A Study of Hyperelliptic-Curve Cryptography

Thesis submitted in partial fulfillment
of the requirements for the award of the degree

of

Master of Science

by

Anindya Ganguly

Under the supervision of

Prof. Abhijit Das
Prof. Dipanwita Roy Chowdhury

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur- 721302, India

Date: _____

Approval of the Viva-Voce Board

Certified that the thesis entitled **"A Study of Hyperelliptic-Curve Cryptography"** submitted by **Mr. Anindya Ganguly** to the Indian Institute of Technology Kharagpur, for the award of the degree of Master of Science has been accepted by the external examiners and that the student has successfully defended the thesis in the viva-voce examination held today.

_____ _____ _____

(Member of the DAC) (Member of the DAC) (Member of the DAC)

_____ _____

(Supervisor) (Joint Supervisor)

_____ _____

(External Examiner) (Chairman)

Certificate

This is to certify that the thesis entitled **"A Study of Hyperelliptic-Curve Cryptography"** submitted by **Mr. Anindya Ganguly (16CS72P01)** to the Indian Institute of Technology Kharagpur, is a record of bonafide research work carried under my supervision and is worthy of consideration for the award of the Master of Science of the Institute.

Date:
Place: IIT Kharagpur

Abhijit Das

Date:
Place: IIT Kharagpur

Dipanwita Roy Chowdhury

To

Parents

Declaration of Authorship

I, Mr. Anindya Ganguly, declare that this thesis titled **"A Study of Hyperelliptic-Curve Cryptography"** and the work presented in it are my own. I confirm that:

- The work contained in the thesis is original and has been done by myself under the general supervision of my supervisors.
- This work was done wholly or mainly while in candidature for a research degree at this Institute.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this Institute or any other institution or university, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Name: ANINDYA GANGULY, 16CS72P01

Date:

Acknowledgement

The path toward this thesis has been serpentine. Having a background in Mathematics, I found it challenging to pursue the degree at the Department of Computer Science and Engineering. To complete this endeavour, I am very much thankful to some special persons who challenged, supported, and stuck with me along the way.

This journey is not possible without the constant motivation and endless support of my parents. I need to thanks them for ensuring the freedom to choose my career. My mother Nandita Ganguly has been my backbone. I would also like to thank my relatives who supported me when my mother got hospitalized. I especially acknowledge the support of my sister Gopa and her husband Tanmoy in my life. Without their help, it is not possible to dream about higher studies.

It is a great pleasure and acknowledge my deepest thank and gratitude to Prof. Abhijit Das and Prof. Dipanwita Roy Chowdhury from the Department of Computer Science and Engineering, IIT Kharagpur, for suggesting the topic and for their kind supervision. Their immense knowledge and plentiful experience have encouraged me all the time during my academic research and daily life. I have been extremely lucky to have supervisors who cared so much about my work and who responded to my questions and queries so promptly.

Having the guidance of Prof. Das is a blessing of God. At many stages in the course of this research work, I benefited from his advice, particularly when exploring new ideas. His positive outlook and confidence in my research work inspired me and gave me confidence. His excellent teaching style encourages me to learn a lot of subjects from the very basics. His courses "Computational Number Theory" and "Foundations of Cryptography" helped me to understand the basics of the research topic. His careful editing contributed enormously to the production of the thesis. Apart from a great supervisor, he is an unparalleled human being. He supports and motivates me a lot during the up and down in the research life. I have learned a lot of things from him apart from academic research. Nights spent at B-217 along with lab members are the most beautiful memories in my life. I am always indebted to Prof. Roy Chowdhury for her treasured support which is influential in shaping my concepts.

I would like to thank my Departmental Academic Committee Members Prof. Soumya Kanti Ghosh, Prof. Shamik Sural, Prof. Pabitra Mitra from the Department of Computer Science and Engineering, IIT Kharagpur. Their valuable suggestions played a crucial role in the entire research work. A special thank to the Faculty advisor (for MS) Prof. Chittaranjan Mondal. I would like to extend my sincere thank to the former HOD Prof. Sudeshna Sarkar and the current HOD Prof. Dipanwita Roy Chowdhury for providing an outstanding research environment and endless facility in the department. I would also like to thank all professors and staff of our department who helped me in various ways during the research journey.

I am thankful to SAC-ISRO, Ahmedabad, and DST SERB for funding the research grant throughout the journey. It is an honor to me to collaborate with Mr. Deval Mehta from Space Application Center, ISRO, Ahmedabad.

I would like to express my deepest gratitude to Prof. Sourav Mukhopadhyay (from the Department of Mathematics, IIT Kharagpur), who introduced me to the field of cryptography, especially public-key cryptography. Besides, I am extremely grateful to Prof. Ramkr-

ishna Nanduri and Prof. Mousumi Mondal (both from the Department of Mathematics, IIT Kharagpur) for clearing doubts related to Algebraic Geometry. It helped me to write the chapter on mathematical backgrounds in the thesis.

My seniors played a key role in the journey of research. I would like to thank Dr. Bidhan Chandra Sardar (IIT Ropar), Dr. Dhiman Saha (IIT Bhilai), Dr. Raju Hazari (NIT Kalikat), and Dr. Sabyasachi Karati (NISER Bhubaneswar) for motivating me in the research journey. I am indebted to Karati-da for helping me in the field of curve-based cryptography.

I also take this opportunity to express my thanks to my lab seniors Swapan-da, Souvik-da, Debranjan-da, Ghosal-da, and Amit-da for constant helping and encouraging throughout the journey. Souvik-da and Pritam helped me a lot when I got stuck with long codes. Special thank also go to Bijoy, Rashid, and Rahul for helping me in the critical time.

Life at KGP is colouful with a strong friend circle. I am lucky to have friends like Ashok, Bubai, Mainak, Nitin, Prem, Ravi, Sudipta, Tapas, Punit, and many others. We all together spent a lot of good times in KGP. I connected myself with societies like Druheen and Boikalik. I would also like to thank all the members of both the societies for a positive mindset. Thanks to the entire KGP community too for providing a fabulous environment.

Last but not least, I would like to thank IIT Kharagpur for the studentship that allows me to conduct the thesis.

Take up one idea. Make that one idea your life - think of it, dream of it, live on that idea. Let the brain, muscles, nerves, every part of your body, be full of that idea, and just leave every other idea alone. This is the way to success.

—Swami Vivekananda

Abstract

Elliptic curves are widely used in cryptographic protocols. Hyperelliptic curves, a generalization of elliptic curves, also hold the promise of large-scale applications. Hyperelliptic curves need half field sizes compared to elliptic curves. However, the arithmetic of hyperelliptic curves is slower than that of elliptic curves. Hyperelliptic curves defined over subfields can accelerate point counting and Jacobian arithmetic.

In this dissertation, a new family of hyperelliptic curves is proposed. The point-counting algorithm for this family is detailed. Existing algorithms for Jacobian arithmetic are compared. Implementation details of the finite-field arithmetic is also presented. The performance for the proposed family is analyzed. Finally, the discrete logarithm problem defined over the Jacobian of the curves is scrutinized.

ElGamal encryption is a popular public-key cryptographic primitive. This can be implemented using the proposed family of hyperelliptic curves. For this purpose, an encoding map is proposed. It is proved that the encoding map is well distributed and efficiently computable. It is established that this variant of ElGamal encryption is no less secure than original ElGamal encryption.

Keywords Hyperelliptic-Curve Cryptography (HECC), Subfield curves, Point-Counting Algorithms, Jacobian Arithmetic, ElGamal Encryption, Message Encoding, Message Indistinguishability, Computational Diffie–Hellman Problem, Discrete Logarithm Problem.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ABBREVIATIONS

DLP Discrete Logarithm Problem

DH Diffie–Hellman

EC Elliptic Curve

ECC Elliptic-Curve Cryptography

HEC Hyperelliptic Curve

HECC Hyperelliptic-Curve Cryptography

SCA Side-Channel Attack

ECDLP Elliptic Curve Discrete Logarithm Problem

HECDLP Hyperelliptic Curve Discrete Logarithm Problem

GF Galois Field

HECDSA Hyperelliptic Curve Digital Signature Algorithm

DDH Decisional Diffie–Hellman

CDH Computational Diffie–Hellman

IND-CPA Indistinguishability against Chosen Plaintext Attack

IND-CCA Indistinguishability against Chosen Ciphertext Attack

OW-CCA One-way Chosen Ciphertext Attack

DTDLA Delay-target Discrete Logarithm Assumption

SGKEA Strong Generalized Knowledge of Exponent Assumption

NOTATIONS AND SYMBOLS

G	a cryptographically suitable group
l	security level
$\mathcal{C}$	hyperellitic curve
g	genus of a curve
p	a prime number
(mod)	modular reduction
$\mathbb{Z}$	set of integers
$\mathbb{Z}_p$	set of integers modulo prime p
$\mathbb{F}_p, \mathbb{F}_q$	finite fields
K	a field
$\overline{K}$	algebraic closure of the field K
GF(p)	Galois field
q	power of a prime ($q = p^5$)
GF(q)	same as $\mathbb{F}_{p^d}$
$\mathbb{J}$	Jacobian of a curve
$\mathbb{J}_p$	Jacobian of a curve defined over $\mathbb{F}_p$
$\mathbb{J}_q$	Jacobian of a curve defined over extension field $\mathbb{F}_q$
$\lvert\mathbb{J}_p\rvert, \lvert\mathbb{J}_q\rvert$	Cardinalities of $\mathbb{J}_p$ and $\mathbb{J}_q$
$\mathbb{A}_K^n$	n-dimensional projective space over K
D	divisor
gcd	greatest common divisor
$\mathcal{O}$	point at infinity
$O()$	big-Oh notation
$Z_{\mathcal{C}}()$	zeta function of curve $\mathcal{C}$

CHAPTER

ONE

INTRODUCTION

1.1 Introduction

Cryptographic primitives are extensively used to secure the modern digital era. Traditionally, cryptography deals with three principal characters: the sender (Alice), the receiver (Bob), and the attacker (Malice). Alice sends an encrypted message via an insecure channel to Bob. Bob decrypts the ciphertext using a predefined secret key, and recovers the message. Malice listens to the channel, and reads the encrypted message. His goal is to figure out the plaintext message from the ciphertext. Cryptographic protocols should ensure that Malice can succeed only with negligible probability.

Of late, digital communications are expanding exponentially. Currently, social media is connecting 2.65 billion people. The number of mobile wallet users is increasing by 140 millions per year. Therefore, security issues are of high concern. An important purpose of cryptography is to enhance authenticity and privacy of digital data. In addition to commercial and online uses like debit and credit cards and net-banking, cryptographic protocols are used by military and government organizations. Hard disks too use encryption to protect their contents. Consequently, the construction of suitable secure, efficient, and lightweight cryptographic primitives is an important and practical domain of research.

Public-key cryptography originates from the seminal discoveries of the Diffie–Hellman key-agreement protocol [11] and the RSA cryptosystem [59]. Since then, quite a few cryptographic schemes are proposed that are based on the difficulty of factoring large integers or computing discrete logarithms in large groups. Protocols based on the hardness of DLP is defined over a cyclic group G, like a multiplicative group of integers modulo n.

Discrete Logarithm Problem (DLP): Suppose $(G, \cdot)$ is an Abelian group. Given $u, v \in G$, find r (if it exists) such that $u^r = v$. Such a group G is said to be cryptographically suitable if

- the group elements are compact,
- the group operation is fast and efficient,
- the order of the group is large and can be computed efficiently,
- DLP is computationally hard in G.

Two popular families of groups are the multiplicative groups of finite fields, and the sets of rational points on elliptic curves defined over finite fields. Proposed by Koblitz [40] and Miller [68], elliptic curves have been used extensively in several cryptographic protocols. The main advantage of ECC is that it provides the same level of security as the traditional multiplicative groups (like $\mathbb{F}_q^*$), but the elliptic-curve groups are much smaller than finite-field groups.

Later in 1989, Koblitz [41] proposes the use of hyperelliptic curves over finite fields for cryptographic purposes. However, these curves are studied less extensively by the cryptographic community than the schemes based on RSA, finite-field, and elliptic-curve discrete logarithms. Genus-two hyperelliptic curves offer the same level of security as elliptic curves, but with half field sizes. To achieve 128 bits of security, elliptic curves need 256-bit fields, whereas hyperelliptic curves require only 128-bit fields. The following table shows that curve-based cryptography needs less bits compared to DH or RSA to achieve the same levels of security.

DH or RSA	ECC	HEC
512	112	56
1024	160	80
2048	224	112
3072	256	128
7680	384	192
15360	512	256

Table 1.1: NIST Guidelines for Key Size

1.2 Motivation and Objectives

Genus-one hyperelliptic curves are known as elliptic curves. Higher-genus curves ($g > 1$) are also considered for cryptographic purposes. For hyperelliptic curves of genus $g > 1$, the Jacobians of the curves provide the underlying Abelian group structure. For large-genus hyperelliptic curves, there exist algorithms faster than the generic square-root methods and having subexponential running times, to solve the DLP. But for $g \leq 3$, no such subexponential algorithm is known.

In hyperelliptic-curve cryptography, generating suitable cryptographically secure curves over finite fields is an important issue. Literature suggests that point-counting algorithms over large prime finite fields are not very efficient. However, subfield hyperelliptic curves are especially attractive from an efficiency point of view. Moreover, subfield hyperelliptic curves offer faster Jacobian arithmetic than hyperelliptic curves over large prime fields.

Furukawa et al. [20] propose an algorithm for the order computation of the Jacobian of the hyperelliptic curve of the form $y^2 = x^5 + ax$ over large prime fields. They also present the new family $y^2 = x^5 + a$. Satoh [62] develops a probabilistic polynomial-time algorithm to identify whether the curve $y^2 = x^5 + ax^3 + bx$ is suitable, that is, whether the order of the Jacobian has a large prime divisor. Buhler and Koblitz [5] propose an algorithm for particular types of curves $y^2 + y = x^n$ over $\mathbb{F}_p$, where n is odd, and p is any prime with $n|(p-1)$. To the best of our knowledge, no families of subfield hyperelliptic curves are explicitly proposed in the literature.

There exist software implementations of elliptic- and hyperelliptic-curve cryptography. Gaudry [30] writes a library for finite-field arithmetic. The $\mathtt{m}_p\mathbb{F}_q$ library is practically used for curve-based public-key cryptography. A HECC software implementation is done by Pelzl et al. [55]. They put execution times in tabulated manner for curves of genus two and three. Avanzi [3] implements a prime-field library nuMONGO which includes elliptic- and hyperelliptic-curve arithmetic. These implementations use large prime fields. We are not aware of any reported implementation that includes subfield curves for cryptographic purposes.

An efficient addition algorithm for the divisor group is required for the hyperelliptic curves. Cantor proposes a fast algorithm for addition using Mumford representation of divisors. This addition of the divisor group in the hyperelliptic curve is not so efficient as elliptic-curve point addition. The performance gap was narrowed by Harley [32]. Later, Lange provides an explicit version of Harley's formula [42]. Lange's explicit version gives a potentially powerful speedup for hyperelliptic-curve addition. To enhance the performance further, Lange [44] also proposes an inversion-free addition algorithm. Several researchers try to improve the performance of divisor-class arithmetic.

In this backdrop, our work is mostly motivated by the need of narrowing of the performance gap between elliptic- and hyperelliptic-curve cryptography. The process involves looking for new families of hyperelliptic curves and going for and tuning software implementations of the Jacobian arithmetic. Eventually, the study should investigate the effectiveness of using these curves in practical cryptographic schemes.

1.3 Contributions of the Dissertation

Let $\mathcal{C}$ be a hyperelliptic curve of genus two defined over a finite field $\mathbb{F}_q$. A suitably large subgroup G of the Jacobian $\mathbb{J}_q$ is to be used to build cryptographic schemes. For cryptographic reasons, the group order n should be a prime. The bit length of n is dictated by the security level l. Since the square-root attacks are the only attacks known for hyperelliptic curves of genus two, we take $l \approx |n|/2$. Since 64-bit security is not considered safe given the available computing powers, and we require $l \geq 80$. Security level $l = 128$ is prescribed for long-term use. We target achieving several security levels depending upon the needs of the cryptographic applications. More specifically, we take $l = 80, 96, 112, 128$.

For achieving l-bit security, we need a field $\mathbb{F}_q$ of bit size $|q| \geq l$. Moreover, the size of the Jacobian $\mathbb{J}_q$ should be a prime (around $2l$-bits). One option is to take an l-bit prime as q. But point-counting algorithms over such large prime fields are mathematically complicated and practically inefficient.

To work around this problem, we have devised a slightly modified approach. We use quintic extensions. For $l \leq 128$, this prime p fits in a 32-bit unsigned integer. We generate a curve over $\mathbb{F}_p$, and compute the order of $\mathbb{J}_p$. Since p is now small, simple and practical point-counting algorithms can be used. We then consider the quintic extension $\mathbb{F}_q = \mathbb{F}_{p^5}$. The curve $\mathcal{C}$ is naturally defined over $\mathbb{F}_q$. Moreover, given the group size $|\mathbb{J}_p|$, the group size $|\mathbb{J}_q|$ can be calculated using simple formulas. We require $n = |\mathbb{J}_q|/|\mathbb{J}_p|$ to be a prime. This approach helps us generate many suitable curves of security level l fairly quickly. On the flip side, we now have to work in a field of bit size $|q| = 5l/4$. For efficiency reasons, we take $\mathcal{C}$ of the form

$$\mathcal{C} : y^2 = x^5 + x + a, \quad a \in \mathbb{F}_p.$$

To compute the orders of the curves over $\mathbb{F}_p$, the baby-step-giant-step method is used. This algorithm may fail on a few occasions, but given that we have a large domain for varying a, many curves with known orders can be generated fairly efficiently. We have $|\mathbb{J}_p| \approx p^2$.

Since $\mathbb{F}_p \subseteq \mathbb{F}_q$, $\mathcal{C}$ is naturally defined over the extension field $\mathbb{F}_q = \mathbb{F}_{p^5}$, and the order of $\mathbb{J}_q$ can be computed by the Newton–Girard formulas. We have $|\mathbb{J}_p| \approx q^2 = p^{10}$. Since $\mathbb{J}_p$ is a subgroup of $\mathbb{J}_q$, the order of $\mathbb{J}_p$ must divide the order of $\mathbb{J}_q$. We use the cofactor

$$n = \frac{|\mathbb{J}_q|}{|\mathbb{J}_p|}.$$

If n is prime, $\mathbb{J}_q$ contains a subgroup G of this order. The bit length of n is

$$|n| = |\mathbb{J}_q| - |\mathbb{J}_p| \approx (10-2)|p| = 8|p|,$$

that is, the security level is $|n|/2 \approx 4|p| = l$, as planned.

Indeed, we have $|n| = 2l$ or $|n| = 2l+1$ if $p \approx 2^{l/4}$. In terms of efficiency of Jacobian arithmetic over $\mathbb{F}_q$, there is hardly any difference in the running times between these two cases. However, for index arithmetic (modulo n), the case $|n| = 2l+1$ introduces some inefficiency. If we use 32-bit words to pack fragments of multiple-precision integers, then for the stated values of l, we need an extra word compared to the case $|n| = 2l$. This may be an issue for some cryptographic algorithms.

Being subfield curves, the members of this family are easy to generate. Although slightly slower than elliptic curves at the same security level, hyperelliptic curves of our family exhibit performance comparable to widely used hyperelliptic curves over prime fields.

A map to encode messages to divisors is developed for the ElGamal encryption scheme. This map is well-distributed, and renders the adapted encryption scheme the same security guarantees as in the original ElGamal encryption.

1.4 Organization of the Thesis

The rest of this thesis is organized as follows.

Chapter 2 deals with the mathematical preliminaries about hyperelliptic curves. More precisely, the concepts of divisors, divisor classes, and the Jacobians are introduced. The discussion also sets up the notations we use in the rest of the thesis.

Chapter 3 starts by pointing out the difficulties for computing the orders of Jacobians. Subsequently, an algorithm is described for computing the order of Jacobians for subfield curves. It also deals with Jacobian arithmetic. Here, Cantor's, Harley's and Lange's algorithms are introduced for computing the sum of two reduced divisors. Inversion-free divisor-class addition is also described. The chapter presents the performance analysis of the proposed subfield curves alongside elliptic curves. The use of Lange's formulas and inversion-free arithmetic leads to comparable performances between subfield hyperelliptic curves and elliptic curves.

Chapter 5 is a discussion of the use of our curves in cryptographic schemes. An adaptation of the ElGamal scheme to the case of our curves raises some security issues which are identified in this chapter. A security analysis is also presented for the adapted scheme.

Chapter 6 concludes the dissertation after summarizing the reported work and identifying some directions for further research.

Appendix A provides a list of several cryptographically suitable hyperelliptic curves at different security levels.

CHAPTER

TWO

MATHEMATICAL BACKGROUND

2.1 Introduction

The chapter introduces the basic mathematics used in our work. It starts with an exposition to hyperelliptic curves. The defining equation for a hyperelliptic curve is followed by the construction of the Jacobian for a curve. The difficulty of the discrete logarithm problem over hyperelliptic curves is also discussed. Detailed coverage of hyperelliptic curves can be found in [7, 50, 22, 9]. For finite fields, the textbook [46] is referred.

2.2 Plane Curves

The section focuses on a few elementary notions from algebraic geometry [19, 33]. The next section specializes to hyperelliptic curves.

2.2.1 Affine and Projective Coordinates

Let K be a field (not necessarily finite), and its algebraic closure is denoted by $\overline{K}$. The set of all n-tuples $(x_1, x_2, \ldots, x_n)$ over K^n is denoted by $\mathbb{A}_K^n$ or $\mathbb{A}_n$, and is called the *n-dimensional affine space* over K. An element from $\mathbb{A}_K^n$ is called a *point*, and each x_i is a *coordinate* of the point. The *affine plane* over K is the affine space $\mathbb{A}_K^2$. The *affine line* over K is $\mathbb{A}_K^1$. Let $F \in K[x_1, x_2, \cdots, x_n]$ be a polynomial. The *set of zeros* of F is defined as

$$Z(F) = \{P \in \mathbb{A}_K^n \mid F(P) = 0\}.$$

The set of zeros for a non-constant polynomial F is known as a *hypersurface* defined by F. An *affine plane curve* is a hypersurface in $\mathbb{A}_K^2$. An affine plane curve $\mathcal{C}$ is defined by a polynomial $F(x_1, x_2) \in K[x_1, x_2]$, and is denoted as $\mathcal{C} : F(x_1, x_2) = 0$. $\mathcal{C}(K)$ denotes the *set of K-rational points* on the curve $\mathcal{C} : F(x_1, x_2) = 0$, and is defined by

$$\mathcal{C}(K) = \{(x_1, x_2) \mid F(x_1, x_2) = 0\}.$$

We define a relation $\sim$ on the affine space $\mathbb{A}_K^{n+1} - \{0, 0, \ldots, 0\}$ of dimension $n+1$ by $(x_0, x_1, \ldots, x_n) \sim (y_0, y_1, \ldots, y_n)$ if and only if there exist $\lambda \in \overline{K} - \{0\}$ such that for all i, we have $x_i = \lambda y_i$. It is easy to see that $\sim$ is an equivalence relation on $\mathbb{A}_K^n - \{(0, 0, \cdots, 0)\}$. The set of all equivalence classes defined by $\sim$ is said to be the *n-dimensional projective space* $\mathbb{P}_K^n$ over K. For $n = 2$, $\mathbb{P}_K^2$ is known as the *projective plane* over K. An element $P \in \mathbb{P}_K^n$ is known as a *point*. The set of homogeneous coordinates for P is the equivalence class of the point P.

Consider the affine plane curve $\mathcal{C} : F(x_1, x_2) = 0$ over the field K with $d = \deg(F)$. Then, $F_h(x_1, x_2, x_3) = x_3^d F(x_1, x_2)$ is a degree-d homogeneous polynomial in $K[x_1, x_2, x_3]$. The polynomial F_h is known as the *homogenization* of F. Putting $x_3 = 1$ converts it into the original polynomial $F(x_1, x_2)$, that is $F_h(x_1, x_2, x_3) = F(x_1, x_2)$. F is known as the *dehomogenization* of the homogeneous polynomial F_h.

A *projective plane curve* $\mathcal{C}$ over a field K is defined by a homogeneous polynomial $H(x_1, x_2, x_3)$, and is denoted by $\mathcal{C} : H(x_1, x_2, x_3) = 0$. The points satisfying $H(x_1, x_2, x_3) = 0$ form the set $\mathcal{C}(K)$ of *K-rational points*.

2.2.2 Coordinate Ring

Let $\mathcal{C} : F(x_1, x_2) = 0$ be a curve defined over K. We assume that F is an irreducible polynomial over K. Although we work with the set $\mathcal{C}(K)$ of K-rational points of the corresponding homogeneous curve, we write the affine equation for notational simplicity. The solutions of $\mathcal{C}$ in the affine plane $\mathbb{A}_K^2$ are the *finite points* on the curve. Other points of the projective plane that lie on the curve are called *points at infinity* on the curve.

A point $P = [a, b, c]$ is called a *regular* or *smooth* or *non-singular* point of $\mathcal{C}$ if P satisfies the following properties.

- If P is a finite point (that is, if $c \neq 0$), then the partial derivatives $\frac{\partial F}{\partial x_1}$ and $\frac{\partial F}{\partial x_2}$ must not vanish simultaneously at $(\frac{a}{c}, \frac{b}{c})$.
- If P is a point at infinity (that is, if $c = 0$), then we must have $a \neq 0$ or $b \neq 0$. Take $a \neq 0$ (the other case can be handled analogously). Choose the polynomial $G(x_2, x_3) = F_h(1, x_2, x_3) \in K[x_2, x_3]$. Then, $(\frac{b}{a}, 0)$ is a finite point on $\mathcal{C}' : G(x_2, x_3) = 0$. The partial derivatives $\frac{\partial G}{\partial x_2}$ and $\frac{\partial G}{\partial x_3}$ should not vanish simultaneously at $(\frac{b}{a}, 0)$.

A non-smooth point on $\mathcal{C}$ is called *non-regular* or *singular*. The curve $\mathcal{C}$ is called *non-singular* or *regular* or *smooth* if all points are smooth on $\mathcal{C}$.

The *coordinate ring* of $\mathcal{C}$ over K is the quotient ring defined as $K[\mathcal{C}] = K[x_1, x_2]/\langle F \rangle$. In a similar manner, the coordinate ring of $\mathcal{C}$ over $\overline{K}$ is defined as $\overline{K}[\mathcal{C}] = \overline{K}[x_1, x_2]/\langle F \rangle$. Elements of $\overline{K}[\mathcal{C}]$ are called *polynomial functions*. Since we assume that F is irreducible over $\overline{K}$, $\langle F \rangle$ is a prime ideal, so $\overline{K}[\mathcal{C}]$ is an integral domain.

The field of fractions of $K[\mathcal{C}]$ is called the *function field of* $\mathcal{C}$ over K, and is denoted by $K(\mathcal{C})$. Similarly, the quotient field of $\overline{K}[\mathcal{C}]$ is known as the *function field of* $\mathcal{C}$ over $\overline{K}$, and is denoted by $\overline{K}(\mathcal{C})$. Elements of $\overline{K}(\mathcal{C})$ can be written as $\frac{g(u,v)}{h(u,v)}$ with $g, h \in \overline{K}[\mathcal{C}]$, $h \neq 0$, and are called a *rational functions* on $\mathcal{C}$. Clearly, $\overline{K}[C]$ is a subring of $\overline{K}(C)$, so every polynomial function is also a rational function.

Let P be a rational point on $\mathcal{C}$, and $r \in \overline{K}(\mathcal{C})$. Then r is said to be *defined at* P if there exist polynomial functions $g, h \in \overline{K}(\mathcal{C})$ such that $r = \frac{g}{h}$ with $h(P) \neq 0$. If no such function exists,

then r is *not defined at* P. If r is defined at P, the value of r at P is $r(P) = \frac{g(P)}{h(P)}$. If $r(P) = 0$, then r has a *zero* at P. If r is not defined at P or if $r(P) = \infty$, then r has a *pole* at P.

2.2.3 Divisors

Divisors are formal sum of rational points.

- A formal sum of rational points on $\mathcal{C}$ is called a *divisor* D. A divisor is therefore of the form
$$D = \sum_{P \in \mathcal{C}} m_P P,\ m_P \in \mathbb{Z},$$
with only a finite number of P for which $m_P \neq 0$.
- A divisor is called an *effective divisor* or a *positive divisor* if $m_P \geq 0$ for all P.
- Let $D = \sum_{P \in \mathcal{C}} m_P P$, and $D' = \sum_{P \in \mathcal{C}} n_P P$ be two effective divisors. We write $D \geq D'$ if each $m_P \geq n_P$.
- The sum $\sum_{P \in \mathcal{C}} m_P$ is the *degree* of D, denoted by $\deg D$.
- The integer m_P is the *order* of D at a point P, and is written as $\mathrm{ord}_P(D)$.
- The set $\mathbf{D}$ of all divisors is an additive group under the addition rule:
$$\sum_{P \in \mathcal{C}} m_P P + \sum_{P \in \mathcal{C}} n_P P = \sum_{P \in \mathcal{C}} (m_P + n_P) P.$$
- The gcd of two divisors $D_1 = \sum_{P \in \mathcal{C}} m_P P$ and $D_2 = \sum_{P \in \mathcal{C}} n_P P$ is defined as
$$\gcd(D_1, D_2) = \sum_{P \in \mathcal{C}} \min(m_P, n_P) P - \left(\sum_{P \in \mathcal{C}} \min(m_P, n_P) \right) \mathcal{O}.$$
- $\mathbf{D}^0$ denotes the set of all degree-zero divisors.
- Let $R \in \overline{\mathbb{F}}_q(\mathcal{C})$. The divisor of R, denoted $\mathrm{div}(R)$, is written as
$$\mathrm{div}(R) = \sum_{P \in \mathcal{C}} (\mathrm{ord}_p R) P,$$
where $\mathrm{ord}_p\, R$ denotes the *order of the rational function R at the point P*.
- A *principal divisor* is the divisor of a rational function, that is, $D = \mathrm{div}(R)$ for some $R \in \overline{\mathbb{F}}_q(\mathcal{C})$. The set of all principal divisors is denoted by $\mathbf{P}$. Each principal divisor has degree zero, that is, $\mathbf{P}$ forms a subgroup of $\mathbf{D}^0$. The quotient group $\mathbf{J} = \mathbf{D}^0/\mathbf{P}$ is called the *Jacobian* of the curve $\mathcal{C}$. If $D_1, D_2 \in \mathbf{D}^0$, we write $D_1 \sim D_2$ if $D_1 - D_2 \in \mathbf{P}$. In this case, D_1 and D_2 are called *equivalent*.
- Two divisors D_1, D_2 are *linearly equivalent*, denoted $D_1 \equiv D_2$, if $D_2 = D_1 + \mathrm{div}(z)$ for some $z \in K$.
  - The relation $\equiv$ is an equivalence relation.
  - For $z \in K$, $D = \mathrm{div}(z)$ if and only if $D \equiv 0$.
  - $D_1 \equiv D_2$ implies that $\deg(D_1) = \deg(D_2)$.
  - If $D_1 \equiv D_2$ and $D_1' \equiv D_2'$, then $D_1 + D_1' = D_2 + D_2'$.

2.2.4 Vector Space $L(D)$

For a divisor $D = \sum_{P \in \mathcal{C}} m_P P$, define a set $L(D) = \{f \in K \mid \text{ord}_P(f) \geq -m_P \text{ for all } P \in \mathcal{C}_K\}$. A rational function $f \in L(D)$ implies that $\text{div}(f) + D \geq 0$ or $f = 0$. Moreover, $L(D)$ is a vector space over K, and $l(D)$ denotes the dimension of $L(D)$.

Properties

- $D \leq D'$ implies $L(D) \subset L(D')$ and $\deg(D' - D) \geq \dim_K(L(D')/L(D))$.
- If $\deg(D) < 0$, then $L(D) = 0$. Also $L(0) = K$.
- For all D, $L(D)$ is finite dimensional. Also, $\deg(D) \geq 0$ implies $l(D) \geq \deg(D) + 1$.
- $L(D) = L(D')$ if $D \equiv D'$.

Theorem 2.2.1 (Riemann Theorem) *There exists an integer g such that $l(D) \geq \deg(D) + 1 - g$ for all divisors D. The smallest such non-negative integer g is called the genus of C.*

2.3 Elliptic and Hyperelliptic Curves

An elliptic curve is a genus-one non-singular irreducible projective curve over K having at least one K-rational point. The usual form of an elliptic curve is $y^2 = x^3 + ax + b$ with $a, b \in K$. The set of all K-rational points is an Abelian group under point addition. The details of elliptic curves are available in [7].

Hyperelliptic curves form a generalization of elliptic curves. The Jacobian of an elliptic curve $\mathcal{E}$ over $\overline{K}$ is isomorphic to the set of all K-rational points on $\mathcal{E}$. But for a hyperelliptic curve, this property does not hold, that is, the set of all K-rational points on a hyperelliptic curve $\mathcal{H}$ no longer possesses the Abelian-group structure.

Definition A non-singular curve $\mathcal{C}$ over K with genus $g > 1$ is called a *hyperelliptic curve* if the function field $K(\mathcal{C})$ is a degree-two separable extension of the rational function field $K(x)$ for some function x. In other words, a hyperelliptic curve $\mathcal{C}$ is a curve over K having a degree-two finite morphism $f : \mathcal{C} \to \mathbb{P}^1_K$ and genus

$$g = \max\{\lfloor(\deg(f(x)) - 1)/2\rfloor, \deg(h(x)) - 1\}.$$

The hyperelliptic curve $\mathcal{C}$ of genus $g > 1$ over K is usually expressed by the equation

$$\mathcal{C} : \; y^2 + h(x)y = f(x), \tag{2.1}$$

where $h(x) \in K[x]$ is a polynomial of degree $\leq g$, and $f(x) \in K[x]$ is a monic polynomial of degree equal to $2g + 1$. The curve $\mathcal{C}$ must be non-singular, that is, it must not contain any solution $P = (x, y) \in \overline{K}^2$ of the equation $y^2 + h(x)y - f(x) = 0$, at which both the partial derivatives vanish:

$$h'(x)y - f'(x) = 0, \text{ and } 2y + h(x) = 0.$$

If the characteristic of the field is not two, then $\mathcal{C}$ can be simplified as $y^2 = f(x)$, where f is monic of degree $2g + 1$. The curve should be smooth, that is, $f(x)$ must be square free, that is, we should have $\gcd(f(x), f'(x)) = 1$.

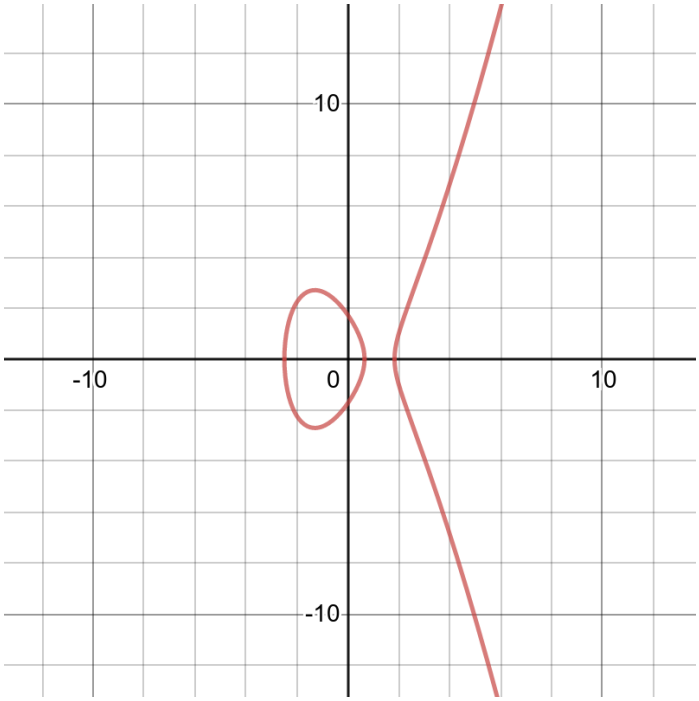

Figure 2.1: Elliptic curve: $\mathcal{E} : y^2 = x^3 - 5x + 3$ over $\mathbb{R}$

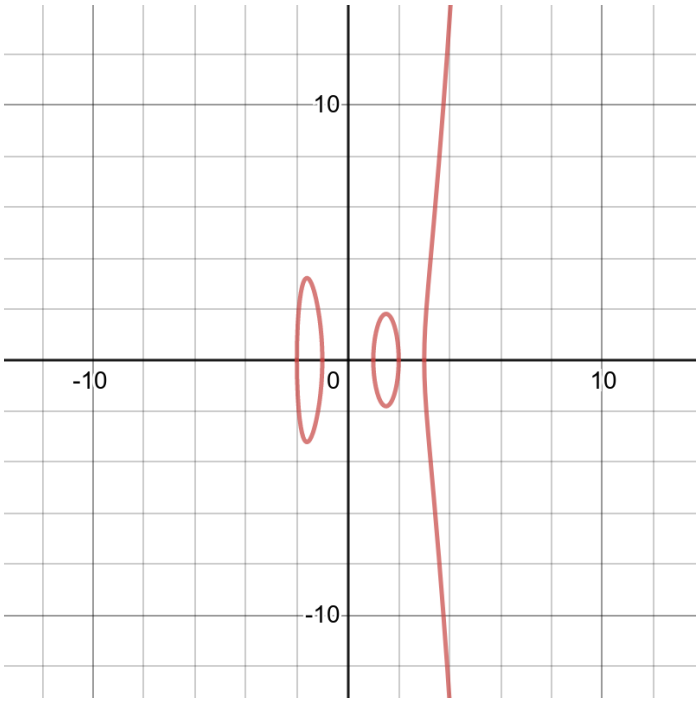

Figure 2.2: Hyperelliptic curve $\mathcal{H} : y^2 = x^5 - 3x^4 + 15x^2 + 4x - 12$ over $\mathbb{R}$

Rational Points

The set $\mathcal{C}(\overline{K})$ of *rational points* on the curve $\mathcal{C}$ includes all the ordered pairs (the *finite points*) $(x, y) \in \overline{K}^2$ which satisfy Eqn (2.1), along with a special *point at infinity* $\mathcal{O}$. For a finite point $P = (u, v) \in \overline{K}^2$ on $\mathcal{C}$, we take the *opposite* of P as the point $\widetilde{P} = (u, -v - h(u))$. The opposite of $\mathcal{O}$ is $\mathcal{O}$ itself. A finite point P is known to be *special* if $P = \widetilde{P}$; otherwise, the point is *ordinary*.

Polynomials and Rational Functions

The *coordinate ring* of a hyperelliptic curve $\mathcal{C}$ over $\overline{K}$, denoted by $\overline{K}[\mathcal{C}]$, is the quotient ring defined as

$$\overline{K}[\mathcal{C}] = \overline{K}[x, y] / \langle y^2 + h(x)y - f(x) \rangle,$$

where $\langle y^2 + h(x)y - f(x) \rangle$ indicates the ideal in $\overline{K}[x, y]$ generated by the polynomial $y^2 + h(x)y - f(x)$. Each element of $\overline{K}[\mathcal{C}]$ is called a *polynomial function* on $\mathcal{C}$. They can be uniquely expressed as $G(x, y) = a(x) + yb(x)$ for $a(x), b(x) \in K[x]$. The *conjugate* of G is $G'(x, y) = a(x) + b(x)(h(x) + y)$.

The *function field* $\overline{K}(\mathcal{C})$ of $\mathcal{C}$ over $\overline{K}$ is the field of fractions of $\overline{K}[\mathcal{C}]$. Each element of $\overline{K}(\mathcal{C})$ is called a *rational function* on $\mathcal{C}$. Every rational function $r(x, y)$ can be represented as $r(x, y) = \frac{G(x,y)}{H(x,y)}$ with $H(x, y) \neq 0$.

The definitions given for general algebraic curves continue to remain valid for hyperelliptic curves. Suppose $P = (x', y')$ be a point the curve $\mathcal{C}$. The divisor of the rational function $r = (x - x')^n$ equals

$$\operatorname{div}(r) = \begin{cases} nP + n\widetilde{P} - 2n\mathcal{O}, & \text{if } P \text{ is an ordinary point,} \\ 2nP - 2n\mathcal{O}, & \text{if} P \text{ is a special point.} \end{cases} \tag{2.2}$$

Definition The *support* of a divisor $D = \sum_{P \in \mathcal{C}} m_P P$ is the set $\operatorname{Supp}(D) = \{P \in \mathcal{C} | m_p \neq 0\}$.

Definition A divisor $D = \sum m_i P - (\sum m_i)\mathcal{O}$ with each $m_i \geq 0$ is called a *semi-reduced divisor*. If $P_i \in \operatorname{Supp}(D)$ is an ordinary point, then $\widetilde{P_i} \notin \operatorname{Supp}(D)$. If $P_i = \widetilde{P_i} \in \operatorname{Supp}(D)$ is a special point, then $m_i = 1$.

For every divisor $D_1 \in \mathbf{D}^0$, there exists a semi-reduced divisor $D_2 \in \mathbf{D}^0$ such that $D_1 \sim D_2$.

Definition A semi-reduced divisor $D = \sum m_i P - (\sum m_i)\mathcal{O}$ is a *reduced* divisor if $\sum m_i \leq g$.

For every divisor $D_1 \in \mathbf{D}^0$, there exists a *unique* reduced divisor $D_2 \in \mathbf{D}^0$ such that $D_1 \sim D_2$.

Definition A divisor $D = \sum m_P P$ is said to be *defined over* K if $D^\sigma \stackrel{def}{=} (x^\sigma, y^\sigma)$ is equal to D for all automorphisms σ of $\overline{K}[x, y]$ over $K[x, y]$. The set $\mathbb{J}$ of all divisor classes in $\mathbf{J}$ that have representatives defined over K is a subgroup of $\mathbf{J}$.

Now, we specialize to finite fields. The Jacobian $\mathbb{J}$ defined over a finite field is a finite Abelian group. Each divisor in the Jacobian has a compact representation. The discrete logarithm problem is computationally difficult in the Jacobian. So one can generate cryptographically suitable hyperelliptic curves, and implement cryptographic primitives based on the group.

Mumford Representation

Each reduced divisor has a unique representation of the form

$$\gcd(\mathrm{div}(u(x)),\ \mathrm{div}(u(x) - v)),$$

abbreviated as (u, v) [45]. Here, $u, v \in \mathbb{F}_q[x]$ must satisfy:

1. u is monic,
2. $\deg(v) < \deg(u) \leq g$,
3. $u|(v^2 + h(u)v - f(u))$.

Let $D = \sum_{i=1}^{l} P_i - l\mathcal{O}$, where $P_i = (x_i, y_i) \neq \mathcal{O}$, $P_i \neq P_j$ for $i \neq j$, and $l \leq g$. Then, the divisor D of the curve $\mathcal{C}$ is defined by the polynomials

$$u(x) = \prod_{i=1}^{l}(x - x_i),$$

and

$$\left(\frac{d}{dx}\right)^j [v(x)^2 + v(x)h(x) - f(x)]_{x=x_i} = 0 \qquad \text{for } 0 \leq j \leq l-1.$$

The polynomial $v(x)$ can be derived from Lagrange's interpolation formula for l rational points on the curve.

$$v(x) = \sum_{i=1}^{l} \frac{\prod_{j\neq i}(x - x_j)}{\prod_{j\neq i}(x_i - x_j)} y_i.$$

In particular, for $g = 2$ and $l = 2$, we have

$$u(x) = (x - x_1)(x - x_2) \text{ and } v(x) = \left(\frac{x - x_2}{x_1 - x_2}\right) y_1 + \left(\frac{x - x_1}{x_2 - x_1}\right) y_2.$$

A single rational point (x_1, y_1) also gives a valid divisor $(x - x_1, y_1)$ (corresponding to $l = 1$). For l=0, we have the reduced divisor $(1, 0)$ which acts as the identity of the group.

2.3.1 Group Law

The Jacobian group is constructed using equivalence classes of reduced divisors. Every class has a unique reduced divisor. Given two reduced divisors, we need to find the unique reduced divisor that represents the sum of the classes of the input divisor classes.

Let the rational points P_1, P_2 define the first divisor, and the points Q_1, Q_2 the second divisor. A degree-three polynomial passing through these four points has two more intersection points R_1, R_2 with $\mathcal{C}$. The opposites $-R_1, -R_2$ of these points define the sum of the input divisors. Geometrically, this is equivalent to saying that the six points $P_1, P_2, Q_1, Q_2, R_1, R_2$ on the cubic add up to zero. Cantor [6] provides an algorithm for performing divisor-class addition. Later, Harley [32] modifies Cantor's algorithm to make it more efficient. These algorithms are studied in detail in Chapter 3. We work with fields of characteristic $\neq 2$, so we use curves of the special form $y^2 = f(x)$. For these curves, the opposite (additive inverse) of the divisor (u, v) is $(u, -v)$. Finally, the reduced divisor $(1, 0)$ is the identity of the group.

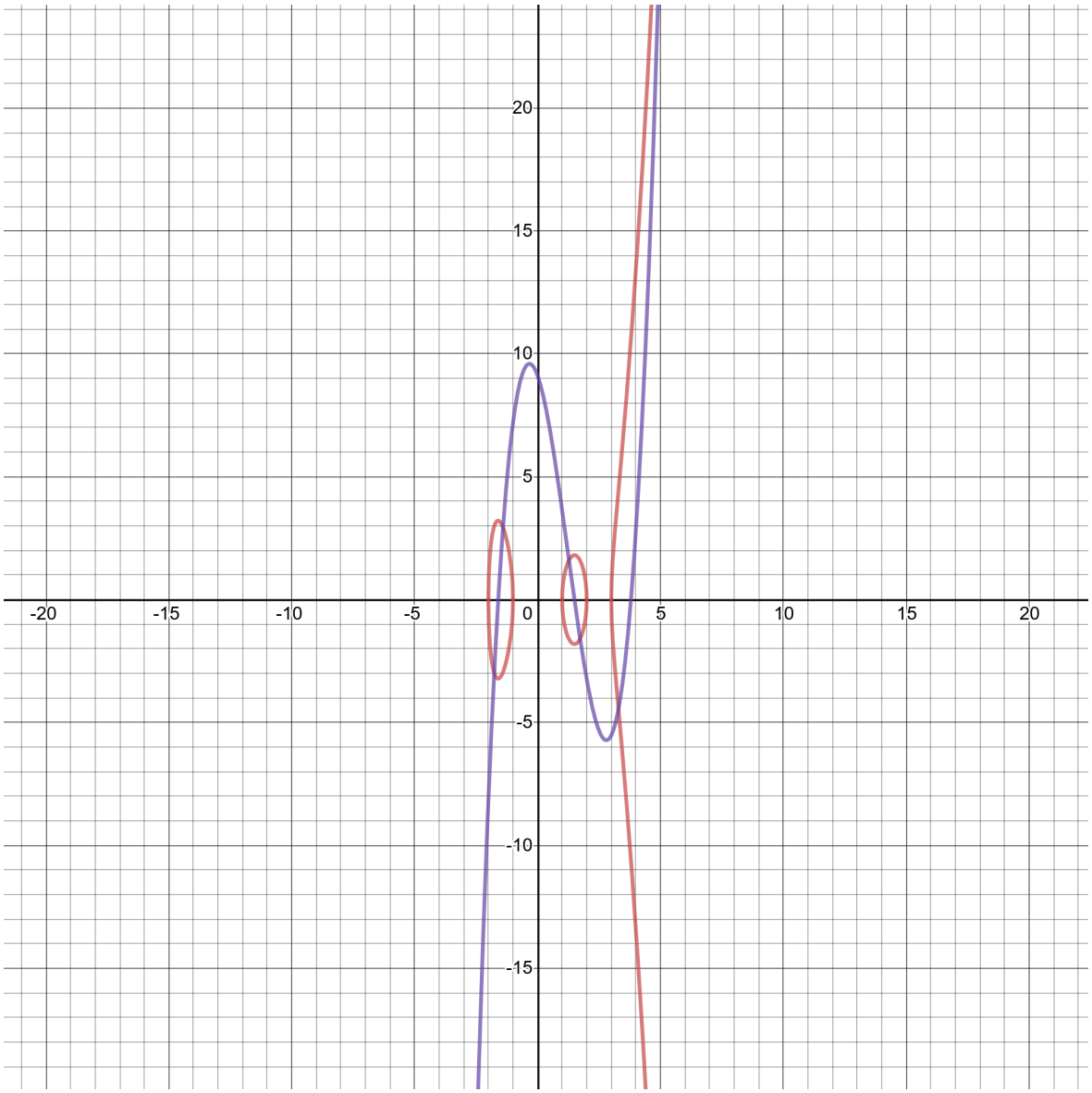

Figure 2.3: Divisor addition on Curve $\mathcal{H} : x^5 - 3x^4 + 15x^2 + 4x - 12$ over $\mathbb{R}$

2.3.2 Scalar Multiplication

Scalar multiplication plays a vital role in hyperelliptic-curve cryptography. Let D be a divisor. We need to compute the n-th multiple $E = nD = D + D + \cdots + D$ of D. Addition and doubling are the basic operations used in scalar multiplication. Some popular algorithms to perform this are the double-and-add, the windowed double-and-add, the nonadjacent-form double-and-add, Montgomery-ladder, and other addition-chain algorithms [51].

The double-and-add algorithm works with the binary representations of n. Let n_i denote the i-th bit n. If $n_i = 1$, then the algorithm performs one doubling and one addition. If $n_i = 0$, then only one doubling suffices. The l-bit windowed method uses the representation of n as $n = n_0 + 2^l n_1 + 2^{2l} n_2 + \cdots + 2^{lt} n_t$. At the cost of some precomputation overhead, the windowed method can reduce many addition steps. All doubling steps are carried out anyway. Usually, $l = 4$ is the most preferred choice. The double-and-add algorithms are not secure against side-channel attack (SCA).

The Montgomery ladder is secure against SCA [52]. This algorithm is treated as a smart modification of the double-and-add algorithm. Some minor differences from the basic double-and-add algorithms follow.

1. The Montgomery ladder needs two variables for storing intermediate values.
2. The Montgomery ladder allows differential addition to enhance efficiency.
3. In each iteration, one doubling and one addition are always performed.

2.4 Discrete Logarithm Problem on the Jacobian

Koblitz suggests the use of the Jacobians $\mathbb{J}$ of hyperelliptic curves in cryptographic applications. The discrete logarithm problem is considered difficult in the Jacobian.

Discrete logarithm problem on HEC: Given two divisors D_1, D_2 such that $D_2 = mD_1$, find m. This m is unique modulo the order of D_1.

To construct 128-bit secure cryptographic protocols, EC and HEC need 256- and 128-bit finite fields, respectively. Pollard's rho method and its variants [23, 57, 69] are well-known for solving the DLP in $O(\sqrt{n})$ time, where n is the group order. Some individual cases of DLP on HEC as reported by [18, 60] have lower complexity than $O(\sqrt{n})$. In 1994, Adleman proposes a subexponential-time algorithm for computing discrete logarithms in large-genus curves [1]. For genus $g > 4$, this algorithm performs better than Pollard's rho method. In 2000, Gaudry [24] reports that the index calculus method has lower complexity than Pollard's rho method for hyperelliptic curves of genus $g > 4$. In 2003, Thériault provides an optimized algorithm for computing discrete logarithms in $\mathbb{J}$ of low-genus curves, which runs in subexponential time [66]. In 2007, Enge and Gaudry [14] propose a much faster algorithm for computing DLP in Jacobians of curves.

Despite the existence of these subexponential-time algorithms, no algorithms faster than the generic square-root methods are known for hyperelliptic curves of genus two. In view of this, our work reported in this thesis concentrates on curves of genus two only.

CHAPTER

THREE

DIVISORS ARITHMETIC

3.1 Introduction

This chapter starts with computing the order of the Jacobian. Counting the cardinality of all rational points is not important in this context, since for hyperelliptic curves the set of all rational points does not form a group. Instead the Jacobian is an additive Abelian group. The Jacobian arithmetic is addressed next. Cantor's algorithm is the entry point for divisor-class arithmetic. For efficiency, some tuned algorithms for genus-two curves are discussed. The performance of the family of subfield curves is mentioned at the end of the chapter.

3.2 Computing the Jacobian Order

Proposing suitable hyperelliptic curves is a challenging problem in curve-based cryptography. The order of the Jacobian should have a large prime factor. To achieve 128-bit security, the bit size of this prime factor should be 256 (or more). A point-counting algorithm is essential to identify suitable curves. Elliptic curves have polynomial-time point-counting algorithms [63, 13, 26]. However, there are no such practical and efficient point-counting algorithms for hyperelliptic curves over large prime fields. For small prime fields, efficient algorithms exist based on the p-adic method [39, 61]. A generalized version of Schoof's algorithm [63] is introduced by Pila [56]. Huang [36] and Adleman [2] also come up with point-counting algorithms. These algorithms are important from theoretical perspectives. Gaudry and Harley [27, 29] propose and implement a point-counting algorithm for hyperelliptic curves defined over large prime fields.

We start with a few mathematical definitions. This is followed by the point-counting algorithm that we use for subfield curves. This section ends with a few examples.

Frobenius endomorphism

Frobenius endomorphism usually apply to commutative rings with prime characteristics. Here, we work with finite fields $K = \mathbb{F}_q = \mathbb{F}_{p^d}$. This maps each element to its $q-$th power:

$\mathcal{F}(\alpha) = \alpha^q$ for $\alpha \in \mathbb{F}_q$.

This map preserves multiplication, that is, $\mathcal{F}(\alpha\beta) = \mathcal{F}(\alpha)\mathcal{F}(\beta)$. The map $\mathcal{F}$ extends to divisors on $\mathbb{J}_q$. For a reduced divisor $D = (u, v) \in \mathbb{J}_q$, we have $\mathcal{F}(D) = \mathcal{F}((u, v)) = (\mathcal{F}(u), \mathcal{F}(v))$.

The Frobenius map is a linear map and satisfies a degree-$2g$ characteristic polynomial with integer coefficients:

$$L(x) = x^4 + s_1x^3 + s_2x^2 + s_1qx + q^2.$$

$L(\mathcal{F})$ is the identity map on the Jacobian $\mathbb{J}_q$. It therefore follows that the cardinality of the Jacobian is equal to $L(1)$.

Zeta function

For a non-singular projective curve $\mathcal{C}$, the zeta function $Z_{\mathcal{C}}(T)$ can be written as

$$Z_{\mathcal{C}}(T) = \frac{L(T)}{(1-T)(1-qT)},$$

where $L(T)$ can be expressed as

$$L(T) = \prod_{i=1}^{2g}(1 - \alpha_i^k T).$$

Riemann's hypothesis for $\mathcal{C}$ implies that $|\alpha_i| = q^{1/2}$. By Hasse's theorem, the cardinality n of the Jacobian of a curve having genus g satisfies

$$\lceil(\sqrt{q} - 1)^{2g}\rceil \leq n \leq \lfloor(\sqrt{q} + 1)^{2g}\rfloor$$

The width w of the Hasse–Weil interval is therefore

$$w = \lfloor(\sqrt{q} + 1)^{2g}\rfloor - \lceil(\sqrt{q} - 1)^{2g}\rceil \approx 4gq^{g-\frac{1}{2}}$$

3.2.1 Order Computation

Let l be the security level (a bit size) we want to achieve. A hyperelliptic curve of this security level can be generated as follows. Take an l-bit prime p. Generate curves C over $\mathbb{F}_p$, and compute the order of its Jacobian over $\mathbb{F}_p$. Repeat until a $2l$-bit prime is obtained as the order. The bottleneck of this approach is the algorithm for computing orders over large fields $\mathbb{F}_p$. The literature provides algorithms to this effect, but these algorithms are complicated and inefficient [61].

To avoid this difficulty, we choose an $l/4$-bit prime p. For $l \leq 128$, this prime p fits in a 32-bit unsigned integer. We generate a curve over $\mathbb{F}_p$, and compute the order of $\mathbb{J}_p$. Since p is now small, simple and practical point-counting algorithms can be used. We then consider the quintic extension $\mathbb{F}_q = \mathbb{F}_{p^5}$. The curve C is naturally defined over $\mathbb{F}_q$. Moreover, given the group size $|\mathbb{J}_p|$, the group size $|\mathbb{J}_q|$ can be calculated using simple formulas. We require $n = |\mathbb{J}_q|/|\mathbb{J}_p|$ to be a prime. This approach helps us generate many suitable curves of security level l reasonably quickly. On the flip side, we now have to work in a field of bit size $|q| = 5l/4$.

In the rest of this section, we discuss the two-stage process we adopted.

- **Compute the order of $\mathbb{J}_p$ for the prime p**

  To compute the order of an element in $\mathbb{J}_p$, we use a baby-step-giant-step algorithm. The order of $\mathbb{J}_p$ lies in the Weil interval $[w_l, w_h]$, where $w_l = \lceil (\sqrt{p}-1)^4 \rceil$ and $w_h = \lfloor (\sqrt{p}+1)^4 \rfloor$. The order of any point in $\mathbb{J}_p$ is an integral divisor of the group order [61]. The following algorithm computes the order of an element P.

  1. Set $W = w_h - w_l$.
  2. Set $S = \left\lceil \sqrt{W} \right\rceil$.
  3. Precompute $-jP$ for $j = 0, 1, 2, \ldots, S-1$, and store the pairs $(-jP, j)$ in a list.
     // Baby steps
     // Notice that $-jP$ is computed as $-(j-1)P + (-P)$ for $j > 0$.
  4. If some $j > 0$ is found such that $-jP = (1, 0)$, return j as the order of P.
     // Recall that $(1, 0)$ is the identity in $\mathbb{J}_p$.
  5. Sort the list with respect to $-jP$.
  6. Compute $Q = w_l P$ using the repeated double-and-add algorithm.
  7. Compute $SP = -(-(S-1)P + (-P))$.
  8. For $i = 0, 1, 2, \ldots, S-1$, repeat // Giant steps
     (a) Search the list for Q using the binary search algorithm.
     (b) If some entry (Q, j) is found in the list, store $k = w_l + iS + j$.
     (c) Update $Q = Q + SP$. // SP was precomputed, so this is just an addition
  9. If there is only one match k, then return this k as the order of P. If there are multiple matches, return the difference between any two consecutive matches as the order of P.

  Given this algorithm for computing element orders, the group order can be computed as follows. We keep on generating random elements P, and compute their orders k. If $2k > w_h$, then k is order of $\mathbb{J}_p$. Otherwise, we keep on computing the lcm of individual element orders until the lcm l satisfies $2l > w_h$. If in several iterations, no such k or l can be obtained, the curve has low exponent, and its order cannot be determined using the above baby-step-giant-step algorithm. In this case, we discard the curve. The failure to pinpoint the order of some of the curves is not a practically severe issue.

  The algorithm makes $O(S \log S)$ group operations, where S is the square-root of the width $W = w_h - w_l$ of the Weil interval. For $p \approx 2^{32}$, we have $W \approx 2 \times 10^{15}$, and $S \approx 5 \times 10^7$, so this algorithm is reasonably efficient.

- **Compute the order of $\mathbb{J}_q$ for $q = p^d$**

  We work in quintic extensions, so $d = 5$ for us. We here provide a treatment for a general d. Since $\mathbb{F}_q$ is an extension of $\mathbb{F}_p$, a curve available from the previous stage continues to remain a curve defined over $\mathbb{F}_q$. It is easy to compute the order of $\mathbb{J}_q$ from the order of $\mathbb{J}_p$. Instead of running a point-counting algorithm for $\mathcal{C}$ over $\mathbb{F}_q$, we now use the L-functions of the curve [39, 7].

  $\mathcal{C} : y^2 = f(x)$ is a genus-two hyperelliptic curve defined over a prime field $\mathbb{F}_p$. Here, $f(x)$ is a monic square-free polynomial of degree five. Let N_d denote the number of rational points on $\mathcal{C}$ over $\mathbb{F}_{p^d}$ (including the point at infinity). Notice that N_d is not the order of the Jacobian group $\mathbb{J}_{p^d}$. It is fairly straightforward to obtain the count N_d (by

exhaustive enumeration) so long as p^d is small. We will shortly see that only N_1 needs to be computed.

The zeta function of the curve $\mathcal{C}$ is defined by the infinite series

$$\begin{aligned} Z_{\mathcal{C}}(T) = \exp\left(\sum_{d=1}^{\infty} \frac{N_d T^d}{d}\right) &= 1 + \left(\sum_{d=1}^{\infty} \frac{N_d T^d}{d}\right) + \frac{1}{2!}\left(\sum_{d=1}^{\infty} \frac{N_d T^d}{d}\right)^2 + \cdots \\ &= 1 + N_1 T + \frac{1}{2}(N_1^2 + N_2)T^2 + \cdots . \end{aligned}$$

It turns out that this function has an alternate expression

$$Z_{\mathcal{C}} = \frac{L(T)}{(1-T)(1-pT)},$$

where

$$L(T) = s_0 + s_1 T + s_2 T^2 + s_3 T^3 + s_4 T^4$$

for some integers s_0, s_1, s_2, s_3, s_4. These integers satisfy $s_0 = 1$ and $s_{4-i} = p^{2-i} s_i$ for $i = 0, 1, 2$. Therefore we can rewrite $L(T)$ as

$$L(T) = 1 + s_1 T + s_2 T^2 + s_1 p T^3 + p^2 T^4.$$

If we can compute the two integers s_1, s_2, then $L(T)$ is fully determined. This function is related to the Jacobian orders as follows.

$$L(1) = |\mathbb{J}_p|, \tag{3.1}$$

$$L(-1) = |\widetilde{\mathbb{J}}_p|. \tag{3.2}$$

Here, $\widetilde{\mathcal{C}}$ is a quadratic twist of $\mathcal{C}$ over $\mathbb{F}_p$ defined by $vy^2 = f(x)$, where v is a quadratic non-residue modulo p. Using two resulting linear equations, we can compute s_1 and s_2. But since the point-counting algorithm is written for curves of the form $y^2 = f(x)$ only, we use an equation other than (3.2). To that end, we make the power-series expansion of the second expression for $Z_{\mathcal{C}}(T)$.

$$\begin{aligned} Z_{\mathcal{C}}(T) &= (1 + s_1 T + s_2 T^2 + s_1 p T^3 + p^2 T^4)(1-T)^{-1}(1-pT)^{-1} \\ &= (1 + s_1 T + s_2 T^2 + s_1 p T^3 + p^2 T^4)(1 + T + T^2 + \cdots)(1 + pT + p^2 T^2 + \cdots) \\ &= 1 + (p + s_1 + 1)T + (p^2 + s_2 + 1 + s_1 + s_1 p + p)T^2 + \cdots . \end{aligned}$$

Comparing this with the first power-series expansion of $Z_{\mathcal{C}}$ (equating coefficients of T and T^2) gives

$$N_1 = p + s_1 + 1, \tag{3.3}$$

$$N_2 = 2(p^2 + s_2 + 1 + s_1 + s_1 p + p) - N_1^2 = p^2 - s_1^2 + 2s_2 + 1. \tag{3.4}$$

The determination of N_2 requires working in the quadratic extension $\mathbb{F}_{p^2}$. It is thus evident that the easiest way to determine $L(T)$ is to use Eqns (3.1) and (3.3).

Let $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ be the four roots (complex numbers) of $L^{(opp)}(T) = T^4 + s_1 T^3 + s_2 T^2 + s_3 T + s_4$ (the opposite of $L(T)$). For each $d = 1, 2, 3, \ldots$, define

$$L_d(T) = (1 - \alpha_1^d T)(1 - \alpha_2^d T)(1 - \alpha_3^d T)(1 - \alpha_4^d T).$$

The connection between these L-polynomials and the Jacobian orders is this:

$$|\mathbb{J}_{p^d}| = L_d(1) = (1-\alpha_1^d)(1-\alpha_2^d)(1-\alpha_3^d)(1-\alpha_4^d). \tag{3.5}$$

(We have $L(T) = L_1(T)$, and $|\mathbb{J}_p| = L_1(1) = L(1)$ which is consistent with Eqn (3.1).)

It therefore follows that if we can compute the four roots $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ with sufficient precision, we readily obtain the Jacobian orders in extension fields.

We can avoid complex arithmetic altogether. Indeed, we do not need to compute the roots $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ of $L^{(opp)}(T)$. The elementary symmetric polynomials in four variables $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ are defined as follows.

$$\begin{aligned}
e_0 &= 1, \\
e_1 &= \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4, \\
e_2 &= \alpha_1\alpha_2 + \alpha_1\alpha_3 + \alpha_1\alpha_4 + \alpha_2\alpha_3 + \alpha_2\alpha_4 + \alpha_3\alpha_4, \\
e_3 &= \alpha_1\alpha_2\alpha_3 + \alpha_1\alpha_2\alpha_4 + \alpha_1\alpha_3\alpha_4 + \alpha_2\alpha_3\alpha_4, \\
e_4 &= \alpha_1\alpha_2\alpha_3\alpha_4, \\
e_k &= 0 \text{ for } k \geq 5.
\end{aligned}$$

Since $L^{(opp)}(T) = T^4 + s_1T^3 + s_2T^2 + s_3T + s_4 = (T-\alpha_1)(T-\alpha_2)(T-\alpha_3)(T-\alpha_4)$, it follows that

$$e_0 = 1, \;\; e_1 = -s_1, \;\; e_2 = s_2, \;\; e_3 = -s_3, \;\; e_4 = s_4, \;\; e_k = 0 \text{ for } k \geq 5.$$

Now, let us define the power sums of the four roots for all $k \geq 1$:

$$p_k = \alpha_1^k + \alpha_2^k + \alpha_3^k + \alpha_4^k.$$

The Newton–Girard formula [7] relates these two sequences as follows:

$$ke_k = \sum_{i=1}^{k} (-1)^{i-1} e_{k-i} p_i$$

for all $k \geq 1$. Since we know the e_k values, we can compute the p_k values iteratively using this formula. More explicitly, we have

$$\begin{aligned}
p_1 &= e_1, \\
p_2 &= e_1p_1 - 2e_2, \\
p_3 &= e_1p_2 - e_2p_1 + 3e_3, \\
p_4 &= e_1p_3 - e_2p_2 + e_3p_1 - 4e_4, \\
p_k &= e_1p_{k-1} - e_2p_{k-2} + e_3p_{k-3} - e_4p_{k-4} \text{ for all } k \geq 5.
\end{aligned}$$

Now, let us come back to our original problem of computing the right side of Eqn (3.5). Notice that $\alpha_1^d, \alpha_2^d, \alpha_3^d, \alpha_4^d$ are the roots of

$$L_d^{(opp)}(T) = (T-\beta_1)(T-\beta_2)(T-\beta_3)(T-\beta_4),$$

where $\beta_i = \alpha_i^d$ for $i = 1, 2, 3, 4$. Name the elementary symmetric polynomials in $\beta_1, \beta_2, \beta_3, \beta_4$ as E_k (for example, $E_1 = \beta_1 + \beta_2 + \beta_3 + \beta_4$), and the power sums as P_k

(for example, $P_2 = \beta_1^2 + \beta_2^2 + \beta_3^2 + \beta_4^2$). These new power sums are related to the old power sums (in α_i) as

$$P_k = \beta_1^k + \beta_2^k + \beta_3^k + \beta_4^k = \alpha_1^{dk} + \alpha_2^{dk} + \alpha_3^{dk} + \alpha_4^{dk} = p_{dk}.$$

We need only P_1, P_2, P_3, P_4 (that is, $p_d, p_{2d}, p_{3d}, p_{4d}$). Consequently, we compute p_k for $k = 1, 2, 3, \ldots, 4d$. Now, we use the Newton–Girard formula for $L_d^{(opp)}(T)$, that is,

$$kE_k = \sum_{i=1}^{k} (-1)^{i-1} E_{k-i} P_i$$

for all $k \geq 1$, and obtain

$$\begin{aligned} E_0 &= 1, \\ E_1 &= P_1, \\ E_2 &= \frac{1}{2}(E_1 P_1 - P_2), \\ E_3 &= \frac{1}{3}(E_2 P_1 - E_1 P_2 + P_3), \\ E_4 &= \frac{1}{4}(E_3 P_1 - E_2 P_2 + E_1 P_3 - P_4). \end{aligned}$$

This, in turn, implies

$$L_d(T) = E_0 - E_1 T + E_2 T^2 - E_3 T^3 + E_4 T^4,$$

and, in particular,

$$|\mathbb{J}_{p^d}| = L_d(1) = E_0 - E_1 + E_2 - E_3 + E_4.$$

We have $|\mathbb{J}_p| \approx q^2 = p^{10}$. Since $\mathbb{J}_p$ is a subgroup of $\mathbb{J}_q$, the order of $\mathbb{J}_p$ must divide the order of $\mathbb{J}_q$. We use the cofactor

$$n = \frac{|\mathbb{J}_q|}{|\mathbb{J}_p|}.$$

If n is prime, $\mathbb{J}_q$ contains a subgroup G of this order. The bit length of n is

$$|n| = |\mathbb{J}_q| - |\mathbb{J}_p| \approx (10 - 2)|p| = 8|p|,$$

that is, the security level is $|n|/2 \approx 4|p| = l$, as planned.

Indeed, we have $|n| = 2l$ or $|n| = 2l + 1$ if $p \approx 2^{l/4}$. In terms of efficiency of Jacobian arithmetic over $\mathbb{F}_q$, there is hardly any difference in the running times between these two cases. However, for index arithmetic (modulo n), the case $|n| = 2l + 1$ introduces some inefficiency. If we use 32-bit words to pack fragments of multiple-precision integers, then for the stated values of l, we need an extra word compared to the case $|n| = 2l$. This may be an issue for some cryptographic algorithms.

We present a set of curves that were obtained by our approach. For efficiency reasons, we take $\mathcal{C}$ of the form $y^2 = x^5 + x + a$. We vary a in the range $[0, 1000]$ and record all the cases where n is a prime. The database of curves obtained by this algorithm is presented in the appendix. A few examples are presented here. All these examples correspond to the 20-bit prime $p = 1048571$. The resulting curves provide 80-bit security.

Example 1
- Curve $\mathcal{C}_1 : y^2 = x^5 + x + 47$
- $|\mathbb{J}_p| = 1099928953312 = 2^{40} + 417325536$
- Count of rational points on $\mathcal{C}_1$ over $\mathbb{F}_p$ is 1048979
- This gives
  $|\mathbb{J}_q| = 1606861421126112580388908685296656425664857224973157020278432$
  $= 2^{200} - 766231328776951530534070445061768573457688096358150229 44$
- The cofactor
  $n = |\mathbb{J}_q|/|\mathbb{J}_p|$
  $= 1460877465119621059080883122151454896336021166011$
  $= 2^{160} - 624172211281859122801710564828123319911376965$ is prime. So this curve is accepted.

Example 2
- $\mathcal{C}_2 : y^2 = x^5 + x + 46$
- $|\mathbb{J}_p| = 1097744558000 = 2^{40} - 1767069776$
- Count of rational points on $\mathcal{C}_2$ over $\mathbb{F}_p$ is 1046895
- This gives:
  $|\mathbb{J}_q| = 1606861421126118518527811084904153739543257852153511445450000$
  $= 2^{200} - 766231328717570141510074370088629789451416292813898513 76$
- The cofactor $n = |\mathbb{J}_q|/|\mathbb{J}_p|$
  $= 1463784456425534398803014685411133451998636874275$
  $= 2^{160} + 228281909463148059932985269485043234270433129 9$ is not a prime. So repeat the algorithm.

Example 3
- $\mathcal{C}_3 : y^2 = x^5 + x + 60$
- $|\mathbb{J}_p| = 1098401972048 = 2^{40} - 1109655728$
- Count of rational points on $\mathcal{C}_3$ over $\mathbb{F}_p$ is 1047522
- This gives
  $|\mathbb{J}_q| = 1606861421126117326279311266898329713697223055120690303050128$
  $= 2^{200} - 766231328729492626508254428328888249799386621025322512 48$
- The cofactor $n = |\mathbb{J}_q|/|\mathbb{J}_p|$
  $= 1462908354152060576672027642006156546558828957461$
  $= 2^{160} + 1406716821157658468342809289873526902896414485$, even though prime, may be discarded because $n > 2^{160}$.

3.3 Arithmetic of Divisors

Now that we have a family of curves available to us, we concentrate on the algorithms for doing arithmetic in the Jacobians of these curves.

Cantor's Arithmetic

The inverse of a reduced divisor $(u(x), v(x))$ is $(u(x), -v(x))$. The additive identity in the Jacobian has the Mumford representation $(1, 0)$. Let $D_1 = (u_1, v_1)$ and $D_2 = (u_2, v_2)$ be two reduced divisors on the given hyperelliptic curve C. We target to compute the unique reduced divisor (u, v) of the sum $D_1 + D_2$ in the Mumford representation. Cantor [6] provides the following algorithm for computing $D_1 + D_2$. Improvements in this algorithm can be found in [42].

Algorithm 1 Cantor's Addition Algorithm

INPUT: Two divisor classes $D_1 = (u_1, v_1)$ and $D_2 = (u_2, v_2)$ on the curve C.
OUTPUT: The unique reduced Mumford divisor $D = (u, v) = D_1 + D_2$.

procedure CANTOR(D_1, D_2)
$d_1 \longleftarrow \gcd(u_1, u_2)$; ▷ $d_1 = e_1 u_1 + e_2 u_2$
$d \longleftarrow \gcd(d_1, v_1 + v_2 + h)$; ▷ $d = c_1 d_1 + c_2(v_1 + v_2 + h)$
$s_1 \longleftarrow e_1 c_1$ $s_2 \longleftarrow e_2 c_1$ and $s_3 \longleftarrow c_2$
$u \longleftarrow \dfrac{u_1 u_2}{d^2}$ and $v \longleftarrow \dfrac{u_1 v_2 s_1 + u_2 v_1 s_2 + s_3(v_1 v_2 + f)}{d} \pmod{u}$
  while $\deg u \leq g$ **do**
    $u' \longleftarrow \dfrac{f - vh - v^2}{u}$
    $v' \longleftarrow (-h - v) \pmod{u'}$
    $u \longleftarrow u'$
    $v \longleftarrow v'$
make u monic
return $D = (u, v)$ ▷ Reduced divisor

Thomas Wollinger mentions (in his Ph.D. thesis [70]) that the explicit version of Cantor's addition algorithm takes $2I + 44M + 4S$ field operations for hyperelliptic curves of genus two over arbitrary finite fields. Here, I stands for field inversion, S for squaring, and M for multiplication of field elements. The doubling formula takes $2I + 42M + 8S$ field operations. Simpler versions of this algorithm are required for the sake of efficiency. Robert Harley provides an optimized practical method.

Harley's Explicit Formulas

Harley's algorithm, for addition in genus-two hyperelliptic curves, is first published in [27]. Later, Harley provides a complete description in his web page [32]. He also includes sample C codes for doubling. The algorithm is based on the theory and the tools presented in Mumford's textbook [53]. It avoids the computation of quadratic forms related to hyperelliptic-curve function fields, and extends the so-called chord-and-tangent law for point addition on elliptic curves. Special attention to different types of divisors is a crucial issue to optimize the field operations.

We now give the details of Harley's addition and doubling algorithms. We use genus-two hyperelliptic curves of the form (2.1) over finite fields having arbitrary characteristics. For odd-characteristic fields, we fix $h = 0$. We assume that in addition operations, the two divisors are co-prime to each other and also to their opposites. Harley provides these subexpressions for addition.

- $k = (f - v_2 h - v_2^2)/u_2$
- $s = (u_2)^{-1}(v_1 - v_2) \pmod{u_1}$
- $l = u_2 s$
- $u = (k - s(h + l + 2v_2))/u_1$
- $u' = u$ made monic

- $v' = -h - (v_2 + l) \pmod{u'}$

Lange simplifies these subexpressions, and counts the number of field operations [43]. The explicit version takes $2I + 3S + 24M$ for addition, and $2I + 6S + 24M$ for doubling. Here, we deal with genus-two curves only. The addition algorithm is presented now.

Algorithm 2 Addition ($\deg u_1 = \deg u_2 = 2$) by Harley's formulas

INPUT: Two divisor classes $D_1 = (u_1, v_1)$ and $D_2 = (u_2, v_2)$, where $u_i = x^2 + u_{i1}x + u_{i0}$ and $v_i = v_{i1}x + v_{i0}$.
OUTPUT: The unique reduced Mumford divisor $D = (u, v) = D_1 + D_2$.

$k = (f - v_2 h - v_2^2)/u_2$

`Subexpressions:` ▷ $M + S$
$d = u_{11}u_{21}$
$t = u_{20} - d + u_{11}^2 - u_{10}$
`Resultant:` ▷ $S + 2M$
$r = u_{20}(t - u_{10}) + u_{10}(u_{21}^2 - d + u_{10})$
`Inverse of` u_2 `modulo` u_1`:` ▷ $I + 2M$
$inv = ((u_{11} - u_{21})x + t)/r$
$s = (v_1 - v_2)inv \pmod{u_1}$ ▷ $5M$
$l = u_2 s$ ▷ $3M$
$u = (k - s(h + l + 2v_2))/u_1$ ▷ $S + 6M$
$u' = u$ made monic ▷ $I + 2M$
$v' = -h - (l + v_2) \pmod{u'}$ ▷ $3M$
return (u', v') ▷ Total $2I + 3S + 24M$

Compared to Cantor's algorithm, there is a significant improvement in terms of squaring and multiplication, but there no change in the number of inversions. Matsuo [49] modifies Harley's algorithm, and reduces the number of multiplications for addition and doubling. In 2002, Lange made a case study on the addition of different types of divisors, and provides optimized algorithms for addition and doubling. This work reduces one inversion.

Lange's addition algorithm which we present now also assumes that the two input divisors (of degree two) are co-prime to each other and to their opposites.

Algorithm 3 Addition ($\deg u_1 = \deg u_2 = 2$) by Lange's formulas

INPUT: Two divisor classes $D_1 = (u_1, v_1)$ and $D_2 = (u_2, v_2)$, where $u_i = x^2 + u_{i1}x + u_{i0}$ and $v_i = v_{i1}x + v_{i0}$.
OUTPUT: The unique reduced Mumford divisor $D = D_1 + D_2 = (u, v)$.

`Compute` $r = Res(u_1, u_2)$`:` ▷ $1S + 3M$
$z_1 = u_{11} - u_{21}$, $z_2 = u_{20} - u_{10}$, $z_3 = z_1 u_{11} + z_2$ and $r = z_3 z_2 + u_{10} z_1^2$

`Compute almost inverse of` $u_2 \pmod{u_1}$ $(inv = r/u_2 \pmod{u_1})$`:`
$inv_0 = z_3$ and $inv_1 = z_1$

`Compute` $s' = sr = (inv(v_1 - v_2)) \pmod{u_1}$ ▷ $5M$

$w_1 = v_{11} - v_{21}$, $w_0 = v_{10} - v_{20}$, $w_3 = w_1 inv_1$ and $w_2 = w_0 inv_0$
$s'_0 = w_2 - w_3 u_{10}$ and $s'_1 = (w_1 + w_0)(inv_1 + inv_0) - w_3(1 + u_{11}) - w_2$;
if $s'_1 = 0$ then

Compute $s'' = x + s_0/s_1 = x + s'_0/s'_1$ and s_1 ▷ $I + 2S + 5M$
$w_1 = (s'_1 r)^{-1}$, $w_2 = w_1 r$ and $w_3 = w_1 {s'_1}^2$
$w_4 = w_2 r$, $w_5 = {w_4}^2$ and $s''_0 = w_2 s'_0$;
we have $w_1 = 1/r^2 s_1$, $w_2 = \frac{1}{s'_1}$, $w_3 = s_1$ and $w_4 = \frac{1}{s_1}$

Compute $l' = u_2 s'' = x^3 + l'_2 x^2 + l'_1 x + l'_0$ ▷ $2M$
$l'_2 = {s''}_0 + u_{21}$, $l'_1 = u_{21} s''_0 + u_{20}$ and $l'_0 = u_{20} s''_0$

Compute $u' = (s(2v_2 + h + l) - t)/u_1 = x^2 + u'_1 x + u'_0$ ▷ $3M$
$u'_0 = (h_2 w_4 - z_1 + s''_0)(s''_0 - u_{11}) - u_{10}$
$u'_0 = l'_1 + u'_0 + w_4(2v_{21} + h_1) + w_5(z_1 + 2u_{21} - f_4)$
$u_1 = h_2 w_4 + 2s''_0 - w_5 - z_1$

Compute $v' = -h - (v_2 + l) \pmod{u'} = v'_1 x + v'_0$ ▷ $4M$
$w_1 = l'_2 - u'_1$, $w_2 = u'_0 + w_1 u'_1 - l'_1$ and $v'_1 = w_3 w_2 + h_2 u'_1 - h_1 - v_{21}$
$w_2 = w_1 u'_0 - l'_0$ and $v'_0 = w_3 w_2 + h_2 u'_0 - h_0 - v_{20}$

return $[u', v']$ ▷ Total $I + 22M + 3S$

Special case $s = s_0$

Compute s ▷ $I + M$
$inv = r^{-1}$ and $s_0 = inv s'_0$

Compute $u' = t - s(2v_2 + h + l)/u_1 = x + u'_0$ ▷ S
$u_0 = f_4 - u_{11} - u_{21} - s_0(h_2 + s_0)$

Compute $v' = -h - (l + v_2) \pmod{u'} = v'_0$ ▷ $2M$
$w_1 = s_0(u'_0 + u_{21}) + v_{21} + h_1 - h_2 u'_0$ and $w_2 = h_0 + v_{20} + s_0 u_{20}$
$v'_0 = w_1 u'_0 - w_2$

return (u', v') ▷ Total $I + 11M + 2S$

For the doubling algorithm, we again assume that the divisor has degree two.

Algorithm 4 Doubling ($\deg u = 2$) by Lange's formulas

INPUT: Divisor class $D = (u, v)$
OUTPUT: The unique reduced Mumford divisor $D = (u', v') = 2D_1 = 2(u, v)$, where $u = x^2 + u_1 x + u_0$ and $v = v_1 x + v_0$.

Compute $\widetilde{v} = (2v + h) \pmod{u} = \widetilde{v}_1 x + \widetilde{v}_0$
$\widetilde{v}_1 = 2v_1 + h_1 - u_1 h_2$ and $\widetilde{v}_0 = 2v_0 + h_0 - u_0 h_2$

Compute resultant $r = Res(u, \widetilde{v})$ ▷ $3M + 2S$
$w_0 = v_1^2$, $w_1 = u_{21}$ and $w_2 = \widetilde{v}_1^2$
$w_3 = \widetilde{v}_1 u_1$ and $r = \widetilde{v}_0(\widetilde{v}_0 - w_3) + w_2 u_0$

Compute $inv' = r\ inv$
$inv'_0 = \widetilde{v}_0 - w_3$ and $inv'_1 = -\widetilde{v}_1$

$\underline{\texttt{Compute } t' = (f - hv - v^2)/u \pmod u = t_0 + t'_1 x}$ ▷ M
$w_3 = w_1 + f_3$, $w_4 = 2u_0$
$t'_0 = u_1(2w_4 - w_3 + h_2 v_1 + f_4 u_1) - w_0 + f_2 - h_2 v_0 - h_1 v_1 - 2f_4 u_0$
$t'_1 = 2(w_1 - f_4 u_1) - h_2 v_1 - w_4 + w_3$

$\underline{\texttt{Compute } s' = (inv' t') \pmod u}$ ▷ $5M$
$w_0 = inv'_0 t'_0$ and $w_1 = inv_1 t'_1$
$s'_0 = w_0 - w_1 u_0$ and $s'_1 = (t'_0 + t'_1)(inv'_1 + inv'_0) - w_1(u_1 + 1) - w_0$
if $s'_1 = 0$ then

$\underline{\texttt{Compute } s'' = x + s_0/s_1 \texttt{ and } s_1}$ ▷ $I + 5M + 2S$
$w_1 = 1/(s'_1 r)$, $w_2 = w_1 r$ and $w_3 = w_1 s'_1$
$w_4 = w_2 r$, $w_5 = w_4^2$ and $s_0 = w_2 s'_0$
we have $w_1 = 1/(s'_1 r^2)$, $w_2 = \frac{1}{s'_1}$, $w_3 = s_1$ and $w_4 = \frac{1}{s_1}$

$\underline{\texttt{Compute } l' = us'' = x^3 + l'_2 x^2 + l'_1 x + l'_0}$ ▷ $2M$
$l'_0 = s''_0 u_0$, $l'_1 = u_0 + s''_0 u_1$ and $l'_2 = s''_0 + u_1$

$\underline{\texttt{Compute } u' = s^2 + (2v + h)s/u + (v^2 + hv - f)/u^2}$ ▷ $2M + S$
$u'_0 = s''_0 + w_5(2u_1 - f_4) + w_4(2v_1 + h_2(s''_0 - u_1) + h_1)$
$u'_1 = h_2 w_4 + 2s''_0 - w_5$

$\underline{\texttt{Compute } v' = -h - (l + v) \pmod{u'} = v'_1 x + v'_0}$ ▷ $4M$
$w_1 = l'_2 - u'_1$, $w_2 = w_1 u'_1 - l'_1 + u'_0$ and $v'_1 = w_3 w_2 + h_2 u'_1 - h_1 - v_1$
$w_2 = w_1 u'_0 - l'_0$ and $v'_0 = w_3 w_2 + h_2 u'_0 - h_0 - v_0$

return $[u', v']$ ▷ Total $I + 22M + 5S$

`Special case` $s = s_0$

$\underline{\texttt{Compute } s \texttt{ and precomputations}}$ ▷ $I + 2M$
$w_1 = 1/r$, $s_0 = w_1 s'_0$ and $w_2 = v_0 + s_0 u_0 + h_0$

$\underline{\texttt{Compute } u' = (f - hv - v^2)/u^2 - (h + 2v)s/u - s^2}$ ▷ S
$u'_0 = f_4 - 2u_1 - h_2 s_0 - s_0^2$

$\underline{\texttt{Compute } v' = -h - (su + v) \pmod{u'}}$ ▷ $2M$
$w_1 = v_1 - h_2 u'_0 + s_0(u_1 - u_0) + h_1$ and $v'_0 = w_1 u'_0 - w_2$

return $[u', v']$ ▷ Total $I + 13M + 3S$

Projective Coordinates

In Chapter 1, projective coordinates are introduced. For affine coordinates, a divisor $D = (u, v)$ can be explicitly written as (u_1, u_0, v_1, v_0). For projective coordinates, one new variable Z is introduced, and this tuple becomes a quintuple (u_1, u_0, v_1, v_0, Z). This quintuple can be viewed as $(x^2 + \frac{u_1}{Z}x + \frac{u_0}{Z}, \frac{v_1}{Z}x + \frac{v_0}{Z})$. To get back the divisor in affine coordinates, one puts $Z = 1$. Divisor-class arithmetic using projective coordinates does not require any field inversion.

The use of projective coordinates for hyperelliptic curves is proposed by Miyamoto in Japanese, and later improved in [44]. Also see [7]. In the algorithm that follows, we deal with a genus-two hyperelliptic curve over a finite field of odd characteristic.

Algorithm 5 Addition ($\deg u_1 = \deg u_2 = 2$) using projective coordinates

INPUT: Two divisor classes $D_1 = (U_1, V_1)$ and $D_2 = (U_2, V_2)$ of the form $U_i(x) = x^2 + \frac{U_{i1}}{Z_i}x + \frac{U_{i0}}{Z_i}$ and $V_i(x) = \frac{V_{i1}}{Z_i}x + \frac{V_{i0}}{Z_i}$.

OUTPUT: The unique reduced Mumford divisor $D = D_1 + D_2 = (U', V')$, where $U' = x^2 + \frac{U'_2}{Z'}x + \frac{U'_1}{Z'}$ and $V' = \frac{V'_1}{Z'}x + \frac{V'_0}{Z'}$.

`Precomputations` ▷ $5M$

$Z = Z_2Z_1$
$\widetilde{U}_{21} = U_{21}Z_1$, $\widetilde{U}_{20} = U_{20}Z_1$
$\widetilde{V}_{21} = V_{21}Z_1$, $\widetilde{V}_{20} = V_{20}Z_1$

`Compute` $r = Res(U_1, U_2)$`:` ▷ $1S + 6M$

$z_1 = Z_2U_{11} - \widetilde{U}_{21}$, $z_2 = \widetilde{U}_{20} - Z_2U_{10}$
$z_3 = z_1U_{11} + Z_1z_2$ and $r = z_3z_2 + U_{10}z_1^2$

`Compute almost inverse of` $U_2 \pmod{U_1}$ $(inv = r/U_2 \pmod{U_1})$`:`

$inv_0 = z_3$, $inv_1 = z_1$

`Compute` s ▷ $5M$

$w_0 = Z_2V_{10} - \widetilde{V}_{20}$ and $w_1 = Z_2V_{11} - \widetilde{V}_{21}$,
$w_3 = w_1inv_1$ and $w_2 = w_0inv_0$
$s_1 = (w_1 + w_0)(inv_0 + inv_1Z_1) - w_3(Z_1 + U_{11}) - w_2$
$s_0 = w_2 - w_3U_{10}$

`Precomputations` ▷ $S + 8M$

$R = rZ$, $s_0 = Zs_0$, $s_3 = Zs_1$ and $\widetilde{R} = s_3R$
$S = s_1s_0$, $S_3 = s_3^2$ and $\widetilde{S} = s_1s_3$
$\widehat{R} = \widetilde{S}\widetilde{R}$ and $\widehat{S} = s_0s_3$

`Compute` l ▷ $3M$

$l_0 = \widetilde{U}_{20}S$ and $l_2 = \widetilde{U}_{21}\widetilde{S}$
$l_1 = (S + \widetilde{S})(\widetilde{U}_{20} + \widetilde{U}_{21}) - l_0 - l_2$
$l_2 = \widehat{S} + l_2$

`Compute` U' ▷ $2S + 8M$

$U'_0 = s_0^2 + z_1s_1(s_1(\widetilde{U}_{21} + z_1) - 2s_0) + \widetilde{S}z_2 + R(2\widetilde{V}_{21}s_1 + r(2\widetilde{U}_{21} + z_1 - f_4Z))$
$U'_1 = 2\widehat{S} - R^2 - z_1\widetilde{S}$

`Precomputations` ▷ $4M$

$l_2 = l_2 - U'_1$ $w_2 = U'_0l_2 - l_0S_3$ and $w_1 = S_3(U'_0 - l_1) + l_2U'_1$

`Adjust` ▷ $3M$

$Z' = S_3\widetilde{R}$, $U'_1 = U'_1\widetilde{R}$ and $U'_0 = U'_0\widetilde{R}$

`Compute` V' ▷ $2M$

$V'_0 = w_0 - \widetilde{V}_{20}\widehat{R}$ and $V'_1 = w_1 - \widetilde{V}_{21}\widehat{R}$

return $(U', V') = (U'_1, U'_0, V'_1, V'_0, Z')$ ▷ Total $47M + 4S$

The doubling operation for genus-two hyperelliptic curves over finite fields of odd characteristics is now presented.

Algorithm 6 Doubling ($\deg u = 2$) using projective coordinates

INPUT: Divisor class $D_1 = (U_1, V_1)$ of the form $U(x) = x^2 + \frac{U_1}{Z}x + \frac{U_0}{Z}$ and $V(x) = \frac{V_1}{Z}x + \frac{V_0}{Z}$.
OUTPUT: The unique reduced Mumford divisor $D = 2D_1 = (U', V')$, where $D = (U', V')$, $U' = x^2 + \frac{U_2'}{Z'}x + \frac{U_1'}{Z'}$ and $V' = \frac{V_1'}{Z'}x + \frac{V_0'}{Z'}$.

`Compute` $r = Res(\widetilde{V}, U)$ ▷ $4M + 3S$
$Z_2 = Z^2$, $\widetilde{V}_1 = 2V_1$ and $\widetilde{V}_0 = 2V_0$
$w_1 = U_1^2$, $w_0 = V_1^2$ and $w_2 = 4w_0$
$w_3 = \widetilde{V}_0 Z - U_1\widetilde{V}_1$ and $r = \widetilde{V}_0 w_3 + w_2 U_0$

`Compute almost inverse :`
$inv_0 = w_3$, $inv_1 = -\widetilde{V}_1$

`Compute` t`:` ▷ $5M$
$w_3 = f_3 Z_2 + w_1$, $w_4 = 2U_0$, $t_1 = w_3 - Z(w_4 + 2f_4U_1 + 2w_1)$
$t_0 = Z(Z(Zf_2 - 2U_0 f_4) - w_0) + U_1(Z(2w_4 + U_1 f_4) - w_3)$

`Compute` s ▷ $7M$
$w_0 = inv_0 t_0$ and $w_1 = inv_1 t_1$
$s_3 = (t_1 + t_0)(inv_1 + inv_0) - w_3(1 + U_1) - w_0$
$s_1 = Zs_3$ and $s_0 = w_0 - w_1 U_0 Z$

`Precomputations` ▷ $2S + 6M$
$R = Z_2 r$, and $\widetilde{R} = s_1 R$, $S1 = s_1^2$ and $S_0 = s_0^2$
$s_0 = s_3 s_0$, $s_1 = s_3 s_1$, $S = Zs_0$, and $\widehat{R} = s_1\widetilde{R}$

`Compute` l ▷ $3M$
$l_0 = s_0 U_0$ and $l_2 = s_1 U_1$
$l_1 = (U_0 + U_1)(s_0 + s_1) - l_0 - l_2$

`Compute` U' ▷ $2S + 8M$
$U_0' = S_0 + R(2V_1 s_3 + rZ(2U_1 - f_4 Z))$
$U_1' = 2S - R^2$

`Precomputations` ▷ $4M$
$l_2 = S + l_2 - U_1'$, $w_0 = l_2 U_0' - l_0 S_1$ and $w_1 = l_2 U_1' + S_1(U_0' - l_1)$

`Adjust` ▷ $3M$
$Z' = \widetilde{R}S_1$, $U_1' = U_1'\widetilde{R}$ and $U_0' = U_0'\widetilde{R}$

`Compute` V' ▷ $2M$
$V_0' = w_0 - V_0\widehat{R}$ and $V_1' = w_1 - V_1\widehat{R}$

return $(U', V') = (U_1', U_0', V_1', V_0', Z')$ ▷ Total $38M + 6S$

Weighted Coordinates

The use of weighted coordinates is first proposed by Lange [44]. A sextuple $(U_1, V_0, V_1, V_0, Z_1, Z_2)$ of a divisor $D = (U, V)$ represents the affine class $(x^2 + \frac{U_1}{Z_1^2}x + \frac{U_0}{Z_1^2}, \frac{V_1}{Z_1^3 Z_2}x + \frac{V_0}{Z_1^3 Z_2})$. This tuple needs one more variable than projective coordinates. To enhance the performance of

divisor-class arithmetic further, two more variables z_1 and z_2 are introduced. These new variables are connected by the relations $z_1 = Z_1^2$, $z_2 = Z_2^2$. We therefore have an 8-tuple $(U_1, U_0, V_1, V_0, Z_1, Z_2, z_1, z_2)$ to represent a reduced divisor. As earlier, $(U_1, U_0, V_1, V_0, 1, 1, 1, 1)$ represents the divisor (U_1, U_0, V_1, V_0) in affine coordinates.

Algorithm 7 Addition ($\deg u_1 = \deg u_2 = 2$) using weighted coordinates

INPUT: Two divisor classes $D_1 = (U_1, V_1)$ and $D_2 = (U_2, V_2)$ of the form $U_i(x) = x^2 + \frac{U_{i1}}{Z_{1i}^2}x + \frac{U_{i0}}{Z_{1i}^2}$ and $V_i(x) = \frac{V_{i1}}{Z_{1i}^3 Z_{2i}}x + \frac{V_{i0}}{Z_{1i}^3 Z_{2i}}$.

OUTPUT: The unique reduced Mumford divisor $D = D_1 + D_2 = (U', V')$, where $U' = x^2 + \frac{U_2'}{Z_1'^2}x + \frac{U_1'}{Z_1'^2}$ and $V' = \frac{V_1'}{Z_1'^3 Z_2'}x + \frac{V_0'}{Z_1'^3 Z_2'}$.

`Precomputations` ▷ $8M$

$z_{23} = Z_{21}Z_{22}$, $z_{13} = Z_{11}Z_{12}$, $z_{14} = z_{11}z_{13}$, $z_{24} = z_{23}z_{21}$
$\widetilde{U}_{21} = z_{11}U_{21}$, $\widetilde{U}_{20} = z_{11}U_{20}$, $\widetilde{V}_{21} = z_{14}V_{21}$, $\widetilde{V}_{20} = z_{14}V_{20}$

`Compute` $r = Res(U_1, U_2)$: ▷ $4S + 11M$

$y_1 = z_{21}U_{11} - \widetilde{U}_{21}$, $y_2 = \widetilde{U}_{21} - z_{21}U_{10}$ and $y_3 = y_1U_{11} + z_{11}y_2$
$r = y_2y_3 + y_1^2U_{10}$, $Z_2' = Z_{11}Z_{21}$, $\widetilde{Z}_2 = Z_{12}Z_{22}$ and $Z_1' = Z_2'^2$
$\widetilde{Z}_2 = rZ_1'\widetilde{Z}_2$, $Z_2' = \widetilde{Z}_2Z_2'$, $\widetilde{Z}_2 = \widetilde{Z}_2^2$ and $z_2' = Z_2'^2$

`Compute almost inverse of` $U_2 \pmod{U_1}$ $(inv = r/U_2 \pmod{U_1})$:

$inv_0 = y_3$, $inv_1 = y_1$

`Compute` s ▷ $8M$

$w_0 = V_{10}z_{24} - \widetilde{V}_{20}$, and $w_1 = z_{24}V_{11} - \widetilde{V}_{21}$
$w_2 = w_0inv_0$ and $w_3 = w_1inv_1$
$s_1 = (z_{11}inv_1 + inv_0)(w_1 + w_0) - w_3(z_{11} + U_{11}) - w_2$
$s_0 = w_2 - w_3U_{10}$

`Precomputations` ▷ $3S + 6M$

$S_1 = s_1^2$, $S_0 = Z_1's_0$, $Z_1' = Z_1's_1$, $S = S_0Z_1'$ and $S_0 = S_0^2$
$R = Z_1'r$, $s_0 = s_0Z_1'$
$s_1 = s_1Z_1'$, $z_1' = Z_1'^2$

`Compute` l ▷ $3M$

$l_2 = s_1\widetilde{U}_{21}$ and $l_0 = s_0\widetilde{U}_{20}$
$l_1 = (\widetilde{U}_{21} + \widetilde{U}_{20})(s_1 + s_0) - l_2 - l_0$ and $l_2 = S + l_2$

`Compute` U' ▷ $6M$

$V_1' = \widetilde{V}_{21}R$
$U_0' = S_0 + \widetilde{Z}_2(2\widetilde{U}_{21} + y_1) + y_1(S_1(y_1 + \widetilde{U}_{21}) - 2s_0) + s_1y_2 + 2V_1'$
$U_1' = 2S - y_1s_1 - z_2'$

`Precomputations` ▷ $2M$

$l_2 = l_2 - U_1'$,
$w_1 = U_1'l_2$ and $w_0 = l_2U_0'$

`Compute` V' ▷ $3M$

$V_0' = w_0 - z_1'(l_0 + \widetilde{V}_{20}R)$ and $V_1' = w_1 - z_1'(l_1' - U_0' + V_1')$

return $(U', V') = (U_1', U_0', V_1', V_0', Z_1', Z_2', z_1', z_2')$ ▷ Total $47M + 7S$

The doubling algorithm for weighted coordinates is as follows.

Algorithm 8 Doubling ($\deg u = 2$) using weighted coordinates

INPUT: Divisor class $D_1 = (U_1, V_1)$ of the form $U_(x) = x^2 + \frac{U_1}{Z_1^2}x + \frac{U_0}{Z_1^2}$ and $V_(x) = \frac{V_1}{Z_1^3 Z_2}x + \frac{V_0}{Z_1^3 Z_2}$.

OUTPUT: Compute unique reduced Mumford divisor $D = 2D_1$, where $D = (U', V')$, $U' = x^2 + \frac{U_2'}{Z_1'^2}x + \frac{U_1'}{Z_1'^2}$ and $V' = \frac{V_1'}{Z_1'^3 Z_2'}x + \frac{V_0'}{Z_1'^3 Z_2'}$.

`Compute` $r = Res(\widetilde{V}, U)$ `and precomputations` ▷ $8M + 3S$
$\widetilde{U}_0 = z_1 U_0$, $w_0 = V_1^2$, $w_1 = U_1^2$ and $w_3 = z_1 V_0 - V_1 U_1$
$r = U_0 w_0 + w_3 V_0$, $\widetilde{Z}_2 = r z_1 Z_2$, $Z_2' = 2 Z_1 \widetilde{Z}_2$ and $\widetilde{Z}_2 = \widetilde{Z}_2^2$

`Compute almost inverse:`
$inv_0 = w_3$, $inv_1 = -V_1$

`Compute` t`:` ▷ $6M + S$
$z_3 = z_1^2$, $w_3 = w_1 + z_3 f_3$, and $t_1 = z_2(w_3 + 2(w_1 - \widetilde{U_0}))$
$z_3 = z_1 z_3$, and $t_0 = z_2(U_1(4\widetilde{U}_0 - w_3) + f_2 z_3) - w_0$

`Compute` s ▷ $5M$
$w_0 = inv_0 t_0$ and $w_1 = inv_1 t_1$
$s_3 = (t_0 + t_1)(inv_0 + inv_1) - w_3(1 + U_1) - w_0$
and $s_0 = w_0 - \widetilde{U}_0 w_1$

`Precomputations` ▷ $3S + 5M$
$Z_1' = z_1 s_1$, $S_0 = s_0^2$, $z_1' = Z_1'^2$ and $S = Z_1' s_0$
$z_2' = Z_2'^2$, $R = Z_1' r$, $s_0 = s_1 s_0$ and $s_1 = s_1 Z_1'$

`Compute` l ▷ $3M$
$l_2 = s_1 U_1$ and $l_0 = s_0 U_0$
$l_1 = (U_0 + U_1)(s_0 + s_1) - l_2 - l_0$
$l_2 = S + l_2$

`Compute` U' ▷ $2M$
$V_1' = V_1 R$, $U_0' = 4(V_1' + 2U_1 \widetilde{Z}_2)$ and $U_1' = 2S - z_2'$

`Precomputations` ▷ $2M$
$l_2 = l_2 - U_1'$, $w_1 = U_1' l_2$ and $w_0 = U_0' l_2$

`Compute` V' ▷ $3M$
$V_1' = w_1 - z_1'(2V_1' + l_1 - U_0')$
$V_0' = w_0 - z_1'(2V_0 R + l_0)$

return $(U', V') = (U_1', U_0', V_1', V_0', Z_1', Z_2', z_1', z_2')$ ▷ Total $34M + 7S$

Further optimizations are reported in the literature [7, 8, 35]. Table 3.1 lists the numbers of arithmetic operations needed to perform addition and doubling for elliptic and hyperelliptic curves.

Algorithms	Addition	Doubling
Elliptic-Curve Arithmetic	$I + 2M + S$	$I + 2M + 2S$
Cantor's Algorithm	$2I + 44M + 4S$	$2I + 42M + 8S$
Harley's Formula	$2I + 24M + 3S$	$2I + 24M + 6S$
Matsuo's Improvement	$2I + 22M + S$	$2I + 23M + 2S$
Lange's Explicit Version	$I + 22M + 3S$	$I + 22M + 5S$
Projective Coordinate	$47M + 4S$	$38M + 6S$
Weighted Coordinate	$47M + 7S$	$34M + 7S$
Costello and Lauter [8]	43M + 4S	30M + 9S
Hisil and Costello [35]	41M + 7S	28M + 8S

Table 3.1: Divisor-class addition and doubling algorithms

3.4 Discrete Logarithm Problem for Subfield Curves

In Chapter 2, we talk about the discrete logarithm problem. There we focus only on generic attacks. In this section, we argue that our family of curves is secure.

We first consider an attack proposed in [7]. Let $\mathbb{J}_q$ be the Jacobian of a genus-g hyperelliptic curve defined over $\mathbb{F}_{p^d}$. Suppose that $p||\mathbb{J}_q|$. There exists a morphism from $\mathbb{J}_q$ to the $\mathbb{F}_q$-vector space of holomorphic differentials of the curve. This vector space and $\mathbb{F}_q^{2g-1}$ are isomorphic. The complexity of computing the map is $O(\log q)$. As a result, discrete logarithms in $\mathbb{J}_q$ are efficiently mapped to those in $\mathbb{F}_q^{2g-1}$. The time complexity of the method is $O((2g-1)\log q^k)$ for a small constant k. For our family, we therefore need to ensure that the condition $p||\mathbb{J}_q|$ does not hold. If this condition holds, we must discard the curve.

The Weil-descent attack reduces the DLP from $E_{\mathbb{F}_{p^d}}$ to the Jacobian of a curve C_p, and computes the discrete logarithm by the index calculus method on the Jacobian. Gaudry, Hess and Smart develop a Weil-descent method for elliptic curves defined over binary fields $\mathbb{F}_{2^d}$ [28]. Galbraith [21] generalizes the attack to hyperelliptic curves defined over even binary extension fields. Diem [10] studies elliptic and hyperelliptic curves over finite extension fields of odd characteristics. Diem's work is the most relevant in the current context. In particular, he shows that when the extension degree d is five, there exist potentially vulnerable elliptic curves. This attack therefore does not apply to our family of hyperelliptic curves. Hess [34] generalizes the Weil-descent construction of the GHS attack to arbitrary Artin–Schreier extensions. However, he concentrates only on small primes like $p = 2, 3$ in his work.

The decomposition attack is mentioned in [25]. Nagao [54] proposes a decomposition attack for hyperelliptic curves over an extension field. For the decomposition of the Jacobian of a genus-g hyperelliptic curve defined over $\mathbb{F}_q = \mathbb{F}_{p^d}$, we need exactly dg divisors. The complexity of this algorithm is $O(q^{2-\frac{2}{ng}})$. In our case, $q \geq 2^{100}$ at all security levels, so this attack is not feasible. The cover decomposition attack on the ECDLP proposed by Joux and Vitse [38] for elliptic curves defined over $\mathbb{F}_{p^6}$ is also not applicable to our family.

Shor's polynomial-time quantum algorithms solve the integer-factoring and the finite-field discrete-logarithm problems [64]. Proos [58] show that Shor's algorithm can solve ECDLP with $O(l)$ qubits and $O(l^3)$ Toffoli gates for a curve over an l-bit field. Huang [37] proposes a quantum algorithm for solving HECDLP over l-bit prime fields using $O(l)$ qubits and $O(l^3)$ Toffoli gates. Replacing the prime-field arithmetic by the extension-field

arithmetic makes Huang's algorithm applicable to our curves as well. We conclude that, like other elliptic and hyperelliptic curves, our family of curves is not considered quantum-safe.

3.5 Mathematical Library

In order to put the performance of our subfield curves to test, we implement a mathematical library. Since these curves use a very specific type of finite fields of the form $\mathbb{F}_{p^5}$ for odd single-precision primes p, we plan to make efficient implementations of the arithmetic in these finite fields. The available general-purpose libraries lack custom-made optimizations tailored to our context.

Our library consists of the following components.

- **A multiple-precision integer library:** Cryptographic primitives require arithmetic modulo the order n of G. This library is also useful for the arithmetic of divisors over prime fields. The multiple-precision integer library caters to this requirement. The arithmetic of G however is not straightaway linked to multiple-precision integers (scalar multiplication involves multiple-precision multipliers though).
- **Arithmetic of prime fields:** The starting point of the Jacobian arithmetic is the arithmetic of $\mathrm{GF}(p) = \mathbb{F}_p$, where p is a single-precision prime.
- **Polynomials over prime fields:** Polynomial arithmetic over $\mathbb{F}_p$ is needed to implement the arithmetic of $\mathbb{J}_p$ and the extension field $\mathbb{F}_q$.
- **Arithmetic of extension fields:** We define $\mathbb{F}_q$ in the standard polynomial-basis representation $\mathrm{GF}(q) = \mathbb{F}_q = \mathbb{F}_p[t]/\langle f(t)\rangle$, where $f(t) \in \mathbb{F}_p[t]$ is an irreducible polynomial of degree five, called the *defining polynomial*. Binomials or trinomials with small non-zero coefficients are chosen as $f(t)$.
- **Polynomials over extension fields:** Polynomials over $\mathbb{F}_q$ are instrumental in implementing the arithmetic of $\mathbb{J}_q$. The elements are bivariate polynomials. The coefficients are elements of $\mathbb{F}_q$, which are polynomials in t over $\mathbb{F}_p$. The variable in a polynomial over $\mathbb{F}_q$ is denoted by x.
- **Jacobian arithmetic over extension fields:** The library continues with the Mumford representation of elements of $\mathbb{J}_q$ as pairs of polynomials over $\mathbb{F}_q$.

3.5.1 Multiple-Precision Integers

The size of the subgroup G over which cryptographic schemes are to be built is a prime $n = |\mathbb{J}_q|/|\mathbb{J}_p|$. At the highest security level, the bits length of n is $|n| = 256$. Index arithmetic calls for an availability of 256-bit multiple-precision arithmetic. Although this can fit in eight 32-bit words, an additional word is used for some functions to work. That is, the multiple-precision integers can accommodate at most $32 \times 9 = 288$ bits in an integer. Another word is used to store the information related to the (actual) word size of the stored integer. In total, an array of ten 32-bit words is used to represent a multiple-precision integer. The array cells have indices $0, 1, 2 \ldots, 9$. The data type is called `hecui` (unsigned integer to be

used in hyperelliptic-curve cryptosystems). It is important to highlight that `hecui` is an *unsigned* integer data type, and cannot store negative integers (of any bit length).

Let a be a *positive* integer of bit length s. The binary representation of a is

$$a = (a_{s-1}a_{s-2}\dots a_2a_1a_0)_2,$$

where each a_i is a bit, and $a_{s-1} = 1$. Let $k = \lceil s/32 \rceil$. The bits of a are stored in an array `hecui` of ten 32-bit unsigned integer words as follows.

9		$k-1$		1	0
$k-1$	$\cdots$	$0\dots0a_{s-1}\dots a_{32(k-1)}$	$\cdots$	$a_{63}\dots a_{34}a_{33}a_{32}$	$a_{31}\dots a_2a_1a_0$

In terms of the radix $R = 2^{32}$, this translates to

$$a = (0\dots0a_{s-1}\dots a_{32(k-1)})_2 \times R^{k-1} + \cdots + (a_{63}\dots a_{34}a_{33}a_{32})_2 \times R + (a_{31}\dots a_2a_1a_0)_2.$$

If k words are needed to represent a, then its maximum radix degree is $k-1$, so $k-1$ (instead of k) is stored in the cell at index 9. Let us call 9 the *degree index* (DEGIDX) of a (not its size index). An integer in this representation is said to be *compact*. Since DEGIDX stores the exact degree of a, the unused cells at indices $k, k+1, \dots, 8$ may have any contents and are not needed to be set to zero.

While it is preferable to store integers in the compact representation, the library allows one or more leading zero words. For example, one may store $k+1$ at DEGIDX. In that case, it is the onus of the programmer to set the k-th and $(k+1)$-st cell of the array to zero. Most of the library functions convert a non-compact representation to the compact one. All results are output by the functions in the compact representation.

As a specific example, consider the 150-bit integer (so $s = 150$ and $k = 5$):

$$a = 1264457234680113638065998229755637130018186384.$$

The hexadecimal representation if a (in groups of eight hexadecimal digits) is

$$a = 38b341\ \ d72ec03c\ \ e6ba740c\ \ 8bf07644\ \ 4b086490.$$

The compact representation of a follows. The cells marked by ? may contain any value.

9	8	7	6	5	4	3	2	1	0
4	?	?	?	?	0038b341	d72ec03c	e6ba740c	8bf07644	4b086490

Here is a non-compact representation of the same integer, which uses three leading zero words.

9	8	7	6	5	4	3	2	1	0
7	?	0	0	0	0038b341	d72ec03c	e6ba740c	8bf07644	4b086490

The integer zero requires special attention. It is the zero polynomial in the radix $R = 2^{32}$. In mathematics, the degree of the zero polynomial is taken as $-\infty$. This not being an integer value, -1 is taken as MINUSINF. This still poses a problem, because the `hecui` array has unsigned integer cells. In a 2's complement machine, the 32-bit signed integer -1 has the same representation as the 32-bit unsigned integer $2^{32} - 1 = 4294967295$. This integer is taken as MINUSINF. Clearly, this big value cannot be the degree of any integer that can be

stored in `hecui`. This is called the compact representation of zero. In this representation, cells at indices $0, 1, 2, \ldots, 8$ can store any values, whereas the cell at index DEGIDX $= 9$ is required to store the special value MINUSINF.

Zero may have non-compact representations too. For example, one may store 0 at indices 0 and 9 in the `hecui` array.

Multiplying two `hecui` integers introduces a fresh problem. The product can have a bit length of 576 and cannot fit in the `hecui` data type. To get around this problem, a doubly multiple-precision integer data type `hecui2` is introduced. The representation mechanism for `hecui2` is the same as that for `hecui`. The sole difference is that the size of a `hecui2` array is 20 (in 32-bit unsigned words), and the degree is stored in the cell at index DEGIDX2 $= 19$. It follows that `hecui2` can store integers of bit length up to $32 \times 19 = 608$. This is slightly larger than the requirement of 576. But then, having an additional word is always safe for all the functions to work appropriately.

3.5.2 Prime-Field Arithmetic

This data structure is rather trivial. Since we eventually work in quintic extensions, we restrict the base field to $\mathrm{GF}(p) = \mathbb{F}_p$, where p is a single-precision prime (that is, $|p| \leq 32$). We have the standard representation of $\mathbb{F}_p$ as the set

$$\mathbb{F}_p = \{0, 1, 2, \ldots, p-1\}.$$

Therefore each element in $\mathbb{F}_p$ fits in a 32-bit unsigned integer data type. The arithmetic of $\mathbb{F}_p$ is integer arithmetic modulo the prime p. So we rename `ui` or `unsigned long int` as `GFpel`.

3.5.3 Prime-Field Polynomial Arithmetic

Polynomial arithmetic over $\mathbb{F}_p$ is needed for two purposes: Jacobian arithmetic over $\mathbb{F}_p$, and arithmetic over $\mathbb{F}_{p^5}$. For both these purposes, we need to deal only with polynomials of degrees at most eight. Such a polynomial has at most nine coefficients. We also need to store the degree of the polynomial. Moreover, p is in our library always a single-precision prime. Consequently, an array of ten 32-bit unsigned integers suffices to store a polynomial over $\mathbb{F}_p$.

Consider a non-zero polynomial of degree $d \geq 0$

$$a(t) = a_d x^d + a_{d-1} t^{d-1} + \cdots + a_2 t^2 + a_1 t + a_0,$$

where $a_0, a_1, a_2, \ldots, a_d \in \mathbb{F}_p = \{0, 1, 2, \ldots, p-1\}$, and $a_d \neq 0$. The *compact representation* of this polynomial stores the coefficients $a_0, a_1, a_2, \ldots, a_d$ at indices $0, 1, 2, \ldots, d$ in the polynomial array. The *degree index* (DEGIDX) in the array is nine. We store the degree d at this index. The array elements $a_{d+1}, a_{d+2}, \cdots, a_8$ can store any value, and need not be initialized to 0.

9		d		2	1	0
d	$\cdots$	a_d	$\cdots$	a_2	a_1	a_0

The library also allows *non-compact representations* of polynomials, where one or more leading zero coefficients are stored. For example, for the above polynomial, one can store

any $d' \in [d+1, 8]$ at DEGIDX. In that case, the array elements $a_{d+1}, a_{d+2}, \ldots, a_{d'}$ must be set to zero.

As a specific example, consider the polynomial

$$a(t) = 994042t^4 + 1041076t^3 + 861038t^2 + 926874t + 1008193$$

over $\mathbb{F}_{1048571}$. The compact representation of this polynomial is given first. Here, a cell marked by ? is allowed to contain any value.

9	8	7	6	5	4	3	2	1	0
4	?	?	?	?	994042	1041076	861038	926874	100819

A non-compact representation of the same polynomial with two leading zero coefficients is given next.

9	8	7	6	5	4	3	2	1	0
6	?	?	0	0	994042	1041076	861038	926874	100819

Let us now look at the representation of the zero polynomial. Conventionally, the degree of the zero polynomial is $-\infty$. In the compact representation of the zero polynomial, we simply store MINUSINF (which is defined as $2^{32} - 1 = 4294967295$) at DEGIDX. All other cells in the array may contain any value.

9	8	7	6	5	4	3	2	1	0
MINUSINF	?	?	?	?	?	?	?	?	?

Non-compact representations of the zero polynomial are also allowed. Here is an example where the zero polynomial is treated as a polynomial of degree bounded by four.

9	8	7	6	5	4	3	2	1	0
4	?	?	?	?	0	0	0	0	0

We name this data structure as `polyp`. Clearly, `polyp` consists of an array of ten `GFpel` elements.

3.5.4 Extension-Field Arithmetic

The arithmetic of extension fields is needed to implement the arithmetic of polynomials over these fields, which in turn leads to our eventual goal of implementing the Mumford arithmetic over extension fields. Elements of $\mathbb{F}_q$ are represented as polynomials over $\mathbb{F}_p$ with degrees ≤ 4. More precisely, we have

$$\mathbb{F}_q = \{a_0 + a_1\theta + a_2\theta^2 + a_3\theta^3 + a_4\theta^4 \mid a_i \in \mathbb{F}_p\}$$

Here, θ is a root of the defining polynomial $f(t)$, that is, $f(\theta) = 0$. We carry out arithmetic in $\mathbb{F}_q$ as the polynomial arithmetic over $\mathbb{F}_p$ modulo the defining polynomial $f(t)$.

Elements of $\mathbb{F}_q$ are polynomials over $\mathbb{F}_p$ with degrees ≤ 4. We can therefore store an element of $\mathbb{F}_q$ in a `polyp` array which consists of ten unsigned integer variables. An element $a_0 + a_1\theta + a_2\theta^2 + a_3\theta^3 + a_4\theta^4 \in \mathbb{F}_q$ can always be stored in a `polyp` array as shown below.

Here, an array cell marked by ? can store any value. Notice however that as a polynomial, this representation is not necessarily compact.

9	8	7	6	5	4	3	2	1	0
4	?	?	?	?	a_4	a_3	a_2	a_1	a_0

As a specific example, take $p = 1048571$, define the quintic extension $\mathbb{F}_q = \mathbb{F}_p[t]/\langle f(t)\rangle$ by the irreducible polynomial $f(t) = t^5 + 2$, and consider the element $\alpha = 2812t^4 + 499743t^3 + 463157t^2 + 1042895t + 268065 \in \mathbb{F}_q$. A compact representation of α is given first. Since the degree of α is four, we store 4 at the cell at index DEGIDX $= 9$.

9	8	7	6	5	4	3	2	1	0
4	?	?	?	?	2812	499743	463157	1042895	268065

Non-compact representations are allowed too. The above α can also be represented as follows. But since some functions may assume α to have degree ≤ 4, this non-compact representation is avoided.

9	8	7	6	5	4	3	2	1	0
6	?	?	0	0	2812	499743	463157	1042895	268065

An element of $\mathbb{F}_q$ can always be assumed to have *formal* degree four even if its actual degree is less than four. Therefore a non-compact representation of $\beta = 463157t^2 + 1042895t$ can be as shown now. This non-compact representation does not have the technical problem of the non-compact representation of α demonstrated above.

9	8	7	6	5	4	3	2	1	0
4	?	?	?	?	0	0	463157	1042895	0

The element 0 of $\mathbb{F}_q$ is the zero polynomial whose degree is $-\infty$. We define the special value MINUSINF $= 2^{32} - 1 = 4294967295$ to stand for $-\infty$. Therefore the compact representation of 0 is as follows.

9	8	7	6	5	4	3	2	1	0
MINUSINF	?	?	?	?	?	?	?	?	?

One may also go for a non-compact representation too. For example, if 0 is again treated as a polynomial of formal degree four, we have the following representation.

9	8	7	6	5	4	3	2	1	0
4	?	?	?	?	0	0	0	0	0

We rename `polyp` as `GFqel`.

3.5.5 Extension-Field Polynomial Arithmetic

Polynomial arithmetic over $\mathbb{F}_q$ is at the heart of Jacobian arithmetic over $\mathbb{F}_q$. Elements from $\mathbb{F}_q$ are themselves polynomials over $\mathbb{F}_p$, and are represented by arrays of elements from $\mathbb{F}_p$. Therefore polynomials over $\mathbb{F}_q$ are bivariate polynomials, and are represented by two-dimensional arrays. We have the quintic extension $\mathbb{F}_q = \mathbb{F}_p(\theta)$. We use t in the text mode (in place of θ) for representing elements of $\mathbb{F}_q$. The polynomials over $\mathbb{F}_q$ have the variable

x. As far as storage is concerned, we do not need to store the variables t or x. It suffices to store only the coefficients.

Consider the non-zero polynomial

$$a(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0 \in \mathbb{F}_q[x]$$

of degree d. Here, $a_i \in \mathbb{F}_q$, and $a_d \neq 0$. Each a_i can be written as

$$a_i = a_{i,0} + a_{i,1}\theta + a_{i,2}\theta^2 + a_{i,3}\theta^3 + a_{i,4}\theta^4$$

with $a_{i,j} \in \mathbb{F}_p$. We store the coefficients $a_{i,j}$ in a two-dimensional array.

Each a_i is of data type `GFqel` which is a renaming of `polyp`. This data type has been defined as an array of ten 32-bit unsigned integers. The array index DEGIDX $= 9$ stores the degree of the polynomial (or an upper bound of the degree). The two-dimensional array for $a(x)$ stores the coefficients $a_i \in \mathbb{F}_q$ in its rows. We again consider polynomials $a(x)$ with $\deg a(x) \leq 8$. For the purpose of Jacobian arithmetic arithmetic over $\mathbb{F}_q$, we do not require polynomials of larger degrees. A polynomial of degree eight has nine coefficients, so nine rows suffice to store $a(x)$. In addition, we need to store the degree of $a(x)$. This can be done externally. However, we prefer to add a row to the two-dimensional array, and store the degree of $a(x)$ at the zeroth index of the extra row. The following figure demonstrates this storage format.

		$j \rightarrow$									
		0	1	2	3	4	5	6	7	8	DEGIDX
$i \downarrow$	0	$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	?	?	?	?	4
	1	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	?	?	?	?	4
	2	$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	?	?	?	?	4
	$\vdots$										
	d	$a_{d,0}$	$a_{d,1}$	$a_{d,2}$	$a_{d,3}$	$a_{d,4}$	?	?	?	?	4
	$d+1$	?	?	?	?	?	?	?	?	?	?
	$\vdots$										
	8	?	?	?	?	?	?	?	?	?	?
	DEGIDX	d	?	?	?	?	?	?	?	?	?

Here, cells marked by "?" can be left uninitialized. Each row is not necessarily in the compact format, since not all a_i are needed to be polynomials of degree four in θ.

As a specific example, take

$$\begin{aligned} a(x) \;=\; & (920890\theta^2 + 1048263)x^3 + \\ & (472979\theta^4 + 903930\theta^3 + 108403\theta^2 + 801667\theta + 426057)x^2 + \\ & (989173\theta^3 + 805031\theta^2 + 919240\theta + 1022862). \end{aligned}$$

The compact representation of this polynomial is given below.

$i\downarrow$ \ $j\rightarrow$	0	1	2	3	4	5	6	7	8	DEGIDX
0	1022862	919240	805031	989173	?	?	?	?	?	3
1	?	?	?	?	?	?	?	?	?	MINUSINF
2	426057	801667	108403	903930	472979	?	?	?	?	4
3	1048263	0	920890	?	?	?	?	?	?	2
4	?	?	?	?	?	?	?	?	?	?
5	?	?	?	?	?	?	?	?	?	?
6	?	?	?	?	?	?	?	?	?	?
7	?	?	?	?	?	?	?	?	?	?
8	?	?	?	?	?	?	?	?	?	?
DEGIDX	3	?	?	?	?	?	?	?	?	?

Let us now consider the zero polynomial. Formally, this polynomial has degree $-\infty$. In the compact representation, we simply store MINUSINF at the (DEGIDX, 0)-th cell of the two-dimensional array.

	0	1	2	3	4	5	6	7	8	DEGIDX
0	?	?	?	?	?	?	?	?	?	?
1	?	?	?	?	?	?	?	?	?	?
2	?	?	?	?	?	?	?	?	?	?
3	?	?	?	?	?	?	?	?	?	?
4	?	?	?	?	?	?	?	?	?	?
5	?	?	?	?	?	?	?	?	?	?
6	?	?	?	?	?	?	?	?	?	?
7	?	?	?	?	?	?	?	?	?	?
8	?	?	?	?	?	?	?	?	?	?
DEGIDX	MINUSINF	?	?	?	?	?	?	?	?	?

A non-compact representation of the zero polynomial is now given. Here, the zero polynomial is treated as a polynomial in x having degree bound three. Each row, on the other hand, is treated as a polynomial in θ of formal degree four.

	0	1	2	3	4	5	6	7	8	DEGIDX
0	0	0	0	0	0	?	?	?	?	4
1	0	0	0	0	0	?	?	?	?	4
2	0	0	0	0	0	?	?	?	?	4
3	0	0	0	0	0	?	?	?	?	4
4	?	?	?	?	?	?	?	?	?	?
5	?	?	?	?	?	?	?	?	?	?
6	?	?	?	?	?	?	?	?	?	?
7	?	?	?	?	?	?	?	?	?	?
8	?	?	?	?	?	?	?	?	?	?
DEGIDX	3	?	?	?	?	?	?	?	?	?

3.5.6 Jacobian Arithmetic over Extension Fields

A reduced divisor in the Mumford representation is a pair of univariate polynomials $u(x)$ and $v(x)$ with coefficients from $\mathbb{F}_q$. We define a data type `Mumq` for this representation. This

is an array of two polynomials over $\mathbb{F}_q$. Therefore, it is an array of two elements of the `polyq` data type.

The `polyq` at index zero stores $u(x)$, whereas the `polyq` at index one stores $v(x)$.

0	1
$u(x)$	$v(x)$

Here, `Mumq` actually uses $2 \times 10 \times 10 = 200$ single-precision unsigned integers. As an example, consider a reduced divisor corresponding to two distinct and non-opposite finite rational points (x_1, y_1) and (x_2, y_2) on C. In this case, $u(x) = u_2x^2 + u_1x + u_0$ and $v(x) = v_1x + v_0$. We have $u_2 = 1$. Suppose that v_1 is non-zero. When the `Mumq` array is viewed as an array of two `polyq` arrays, we have the following (compact) representation. The empty cells are not used in the storage, and can be left uninitialized.

Index	$D[0] = u(x)$	$D[1] = v(x)$
0	$u_0 = x_1x_2$	$v_0 = \dfrac{x_1y_2 - x_2y_1}{x_1 - x_2}$
1	$u_1 = -(x_1 + x_2)$	$v_1 = \dfrac{y_1 - y_2}{x_1 - x_2}$
2	$u_2 = 1$	
3		
4		
5		
6		
7		
8		
9	2	1

Now, each coefficient of $u(x)$ and $v(x)$ is an array of ten unsigned integers. We can write $u_i = u_{i4}t^2 + u_{i3}t^3 + u_{i2}t^2 + u_{i1}t + u_{i0}$ for $i = 0, 1, 2$, and $v_i = v_{i4}t^2 + v_{i3}t^3 + v_{i2}t^2 + v_{i1}t + v_{i0}$ for $i = 0, 1$. Here, u_{ij} and v_{ij} are elements of $\mathbb{F}_p$ (and so single-precision unsigned integers). Therefore the `Mumq` data type has the following representation shown at the level of 32-bit words. The empty cells are not used, and may contain any values. We assume that all u_i and v_i have degree four. If not, the entry 4 at DEGIDX position of each `polyp` array is to be set appropriately. Otherwise, the representation is correct but not compact.

	$D[0] = u(x)$										$D[1] = v(x)$									
Index	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
0	u_{00}	u_{01}	u_{02}	u_{03}	u_{04}					4	v_{00}	v_{01}	v_{02}	v_{03}	v_{04}					4
1	u_{10}	u_{11}	u_{12}	u_{13}	u_{14}					4	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}					4
2	u_{20}	u_{21}	u_{22}	u_{23}	u_{24}					4										
3																				
4																				
5																				
6																				
7																				
8																				
9	2										1									

This may look somewhat wasteful of space, because at most 32 of the 200 cells contain useful values. But this representation is favored for time efficiency. First, a `Mumq` data type is an array, so passing it to a function is the same as passing a pointer. Second, the individual polynomials $u(x)$ and $v(x)$ are readily available in `Mumq` as `polyq` data types, so we do not need any conversion between `Mumq` and `polyq`.

Table 3.2 summarizes the user-defined data types used in our library.

Definition	Construction
`ui`	renamed `unsigned integer` data type
`hecui`	ui data structure array of size 10
`hecui2`	ui data structure array of size 20
`GFpel`	renamed `ui` data type
`polyp`	`GFpel` array of size 10
`GFqel`	renamed `polyp` data type
`polyq`	`GFqel` array of size 10
`Mumq`	`polyq` array of size 2

Table 3.2: Defined data structure

3.6 Performance Analysis

Since the arithmetic of hyperelliptic curves is somewhat inefficient in comparison with the elliptic-curve point arithmetic, reducing the performance gap is a point of concern. In this section, we analyze the practical performances of hyperelliptic and elliptic curves [48]. However, the entire computation depends on the underlying field operation. Therefore, the construction of an efficient finite field plays a vital role in the Jacobian arithmetic. Since we work with subfield curves, we focused on the construction of the quintic extension fields. We prefer to have only small integers (positive or negative) as the non-zero coefficients of $f(x)$. Table 3.3 lists some l-bit primes and some corresponding monic irreducible polynomials (the defining polynomials) used to represent the extension field $\mathbb{F}_q = \mathbb{F}_{p^5}$.

Prime length l	**Prime** p	**Extending polynomial** $f(x)$
20	1048571	$x^5 - 2$ or $x^5 + 2$
24	16777199	$x^5 + x - 3$ or $x^5 - 4x - 1$
28	268435399	$x^5 - x - 2$
32	4294836163	$x^5 + 2x - 1$

Table 3.3: Construction of efficient extension field

We consider two hyperelliptic-curve families: the first is over large prime fields, and the second is that of subfield curves defined over quintic extensions. This comparative study is based on point addition, point doubling, and scalar multiplication. Scalar multiplication uses the 4-bit windowed multiplication method for both elliptic and hyperelliptic curves. All the curves used in our experiments offer 128-bit security. The parameters of these curves are listed now.

1. Elliptic Curve: Curve P-256 [48]
   - ⊕ Prime $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$ of size 256 bits
   - ⊕ Curve $\mathcal{E} : y^2 \equiv x^3 - 3x + b \pmod{p}$, where $b = 24551555460089438177402939151974517847691080581611912380 65$
   - ⊕ Group order $n =$ 1157920892103562487626974469494075735299969552241357603424222590610685120 44369
2. Hyperelliptic curve: Generic-1271 [4]
   - ⊕ Prime $p = 2^{127} - 1$ of size 128 bits
   - ⊕ Curve $\mathcal{C}_1 : y^2 = x^5 + f_3x^3 + f_2x^2 + f_1x + f_0 \pmod{p}$, where

     $f_0 = 66679866221737283378235608571799928 16,$
     $f_1 = 90907655901711006083734360528442376758,$
     $f_2 = 13271361720934533507512505944425618802 1,$
     $f_3 = 34744234758245218589390329770704207149.$
   - ⊕ Group order $n =$ 2894802230932904884816923999565902513845117797309155137410147573289258 0332259
3. Subfield curve
   - ⊕ Base prime $p = 4294836163$ of size 32 bits
   - ⊕ Monic irreducible polynomial $f(x) = x^5 + 2x - 1$
   - ⊕ Curve $\mathcal{C} : y^2 = x^5 + x + a$, where $a \in \mathbb{F}_p$. As a sample, we take $a = 23$.
   - ⊕ Group order $n =$ 1157643261432762193010464109587902557945749684746506164802945703526927 70626891

We run our codes in the Linux environment on an Intel core-i7 3.10 GHz desktop machine. The codes are complied by the GNU C compiler (`gcc`) version 5.5.0. We use three mathematical libraries for three sets of implementations. As mentioned in the previous section, we ourselves have developed a mathematical library explicitly suited to the subfield curves. This library can easily handle elliptic and hyperelliptic curves over prime fields. A trendy and commonly available library for multiple-precision integer arithmetic is the GNU multiple-precision library GMP [31]. This library does not support polynomial arithmetic, so the arithmetic of subfield curves cannot be straightaway implemented using GMP. Moreover, despite its popularity, GMP is known to be not one of the fastest available libraries. The number theory library NTL [65] is a public-domain and fast library, popular among number theorists. We have used NTL version 11.3.2. NTL supports the arithmetic of both multiple-precision integers and polynomials, and is thus suitable for all of the three curves.

We first compare the performance of Cantor's algorithm for hyperelliptic curves with that of elliptic curves in Table 3.4. The table illustrates that Cantor's algorithm is significantly inefficient compared to the elliptic-curve arithmetic. For the hyperelliptic curve over prime fields, NTL is the fastest library, whereas our library is the slowest. Our library is not optimized for multiple-precision integer arithmetic which is very infrequently needed in cryptographic protocols involving subfield curves.

Curve (Library)	Doubling	Addition	Scalar multiplication
P-256 (NTL)	0.000003	0.000003	0.001375
Generic-1271 (Our library)	0.000191	0.000201	0.038537
Generic-1271 (NTL)	0.000020	0.000022	0.007514
Generic-1271 (GMP)	0.000054	0.000058	0.033367
Subfield curve $\mathcal{C}$ (Our library)	0.000034	0.000038	0.011614
Subfield curve $\mathcal{C}$ (NTL)	0.000100	0.000102	0.034476

Table 3.4: Comparison of Cantor's algorithm with elliptic-curve arithmetic (all times are in milliseconds)

Coordinate	Curve (Library)	Doubling	Addition	Scalar multiplication
Affine	Generic-1271 (NTL)	0.000007	0.000009	0.002439
Affine	Subfield curve $\mathcal{C}$ (Our library)	0.000009	0.0000010	0.003021
Affine	Subfield curve $\mathcal{C}$ (NTL)	0.000028	0.000026	0.008442
Projective	Generic-1271 (NTL)	0.000007	0.000007	0.002466
Projective	Subfield curve $\mathcal{C}$ (Our library)	0.000011	0.000012	0.003167
Projective	Subfield curve $\mathcal{C}$ (NTL)	0.000026	0.000028	0.008604
Weighted	Generic-1271 (NTL)	0.000007	0.000009	0.002576
Weighted	Subfield curve $\mathcal{C}$ (Our library)	0.000008	0.000012	0.002944
Weighted	Subfield curve $\mathcal{C}$ (NTL)	0.000025	0.000031	0.008507

Table 3.5: Comparison with different coordinates (all times are in milliseconds)

Table 3.5 illustrates the performance of the subfield curve and generic-1271 in different coordinates systems. For affine coordinates, Lange's explicit formula is implemented. NTL being the most efficient multiple-precision integer library, we report the timings of P-256 and Generic-1271 for this library only. For subfield curves, algorithms are implemented using both our mathematical library and NTL. The first inference we draw from these figures is that the performance gap between elliptic and hyperelliptic curves is now significantly reduced. Second, there is a stiff competition between hyperelliptic curves over prime fields and hyperelliptic curves over extension fields. This, in turn, boosts interest in furthering work on our proposed family of subfield curves.

3.7 Conclusion

Our experiments reported in this chapter have been able to narrow the gap between the performances of elliptic and hyperelliptic curves. We have also established our proposed family of subfield curves to be nearly as efficient and practical as curves over prime fields. Possibilities of further performance enhancements of our family of curves are worth investigating.

CHAPTER

FOUR

CRYPTOGRAPHIC PRIMITIVES

4.1 Introduction

The most common cryptographic primitives that can be built on the arithmetic of hyperelliptic curves are listed now. These are well-known techniques, and can be found in many references like [9].

- Diffie–Hellman key exchange.
- ElGamal encryption.
- HECDSA signatures.
- Challenge-response authentication schemes based on encryption and signatures.
- Schnorr's zero-knowledge identification scheme.

All these schemes are more or less straightforward to implement using the hyperelliptic-curve library. Moreover, the performances of these implementations depend solely on the efficiency of scalar multiplication, justifying the practical study reported in the last chapter.

Only an implementation of the ElGamal scheme introduces some problem. Let G be the subgroup of $\mathbb{J}_q$ of prime order n, that we use in our cryptographic protocols. Encoding messages to the members of this group can be done. For example, if γ is a generator of G, then an integer k modulo n can be mapped to $k\gamma$. But then, decoding the message becomes very problematic, because retrieving k from $k\gamma$ is equivalent to solving the discrete logarithm problem in G. In this chapter, we explain how we can get around this difficulty by mapping messages to divisors in $\mathbb{J}_q$. Since the encoded messages now belong to a group larger than G, this encoding scheme raises some security concerns. We deal with these issues, and establish that this leads to no information leakage.

4.2 ElGamal encryption

In 1985, Taher ElGamal proposes this scheme [12]. A few years later, Tsiounis and Yung give a concrete proof for the security of ElGamal encryption [67]. They establish that the DDH assumption directly implies the security of ElGamal encryption, and conversely. At present, proving the IND-CCA1 security of ElGamal encryption is a major open problem. Lipmaa shows that ElGamal encryption is IND-CCA1 secure based on some non-standard assumptions [47]. Wu and Stinson also show that ElGamal encryption is OW-CCA1 secure under the delay-target discrete logarithm assumption (DTDLA), and the strong generalized knowledge of exponent assumption (SGKEA) [71]. However, ElGamal encryption is not IND-CCA2 secure because it is a homomorphic encryption. The bit-security analysis of the hyperelliptic-curve Diffie–Hellman problem is provided by Zhang [72].

In our context, the ElGamal scheme raises an issue. The scheme needs a mapping of messages (bit strings) to the elements of the group, in which we are working. Fouque, Joux and Tibouchi propose an injective encoding for elliptic curves. This construction uses the existence of a covering curve of genus two, for which a bijective encoding is known [16]. Later, Fouque and Tibouchi propose a "nearly bijection" encoding map. However, they use a curve defined over prime fields [17]. No such map exists for subfield curves. In our case of subfield hyperelliptic curves, the group G in which the ElGamal scheme works is a proper subgroup of the Jacobian $\mathbb{J}_q$ over the quintic extension. This is because the Jacobian $\mathbb{J}_q$ is the internal direct sum of G with the Jacobian $\mathbb{J}_p$ over the ground field. An efficient and reversible encoding of messages to elements of G is not straightforward.

We work around this difficulty by mapping messages to the strictly larger group $\mathbb{J}_q$, that is, each encoded message now has a component in a cryptographically small group $\mathbb{J}_p$ which plays no role in the security of the ElGamal scheme. The question that our encoding scheme raises is whether this component has any potential of leaking important cryptographic secrets.

In what follows, we elaborate our adaptation of the ElGamal scheme for subfield curves along with our message encoding and decoding techniques. We denote our encoding scheme by the function θ.

— Public parameters

1. Field sizes p and $q = p^5$, the element $a \in \mathbb{F}_p$ defining the curve $y^2 = x^5 + x + a$, the size n of the group G, a base point $P \in G$, the message length l, and the padding length l'.

— Key pair of the recipient

2. (x, Y), where $x \in_U \mathbb{Z}_n$ (private key), and $Y = xP \in G$ (public key).

— Encoding

3. Let the message be $m \in \{0,1\}^l$.

4. Break m into two $\frac{l}{2}$-bit chunks: $m = m_0 \,||\, m_1$. For each $b \in \{0,1\}$, generate $r_b \in_U \{0,1\}^{l'}$ such that $x_b^5 + x_b + a$ is a square in $\mathbb{F}_q$, where $x_b = 0 \,||\, b \,||\, m_b \,||\, r_b$. Let y_b be a square root of $x_b^5 + x_b + a$ in $\mathbb{F}_q$.

5. Take the divisor $(u_2, u_1, u_0, v_1, v_0)$ with the two rational points (x_0, y_0) and (x_1, y_1) as the message representative M. Notice that $M \in \mathbb{J}_q$ (we do not, in general, have $M \in G$).

— Encryption

6. Generate $k \in_U \mathbb{Z}_n$, and set $R = kP \in G$.
7. Compute $S = M + kY \in \mathbb{J}_q$.
8. Send (R, S) to the recipient.

— Decryption

9. Recover $M = S - xR \in \mathbb{J}_q$.

— Decoding

10. Let $M = (u_2, u_1, u_0, v_1, v_0)$. We have $x_0 + x_1 = -u_1$, and $x_0 x_1 = u_0$. Solve these equations (quadratic) to obtain x_0, x_1. Notice that x_b has second msb b.
11. Recover m_0, m_1 from x_0, x_1 after removing the paddings. Output $m = m_0 \,||\, m_1$.

The encoding map θ used in our variant of ElGamal encryption has some desirable properties.

- The encoding map is efficiently computable in polynomial time. The inverse of the map is also efficiently computable.
- This map can be applied for all forms of subfield hyperelliptic curves.
- Our encoding scheme is a probabilistic map due to the concatenated pseudorandom bits.
- This map does not preserve arithmetic operation. Let $D_1 = \theta(k_1)$ and $D_2 = \theta(k_2)$. Then, any correlation between k_1 and k_2 does not reflect on D_1 and D_2.
- This is a well-distributed map.

Now we prove that our point-encoding scheme is well-distributed. To that effect, we use character sums. Similar types of results can be found in [15, 17]. Using this result, we obtain a bound on statistical distance. Later, we prove that our encoding map terminates in polynomial time.

Theorem 4.2.1 *Let χ be any character of the Abelian group $GF(q)$. The character sum is defined as*

$$T(\chi) = \sum_{u \in \mathbb{F}_q} \chi(\theta(u)).$$

Then, for a non trivial character, we have $T(\chi) \leq 2\sqrt{q} + 11$.

Proof Let χ be a non-trivial character of $\mathbb{J}_q$. Then, χ is an unramified, non-trivial Artin character of $\mathcal{C}$. Applying Riemann's hypothesis for the L-function, we have [15]

$$\left| \sum_{P \in \mathcal{C}_q} \chi(P) \right| \leq 2\sqrt{q}$$

Now,

$$\sum_{u \in \mathbb{F}_q} \chi(\theta(u)) = |R|\chi((0,0)) + \sum_{P \in \mathcal{C}_p - Wei} \chi(P)$$

$$= |R|\chi((0,0)) - \sum_{P \in Wei} \chi(P) + \sum_{P \in \mathcal{C}_q} \chi(P)$$

Here, R denotes the set of all zeros of the curve polynomial $f(x)$. Therefore, $|R| = 2g+1 = 5$. The set of all Weierstrass points is denoted by Wei. The cardinality of this set is $2g+2 = 6$. It therefore follows that θ is a $(2 + \frac{11}{\sqrt{q}})$ well-distributed map.

Since θ is $(2 + \frac{11}{\sqrt{q}})$ well-distributed to the curve $\mathcal{C}$, for all $D \in \mathbb{J}_q$, we have

$$\left| \frac{N(D)}{q^2} - \frac{1}{|\mathbb{J}_q|} \right| < (2\sqrt{q} + 11)^2,$$

where $N(D)$ be the number of preimages of D under θ.

The statistical distance between the distribution defined by the point-encoding map on $\mathbb{J}_q$ and the uniform distribution is

$$\sum_{D \in \mathbb{J}_q} \left| \frac{N(D)}{q^2} - \frac{1}{|\mathbb{J}_q|} \right| \leq \left(2 + \frac{11}{\sqrt{q}} \right)^2.$$

Therefore, the bound on the statistical distance is $\frac{c}{\sqrt{q}} + O(\frac{1}{q})$ (where c is a positive constant). The proof of this statement can be found in [17].

Now, we prove that our encoding map is efficiently computable and is expected to terminate on any input m in polynomial time. To establish the statement, we prove the following theorem. Similar results for elliptic curves can be found in [16].

Theorem 4.2.2 *For large enough q, the expected number of iterations in θ on any input message m is less than three.*

Proof Let $x \in \mathrm{GF}(q)$ be a random element. Suppose that $m \in \{0,1\}^k$ is the message, and the k least significant bits of x coincide with m. $\Pr(m)$ denotes the probability that the k least significant bits of a random element $x \in \mathrm{GF}(q)$ coincide with the message m. For each x, we have at most two y, that is, at most two points.

$$\Pr(m) \geq \frac{1}{2} \frac{|\{(x,y) \in \mathcal{C}_q \mid \ lsb_k(x) = m\}|}{|\{(x,y) \in \mathbb{F}_q \mid \ lsb_k(x) = m\}|} \tag{4.1}$$

Here, "$lsb_k(x) = m$" means that the k least significant bits of x coincide with m.

Now, we obtain tight bounds on the numerator and the denominator. We apply the one-dimensional Erdös–Turán–Koksma inequality to obtain a tight bound for the numerator. For any interval $I \in \mathbb{R} - \mathbb{Z}$ of length l and any positive integer K with $|K| \leq |q|$, we have

$$\left| \frac{|(x,y) \in \mathcal{C}_q - \mathcal{O} \mid \frac{x}{q} \in I \mid}{N} - l \right| \leq \frac{3}{K+1} + \frac{3}{N} \sum_{j=1}^{K} \frac{T(\chi_j)}{j}, \tag{4.2}$$

where N is the cardinality of set of rational points, and χ_j denotes the additive character $x \mapsto e^{2i\pi jx/q}$. Clearly, the length $l \geq 2^{-k}(1 - \frac{2}{q})$. Putting $K = \sqrt{q} - 1$, and using Theorem 4.2.1, we get

$$\left| \{(x,y) \in \mathcal{C}_q - \mathcal{O} \mid \frac{x}{q} \in I\} \right| \geq lN - \frac{3N}{\sqrt{q}} - 3\Big((2g-2)\sqrt{q} + 4g + 3\Big) \log \sqrt{q}$$

$$\geq (l - \frac{3}{\sqrt{q}})\Big((q+1) - 2g\sqrt{q}\Big) - 3(2g-2)\sqrt{q} \log \sqrt{q} - (12g+9) \log \sqrt{q}$$

$$\geq l(q+1) - 6\sqrt{q} \log \sqrt{q}.$$

For genus-two curves, the Hasse–Weil bound states that $|N - (q+1)| \leq 4\sqrt{q}$.

A bound for the denominator is

$$|\{x \in \text{ GF}(q) \mid lsb_k(x) = m\}| \leq 2^{-l}.q$$

From the inequality (4.2), we then have

$$\Pr(m) \geq \frac{1}{2} \frac{2^{-k}(1 - \frac{2}{q})(1+q) - 6\sqrt{q} \log \sqrt{q}}{2^{-k}.q}$$

$$\geq \frac{1}{2} \frac{1}{q} \left(1 - \frac{2}{q}\right)(1+q) - \frac{3 \log \sqrt{q}.2^k}{\sqrt{q}}$$

$$\geq \frac{1}{2} - \frac{1}{q} - \frac{3 \log q}{q^\epsilon}$$

Therefore, the expected number of iterations is at most 3, when q is large enough.

Given a random padding, the function θ can be computed in deterministic polynomial time. Moreover, the image of the encoding map covers almost all reduced divisors of $\mathbb{J}_q$.

4.3 Formal Security

Now we define the Diffie–Hellman problems. Let P be a generator of an additive cyclic group G of order n. Let $a, b \in Z_n$. A triple $[aP, bP, abP]$ is called a *Diffie–Hellman triple*. Two important problems associated with these triples go as follows.

DDH Problem: Let $[aP, bP, X]$ be a given triple. The decisional Diffie–Hellman problem deals with deciding whether the given triple is a Diffie–Hellman triple, that is, whether $X = abP$. The DDH assumption is that it is computationally infeasible to solve the DDH problem. Let $\mathcal{O}$ be an oracle that, given a triple, identifies whether the triplet is a Diffie–Hellman triple. Under the DDH assumption, no such oracle having polynomial running time can exist.

CDH Problem: Let aP, bP be supplied as inputs. The computational Diffie–Hellman problem deals with the determination of the group element abP. The CDH assumption is that it is computationally infeasible to solve the CDH problem. In other words, an oracle $\mathcal{O}$ that, given aP, bP, returns abP in polynomial time cannot exist.

The security of the original ElGamal encryption scheme is equivalent to the DH problems. Here, we establish that our adaptation of the ElGamal scheme is also secure under the DH assumptions.

First, we note that the divisor $kY = kxP \in G$ masks the encoded message M. Therefore the knowledge of M is equivalent to the knowledge of the mask. The sender can calculate it from k and the recipient's public key $Y = xP$, whereas the recipient can compute the mask from $R = kP$ and his private key x. To a passive eavesdropper, the challenge is to compute the mask kxP from the knowledge of kP and xP alone. Consequently, ElGamal decryption is as difficult as solving the CDH to the eavesdropper.

What remains is to justify that our encoding of messages to divisors in $\mathbb{J}_q$ (not in G) does not lead to some information leakage. Since $\mathbb{J}_q$ is the internal direct sum of G and $\mathbb{J}_p$, the encoded message can be uniquely decomposed as $M = M_G + M_p$, where $M_G \in G$, and $M_p \in \mathbb{J}_p$. The n-th multiple of any element of G is zero, and therefore $nS = nM = nM_p$. The size e of the small group $\mathbb{J}_p$ can be easily computed, and is coprime to n. If $\eta \equiv n^{-1} \pmod{e}$, then $M_p = \eta n M_p = \eta n S$, that is, M_p can be determined by any passive eavesdropper. The relevant concern is therefore whether M_p reveals some partial information about m.

This is where the random padding strings r_0, r_1 play a crucial role. If the padding length l' is somewhat larger than the bit size of p, then for each message m, we expect each element of $\mathbb{J}_p$ to appear as M_p. Moreover, the different elements of $\mathbb{J}_p$ are expected to appear as M_p with nearly equal probabilities. So we expect that the padding destroys all correlations between m and M_p. What the eavesdropper sees in S is a random element of the group $\mathbb{J}_p$. Moreover, M_p has nothing to do with the key pair (in particular, the private key x) of the recipient, because it is generated independently before the application of any key-related quantity.

4.4 Conclusion

A new encoding scheme for ElGamal encryption is proposed. This encoding is well distributed which implies that our variant of ElGamal encryption is equivalent to the original ElGamal scheme in terms of formal security. In particular, our scheme is IND-CPA secure under the DDH assumption. Moreover, proving the IND-CCA1 security of our scheme is an open problem (like original ElGamal encryption). One can however furnish IND-CCA1 security proofs based on non-standard assumptions as in Lipmaa's work [47].

CHAPTER

FIVE

CONCLUSION AND FUTURE SCOPE

5.1 Work reported in the Dissertation

In this dissertation, a new family of hyperelliptic curves is proposed. To the best of our knowledge, we are the first to report an explicitly constructed family of cryptographically suitable subfield hyperelliptic curves. The main technical novelties of the work include the following.

- We use subfield curves over quintic extensions.
- Since the base field is a single-precision prime, point counting in the extension field is very efficient, and many curves in the family can be generated with little effort.
- We have developed a library which is optimized for the Jacobian arithmetic of our family of curves.

In this work, we initially focus on constructing cryptographically suitable hyperelliptic curves. An efficient point-counting algorithm plays a vital role in choosing suitable curves. Point-counting algorithms over large prime fields are more complicated and practically more inefficient for hyperelliptic curves than for elliptic curves. So we come up with the idea of using subfield curves. Point-counting algorithms are efficient for curves defined over small prime fields. If we treat these curves as those over the quintic extensions, the point-counting problem reduces to lifting the curve size over the ground field to the curve size over the extension. This lifting can be done very efficiently using the concept of L-functions of the curve. We nevertheless require the cofactor to be a prime. This requirement decreases the yield of cryptographically suitable curves, but the same search pertains to the case of curves over large prime fields. To sum up, members of our family can be generated much faster than other families in use in the literature.

Our experiments reported in this work have been able to narrow the gap between the performances of elliptic and hyperelliptic curves. We have also established our proposed family of subfield curves to be nearly as efficient and practical as curves over prime fields. We have investigated the security issues for our family of curves. All existing algorithms

for solving the discrete logarithm problem have been found to be inefficient for our family. However, like all other curves, our family of curves is not quantum secure.

Various public-key cryptographic primitives can be implemented using our curves, like encryption, signature, and authentication schemes. In order to adopt ElGamal encryption to the subfield curves, one "almost bijective" point-encoding map is required. No such map exists for our family of curves in the literature. We have designed an encoding scheme which differs from the existing schemes in that the messages are encoded to a strictly larger group than the group where we use the Diffie–Hellman assumptions. We have justified that ciphertexts produced by this scheme are not expected to leak any information about the plaintext messages, in the statistical sense.

5.2 Future Scopes

Hyperelliptic-curve cryptography is an active area of contemporary research. This area is less explored compared to elliptic-curve cryptography. Although this work opens a new direction of research in this area, we envisage some extensions of our study.

- **Enhance the performance:** Our experimental observations suggest that the subfield hyperelliptic curves are a viable substitute of elliptic curves. To improve the performance of the subfield family of curves, more attention may be put on
  1. more efficient point-counting algorithms,
  2. optimized quintic extension fields,
  3. dedicated addition and scalar multiplication formulas.
- **CCA for ElGamal encryption:** Our adaptation of ElGamal encryption is secure in the sense of indistinguishability under the DDH assumption. But the CCA1 security of ElGamal encryption requires non-standard assumptions. Our scheme uses sufficiently long random paddings, and therefore the CCA1 security of our scheme differs subtly from that of original ElGamal encryption. It remains open whether our adaptation of ElGamal encryption can be proved to be CCA1 secure under standard assumptions only.
- **Post-quantum cryptography:** Elliptic and hyperelliptic curves are not quantum safe. Isogeny-based cryptography is a popular part of curve-based cryptography relevant for the post-quantum era. It is interesting to investigate whether our family of subfield hyperelliptic curves can play any role in isogeny-based cryptography.

PUBLICATION

- Anindya Ganguly, Abhijit Das, Dipanwita Roy Chowdhury, and Deval Mehta, *A Family of Subfield Hyperelliptic Curve for Use in Cryptography*, 22^{nd} International Conference on Information and Communications Security (ICICS 2020), Copenhagen, Denmark, 2020.

BIBLIOGRAPHY

[1] L. M. ADLEMAN, J. DEMARRAIS, AND M.-D. HUANG, *A subexponential algorithm for discrete logarithms over the rational subgroup of the jacobians of large genus hyperelliptic curves over finite fields*, in International Algorithmic Number Theory Symposium, Springer, 1994, pp. 28–40.

[2] L. M. ADLEMAN AND M.-D. A. HUANG, *Counting rational points on curves and abelian varieties over finite fields*, in International Algorithmic Number Theory Symposium, Springer, 1996, pp. 1–16.

[3] R. M. AVANZI, *Aspects of hyperelliptic curves over large prime fields in software implementations*, in International Workshop on Cryptographic Hardware and Embedded Systems, Springer, 2004, pp. 148–162.

[4] J. W. BOS, C. COSTELLO, H. HISIL, AND K. LAUTER, *Fast cryptography in genus 2*, in Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2013, pp. 194–210.

[5] J. BUHLER AND N. KOBLITZ, *Lattice basis reduction, jacobi sums and hyperelliptic cryptosystems*, Bulletin of the Australian Mathematical Society, 58 (1998), pp. 147–154.

[6] D. G. CANTOR, *Computing in the Jacobian of a hyperelliptic curve*, Mathematics of Computing, 48 (1987), pp. 99–101.

[7] H. COHEN, G. FREY, R. AVANZI, C. DOCHE, T. LANGE, K. NGUYEN, AND F. VERCAUTEREN, *Handbook of elliptic and hyperelliptic curve cryptography*, Chapman and Hall/CRC, 2005.

[8] C. COSTELLO AND K. LAUTER, *Group law computations on jacobians of hyperelliptic curves*, in Selected Areas in Cryptography, Springer, 2012, pp. 92–117.

[9] A. DAS AND C. VENI MADHAVAN, *Public-Key Cryptography: Theory and Practice: Theory and Practice*, Pearson Education India, 2004.

[10] C. DIEM, *The GHS attack in odd characteristic*, J. Ramanujan Mathematical Society, 18 (2003), pp. 1–32.

[11] W. DIFFIE AND M. HELLMAN, *New directions in cryptography*, IEEE transactions on Information Theory, 22 (1976), pp. 644–654.

[12] T. ELGAMAL, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory, 31 (1985), pp. 469–472.

[13] N. D. ELKIES ET AL., *Elliptic and modular curves over finite fields and related computational issues*, AMS IP STUDIES IN ADVANCED MATHEMATICS, 7 (1998), pp. 21–76.

[14] A. ENGE AND P. GAUDRY, *An l (1/3+ ε) algorithm for the discrete logarithm problem for low degree curves*, in Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2007, pp. 379–393.

[15] R. R. FARASHAHI, P.-A. FOUQUE, I. SHPARLINSKI, M. TIBOUCHI, AND J. VOLOCH, *Indifferentiable deterministic hashing to elliptic and hyperelliptic curves*, Mathematics of Computation, 82 (2013), pp. 491–512.

[16] P.-A. FOUQUE, A. JOUX, AND M. TIBOUCHI, *Injective encodings to elliptic curves*, in Australian Conference on Information Security and Privacy, Springer, 2013, pp. 203–218.

[17] P.-A. FOUQUE AND M. TIBOUCHI, *Deterministic encoding and hashing to odd hyperelliptic curves*, in International Conference on Pairing-Based Cryptography, Springer, 2010, pp. 265–277.

[18] G. FREY AND H.-G. RÜCK, *A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves*, Mathematics of computation, 62 (1994), pp. 865–874.

[19] W. FULTON, *A note on weakly complete algebras*, Bulletin of the American Mathematical Society, 75 (1969), pp. 591–593.

[20] E. FURUKAWA, M. KAWAZOE, AND T. TAKAHASHI, *Counting points for hyperelliptic curves of type y 2= x 5+ ax over finite prime fields*, in International Workshop on Selected Areas in Cryptography, Springer, 2003, pp. 26–41.

[21] S. D. GALBRAITH, *Weil descent of jacobians*, Electronic Notes in Discrete Mathematics, 6 (2001), pp. 459–468.

[22] ——, *Mathematics of public key cryptography*, Cambridge University Press, 2012.

[23] R. GALLANT, R. LAMBERT, AND S. VANSTONE, *Improving the parallelized pollard lambda search on anomalous binary curves*, Mathematics of Computation, 69 (2000), pp. 1699–1705.

[24] P. GAUDRY, *An algorithm for solving the discrete log problem on hyperelliptic curves*, in International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2000, pp. 19–34.

[25] ——, *Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem*, Journal of Symbolic Computation, 44 (2009), pp. 1690–1702.

[26] P. GAUDRY AND N. GÜREL, *An extension of kedlaya's point-counting algorithm to superelliptic curves*, in International Conference on the Theory and Application of Cryptology and Information Security, Springer, 2001, pp. 480–494.

[27] P. GAUDRY AND R. HARLEY, *Counting points on hyperelliptic curves over finite fields*, in International Algorithmic Number Theory Symposium, Springer, 2000, pp. 313–332.

[28] P. GAUDRY, F. HESS, AND N. SMART, *Constructive and destructive facets of Weil descent*, Journal of Cryptology, 15 (2002).

[29] P. GAUDRY AND E. SCHOST, *Genus 2 point counting over prime fields*, Journal of Symbolic Computation, 47 (2012), pp. 368–400.

[30] P. GAUDRY AND E. THOMÉ, *The mpFq library and implementing curve-based key exchanges*, in SPEED: Software Performance Enhancement for Encryption and Decryption, Amsterdam, Netherlands, Jun 2007, ECRYPT Network of Excellence in Cryptology, pp. 49–64.

[31] T. GRANLUND, *The GNU multiple precision arithmetic library*, `https://gmplib.org/`, (1996).

[32] R. HARLEY, *Fast arithmetic on genus 2 curves*, available at `http://cristal.inria.fr/~harley/hyper`, (2000).

[33] R. HARTSHORNE, *Algebraic geometry*, vol. 52, Springer Science & Business Media, 2013.

[34] F. HESS, *The ghs attack revisited*, in International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2003, pp. 374–387.

[35] H. HISIL AND C. COSTELLO, *Jacobian coordinates on genus 2 curves*, Journal of Cryptology, 30 (2017), pp. 572–600.

[36] M.-D. HUANG AND D. IERARDI, *Counting rational points on curves over finite fields*, in Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science, IEEE, 1993, pp. 616–625.

[37] Y. HUANG, Z. SU, F. ZHANG, Y. DING, AND R. CHENG, *Quantum algorithm for solving hyperelliptic curve discrete logarithm problem*, Quantum Information Processing, 19 (2020), p. 62.

[38] A. JOUX AND V. VITSE, *Cover and decomposition index calculus on elliptic curves made practical*, in Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2012, pp. 9–26.

[39] K. S. KEDLAYA AND A. V. SUTHERLAND, *Computing L-series of hyperelliptic curves*, in International Algorithmic Number Theory Symposium, Springer, 2008, pp. 312–326.

[40] N. KOBLITZ, *Elliptic curve cryptosystems*, Mathematics of Computation, 48 (1987), pp. 203–209.

[41] N. KOBLITZ, *Hyperelliptic cryptosystems*, Journal of Cryptology, 1 (1989), pp. 139–150.

[42] T. LANGE, *Efficient arithmetic on genus 2 hyperelliptic curves over finite fields via explicit formulae*, IACR Cryptology ePrint Archive, 121 (2002).

[43] ——, *Efficient arithmetic on hyperelliptic curves*, Cryptology ePrint Archive, 107 (2002).

[44] ——, *Inversion-free arithmetic on genus 2 hyperelliptic curves.*, IACR Cryptology ePrint Archive, 147 (2002).

[45] ——, *Formulae for arithmetic on genus 2 hyperelliptic curves*, Applicable Algebra in Engineering, Communication and Computing, 15 (2004), pp. 295–328.

[46] R. LIDL AND H. NIEDERREITER, *Finite fields*, vol. 20, Cambridge university press, 1997.

[47] H. LIPMAA, *On the CCA1-security of ElGamal and Damgard's ElGamal*, in International Conference on Information Security and Cryptology, Springer, 2010, pp. 18–35.

[48] G. LOCKE AND P. GALLAGHER, *Fips pub 186-3: Digital signature standard (dss)*, Federal Information Processing Standards Publication, 3 (2009), pp. 186–3.

[49] K. MATSUO, J. CHAO, AND S. TSUJII, *Fast genus two hyperelliptic curve cryptosystems*, in ISEC2001-31, IEICE, 2001.

[50] A. MENEZES, Y. WU, AND R. ZUCCHERATO, *An elementary introduction to hyperelliptic curves*, Research report, Faculty of Mathematics, University of Waterloo, 1996.

[51] A. J. MENEZES, P. C. VAN OORSCHOT, AND S. A. VANSTONE, *Handbook of Applied Cryptography*, CRC press, 1996.

[52] P. L. MONTGOMERY, *Speeding the pollard and elliptic curve methods of factorization*, Mathematics of computation, 48 (1987), pp. 243–264.

[53] D. MUMFORD, *Tata lectures on theta II, volume 43 of*, Progress in mathematics, (1984), pp. 243–265.

[54] K.-I. NAGAO, *Decomposition attack for the jacobian of a hyperelliptic curve over an extension field*, in International Algorithmic Number Theory Symposium, Springer, 2010, pp. 285–300.

[55] J. PELZL, T. WOLLINGER, J. GUAJARDO, AND C. PAAR, *Hyperelliptic curve cryptosystems: Closing the performance gap to elliptic curves*, in International Workshop on Cryptographic Hardware and Embedded Systems, Springer, 2003, pp. 351–365.

[56] J. PILA, *Frobenius maps of abelian varieties and finding roots of unity in finite fields*, Mathematics of Computation, 55 (1990), pp. 745–763.

[57] J. M. POLLARD, *Monte carlo methods for index computation*, Mathematics of computation, 32 (1978), pp. 918–924.

[58] J. PROOS AND C. ZALKA, *Shor's discrete logarithm quantum algorithm for elliptic curves*, arXiv preprint quant-ph/0301141, (2003).

[59] R. L. RIVEST, A. SHAMIR, AND L. M. ADLEMAN, *Cryptographic communications system and method*, Sept. 20 1983. US Patent 4,405,829.

[60] H.-G. RÜCK, *On the discrete logarithm in the divisor class group of curves*, Mathematics of Computation of the American Mathematical Society, 68 (1999), pp. 805–806.

[61] S. SADANANDAN, *Counting in the Jacobian of hyperelliptic curves*, PhD thesis, Technische Universität München, 2010.

[62] T. SATOH, *Generating genus two hyperelliptic curves over large characteristic finite fields*, in Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2009, pp. 536–553.

[63] R. SCHOOF, *Elliptic curves over finite fields and computation of square roots mod p*, Mathematics of Computation, 44 (1985), pp. 483–494.

[64] P. W. SHOR, *Algorithms for quantum computation: discrete logarithms and factoring*, in Proceedings 35th annual symposium on foundations of computer science, IEEE, 1994, pp. 124–134.

[65] V. SHOUP, *NTL: A library for doing number theory*, http://www. shoup. net/ntl/, (2001).

[66] N. THÉRIAULT, *Index calculus attack for hyperelliptic curves of small genus*, in International Conference on the Theory and Application of Cryptology and Information Security, Springer, 2003, pp. 75–92.

[67] Y. TSIOUNIS AND M. YUNG, *On the security of ElGamal based encryption*, in International Workshop on Public Key Cryptography, Springer, 1998, pp. 117–134.

[68] M. S. VICTOR, *Use of elliptic curves in cryptography*, in CRYPTO, Springer, 1986, pp. 417–426.

[69] D. WIEDEMANN, *Solving sparse linear equations over finite fields*, IEEE transactions on information theory, 32 (1986), pp. 54–62.

[70] T. WOLLINGER, *Software and hardware implementation of hyperelliptic curve cryptosystems*, PhD dissertation, Ruhr University, Bochum (2004).

[71] J. WU AND D. R. STINSON, *On the security of the ElGamal encryption scheme and Damgard's variant.*, IACR Cryptology ePrint Archive, 2008 (2008), p. 200.

[72] F. ZHANG, *Bit security of the hyperelliptic curves Diffie-Hellman problem*, in International Conference on Provable Security, Springer, 2017, pp. 219–235.

APPENDIX

A

A DATABASE OF SUBFIELD HYPERELLIPTIC CURVES

This appendix lists some curves of our family at various security levels. These curves are of the special form $y^2 = x^5 + x + a$, $a \in \mathbb{F}_p$, where p is a single-precision prime. The curves are naturally defined over the quintic extension $\mathbb{F}_q = \mathbb{F}_{p^5}$. We represent $\mathbb{F}_q$ as $\mathbb{F}_p[t]/\langle f(t)\rangle$, where $f(t) \in \mathbb{F}_p[t]$ is a monic irreducible polynomial of degree 5. The Jacobian of a curve over $\mathbb{F}_p$ and $\mathbb{F}_q$ are denoted by $\mathbb{J}_p$ and $\mathbb{J}_q$, and their sizes by $n_p = |\mathbb{J}_p|$ and $n_q = |\mathbb{J}_q|$. We have $\mathbb{J}_q = \mathbb{J}_p \oplus G$. For all the curves listed here, G is a group of prime order $n = |G| = n_q/n_p$. At all security levels, it is recommended to use the curves with $n = 2^{\cdots} - \cdots$. The curves with $n = 2^{\cdots} + \cdots$ should work well, but would be slightly (and unnecessarily) inefficient.

- **Security level 80**

  $p = 2^{20} - 5 = 1048571$
  Group size: $n \approx 2^{160}$
  $f(t) = t^5 + 2$ or $t^5 - 2$

		Curve 1: $y^2 = x^5 + x + 47$
n_p	=	1099928953312
n_q	=	1606861421126112580388908685296656425664857224973157020278432
n	=	1460877465119621059080883122151454896336021166011
	=	$2^{160} - 624172211281859122801710564828123319911376965$
		Curve 2: $y^2 = x^5 + x + 52$
n_p	=	1101226502688
n_q	=	1606861421126113461086479300845938085559612360640338474575648
n	=	1459156147444600848921990361604654440813312450921
	=	$2^{160} - 234548988630206928169447111162857842620092055$
		Curve 3: $y^2 = x^5 + x + 60$
n_p	=	1098401972048
n_q	=	1606861421126117326279311266898329713697223055120690303050128
n	=	1462908354152060576672027642006156546558828957461
	=	$2^{160} + 1406716821157658468342809289873526902896414485$

	Curve 4: $y^2 = x^5 + x + 125$	
n_p	=	1099905028412
n_q	=	1606861421126112664264329356881888234950517280688763783588852
n	=	1460909241815210664385164137239675033477318704371
	=	$2^{160} - 592395515692253818520695476607986178613838605$
	Curve 5: $y^2 = x^5 + x + 128$	
n_p	=	1098862311784
n_q	=	1606861421126115007639651529082548438333685298910613718483384
n	=	1462295506811385516979046962482432086750728993651
	=	$2^{160} + 7938694804825987753621297661490670947964506 75$
	Curve 6: $y^2 = x^5 + x + 135$	
n_p	=	1100360173512
n_q	=	1606861421126111913860894023889193743946630991364868722113752
n	=	1460304961781305559328768597218998239924941278771
	=	$2^{160} - 119667554959735887491623549728477973099126 4205$
	Curve 7: $y^2 = x^5 + x + 148$	
n_p	=	1098893705344
n_q	=	1606861421126119834450206209018830850253025931056419779327104
n	=	1462253731468144638088689800544741093830940632041
	=	$2^{160} + 7520941372417198850049678284580741750080890 65$
	Curve 8: $y^2 = x^5 + x + 343$	
n_p	=	1101309318912
n_q	=	1606861421126115639306162861717792372727421177077000367346432
n	=	1459046421865891542892103065453400964536213519461
	=	$2^{160} - 245521546501137531158176726288205511971902 3515$
	Curve 9: $y^2 = x^5 + x + 360$	
n_p	=	1099970144812
n_q	=	1606861421126117231030132592235323832858772868149439974394412
n	=	1460822758422004180498434689942452533171210700801
	=	$2^{160} - 678878908898737705250142773830486484721842175$
	Curve 10: $y^2 = x^5 + x + 385$	
n_p	=	1099461685888
n_q	=	1606861421126115742738304958065295704462515816163847011118208
n	=	1461498333003123454030625297220885364941452791141
	=	$2^{160} - 330432777946417305953549539765471447975183 5$
	Curve 11: $y^2 = x^5 + x + 436$	
n_p	=	1099799414512
n_q	=	1606861421126115338867183146480213815242192344440909236166352
n	=	1461049533145194036170711943988005525451510665571
	=	$2^{160} - 452104185708882032972888728277494204421877405$
	Curve 12: $y^2 = x^5 + x + 488$	
n_p	=	1099481205214
n_q	=	1606861421126115557527156151050826324869375615982353266621454
n	=	1461472386709293922647242752643794739359463213161
	=	$2^{160} - 292506216089955564420800724882802964693298 15$
	Curve 13: $y^2 = x^5 + x + 523$	
n_p	=	1100056630044
n_q	=	1606861421126114123550377839447918438496106934464574655532204
n	=	1460707910157173431610936437387807239751869334141
	=	$2^{160} - 793727173729486592748395328475779904063208835$
	Curve 14: $y^2 = x^5 + x + 577$	
n_p	=	1098568824086
n_q	=	1606861421126114707650818530840847414520145566111950667084106
n	=	1462686165760265282564978119696084963927833309071
	=	$2^{160} + 1184528429362364361293286979801944271900766095$

	Curve 15: $y^2 = x^5 + x + 646$	
n_p	=	1098710175364
n_q	=	1606861421126116376993080869870404215677603366997459169801284
n	=	146249798823767797478945353996293256056456919128 1
	=	$2^{160} + 9963509067750565857687072466495408 0863664830 5$
	Curve 16: $y^2 = x^5 + x + 673$	
n_p	=	1100032022354
n_q	=	1606861421126111820155195562644361578706177659012396993250154
n	=	1460740586158145178663909994246819699165815270701
	=	$2^{160} - 7610511727577395397748384694633204901172722 75$
	Curve 17: $y^2 = x^5 + x + 693$	
n_p	=	1098025206412
n_q	=	1606861421126110621794121470203316417504777617093838423172252
n	=	1463410322224593420706581649157701374345139259821
	=	$2^{160} + 1908684893690502502896816441418354689206716845$
	Curve 18: $y^2 = x^5 + x + 718$	
n_p	=	1101093783528
n_q	=	1606861421126112025660284680033077115489920655383651518334008
n	=	1459332025268173470668564375610569699463592243411
	=	$2^{160} - 2169612062729447535120457105713320192340299565$
	Curve 19: $y^2 = x^5 + x + 732$	
n_p	=	1098917189692
n_q	=	1606861421126118535083575286515181795493240549966445319349012
n	=	1462222482457011214539591232159518675832500447211
	=	$2^{160} + 7208451261082963359063994432356561765679042 35$
	Curve 20: $y^2 = x^5 + x + 755$	
n_p	=	1099900264232
n_q	=	1606861421126112297242981868150444155527852891237473834616552
n	=	1460915569693126180642662882606005417563259748761
	=	$2^{160} - 5860676377767375610219501102776020926727942 15$
	Curve 21: $y^2 = x^5 + x + 769$	
n_p	=	1100170674704
n_q	=	1606861421126110825226757400366076592663106907954646078212944
n	=	1460556491890165629824887331044561101987468439061
	=	$2^{160} - 9451454407372883787975016717219176684641039 15$
	Curve 22: $y^2 = x^5 + x + 840$	
n_p	=	1098848394466
n_q	=	1606861421126117268086730979417750166196268372103142166068706
n	=	1462314027320386593162214538399878838337661376641
	=	$2^{160} + 8123899894836749585297056835958186817288336 65$
	Curve 23: $y^2 = x^5 + x + 842$	
n_p	=	1098537038976
n_q	=	1606861421126120277776367891050984864166029916766247057410176
n	=	1462728487174136839885420718352524262412685476201
	=	$2^{160} + 1226849843233921681735885636241242756752933225$
	Curve 24: $y^2 = x^5 + x + 894$	
n_p	=	1099382490234
n_q	=	1606861421126114232400408871710249326594175550969338476155994
n	=	1461603614210827558909979749564061244231918974641
	=	$2^{160} + 1019768799246407062949168477782245759864316 65$
	Curve 25: $y^2 = x^5 + x + 925$	
n_p	=	1100277362644
n_q	=	1606861421126111558685451957622613824541615138891652010177884
n	=	1460414869633211977910042144308469997955806765211
	=	$2^{160} - 1086767697690940293642688407813021700125777765$

- **Security level 96**

$p = 2^{24} - 17 = 16777199$
Group size: $n \approx 2^{192}$
$f(t) = t^5 + t - 3$ or $t^5 - 4t - 1$

	Curve 1: $y^2 = x^5 + x + 8$	
n_p	=	281405073717438
n_q	=	1766829161770434166957255033723286569895571241304785288893503449734184998
n	=	627859739140498654643103856135846963395600726806604281262 1
	=	2^{192} + 14956560183057825952491381508032178536518236020082997 25
	Curve 2: $y^2 = x^5 + x + 36$	
n_p	=	281393693383592
n_q	=	1766829161770434168216884428148876763294620088602430264488795231689851752
n	=	627885131512850586364816146359169184062897895892740910748 1
	=	2^{192} + 17495797418250998123720403840254245266235144633745945 85
	Curve 3: $y^2 = x^5 + x + 182$	
n_p	=	281541581675196
n_q	=	1766829161770434163587554797062952451467097562619621151021438808266205996
n	=	627555315721980610048989743094671215077676680825256610730 1
	=	2^{192} − 15485781668746633458919922609542653255886362114684055 95
	Curve 4: $y^2 = x^5 + x + 268$	
n_p	=	281498912176744
n_q	=	1766829161770434162429223601115712019598725263744696446011536199745203224
n	=	627650440318966372485258371346872901500027191123739898267 1
	=	2^{192} − 59733219701703898320570973893740110208353322663553022 5
	Curve 5: $y^2 = x^5 + x + 341$	
n_p	=	281373604787448
n_q	=	1766829161770434166541188443349370421721554172247390232834849780934101528
n	=	627929959210321754750916149191583807284002131722939130646 1
	=	2^{192} + 21978567165367836733720687081716567376658727653567935 65
	Curve 6: $y^2 = x^5 + x + 484$	
n_p	=	281461060324438
n_q	=	1766829161770434164183411502907892843489623402464302812164433756773126598
n	=	627734848910831137733258439468985097848078000810469308932 1
	=	2^{192} + 24675372163061349679497148218456237842456364065857642 5
	Curve 7: $y^2 = x^5 + x + 497$	
n_p	=	281472323762384
n_q	=	1766829161770434162610011233530716659599331873478697370943768486197875504
n	=	627709729380702066173745826282417926878865026392854628243 1
	=	2^{192} − 44415796601020983311603834871473137051805354882304 65
	Curve 8: $y^2 = x^5 + x + 577$	
n_p	=	281658747071408
n_q	=	1766829161770434162932093310996743258492031272008478468936729247001408048
n	=	627294263054609085358907051916855580863335672946662555308 1
	=	2^{192} − 41591048405899102467189040391106074689987149974089598 15
	Curve 9: $y^2 = x^5 + x + 639$	
n_p	=	281345919749546
n_q	=	1766829161770434162218234826812058807817981753245158656257112288959615166
n	=	627991748856093103073353346365710651828453282413191931197 1
	=	2^{192} + 28157531742502668977440404494401021821773796678847990 75

Curve 10: $y^2 = x^5 + x + 710$

$$
\begin{aligned}
n_p &= 281367351409064 \\
n_q &= 176682916177043416513731406081750493437573745245613090396973554131626882 4 \\
n &= 627943914929114028219885061369509892757658831832958963359 1 \\
&= 2^{192} + 233741390445951836306119048743251147423287386555512069 5
\end{aligned}
$$

Curve 11: $y^2 = x^5 + x + 742$

$$
\begin{aligned}
n_p &= 281581471305832 \\
n_q &= 176682916177043415861272677279965581225427882960346923694325429586549815 2 \\
n &= 627466414454323620590354738744781295519384262997413618076 1 \\
&= 2^{192} - 243759084344455793224203575985346090851281448989833213 5
\end{aligned}
$$

Curve 12: $y^2 = x^5 + x + 745$

$$
\begin{aligned}
n_p &= 281409926501296 \\
n_q &= 176682916177043416428464794881903375654643084111774595151334995830393377 6 \\
n &= 627848911990067003690168932420800432276953552139688517163 1 \\
&= 2^{192} + 138738451398927306589990100033790666718007693285065873 5
\end{aligned}
$$

Curve 13: $y^2 = x^5 + x + 761$

$$
\begin{aligned}
n_p &= 281466285920208 \\
n_q &= 176682916177043416138866560091241672577974099265755988374780400425933828 8 \\
n &= 627723194624917547973678030764783637386531313594346766451 1 \\
&= 2^{192} + 130210862494715900990884440169957762957691479433151615
\end{aligned}
$$

Curve 14: $y^2 = x^5 + x + 790$

$$
\begin{aligned}
n_p &= 281393486225372 \\
n_q &= 176682916177043416509580018158375424739029085387239363459490320291210685 2 \\
n &= 627885593753707519989000196555418743563030639284066519659 1 \\
&= 2^{192} + 175420215039443605421254234652101952795094837663068369 5
\end{aligned}
$$

Curve 15: $y^2 = x^5 + x + 799$

$$
\begin{aligned}
n_p &= 281453652678516 \\
n_q &= 176682916177043416441715677978732239966638839708626180231404935244546199 6 \\
n &= 627751370414280741305497668621799172263183682339640918303 1 \\
&= 2^{192} + 411968756126649219187263010325306529481378932374670135
\end{aligned}
$$

Curve 16: $y^2 = x^5 + x + 802$

$$
\begin{aligned}
n_p &= 281374584594426 \\
n_q &= 176682916177043416523685083821752919680900239655151561473071012132436876 6 \\
n &= 627927772622799585745828476338868907238230694642836122609 1 \\
&= 2^{192} + 217599084131509362249534018102265627995150196432671319 5
\end{aligned}
$$

Curve 17: $y^2 = x^5 + x + 805$

$$
\begin{aligned}
n_p &= 281479888558072 \\
n_q &= 176682916177043416191690956440299979680852545507142432497553655260461255 2 \\
n &= 627692859628911059652209693781290730984446265962669901159 1 \\
&= 2^{192} - 173139097570167313692485394759106257892784837335501305
\end{aligned}
$$

Curve 18: $y^2 = x^5 + x + 814$

$$
\begin{aligned}
n_p &= 281465240556912 \\
n_q &= 176682916177043416291512582659585607488103521044602437004516791920316587 2 \\
n &= 627725525991968095812732138733455422921059078395575892558 1 \\
&= 2^{192} + 153524533000194291531964126887813108235339491724412685
\end{aligned}
$$

Curve 19: $y^2 = x^5 + x + 966$

$$
\begin{aligned}
n_p &= 281479745017032 \\
n_q &= 176682916177043416259828555869736425119642574483049349438011150722665727 2 \\
n &= 627693179721875000358571380912787303256708510710001851407 1 \\
&= 2^{192} - 169938167930760250075614079793383535270337364015998825
\end{aligned}
$$

Curve 20: $y^2 = x^5 + x + 967$

$$
\begin{aligned}
n_p &= 281469398660424 \\
n_q &= 176682916177043416086354058750051782456468311919530868407706150233050802 4 \\
n &= 627716252700709359575710538385500529097527364203285276615 1 \\
&= 2^{192} + 607916204128319213159606473388748729181975688182532 5 5
\end{aligned}
$$

- **Security level 112**

  $p = 2^{28} - 57 = 268435399$
  Group size: $n \approx 2^{224}$
  $f(t) = t^5 - t - 2$

Curve 1: $y^2 = x^5 + x + 10$

$$
\begin{aligned}
n_p &= 72066946475789318 \\
n_q &= 19426647671364916683012083899672713893728562795562865259575337640774679869469\\
&\quad 00119638 \\
n &= 269563907191367442197648740228967085362950573849443117237688443332 41 \\
&= 2^{224} - 3555948013895574902141064122922137342087037596260757334765915975
\end{aligned}
$$

Curve 2: $y^2 = x^5 + x + 167$

$$
\begin{aligned}
n_p &= 72063113090196194 \\
n_q &= 19426647671364916683008093512655001061618780782383162257441236025589016464341\\
&\quad 68946274 \\
n &= 269578246599615879132633353667741932871155841730032020022864850723 21 \\
&= 2^{224} - 212200718905188140367972024543738652156024953737047881712517 6895
\end{aligned}
$$

Curve 3: $y^2 = x^5 + x + 170$

$$
\begin{aligned}
n_p &= 72057584612233888 \\
n_q &= 19426647671364916683004842922884775150704843518700003588105132774556121025477\\
&\quad 52490848 \\
n &= 269598929466013124938935831965866745736681408669268186138493972546 71 \\
&= 2^{224} - 5372054932730077343189043295609996900355561375386725421299454 5
\end{aligned}
$$

Curve 4: $y^2 = x^5 + x + 192$

$$
\begin{aligned}
n_p &= 72054687534662708 \\
n_q &= 19426647671364916683057076572620489735254797925338736417082413901527540789004\\
&\quad 27906948 \\
n &= 269609769135693107843388176485153941485034412460589773677242428797 81 \\
&= 2^{224} + 1030246418670989671802561495763474866296823518404886620632630565
\end{aligned}
$$

Curve 5: $y^2 = x^5 + x + 194$

$$
\begin{aligned}
n_p &= 72065702524247044 \\
n_q &= 19426647671364916682996549532824745531414917182413839450485699563953234175285\\
&\quad 99211924 \\
n &= 269568560229169707214097046490838232304493485883171186745360165460 21 \\
&= 2^{224} - 3090644233669073257310437935807443187795834223453806567593703195
\end{aligned}
$$

Curve 6: $y^2 = x^5 + x + 303$

$$
\begin{aligned}
n_p &= 72058964178983856 \\
n_q &= 19426647671364916682983943236039483872887227198337825527367475066754118858720\\
&\quad 08604496 \\
n &= 269593768002437344677038236177557391782503347616858091909200958981 91 \\
&= 2^{224} - 5698669069053269631914692638914953868096608547632901835143510 25
\end{aligned}
$$

Curve 7: $y^2 = x^5 + x + 331$

$$
\begin{aligned}
n_p &= 72062728784482448 \\
n_q &= 19426647671364916682947230511629856466870985351919765711860236226058976099970\\
&\quad 84833648 \\
n &= 269579684242378196637509441380169250182192843360976813755379864331 51 \\
&= 2^{224} - 1978242912820130916070949002705655417860086442891105565623816065
\end{aligned}
$$

Curve 8: $y^2 = x^5 + x + 368$

$$
\begin{aligned}
n_p &= 72058605715813268 \\
n_q &= 19426647671364916682982750896674997329332552560320620752679753540373101122264\\
&\quad 46748388 \\
n &= 269595109125206636781991089703567698288644460839729300184136217183 41 \\
&= 2^{224} - 4357546299761164679061166628608447726983385676424626899885308 75
\end{aligned}
$$

Curve 9: $y^2 = x^5 + x + 381$

$$
\begin{aligned}
n_p &= 72055042711573672 \\
n_q &= 19426647671364916682999842802040689696347925770268035243651088542525472558692\\
&\quad 96329352 \\
n &= 269608440163266422269687597896293105601453475374534057370090965329 41 \\
&= 2^{224} + 897349176002432301744702609679886508203114912833255905486283725
\end{aligned}
$$

Curve 10: $y^2 = x^5 + x + 421$

$$
\begin{aligned}
n_p &= 72059845979911296 \\
n_q &= 19426647671364916682968292360184943618158928468376626770093951170935926063673\\
&\quad 04954496 \\
n &= 269590468966317799447612427637153057247487257162837303635232159167 01 \\
&= 2^{224} - 899770518859849905772323304324948888418706256842117580394332515
\end{aligned}
$$

Curve 11: $y^2 = x^5 + x + 502$

$$
\begin{aligned}
n_p &= 72058998505993204 \\
n_q &= 19426647671364916682979738906092320154005768688245563951856695406406348302865\\
&\quad 29377004 \\
n &= 269593639575065520844060827827181242665044737694771696968914144609 51 \\
&= 2^{224} - 582709644087710260932304301506407132670653063402784212195788265
\end{aligned}
$$

Curve 12: $y^2 = x^5 + x + 622$

$$
\begin{aligned}
n_p &= 72061885460545938 \\
n_q &= 19426647671364916682942144012250468019704059963641103452800691347794326042277\\
&\quad 06086578 \\
n &= 269582839072411652757005585839796508485386681657213586020846747312 81 \\
&= 2^{224} - 166275990947451896645650303997982509847625681921387901893551793 5
\end{aligned}
$$

Curve 13: $y^2 = x^5 + x + 623$

$$
\begin{aligned}
n_p &= 72067207730517258 \\
n_q &= 19426647671364916683006686780486296165066947136977676544776343787538203620876\\
&\quad 85938478 \\
n &= 269562929980685170927611538396183421493291899083029720203098175460 91 \\
&= 2^{224} - 365366908212270190586124740128852430795451423760046079379270312 5
\end{aligned}
$$

Curve 14: $y^2 = x^5 + x + 668$

$$
\begin{aligned}
n_p &= 72061111483533016 \\
n_q &= 19426647671364916682973413249329669991187997988520173271197341795366648884352\\
&\quad 72542616 \\
n &= 269585734544271922038590624309040656196858433545857004254812699506 01 \\
&= 2^{224} - 137321272344759080795265611556505395130106795487205562234029861 5
\end{aligned}
$$

Curve 15: $y^2 = x^5 + x + 837$

$$
\begin{aligned}
n_p &= 72058372429019606 \\
n_q &= 19426647671364916682995817265498855824656315640168777832851967334671387466423\\
&\quad 78507166 \\
n &= 269595981931189823934596244970244471224558762883655487757323550652 61 \\
&= 2^{224} - 348474031657401207390589995183551181268134175023705371255183955
\end{aligned}
$$

Curve 16: $y^2 = x^5 + x + 844$

$$
\begin{aligned}
n_p &= 72059654226424944 \\
n_q &= 19426647671364916682986354146768295734382393685837493269582663441551785357172\\
&\quad 70359824 \\
n &= 269591186356830792016394889253568104188469096656739603736507136752 71 \\
&= 2^{224} - 828031467560593027526161662820254790234756866612107452896573945
\end{aligned}
$$

Curve 17: $y^2 = x^5 + x + 902$

$$
\begin{aligned}
n_p &= 72060405252059136 \\
n_q &= 19426647671364916682950110973077744496285041627777005599416450772790840205036\\
&\quad 01852416 \\
n &= 269588376632253224130252578591418242442018781648579707556497830659 81 \\
&= 2^{224} - 110900392531738164175722787780642943526625768260172545382718323 5
\end{aligned}
$$

Curve 18: $y^2 = x^5 + x + 911$

$$
\begin{aligned}
n_p &= 72065341732055666 \\
n_q &= 19426647671364916682997718485832680028658983543628851652699976857391091785675\\
&\quad 17826266 \\
n &= 26956990981316159762796056940425752195755266143100626359004078694101 \\
&= 2^{224} - 29556858344800318709581465938784778818782794399461220995315551 15
\end{aligned}
$$

Curve 19: $y^2 = x^5 + x + 921$

$$
\begin{aligned}
n_p &= 72056858044997272 \\
n_q &= 19426647671364916682999190570814031790685002832871294270300764171375470724042\\
&\quad 00528552 \\
n &= 26960164789912957349366841098005472322992065344855636209753047431991 \\
&= 2^{224} + 21812276231755469982601098584164935492092231506372864943718 2775
\end{aligned}
$$

Curve 20: $y^2 = x^5 + x + 992$

$$
\begin{aligned}
n_p &= 72058657767456704 \\
n_q &= 19426647671364916682980833392391258236466985868268622324365643020920312501730\\
&\quad 69362624 \\
n &= 26959491438291019322647634927190765837192795005589393164783338197481 \\
&= 2^{224} - 45522885962047201938015982886483644434941695117931632027205 1735
\end{aligned}
$$

- **Security level 128**

  $p = 2^{32} - 2^{17} - 61 = 4294836163$
  Group size: $n \approx 2^{256}$
  $f(t) = t^5 + 2t - 1$

Curve 1: $y^2 = x^5 + x + 23$

$$
\begin{aligned}
n_p &= 18445535354239713704 \\
n_q &= 21353349706355382679157775197589488265763737459782284278839102711572096041688632540250840844361426 4 \\
n &= 11576432614327621930104641095879025579457496847465061648029457035269277062689 1 \\
&= 2^{256} - 27763094039976122524574049897652058695016190899475591630136552203590130 45
\end{aligned}
$$

Curve 2: $y^2 = x^5 + x + 43$

$$
\begin{aligned}
n_p &= 18445935166209787132 \\
n_q &= 21353349706355382679157747854926939171991413663566895389734461618524760141066020055674233935464572 \\
n &= 11576181697456872262420761759260202136459339672153810084193464529106107341392 1 \\
&= 2^{256} - 30272262747472799363367416085886488676587944102463197522938716852056226015
\end{aligned}
$$

Curve 3: $y^2 = x^5 + x + 64$

$$
\begin{aligned}
n_p &= 18445136678282565974 \\
n_q &= 21353349706355382679157792729998647197378310853587689275903740743929213110670786831380314590392 14 \\
n &= 11576682829082512160080896513327878606880411023862946423245757677258339118026 1 \\
&= 2^{256} - 25260946491073822762019875409121784465874427011099807000007235329738459675
\end{aligned}
$$

Curve 4: $y^2 = x^5 + x + 67$

$$
\begin{aligned}
n_p &= 18445849105501231246 \\
n_q &= 21353349706355382679157749475859589925545244956578464359538602235504446087917044020144905907 26 \\
n &= 11576235707136424378990376310742881180866621430917799735289376946800459923938 1 \\
&= 2^{256} - 29732165951951633667221901259096044603770356462566686563814539908530400555
\end{aligned}
$$

Curve 5: $y^2 = x^5 + x + 144$

$$
\begin{aligned}
n_p &= 18445751928370191294 \\
n_q &= 21353349706355382679157735831932620478000702876379969718237133756292965468907787038588 5 \\
n &= 11576296693827486377044452211644247757350479468014318249189168293130831376963 1 \\
&= 2^{256} - 29122299041331653126462892245430279765189985497381547565901076604815870305
\end{aligned}
$$

Curve 6: $y^2 = x^5 + x + 155$

$$
\begin{aligned}
n_p &= 18445736313288916512 \\
n_q &= 21353349706355382679157750528092293986922955482322272606981646178787072859654529364241075018278 72 \\
n &= 11576306493643046186246932619465423544684636702043143030691277649401270377428 1 \\
&= 2^{256} - 29024300885733561101658814033672406423617645209133732544807513900425865655
\end{aligned}
$$

Curve 7: $y^2 = x^5 + x + 212$

$$
\begin{aligned}
n_p &= 18445853801712721228 \\
n_q &= 21353349706355382679157708473188026677483719763350023908981244123734817700625626026766488880658 68 \\
n &= 11576232759891383677964642923296806516305945207503179852282949365392957053888 1 \\
&= 2^{256} - 29761638402358643924555775719842690210532590608765516628090353983559101055
\end{aligned}
$$

Curve 8: $y^2 = x^5 + x + 269$

$$
\begin{aligned}
n_p &= 18445638320520338292 \\
n_q &= 21353349706355382679157781463870893008372063365219089615280743470464965687969610487543262849961 32 \\
n &= 11576367992968984943349624617615980104035195104755325183244311152735197130102 1 \\
&= 2^{256} - 28409307626345990074738832528106812918033618087312207014472480561158338915
\end{aligned}
$$

Curve 9: $y^2 = x^5 + x + 363$

$$
\begin{aligned}
n_p &= 18445472663011544842 \\
n_q &= 21353349706355382679157775170943570719217922846679304681807786487098495049290767015727502636798 42 \\
n &= 11576471959525853775183770975654401805301198887003350612621643900819763471750 1 \\
&= 2^{256} - 27369642057657671733275252143889800257995795607057913241144999715494922435
\end{aligned}
$$

Curve 10: $y^2 = x^5 + x + 412$

$$
\begin{aligned}
n_p &= 18444841786540635846 \\
n_q &= 21353349706355382679157777775633984800472760421902467183486829023088441463203433794297640086796 46 \\
n &= 11576867914333161036563960566220649405514953025089867699041838253552849261530 1 \\
&= 2^{256} - 23410093984585057931379346481413798120454414741887049039201472384637024635
\end{aligned}
$$

Curve 11: $y^2 = x^5 + x + 417$

$$
\begin{aligned}
n_p &= 18445562932204446376 \\
n_q &= 21353349706355382679157793989576915263260948451527014167718887086444746679597836680968896327293 36 \\
n &= 11576415306400965450085691800918462966780372766680368292980649055780319779421 1 \\
&= 2^{256} - 27936173306540922714066999503278185466256998836881109651093450109931845725
\end{aligned}
$$

Curve 12: $y^2 = x^5 + x + 503$

$$
\begin{aligned}
n_p &= 18445576946900352966 \\
n_q &= 21353349706355382679157782993467744671522701595428909870584092975047751379583362233850295979634 66 \\
n &= 11576406510799685281579460889991769951018474700915974281397533977110521412175 1 \\
&= 2^{256} - 28024129319342607776376108770208343085237656480821225482244236807915518185
\end{aligned}
$$

Curve 13: $y^2 = x^5 + x + 620$

$$
\begin{aligned}
n_p &= 18445292073373061752 \\
n_q &= 21353349706355382679157797181492245179134570686757856600941603181168162065484820186654587213304 72 \\
n &= 11576585299606225736425042746547797379930579840663315380816973335740439342561 1 \\
&= 2^{256} - 26236241253938059320557543209934053964186259007410231287850650508736214325
\end{aligned}
$$