identical to the lower bound for the degree that occurs in the sieving process. If this occurs, a can also be used to initialize a sieving process as described in Section 3.2.1.

The iterative process succeeds with the same time complexity as the search for general relations, provided that the smoothness probability is sufficiently high to make the first step of the iteration efficient. Subsequent iterations will start with a lower-degree a, resulting in a lower degree to factor. Since $\deg \alpha$ decreases by a factor of $n - 1$ each time $\deg a$ is reduced by one, this creates a condition that n is required to be small enough.

In [18], the authors suggest that for $n \approx g^{\frac{1}{3}}$ and $d \approx g^{\frac{2}{3}}$, this condition is satisfied because $\deg \alpha \approx g^{\frac{4}{3}}$ in the first step. As a result, the degree ratio between $\frac{\mathcal{N}(\alpha)}{a}$ and $g-1$ is comparable to that of an unconditioned relation search, where elements of approximately d are factored over a factor base of size roughly $L_{q^{\frac{2}{3}g}}\left[\frac{1}{2}, O(1)\right]$. The resulting algorithm is the one we already cited in Algorithm 3.1.5.

One problem with this argument is that it loses its validity for $n > g^{\frac{1}{3}}$ because the degree of the polynomials generated is too high. In particular, if $n \approx d$, we have $\deg \alpha \approx g^{\frac{3}{2}}$ in the first iteration and the discrepancy between this degree and $\deg a - 1$ is too large to hold the desired smoothness probability. We want to contribute an approach that reduces the degree of the elements in x at the expense of a higher degree in y to cover these cases.

Lemma 3.4.5. *Let $\langle a, b\rangle$ represent an ideal with $a, b, \in \mathbb{F}_q[x],\ \deg b < \deg a$. Consider the $\mathbb{F}_q[x]$-module of rank $k \leq n$ generated by $a, (y-b)^i$ with $0 < i < k$. Then there is a submodule basis $G_0, \ldots G_{k-1}$ such that $\deg_x G_i < \left\lceil \frac{\deg a}{k+1} \right\rceil$ for all $0 \leq i < k$.*

Proof. We provide a constructive proof.

We start by expanding $(y-b)^2 = y^2 - 2yb + b^2$. By adding $2b \cdot (y-b)$ two times, we get a preliminary form $y^2 - b^2$. By subtracting a sufficiently often, we can further reduce the last coefficient to $b^2 \pmod a$. By induction, we can apply this process to the entire generating system. Therefore, we assume we start with a generating set of $G_0 = a$ and $G_i := y^i - \left(b^i \pmod a\right)$.

From now on, we will handle the generators as polynomials in y with coefficients from $\mathbb{F}_q[x]$. From the initial reduced elements G_i we can proceed by lowering the degree of the constant term. We know that G_0 has a constant term of degree $\deg a$ and all other G_i of degree at most $\deg a - 1$. Assume that the degree of the constant term of G_1 equals $\deg a - 1$.

By multiplying G_1 by an appropriate constant from $\mathbb{F}_q$ and subtracting the result from the G_i for $i > 1$, we achieve that the constant term degree of all $G_i, i > 1$ is below $\deg a - 2$. The degrees of all other higher-order coefficients will still not exceed degree zero since we picked the constant from the subfield $\mathbb{F}_q$. By proceeding in the same way with the other G_i, we will produce a new generating set G_i' satisfying $\deg_x G_i' \leq \deg a - i$.

Now we can generate even lower degree elements with respect to x by the following procedure. Assume that the G_i are sorted by $\deg_x G_i$ descending. Then, by multiplying G_1 by a polynomial p of degree $\deg G_0 - \deg G_1$, we can achieve that $G_0 - pG_1$ has degree less than $\deg G_1$ in the constant coefficient. All higher order coefficients of $G_0 - pG_1$ have degree bounded by the sum of $\deg p$ and the degree of G_1 in these higher-order coefficients. If the degree of the constant term of $G_0 - pG_1$ matches the degree of any further G_i, we can further reduce the degree of the constant term of the element. Note that this time the maxima of the higher order coefficients degree are not increasing in degree, because the maximum of $G_0 - pG_1$ is already increased by $\deg p$ over the coefficient of G_i.

In total, we are able to reduce the degree of the constant term of G_0 by at least $k + \deg p$. Simultaneously, the degree of all other coefficients increases by at most $\deg p$ as long as the gap to G_1 is at most one with regards to this property. The usual case will be $\deg p = 1$. We then proceed by doing the same to the next elements decreasing the constant term by at least k while increasing all other coefficients only by one in their degree in x. The only exception is for the last element, where the degree increases by two, since the degrees of all later elements have already been increased, and p adds one more factor of degree one.

We will proceed with this step-down technique until $\deg_x G_i$ is no longer decreasing.

Assume that for the first element we started with — $G_0 = a$ — a total of $c < k$ iterations are performed. Then, we lowered the degree of its constant term to $\deg a - kc$, while the degree of all higher-order coefficients equal c. We reach the stopping condition once $\deg a - kc \leq c$, and therefore $c = \left\lceil \frac{\deg a}{k+1} \right\rceil$. Note that in the same iteration — or in the ones before — the other generators also gained $\deg_x G_i = \left\lceil \frac{\deg a}{k+1} \right\rceil$. This completes the proof and determines the stop condition for the other generators. □

The lemma proves that we can trade the degree of the ideal generators in x by increasing their degree of y. We will now provide an algorithm that will leverage this trade.

Algorithm 3.4.6.

Input: A $\mathbb{F}_q[q]$-module of rank n generated by $a, (y-b)^i$ for $0 \leq i \leq n$ and two parameters $B', M' \in \mathbb{N}$. Further, let $k = \min\left\{n, \left\lfloor \sqrt{\frac{d}{n} \deg a} \right\rfloor\right\}$.

Output: If it exists, an element z selected from the $q^{kM'}$ elements of the module with

lowest degree in x that decomposes into factors of degree less than B'.

1: Reduce the first k ideal generators to degree $\frac{\deg a}{k+1}$. Name the new generators $G_0, \ldots G_{k-1}$
2: **for** $\alpha = \sum_{i=0}^{k-1} c_i G_i$ with $c_i \in \mathbb{F}_q[x],\ \deg c_i \leq M'$ **do**
3: Compute $z := \frac{\mathcal{N}(\alpha)}{a}$.
4: Attempt to decompose z over $\{p \in \mathbb{F}_q[x] \mid p \text{ is a prime polynomial, } \deg p \leq B'\}$
5: **if** z is B' smooth **then**
6: Compute divisors corresponding to the factors of z of degree at most B'
7: Return the decomposition of z
8: **end if**
9: **end for**

In the following lemma, we provide an estimate of the degree of the elements generated by the algorithm.

Lemma 3.4.7. *Assume that the input divisor is a reduced divisor on a $C_{n,d}$ curve. Then z in line 3 of Algorithm 3.4.6 has a degree bounded by*

$$M'n + 2\sqrt{\deg a \cdot n \cdot d} - \deg a.$$

Proof. Following Lemma 3.4.5, the reduced generators of the sub-ideal have degree $k-1$ in y and degree $\frac{\deg a}{k+1}$ in x. Therefore, α has degree $M' + \frac{\deg a}{k+1}$ in x and the norm of α is a polynomial of degree

$$\left(M' + \frac{\deg a}{k+1}\right) n + (k-1)d$$

by applying Theorem 3.1.4. The terms influenced by k are balanced when $k = \frac{\sqrt{\deg a \cdot n + d}}{\sqrt{d}}$. We will first assume this choice of k is less than n.

Inserting the optimal k results in a degree of

$$M'n + 2\left(\frac{\sqrt{\deg a \cdot n + d}}{\sqrt{d}} - 1\right) d = M'n + 2\left(\sqrt{d}\sqrt{\deg a \cdot n + d} - d\right)$$
$$\leq M'n + 2\left(\sqrt{\deg a \cdot n \cdot d}\right).$$

Note that we ignored the fact that k is not an integer with this choice. This is compensated by the inequality since the additional $\frac{1}{k}d$ factor from rounding k is small enough. This completes the proof for $\frac{\sqrt{\deg a \cdot n + d}}{\sqrt{d}} \leq n$.

Conversely, when $n < \frac{\sqrt{\deg a \cdot n + d}}{\sqrt{d}}$ we can select k to be at most n. Then, we have a degree of at most

$$\left(M' + \frac{\deg a}{k+1}\right) n + (k-1)d < M'n + \deg a + (n-1)d$$
$$< M'n + \deg a + \left(\frac{\sqrt{\deg a \cdot n + d}}{\sqrt{d}} - 1\right) d$$
$$< M'n + \deg a + \sqrt{\deg a \cdot n \cdot d}$$
$$< M'n + 2\sqrt{\deg a \cdot n \cdot d},$$

because $\deg a \leq g$ and $n \cdot d \geq 2g$.

In both situations, we only have to subtract $\deg a$ – that is the degree of the denominator of z – to achieve the claimed result. □

Since the product $n \cdot d$ is $2g + o(1)$ for $g \to \infty$ and the degree of a also equals approximately g in the beginning of the reduction, we produce elements of degree only slightly larger than g during the first stage of the iterative process. This is a significantly lower degree compared to the above approach. However, the benefit of this method will decrease for lower degrees of a, because the scaling is limited by the square root.

Theorem 3.4.8. *Let an ideal be generated by a and $b - y$ in the function field $\mathbb{F}_q(x, y)$ of a $C_{n,d}$ curve of genus $g = \frac{(n-1)(d-1)}{2}$. Furthermore, let $\deg a = g^\alpha > 1$ for some $0 < \alpha < 1$, and assume $\frac{n+d-1}{2g} \leq 1$. Then, Algorithm 3.4.6 decomposes the ideal into ideals of degree*

less than or equal to $\deg a - 1$ *in expected time of*

$$L_{q^\eta}\left(\frac{1}{2}, \sqrt{2} + o(1)\right) \text{ for } \eta = \frac{g^{1-\alpha}}{\ln q} \cdot 8 \cdot \ln\ln q^g.$$

Proof. We start by computing the approximate degree of z generated in line 3 of the algorithm. By the assumption of $\frac{n+d-1}{g} \leq 1$ and Lemma 3.4.7, we get

$$\deg z - Mn < 2\sqrt{g^\alpha \cdot n \cdot d} \leq 2\sqrt{g^\alpha 2g(1 + \frac{nd - 2g}{2g})} < 4\sqrt{g^{1+\alpha}} - g^\alpha = 4g^{\frac{1+\alpha}{2}}.$$

The complexity of the decomposition of elements of this degree is mostly based on the smoothness probability, and so on the difference in degree between the factor base elements and the elements to be factored. We see that $\frac{\deg z}{\deg a}$ is about $4g^{\frac{1-\alpha}{2}} = u$. From the calculations performed in the proof of Theorem 2.4.7 we see that we have a subexponential smoothness probability of $L_{q^\eta}\left(\frac{1}{2}, -\frac{\sqrt{2}}{2}\right)$ if η satisfies $u = \sqrt{2\frac{\eta \ln q}{\ln\ln q^\eta}}$. We use the upper bound $\ln\ln q^\eta < \ln\ln q^g$ to be able to solve the equation for η and get the claimed value in the theorem. Note that because $g^\alpha > 1$, we know $g^\alpha - 1 \geq 1$ and the requirements of Theorem 2.4.7 are thus met. Finally, we take into account that the actual decomposition into ideals of degree at most $g^\alpha - 1$ can have twice as many factors if $g^\alpha = 2$. This is compensated for by squaring the smoothness probability. □

A consequence of Theorem 3.4.8 is that for rather small values of q and $d \geq g^{\frac{2}{3}}$ we can expect the entire DLP solving algorithm to be dominated by the search for general relations using the sieving setup that will create elements of size approximately d. On the other hand, when $g^{\frac{1}{3}} < n < d < g^{\frac{2}{3}}$, we still require a factor base of an approximate size of $L_{q^{\frac{2g}{3}}}[\frac{1}{2}, \frac{\sqrt{2}}{2}]$ in order to be able to resolve the two special divisors in a reasonable time.

However, for curves of low degree and large q, where the optimal factor base size with respect to the general relation search is q^r for $r < 1$ and so $\eta!$ is small enough not to dominate the complexity, we can perform the descent down to degree one by allowing all primes of degree one in the decomposition of our two special elements. In that case, we then can select the factor base to include these primes without expanding its optimal size.

3.5. Summary

In this chapter, we demonstrated that sieving methods are generally superior to traditional techniques, such as generating randomized divisors and trial dividing them, when searching for unrestricted relations on a $C_{n,d}$ curve. This is evident not only in theory, due to the reduced complexity per tested divisor, but also in practice, owing to the small constant factors hidden in the O-notation.

For hyperelliptic curves, where $g = \frac{d-1}{2}$, the sieve method requires the factorization of elements of slightly higher degree compared to pseudo-randomly generated divisors of degree g and a simple trial division. However, we have shown that for curves of higher genus, this difference in degree has a negligible impact on the optimal factor base size. In the case of lower-genus hyperelliptic curves, the improved efficiency of the sieving process compensates for any disadvantage in smoothness probability.

For non-hyperelliptic $C_{n,d}$ curves, the divisors generated by the sieve array are of degree approximately d. This lower degree allows for a smaller factor base compared to what would be required for divisors of degree g when using Algorithm 1.2.50. This advantage is especially pronounced for curves of small degree, such as $C_{3,4}$ curves, which are the simplest non-hyperelliptic cases. We have shown that, with an appropriately chosen double-large-prime strategy, the difficulty of solving the DLP for these curves scales with $O(q)$, which corresponds to $O(\sqrt[3]{q^g})$.

On the other hand, when the field $\mathbb{F}_q$ is fixed and the curve parameters are increased such that the optimal factor base size becomes subexponential, we demonstrated that the complexity of the DLP decreases as the difference between the curve invariants n and d diminishes. Specifically, for the curves discussed in [18], we showed that sieving methods yield similar complexities. In addition, we have complemented our methods with an approach that takes into account the reduced factor base size when generating the two special relations, expanding the set of curves with a proven DLP complexity of $L_{q^g}\left[\frac{1}{3}, O(1)\right]$.

Overall, we have developed a comprehensive and efficient toolkit for solving DLPs on $C_{n,d}$ curves.

4

Implementation and Numerical Results

In this chapter, we detail our implementations of the algorithms presented in this thesis, as well as the results of practical experiments.

We first present an adaptation of the sieving algorithm for hyperelliptic curves over extension fields of characteristic 2, building on the work of [62]. Subsequently, we introduce an implementation of the general relation-finding process and the large-prime graph method from Section 3.3 for curves over finite fields with an odd prime order. A notable feature of this second implementation is its design utilizing the computational power of graphics processing units, offering a significant speed-up compared to traditional CPU-based methods.

After presenting the implementations, we discuss experiments conducted with each, including the resolution of specific discrete logarithm problems (DLPs) and performance evaluations using test curves of varying genus.

4.1. Description of Implementations

4.1.1. Implementation for Large Genus Hyperelliptic Curves over Even Characteristic Fields

4.1.1.1. Overview and Experiments Setup

The initial implementation used in our experiments is an evolution of the framework developed by Velichka et al. [62]. It is written in C++ and uses Shoup's NTL library [54] to handle polynomials over a characteristic two finite field.

The factor base computation and the special divisor decomposition are processed serially using a single CPU core. We used the OpenMPI library to distribute the general relation search and the distributed sieve initialization to multiple CPU cores on a computer cluster.

The implementation uses the efficient sieve iteration described in Section 3.2.3, as well as the sieve initialization described in Theorem 3.2.9 that reuses the roots of the defining polynomial modulo the factor base elements.

Finally, for the linear algebra step, we used two implementations. For the curves of Section 4.2.1, we used a slight variant of the original implementation of Jacobson, Stein and Velichka. This is a minor variation of the Wiedemann algorithm (see Algorithm A.4.11). The exact adjustments we made for our implementation will be described in the next subsection.

We performed all experiments with this implementation on the TOP 500 [56] high-performance cluster (HPC) named "CARL", located at the University of Oldenburg (Germany) and funded by the DFG through its Major Research Instrumentation Programme (INST 184/157-1 FUGG) and the Ministry of Science and Culture (MWK) of the Lower Saxony State. The computing nodes used for the experiments featured two Intel Xeon E5-2650 v4 Processors with 12 CPU cores each. Each node is equipped with 128 gigabytes of DDR4 memory. The nodes are interconnected with a 54.54 Gbit/s Infiniband interconnect for the jobs exceeding the capacity of a node. For the experiments of Section 4.2.4, the implementation utilizes an NVIDIA Tesla P100 graphics card in the testing nodes equipped with 3840 compute cores and 16 gigabytes of HBM2 memory.

4.1.1.2. Implementation Details

Specialized Arithmetic in Characteristic 2

Since this implementation uses fields of characteristic 2 only, we decided to use the special arithmetic of those fields to speed up our sieve code.

In the case where q is even, there exist hardware-supported operations on $\mathbb{N} \cup \{0\}$, allowing Algorithm 3.2.20 to be implemented more efficiently and better embedded into Algorithm 3.2.19. This operation is the *exclusive-or* operation, which we defined in Appendix A.5.2.

We will leverage the fact that $\oplus$ is a hardware-supported operation, as fast as standard integer addition, to accelerate the computation of the ν function. In this context, ν is the function mapping polynomials to sieve array entries.

As shown in Appendix A.5.2, $\oplus$ defines a group law on $\mathbb{N} \cup \{0\}$ with 0 being the neutral element. In fact, the following lemma proves $(\mathbb{F}_2[x], +) \cong (\mathbb{N}_0, \oplus) \cong (\mathbb{F}_q[x], +)$ with the latter isomorphism given by ν.

Lemma 4.1.1. *Let q be even and let $f, g \in \mathbb{F}_q[x]$. Further, let m, n be two natural numbers in their 2-adic representation as in Definition A.5.2. Then the following assertions hold.*

a) If $a_i \neq 0$ implies $b_i = 0$ for all $i \leq \min\left(\lfloor \log_2\ n \rfloor, \lfloor \log_2\ m \rfloor\right) + 1$, then $m + n = m \oplus n$.

b) $\nu(f + g) = \nu(f) \oplus \nu(g)$

Remark 4.1.2. *The operation $\oplus$ allows us to combine the lists H and L from Algorithm 3.2.20 and merge the algorithm with the sieve iteration Algorithm 3.2.19. This is because, if the list S from Algorithm 3.2.19 already holds evaluated values, the only term we must add in order to get the next value is $\nu(a_{is}p)q^s$. We can do this by using $\oplus$, because the addition of vectors translates to using $\oplus$ with the evaluated integers.*

Note that multiplying by q^s is just an integer shift in this setting. Furthermore, we can calculate $\nu(a_{is}p)$ by using $\deg(p) + 1$ multiplications in $\mathbb{F}_q$ plus $\deg(p)$ $\oplus$ operations, which corresponds to addition over our field.

This yields the following algorithm, which is exactly Algorithm 3.2.20 with the proposed change as in the above remark.

Algorithm 4.1.3 (Optimized Sieving in Characteristic 2)**.**

Input: Polynomials $p, r \in \mathbb{F}_q[t]$ with $\deg(p) = n$ and a sieve radius M.
p is given as an array $P[]$ of integer representations of its coefficients.

Output: Modifies the sieve-array $R[]$ such that $n := \deg(p)$ is added to all
$R[\nu(k \cdot p + r)]$ with $\deg(k \cdot p + r) \leq M$.

Set $S[i] = \nu(r)$ for all $0 \leq i < M - n + 2$.
Set $k' = 0$ and $K[i] = 0$ for all $0 \leq i < M - n + 1$.
$R[S[0]] \leftarrow R[S[0]] + n$
while $(k' < q^{M-n+1})$ **do**
$k' \leftarrow k' + 1$
$l \leftarrow 0$

while $K[l] = q - 1$ **do**
$K[l] \leftarrow 0$
$l \leftarrow l + 1$
end while
$K[l] \leftarrow K[l] + 1$
$S[l] \leftarrow S[l + 1]$

$temp \leftarrow 0$
for $i = n; i \geq 0; i = i - 1$ **do**
$temp \leftarrow (temp << n)$
$temp \leftarrow temp \oplus \text{FieldMult}_{\mathbb{F}_q}(K[l], P[i])$
end for
$temp \leftarrow temp << (l \cdot n)$
$S[l] \leftarrow S[l] \oplus temp$
$R[S[l]] \leftarrow R[S[l]] + n$

while $l > 0$ **do**
25: $l \leftarrow l - 1$
$S[l] \leftarrow S[l+1]$
end while
end while

We see that the main loop of the algorithm consists of three blocks. The first, starting at line 8, finds the most significant coefficient of k changing in the current step. The second block from lines 15 to 22 adds $K[l]P$ to $S[l]$ by iterating through the coefficients of P. Finally, the last block starting in line 24 updates the lower indexed values of the array S to ensure our loop invariants.

We implemented this pseudo-code almost directly. This gave us a very compact sieve implementation.

The same arithmetical tricks can also be used for root computations. Since ν is additively homomorphic with respect to $\oplus$, it is also possible to store $\nu(\pm B_i)$ from Theorem 3.2.13 and process the necessary operations for computing new roots on the memory addresses directly. We used this benefit throughout the implementation in a way that the polynomial representation of elements — both for the factor base elements and their roots and for the field arithmetic itself — is barely used during the sieve setup phase.

Precomputations

The implementation makes significant use of precomputations to lower the arithmetic overhead, in particular of the field multiplication. We use the precomputations whenever $B > 1$ and $M > 1$, i.e., the factor base contains primes of degrees higher than one, and the sieve array contains more than just the constant value used. Such a setup is optimal once g exceeds a value of approximately 10 for the parameter range of our experiment.

In this setting, an array of q^2 elements is stored containing the memory addresses $\nu(a \cdot b)$ for all $a, b \in \mathbb{F}_q$. We use the array for performing the constant multiplication as needed in line 20 of Algorithm 4.1.3.

Note that due to the $\oplus$ operation replacing the addition directly, it does not require any pre-computation.

Linear Algebra

The linear algebra phase of the implementation in [62] — that is the variant of the Wiedemann algorithm using Theorem A.4.12 — works by solving an under-constrained linear system, i.e., with more columns than rows. While a solution is likely in this setting, applying a Wiedemann-based approach presents technical challenges for this task, because the algorithm requires square matrices to find a divisor of the minimal polynomial of this matrix. Therefore, the non-square matrix is transformed into a square matrix by taking multiple random combinations of the previous rows as additional rows.

For this pattern, the implementation must insert enough extra coefficients for the artificial rows. Otherwise, overly trivial linear combinations decrease the probability of obtaining a sequence generating polynomial with non-zero constant coefficient that we need to find a solution to the non-homogeneous system. On the other hand, using too complex linear combinations compromises the sparseness of the linear system – in particular, because there are some single rows with high density due to the self-initialization.

We chose a probabilistic approach for our implementation that gives a better average runtime. First, we remove all found relations that contain a prime factor that is unique among all relations. Furthermore, we remove this prime factor from the factor base. Therefore, we cancel out the same amount of rows and columns. We further reduce the number of rows by removing all primes that did not appear in any factorization found. This way, we are left with more columns than rows.

Next, we start by taking the square submatrix containing all rows but only the first columns. If the Wiedemann algorithm succeeds in finding a solution to our system, we have already solved the discrete logarithm problem. Otherwise, the algorithm likely found a divisor of the matrix's minimal polynomial with trivial constant coefficient and thus gives a vector in the matrix's kernel. We use this vector as a guiding information regarding which matrix column — equivalent to one relation — we replace by one of the previously discarded relations. We then repeat the process.

It is immediate that this approach will succeed if and only if the initial linear system also has a solution. Thus, we have the advantage of a square linear system with the probability to require multiple iterations to find a solution. As for the runtime, we observed that among all our experiments we had only one curve that required to run more than one iteration. Still, the benefit was large enough to compensate for the extra iteration.

4.1.2. Implementation for Small Genus Hyperelliptic Curves over Odd Characteristic Fields on Graphics Cards

The second implementation for our experiments complements the first one. While the first was optimized for high-genus curves over extension fields of $\mathbb{F}_2$, the second is an implementation of Algorithm 3.3.1 designed for low-genus curves over prime fields $\mathbb{F}_p$, where $p \geq 5$. Unlike the first implementation, this one does not use MPI for parallelizing the relation search across multiple compute nodes. Instead, we focused on optimizing the implementation to leverage heterogeneous computing, accelerating the relation search.

Heterogeneous computing is a recent trend in computational number theory that takes advantage of the ability to use various computational devices within a single system to handle intensive tasks. Although this idea is not new — dating back to floating-point co-processors and acceleration boards in the early days of home computing — the recent availability of affordable computational power, particularly through graphics processing units (GPUs), has revitalized interest in heterogeneous computing.

The core idea is to utilize specialized hardware that delivers exceptional performance for specific applications. In our case, we use this hardware to handle the most computationally expensive parts, while the coordination, result aggregation, and other algorithmic components remain on a standard CPU.

GPUs, in particular, stand out for their numerous arithmetic logic units (ALUs), which can execute simple arithmetic operations efficiently but lack many of the complex functions found in a traditional CPU. Additionally, their control flow is simplified, often because they are built as SIMD units. SIMD, which stands for *single instruction, multiple data*, means that large blocks of arithmetic units execute the same instruction simultaneously,

each working with a different data stream.

This makes such processors especially well-suited for algorithms that are highly parallelizable and involve minimal branching instructions. This is crucial because each branching point can stall the units that are on a different branch. Thus, deeply nested branches and loops with variable depths can cause significant performance issues on these devices. In [61], we discuss techniques for implementing arithmetic in prime fields of odd characteristic for SIMD devices with minimal branching.

To fully exploit the capabilities of modern GPUs, we implemented the sieve initialization and the sieving process in a highly parallel manner. As a demonstrator, we employed the algorithms presented in Section 3.3.1 to solve the discrete logarithm problem for small-genus curves in our host program. This choice was made because solving the discrete logarithm problem for small-genus curves demands more computational power compared to higher-genus cases.

The remainder of this section is structured as follows: First, we provide a brief introduction to OpenCL, an open standard and programming model for heterogeneous hardware platforms. Next, we detail our implementation. Finally, we present computational results and timings observed with our implementation.

4.1.2.1. An Introduction to OpenCL

In this section, we will give a short introduction to OpenCL to explain the constraints on our implementation. First, we will give insight into how work is partitioned and distributed in the OpenCL programming model.

The OpenCL Work Model

The smallest unit of work in OpenCL is a *task*. Each task is the work that is executed on a *stream-core* and is similar to an ordinary single-threaded program. The programming of the kernel is done at the task level such that the co-processor executes the OpenCL code similarly to ordinary programming code.

The next relevant level of execution hierarchy is a *warp* (in NVIDIA terminology) or a *wavefront* (in AMD terminology), which are equivalent concepts. A warp represents

the smallest unit of tasks that a co-processor can execute as a group. All tasks within a warp must be executed using the exact same instruction set and must follow the same branches, even if they operate on different data. If a code branch depends on data values, both branches are executed, and only the relevant result is retained for each task. Similarly, if loops within a warp have different lengths, all cores processing the warp must continue until the longest loop completes. If the total number of tasks is less than the warp size, the device will still consume the same time and resources as it would for a full warp. On NVIDIA GPUs as well as newer AMD GPUs, a warp or wavefront contains exactly 32 tasks, while on earlier AMD GPUs each wavefront has 64 tasks.

In the OpenCL programming model, the next execution unit is a *work-group*. Like a warp, a work-group comprises tasks that follow the same execution path. However, tasks within a work-group can be executed serially, while those in a warp execute simultaneously. A work-group is the largest unit in OpenCL that can be synchronized within the code via synchronization barriers. These barriers ensure that all memory operations within a work-group are completed before the code proceeds, allowing tasks to synchronize at specific points. For optimal performance, the work-group size should be a multiple of the warp size. Most modern frameworks impose limits on the number of tasks per work-group, as most implementations cannot synchronize an arbitrary number of tasks.

The *global work* represents the highest level of task aggregation on an OpenCL-compatible device, as defined in the OpenCL standard. It is at this level that the host code operates, handling data transfers between the host program and the OpenCL co-processor. All tasks within the global work level run the same code, though not necessarily at the same time. Synchronization across all tasks at the global work level is only possible when the co-processor code finishes and control returns to the host program. The global work consists of one or more work-groups, all of equal size.

The OpenCL Hardware Model

Similar to the task level, the OpenCL standard defines multiple hardware levels. Each *OpenCL device* is composed of one or more *compute units*, which are independent units with respect to control flow. Each compute unit is responsible for managing the synchro-

nization of work-groups, and each work-group is assigned to exactly one compute unit. Compute units are truly independent in terms of control flow, so two compute units can execute different programs if shared resources permit. The exact number of compute units varies depending on the device and its manufacturer. As a result, the total workload can be executed serially on a few compute units or in parallel across multiple compute units, with any mixed mode in between.

Each compute unit, in turn, consists of one or more *stream cores*, each of which can perform task operations. While each stream core has its own register memory, it shares control flow with other stream cores within the same device. The total number of stream cores per compute unit is designed to be a divisor of the number of tasks within each warp.

The OpenCL Memory Model

Multiple different memory locations are present on an OpenCL device. The memory location offering the highest bandwidth is the *register memory*. Register memory is located at the stream core level and OpenCL programs use it for private variables, temporary values and small local arrays in each task. Compared with the register memory in an ordinary CPU, a graphics card offers a large amount of register space. This allows the program to perform large computations using this memory location only.

In addition to register memory, there is a certain portion of *local memory* that is placed at the compute unit level. Local memory is visible exclusively at the work-group level and the program can use it for synchronization and collaboration within a work-group. Reading and writing to local memory are more time-consuming than the same operations on register memory but the access is cheaper than for the further memory regions. Modern graphics processing units (GPUs) allow reading and writing one or two words (32 to 64 bit) per task per clock cycle from and to local memory. For sharing data between threads, the program must synchronize the local memory access explicitly by using barriers if the corresponding work-group exceeds the size of one warp or wave-front. When using a CPU as an OpenCL device, the local memory is emulated using the system RAM or cache.

The largest portion of available memory is *global memory*, which an OpenCL program uses for communication with the host program and thus as the place to get inputs and store

results from and to each task. Access to global memory is slower than for the previously mentioned regions. Furthermore, it takes multiple clock cycles as a latency. Within a work-group, it is possible to synchronize reads and writes from and to global memory. Outside the boundary of a work-group, this option does not exist. While the OpenCL global memory space resides in system RAM when using a CPU, it is hosted on the GPU's dedicated hardware for textures and frame buffers. Therefore, the host-to-client communication before and after the computations on the device is limited by the available bandwidth of the PCIe bus system.

4.1.2.2. Implementation Details

Our implementation uses the new large-prime graph scheme presented in Algorithm 3.3.1 for small genus hyperelliptic curves over prime fields of odd characteristic. The factor base storage, graph storage, and large-prime factor elimination are handled by the host program on a CPU. In contrast, the factor base generation, (self-)initialization, and sieving are executed on the GPU. For these operations, we employ branch-free arithmetic as detailed in [61]. We assume that the base field fits within the word size of the GPU, allowing us to handle prime fields up to 32 bits in size. This is sufficient for our tests since most degree-0 polynomials will be elements of the factor base; for larger field sizes, the factor base size might exceed practical RAM capacities. However, a 64-bit arithmetic variant, which could manage larger fields, is feasible at a modest additional cost.

To ensure branch-free algorithms, we use modular inversion based on Fermat's Little Theorem, i.e., if a is invertible modulo a prime p, then $a^{-1} \equiv a^{p-2} \pmod{p}$. This enables a branch-free implementation, requiring $O(\log p)$ finite field operations, avoiding the Euclidean algorithm, which, though faster on single cores, does not scale well in SIMD contexts. To minimize the number of inversions, we use a standard technique: we multiply elements that require inversion together, compute one inversion, and then obtain individual results through modular multiplication.

For the factor base generation, we use the Tonelli-Shanks algorithm (see Algorithm A.3.7) to find roots of the function field-defining polynomial. We apply this process to all degree-1 prime polynomials whose constant term is below a predefined bound. Approx-

imately half of these polynomials have no root, as these primes are inert in the function field. If no root is found, a distinguished value is returned to the host, and the corresponding prime polynomial is excluded from the factor base.

For the sieve initialization, we do not fully apply self-initialization, since storing 2^g finite field elements for each factor base element in device memory would exceed the available memory capacity, particularly on GPUs. Therefore, we compute only the necessary inverses a^{-1} as specified in Theorem 3.2.13. When sieve polynomials share the same a during self-initialization, a^{-1} is common to all, saving instructions by precomputing and storing this single value per factor base element.

The code fragment that computes a^{-1} first reduces $a \mod p$ for the degree-one prime polynomial $p \in \mathcal{F}$. This is done by evaluating a at the constant term of the monic p. We then invert $a \mod p$ using the procedure described above. Each computational core processes four different values of p concurrently, replacing the individual inversions with three modular multiplications, reducing the need for multiple modular inversions. The implementation of the initialization follows directly from Theorem 3.2.9.

For the sieving process, we utilize OpenCL *atomic operations*, which allow simple arithmetic on local or global memory without further synchronization. Atomic operations serialize multiple simultaneous memory access requests to the same address, ensuring orderly operations by queuing instructions. For example, if two cores attempt to add a value to the same array entry simultaneously, the hardware scheduler assigns an execution order.

Due to limited local memory space, we store the sieve array in global device memory. To reduce PCIe bus communication, we retain most intermediate results in device memory. Parameters for initialization are the only data sent to the GPU; relations with at most two missing large-prime factors are transferred back to the host. We divide the sieving process into two stages, with the first stage directly attached to the initialization to ensure synchronization across work-groups, as OpenCL lacks a direct inter-group synchronization method. The algorithm below describes the process applied to each sieve polynomial root.

Algorithm 4.1.4.

1: Given a root s of the sieve polynomial modulo a norm a.
2: $atomicAdd(sieveArray[s], 257)$
3: Global synchronization by termination of OpenCL code. The following code fragment is part of a second OpenCL program.
4: $t := atomicSubtract(sieveArray[s], 1)$
5: Let $t = u \cdot 256 + v$
6: **if** $u \geq g$ **then**
7: $\quad r = atomicAdd(globalResultNumber, 1)$
8: $\quad$ Store s and a at the address r.
9: **end if**
10: **if** $v = 1$ **then**
11: $\quad sieveArray[s] \leftarrow 0$
12: **end if**

The implemented sieve works in two phases, separated by a synchronization step. In the first pass, we loop over all roots and add entries to the sieving array, while in the second pass we read back the results. If sufficiently many roots match, we write them out to a storage space for found relations.

To implement this scheme efficiently, we use an encoding that adds the number $257 = 256 + 1$ into each memory cell accessed. This ensures that after the first phase, we have two copies of the counter with one byte of distance in memory. We use these two copies in the second phase: one remains constant to indicate the total count, while the second decreases with every access.

By doing so, each thread accessing a specific memory address in the second phase knows if the value written in the first phase was high enough to constitute a relation. Therefore, each thread can individually and independently decide if the handled prime and root are worth storing. At the same time, the decreasing copy of the counter allows us to track if other threads still need to access this specific memory address. If not, it is safe for the

current thread to reset the content of the address to zero, cleaning up the sieve array for the next sieve polynomial.

This is only the brief version of the implemented scheme. For example, the sieve array entry at $s = 0$ usually satisfies $u \geq g$, but the relation corresponding to this position is trivial.

Algorithm 3.3.1 requires us to find $O(q)$ double-large-prime relations. For double-large-prime sieving, we have to recall that a sieve array entry of $g - 1$ can also correspond to a relation with one large-prime factor of degree two. Therefore, we must distinguish between a single-large-prime relation with a prime factor of degree two and the desired one with two large-prime factors of degree one. Additionally, for the desired situation, we must reconstruct the two missing divisors at low cost. Finally, for the single-large-prime relations found, we need to recover the missing large-prime factor.

Theorem 3.2.13 allows us to initialize the sieve without computing the full sieve polynomial. Therefore, we investigated a way of computing the missing factors without computing the entire sieve polynomial. We will derive an explicit form of the sieve polynomials as constructed in Algorithm 3.2.3 for hyperelliptic curves. This will later allow computing the required coefficients without calculating the entire sieve polynomial.

Lemma 4.1.5. *Let C be a hyperelliptic curve defined by $y^2 = f(x)$ over $\mathbb{F}_q$ for odd q. Further, let $a, b \in \mathbb{F}_q[x]$ with $a \mid b^2 - f$ represent an ideal generated by $\langle a, b - y\rangle$ on the curve. Then the sieve polynomial $f_{a,b}$ corresponding to the ideal $\langle a, b - y\rangle$ as constructed by Algorithm 3.2.3 equals*

$$f_{a,b}(s) = a \cdot s^2 + 2bs + \frac{b^2 - f}{a}.$$

Proof. Let $\alpha = a \cdot s + (y - b)$ as in Algorithm 3.2.3 and Theorem 3.2.2. Then the norm of α is equal to multiplying α by its conjugate, since the function field of C is a quadratic extension of $\mathbb{F}_q(x)$. Since q is odd, $\bar{\alpha} = a \cdot s + (-y - b)$. Consequently

$$\begin{aligned}\mathcal{N}(\alpha) = \alpha\bar{\alpha} &= (a \cdot s + (y - b))(a \cdot s + (-y - b)) \\ &= a^2s^2 + as(-2b) - (y^2 - b^2) = a^2s^2 - 2asb + (b^2 - f)\end{aligned}$$

Since the sieve polynomial should exclude the known prime divisors dividing a, the final sieve polynomial is given by

$$f_{a,b}(s) = \frac{\mathcal{N}(a \cdot s + (y - b))}{a} = as^2 - 2sb + \frac{b^2 - f}{a}.$$

□

For this implementation, we restrict to sieve arrays of size q. Therefore, s is only a constant polynomial and thus in $\mathbb{F}_q$. Furthermore, we can always choose a such that it is of optimal degree g and that it is a product of monic linear factors. Also, we can assume that f is monic.

Lemma 4.1.6. *Given the setting of Lemma 4.1.5. Let p_i denote the i-th coefficient of a polynomial p. Then the following assertions regarding the sieve polynomial $f_{a,b}$ hold.*

$$\begin{aligned} f_{a,b_{g+1}} &= -1 \\ f_{a,b_g} &= -f_{2g} + s^2 + a_{g-1} \\ f_{a,b_{g-1}} &= -f_{2g-1} + a_{g-2} - a_{g-2}(-f_{2g} + a_{g-1}) - 2sb_{g-1} + a_{g-1}s^2 \end{aligned}$$

Proof. The formulas arise from the formula in Lemma 4.1.5. We start with the fraction $\frac{b^2-f}{a}$. First of all, the degree of b is less than or equal to $g-1$. Since $\deg f = 2g+1$, we see that the highest three coefficients of $b^2 - f$ are identical to $-f$. By a polynomial division we obtain

$$\frac{b^2 - f}{a} = -x^{g+1} + (-f_{2g} + a_{g-1})x^g + \left(-f_{2g-1} + a_{g-2} - a_{g-2}(-f_{2g} + a_{g-1})\right) x^{g-1} + r(x),$$

where r is a polynomial of degree $g-2$ in x.

The middle term $2sb$ has no term of degree g, because $s \in \mathbb{F}_q$ and $\deg b \leq g-1$. The highest degree term is $2sb_{g-1}$. With the same arguments, the highest degree terms of the first term as^2 are s^2x^g and $a_{g-1}s^2x^{g-1}$. □

Given the three highest degree coefficients of $f_{a,b}$, we can reconstruct the missing factors of single and double-large-prime-relations.

Corollary 4.1.7. *Let $s \in \mathbb{F}_q$ such that s represents a single-large-prime relation in the sieve array corresponding to the sieve polynomial $f_{a,b}$. Let $c_1, \cdot, c_g$ denote the constant coefficients of the known linear factors dividing $f_{a,b}(s)$.*

Then the norm of the single-large-prime factor has constant coefficient

$$c_{lp} = f_{a,b_g}(s) - \sum_{i=1}^{g} c_i = -f_{2g} + (s^2+1)a_{g-1} - \sum_{i=1}^{g} c_i.$$

If s represents a double-large-prime relation instead, the unknown factor of the prime decomposition of $f_{a,b_g}(s)$ is quadratic. Let l be the missing linear term and c be the constant term of the unknown quadratic polynomial. Then

$$l = f_{a,b_g}(s) - \sum_{i=1}^{g-1} c_i = -f_{2g} + (s^2+1)a_{g-1} - \sum_{i=1}^{g-1} c_i$$
$$c = f_{a,b_{g-1}}(s) - l - \sum_{i \neq j} c_i c_j.$$

Both assertions in the corollary are consequences of a direct polynomial division. We use both formulas to reconstruct the missing large-prime factors from the almost smooth sieve array entries we have found. This can be done without computing and factoring the entire sieve polynomial. For the double-large-prime relations, we factor the quadratic polynomial by completing the square. To do so, we used $O(q)$ precomputations to compute the squares and their square roots in $\mathbb{F}_q$.

4.2. Experimental Results

4.2.1. A Family of Curves from Weil Descent

We test our sieve implementation described in Section 4.1.1 on four imaginary hyperelliptic curves of genus 31 defined over different fields of characteristic 2. These curves and their corresponding discrete logarithm problems were constructed using the Weil descent technique (Section 1.2.6). The Jacobians of all these curves have order $2 \cdot p$, where p is prime,

making them challenging to solve using only generic algorithms.

Based on the base-2 logarithm of the size of their Jacobians, we designated the curves as *C62*, *C93*, *C124*, and *C155*. The precise equations for each curve and the points used for the discrete logarithm problems are provided in Appendix C.1.

For our experiment, we utilized three different implementations. The first is based on Algorithm 1.2.50, using a randomized sequence of divisors and trial factoring. This implementation follows the work of Jacobson, Menezes, and Stein [32]. The second implementation used is the original sieve by Jacobson, Stein, and Velichka [62]. Lastly, we tested our own modified sieve implementation, as previously described. All three implementations incorporated a single-large-prime method similar to the one outlined in Section 3.1.4. In our variant, we allowed large-prime factors to have degrees up to a certain bound, which served as an additional parameter in the implementations.

To determine optimal parameters for factor base sizes and sieve bounds, we conducted tests by finding at least 5% of the necessary relations under different configurations, selecting those that offered the shortest estimated times for the complete search. In this process, we factored in the initialization times for the sieves and extrapolated them for the entire relation search. We also accounted for the increasing number of relations found through our single-large-prime algorithm. The parameters we chose are summarized in Table 4.1.

We structured the parameter table as follows. First, we begin with the chosen factor base bound $B = m$ and the size of the factor bases $\#\mathcal{F}$. Next are the *large-prime bound* for the large-prime method and the sieve array bound M.

The parameter P_{Once} equals the parameter k in Theorem 3.2.14, i.e., it determines the number of prime divisors of the factor base involved in creating the sieve polynomials. This comes with the twist that in the implementation of [62], as well as in our modification of it, all these factors were required to have the same degree. As a result, depending on the prime factors of $g - M$, it was not always possible to initialize the sieve with optimal degree inputs.

Finally, the parameter T determines the tolerance value for the sieving algorithm as described in Definition 1.2.55.

Note that for the random-divisors-based implementation, the required CPU time was

prohibitively long for the two larger Jacobians. Therefore, we compared this implementation only with the two smaller Jacobians. Furthermore, we use the hyperelliptic involution to halve the number of relations that we require for solving the discrete logarithm problem over our factor bases.

	C62		C93		C124	C155
Implementation	Random Walk	Both Sieves	Random Walk	Both Sieves	Both Sieves	Both Sieves
B	7	6	5	5	4	4
$\#\mathcal{F}$	3,288	948	7,744	7,744	17,744	273,056
Large-Prime Bound	8	7	6	6	5	5
M		8		4	4	4
P_{Once}		8		9	9	9
T	30	30	30	30	30	30

Table 4.1.: Parameters for DLP Solving on the Example Curves

Note that in [62] some experiments were done using *small prime sieving*. Small prime sieving is a technique where the sieve is run only with smaller degree primes and a considerably lower threshold bound T. Then all sieve array entries exceeding T were trial divided to ensure full smoothness. We refined the implementation and fixed some minor issues in Velichka's sieve. Thereby, we found out that it was more beneficial to use the entire factor base, exactly as with our new implementation.

To ensure all approaches work with almost equal factor base sizes, we decided to choose $B = m \in \mathbb{N}$, where B is the maximum degree of the factor base elements and q^m is the approximate size of the factor base. This may lead to a non-optimal factor base size but makes the results less sensitive to the implementation and randomness. In Section 4.2.3 we will provide the results of other experiments that suggest that $m = B$ is in fact ideal for many curves.

We structure each result table as follows. For all three methods we list two kinds of times. If no additional indicator is given, the time is the actual time that passed during the corresponding segment of our experiments. In contrast, we also provide the *core times*, i.e., the runtimes the routines would require if run sequentially on a single compute core. To compute these values, we multiplied the real timings by the number of CPU cores involved for better comparability and to account for parallelism. We provide these two types of timings for the sieve setup and the general relation search. We ran the linear algebra computation as well as the search for the two special relations on a single CPU

	Random Walk [32]	Sieving [62]	Our Implementation
Setup Time	1.32s	0.40s	0.41s
Relation Time	45.81s	25.35s	0.40s
Setup Core Time	31.68s	9.60s	9.84s
Relation Core Time	18m 19.44s	10m 8.4s	9.60s
Special Relations Time		5.42s	0.57s
Linear Algebra Time	1.39s	4.14s	0.94s
Total Time	48.52s	35.31s	2.32s
Total Core Time	18m 52.52s	10m 27.56s	20.95s

Table 4.2.: Timings of DLP Solving on the 62 bit Jacobian of C62

core only. Therefore, the two timings are equal for this phase.

	Random Walk [32]	Sieving [62]	Our Implementation
Setup Time	3.47s	2.09s	2.08s
Relation Time	1h 33m 14.87s	6m 34.62s	11.73s
Setup Core Time	1m 23.28s	50.16s	49.92s
Relation Core Time	1d 13h 18m	2h 37m 51s	4m 41.52s
Special Relations Time		2m 19.4s	2m 2.82s
Linear Algebra Time	9.98s	7m 56.76s	58.15s
Total Time	1h 33m 29s	16m 52.87s	8m 32.41s
Total Core Time	1d 14h 53m	2h 48m 58s	4m 4.70s

Table 4.3.: Timings of DLP Solving on the 93 bit Jacobian of C93

Finally, we give the timings for the linear algebra calculations and aggregate the total CPU-Core time as well as the real used time. Note that the linear algebra matrices used for both sieving algorithms were equal. On the other hand, the matrices produced by the random walk implementation of [32] were more sparse. This holds because matrices generated from the randomly created divisors process lack the extra entries caused by the self-initialization process.

The two smallest examples are completed within a few minutes by the two sieving-based algorithms. For the random-divisors-based implementation, the curve C93 is already more challenging. Overall, we see that the sieving based implementations have large timing advantages over the random-divisors-based approach, although the linear algebra phase consumes more time.

The C124 test curve benefits more from the new algorithms than the other curves. This is primarily because, with the chosen parameters, the factor base and sieve arrays are rela-

	Sieving [62]	Our Implementation
Setup Time	4.45s	4.47s
Relation Time	18h 33m 38s	8m 12.25s
Setup Core Time	1m 46.8s	1m 47.28s
Relation Core Time	18d 13h 28m	3h 16m 54s
Special Relations Time	1h 0m 6s	21m 35.55s
Linear Algebra Time	1h 0m 1s	7m 51.16s
Total Time	20h 33m 50s	38m 43.43s
Total Core Time	18d 15h 29m	3h 48m 8s

Table 4.4.: Timings of DLP Solving on the 124 bit Jacobian of C124

	Sieving [62]	Our Implementation
Setup Time	1m 19.13s	1m 20.22s
Relation Time	15d 9h 31m	3h 57m 19s
Setup Core Time	31m 39.21s	32m 5.29s
Relation Core Time	369d 9h	3d 22h 56m
Special Relations Time	2h 38m 21s	27m 49.47s
Linear Algebra Time	20d 2h 0m	2d 14h 56m
Total Time	35d 13h 40m	2d 18h 55m
Total Core Time	389d 14h 10m	6d 14h 25m

Table 4.5.: Timings of DLP Solving on the 155 bit Jacobian of C155

tively small compared to the entire group size. As a result, the implementations need to run a larger number of sieves to find each individual relation, requiring frequent reinitializations. Thus, $C124$ gains significantly from the new, more efficient initialization routines. However, this routine is not yet implemented for initializing the sieve used to find the two special relations, making this step a runtime bottleneck. Based on a limited set of generated relations and results from other experiments, we estimate that the total core time for the random walk implementation on curve $C124$ would be slightly more than one year. Consequently, we opted not to run the random walk-based algorithm for the two larger Jacobians.

Also, we conclude that the sieving-based algorithms make the 155-bit example feasible on general-purpose hardware within a week. In particular, our new implementation suggests that a desktop computer is also capable of running a solver for this problem range in decent time. For larger experiments, we will need to drop the original implementation from [62] due to timing restrictions on our computer cluster.

In the next section, we provide a test result of the new implementation with an even

more complex discrete logarithm problem. To do this, we will also switch to using multiple computing nodes.

4.2.2. 184 Bit Hyperelliptic DLP from a Trapdoor Curve

In [59], Teske describes a scheme for creating elliptic curve discrete logarithm problems that include a trapdoor, i.e., that allow the creator of the underlying curve to solve the discrete logarithm problem without knowledge of the used private key with feasible effort. The author mentioned key escrow applications as one beneficiary of such discrete logarithm problems.

The technique behind this system involves constructing an elliptic curve that is susceptible to the Weil descent attack (see Section 1.2.6) over the field $\mathbb{F}_{2^{161}}$. Through Weil descent, discrete logarithm problems on these curves can be translated into discrete logarithm problems on hyperelliptic curves of genus 7 or 8 over $\mathbb{F}_{2^{23}}$. Using suitable algorithms, such as the Enge-Gaudry algorithm (see Algorithm 1.1.12), the resulting problem is considered challenging but feasible. Specifically, based on the results of Jacobson, Menezes, and Stein [32], Teske estimated that solving a discrete logarithm problem in the genus 8 case could take up to 200,000 CPU days on a Pentium III workstation. This estimate aligns with the predicted runtime for the C155 curve we examined in the previous section.

Once the elliptic curve and the descent map are constructed, a second elliptic curve is created, isomorphic to the first via a randomly generated isogeny. A discrete logarithm problem on this new curve can be resolved by the curve's creator using the known isogeny and the Weil descent. However, for external attackers, discovering the inverse isogeny — and thus identifying the vulnerable curve — presents significant challenges, making it difficult to break the discrete logarithm problem without resorting to a generic attack. As of 2017, discrete logarithm problems on non-vulnerable curves with a Jacobian of size 2^{161} are considered unbreakable within practical time limits using generic algorithms like Pollard's Rho algorithm (see 1.1.7).

For sieving-based methods, we assume that breaking a discrete logarithm problem on a genus 8 curve over $\mathbb{F}_{2^{23}}$ would require more time than solving the genus 31 problem over $\mathbb{F}_{32}$. This is because the size of the Jacobian for the genus 8 curve is $2^{8 \cdot 23} = 2^{184}$, approximately

Factor Base Size	$3,700,000$ $(B = 1)$
Sieve Array Size	$8,388,607$ $(M = 0)$
Total Relations	$1,850,005$
Relations from Large-Prime Factors	$1,754,816$ (86%)
Sieve Polynomials Used	$268,872,823$
Smoothness Probability	$1.512 \cdot 10^{-10}$
Large-Prime Smoothness Probability	$1.519 \cdot 10^{-9}$
Setup Time	3m 53s
Relation Time	6h 17m 11s
Setup Core Time	4d 3h 25m
Relation Core Time	402d 1h 16m
Large-Prime Combining Core Time	1d 15h 13m
Special Relations Core Time	14h 59m 55s
Linear Algebra "krylov" Time	17h 40m 39s
Linear Algebra "plingen_pz" Time	6h 58m 5s
Linear Algebra "mksol" Time	2h 17m 16s
Misc Linear Algebra Time	7m 12s
Total Linear Algebra Core Time	393d 18h 37m
Total Core Time	795d 20h 2m

Table 4.6.: Timings of DLP Solving on the 184 bit Jacobian of C184

$536,870,912$ times larger. For random-walk-based algorithms, this factor is less significant, as it is possible to operate solely within a subgroup of size 2^{161} generated by the image of the elliptic curve's Jacobian.

To validate the efficiency of the algorithms proposed in this thesis, we created a random discrete logarithm problem on the first example curve outlined in the appendix of [59]. We then mapped this problem to the Jacobian of a genus 8 hyperelliptic curve over $\mathbb{F}_{2^{23}}$. To solve the hyperelliptic curve discrete logarithm problem, we employed the modified Velichka implementation described in Section 4.1.1.

For our experiments, we utilized a subset of the compute nodes from the Top 500 cluster "Carl" at the University of Oldenburg, as detailed in Section 4.1.1.1. To collect the general relations, we deployed 64 compute nodes, leveraging a total of 1,536 processor cores for the sieving process.

During the computation of general relations, it became apparent that the original large-prime combination algorithm from [62] was not optimized for the large factor base sizes used in this example. As a result, the large-prime combination step consumed nearly three-

quarters of the total runtime, while utilizing only a single CPU core. Although this approach saved overall CPU time, reducing the number of potential large-prime factors might have shortened the actual relation collection time.

Finding the special relations was relatively straightforward because our implementation successfully identified the two required relations after 153 and 284 sieve setups. Although this is roughly twice and three times the theoretically expected number of trials, it still falls within the anticipated range.

For the linear algebra phase, we transitioned from the matrix representation outlined in Algorithm 1.1.13—employed in the [62] implementation—to the format specified in Algorithm 1.1.15. Using this updated representation, we utilized the tools from CADO-NFS 2.3 [57], an efficient implementation of the number field sieve that includes a state-of-the-art solver for homogeneous linear systems over finite fields. This software employs a multi-phase variant of Coppersmith's block Wiedemann algorithm [13], which is more easily parallelized than the original Algorithm A.4.12. The "krylov" and "mksol" phases were executed on a 4x4 MPI and 6x4 local grid setup, engaging 384 CPU cores in total. Due to the algorithm's requirements, the intermediate "plingen_pz" phase ran on the same 16 cluster nodes with pure MPI parallelization in a 16x16 grid, utilizing 256 CPU cores. The overall linear algebra phase required a total wall-clock time equivalent to 393 days, 18 hours, and 37 minutes of single-core CPU time. This indicates that the chosen factor base size was well-balanced in terms of single-core computational efficiency.

By applying the algorithms proposed in Chapter 3 to the [62] implementation, we were able to solve the problem in under 800 single-core CPU days. Its worth noting that our test machine was equipped with an Intel XEON E5-2650 v4 processor, which has a higher clock rate than the 1 GHz Intel Pentium III workstation used in [32]. Additionally, the newer processor benefits from architectural improvements that increase instructions per clock cycle. However, because our algorithms do not exploit modern techniques like SSE or AVX and instead rely on basic arithmetic, a single core of a recent 2.2 GHz processor is estimated to be about four times faster than the older Pentium III cores. Taking these factors into account, we estimate that the algorithmic enhancements led to a tenfold improvement in runtime, without considering the added complexity from the larger Jacobian.

4.2.3. Investigation of the Smoothness Probability and Algorithm Runtime for Varying Genus

The scope of our third series of experiments is to verify the outcomes of the smoothness probability analysis of Chapter 2, as well as the theoretical complexity analysis of the sieving algorithm as presented in Section 3.2. In order to do so, we constructed hyperelliptic curves of different genera over characteristic 2 fields such that the overall size of the Jacobian was approximately 2^{120}.

We constructed the factor base with a fixed size of $20,000$ elements. Therefore the maximum degree B of factor base elements is the least integer satisfying $\frac{1}{B}q^B \geq 20,000$. The value of $20,000$ is slightly lower than the theoretically optimal factor base size for a Jacobian of size 2^{120}, but the experiments in the previous two sections suggest that for our current implementation the timings are more balanced with a smaller factor base.

The lowest genus tested was $g = 8$. For a genus 8 curve and $q = 2^{15}$ we have $\#\mathcal{F} < q$, but the factor base size is still above the optimal factor base size for the single-large-prime algorithm 3.1.8. Therefore, this example curve represents the edge case between the presumed exponential time complexity for low-genus curves and the subexponential complexity for high-genus curves.

The following table gives an overview of the fixed parameters we chose for this series of experiments.

q	2^1	2^2	2^3	2^4	2^5	2^6	2^8	2^{10}	2^{12}	2^{15}
g	120	60	40	30	24	20	15	12	10	8
$\#\mathcal{F}$	20,000									
$\log_q(B \cdot \#\mathcal{F})$	18.45	8.72	5.62	4.15	3.25	2.65	1.91	1.52	1.27	0.95
B	20	10	6	5	4	3	2	2	2	1

Table 4.7.: Overview of genus and finite field sizes for experiments of Section 4.2.3

For all curves we tested multiple combinations of the sieve array size M and the number of primes for initializing the sieve array P_{Once}. Due to the implementation of the sieve polynomial generation, we had to ensure that $\frac{g+1+M}{P_{Once}}$ is an integer lower than the factor base bound B. In those cases where this was not possible and the fraction was not an

integer, the elements associated with the sieve array were of slightly higher than optimal degree and thus worsened the smoothness probability.

The exact curve equations and the results for all experiments conducted can be found in Appendix C.2. For the analysis, we extract some of the data to support our analysis of the data series.

Influence of the Algorithm Parameters regarding the Smoothness Probability

By Theorem 2.5.1 the expectation is that for factor bases of identical size and constant q^η the smoothness probability does not alter significantly. If η is increased, we expect a lower smoothness probability, because the divisors associated with the sieve array are of higher degree.

In our experiments we verified the latter assumption by modifying the sieve bound M as well as P_{Once}. A larger sieve bound will generally increase the number of divisors tested for smoothness, but also increases the best achievable value of η, because $\eta_{opt} = g + 1 + M$ by Theorem 3.2.6. We note that not all elements associated with the sieve array will always have the same degree. To denote that, the appendix tables designate the η value associated with the largest degrees of array elements by η_{max}.

The high-genus curves — $C60$ and $C120$ — offered multiple configurations for M and P_{Once}, resulting in different element sizes η_{max} for the sieve array. Since the initialization system uses divisors of a single degree, we could isolate variables: we tested fixed M with varying η_{max} by changing P_{Once}, and vice versa. We observed that increasing η_{max} consistently reduced the smoothness probability as expected, whereas the size of the sieve array itself M had minimal impact.

Notably, when running sieves of varying sizes with fixed P_{Once} and identical η_{max}, the smoothness probabilities remained within a very tight range. It is important to note, however, that this implies some of these sieves were not initialized optimally with respect to achieving the ideal η_{max}.

In contrast, comparing different values of η_{max}, but either the same M or the same P_{Once}, we saw that an increased value of η_{max} always gave the expected reduction in smoothness probability.

An interesting observation occurred when varying P_{Once} while keeping the sieve array size and η_{max} constant. Doing so has only a minor influence on the factor base, unless P_{Once} becomes very large. For almost all curves, we observed that if $\frac{\eta}{P_{Once}} < \frac{B}{2}$, the smoothness probability decreases dramatically. This was particularly evident for the genus 40, 60, and 120 curves. For curves of lower degree, the parameter constraints often precluded testing this scenario. Unfortunately, we did not find a definitive explanation for this behavior. We hypothesize that prime divisors of degree less than $\frac{B}{2}$ are scarce. Consequently, relations containing many such small factors are rare, yet the initialization strategy forces their inclusion. The exact reasons for the observation may be worth an investigation in a future work.

Our experiments confirm that the smoothness probability is independent of the specific sieve polynomial used, provided the polynomial is constructed from prime divisors of sufficiently large degree. This is a crucial finding, as it supports the general assumption that divisors in the search range behave like random divisors of the same degree with respect to smoothness. The precise discrepancy between this assumption and the actual smoothness probability — along with potential strategies to achieve 'better-than-expected' outcomes — are topics worthy of further investigation, though they lie beyond the scope of this thesis.

Influence of the Curve regarding the Smoothness Probability

Up to this point, we only investigated changes in the smoothness probability when changing the algorithm parameters for the same curve. However, for curves of generally high enough degree, Theorem 2.5.1 also implied that the smoothness probability should not change significantly for factor bases of identical size and constant q^{η}, but different curves with different base field sizes and curve genera.

Table 4.8 summarizes the least possible theoretic values η_{opt} for the curves in our experiments. Alongside the theoretically optimal values of η_{opt}, we also give the actual values η_{max} that differ from the optimum due to the limitation of the initialization system to use divisors of one degree only.

The last row of table 4.8 gives the measured smoothness probability for the given parameters. We observe that the correlation between the smoothness probability and $q^{\eta_{max}}$ is

weaker than expected. This is particularly true for the lower genus curves. For example, the measured smoothness probability was worst for the genus 12 curve, although the ratio between q^η and the factor base size was best among the curves.

q	2^1	2^2	2^3	2^4	2^5
g	120	60	40	30	24
B	18	9	6	5	4
η_{opt}	138	69	46	35	28
η_{act}	139	70	46	36	28
$\log_2 q^{\eta_{\text{opt}}}$	138	138	138	140	140
$\log_2 q^{\eta_{\max}}$	139	138	138	144	140
$\mathcal{P}(\eta_{act}, \mathcal{F})$	$1.32 \cdot 10^{-8}$	$1.02 \cdot 10^{-8}$	$5.37 \cdot 10^{-9}$	$1.06 \cdot 10^{-8}$	$1.24 \cdot 10^{-8}$
q	2^6	2^8	2^{10}	2^{12}	2^{15}
g	20	15	12	10	8
B	3	2	2	2	1
η_{opt}	23	17	13	12	9
η_{act}	23	17	13	12	9
$\log_2 q^{\eta_{\text{opt}}}$	138	136	130	144	135
$\log_2 q^{\eta_{\text{act}}}$	138	136	130	144	135
$\mathcal{P}(\eta_{act}, \mathcal{F})$	$6.76 \cdot 10^{-10}$	$1.50 \cdot 10^{-8}$	$1.10 \cdot 10^{-11}$	$1.31 \cdot 10^{-10}$	$5.11 \cdot 10^{-9}$

Table 4.8.: Measured smoothness probability for $M = B - 1$ and chosen parameters

The proof of Theorem 2.4.7 suggests that the smoothness probability is particularly good when as many elements of degree B as possible are included in the factor basis. In this case, there are particularly many genuinely different ways to generate smooth divisors over the factor basis that contain few prime factors of high degree. In the asymptotically written smoothness probability estimates, this effect is hidden in the $o(1)$ term.

To confirm our conjecture, we added an additional series of experiments. For this series, we chose the genus 24 curve over $\mathbb{F}_{2^5}$, because this curve has a high smoothness probability, as well as a short compute time for testing one divisor for smoothness. For all the extra tests

we chose the sieve bound M to be three. Therefore, the sieve array had $1,048,576$ elements. By using $P_{Once} = 7$ and so prime divisors of degree three for the sieve initialization, we optimize the divisor degree η_{max} associated with the sieve array.

The extra test series runs over factor bases of different size. Since all factor bases with a size below 600 elements contain only prime divisors of degree two or less, we chose 700 elements as a minimum to ensure there are enough degree three prime divisors for initialization. Starting with 1,000 elements we increased the number of factor base elements by 500. From a size of 20,000 elements onwards, we increased the step size in steps of 2,000 elements.

Overall, we chose the range for $\#\mathcal{F}$ to span from 700 to 274,000 elements. This range covers all possibilities for building factor bases with elements up to degree four. Increasing the factor base size to the next step of 276,000 elements already includes prime divisors of degree five. All experiments were run until at least $\frac{\#F}{2}$ smooth elements were found.

We included all results of this series of experiments in Table C.19 in the appendix of the thesis. Figure 4.1 contains a visualization of the number of required sieve polynomials (light gray) and the required time to complete the simulation (dark gray). The figure does not include the results for $\#\mathcal{F} < 9,000$ as including them would have caused a bad scaling for the larger factor base sizes.

As expected, the smoothness probability increases monotonically with a growing factor base size. Interestingly, we observed that for the first degree-four elements added to the factor base, the smoothness probability grows more slowly than the rate of increase in factor base size. Thus, between 11,544 and approximately 24,000 factor base elements, the number of divisors tested to complete the set of relations increases, as the higher smoothness probability does not sufficiently offset the increased demand for relations caused by the expanded factor base.

With further increases in factor base size, this effect diminishes as the growth rate of the smoothness probability surpasses that of the factor base size. Starting at around 42,000 factor base elements, the required number of tested divisors is actually lower than it was with 11,544 factor base elements.

We conclude that the asymptotic $o(1)$ term of our smoothness probability formulas

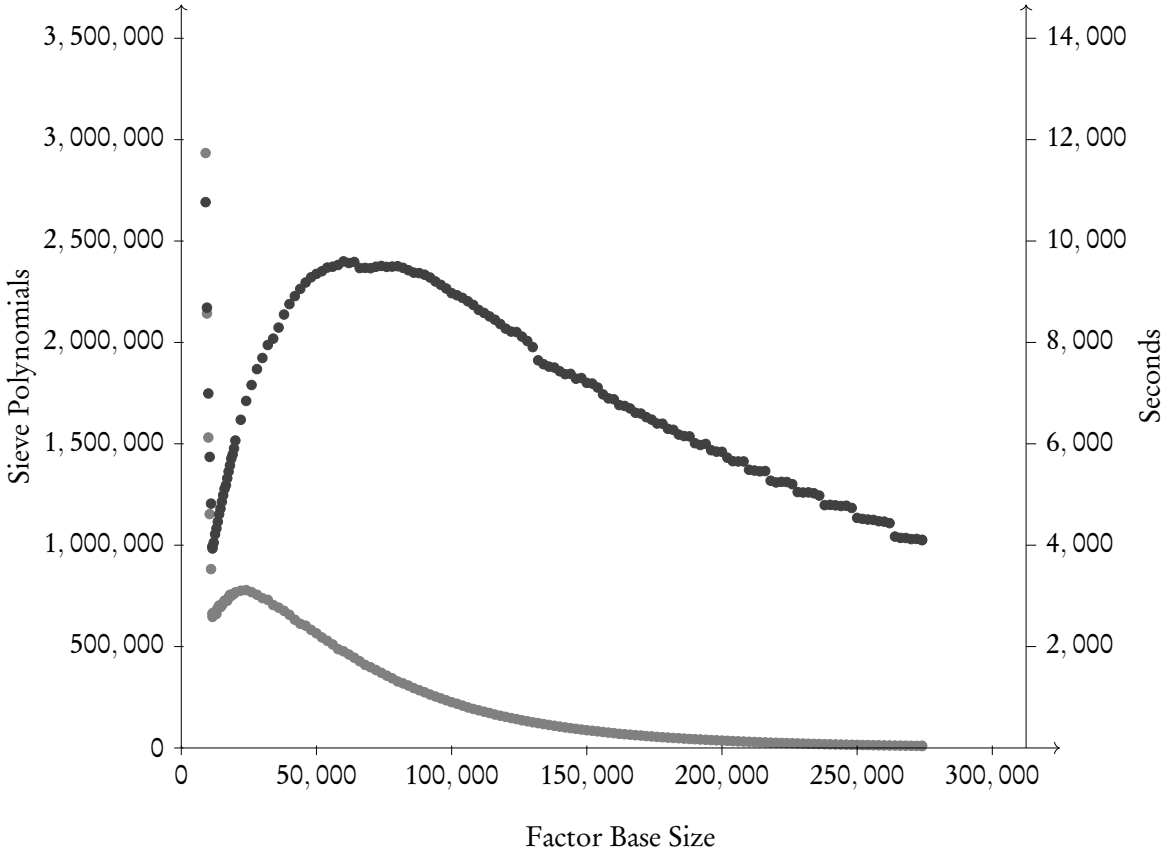

Figure 4.1.: Required Sieve Polynomials and Timings for Varying Factor Base Sizes

masks a practical reality: when the factor base size is only marginally larger than $\frac{1}{B-1}q^{B-1}$, the increase in smoothness probability fails to offset the computational overhead of processing the expanded factor base.

Therefore, we anticipate that the ratio between factor base elements and smoothness probability is optimal when $\log_q(B \cdot \#\mathcal{F})$ is close to an integer, which is either B or $B-1$. We have added this number into the list of all test curves for the original element.

While not a formal proof, the data correlates strongly with the observed smoothness probabilities. When the factor base size is close to $\frac{1}{B}q^B$ or slightly above $\frac{1}{B-1}q^{B-1}$, we observe a high smoothness probability relative to the factor base size. Conversely, the worst-case scenario occurs when $|\log_q(B \cdot \#\mathcal{F}) - B| \approx 0.5$ for large q. For smaller q, this effect is mitigated because the absolute ratio between the actual factor base size and the nearest factor base size that would use all divisors of degree B, is bound by q.

The assumptions of Theorem 2.4.7 require that $\sqrt{\eta}\sqrt{\ln\ln q^g} \geq \frac{\ln 2+4d+4g}{\sqrt{\ln q}}$ holds. We

q	2^1	2^2	2^3	2^4	2^5
g	120	60	40	30	24
B	18	9	6	5	4
$\mathcal{P}(\eta_{act}, \mathcal{F})$	$1.32 \cdot 10^{-8}$	$1.02 \cdot 10^{-8}$	$5.37 \cdot 10^{-9}$	$1.06 \cdot 10^{-8}$	$1.24 \cdot 10^{-8}$
$\log_q(B \cdot \#\mathcal{F})$	18.45	8.72	5.62	4.15	3.26
q	2^6	2^8	2^{10}	2^{12}	2^{15}
g	20	15	12	10	8
B	3	2	2	2	1
$\mathcal{P}(\eta_{act}, \mathcal{F})$	$6.76 \cdot 10^{-10}$	$1.50 \cdot 10^{-8}$	$1.10 \cdot 10^{-11}$	$1.31 \cdot 10^{-10}$	$5.11 \cdot 10^{-9}$
$\log_q(B \cdot \#\mathcal{F})$	2.65	1.91	1.52	1.27	0.95

Table 4.9.: Complemented Version of Table 4.8

observe that as the genus increases, the left-hand side of this inequality grows faster than the right-hand side. Overall, this requirement imposes a lower bound on η, which in turn implies a lower bound on g for any $C_{n,d}$ curve. Therefore, for increasing q, the genus must also increase for the theorem to hold; consequently, as $q^g \to \infty$, the observed deviation in smoothness probability is $o(1)$.

Influence of Curve and Algorithm Parameters regarding the Sieving Algorithm Performance

To evaluate the performance, we measure the throughput in terms of tested divisors per second. Equivalently, one can analyze the average time required to test a single divisor for smoothness. Like the smoothness probability itself, this performance metric depends on multiple factors. The supplementary tests in the previous section used the sieve array sizes and P_{Once} values listed in Table C.19 in the appendix. These tests demonstrate that the smoothness probability significantly impacts the average test time per divisor. This occurs because a higher smoothness probability requires more frequent post-processing of relations, specifically the verification of relations and checks for duplicates. Consequently, the performance penalty exceeds the expected cost of a larger factor base, resulting in longer initialization and sieving times than anticipated.

Isolating the influence of the parameter P_{Once} is challenging because the value $\frac{g-M}{P_{Once}}$ must remain an integer between $\frac{B}{2}$ and B to avoid drastically worsening the smoothness probability, as described in the previous section. These conditions were satisfied only for tests on curves of genus 20, 24, 60, and 120. In additional evaluations where the sieve array size

was held constant while the associated degree varied with P_{Once}, we observed that performance generally improves with larger P_{Once}. This is due to the more efficient utilization of the algorithm described in Theorem 3.2.13. We therefore recommend maximizing P_{Once} as much as possible, provided the smoothness probability remains unaffected. Ultimately, the impact of the smoothness probability on overall performance dominates the sensitivity to P_{Once}.

The second critical parameter is the sieve array size, which must also be balanced against the smoothness probability to optimize performance. When analyzing the sieve array size in isolation, the algorithm performs optimally for sizes between 2^{20} and 2^{24}. For smaller sizes, detailed computational logs reveal that the initialization phase dominates the total runtime. Conversely, for larger arrays, the sieving phase becomes the bottleneck, and the computation speed slows down as the array grows. This slowdown cannot be explained by general complexity theory regarding the test time per divisor.

Instead, we attribute this behavior to hardware-specific constraints rather than algorithmic complexity. As the sieve is implemented as a byte array, arrays up to 2^{22} elements fit entirely within the CPU's L3 cache of our test CPU. However, when accounting for auxiliary data like the factor base elements and their roots, the effective cache limit is approximately 2^{20} sieve array elements. Beyond this point, cache lines must be spilled to main RAM, causing the process to become memory-bandwidth bound. This hardware bottleneck outweighs the theoretical reduction in relation testing complexity offered by larger arrays. From approximately 2^{24} elements upwards, the cache controller can no longer compensate for this latency.

This must be considered for larger Jacobians, since in this situation one will often deal with larger sieve arrays in the machine RAM. Aside from this, the specific genus of the curve does not appear to inherently impact the relation smoothness testing speed. Since the smoothness probability also stabilizes for sufficiently large genus, the overall performance becomes highly predictable, provided the sieve array size and P_{Once} are tuned correctly. This underscores the necessity of initializing the sieve with a configurable number of factor base divisors of varying degrees to ensure optimal performance. This flexibility is particularly vital for production-grade implementations intended to solve discrete logarithm problems

over algebraic curves efficiently.

4.2.4. GPU Experiments with Small Genus

In our final series of experiments, we use the implementation described in Section 4.1.2 for generating the general-purpose relations on hyperelliptic curves of small genus over prime fields of odd characteristic. To compare the results with those of Section 4.2.3, most experiments handle curves with a 120-bit Jacobian of genus between 4 and 8. Additionally, we tested a curve of genus 8 with a 182-bit Jacobian. For this curve, the Jacobian is of similar size to the one presented in Section 4.2.2.

Besides the experiments with Algorithm 3.3.1, we also compared the result to the ordinary large-prime collecting algorithm. The only difference of the two implementations is the different tolerance limit and the way the implementation processes large-prime relations. For both implementations and all curves, we used factor base sizes of $q^{\frac{2g-1}{2g+1}+\frac{2(\log_q(g-1)!-\log_q g)}{2g+1}}$ – which is the approximate optimum for single-large-prime algorithms – and $q^{\frac{g-1}{g}+\log_q g}$ – which is the theoretic optimal for Algorithm 3.3.1. We provide the curve equations and more detailed results in Appendix C.3. For the timings, we calculated exactly $\frac{\#\mathcal{F}}{2}$ relations. This is a complete set of relations due to the hyperelliptic involution.

$q = \#\mathbb{F}_p$	Genus	$\#\mathcal{F}$	Sieve Polynomials	Full Relations	Single-Large-Prime Relations	Computation Time
32,771	8	20,992	1,562,112	2,675	7,821 (75 %)	3m 6s
32,771	8	11,776	84,763,264	742	5,011 (87%)	2h 47m 21s
144,439	7	50,142	4,768,832	3,626	21,634 (85 %)	9m 51s
144,439	7	26,684	245,313,216	1,237	12,084 (90 %)	8h 13m 29s
1,048,627	6	196,820	5,936,800	10,170	88,340 (89 %)	17m 49s
1,048,627	6	82,376	652,891,008	2,503	38,514 (93 %)	1d 1h 59m
16,777,289	5	1,084,416	20,356,000	34,336	507,623 (93 %)	9h 49m 45s
16,777,289	5	340,192	3,566,849,856	5,757	164,456 (96 %)	25d 22h 44s
1,073,741,651	4	11,565,568	147,736,344	191,873	5,590,880 (96 %)	33d 2h 47m

Table 4.10.: Graphics Card Implementation Results for the Single-Large-Prime Algorithm

For the genus 8 curve, we observe that the GPU implementation requires about 3 minutes and 6 seconds for computing all necessary relations with the single-large-prime method for a factor base size of about $20,000$ elements. This is 73.75 times faster than the relation computation of a similar-sized curve with same factor base size using the CPU implementation of the single-large-prime algorithm as given in table C.8. It should be noted that the relation collection in the CPU example used only a single core. Using an entire node, the speedup is only a factor of 3.07.

Overall, the genus seven and eight curves are not well-suited to GPU hardware because the factor base and sieve array sizes are relatively small compared to the number of computational cores on the GPU. This makes it difficult to hide memory latency effectively. Consequently, we observe only a slight performance drop from the genus eight to the genus six curve. For the lower genera, there is an increase in problem complexity that aligns with theoretical predictions on reduced smoothness probabilities. For the smaller factor base sizes that are theoretically ideal for Algorithm 3.3.1, we observe a larger decrease in smoothness probability, resulting in a significant increase in runtime. Due to this and limitations in computing time on our test cluster, we had to omit experiments with smaller factor base sizes for the genus four curve.

For the new prime graph algorithm, we had to experiment with the factor c of Algorithm

3.3.1. This determines the exact number of single- and double-large-prime relations created before the relation combination algorithm starts. In theory, there are $q - \mathcal{F}$ divisors with $\frac{q-\mathcal{F}}{2}$ different norms that can be large-prime divisors. Due to the hyperelliptic involution, we can build the graph from the norms, since the two divisors of the same norm are only inverses in the Jacobian.

In our first experiments, we tested $q - \#\mathcal{F}$ as well as exactly q double-large-prime relations as targets for the search phase. Note that for the lower genus curves, the factor base is only a small fraction of all degree one divisors, and therefore both values are close. For the genus eight curve, these choices did not fit well to the problem. With the first setting, we collected insufficiently many double-large-prime relations to generate enough relations from the collected elements afterwards, although the graph was fully connected. When up to q double-large-prime relations were collected instead, the algorithm did not even enter the combination phase, because the single-large-prime relations were already enough to create all required relations.

$q = \#\mathbb{F}_p$	Genus	$\#\mathcal{F}$	Sieve Polynomials	Full Relations	Single-Large-Prime Relations	Relations from LP Graph	Computation Time
32,771	8	20,992	796,160	1,280	2,693 (25 %)	6,523 (62 %)	1m 52s
32,771	8	11,506	22,981,760	201	525 (8 %)	5,162 (87 %)	47m 39s
144,439	7	50,521	1,395,136	1,032	2,367 (9 %)	21,672 (86 %)	3m 17s
144,439	7	26,642	46,912,512	234	554 (4 %)	12,554(94 %)	1h 42m 9s
1,048,627	6	197,020	1,316,128	2,222	5,236 (5 %)	90,952 (92 %)	4m 49s
1,048,627	6	82,035	88,501,088	328	800 (1 %)	40,060 (97 %)	3h 43m 39s
16,777,289	5	1,083,918	2,955,744	5,032	12,039 (2 %)	525,137 (96 %)	1h 27m 51s
16,777,289	5	340,192	289,140,032	448	1,289 (0.76 %)	168,503 (98 %)	2d 3h 50m
1,073,741,651	4	11,565,506	9,685,328	12,674	25,812 (0.45 %)	5,744,298 (99 %)	2d 6h 13m
1,073,741,651	4	2,173,682	1,448,143,728	451	1,625 (0.15 %)	1,084,900 (99 %)	63d 20h 45m

Table 4.11.: Graphics Card Computing Results for Algorithm 3.3.1

Finally, it turned out that averaging the two values to $q - \frac{\#\mathcal{F}}{2}$ was a suitable choice for all our testing curves. Therefore, we adjusted c of Algorithm 3.3.1 so that the algorithms collect exactly $q - \frac{\#\mathcal{F}}{2}$ double-large-prime relations during the search phase. Since the smoothness probability does not change during the relation search, this parametrization

made the overall runtime predictable once we found a small percentage of all double-large-prime relations.

The new algorithm has similar scaling properties as the single-large-prime algorithm. Between the genus 8 and the genus 6 curve, the timing barely increases with respect to the theoretic algorithmic complexity. Starting with the genus 5 curve, the timing then follows the theoretic estimates. For the problems with reduced factor base size, the new algorithm has better scaling properties compared to the single-large-prime algorithm. In theory, the decrease of smoothness probability shall be $\left(\frac{\#\mathcal{F}_{Large}}{\#\mathcal{F}_{Small}}\right)^{g}$ for the single-large-prime algorithm and $\left(\frac{\#\mathcal{F}_{Large}}{\#\mathcal{F}_{Small}}\right)^{g-1}$ for the large-prime graph algorithm. These estimates are well confirmed by our experiments. Note that a smaller factor base allows us to initialize and sieve more quickly. Therefore, on both implementations, a factor of $\frac{\#\mathcal{F}_{Large}}{\#\mathcal{F}_{Small}}$ is saved with respect to the timing.

Overall, we see that the new algorithm has more headroom for balancing the runtime with the linear algebra phase since the penalty for decreased factor base sizes — which is beneficial for the linear algebra phase — is reduced. Also, we see that the scaling towards theoretically harder discrete logarithm problems improves compared to the large-prime graph algorithm. Also, by evaluating table 4.12, we see that for the harder problems, the new algorithm is scaling better.

Genus	4		5		6		7		8	
Factor Base Size	SLP Opt	LPG Opt	SLP Opt	LPG Opt	SLP Opt	LPG Opt	SLP Opt	LPG Opt	SLP Opt	LPG Opt
Single LP Alg.	1,465.9 %	na.	671.3 %	1,201.7 %	369.9 %	697.1 %	300.0 %	483.1 %	166.7 %	351.2 %
LP Graph Alg.	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %	100 %

Table 4.12.: Relative Computation Duration normed to Large-Prime Graph Algorithm

The large-prime graph algorithm generates relations with more divisors involved compared to the single-large-prime variant. Therefore, we provide the average weight for each matrix column in Table 4.13.

Genus	4		5		6		7		8	
Factor Base Size	SLP Opt	LPG Opt	SLP Opt	LPG Opt	SLP Opt	LPG Opt	SLP Opt	LPG Opt	SLP Opt	LPG Opt
Single LP Alg.	15.60	na.	18.66	18.92	20.97	21.86	22.95	24.03	23.82	25.87
LP Graph Alg.	44.34	63.28	44.29	60.43	49.56	62.35	54.33	61.32	64.47	85.01

Table 4.13.: Column Weights of tested Large-Prime Algorithms

We observe that the single-large-prime algorithm has a low sensitivity to the actual size of the factor base as well as to the genus of the curve with regard to the average row weight. One reason is that for a lower factor base size, the created relations have exactly the same structure with the only change being that the proportion of relations arising from large-prime relations increases. Especially for the lower genera, the proportion is already large for the larger factor base size. Therefore, the weight of the created relations has a low sensitivity. We also see that the weight of the created relations is almost constant between genus six and eight. The primary reason, again, is the number of full relations involved. For genus eight, the factor base size almost equals the size of the potential large-prime divisors. Therefore, the proportion of relations combined from single-large-prime relations is lower, and the weight remains almost constant, although the weight of a single relation scales linearly with g.

In contrast for the large-prime graph algorithm, there is a noticeable difference between the larger and the smaller factor bases in terms of the relation weight, although the number of relations generated from the graph barely differs. This is because, for the smaller factor bases, fewer single-large-prime relations are found. Therefore, the depth of the graph becomes larger until enough relations are found. For lower genera, we see that the number of divisors per relation is often significantly fewer than the number of factor base elements corresponding to isolated large-prime relations. This is because, even for the genus four curve, the self-initialization step utilizes only the first 200 divisors of the factor base. Consequently, the final generated relations often contain these divisors with multiplicities, especially for lower genera, where more double-large-prime relations are used to create one full relation.

In addition to experiments with a constant Jacobian magnitude, we generated a genus eight curve over $\mathbb{F}_{8388673}$, where 8388673 is the smallest prime greater than 2^{23}. We tested

relation generation for this curve with two different factor base sizes. As before, the smaller factor base was optimized for the large-prime graph algorithm, while the larger base, set to 3,700,000, allowed us to compare timings with the CPU implementation tested in Section 4.2.2.

$\#\mathcal{F}$	Sieve Polynomials	Full Relations	Single-Large-Prime Relations	Relations from LP Graph	Computation Time
3,696,584	6,513,664	94,404	212,400 (11 %)	1,541,488 (83 %)	8h 41m 10s
1,442,046	2,852,628,352	8,752	19963 (2 %)	692,308 (96 %)	66d 21h

Table 4.14.: Graphics Card Computing Results for a Curve of Genus 8 over $\mathbb{F}_{8388673}$

We provide the timings and further details in Table 4.14. We see that the combination of the graphics card and the large-prime graph algorithm managed to compute all required general-purpose relations about 1.126 times faster than the CPU implementation running on a single CPU core. Unfortunately, we cannot measure the effect of different implementations precisely, because the arithmetic of extension fields of $\mathbb{F}_2$, as on the CPU, and the arithmetic of odd characteristic prime fields as on the GPU, differ by nature.

By using the single-large-prime algorithm instead of the large-prime graph algorithm, we would have to compute about three times more relations. Therefore, the adjusted advantage of the graphics card over a single CPU core is rather 375 times the CPU performance. Also, the CPU has multiple cores, and therefore, we can conclude that for the given setting, a single graphics card offers approximately the same performance as 15 full cluster nodes with 24 CPU cores each. Our GPU implementation does not rely on special features of the Tesla P100 graphics card. Thus, we expect a similar performance on high-end customer cards. Our results indicate that for further attempts to solve hard discrete logarithm problems over algebraic curves, a graphics-card-optimized implementation should be considered with respect to cost efficiency.

As discussed in Section 4.2.2, the factor base size and the duration of solving the linear algebra problem are serious issues. With a size of about 39% of the larger example, the second test reduces the space and timing requirements by a factor of 6.5. Due to the large

genus, the theoretic computational effort for creating enough double-large-prime relations increases by a factor of $2.5^6 \approx 244$.

4.3. Discussion of Results

Having analyzed the individual experimental results in the preceding section, we now turn our focus to the broader implications of our findings for large-scale discrete logarithm problems (DLPs) over algebraic curves.

Feasibility studies for the discrete logarithm problem are frequently framed within a context of limited resources. A standard benchmark in the literature assumes a fixed timeframe—typically one year—and a budget of $1,000,000. Based on the architecture and costs of the computer cluster utilized in our experiments, this budget translates to a cluster of approximately 60 computing nodes, each equipped with a graphics card of the specifications detailed in Section 4.1.1.1. In this calculation, we allocate half of the total budget to hardware procurement, with the remainder reserved for operational costs, primarily electricity.

We begin by examining the feasibility of solving DLPs on imaginary model hyperelliptic curves. As established in Theorem 2.5.1 and verified by the scaling experiments in Section 4.2.3, these problems exhibit uniform complexity once the condition $\sqrt{\frac{g+1}{2\ln q}}\sqrt{\ln\ln q^{g+1}} \geq 1$ is met. If the genus is smaller, the complexity class of the algorithm is less than or equal to $O\left(q^2\right)$, depending on the chosen large-prime method. In the latter setting, the memory requirement is $O\left(q\right)$ for the matrix in the linear algebra phase, as well as for the large-prime graph of Algorithm 3.3.1.

For the linear algebra phase, we utilized the high-performance implementation integrated into CADO-NFS [57]. As demonstrated in Section 4.2.2 for a genus 8 curve with a Jacobian of approximately $2^{23.8} = 2^{184}$ elements, this implementation processed a factor base of 3,700,000 elements—with an average relation weight of 23.8 non-zero coefficients—in approximately 16.46 node-days.

The most efficient relation collection was achieved using our GPU-based implementation, also evaluated in Section 4.2.4 for a genus 8 curve with 2^184 elements in its Jacobian.

For a factor base of 3,696,584 elements, the large-prime graph algorithm required 8 hours and 41 minutes. By contrast, a smaller factor base of 1,442,046 elements required over 67 days, illustrating the critical nature of factor base optimization.

Balancing for a genus 8 curve, an extrapolation will give us predictions for low- and higher-genus curves. On the one hand, we can solve the DLP with our low-genus implementation. On the other hand, for $2^{150} \leq \#Jac(C) \leq 2^{500}$, we see that the above condition $\sqrt{\frac{g+1}{2\ln q}}\sqrt{\ln\ln q^g} \geq 1$ is satisfied from genus eight onwards. Therefore, the balancing gives the feasible sizes as well.

From our experiments, we can calculate that for our implementations and for our hardware, the optimal factor base size would be about 1,900,000 elements. Then the two phases are fully balanced and require about 12.5 days. Note that we had to consider the higher relation weight of the large-prime graph algorithm, as we detailed in Table 4.13.

Based on the assumptions regarding available time and resources, we can estimate that we are able to handle problems that are about 935 times harder than the ones in our experiments. Due to the quadratic increase of the algorithm complexity with rising q, we can calculate that the maximal q we can handle for a genus 8 curve within our self-given constraints is approximately 2^{28}. Since the complexity for larger genera remains constant, based on the experiments in Section 4.2.3, we conclude that any hyperelliptic curve discrete logarithm problem with genus $g \geq 8$ and $\#JacC \leq 2^{224}$ is feasible within our restrictions.

For hyperelliptic curves of lower genus, we can use larger values of q, but the size of the Jacobian will decrease due to the exponential complexity for low-genus curves. Overall, we will balance $q^{2^{\frac{g-1}{g}}} = 2^{28\cdot 2\cdot\frac{7}{8}}$ for our estimates. Table 4.15 provides our full feasibility estimates for hyperelliptic curves.

Genus	2	3	4	5	6	7	≥ 8
Maximal q	$\approx 2^{49}$	$\approx 2^{36.75}$	$\approx 2^{32.67}$	$\approx 2^{30.63}$	$\approx 2^{29.4}$	$\approx 2^{28.58}$	$\leq 2^{28}$
Size of Jacobian	$\approx 2^{98}$	$\approx 2^{110.25}$	$\approx 2^{130.66}$	$\approx 2^{153.13}$	$\approx 2^{176.4}$	$\approx 2^{200.08}$	$\approx 2^{224}$

Table 4.15.: Prediction of the Feasibility of Hyperelliptic Discrete Logarithm Problems

For non-hyperelliptic curves, the feasibility depends on the degree η of elements that are associated with the sieve arrays and the complexity of the descent of the special elements

described in Section 3.4.2. As described in the discussion of Theorem 3.4.8, for general curves, we cannot assume that achieving a complexity of $L_{q^g}\left[\frac{1}{3}, O(1)\right]$ is possible.

The genus eight hyperelliptic example corresponds to a value of $\eta = 9$ that is feasible with our methods and restrictions for $q \leq 2^{28}$. The same η can be reached on $C_{5,11}$ curves with genus $g = \frac{(n-1)(d-1)}{2} = 20$ without falling into the descent restriction. These curves have a Jacobian of size 2^{560}.

We can conclude that any discrete logarithm problem for $C_{n,d}$ curves with $Jac(C) \leq 2^{560}$ and $n \geq g^{\frac{1}{3}} \geq 5$ is feasible with the methods described in this work. Higher values of n might lead to lower values of η in the relation search, but curves with higher n have the same complexity for the descent method as the $C_{5,11}$ curve. For lower degrees of n, the feasibility will scale between this value and those of hyperelliptic curves. We describe this scaling in Table 4.15.

Outlook

In this thesis, we demonstrate the capacities of specialized algorithms and implementations. Our focus was on the generation of relations in the Jacobian of a $C_{n,d}$ curve over a given factor base. Some of these techniques can be adapted to other algorithms as well. In particular, the sieve-step algorithm may be useful for any polynomial-based sieve algorithm, such as the function field sieve [2], or for structure computations in the class group of function fields. However, naturally focusing on a specific part of the algorithm also leaves open questions that may be worthy of future investigation.

One promising area of exploration is whether a specialization of linear algebra routines could similarly improve overall complexity. For instance, in our experiments, we observed that even with the large-prime graph algorithm (Algorithm 3.3.1), all matrix coefficients had absolute values less than or equal to 4. This observation suggests that in the Wiedemann algorithm (Algorithm A.4.11), modular multiplication could potentially be replaced with a small number of additions and subtractions, leading to a more efficient approach.

We assume that limiting combined relations — specifically those where multiplicity of factor base divisors is bounded — does not significantly increase the search effort. Under this assumption, the following conjecture arises. By adjusting the parameter r in Theorem 3.3.7, we can provide a straightforward proof, effectively balancing the overall complexity.

Conjecture 1. *Let C be a a $C_{n,d}$ curve over $\mathbb{F}_q$ and let $\mathcal{F}$ be a factor base of degree one divisors of C of size q^r with $r < 1$. Furthermore, assume $\log_q(\eta - 2) < \frac{1}{\eta - 1}$.*

Then we conjecture that there is an algorithm for solving the discrete logarithm problem on the curve with an expected time complexity of

$$O\left(q^{2-\frac{2}{\eta-1}} \cdot (\ln q)^2 \cdot (\eta-2)^2 \cdot g^{1-\frac{1}{\eta-1}}\right)$$

where $g = \frac{(n-1)(d-1)}{2}$ *is the genus of the curve.*

In particular, when the curve is an imaginary model hyperelliptic curve, the algorithm has an expected runtime of

$$O\left(q^{2-\frac{2}{g}} \cdot (\ln q)^2 \cdot g^{3-\frac{1}{g}}\right).$$

Beyond theoretical advancements in the linear algebra phase, a new implementation that provides a constant speedup could extend the boundaries of feasible discrete logarithm computations within a fixed timeframe. Our results demonstrate that our algorithms benefit significantly from highly parallel hardware, such as graphics cards. Consequently, a promising path for future research is the acceleration of large-scale linear algebra computations using highly parallel GPU-based architectures.

A second aspect, which we only briefly addressed, concerns the creation of the two special relations. By selecting elements with smaller norms from the ideals to be decomposed, it seems feasible to maintain a low complexity, ensuring that generating these relations does not become a bottleneck in the entire DLP algorithm. However, for the approach used in this thesis we still can only prove a minimum complexity of $L_{q^g}\left[\frac{1}{3} + o(1), O(1)\right]$ with the methods provided in this thesis. As we have seen with recent advancements for discrete logarithm problems in finite fields [31] and [3], using additional structures might improve on our results and improve the bound.

As our methods already suggest that for $n \approx d$ we can compute the general relations sufficiently fast, we state the following conjecture:

Conjecture 2. *Let* $q = p^k$ *for a prime* p *and* $k \in \mathbb{N}$. *There exists an infinite family of curves over* $\mathbb{F}_q$ *for which the discrete logarithm problem is solvable with an expected subexponential complexity of* $L_{q^g}\left[\frac{1}{4} + o(1), O(1)\right]$ *using a sieving-based algorithm.*

From an application perspective, several challenges remain. As highlighted in Section 3.3.3, the effectiveness of the new methods for genus-two hyperelliptic curves is still uncertain. One crucial aspect to investigate is the actual size of the factor base, as the duration of the relation search appears to be mostly independent of the factor base size at such low genus. Additionally, the previously mentioned potential improvements in linear algebra

could contribute to creating an algorithm that surpasses the generic approach for these curves.

Thanks to the flexibility in factor base sizing, we conjecture that the new methods could give rise to an algorithm that offers near-perfect parallelism across multiple computing systems.

Conjecture 3. *Let C be a hyperelliptic curve of genus two or a non-hyperelliptic curve of genus three. Then there is an algorithm that solves any given instance of the discrete logarithm problem on C in expected time*

$$\frac{1}{m} O\left(q \cdot \ln q \cdot \ln \ln q\right),$$

where $m < q^{\frac{1}{2}}$ is the number of machines available for solving the problem.

Finally, we also hope that this work gives rise to new examples of elliptic curve discrete logarithm computations. While the methods of this thesis are not directly applicable to elliptic curves, they are applicable to low-degree non-hyperelliptic curves. Assume that d_γ and d_β of Theorem 1.2.61 are both greater than one. Then, the resulting curve of the Weil descent attack will be non-hyperelliptic. However, there is no reason why these curves must have a large genus. Therefore, we can expect to find elliptic curves that are vulnerable to a non-hyperelliptic Weil descent attack.

Conjecture 4. *There is a family of elliptic curves over $\mathbb{F}_q$ with $q = 2^n$ and n composite, such that*

a) any instance of the discrete logarithm problem on these curves is solvable in expected subexponential time complexity via a Weil descent attack.

b) the curves neither satisfy the conditions of Corollary 1.2.62 nor are they isogenous to any curve that satisfies these conditions.

A

Complexity Theory and Examples

A.1. Subexponential Complexity Theory

Subexponential complexity is a common concept in computational number theory. Prominent examples include the quadratic sieve and the number field sieve, both of which exhibit subexponential runtime. The primary purpose of these complexity classes is to describe smoothness probabilities, i.e., the probability of a random element factoring over a given set of elements with a known size. We will follow the notation and theorems of [22].

Definition A.1.1 (L-Function)**.** Let $n \in \mathbb{N}$, $\alpha \in [0, 1]$ and $\beta > 0$. Then the subexponential *L-function* is defined by

$$L_n[\alpha, \beta] := e^{\beta \cdot (\ln n)^{\alpha} (\ln \ln n)^{1-\alpha}}.$$

An algorithm with proven time-complexity of $L_n[\alpha, \beta]$ with α, β fixed and $0 < \alpha < 1$ will be said to have *subexponential runtime*.

Lemma A.1.2. *Let* $\alpha \in [0, 1]$ *and* $\beta, \beta' > 0$. *Then*

$$\begin{aligned} L_n[0, \beta] &= (\ln n)^\beta \\ L_n[1, \beta] &= n^\beta \\ L_n[\alpha, \beta] + L_n[\alpha, \beta'] &= L_n[\alpha, max\{\beta, \beta'\} + o(1)] \\ L_n[\alpha, \beta] \cdot L_n[\alpha, \beta'] &= L_n[\alpha, \beta + \beta'] \\ L_n[0, \beta] &= L_n[\alpha, o(1)] \textit{for any } \alpha > 0. \end{aligned}$$

The listed properties are due to the properties of the exponential function and thus trivial to prove. In the following example, we will see that the subexponential function can be used to scale more freely between exponential and polynomial complexity classes. Note that any exponential term with a fixed exponent will asymptotically exceed any subexponential expression, as long as $\alpha < 1$. The same is valid for subexponential and polynomial classification with $\alpha > 0$.

We now consider how a change of basis affects the complexity analysis in the subexponential expression. If the reference is an exponential expression of some n, we only change the secondary argument of the L function.

Lemma A.1.3. *Let x be exponential in some quantity n, i.e.,* $x = n^\beta$. *Then*

$$L_x[\gamma, \delta] = L_n[\gamma, \delta\beta^\gamma + o(1)],$$

where $o(1)$ *indicates a term converging to* 0 *for* $n \to \infty$.

Proof. We can prove the lemma by a direct calculation:

$$\begin{aligned} L_x[\gamma, \delta] &= e^{\delta(\ln x)^\gamma (\ln \ln x)^{1-\gamma}} &&= e^{\delta\left(\ln n^\beta\right)^\gamma \left(\ln \ln n^\beta\right)^{1-\gamma}} \\ &= e^{\delta(\beta \ln n)^\gamma (\ln \beta \ln n)^{1-\gamma}} &&= e^{\delta\beta^\gamma (\ln n)^\gamma (\ln \beta + \ln \ln n)^{1-\gamma}} \\ &= e^{\delta\beta^\gamma + o(1)(\ln n)^\gamma (\ln \ln n)^{1-\gamma}} &&= L_n[\gamma, \delta\beta^\gamma + o(1)]. \end{aligned}$$

□

On the other hand, when the reference value itself is subexponential, we prove that both arguments change.

Lemma A.1.4. *Let x be subexponential in the quantity n, i.e., $x = L_n[\alpha, \beta]$. Then*

$$L_x[\gamma, \delta] = L_n[\alpha\gamma, \delta\beta^\gamma + o(1)],$$

where $o(1)$ indicates a term converging to 0 for $n \to \infty$.

Proof. Again the assertion follows from a direct computation.

$$\begin{aligned}
L_x[\gamma, \delta] &= e^{\delta(\ln x)^\gamma (\ln\ln x)^{1-\gamma}} \\
&= e^{\delta(\ln L_n[\alpha,\beta])^\gamma (\ln\ln L_n(\alpha,\beta))^{1-\gamma}} \\
&= e^{\delta\left(\beta(\ln n)^\alpha (\ln\ln n)^{1-\alpha}\right)^\gamma \left(\ln\left(\beta(\ln n)^\alpha (\ln\ln n)^{1-\alpha}\right)\right)^{1-\gamma}} \\
&= e^{\delta\beta^\gamma (\ln n)^{\alpha\gamma} (\ln\ln n)^{(1-\alpha)\gamma} (\ln\beta + \alpha\ln\ln n + (1-\alpha)\ln\ln\ln n)^{1-\gamma}} \\
&= e^{(\delta\beta^\gamma + o(1))(\ln n)^{\alpha\gamma} (\ln\ln n)^{(1-\alpha)\gamma} (\alpha\ln\ln n)^{1-\gamma}} \\
&= e^{(\delta\beta^\gamma + o(1))(\ln n)^{\alpha\gamma} (\alpha\ln\ln n)^{1-\alpha\gamma}} = L_n[\alpha\gamma, \delta\beta^\gamma + o(1)].
\end{aligned}$$

□

Finally, we will give some examples of the bit complexity of subexponential algorithms with respect to a reference value of n ranging from 2^{80} to 2^{200}.

Example A.1.5. *The table lists $L_n[\alpha, \beta]$ for $\alpha \in \left\{\frac{1}{3}, \frac{1}{2}\right\}$ and $\beta \in \left\{1, \sqrt{2}, 2\right\}$, which are the most common cases. For comparison we add linear and quadratic polynomial bit-complexity as well as square root exponential bit-complexity.*

n	$\ln n$	$(\ln n)^2$	$L_n\left[\frac{1}{3}, 1\right]$	$L_n\left[\frac{1}{3}, \sqrt{2}\right]$	$L_n\left[\frac{1}{3}, 2\right]$	$L_n\left[\frac{1}{2}, 1\right]$	$L_n\left[\frac{1}{2}, \sqrt{2}\right]$	$L_n\left[\frac{1}{2}, 2\right]$	$\sqrt{n}$
2^{80}	*56*	*3136*	$2^{13.9}$	$2^{19.6}$	$2^{27.7}$	$2^{21.5}$	$2^{30.4}$	2^{43}	2^{40}
2^{100}	*69*	*4761*	$2^{15.5}$	2^{22}	2^{31}	$2^{24.7}$	$2^{34.9}$	$2^{49.5}$	2^{50}
2^{120}	*83*	*6889*	2^{17}	2^{24}	$2^{33.9}$	$2^{27.6}$	$2^{39.1}$	$2^{55.3}$	2^{60}
2^{140}	*97*	*9409*	$2^{18.2}$	$2^{25.8}$	$2^{36.5}$	$2^{30.3}$	2^{43}	$2^{60.7}$	2^{70}

n	$\ln n$	$(\ln n)^2$	$L_n\left[\frac{1}{3}, 1\right]$	$L_n\left[\frac{1}{3}, \sqrt{2}\right]$	$L_n\left[\frac{1}{3}, 2\right]$	$L_n\left[\frac{1}{2}, 1\right]$	$L_n\left[\frac{1}{2}, \sqrt{2}\right]$	$L_n\left[\frac{1}{2}, 2\right]$	$\sqrt{n}$
2^{160}	*111*	$2^{13.5}$	$2^{19.4}$	$2^{27.5}$	$2^{38.9}$	$2^{32.9}$	$2^{46.6}$	$2^{65.9}$	2^{80}
2^{180}	*125*	$2^{13.9}$	$2^{20.5}$	$2^{29.1}$	$2^{41.2}$	$2^{35.4}$	$2^{50.1}$	$2^{70.8}$	2^{90}
2^{200}	*139*	$2^{14.2}$	$2^{21.6}$	$2^{30.6}$	$2^{43.3}$	$2^{37.7}$	$2^{53.3}$	$2^{75.4}$	2^{100}

Table A.1.: Bit Complexities for Important Complexity Classes

The complexity class $L_n\left[\frac{1}{3}, 2\right]$ will yield a lower complexity than $L_n\left[\frac{1}{2}, 1\right]$ for large n, but performs worse for $n < 2^{200}$. This applies, for example, to integer factorization algorithms. Concretely, the general number field sieve [39] has complexity class $L_n\left[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right] \approx L_n\left[\frac{1}{3}, 1.92\right]$, which is slower for small input sizes than the multi-polynomial quadratic sieve with runtime $L_n\left[\frac{1}{2}, 1\right]$. Therefore, the number field sieve is barely used for small and medium-sized inputs, where the quadratic sieve [50] and the elliptic curve factorization [40] are more efficient.

The overall bit-complexity that is assumed to be a hard problem or to deliver a certain measurement of security in cryptographic applications increases over time. Until 2017, a bit-security of more than 80 bits could still be considered to be secure. For future approximations we refer to [6].

A.2. Results from Probability Theory

We use the following theorems for some of the estimations in the expected time computations. In particular, we deal with the question of how many random elements from a finite set we have to draw until one can expect a collision. This is the basis for Pollard's rho algorithm 1.1.8 as well as for the runtime estimate for the presented large-prime algorithms.

Theorem A.2.1 (Extended Birthday Paradox)**.** *Given a finite set S of size n, let $s_1, \ldots, s_l$ be randomly chosen elements from S. Then the probability $P(n,l)$ that there are $1 \leq i,j \leq l$ with $i \neq j$ such that $s_i = s_j$ is approximately*

$$P(n,l) \approx 1 - e^{-\frac{l(l-1)}{2n}}.$$

Conversely, the least number d to achieve a probability of at least p equals

$$l \approx \sqrt{2n \ln(\frac{1}{1-p})}.$$

Proof. We investigate the complementary probability for a collision, i.e., that all s_i are different from the previously found ones. The probability $\widetilde{P}(d,n)$ is precisely given by

$$\begin{aligned}\widetilde{P}(n,l) &= \prod_{i=1}^{l}(1-\frac{i}{n}) \\ &\leq \prod_{i=1}^{l} e^{-\frac{i}{n}}, \text{ since } (1-x) \leq e^{-x} \\ &= e^{-\frac{l(l-1)}{2n}}.\end{aligned}$$

Using $P(n,l) = 1 - \widetilde{P}(n,l)$ and the properties of the exponential function completes the proof. □

We see that the probability of a collision is already pretty high if the number of randomly chosen elements is in $O(\sqrt{l})$. Conversely, we can estimate the number of collisions in a larger list of randomly selected elements.

Theorem A.2.2 (Number of Expected Collisions)**.** *Given a finite set S of size n, let $s_1, \ldots, s_l$ be randomly chosen elements from S. Then the expected number of duplicates in the list S is about $\frac{l^2}{n}$. Conversely, we have to draw at least $l = \sqrt{r \cdot n}$ to ensure an expected number of r duplicates.*

Proof. Denote the list $s_1, \ldots, s_l$ by L and let $a \in L$. Then for each $b \in L$ the probability that $a \neq b$ is $\left(1 - \frac{1}{n}\right)$. Therefore, the probability that no other element of the list S equals a is given by $\left(1 - \frac{1}{n}\right)^{l-1}$.

Consequently, the expected number of elements in S that are not equal to any other element of S is given by $l \cdot \left(1 - \frac{1}{n}\right)^{l-1}$. Also, the expected number of elements that are equal to another element is $l \cdot (1 - \left(1 - \frac{1}{n}\right)^{l-1})$.

This formula can be easily approximated by computing

$$l \cdot \left(1 - \left(1 - \frac{1}{n}\right)^{l-1}\right) = \frac{l(l-1)}{n} - \frac{l(l-1)(l-2)}{n^2} + O\left(\frac{1}{n^3}\right) \approx \frac{l^2}{n}.$$

We conclude the remaining assertion by rearranging the equation $r = \frac{l^2}{n}$. □

A.3. Polynomial Factoring and Root Finding

Assume $F \mid K$ is a function field arising from a plane smooth algebraic curve given by $f(x, y) = 0$. As described in the previous sections, it is possible to represent the divisor classes having at least one affine representative in $Pic(F)$ by pairs of polynomials $(a, b), a, b \in K[x]$ such that b is a root of f modulo a. In this appendix we will investigate the opportunities to compute such pairs a and b. Besides the motivation for building divisors in general, the algorithms we describe in this section are also utilized by the discrete logarithm solving algorithms.

For a composite a, possible choices of b can be computed by using the Chinese remainder theorem on the prime divisors of a and the corresponding roots. Therefore, it is sufficient to restrict ourselves to the question of how to decompose polynomials into their prime factors and how to compute the roots for prime divisors. Since solving these algorithmic problems is more convenient over finite fields, we will restrict ourselves to this case.

We will start with a strategy to factor polynomials of arbitrary degree. Later, we will then mention two more efficient algorithms that allow computing square roots of elements in finite fields. This is of interest, since the root computation for quadratic polynomials can be transformed into this kind of problem. Furthermore, solving quadratic equations is the most common case when working with hyperelliptic Jacobians.

Note that for divisors we can make a base field change for computing the roots of a divisor norm. Assume there is a divisor (a, b) with $\deg a > 1$. Then b is equal to the root of the function field defining polynomial over the extension field $\mathbb{F}_{q^{\deg n}}$. Therefore, for our purpose, we can restrict ourselves to root-finding algorithms only.

A.3.1. Computing Roots of Polynomials of Arbitrary Degree

Most factorization algorithms for polynomials over finite fields require additional conditions regarding the degree of potential factors. The proposed algorithm requires ensuring that all divisors of the polynomial to be factored are of degree one. In our case, it is simple to separate the degree-one factors from prime factors of higher degree.

Lemma A.3.1. *Let $f \in \mathbb{F}_q[y]$ be a monic polynomial. Then the polynomial $gcd(f, y^q - y)$ is the product of all degree-one prime polynomials of $\mathbb{F}_q[y]$ dividing f.*

Proof. First of all, it is immediate that $y^q - y$ is the product of all degree-one polynomials due to Fermat's lemma. Therefore, any degree one polynomial dividing f also divides $gcd(f, y^q - y)$, and $gcd(f, y^q - y)$ has no more factors up to signs, since $\mathbb{F}_q[y]$ is a unique factorization domain. □

Now we find linear factors of a polynomial by using a simplified variant of the Cantor-Zassenhaus algorithm [9]. We will state it for odd characteristic first. We will discuss a generalization for $2 \mid q$ later in Algorithm A.3.16 at the end of Section A.3.3.

Algorithm A.3.2 (Cantor-Zassenhaus)**.**

Input: A polynomial $f \in \mathbb{F}_q[y]$, q odd prime power of degree greater than one with all factors of f being of degree one.

Output: Three polynomials f_{-1}, f_0, f_1 satisfying $f = \prod_{i=-1}^{1} f_i$ with at least two f_i being nontrivial.

1: Select $g \in_R \{h \in \mathbb{F}_q[y] \mid h \notin \mathbb{F}_q,\ \deg(h) < \deg(f)\}$
2: $g \leftarrow g^{\frac{q-1}{2}}$
3: Compute $f_i \leftarrow \gcd(f, g + i)$ for all $-1 \le i \le 1$.
4: **if** $(1 \neq f_{-1} \neq f) \vee (1 \neq f_0 \neq f)$ **then**
5: **return** f_{-1}, f_0, f_1
6: **else**
7: **Goto** 1.
8: **end if**

Lemma A.3.3. *Algorithm A.3.2 is correct.*

Proof. Let $p \in \mathbb{F}_q[y]$ with $p \mid f$. Then the residue class field $\mathbb{F}_q[y]/(p)$ is isomorphic to $\mathbb{F}_q$, and thus for any $g \in \mathbb{F}_q[y]/(p)$, we know that $g^{q-1} = 1$ or $g^q = 0$. Therefore, $g^{\frac{q-1}{2}} \in \{-1, 0, 1\}$.

We can conclude that $f_{-1}f_0f_1 = f$ because each factor will belong to one of the three groups, and the algorithm stops as soon as $g^{\frac{q-1}{2}}$ does not have the same residue for all $p \mid f$. Thus $f_{-1}f_0f_1 = f$ is a nontrivial decomposition. □

By using Algorithm A.3.2 iteratively we can obtain all roots of a given polynomial in $\mathbb{F}_q[Y]$. This is of particular importance for computing all divisors lying over the same $x-$coordinate of the rational function field. For the special class of hyperelliptic curves and equations of degree two, we will derive separate methods that work without continuously trying different random polynomials and trial gcd calculations.

A.3.2. Finding Roots of Quadratic Polynomials in Odd Characteristic

In odd characteristic, it is possible to compute the roots of every non-degenerate quadratic polynomial by taking square roots in the finite field by the following lemma.

Lemma A.3.4. *Let q be an odd prime power and $f = y^2 + by + c \in \mathbb{F}_q[y]$. Then computing the roots of f – if any exist – is equivalent to solving the equation*

$$n^2 = \frac{b^2}{4} - c \text{ over } \mathbb{F}_q.$$

Proof. By the coordinate substitution $y' := y - \frac{b}{2}$ we see that

$$y'^2 - \left(\frac{b^2}{4} - c\right) = y^2 + by + c.$$

Thus, solving the equation $n^2 = \frac{b^2}{4} - c$ is equivalent to the root computation using the inverse coordinate transformation. The transformation is always possible, since $2 \in \mathbb{F}_q^*$. □

Due to this simplification we are able to compute square roots of elements in $\mathbb{F}_q$ efficiently.

Lemma A.3.5. *Let q be an integer satisfying $q \equiv 3 \pmod 4$ and $c \in \mathbb{F}_q^*$, c a square in $\mathbb{F}_q$. Then the equation $n^2 = c,\ n \in \mathbb{F}_q$ has the only solutions $n = \pm c^{\frac{q+1}{4}}$.*

Proof. c is a square in $\mathbb{F}_q$. Therefore, we know that it is represented by an odd number in $\mathbb{F}_q^*$. Since $q - 1 = 2r$ with odd r, we conclude that $\operatorname{ord} c \mid r$. Thus $c^{\frac{q-1}{2}} = 1$, and it follows that

$$c = c^{\frac{q-1}{2}+1} = c^{\frac{q+1}{2}} = \left(c^{\frac{q+1}{4}}\right)^2$$

which completes the proof. □

If $q \not\equiv 3 \pmod 4$, it follows that $4 \nmid q+1$ and therefore the construction of roots is more difficult. We will state the following theorem that will help constructing the algorithm of Tonelli and Shanks [53] for solving the equation in these cases.

Theorem A.3.6. *Let $x, c \in \mathbb{F}_q^*$ with $q - 1 = r2^s$ such that $x^2 = cz$ for an element $z \in Syl_2(\mathbb{F}_q^*)$ with $\operatorname{ord}(z) = 2^k$ and $k < s$. Then there is another $w \in Syl_2(\mathbb{F}_q^*)$ that satisfies $(wx)^2 = yz'$ with $z' \in Syl_2(\mathbb{F}_q^*)$ and $\operatorname{ord}(z') < \operatorname{ord}(z)$.*

Proof. Since $\mathbb{F}_q^*$ is cyclic, we know that $Syl_2(\mathbb{F}_q^*)$ is also cyclic, and there exist elements of order equal to each divisor of 2^s. Therefore, we can select $w \in Syl_2(\mathbb{F}_q^*)$ such that $\operatorname{ord}(w) = 2^{k+1}$.

With this choice, we see that

$$(wx)^2 = w^2x^2 = cw^2z = cz'$$

with $z' := w^2z$. Since $\operatorname{ord}(w^2) = 2^k = \operatorname{ord}(z)$ we can calculate

$$(w^2z)^{2^{k-1}} = -1 \cdot -1 = 1$$

and thus $\operatorname{ord}(z') \leq 2^{k-1}$ which proves the theorem. □

Algorithm A.3.7 (Tonelli-Shanks Algorithm for Finite Fields)**.**

Input: A square element $c \in \mathbb{F}_q$ with q odd prime power.

Output: An element $x \in \mathbb{F}_q$ satisfying $x^2 = c$.

1: Factorize $q - 1 = r2^s$ with odd r.
2: Select $w \in_R \mathbb{F}_q$ that is a non-square, i.e., there is no $n \in \mathbb{F}_q : n^2 = w$
3: $w' \to w^r$.
4: $x \to c^{\frac{r+1}{2}}$
5: $z \to c^r$
6: **while** $z \neq 1$ **do**
7: Let $2^k = \text{ord}(z)$
8: $b \to w'^{s-k-1}$.
9: $x \to bx$
10: $z \to b^2 z$
11: **end while**
12: **return** x

Lemma A.3.8. *Algorithm A.3.7 is correct.*

Proof. First of all, we see that the elements x, z from the initialization satisfy

$$cz = cc^r = \left(c^{\frac{r+1}{2}}\right)^2 = x^2,$$

so this choice of x and z satisfy the first condition of Theorem A.3.6. Furthermore, since c is a square, we conclude $(c^r)^{2^{s-1}} = c^{r2^{s-1}} = n^{r2^s} = 1$ for any square-root n of c. So, the order of c^r is a power of two with exponent $k < s$. This completes the prerequisite conditions for Theorem A.3.6.

The choice of w is a random non-square element. This ensures $w' = w^r$ has order 2^s, and thus we can use w' to generate elements of an arbitrary power of 2 order in $Syl_2(\mathbb{F}_q)$.

With this element at hand, the algorithm uses Theorem A.3.6 iteratively, until the order of the error term z drops to zero, i.e., $z = 1$. Since the condition for Algorithm A.3.6 holds,

in each iteration the returned value is a root of c in $\mathbb{F}_q$. □

A.3.3. Finding Roots of Quadratic Polynomials in Characteristic 2

In characteristic 2, we have to pay more attention to the quadratic equation to solve, because the transformations presented in A.3.4 require two to be invertible. Thus, we replace the normal form as given in Lemma A.3.4.

Lemma A.3.9. *Let $c \in F_q$ with q a power of two. Then the equation $n^2 = c$ has the unique solution $n = c^{\frac{q}{2}}$ in $\mathbb{F}_q$.*

The lemma is trivial to prove, since $c^q = c$ for every element in $\mathbb{F}_q$, and squaring is an isomorphism in characteristic 2. We see that the degenerate case without a linear term is the special case in even characteristic.

Lemma A.3.10 (Normal Form in Characteristic 2)**.** *Let $b, c \in \mathbb{F}_q$, $b \neq 0$. Then solving the equation $n^2 + bn + c = 0$ is equivalent to solving the equation $n'^2 + n' + d = 0$ with $d := c \cdot b^{-2}$.*

Proof. First of all, we multiply the initial equation by $b^{-2} \neq 0$. This gives

$$\begin{aligned} 0 &= b^{-2}n^2 + b^{-1}n + c \cdot b^{-2} \\ &= n'^2 + n' + c \cdot b^{-2} \end{aligned}$$

for $n' := n \cdot b^{-1}$. This completes the proof. □

The equation introduced in Lemma A.3.10 is an *Artin-Schreier equation*. We will see that the *trace* of d determines the solvability of the Artin-Schreier equation in $\mathbb{F}_q$

Definition A.3.11 (Trace)**.** Let $d \in \mathbb{F}_q$ with $q = 2^k$. Then the *trace* of d is defined as

$$Tr(d) := d + d^2 + \ldots + d^{2^{k-2}} + d^{2^{k-1}}.$$

Lemma A.3.12 (Properties of the Trace)**.** *Let $x, y \in \mathbb{F}_q$, $q = 2^k$. Then the following properties of the trace hold:*

1. $Tr(x) \in \{0, 1\}$.
2. $Tr(x) = Tr(x^2)$.
3. *Tr is an additive homomorphism from $\mathbb{F}_q$ to $\mathbb{F}_2$, i.e.,* $Tr(x + y) = Tr(x) + Tr(y)$.

Proof. 1. By applying the formula $x^{2^k} = x$ in $\mathbb{F}_q$ we see that $Tr(x)^2 + Tr(x) = 0$ in $\mathbb{F}_q$ and thus $Tr(x)^2 = Tr(x)$. This implies directly $Tr(x) \in \mathbb{F}_2$.

2. Follows directly from the considerations made for (1) since $Tr(x^2) = Tr(x)^2$ since we work in characteristic 2.

3. Is immediate since the exponents are powers of two and continuous squaring is an additive homomorphism in characteristic 2.

□

Theorem A.3.13 ([10])**.** *Let $d \in \mathbb{F}_q$. Then the Artin-Schreier equation $n^2 + n + d = 0$ is solvable over $\mathbb{F}_q$ if and only if $Tr(d) = 0$.*

Proof. Let $n' \in \mathbb{F}_q$ be a solution of $n^2 + n + d = 0$. Then

$$Tr(d) = Tr(n'^2 + n') = Tr(n'^2) + Tr(n') = 2Tr(n') = 0$$

by using all properties from A.3.12.

Conversely, let n' be a solution of the quadratic equation over a quadratic extension field of $\mathbb{F}_q$, then

$$\begin{aligned} 0 = Tr(d) &= d + d^2 + \ldots + d^{2^{k-2}} + d^{2^{k-1}} \\ &= (n'^2 + n') + (n'^2 + n')^2 + \ldots + (n'^2 + n')^{2^{k-2}} + (n'^2 + n')^{2^{k-1}} \\ &= (n'^2 + n') + n'^4 + n'^2 + \ldots + (n'^{2^{k-1}} + n'^{2^{k-2}}) + (n'^{2^k} + n'^{2^{k-1}}) \\ &= n' + n'^{2^k}. \end{aligned}$$

Therefore, $n' = n'^{2^k}$ holds, which implies $n' \in \mathbb{F}_q$. □

By using the trace we can compute solutions of $n^2 + n + d = 0$ in an analogous way to the Tonelli-Shanks algorithm once we have ruled out the trivial cases.

Lemma A.3.14. *Let $d \in \mathbb{F}_q$, $Tr(d) = 0$ and $q = 2^k$ with odd k. Then the Artin-Schreier equation $n^2 + n + d = 0$ has solutions x and $x + 1$ with*

$$x := \sum_{i=0}^{\frac{k-1}{2}} d^{4^i}.$$

Proof. A direct calculation gives

$$x^2 + x = \sum_{i=0}^{\frac{k-1}{2}} d^{2\cdot 4^i} + \sum_{i=0}^{\frac{k-1}{2}} d^{4^i} = \sum_{j=0}^{k} d^{2^j} = Tr(d) + d^{2^k} = d.$$

Therefore x is a solution of the equation, and due to the characteristic, $x + 1$ is as well. □

For even k, we will use the following theorem to calculate a solution directly. Analogously to the Tonelli-Shanks algorithm, the formula utilizes a randomly selected element with trace being equal to one.

Theorem A.3.15. *Let $d \in \mathbb{F}_q$, $Tr(d) = 0$ and $q = 2^k$ with odd k. Given a randomly selected element $r \in \mathbb{F}_q$ satisfying $Tr(r) = 1$, the Artin-Schreier equation $n^2 + n + d = 0$ has solutions x and $x + 1$ with*

$$x := \sum_{i=0}^{k-1} \left(r^{2^i} \sum_{j=0}^{i} d^{2^j} \right).$$

Proof. As before we can calculate

$$\begin{aligned} x^2 + x &= \left(\sum_{i=0}^{k-1}\left(r^{2^i}\sum_{j=0}^{i} d^{2^j}\right)\right)^2 + \left(\sum_{i=0}^{k-1}\left(r^{2^i}\sum_{j=0}^{i} d^{2^j}\right)\right) \\ &= \left(\sum_{i=1}^{k}\left(r^{2^i}\sum_{j=1}^{i} d^{2^j}\right)\right) + \left(\sum_{i=0}^{k-1}\left(r^{2^i}\sum_{j=0}^{i} d^{2^j}\right)\right) \\ &= d\cdot\sum_{i=0}^{k-1} r^{2^i} + r^{2^k}\cdot\sum_{i=1}^{k} d^{2^i} = d\cdot Tr(r) + r\cdot\sum_{i=0}^{k-1} d^{2^i} \\ &= d\cdot Tr(r) + r\cdot Tr(d) = d\cdot 1 + r\cdot 0 = d. \end{aligned}$$

As before, the second solution equals $x+1$. □

With the properties of the trace at hand, we may now also state the universal root finding algorithm that uses the same idea as of Algorithm A.3.2 but generalized to characteristic 2.

Algorithm A.3.16 (Cantor-Zassenhaus for Characteristic 2)**.**

Input: A polynomial $f \in \mathbb{F}_q[Y]$, q even prime power of degree $k > 1$ with all factors of f being of degree one.

Output: Two polynomials $1 \neq f_0, f_1 \neq f$ satisfying $f = f_0 f_1$.

1: Select $g \in_R \{h \in \mathbb{F}_q[Y],\ h \notin \mathbb{F}_q,\ \deg(h) < \deg(f)\}$
2: $g \leftarrow \sum_{i=0}^{k-1} g^{2^k}$
3: Compute $f_i \leftarrow \gcd(f, g+i)$ for $i \in \{0,1\}$.
4: **if** $(1 \neq f_0) \wedge (1 \neq f_1)$ **then**
5: **return** f_0, f_1
6: **else**
7: **Goto** 1.
8: **end if**

The correctness of the algorithm is evident by the definition of the trace. For each prime p dividing f, the trace of g in the field modulo p is either 0 or 1. Therefore, the probability

is high for finding elements having different traces modulo different prime factors, which delivers the factorization.

A.4. Solving Linear Systems

For the algorithms presented in Section 1.1.3, we are required to solve linear systems over the ring $\mathbb{Z}_n$ with n being the order of a Jacobian. By the Chinese remainder theorem and Hensel's lemma, we can restrict to n being a prime. We denote by K the base field over which the linear system is defined. Note that the context of this section does not require K to be finite. Furthermore, we will denote the dimension of the row-space by m and denote vector transposition by a superscript t.

A naive way to solve linear systems is to use the Gauss elimination algorithm that is capable of solving linear systems in $O(m^3)$ finite field operations. For our purposes the structure of our matrix will be very sparse, and one can achieve a runtime of approximately $O(m^2)$. For stating the algorithm, we will need the concept of minimal polynomials.

Definition A.4.1. Let $A \in K^{m\times m}$ be a square matrix. Then there is a unique non-zero monic polynomial $f_{A,K} \in K[t]$ satisfying

$$f_{A,K} \cdot A := f_{A,K}(A) = 0.$$

The polynomial $f_{A,K} \in K[t]$ is the *minimal polynomial* of A over K.

The existence of $f_{A,K}$ follows directly from the *Cayley-Hamilton theorem* that is stated in Theorem A.4.2.

Theorem A.4.2 (Cayley-Hamilton)**.** *Let A be a square matrix over a field K and I the identity matrix of same dimensions as A. Then A is annulled by its characteristic polynomial that is*

$$char(A) := det(A - t \cdot I) \in K[t].$$

Additionally, the polynomials annulling A form an ideal, and $K[t]$ is a principal ideal do-

main. Therefore, it follows that $f_{A,K} \mid char(A)$.

We can use the minimal polynomial to find kernel vectors of a given matrix without using the Gaussian elimination algorithm.

Lemma A.4.3. *Let* $A \in K^{m\times m}$ *be a square matrix with minimal polynomial* $f_{A,K} = \sum_{i=0}^{m'} a_i t^i$. *If* A *is not invertible then there is a non-zero vector* v *such that*

$$\sum_{i=1}^{m'} a_i A^{i-1} v$$

is a non-zero element of the kernel of A.

Proof. Since A is not invertible, it follows that there is a vector $v' \in K^m$ such that $Av' = 0$. Since $f_{A,K}(A)w = 0$ for all $w \in K^m$, this implies $a_0 = 0$, because all higher powers of A multiplied by v' are already equal to zero.

We know that $\sum_{i=1}^{m'} a_i A^{i-1}$ is not the zero matrix, because the opposite is a contradiction to the definition of $f_{A,K}$. Let w be a non-zero row vector of $\sum_{i=1}^{m'} a_i A^{i-1}$ with an entry at its l-th column. Then for v being the lth unit vector we get

$$\sum_{i=1}^{m'} a_i A^{i-1} v \neq 0$$
$$A \sum_{i=1}^{m'} a_i A^{i-1} = f_{A,K}(A)\, v = 0$$

proving the lemma. □

For invertible matrices, we state a similar approach to solve an arbitrary linear system.

Theorem A.4.4. *Let* $A \in K^{m\times m}$ *be an invertible square matrix with minimal polynomial* $f_{A,K} = \sum_{i=0}^{m'} a_i t^i$. *Then for any* $0 \neq v \in V$ *the vector* w *given by*

$$w := -a_0^{-1} \sum_{i=1}^{m'} a_i A^{i-1} v$$

satisfies $Aw = v$.

Proof. For invertible matrices zero is no eigenvalue and thus the characteristic polynomial is not divisible by t. This implies $a_0 \neq 0$. Since $f_{A,\mathbb{F}_q}(A)$ is the zero matrix we conclude

$$\sum_{i=0}^{m'} a_i A^i v = f_{A,K}(M)\, v = 0.$$

Therefore

$$a_0 v = -A \left(\sum_{i=1}^{m'} a_i A^{i-1} v \right),$$

which proves the theorem. □

Given the minimal polynomial it is possible to replace the Gaussian algorithm by using matrix-vector multiplications only. The computation of the minimal polynomial itself is usually not effective using the definition of the characteristic polynomial and determinant calculation. Instead, we can use a probabilistic approach. This approach makes use of the *structure theorem*.

Theorem A.4.5 (Theorem 7.5 of [38])**.** *Let T be a finitely generated R-module with R being a principal ideal domain. Then there are $d_1 \mid \ldots \mid d'_m \in R$ such that*

$$T \cong \bigoplus_{i=1}^{m'} R/(d_i)$$

We know that K^m is a finitely generated $K[t]$-module for any field K with the scalar multiplication given by the insertion of A into a polynomial and its action onto a vector. We see that the existence of the minimal polynomial is equivalent to the existence of $d'_m \in K[t]$ since $f_{A,K}$ is analogous to d'_m here.

Corollary A.4.6. *Let $T =< v >:=< v, Av, A^2v, \ldots >$ with notation from Theorem A.4.4. Then T is a finitely generated $K[t]$ submodule of K^m and thus there is a unique monic polynomial f annulling the sequence of generators $v, Av, A^2v, \ldots$, i.e., for $f = \sum_{i=0}^{m'} a_i t^i$ it is*

$$\sum_{i=0}^{m'} \left(a_i A^i v\right) = 0.$$

$f \mid f_{A,K}$ holds, because T is a sub-module of K^m. Therefore, if $a_0 \neq 0$, the algorithm given by the proof of Theorem A.4.4 can be done using f. This is also possible if A itself is not invertible.

The described method of treating a vector space as a $K[t]$ module can be extended to K by multiplication of a randomly generated vector.

Corollary A.4.7. *Let $r \in K^n$ be a randomly generated non-zero vector. Consider the map*

$$\begin{aligned} \psi : T &\to K \\ v &\mapsto r^t v \end{aligned}$$

with T defined as in the previous corollary and the superscript t denoting vector transposition. Then there is a unique monic polynomial $h \mid f$ of minimal degree, $\sum_{i=0} b_i t^i = h \in K[t]$ satisfying $0 = \sum_{i=0}^{n'} b_i r^t A^{i+j} v$ for all $j \geq 0$. Thus h is annihilating the entire sequence $r^t v, r^t A v, r^t A^2 v, \ldots$.

We can hope that $h = f$ holds. Then, we only need to compute the minimal polynomial of a sequence of elements in K to find the minimal polynomial we need for solving a linear system. If $h \neq f$, one would gain a divisor of f. It is possible to prove that the least common multiple of all possible h is equal to f. We can find f by choosing different r iteratively.

Definition A.4.8. Let $(a_i) \subset K$ be a sequence. Assume there are coefficients $0 \neq c_0, \ldots c_{m-1}, c_m$ such that for all $j > 0$: $0 = \sum_{k=0}^{m} c_k a_{j-k}$.

Then we say (a_i) is *linearly generated* by the polynomial $\sum_{k=0}^{m} c_k t^k$. The minimal m for which such a polynomial exists is the *linear complexity* of the sequence.

For finding a generating polynomial for a sequence of elements, we can use the *Berlekamp-Massey-Algorithm* that is usually used in coding theory for examining the length and structure of linear feedback shift registers. Note that to find a polynomial for a sequence of linear complexity n we need the first $2n$ terms of the sequence for fitting the coefficients.

Algorithm A.4.9 (Berlekamp-Massey)**.**

Input: The first $2s$ terms of a linearly generated sequence $(a_i), a_i \in K$ of linear complexity $m \leq s$

Output: A generating polynomial f of the sequence of minimal length and the linear complexity m' of (a_i).

$m \leftarrow 0$
$l \leftarrow -1$
$C := \sum_{k=0}^{m} c_k t^k \leftarrow 1$
$G \leftarrow C$
$d' \leftarrow 1$
for $0 \leq i \leq 2m$ **do**
  $d \leftarrow a_i - \sum_{k=1}^{m} c_k a_{i-l}$
  if $d \neq 0$ **then**
    $T \leftarrow C$
    $C \leftarrow C - \frac{d}{d'} G t^{i-l}$
    if $2n < i$ **then**
      $m \leftarrow i + 1 - m$
      $G \leftarrow T$
      $l \leftarrow i$
      $d' \leftarrow d$
    end if
  end if
end for
return C, m

Theorem A.4.10. *Algorithm A.4.9 is correct.*

Proof. The operating principle of Algorithm A.4.9 is to iterate over the elements of the sequence and fix occurring errors in the linear combination by adjusting the polynomial

iteratively. Therefore, there are the following invariants and variables used in each iteration:

1. d is the error when computing a_i from its predecessors. If $d = 0$, no further action takes place.

2. m is the current length of the defining polynomial C.

3. All variables indicated with a $'$-symbol are the last values of d and C when the length had changed for the last time. The sequence index when the last length modification appeared is stored in l.

4. We assume that C generates the subsequence $(a_k),\ k < i$ correctly and has its only error at the current index i. The same applies to G, i.e., G generates the sequence up to index $l - 1$ correctly and then had an error of d' at index l.

When $d = 0$, the polynomial is able to generate $(a_k),\ k \leq i$ correctly and the algorithm continues with the next possible i. If $d \neq 0$, there is an update to C given by

$$C_{\text{Update}} := C - \frac{d}{d'} G t^{i-l}$$

Note that the high degree shift multiplied to G makes the new G term act further behind in the sequence, since the low order coefficients act on the most recent sequence values.

Now we can calculate that for $k < i$

$$a_k = \sum_{j=1}^{m} c_j a_{k-j}, \text{ by induction, and}$$
$$0 = \sum_{j=0}^{m'} G_j a_{k-i+l-j}$$

holds, since G was able to generate the sequence up to index $l-1$ correctly and for $k = i-1$ we get $k - i + l - 1 = l - 1$.

Combining both assertions yields that the updated polynomial C_{Update} also generates the

sequence up to index $i-1$. Moreover for index i we know that

$$a_i = d + \sum_{j=1}^{m} c_j a_{i-j}, \text{ by definition of } d, \text{ and}$$
$$d' = \sum_{j=0}^{m'} G_j a_{i-i+l-j} = \sum_{j=0}^{m'} G_j a_{l-j} \text{ by the induction invariant.}$$

By the construction of C_{Update}, we see that the newly generated polynomial also generates a_i.

It is left to restore the loop invariant once the degree of C increases. This is done if $m < \frac{i}{2}$, which implies that the current length of the sequence is more than twice as long as the current generating polynomial. Thus, a polynomial of degree at most m cannot compensate for an error at the current position.

The convenience of the correction to $i+1m$ and the minimality of the resulting degree are difficult to prove. We refer to the corresponding theorems in [41]. □

By using the Berlekamp-Massey-Algorithm, Wiedemann found an algorithm that is able to compute solutions to sparse linear systems over a finite field $\mathbb{F}_q$ [65].

Algorithm A.4.11 (Wiedemann-Algorithm)**.**

Input: A square matrix $A \in K^{m \times m}$ and a vector $v \in \mathbb{F}_q^m$.

Output: A vector $v' \in K^m$ satisfying $Av' = v$ if any exists. Else v' satisfies $Av' = 0$ by high probability.

1: Choose random $0 \neq r \in K^m$.
2: $w \leftarrow v$
3: $s_0 \leftarrow r^t v$
4: **for** $1 \leq i \leq 2m$ **do**
5: $\quad w \leftarrow Mw$
6: $\quad a_i \leftarrow r^t w$
7: **end for**
8: Compute the sequence generating polynomial $f = \sum_{i=0}^{m'} c_i t^i$ of the sequence (a_i)

using the Berlekamp-Massey-Algorithm A.4.9.

9: $v' \leftarrow c_1 v$

10: $w \leftarrow v$

11: **for** $1 \leq i \leq m' - 1$ **do**

12: $\quad w \leftarrow Av$

13: $\quad v' \leftarrow v' + c_{i+1} v$

14: **end for**

15: **if** $c_0 = 0$ **then**

16: $\quad$ **return** v'

17: **end if**

18: $v' \leftarrow -\frac{1}{c_0} v'$

19: **if** $Av' = v$ **then**

20: $\quad$ **return** v'

21: **else**

22: $\quad$ **Goto** Step 1.

23: **end if**

The functionality of Algorithm A.4.11 follows directly from the previous theorems. If the matrix is singular, it is possible that the sequence generating polynomial has a trivial coefficient. Consequently, v' is an element of the matrix kernel. Otherwise, the generating polynomial will be a divisor of the minimal polynomial of the sequence $v, Av, \ldots$, and in case of equality, we will gain a solution of the linear system by Corollary A.4.6.

For our use in discrete logarithm solving algorithms, methods for non-square matrices can also be useful.

Theorem A.4.12 (Kaltofen-Saunders)**.** *Let $A \in K^{m \times m}$ be a square matrix of rank r. Con-*

sider the matrices $\widetilde{A} := UAL$ *with*

$$U := \begin{pmatrix} 1 & u_2 & \dots & u_{m-1} & u_n \\ & 1 & u_2 & & u_{m-1} \\ & & 1 & \ddots & \vdots \\ & & & \ddots & u_2 \\ & & & & 1 \end{pmatrix} \text{ and } L := \begin{pmatrix} 1 & & & & \\ l_2 & 1 & & & \\ \vdots & l_2 & 1 & & \\ l_{m-1} & & \ddots & \ddots & \\ l_n & l_{m-1} & \dots & l_2 & 1 \end{pmatrix}$$

with $u_2, \dots, u_m, l_u, \dots, l_m \in_R S \subseteq K$ *randomly chosen. Then for the leading square sub-matrix* $\widetilde{A}_i$ *of* $\widetilde{A}$ *of dimension* $i \times i$, $i \leq r$ *holds*

$$Prob(\det(\widetilde{A}_i) \neq 0) \geq 1 - \frac{r}{\#S}.$$

The proof is obtained by restricting the proof of [34, Theorem 2] to only one of the matrices $\widetilde{A}_i$. *In the original paper, the authors computed the probability that all sub-matrices of this form are regular.*

Note that for our purposes we will later have to solve linear systems over a finite field that has an order of about the group size of our Jacobian. Thus, we can conclude $\#K \sim \#S >> m \geq r$. Therefore, the probability of obtaining a nonsingular matrix from multiplying with the randomly chosen Toeplitz matrices is rather high.

The use of Toeplitz matrices has an advantage when computing the series of matrix vector multiplication. Multiplication of a Toeplitz matrix by a vector can be handled similarly to polynomial evaluation by using a Horner scheme. Therefore, it is more efficient than using arbitrary random square matrices.

The algorithm of Wiedemann A.4.11 is common for all discrete logarithm solvers that we introduced in Section 1.1.3. Therefore, the exact complexity of the algorithm is significant for all presented discrete logarithm problem solvers. We will start with the complexity of the sub-algorithm by Berlekamp and Massey A.4.9.

Theorem A.4.13. *Given the first* $2m$ *entries of a linear generated sequence over a finite field* K *of linear complexity at most* m. *Then, the Berlekamp-Massey algorithm will find the*

exact linear complexity m' and a sequence generating polynomial using at most $10mm' + 3\log_2(2m)$ *finite field operations.*

Proof. The central loop of Algorithm A.4.9 is performed exactly $2m$ times.

In each iteration, the calculation of d requires at most $2m'$ finite field operations, of which half are multiplications and half are additions. If $d \neq 0$, the current generating polynomial will be updated by using one modular multiplication and one inversion for computing $\frac{d}{d'}$ plus at most $3m'$ operations for constant polynomial multiplication, the shift by t^{i-l} and subtraction of the result from the old generating polynomial. This gives the necessary $10m'm \leq 10m^2$ operations.

Restoring the loop invariants requires three arithmetic operations on each update. The restoration occurs at most $\log_2(2m)$ times. □

In the algorithms used later we will especially use matrices having a low number of non-zero entries in each row or column respectively.

Theorem A.4.14. *Given a sparse, regular matrix $A \in K^{m\times m}$ with an average density of Δ entries per row and a vector $v \in K^m$. Then, the Wiedemann algorithm solves the linear system $Ax = v$ with high probability using at most $(6\Delta + 12)m^2 + 3\log_2(2m)$ finite field operations in K.*

Proof. The creation of the sequence in the first stage of the algorithm, i.e., lines 3–7, requires $2m$ matrix vector products. The runtime for each product is dependent on the total number of matrix entries that is Δm. Furthermore, for each result we have to calculate the product with the row vector r which takes n additions and n multiplications in K.

Therefore, the first stage of the algorithm requires $(4\Delta+2)m^2$ operations in total. Besides the embedded Berlekamp-Massey algorithm that requires $10m^2$ operations as proved in Theorem A.4.13, we need m more matrix vector products that require $2\Delta m$ operations for combining and testing the result. This completes the proof. □

For a non-square matrix $A \in K^{m\times n}$ $m \leq n$ we will use the approach of Kaltofen and Saunders A.4.12. Since we assume that the matrix A has rank m, we can save the computation using the upper Toeplitz matrix U and restrict to L only. Also, we are going to use

the initial $m \times m$ matrix. Then, it is sufficient to use the first m columns of L, so we use a variant $L' \in K^{n \times m}$,

$$L' := \begin{pmatrix} 1 & & & & \\ l_2 & 1 & & & \\ \vdots & l_2 & 1 & & \\ l_{n-1} & & \ddots & \ddots & \\ l_m & l_{m-1} & \dots & l_2 & 1 \\ \vdots & & & \vdots & \vdots \\ l_n & l_{n-1} & \dots & l_{n-m+1} & l_{n-m} \end{pmatrix}$$

such that $\widetilde{A} := AL'$ is a square $m \times m$ matrix.

Multiplying out AL' for the Wiedemann algorithm is not an efficient solution, because the multiplication will increase the density of the matrix and thus Δ rapidly. Instead, we can add the multiplication of L' to all computations where normally only one multiplication by A is required in the classical algorithm.

Algorithm A.4.15.

Input: The initial $n \times m$ Toeplitz matrix L' and a vector $v = (v_1, \dots, v_m)^t$.

Output: The product $w = Av$ with $w = (w_1, \dots, w_n)^t$.

Set $f = \sum_{i=0}^{m-1} l_{i+1} t^i \in K[t]$ and $g = \sum_{i=0}^{m-1} w_{i+1} t^i \in K[t]$.
Compute $h = f \cdot g \equiv \sum_{i=0}^{m-1} w_{i+1} t^i \pmod{t^m}$.
Compute $w_k = \sum_{i=1}^{m} l_{k-m} v_k$ for all $n \geq k > m$.
return $(w_1, \dots, w_n)$

We see that we can do the computation of the matrix-vector product by using a lower half-size polynomial multiplication and by padding the remaining $n - m$ entries manually. Thus, it is more efficient than an ordinary matrix and vector product.

Theorem A.4.16. *Algorithm A.4.15 has a runtime of*

$$O(m \log m \log\log m) + 2(n-m)m$$

finite field operations using the Schönhage-Strassen algorithm for polynomial multiplication or alternatively

$$O(m^{\log_2 3}) + 2(n-m)m \approx O(m^{1.58}) + 2(n-m)m$$

operations using the Karatsuba-Ofman algorithm.

Proof. The claimed complexity follows directly from the complexity assumptions on the embedded polynomial multiplication algorithms. Note that for computing only the lower m coefficients of the product, we only have to compute two of the three products in Karatsuba's method. Therefore, and due to its simple implementation, the Karatsuba method is often preferred for computing the product. □

A.5. Gray Codes

For changing the roots for self-initialization efficiently, a technique called a Gray code can be used to step through the list of options to select the next sieve polynomial.

Definition A.5.1. For $n \in \mathbb{N}$, an n-bit *Gray code* is a sequence v_i of integers with binary representation of at most n bits such that each v_i is unique in the sequence, $v_i < 2^n$ for all $i < 2^n$ and for any i the hamming distance of v_i and v_{i+1} is exactly one, i.e., the binary representation of the two integers differs in exactly one bit.

Essentially, an n-bit Gray code is a reordering of all integers less than 2^n. Two consecutive sieve polynomials differ only by one exchange when this code is used to order the choices between B and B'.

There exist multiple different Gray codes, but for our purpose, the most common one will suffice. To define it, we first need two special operations on $\mathbb{N} \cup \{0\}$ that are common

bit-wise operations used in computer science. The first is the exclusive-or operation that also will play an important role for implementing the sieve iteration in characteristic 2.

Definition A.5.2. Let $n, m \in \mathbb{N} \cup \{0\}$ be written in two-adic representations

$$n = \sum_{i=0}^{\lfloor log_2\ n \rfloor + 1} a_i 2^i$$
$$m = \sum_{i=0}^{\lfloor log_2\ m \rfloor + 1} b_i 2^i,$$

$a_i, b_i \in \{0, 1\}$.

Then the *exclusive or*-operation $\oplus$ on $\mathbb{N} \cup \{0\}$ is defined as

$$m \oplus n := \sum_{i=0}^{max(\lfloor log_2\ n \rfloor, \lfloor log_2\ m \rfloor)+1} \delta(a_i, b_i) 2^i, \text{ with}$$
$$\delta(a_i, b_i) := \begin{cases} 0 & a_i = b_i \\ 1 & a_i \neq b_i \end{cases}.$$

Note that for the definition to be complete we have to pad the shorter representation with zeros.

Definition A.5.3. Let $n, m \in \mathbb{N} \cup \{0\}$ with $m \leq \log_2\ n\ + 1$ and

$$n = \sum_{i=0}^{\lfloor log_2\ n \rfloor + 1} a_i 2^i,$$

$a_i \in \{0, 1\}$.

Then the *binary right shift* of n by m bits is defined as

$$n \gg m := \sum_{i=0}^{\lfloor log_2\ n \rfloor + 1 - m} a_{i+m} 2^i.$$

With these operators we can easily construct a simple-to-calculate Gray code. In our implementations we use this code to generate an order of roots such that our implementation

of the technique of Theorem 3.2.13 is optimal.

Theorem A.5.4. *For $n \in \mathbb{N}$, the sequence*

$$v_i := i \oplus (i \gg 1),$$

$0 \leq i < 2^n$ *defines a n bit Gray code.*

Proof. First of all, we will prove the bijectivity of the function. It is obvious that $\{x < 2^n\}$ is an n-dimensional $\mathbb{F}_2$ vector space with addition given by $\oplus$. From this point of view, the map

$$i \mapsto i \oplus (i \gg 1)$$

defines a $\mathbb{F}_2$ vector space homomorphism with representative matrix of form

$$\begin{pmatrix} 1 & 1 & 0 & \dots & 0 \\ 0 & 1 & \ddots & \ddots & \vdots \\ & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & 1 & 1 \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix}.$$

Since this matrix has determinant one, the map is surjective as well as injective.

The second property is equivalent to the expression $v_i \oplus v_{i+1}$ having only one bit set. To prove this we will need

$$(a \oplus b) \gg 1 = (a \gg 1) \oplus (b \gg 1)$$

for any $a, b \in \mathbb{N}$, which can be proven by a direct computation. Furthermore, we need that if s_i is the position of the most significant bit changed when adding one to $i \in \mathbb{N}$, then

$$i \oplus (i+1) = \sum_{j=0}^{s_i} 1 \cdot 2^j$$

and thus a vector containing only ones up to position s_i and zeros for all further positions.

This is plain, because s_i is the position the carry-bit overflows to, and $i+1$ has only zeros below position s_i, while i must have ones below s_i. For any position exceeding s_i, both numbers are equal, and thus the $\oplus$ zeros those positions.

Combining both formulas yields

$$\begin{aligned} v_i \oplus v_{i+1} &= (i \oplus (i \gg 1)) \oplus (i+1 \oplus (i+1 \gg 1)) \\ &= (i \oplus (i+1)) \oplus ((i \oplus (i+1)) \gg 1) \\ &= 2^{s_i}. \end{aligned}$$

This proves that v_{i+1} and v_i differ only by the s_i-th bit set, and thus that the formula gives a valid Gray code. □

B

Bibliography

[1] Leonard M. Adleman. A Subexponential Algorithm for the Discrete Logarithm Problem with Applications to Cryptography (Abstract). In *FOCS*, pages 55–60. IEEE Computer Society, 1979.

[2] Leonard M. Adleman and Ming-Deh A. Huang. Function Field Sieve Method for Discrete Logarithms over Finite Fields. *Information and Computation*, 151(1):5–16, 1999.

[3] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. *IACR Cryptology ePrint Archive*, 2013:400, 2013.

[4] Daniel J. Bernstein and Tanja Lange. Analysis and optimization of elliptic-curve single-scalar multiplication. In *Finite Fields and Applications*, volume 5181 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2008.

[5] DJ Bernstein and T Lange. Explicit-formulas database [Electronic resource]. *URL: https://hyperelliptic.org/EFD*, 2024.

[6] Joppe W. Bos, Marcelo E. Kaihara, Thorsten Kleinjung, Arjen K. Lenstra, and Peter L. Montgomery. On the Security of 1024-bit RSA and 160-bit Elliptic Curve Cryptography. *IACR Cryptology ePrint Archive*, 2009:389, 2009.

[7] S. Bosch. *Algebra, 9. Auflage*. Springer-Lehrbuch. Springer Berlin Heidelberg, 2020.

[8] David G. Cantor. Computing in the Jacobian of a Hyperelliptic Curve. *Mathematics of Computation*, 48(177):95–101, January 1987.

[9] David G. Cantor and Hans Zassenhaus. A New Algorithm for Factoring Polynomials Over Finite Fields. *Mathematics of Computation*, 36(154):587–592, April 1981.

[10] Jørgen Cherly, Luis Gallardo, Leonid Vaserstein, and Ethel Wheland. Solving quadratic equations over polynomial rings of characteristic two. *Publicacions Matemàtiques*, 42(1):131–142, 1998.

[11] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2012.

[12] An Commeine and Igor A. Semaev. An Algorithm to Solve the Discrete Logarithm Problem with the Number Field Sieve. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 174–190. Springer, 2006.

[13] Don Coppersmith. Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm. *Math. Comput.*, (205):333–350, January.

[14] Claus Diem. On the discrete logarithm problem in class groups of curves. *Math. Comput.*, 80:443–475, 01 2011.

[15] Claus Diem and Sebastian Kochinke. Computing discrete logarithms with pencils. August 2017.

[16] Andreas Enge. Computing discrete logarithms in high-genus hyperelliptic Jacobians in provably subexponential time. *Math. Comput.*, 71(238):729–742, 2002.

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

[18] Andreas Enge and Pierrick Gaudry. An L(1/3 + epsilon) Algorithm for the Discrete Logarithm Problem for Low Degree Curves. *CoRR*, abs/cs/0703032, 2007.

[19] Andreas Enge and Andreas Stein. Smooth ideals in hyperelliptic function fields. *Math. Comput.*, 71(239):1219–1230, 2002.

[20] Federal Office for Information Security (BSI). BSI Technical Guideline TR-02102-1: Cryptographic Mechanisms: Recommendations and Key Lengths. Technical Report BSI TR-02102-1, BSI, Bonn, Germany, 2024. Version 2024-01.

[21] Ralf Flassenberg and Sachar Paulus. Sieving in Function Fields. *Experimental Mathematics*, 8(4):339–349, 1999.

[22] J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[23] Steven D. Galbraith, Florian Hess, and Nigel P. Smart. Extending the GHS Weil Descent Attack. *IACR Cryptology ePrint Archive*, 2001:54, 2001.

[24] Pierrick Gaudry. An Algorithm for Solving the Discrete Log Problem on Hyperelliptic Curves. In Bart Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2000.

[25] Pierrick Gaudry, Emmanuel Thomé, Nicolas Thériault, and Claus Diem. A double large prime variation for small genus hyperelliptic index calculus. *IACR Cryptology ePrint Archive*, 2004:153, 2004.

[26] Ryuichi Harasawa and Joe Suzuki. Fast Jacobian Group Arithmetic on CabCurves. In Wieb Bosma, editor, *Algorithmic Number Theory*, pages 359–376, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[27] Florian Hess. Computing relations in divisor class groups of algebraic curves over finite fields. *Preprint*, 2004.

[28] Florian Hess. Weil Descent Attacks. In I.F. Blake, G. Seroussi, and N.P. Smart, editors, *Advances in Elliptic Curve Cryptography*, Advances in Elliptic Curve Cryptography. Cambridge University Press, 2005.

[29] C. Huneke and I. Swanson. *Integral Closure of Ideals, Rings, and Modules*. Integral closure of ideals, rings, and modules. Cambridge University Press, 2006.

[30] Bundesinstitut für Sicherheit in der Informationstechnik. BSI TR-0311: 1Elliptic Curve Cryptography, 2012.

[31] Antoine Joux. A New Index Calculus Algorithm with Complexity $L(1/4 + o(1))$ in Small Characteristic. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography*, volume 8282 of *Lecture Notes in Computer Science*, pages 355–379. Springer, 2013.

[32] Michael J. Jacobson Jr., Alfred Menezes, and Andreas Stein. Solving Elliptic Curve Discrete Logarithm Problems Using Weil Descent. *IACR Cryptology ePrint Archive*, 2001:41, 2001.

[33] Michael J. Jacobson Jr., Renate Scheidler, and Andreas Stein. Cryptographic Aspects of Real Hyperelliptic Curves. *IACR Cryptology ePrint Archive*, 2010:125, 2010.

[34] Erich Kaltofen and B. David Saunders. On Wiedemann's Method of Solving Sparse Linear Systems. In Harold F. Mattson, Teo Mora, and T. R. N. Rao, editors, *AAECC*, volume 539 of *Lecture Notes in Computer Science*, pages 29–38. Springer, 1991.

[35] Kamal Khuri-Makdisi. Linear algebra algorithms for divisors on an algebraic curve. *Mathematics of Computation*, 73(245):333–357, 2004.

[36] Thorsten Kleinjung. Quadratic sieving. *Math. Comput.*, 85(300), 2016.

[37] Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition*. Addison-Wesley, 1981.

[38] Serge Lang. *Algebra*. Adison-Wesley, 3 edition, 1995.

[39] A.K. Lenstra and H.W.J. Lenstra. *The Development of the Number Field Sieve*. Lecture Notes in Mathematics. Springer, 1993.

[40] Hendrik W. Lenstra. Factoring integers with elliptic curves. *The Annals of Mathematics*, 126(3):649–673, November 1987.

[41] J. Massey. Shift register synthesis and BCH decoding. *Journal of Complexity*, 15:122–127, 1969.

[42] Kraitchik Maurice. *Théorie des nombres*. Gauthier-Villars et cie.

[43] Alfred Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Information Theory*, 39(5):1639–1646, 1993.

[44] Shinji Miura. Algebraic geometric codes on certain plane curves. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 76(12):1–13, 1993.

[45] Gerald Myerson. On resultants. *Proc. Amer. Math. Soc.*, 89(3):419–420, 1983.

[46] National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS). Technical Report FIPS PUB 186-5, U.S. Department of Commerce, Gaithersburg, MD, February 2023. Revised February 3, 2023.

[47] National Institute of Standards and Technology (NIST). Stateless Hash-Based Digital Signature Standard. Technical Report FIPS PUB 205, U.S. Department of Commerce, Gaithersburg, MD, August 2024. SLH-DSA.

[48] Sachar Paulus and Andreas Stein. Comparing Real and Imaginary Arithmetics for Divisor Class Groups of Hyperelliptic Curves. In Joe Buhler, editor, *ANTS*, volume 1423 of *Lecture Notes in Computer Science*, pages 576–591. Springer, 1998.

[49] J. M. Pollard. Monte Carlo Methods for Index Computation $(\mathrm{mod}\, p)$. *Mathematics of Computation*, 32(143):918–924, July 1978.

[50] Carl Pomerance. The Quadratic Sieve Factoring Algorithm. In Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, editors, *EUROCRYPT*, volume 209 of *Lecture Notes in Computer Science*, pages 169–182. Springer, 1984.

[51] Palash Sarkar and Shashank Singh. A New Method for Decomposition in the Jacobian of Small Genus Hyperelliptic Curves. *IACR Cryptology ePrint Archive*, 2014:815, 2014.

[52] J.P. Serre and M.J. Greenberg. *Local Fields*. Graduate Texts in Mathematics. Springer, 1980.

[53] Daniel Shanks. Five number-theoretic algorithms. In *Proceedings of the Second Manitoba Conference on Numerical Mathematics (Univ. Manitoba, Winnipeg, Man., 1972)*, 1973.

[54] V. SHOUP. NTL : A Library for Doing Number Theory. *www.shoup.net/ntl/*.

[55] Henning Stichtenoth. *Algebraic function fields and codes.* Universitext. Springer, 1993.

[56] Erich Strohmaier, Jack Dongarra, Horst Simon, and Martin Meuer. TOP500 Supercomputer Site.

[57] The CADO-NFS Development Team. CADO-NFS, An Implementation of the Number Field Sieve Algorithm, 2017. Version 2.3.0 - Git Version July 4th, 2017.

[58] David C. Terr. A modification of Shanks' baby-step giant-step algorithm. *Math. Comput.*, 69(230):767–773, 2000.

[59] Edlyn Teske. An Elliptic Curve Trapdoor System. *Journal of Cryptology*, 19(1):115–133, 2006.

[60] Nicolas Thériault. Index Calculus Attack for Hyperelliptic Curves of Small Genus. In Chi-Sung Laih, editor, *Advances in Cryptology - ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 75–92. Springer Berlin Heidelberg, 2003.

[61] Wilke Trei. Efficient Modular Arithmetic for SIMD Devices. *IACR Cryptology ePrint Archive*, 2013:652, 2013.

[62] M. D. Velichka, Michael J. Jacobson Jr., and Andreas Stein. Computing Discrete Logarithms in the Jacobian of High-Genus Hyperelliptic Curves over Even Characteristic Finite Fields. *IACR Cryptology ePrint Archive*, 2011:98, 2011.

[63] Ulrich Vollmer. Asymptotically Fast Discrete Logarithms in Quadratic Number Fields. In Wieb Bosma, editor, *ANTS*, volume 1838 of *Lecture Notes in Computer Science*, pages 581–594. Springer, 2000.

[64] André Weil. *Adeles and algebraic groups*. 1982.

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

C

Experiment Curves and Additional Numeric Results

C.1. Weil Descent Curves from Sections 4.2.1 and 4.2.2

The following hyperelliptic curves, used in Sections 4.2.1 and 4.2.2 for testing our implementation, were obtained from elliptic curves via Weil descent. The first four curves are curves of genus 31 over a field extension of $\mathbb{F}_2$. Details about these hyperelliptic curves and their generation using Weil Descent can be found in [32]. Furthermore, these curves were also used in [62] to test a precursor to the sieve developed in this thesis.

C62, $\mathbb{F}_{2^2} = \mathbb{F}_2[w]/(w^2 + w + 1)$
$f(x) = x^{63} + w^2x^{62} + x^{48} + w^2$
$h(x) = x^{31} + x^{30} + wx^{28} + x^{24} + w^2x^{16} + w^2$

Table C.1.: Curve C62 - Genus 31 over $\mathbb{F}_{2^2}$

C93, $\mathbb{F}_{2^3} = \mathbb{F}_2[w]/(w^3 + w + 1)$
$f(x) = w^4x^{63} + w^5x^{62} + w^5x^{60} + w^3x^{56} + w^5x^{48} + wx^{32} + w^5$
$h(x) = w^2x^{31} + w^5x^{30} + x^{28} + w^6x^{24} + w^6$

Table C.2.: Curve C93 - Genus 31 over $\mathbb{F}_{2^3}$

C124, $\mathbb{F}_{2^4} = F_2[w]/(w^4 + w + 1)$
$f(x) = w^3x^{63} + w^7x^{60} + w^3x^{56} + w^3x^{48} + 1$
$h(x) = w^9x^{31} + w^{12}x^{30} + w^8x^{28} + w^{13}x^{24} + w^6x^{16} + w$

Table C.3.: Curve C124 - Genus 31 over $\mathbb{F}_{2^4}$

C155, $\mathbb{F}_{2^5} = F_2[w]/(w^5 + w^2 + 1)$
$f(x) = w^4x^{63} + w^6x^{62} + w^{15}x^{60} + w^{26}x^{56} + w^{25}x^{48} + w^7x^{32} + w^{13}$
$h(x) = w^2x^{31} + w^7x^{30} + w^{30}x^{28} + w^{22}x^{24} + w^3x^{16} + w^{22}$

Table C.4.: Curve C155 - Genus 31 over $\mathbb{F}_{2^5}$

Similar to the previous curves, the following curve was created by a Weil descent of an elliptic curve in [59]. The original curve was defined over $\mathbb{F}_{2^{161}}$ while the resulting hyperelliptic curve has genus 8 over $\mathbb{F}_{2^{23}}$. Therefore, the size of the Jacobian of the descended curve is increased by a factor of 2^{23} compared to the original.

C184, $\mathbb{F}_{2^{23}} = F_2[w]/(w^{23} + w^{21} + w^{20} + w^{18} + w^{15} + w^{13}$
$+w^{12} + w^{11} + w^9 + w^8 + w^2 + w + 1)$

$f(x) = (w^{21} + w^{18} + w^{14} + w^{12} + w^{11} + w^{10} + w^7 + w^4)x^{17} +$
$(w^{22} + w^{21} + w^{20} + w^{19} + w^{18} + w^{16} + w^{15} + w^{14} + w^{10} + w^4 + w^3 + w^2)x^9 +$
$(w^{21} + w^{19} + w^{18} + w^{17} + w^{16} + w^{15} + w^{13} + w^{12} + w^{10} + w^9 + w^8 + w^5 + w^4 + w)x^8 +$
$(w^{22} + w^{20} + w^{19} + w^{18} + w^{17} + w^{15} + w^{11} + w^8 + w^6 + w^4 + w^2 + w)x^5 +$
$(w^{19} + w^{18} + w^{17} + w^{15} + w^{14} + w^{12} + w^6 + w^5 + w^4)x^4 +$
$(w^{19} + w^{18} + w^{17} + w^{14} + w^{13} + w^{10} + w^9 + w^7 + 1)x^3 +$
$(w^{22} + w^{20} + w^{16} + w^{12} + w^8 + w^7 + w^6 + w^5 + w^3 + 1)x^2 +$
$(w^{20} + w^{19} + w^{18} + w^{17} + w^{13} + w^{12} + w^{11} + w^6 + w^3 + w)x$

$h(x) = (w^{17} + w^{15} + w^{14} + w^{13} + w^{12} + w^{11} + w^7 + w^6 + w^5 + w^3 + w)x^8 +$
$(w^{22} + w^{21} + w^{20} + w^{17} + w^{16} + w^{13} + w^9 + w^8 + w^7 + w^6 + w^5 + w^4 + w^2)x^4 +$
$(w^{21} + w^{20} + w^{18} + w^{15} + w^{12} + w^9 + w)x^2 +$
$(w^{21} + w^{20} + w^{19} + w^{16} + w^{12} + w^{10} + w^9 + w^7 + w^6 + w^2 + w)x$

Table C.5.: Curve C184 - Genus 8 over $\mathbb{F}_{2^{23}}$

C.2. Scaling Test Curves and Additional Results for Section 4.2.3

The curves used in Section 4.2.3 were generated by carefully selecting their coefficients. This approach ensures a consistent number of coefficients per polynomial, while also achieving the required degree to obtain the correct genus over the base fields. As for the examples before, the curve equations are given by $y^2 + h(x)v = f(x)$, where $h, f \in \mathbb{F}_q[x]$. Besides the curve equations, we also present the extended timing tables as described in Section 4.2.3.

All experiments, with the exception of those described in Table C.19, use a factor base size of 20,000 elements. Therefore, we will not mention the used factor base size explicitly.

Genus 8 over $\mathbb{F}_{2^{15}}$

C8, $\mathbb{F}_{2^{15}} = \mathbb{F}_2[w]/(w^{15} + w^{14} + w^{12} + w^7 + w^6 + w^2 + 1)$
$f(x) = x^{17} + x^{14} + x^{10} + x^6 + 1$
$h(x) = x^8 + x^5 + x^4 + x^3 + 1$

Table C.6.: Curve C8 - Genus 8 over $\mathbb{F}_{2^{15}}$

M	P_{ONCE}	Array Size	η_{max}	Sieve Polyn.	$\mathcal{P}(\eta_{max}, \mathcal{F})$	Core time	Divisor Time
0	8	32,768	9	59,622,147	$5.11 \cdot 10^{-9}$	14h 26m	$2.66 \cdot 10^{-8} s$

Table C.7.: Results for the Curve C8 - Genus 8 over $\mathbb{F}_{2^{15}}$

In the following additional table, we present the results for the genus-8 curve with an activated single-large-prime method. Here, the smoothness probability and the time per divisor were identical to the above results. Therefore, we give the number of relations found by the large-prime method instead.

M	P_{ONCE}	Array Size	η_{max}	Sieve Polyn.	Core time	Found LP	Relations from LP
0	8	32,768	9	14,138,840	3h 48m 39s	9,940	7,715 (77%)

Table C.8.: Results for the Curve C8 with the Single-Large-Prime Algorithm

Genus 10 over $\mathbb{F}_{2^{12}}$

C10, $\mathbb{F}_{2^{12}} = \mathbb{F}_2[w]/(w^{12} + w^{11} + w^9 + w^6 + w^5 + w + 1)$
$f(x) = x^{21} + x^{14} + x^{10} + 1$
$h(x) = x^{10} + x^9 + x^7 + x^6 + x^3 + 1$

Table C.9.: Curve C10 - Genus 10 over $\mathbb{F}_{2^{12}}$

M	P_{ONCE}	Array Size	η_{max}	Sieve Polyn.	$\mathcal{P}(\eta_{max}, \mathcal{F})$	Core time	Divisor Time
0	10	4,095	11	36,645,304	$6.66 \cdot 10^{-8}$	1d 11h 15m	$8.45 \cdot 10^{-7} s$
1	9	16,777,216	12	4,537,711	$1.31 \cdot 10^{-10}$	13h 33m	$6.41 \cdot 10^{-10} s$

Table C.10.: Results for the Curve C10 - Genus 10 over $\mathbb{F}_{2^{12}}$

Genus 12 over $\mathbb{F}_{2^{10}}$

$$\begin{aligned}&\text{C12, } \mathbb{F}_{2^{10}} = \mathbb{F}_2[w]/(w^{10} + w^7 + w^5 + w^3 + 1)\\ &f(x) = x^{25} + x^{18} + x^{14} + 1\\ &h(x) = x^{12} + x^{11} + x^7 + x^6 + x^3 + 1\end{aligned}$$

Table C.11.: Curve C12 - Genus 12 over $\mathbb{F}_{2^{10}}$

M	P_{ONCE}	Array Size	η_{max}	Sieve Polyn.	$\mathcal{P}(\eta_{max}, \mathcal{F})$	Core time	Divisor Time
1	11	1,048,576	13	870,778,182	$1.1 \cdot 10^{-11}$	3d 16h 34m	$3.49 \cdot 10^{-10} s$

Table C.12.: Results for the Curve C12 - Genus 12 over $\mathbb{F}_{2^{10}}$

Genus 15 over $\mathbb{F}_{2^8}$

$$\begin{aligned}&\text{C15, } \mathbb{F}_{2^8} = \mathbb{F}_2[w]/(w^8 + w^7 + w^6 + w^5 + w^4 + w^3 + 1)\\ &f(x) = x^{31} + x^{28} + x^{24} + x^{16} + 1\\ &h(x) = x^{15} + x^{14} + x^{12} + x^8 + x^7 + 1\end{aligned}$$

Table C.13.: Curve C15 - Genus 15 over $\mathbb{F}_{2^8}$

M	P_{ONCE}	Array Size	η_{max}	Sieve Polyn.	$\mathcal{P}(\eta_{max}, \mathcal{F})$	Core time	Divisor Time
1	7	65,536	17	14,599,832	$1.05 \cdot 10^{-8}$	11h 54m	$4.48 \cdot 10^{-8} s$
2	6	16,777,215	19	10,913,239	$5.56 \cdot 10^{-11}$	3d 12h 46m	$1.67 \cdot 10^{-9} s$

Table C.14.: Results for the Curve C15 - Genus 20 over $\mathbb{F}_{2^8}$

Genus 20 over $\mathbb{F}_{2^6}$

C20, $\mathbb{F}_{2^6} = \mathbb{F}_2[w]/(w^6 + w + 1)$
$f(x) = x^{41} + x^{38} + x^{34} + x^{26} + 1$
$h(x) = x^{20} + x^{19} + x^{17} + x^{13} + x^5 + 1$

Table C.15.: Curve C20 - Genus 20 over $\mathbb{F}_{2^6}$

M	P_{ONCE}	Array Size	η_{max}	Sieve Polyn.	$\mathcal{P}(\eta_{max}, \mathcal{F})$	Core time	Divisor Time
2	6	262,144	23	51,591,339	$7.39 \cdot 10^{-10}$	2d 0h	$1.28 \cdot 10^{-8}s$
2	8	262,144	25	3,931,674,790	$9.70 \cdot 10^{-12}$	12d 2h 53m	$1.01 \cdot 10^{-9}s$
2	9	262,144	23	56,462,644	$6.76 \cdot 10^{-10}$	18h 25m	$4.48 \cdot 10^{-9}s$
3	8	16,777,216	25	56,590,963	$1.05 \cdot 10^{-11}$	12d 7h 10m	$1.12 \cdot 10^{-9}s$

Table C.16.: Results for the Curve C20 - Genus 20 over $\mathbb{F}_{2^6}$

Genus 24 over $\mathbb{F}_{2^5}$

C24, $\mathbb{F}_{2^5} = \mathbb{F}_2[w]/(w^5 + w^2 + 1)$
$f(x) = x^{49} + x^{33} + x^{29} + 1$
$h(x) = x^{24} + x^{23} + x^{19} + x^6 + x^3 + 1$

Table C.17.: Curve C24 - Genus 24 over $\mathbb{F}_{2^5}$

M	P_{ONCE}	Array Size	η_{max}	Sieve Polyn.	$\mathcal{P}(\eta_{max}, \mathcal{F})$	Core time	Divisor Time
2	7	32,768	28	24,354,016	$1.25 \cdot 10^{-8}$	19h 58m	$9.01 \cdot 10^{-8} s$
2	11	32,768	27	1,889,890	$1.61 \cdot 10^{-7}$	2h 23m	$1.39 \cdot 10^{-7} s$
3	5	1,048,576	29	6,699,997	$1.42 \cdot 10^{-9}$	7h 13m	$3.70 \cdot 10^{-9} s$
3	7	1,048,576	28	769,621	$1.24 \cdot 10^{-8}$	1h 40m	$7.43 \cdot 10^{-9} s$
3	10	1,048,576	29	7,476,031	$1.28 \cdot 10^{-9}$	3h 36m	$1.65 \cdot 10^{-9} s$
4	5	33,554,432	29	204,308	$1.45 \cdot 10^{-9}$	6h 15m	$3.28 \cdot 10^{-9} s$
4	10	33,554,432	29	223,279	$1.33 \cdot 10^{-9}$	5h 32m	$2.66 \cdot 10^{-9} s$

Table C.18.: Results for the Curve C24 - Genus 24 over $\mathbb{F}_{2^5}$

$\#\mathcal{F}$	B	Sieve Polynomials	$\mathcal{P}(27, \mathcal{F})$	Time (s)	Divisor Time (s)
700	3	7,631,723,183	$4.44 \cdot 10^{-14}$	3,688,700	$4.61 \cdot 10^{-10}$
1,000	3	4,911,322,047	$9.81 \cdot 10^{-14}$	2,599,107	$5.05 \cdot 10^{-10}$
1,500	3	2,889,024,048	$2.49 \cdot 10^{-13}$	1,617,968	$5.34 \cdot 10^{-10}$
2,000	3	1,325,275,246	$7.23 \cdot 10^{-13}$	963,488	$6.93 \cdot 10^{-10}$
2,500	3	778,401,277	$1.54 \cdot 10^{-12}$	631,916	$7.74 \cdot 10^{-10}$
3,000	3	438,132,659	$3.28 \cdot 10^{-12}$	406,468	$8.85 \cdot 10^{-10}$
3,500	3	260,581,450	$6.42 \cdot 10^{-12}$	270,546	$9.90 \cdot 10^{-10}$
4,000	3	149,847,405	$1.28 \cdot 10^{-11}$	180,945	$1.15 \cdot 10^{-10}$
4,500	3	90,480,178	$2.38 \cdot 10^{-11}$	123,642	$1.30 \cdot 10^{-9}$
5,000	3	53,160,688	$4.49 \cdot 10^{-11}$	86,290	$1.55 \cdot 10^{-9}$
5,500	3	34,404,655	$7.64 \cdot 10^{-11}$	63,286	$1.75 \cdot 10^{-9}$
6,000	3	23,816,392	$1.20 \cdot 10^{-10}$	47,652	$1.91 \cdot 10^{-9}$
6,500	3	16,063,723	$1.93 \cdot 10^{-10}$	36,163	$2.15 \cdot 10^{-9}$
7,000	3	11,216,908	$2.98 \cdot 10^{-10}$	27,524	$2.34 \cdot 10^{-9}$
7,500	3	7,648,183	$4.68 \cdot 10^{-10}$	21,224	$2.65 \cdot 10^{-9}$
8,000	3	5,396,761	$7.08 \cdot 10^{-10}$	16,749	$2.96 \cdot 10^{-9}$

Table C.19.: Additional results for the curve C24 with different factor base sizes

$\#\mathcal{F}$	B	Sieve Polynomials	$\mathcal{P}(27, \mathcal{F})$	Time (s)	Divisor Time (s)
8,500	3	4,036,641	$1.01 \cdot 10^{-9}$	13,425	$3.17 \cdot 10^{-9}$
9,000	3	2,934,014	$1.46 \cdot 10^{-9}$	10,767	$3.50 \cdot 10^{-9}$
9,500	3	2,142,601	$2.12 \cdot 10^{-9}$	8,685	$3.87 \cdot 10^{-9}$
10,000	3	1,530,969	$3.12 \cdot 10^{-9}$	6,993	$4.36 \cdot 10^{-9}$
10,500	3	1,153,363	$4.35 \cdot 10^{-9}$	5,741	$4.75 \cdot 10^{-9}$
11,000	3	882,090	$5.95 \cdot 10^{-9}$	4,820	$5.21 \cdot 10^{-9}$
11,500	3	662,893	$8.28 \cdot 10^{-9}$	3,975	$5.72 \cdot 10^{-9}$
11,544	3	646,365	$8.52 \cdot 10^{-9}$	3,935	$5.81 \cdot 10^{-9}$
12,000	4	668,091	$8.57 \cdot 10^{-9}$	4,051	$5.78 \cdot 10^{-9}$
12,500	4	660,024	$9.04 \cdot 10^{-9}$	4,216	$6.09 \cdot 10^{-9}$
13,000	4	661,009	$9.39 \cdot 10^{-9}$	4,330	$6.25 \cdot 10^{-9}$
13,500	4	690,361	$9.33 \cdot 10^{-9}$	4,462	$6.16 \cdot 10^{-9}$
14,000	4	702,734	$9.51 \cdot 10^{-9}$	4,611	$6.26 \cdot 10^{-9}$
14,500	4	694,259	$9.97 \cdot 10^{-9}$	4,723	$6.49 \cdot 10^{-9}$
15,000	4	701,965	$1.02 \cdot 10^{-8}$	4,856	$6.60 \cdot 10^{-9}$
15,500	4	713,750	$1.04 \cdot 10^{-8}$	4,986	$6.66 \cdot 10^{-9}$
16,000	4	726,631	$1.05 \cdot 10^{-8}$	5,110	$6.71 \cdot 10^{-9}$
16,500	4	726,850	$1.08 \cdot 10^{-8}$	5,183	$6.80 \cdot 10^{-9}$
17,000	4	725,898	$1.12 \cdot 10^{-8}$	5,320	$6.99 \cdot 10^{-9}$
17,500	4	743,027	$1.12 \cdot 10^{-8}$	5,453	$7.00 \cdot 10^{-9}$
18,000	4	755,606	$1.14 \cdot 10^{-8}$	5,574	$7.04 \cdot 10^{-9}$
18,500	4	749,835	$1.18 \cdot 10^{-8}$	5,709	$7.26 \cdot 10^{-9}$
19,000	4	757,211	$1.20 \cdot 10^{-8}$	5,789	$7.29 \cdot 10^{-9}$
19,500	4	760,709	$1.22 \cdot 10^{-8}$	5,908	$7.41 \cdot 10^{-9}$
20,000	4	767,538	$1.24 \cdot 10^{-8}$	6,065	$7.54 \cdot 10^{-9}$
22,000	4	775,689	$1.35 \cdot 10^{-8}$	6,474	$7.96 \cdot 10^{-9}$
24,000	4	778,537	$1.47 \cdot 10^{-8}$	6,846	$8.39 \cdot 10^{-9}$

Table C.19.: Additional results for the curve C24 with different factor base sizes

$\#\mathcal{F}$	B	Sieve Polynomials	$\mathcal{P}(27, \mathcal{F})$	Time (s)	Divisor Time (s)
26,000	4	768,590	$1.61 \cdot 10^{-8}$	7,160	$8.88 \cdot 10^{-9}$
28,000	4	755,715	$1.77 \cdot 10^{-8}$	7,473	$9.43 \cdot 10^{-9}$
30,000	4	738,921	$1.94 \cdot 10^{-8}$	7,693	$9.93 \cdot 10^{-9}$
32,000	4	730,368	$2.09 \cdot 10^{-8}$	7,949	$1.04 \cdot 10^{-8}$
34,000	4	704,740	$2.30 \cdot 10^{-8}$	8,075	$1.09 \cdot 10^{-8}$
36,000	4	692,118	$2.48 \cdot 10^{-8}$	8,293	$1.14 \cdot 10^{-8}$
38,000	4	676,310	$2.68 \cdot 10^{-8}$	8,550	$1.21 \cdot 10^{-8}$
40,000	4	656,858	$2.90 \cdot 10^{-8}$	8,756	$1.27 \cdot 10^{-8}$
42,000	4	632,253	$3.17 \cdot 10^{-8}$	8,913	$1.34 \cdot 10^{-8}$
44,000	4	612,446	$3.43 \cdot 10^{-8}$	9,056	$1.41 \cdot 10^{-8}$
46,000	4	603,124	$3.64 \cdot 10^{-8}$	9,182	$1.45 \cdot 10^{-8}$
48,000	4	582,579	$3.93 \cdot 10^{-8}$	9,284	$1.52 \cdot 10^{-8}$
50,000	4	565,380	$4.22 \cdot 10^{-8}$	9,352	$1.58 \cdot 10^{-8}$
52,000	4	545,162	$4.55 \cdot 10^{-8}$	9,404	$1.65 \cdot 10^{-8}$
54,000	4	528,668	$4.87 \cdot 10^{-8}$	9,477	$1.71 \cdot 10^{-8}$
56,000	4	510,620	$5.23 \cdot 10^{-8}$	9,494	$1.77 \cdot 10^{-8}$
58,000	4	487,147	$5.68 \cdot 10^{-8}$	9,528	$1.87 \cdot 10^{-8}$
60,000	4	476,746	$6.00 \cdot 10^{-8}$	9,599	$1.92 \cdot 10^{-8}$
62,000	4	460,981	$6.41 \cdot 10^{-8}$	9,566	$1.98 \cdot 10^{-8}$
64,000	4	445,044	$6.86 \cdot 10^{-8}$	9,587	$2.05 \cdot 10^{-8}$
66,000	4	427,556	$7.36 \cdot 10^{-8}$	9,466	$2.11 \cdot 10^{-8}$
68,000	4	409,538	$7.92 \cdot 10^{-8}$	9,471	$2.21 \cdot 10^{-8}$
70,000	4	398,097	$8.39 \cdot 10^{-8}$	9,464	$2.27 \cdot 10^{-8}$
72,000	4	384,503	$8.93 \cdot 10^{-8}$	9,493	$2.35 \cdot 10^{-8}$
74,000	4	370,666	$9.52 \cdot 10^{-8}$	9,509	$2.45 \cdot 10^{-8}$
76,000	4	356,594	$1.02 \cdot 10^{-7}$	9,490	$2.54 \cdot 10^{-8}$
78,000	4	343,655	$1.08 \cdot 10^{-7}$	9,496	$2.64 \cdot 10^{-8}$

Table C.19.: Additional results for the curve C24 with different factor base sizes

$\#\mathcal{F}$	B	Sieve Polynomials	$\mathcal{P}(27, \mathcal{F})$	Time (s)	Divisor Time (s)
80,000	4	328,170	$1.16 \cdot 10^{-7}$	9,503	$2.76 \cdot 10^{-8}$
82,000	4	318,523	$1.23 \cdot 10^{-7}$	9,472	$2.84 \cdot 10^{-8}$
84,000	4	307,826	$1.30 \cdot 10^{-7}$	9,424	$2.92 \cdot 10^{-8}$
86,000	4	295,056	$1.39 \cdot 10^{-7}$	9,376	$3.03 \cdot 10^{-8}$
88,000	4	284,882	$1.47 \cdot 10^{-7}$	9,367	$3.14 \cdot 10^{-8}$
90,000	4	274,716	$1.56 \cdot 10^{-7}$	9,336	$3.24 \cdot 10^{-8}$
92,000	4	263,480	$1.67 \cdot 10^{-7}$	9,281	$3.36 \cdot 10^{-8}$
94,000	4	253,722	$1.77 \cdot 10^{-7}$	9,203	$3.46 \cdot 10^{-8}$
96,000	4	244,671	$1.87 \cdot 10^{-7}$	9,136	$3.56 \cdot 10^{-8}$
98,000	4	234,998	$1.99 \cdot 10^{-7}$	9,064	$3.68 \cdot 10^{-8}$
100,000	4	225,806	$2.11 \cdot 10^{-7}$	8,972	$3.79 \cdot 10^{-8}$
102,000	4	218,074	$2.23 \cdot 10^{-7}$	8,930	$3.91 \cdot 10^{-8}$
104,000	4	208,876	$2.37 \cdot 10^{-7}$	8,879	$4.05 \cdot 10^{-8}$
106,000	4	199,876	$2.53 \cdot 10^{-7}$	8,812	$4.20 \cdot 10^{-8}$
108,000	4	192,342	$2.68 \cdot 10^{-7}$	8,740	$4.33 \cdot 10^{-8}$
110,000	4	185,663	$2.83 \cdot 10^{-7}$	8,642	$4.44 \cdot 10^{-8}$
112,000	4	178,859	$2.99 \cdot 10^{-7}$	8,582	$4.58 \cdot 10^{-8}$
114,000	4	172,521	$3.15 \cdot 10^{-7}$	8,514	$4.54 \cdot 10^{-8}$
116,000	4	164,789	$3.36 \cdot 10^{-7}$	8,448	$4.89 \cdot 10^{-8}$
118,000	4	158,945	$3.54 \cdot 10^{-7}$	8,363	$5.02 \cdot 10^{-8}$
120,000	4	153,471	$3.73 \cdot 10^{-7}$	8,272	$5.14 \cdot 10^{-8}$
122,000	4	147,500	$3.94 \cdot 10^{-7}$	8,214	$5.31 \cdot 10^{-8}$
124,000	4	142,299	$4.16 \cdot 10^{-7}$	8,205	$5.50 \cdot 10^{-8}$
126,000	4	136,364	$4.41 \cdot 10^{-7}$	8,115	$5.68 \cdot 10^{-8}$
128,000	4	131,711	$4.63 \cdot 10^{-7}$	8,028	$5.81 \cdot 10^{-8}$
130,000	4	126,553	$4.90 \cdot 10^{-7}$	7,908	$5.96 \cdot 10^{-8}$
132,000	4	122,429	$5.14 \cdot 10^{-7}$	7,647	$5.96 \cdot 10^{-8}$

Table C.19.: Additional results for the curve C24 with different factor base sizes

$\#\mathcal{F}$	B	Sieve Polynomials	$\mathcal{P}(27, \mathcal{F})$	Time (s)	Divisor Time (s)
134,000	4	117,686	$5.43 \cdot 10^{-7}$	7,570	$6.13 \cdot 10^{-8}$
136,000	4	113,611	$5.71 \cdot 10^{-7}$	7,523	$6.31 \cdot 10^{-8}$
138,000	4	109,353	$6.02 \cdot 10^{-7}$	7,504	$6.54 \cdot 10^{-8}$
140,000	4	105,278	$6.34 \cdot 10^{-7}$	7,431	$6.73 \cdot 10^{-8}$
142,000	4	101,143	$6.70 \cdot 10^{-7}$	7,373	$6.95 \cdot 10^{-8}$
144,000	4	97,531	$7.04 \cdot 10^{-7}$	7,380	$7.22 \cdot 10^{-8}$
146,000	4	94,430	$7.37 \cdot 10^{-7}$	7,284	$7.36 \cdot 10^{-8}$
148,000	4	90,759	$7.78 \cdot 10^{-7}$	7,297	$7.67 \cdot 10^{-8}$
150,000	4	87,396	$8.18 \cdot 10^{-7}$	7,202	$7.86 \cdot 10^{-8}$
152,000	4	84,876	$8.54 \cdot 10^{-7}$	7,189	$8.08 \cdot 10^{-8}$
154,000	4	81,958	$8.96 \cdot 10^{-7}$	7,111	$8.27 \cdot 10^{-8}$
156,000	4	78,713	$9.45 \cdot 10^{-7}$	6,975	$8.45 \cdot 10^{-8}$
158,000	4	75,790	$9.94 \cdot 10^{-7}$	6,896	$8.68 \cdot 10^{-8}$
160,000	4	72,975	$1.05 \cdot 10^{-6}$	6,878	$8.99 \cdot 10^{-8}$
162,000	4	69,705	$1.11 \cdot 10^{-6}$	6,764	$9.25 \cdot 10^{-8}$
164,000	4	67,698	$1.16 \cdot 10^{-6}$	6,747	$9.50 \cdot 10^{-8}$
166,000	4	65,424	$1.21 \cdot 10^{-6}$	6,698	$9.76 \cdot 10^{-8}$
168,000	4	62,938	$1.27 \cdot 10^{-6}$	6,613	$1.00 \cdot 10^{-7}$
170,000	4	61,200	$1.32 \cdot 10^{-6}$	6,597	$1.03 \cdot 10^{-7}$
172,000	4	58,533	$1.40 \cdot 10^{-6}$	6,526	$1.06 \cdot 10^{-7}$
174,000	4	56,885	$1.46 \cdot 10^{-6}$	6,477	$1.09 \cdot 10^{-7}$
176,000	4	54,331	$1.54 \cdot 10^{-6}$	6,400	$1.12 \cdot 10^{-7}$
178,000	4	52,793	$1.61 \cdot 10^{-6}$	6,397	$1.16 \cdot 10^{-7}$
180,000	4	51,132	$1.68 \cdot 10^{-6}$	6,295	$1.17 \cdot 10^{-7}$
182,000	4	49,297	$1.76 \cdot 10^{-6}$	6,276	$1.21 \cdot 10^{-7}$
184,000	4	47,964	$1.83 \cdot 10^{-6}$	6,183	$1.23 \cdot 10^{-7}$
186,000	4	46,084	$1.92 \cdot 10^{-6}$	6,153	$1.27 \cdot 10^{-7}$

Table C.19.: Additional results for the curve C24 with different factor base sizes

$\#\mathcal{F}$	B	Sieve Polynomials	$\mathcal{P}(27, \mathcal{F})$	Time (s)	Divisor Time (s)
188,000	4	44,321	$2.02 \cdot 10^{-6}$	6,146	$1.32 \cdot 10^{-7}$
190,000	4	43,210	$2.10 \cdot 10^{-6}$	6,012	$1.33 \cdot 10^{-7}$
192,000	4	41,828	$2.19 \cdot 10^{-6}$	5,976	$1.36 \cdot 10^{-7}$
194,000	4	40,409	$2.29 \cdot 10^{-6}$	5,998	$1.42 \cdot 10^{-7}$
196,000	4	38,895	$2.40 \cdot 10^{-6}$	5,874	$1.44 \cdot 10^{-7}$
198,000	4	37,752	$2.50 \cdot 10^{-6}$	5,845	$1.48 \cdot 10^{-7}$
200,000	4	36,300	$2.63 \cdot 10^{-6}$	5,842	$1.53 \cdot 10^{-7}$
202,000	4	35,343	$2.73 \cdot 10^{-6}$	5,726	$1.55 \cdot 10^{-7}$
204,000	4	33,834	$2.88 \cdot 10^{-6}$	5,657	$1.59 \cdot 10^{-7}$
206,000	4	32,703	$3.00 \cdot 10^{-6}$	5,652	$1.65 \cdot 10^{-7}$
208,000	4	31,552	$3.14 \cdot 10^{-6}$	5,652	$1.71 \cdot 10^{-7}$
210,000	4	30,481	$3.29 \cdot 10^{-6}$	5,488	$1.72 \cdot 10^{-7}$
212,000	4	29,374	$3.44 \cdot 10^{-6}$	5,473	$1.78 \cdot 10^{-7}$
214,000	4	28,307	$3.61 \cdot 10^{-6}$	5,457	$1.84 \cdot 10^{-7}$
216,000	4	27,315	$3.77 \cdot 10^{-6}$	5,463	$1.91 \cdot 10^{-7}$
218,000	4	26,292	$3.95 \cdot 10^{-6}$	5,272	$1.91 \cdot 10^{-7}$
220,000	4	25,461	$4.12 \cdot 10^{-6}$	5,238	$1.96 \cdot 10^{-7}$
222,000	4	24,734	$4.28 \cdot 10^{-6}$	5,250	$2.02 \cdot 10^{-7}$
224,000	4	23,806	$4.49 \cdot 10^{-6}$	5,246	$2.10 \cdot 10^{-7}$
226,000	4	23,044	$4.68 \cdot 10^{-6}$	5,204	$2.15 \cdot 10^{-7}$
228,000	4	22,216	$4.89 \cdot 10^{-6}$	5,048	$2.17 \cdot 10^{-7}$
230,000	4	21,527	$5.09 \cdot 10^{-6}$	5,039	$2.23 \cdot 10^{-7}$
232,000	4	20,698	$5.35 \cdot 10^{-6}$	5,041	$2.32 \cdot 10^{-7}$
234,000	4	20,045	$5.57 \cdot 10^{-6}$	5,024	$2.39 \cdot 10^{-7}$
236,000	4	19,378	$5.81 \cdot 10^{-6}$	4,979	$2.45 \cdot 10^{-7}$
238,000	4	18,640	$6.09 \cdot 10^{-6}$	4,790	$2.45 \cdot 10^{-7}$
240,000	4	18,034	$6.35 \cdot 10^{-6}$	4,796	$2.54 \cdot 10^{-7}$

Table C.19.: Additional results for the curve C24 with different factor base sizes

$\#\mathcal{F}$	B	Sieve Polynomials	$\mathcal{P}(27, \mathcal{F})$	Time (s)	Divisor Time (s)
242,000	4	17,342	$6.65 \cdot 10^{-6}$	4,786	$2.63 \cdot 10^{-7}$
244,000	4	16,779	$6.93 \cdot 10^{-6}$	4,773	$2.71 \cdot 10^{-7}$
246,000	4	16,259	$7.21 \cdot 10^{-6}$	4,779	$2.80 \cdot 10^{-7}$
248,000	4	15,614	$7.57 \cdot 10^{-6}$	4,735	$2.89 \cdot 10^{-7}$
250,000	4	15,107	$7.89 \cdot 10^{-6}$	4,538	$2.86 \cdot 10^{-7}$
252,000	4	14,522	$8.27 \cdot 10^{-6}$	4,519	$2.97 \cdot 10^{-7}$
254,000	4	14,112	$8.58 \cdot 10^{-6}$	4,506	$3.05 \cdot 10^{-7}$
256,000	4	13,616	$8.97 \cdot 10^{-6}$	4,499	$3.15 \cdot 10^{-7}$
258,000	4	13,220	$9.31 \cdot 10^{-6}$	4,473	$3.23 \cdot 10^{-7}$
260,000	4	12,716	$9.75 \cdot 10^{-6}$	4,465	$3.35 \cdot 10^{-7}$
262,000	4	12,297	$1.02 \cdot 10^{-5}$	4,435	$3.44 \cdot 10^{-7}$
264,000	4	11,855	$1.06 \cdot 10^{-5}$	4,170	$3.35 \cdot 10^{-7}$
266,000	4	11,467	$1.11 \cdot 10^{-5}$	4,144	$3.45 \cdot 10^{-7}$
268,000	4	11,101	$1.15 \cdot 10^{-5}$	4,142	$3.56 \cdot 10^{-7}$
270,000	4	10,716	$1.20 \cdot 10^{-5}$	4,120	$3.67 \cdot 10^{-7}$
272,000	4	10,362	$1.25 \cdot 10^{-5}$	4,124	$3.80 \cdot 10^{-7}$
274,000	4	9,992	$1.31 \cdot 10^{-5}$	4,099	$3.91 \cdot 10^{-7}$
274,016	4	9,984	$1.31 \cdot 10^{-5}$	4,106	$3.92 \cdot 10^{-7}$

Table C.19.: Additional results for the curve C24 with different factor base sizes

Genus 30 over $\mathbb{F}_{2^4}$

C30, $\mathbb{F}_{2^4} = \mathbb{F}_2[w]/(w^4 + w + 1)$
$f(x) = x^{61} + x^{45} + x^{36} + 1$
$h(x) = x^{30} + x^{29} + x^{25} + x^{13} + x^{10} + 1$

Table C.20.: Curve C30 - Genus 30 over $\mathbb{F}_{2^4}$

M	P_{ONCE}	Array Size	η_{max}	Sieve Polyn.	$\mathcal{P}(\eta_{max}, \mathcal{F})$	Core time	Divisor Time
3	9	65,536	34	1,255,234	$1.22 \cdot 10^{-7}$	3h 27m	$1.51 \cdot 10^{-7}s$
3	13	65,536	35	1,650,062	$9.25 \cdot 10^{-8}$	2h 24m	$7.99 \cdot 10^{-8}s$
4	5	1,048,576	36	899,335	$1.06 \cdot 10^{-8}$	4h 21m	$1.66 \cdot 10^{-8}s$
4	13	1,048,576	35	1,177,523	$8.09 \cdot 10^{-9}$	2h 16m	$6.61 \cdot 10^{-9}s$
5	5	16,777,216	36	330,619	$1.8 \cdot 10^{-9}$	5h 57m	$3.86 \cdot 10^{-9}s$
5	6	16,777,216	37	341,347	$1.75 \cdot 10^{-9}$	6h 39m	$4.18 \cdot 10^{-9}s$
5	8	16,777,216	37	324,692	$1.83 \cdot 10^{-9}$	6h 24m	$4.23 \cdot 10^{-9}s$

Table C.21.: Results for the Curve C30 - Genus 30 over $\mathbb{F}_{2^4}$

Genus 40 over $\mathbb{F}_{2^3}$

C40, $\mathbb{F}_{2^3} = \mathbb{F}_2[w]/(w^3 + w + 1)$

$f(x) = x^{81} + x^{55} + x^{39} + 1$

$h(x) = x^{40} + x^{30} + x^{26} + x^{13} + x^{10} + 1$

Table C.22.: Curve C40 - Genus 40 over $\mathbb{F}_{2^3}$

M	P_{ONCE}	Array Size	η_{max}	Sieve Polyn.	$\mathcal{P}(\eta_{max}, \mathcal{F})$	Core time	Divisor Time
5	7	262,144	46	7,104,721	$5.37 \cdot 10^{-9}$	18h 41m	$3.07 \cdot 10^{-8}s$
6	8	2,097,152	49	19,133,449	$2.49 \cdot 10^{-10}$	1d 5h 59m	$2.69 \cdot 10^{-9}s$
6	11	2,097,152	48	11,544,356	$4.13 \cdot 10^{-10}$	16h 47m	$2.5 \cdot 10^{-9}s$
6	17	2,097,152	47	18,657,901	$2.56 \cdot 10^{-10}$	1d 2h 53m	$2.47 \cdot 10^{-9}s$
7	8	16,777,216	49	2,433,140	$2.45 \cdot 10^{-10}$	1d 18h 39m	$3.76 \cdot 10^{-9}s$
7	11	16,777,216	48	1,375,407	$4.33 \cdot 10^{-10}$	1d 1h 45m	$4.02 \cdot 10^{-9}s$

Table C.23.: Results for the Curve C40 - Genus 40 over $\mathbb{F}_{2^3}$

Genus 60 over $\mathbb{F}_{2^2}$

$$\begin{aligned}&\text{C60, } \mathbb{F}_{2^2} = \mathbb{F}_2[w]/(w^2 + w + 1)\\ &f(x) = x^{121} + x^{22} + w^2 + w + 1\\ &h(x) = x^{60} + 1\end{aligned}$$

Table C.24.: Curve C60 - Genus 60 over $\mathbb{F}_{2^2}$

M	P_{ONCE}	Array Size	η_{max}	Sieve Polyn.	$\mathcal{P}(\eta_{max}, \mathcal{F})$	Core Time	Divisor Time
8	10	262,144	71	16,823,869	$2.67 \cdot 10^{-9}$	17h 54m	$1.46 \cdot 10^{-8}$s
8	13	262,144	69	3,737,304	$1.02 \cdot 10^{-8}$	8h 4m	$2.96 \cdot 10^{-8}$s
8	17	262,144	70	29,939,872	$1.27 \cdot 10^{-9}$	1d 3h 1m	$1.24 \cdot 10^{-8}$s
9	7	1,048,576	72	5,837,011	$1.63 \cdot 10^{-9}$	19h 37m	$1.15 \cdot 10^{-8}$s
9	10	1,048,576	71	4,182,328	$2.28 \cdot 10^{-9}$	9h 58m	$8.18 \cdot 10^{-9}$s
9	17	1,048,576	70	17,550,430	$5.43 \cdot 10^{-10}$	1d 10h 2m	$6.66 \cdot 10^{-9}$s
10	6	4,194,304	73	2,289,960	$1.04 \cdot 10^{-9}$	20h 13m	$7.58 \cdot 10^{-9}$s
10	7	4,194,304	72	1,465,337	$1.63 \cdot 10^{-9}$	12h 59m	$7.6 \cdot 10^{-9}$s
10	8	4,194,304	73	2,319,752	$1.03 \cdot 10^{-9}$	18h 51m	$6.97 \cdot 10^{-9}$s
10	10	4,194,304	71	1,023,859	$2.33 \cdot 10^{-9}$	9h 2m	$7.57 \cdot 10^{-9}$s
11	6	16,777,216	73	550,997	$1.08 \cdot 10^{-9}$	1d 2h 22m	$1.03 \cdot 10^{-8}$s
11	7	16,777,216	72	355,172	$1.68 \cdot 10^{-9}$	17h 48m	$1.07 \cdot 10^{-8}$s
11	8	16,777,216	73	549,184	$1.09 \cdot 10^{-9}$	1d 1h 37m	$1.0 \cdot 10^{-8}$s
12	6	67,108,864	73	141,413	$1.05 \cdot 10^{-9}$	1d 0h 9m	$9.16 \cdot 10^{-9}$
12	8	67,108,864	73	134,004	$1.11 \cdot 10^{-9}$	1d 2h 4m	$1.04 \cdot 10^{-8}$

Table C.26.: Results for the Curve C60 - Genus 60 over $\mathbb{F}_{2^2}$

Genus 120 over $\mathbb{F}_2$

C120, $\mathbb{F}_2$

$f(x) = x^{241} + x^{22} + w^2 + w + 1$

$h(x) = x^{120} + 1$

Table C.27.: Curve C120 - Genus 120 over $\mathbb{F}_2$

M	P_{ONCE}	Array Size	η_{max}	Sieve Polyn.	$\mathcal{P}(\eta_{max}, \mathcal{F})$	Core time	Divisor Time
18	6	524,288	139	1,441,098	$1.32 \cdot 10^{-8}$	1d 3h 41m	$1.32 \cdot 10^{-7}s$
18	10	524,288	141	2,537,769	$7.52 \cdot 10^{-9}$	14h 11m	$3.84 \cdot 10^{-8}s$
18	11	524,288	142	3,822,808	$4.99 \cdot 10^{-9}$	16h 55m	$3.04 \cdot 10^{-8}s$
18	17	524,288	139	52,061,392	$3.66 \cdot 10^{-10}$	6d 17h 44m	$2.13 \cdot 10^{-8}s$
19	10	1,048,576	141	1,283,454	$7.43 \cdot 10^{-9}$	9h 59m	$2.67 \cdot 10^{-8}s$
19	11	1,048,576	142	1,812,481	$5.26 \cdot 10^{-9}$	11h 52m	$2.25 \cdot 10^{-8}s$
20	10	2,097,152	141	648,919	$7.35 \cdot 10^{-9}$	7h 50m	$2.07 \cdot 10^{-8}s$
20	11	2,097,152	142	914,490	$5.21 \cdot 10^{-9}$	9h 31m	$1.78 \cdot 10^{-8}s$
20	20	2,097,152	141	411,017,070	$1.16 \cdot 10^{-11}$	106d 13h 15m	$1.07 \cdot 10^{-8}s$
21	7	4,194,304	143	488,891	$4.88 \cdot 10^{-9}$	13h 16m	$2.33 \cdot 10^{-8}s$
21	9	4,194,304	142	412,509	$5.78 \cdot 10^{-9}$	10h 4m	$2.09 \cdot 10^{-8}s$
21	11	4,194,304	142	466,362	$5.11 \cdot 10^{-9}$	10h 36m	$1.95 \cdot 10^{-8}s$
21	14	4,194,304	143	488,558	$4.88 \cdot 10^{-9}$	11h 15m	$1.98 \cdot 10^{-8}s$
22	7	8,388,608	143	247,092	$4.82 \cdot 10^{-9}$	15h 32m	$2.70 \cdot 10^{-8}s$
22	14	8,388,608	143	349,118	$3.41 \cdot 10^{-9}$	18h 1m	$2.21 \cdot 10^{-8}s$
23	6	16,777,216	145	199,196	$2.99 \cdot 10^{-9}$	1d 0h 56m	$2.69 \cdot 10^{-8}s$
23	8	16,777,216	145	195,658	$3.05 \cdot 10^{-9}$	22h 36m	$2.48 \cdot 10^{-8}s$
23	12	16,777,216	145	255,707	$2.33 \cdot 10^{-9}$	1d 2h 8m	$2.19 \cdot 10^{-8}s$
24	6	33,554,432	145	100,387	$2.97 \cdot 10^{-9}$	23h 17m	$2.49 \cdot 10^{-8}s$
24	8	33,554,432	145	97,484	$3.06 \cdot 10^{-9}$	1d 0h 13m	$2.67 \cdot 10^{-8}s$
24	12	33,554,432	145	120,721	$2.47 \cdot 10^{-9}$	1d 1h 50m	$2.30 \cdot 10^{-8}s$

Table C.29.: Results for the Curve C120 - Genus 120 over $\mathbb{F}_2$

C.3. Test Curves and Detailed Results for Section 4.2.4

Genus 4 over $\mathbb{F}_{1,073,741,651}$

G4, $\mathbb{F}_{1,073,741,651}$, where
$1,073,741,651 = 2^{30} - 173$
$y^2 = x^9 + x + 8$

Table C.30.: Curve G4 - Genus 4 over $\mathbb{F}_{1,073,741,651}$

Algorithm	Single-Large-Prime		Large-Prime Graph	
Norms in $\mathcal{F}$	5,782,784	1,086,976	5,782,784	1,086,976
Sieve Polynomials	146,974,904	n.a.	9,685,328	1,448,143,728
Found Full Relations	191,904	n.a.	12,674	451
Found SLP Relations	82,425,226	n.a.	5,770,332	1,088,160
Relations from SLP	5,590,880	n.a.	25,812	1,625
Found DLP Relations			1,062,188,904	1,071,566,516
Relations from LPG			5,744,298	1,084,900
Relations included to Graph			5,443,930	931,864
Avg. Relation Weight	15.60	n.a.	44.34	63.28
Relation Collection Time (s)	2,861,219	n.a.	194,939	5,517,712
Large-Prime Graph Build Time (s)			240	168
Total Time (s)	2,861,219	n.a.	195,179	5,517,880

Table C.31.: Results for the Curve G4 - Genus 4 over $\mathbb{F}_{1,073,741,651}$

Genus 5 over $\mathbb{F}_{16,777,289}$

G5, $\mathbb{F}_{16,777,289}$, where
$16,777,289 = 2^{24} + 73$
$y^2 = x^{11} + 2$

Table C.32.: Curve G5 - Genus 5 over $\mathbb{F}_{16,777,289}$

Algorithm	Single-Large-Prime		Large-Prime Graph	
Norms in $\mathcal{F}$	542,208	170,240	542,208	170,240
Sieve Polynomials	20,356,000	3,566,849,856	2,955,744	289,140,032
Found Full Relations	34,336	5,784	5,032	448
Found SLP Relations	2,497,791	1,517,619	436,642	135,820
Relations from SLP	507,623	164,456	12,039	1289
Found DLP Relations			15,798,521	16,471,229
Relations from LPG			525,137	168,503
Relations included to Graph			496,746	128,838
Avg. Relation Weight	18.66	18.92	44.29	60.42
Relation Collection Time (s)	35,385	2,241,808	5,259	186,545
Large-Prime Graph Build Time (s)			12	5
Total Time (s)	35,385	2,241,808	5,271	186,550

Table C.33.: Results for the Curve G5 - Genus 5 over $\mathbb{F}_{16,777,289}$

Genus 6 over $\mathbb{F}_{1,048,627}$

G6, $\mathbb{F}_{1,048,627}$, where
$1,048,627 = 2^{20} + 51$
$y^2 = x^{13} + x^8 + x^5 + x^2 + x + 1$

Table C.34.: Curve G6 - Genus 6 over $\mathbb{F}_{1,048,627}$

Algorithm	Single-Large-Prime		Large-Prime Graph	
Norms in $\mathcal{F}$	98,410	41,188	98,410	41,188
Sieve Polynomials	5,936,800	652,891,008	1,316,128	88,501,088
Found Full Relations	10,070	2,674	2,222	328
Found SLP Relations	218,323	166,334	68,200	27,526
Relations from SLP	88,340	38,514	5,236	800
Found DLP Relations			882,032	979,913
Relations from LPG			90,952	40,060
Relations included to Graph			80,428	28,271
Avg. Relation Weight	20.97	21.86	44.12	58.08
Total Time (s)	1,069	93,515	289	13,419

Table C.35.: Results for the Curve G6 - Genus 6 over $\mathbb{F}_{1,048,627}$

Genus 7 over $\mathbb{F}_{144,439}$

G7, $\mathbb{F}_{144,439}$, where $144,439 = 2^{17} + 13,367 \approx 2^{17.14}$ $y^2 = x^{15} + x + 6$

Table C.36.: Curve G7 - Genus 7 over $\mathbb{F}_{144,439}$

Algorithm	Single-Large-Prime		Large-Prime Graph	
Norms in $\mathcal{F}$	25,071	13,342	25,071	13,342
Sieve Polynomials	4,768,832	245,313,216	1,395,136	46,912,576
Found Full Relations	3,437	1,258	1,032	234
Found SLP Relations	31,926	29,587	15,630	7,948
Relations from SLP	21,634	12,084	2,367	554
Found DLP Relations			103,738	123,149
Relations from LPG			21,672	12,554
Relations included to Graph			14,439	8,231
Avg. Relation Weight	22.94	24.03	46.13	57.49
Total Time (s)	591	29,609	197	6,129

Table C.37.: Results for the Curve G7 - Genus 7 over $\mathbb{F}_{144,439}$

Genus 8 over $\mathbb{F}_{32,771}$

G8, $\mathbb{F}_{32,771}$, where
$32,771 = 2^{15} + 3$
$y^2 = x^{17} + x + 6$

Table C.38.: Curve G8 - Genus 8 over $\mathbb{F}_{32,771}$

Algorithm	Single-Large-Prime		Large-Prime Graph	
Norms in $\mathcal{F}$	10,496	5,888	10,496	5,888
Sieve Polynomials	1,562,112	84,763,264	796,160	22,982,144
Found Full Relations	2,675	877	1,280	201
Found SLP Relations	5,409	7,207	6,780	3,340
Relations from SLP	7,821	5,011	2,693	525
Found DLP Relations			15,495	23,543
Relations from LPG			6,523	5,162
Relations included to Graph			4,638	2,982
Avg. Relation Weight	23.82	25.89	41.51	56.04
Total Time (s)	187	10,041	112	2,859

Table C.39.: Results for the Curve G8 - Genus 8 over $\mathbb{F}_{32,771}$

Genus 8 over $\mathbb{F}_{8,388,673}$

G'8, $\mathbb{F}_{8,388,673}$, where
$8,388,673 = 2^{23} + 65$
$y^2 = x^{17} + x + 28$

Table C.40.: Curve G'8 - Genus 8 over $\mathbb{F}_{8,388,673}$

Algorithm	Large-Prime Graph	
Norms in $\mathcal{F}$	1,848,292	721,023
Sieve Polynomials	6,513,664	2,881,397,248
Found Full Relations	94,404	8,752
Found SLP Relations	1,074,813	377,768
Relations from SLP	212,400	19,963
Found DLP Relations	5,465,687	7,289,882
Relations from LPG	1,541,488	692,308
Relations included to Graph	1,498,426	354,619
Avg. Relation Weight	51.03	73.98
Total Time (s)	31,270	5,777,999

Table C.41.: Results for the Curve G'8 - Genus 8 over $\mathbb{F}_{8,388,673}$