TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Effiziente Algorithmen

Counting in the Jacobian of Hyperelliptic Curves

In the light of genus 2 curves for cryptography

Sandeep Sadanandan

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: UNIV.-PROF. DR. GEORG CARLE

Prüfer der Dissertation:

1. UNIV.-PROF. DR. ERNST W. MAYR
2. UNIV.-PROF. DR. HANS-JOACHIM BUNGARTZ

Die Dissertation wurde am 06.05.2010 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 24.09.2010 angenommen.

Document Classification according to ACM CCS (1998)

Categories and subject descriptors:

Abstract

With the drastic increase in the number and the use of handheld devices, – mobile phones and smart cards – light weight cryptography has come to the lime light. Elliptic and Hyperelliptic curve cryptosystems (ECC, HECC) are emerging as the best solutions for light weight cryptography. Like in any traditional cryptosystem, the size of the cipher-text space is a significant factor indicating the achievable security level. In the traditional systems, the size of the group on which the system is defined, defines the cipher-text space. Unlike the traditional ones, ECC and HECC are defined on the Jacobian of curves. Gaining knowledge about the size of the Jacobian is not straightforward and the computation is inefficient. Since the generic group order counting methods are incapable of counting in large groups of cryptographic size (2^{160}), special methods were devised for counting in the Jacobian of hyperelliptic curves, which are not yet fast enough for practical applications. In this thesis, we are bringing together the advances in group order counting and the special properties of genus 2 hyperelliptic curves. The Primorial method for group order counting is applied in a special interval where the group order is predicted to be. The resulting method provides an improvement factor of $\frac{P}{\varphi(P)}$, where P is the product of the first p primes and φ is Euler's totient function. We will see that the value of p varies, depending on the expected size of the Jacobian. It will also be shown that the factor of improvement becomes larger when the size of the group gets larger.

Acknowledgment

It would not have been possible to finish this thesis, without the help from many people. On successful completion of the dissertation I take this opportunity to thank all those involved directly or indirectly in its accomplishment.

First and foremost, I thank Prof. Dr. Ernst W Mayr for accepting me as a doctoral candidate to work at the Chair for Efficient Algorithms. He has been very kind to me all through the 4 years of my doctoral studies. For any kind of academic or non-academic matters, I could approach him and there was always some solution to my problems.The discussions with him were invaluable; the discussions and his infinite energy have inspired and encouraged me innumerous times during the past few years.

I also thank Dr. Peter Ullrich, who guided me through the first few months of doctoral studies. It was he who gave me the initial momentum which helped me continue my research on the topic of this thesis.

The atmosphere in the institute was always very warm. Hanjo, who could be very well be called the "Guru" of the chair was there to answer any logistical queries. Ernst Bayer and the secretaries were more than friendly and helpful whenever I was in need. Johannes Nowak, Johannes Krugel and Stefan Schmidt were always there for any little puzzles or riddles or some random proofs/jokes. I thank all of them for being wonderful.

Many thanks to the member of GraduiertenKollege GKAAM and German Research Foundation (DFG - Deutsche Forschungsgemeinschaft) for the scholarship they provided me for the initial couple of years.

Last but not the least - family and friends. On this occasion I not only thank them, but also express my love for all of them. I thank Rajyalakshmi for being there for almost anything I needed during my doctoral studies. I also thank Suthirtha Saha who helped me initially with coming to Germany. The list would get too long, if I am to do so. I once again thank everybody who helped directly or indirectly in the making of this thesis.

Sandeep Sadanandan
München, May 2010

It is better to do the right problem the wrong way than the wrong problem the right way.

- Richard Hamming.

Contents

List of Tables

List of Figures

Preface

The world of communication is expanding day by day and the number of devices for communication are increasing exponentially. Especially, the world of handheld devices has trillions of devices currently in use where as there were practically none in the early ’90s. With increased communication does come more need of security.

As the device-to-device handshakes are common, public key cryptosystems are of supreme importance. The small devices have no power (neither computing nor battery) to use the traditional “hefty” algorithms like RSA. Cryptographers have been developing new cryptosystems based on elliptic and hyperelliptic curves for the past two decades. The developments made in this field are fascinating.

But, unlike traditional systems based on RSA, where the security level of a cryptosystem could be easily computed, cryptosystems based on the curves need more information to assess and assert the security. One of the parameters needed for asserting the security is the size of Jacobian[1] of the curves.

In this thesis we propose a method for counting the elements in the Jacobian. The method we propose makes use of a generic group order counting algorithm applied in the special conditions of hyperelliptic curves of genus 2 (which are best known for hyperelliptic curve cryptosystems).

While the best algorithms of the day are still exponential ($O(N^{\frac{1}{2}})$), our method helps to reduce the time/space requirement to a fraction $\frac{\varphi(P)}{P}$ where P is the product of primes up to p. The value of p depends on the size of the curve. For the curves which are large enough for cryptographic purposes, the reduction is up to 60%.

[1] Jacobian is a group of points or combinations-of-points on the curve in elliptic curves or hyperelliptic curves respectively

Chapter 1

Introduction

> "Basic research is what I am doing,
> when I don't know what I am doing."
>
> - Wernher Von Braun.

With the migration of communication from paper to emails/SMS and other digital media, the world has moved completely to the digital era. Even wires and cables are technologies of the past. Now all happens in the air, wireless.

Technology has made life very easy. Connectivity is the new "keyword". But there is no coin without another side.

With the old media of communication, it was easier to know when the messages were tampered with. It was easier to assure the integrity of messages, non-repudiation was an integral part of the system.

But with the advent of the digital age, a message can be copied without leaving any trace on the original one. Creating fake messages is a million times easier. Non-repudiation doesn't even exist in the horizon, unless it is specifically implemented.

With all the communication, not to mention all the financial, defense, military intelligence communications too, happening through live wire, or even wireless, it is imperative that we need methods to keep the information secure and also its integrity has to be confirmed.

Security - this has been one of the silent parts of all the communication technologies - dating back to 1940s, starting with ENIGMA. Everyone appreciated the revolutions in communication technology, without really realising the importance of the silent revolutions happening in the security area, which in fact made the communication revolutions possible.

1.1 Light weight cryptography

The very first form of security was Private Key Cryptosystem. Starting from the age of Caesar, it monopolised the message-security field up until mid 1970s.

Private key cryptosystems were good to keep unwanted people from reading the messages. And that was the only thing a private key system could do. But it was FAST.

However, with the discovery of public key cryptosystems in 1976 [DH76], all the other requirements of non-repudiation and message-signing could be satisfied. But it was not fast enough to serve all the communication needs.

The computation technology grew and secure communication developed into a proper blend of both public and private key cryptosystems. The processor power was too good to notice the large requirements of public key cryptosystems. Even when man thrived for speed, most of the communication needs were served with the above mentioned rudimentary combination.

The new era in which billions or trillions of text messages are sent everyday, millions of cars are opened with remote controls, with the large scale use of mobile phones, smart cards and remote control keys, cryptosystems are entering a new dimension of communication.

Most of the devices are getting smaller and the computing power is increasing, but the communication needs and security needs are on a rise that there is a need for better and light weight cryptosystems. So the ease and facilities of new-digital age comes with a bottleneck, through which well encrypted messages cannot easily pass.

That is the beginning of light-weight cryptography - a new branch for cryptosystems which equal the old ones in security, but which needs less resources.

1.2 Generic algorithms

Every cryptosystem has been developed to enable a channel through which the information could be sent from a sender, to the receiver, from which an eavesdropper cannot attain any information.

Some methods were developed to have channels which would inform the concerned parties, when the message was read/tampered-with by the eavesdropper.

In this section, I give a review of the main cryptosystems in existence.

1.2.1 Private key systems

Private key[1] cryptosystems have always worked by making a reversible change to the message and sending it over to the receiving end. The part of the cryptosystem at the receiving end knows how to reverse the change to get the original message back.

The main advantage of this system is the speed. Whether it be simple toggling of every bit of the message or XOR-ing the whole message with a randomly generated bit-stream, the operations are very fast with computing devices.

The main disadvantage is, both the parties - the sending and receiving ends - have to know exactly what the other party would do, otherwise the reversal of roles could not happen. This is equivalent to having a private conversation ever before the communication happened - so that they could agree upon the secret method/key they wanted to engage to encrypt/decrypt.

This also means that, every person has to have a private meeting, or a secret code/key for every other person with whom he/she has to communicate. This requires a sophisticated key-management system to keep the collection of keys.

1.2.2 Public key systems

Public key cryptosystems were introduced in 1976 [DH76] and they work based on the possession of two keys by each party in the communication. The encryption and decryption are not the mirror images of each other.

Both the sender and receiver possess two keys each - a private key and a public key. As the name indicates, the public key is available in the public domain, where as the private key is secret. The sender uses the public key of the receiver to encrypt the message and it can only be unlocked (decrypted) using the private key of the receiver. The key observation to make here is that the decryption is not the reversal of encryption.

The advantages of public key cryptosystem over private key cryptosystem are:

- Key management is easier

  Instead of n^2 keys for n participants in private key cryptosystem, a public key cryptosystem needs only $2n$ keys for the same number of participants.

- Not just encryption

  Public key cryptosystem provides the facilities for digital signature which in turn makes non-repudiation also possible.

[1] A key in this section means the secret word/bit-stream

- No preparation is required

  The two parties need not necessarily meet to start a secure conversation. The secret keys could be exchanged in public, without having the doubt of being eavesdropped.

  Example: ElGamal [EG85] cryptosystem uses diffie-hellman key exchange [DH76] which doesn't require the meeting of the parties.

How does it work?

Public key cryptosystems are based on one-way functions. A one-way function is a function for which it is easy to find out the image of an element, given any element where as given an image, calculating the pre-image is infinitely hard. For more details see 2.1

One of the often quoted examples of a one-way function is Integer Factorisation. It is easy to multiply any two numbers to find their product. But once an arbitrary number is given, it is hard to find out its factors. The often used way is trying to divide the original number with every number smaller than that. It should be clear that the larger the number is, the harder would be the problem to be solved.

Not surprisingly, one of the major public key cryptosystems, RSA uses precisely this mathematical problem or one-way function, for its security.

Another often used mathematical one-way function is called Discrete Logarithm Problem (DLP). In simple words, discrete logarithm problem is the problem of finding out the logarithm of a given element, in a group. Logarithm being the reverse of exponentiation - which could be done easily as it is just repeated multiplication. But for finding the logarithm, the only way is to try every possible number which could be the logarithm.

A formal definition of DLP is beyond the scope of this chapter. Being the main problem which is used for the cryptosystem which is the focus of this thesis, DLP is explained in depth in 2.1.

Even without going to the details of DLP, it should not be hard to understand that the larger the size of the group, the harder the problem would be.

As one could imagine, these advantages do not come for free. The main disadvantage of public key cryptosystem is that it is comparatively slower than private key cryptosystems. In other words, public key cryptosystem needs more computing power to have the equivalent security of a private key cryptosystem which has less requirements.

The requirement of computing power is represented with the number of bits of the keys needed for security. In the table 1.1, one can see that in comparison with private

Bits of Security with Private Key	Symmetric Algorithm	Public key RSA/DH (key size)
80	2TDEA	1024
112	3TDEA	2048
128	AES-128	3072
192	AES-192	7680
256	AES-256	15360

Table 1.1: NIST recommended Key Sizes

key cryptosystem of small bit-sized keys, public key cryptosystem needs much longer keys.

As the public key systems need more resources, all the communications are still based on private key systems. But the public key systems are used to set up the communication - because that is the only kind of system which can be used for key-exchange without previous agreement.

1.3 Improvement over traditional systems

Public key cryptosystem was used as the encryption method which was used to agree upon the secret key for a private key cryptosystem. One time use of public key systems to establish the connection was not very expensive (in terms of resources).

And the traditional computing devices could very well cope with this requirement. But in the new climate, with very limited computational resources (handheld devices) and limited power-supply (battery driven), public key cryptosystem has been struggling to cope up.

The only way to solve this problem is having stronger and securer cryptosystems with short key-lengths. New one-way functions were sought for, or the old ones had to be applied in new setups.

The discrete logarithm problem, which was mentioned in the previous section was found to be applicable on the groups of points of algebraic curves, called Elliptic Curves [Kob87]. Later, the method was developed to be applicable on general forms of these curves, which are called Hyperelliptic Curves [Kob89].

The newly developed cryptosystems promised better results with smaller keys in comparison with the traditional public key cryptosystem.

1.3.1 What is this thesis about?

The security level of a cryptosystem is proportional to the size of the underlying group and could be raised by having a larger group. The size of the underlying group is an indicator of the size of the cipher-text space. Larger the cipher-space, harder it would be to crack the system.

From that it directly follows that the knowledge about the security level requires the knowledge of the size of the group. In case of hyperelliptic curve cryptosystems finding out the size of the underlying group is still a hard problem.

In traditional methods, the elements in the group were just numbers in a certain range. The number of elements could be calculated easily. But in the new systems, the elements of the system are sets of points, with special properties, on the curves - which don't have any specific range and counting them is hard.

Counting the number of elements in the group on which a hyperelliptic cryptosystem is based, is the main theme of this thesis.

From the table 1.1, one can see that an 80 bit field is necessary for a secure hyperelliptic curve cryptosystem. Such a cryptosystem would have the group order $\sim 2^{160}$. The present "state of the art" counting algorithms are of the order $O(\sqrt{N})$ where N is the size of the group.

For the genus 1 hyperelliptic curves which are called elliptic curves the counting problem has been solved successfully [Atk92, BSS99]. It has also been proved that curves of genus larger than 2 are not secure for cryptography. Hence, this thesis focuses on group order counting in the setting of genus 2 hyperelliptic curves.

1.4 Bird's view

The rest of the thesis is organised in the following way.

Chapter Two: "Mathematical Background"

> This chapter lays the foundation of the mathematics required for the rest of the thesis. I have tried to make the chapter as much self contained as possible. In this chapter, the reader could find the basics of different geometries - affine and projective - followed by the basics of hyperelliptic curves.
>
> In this chapter, which is the longest one in the thesis, one could also see the operations needed for making a hyperelliptic curve useful for cryptography.

Chapter Three: "State of the art"

In this chapter, which has mainly two sections, I try to sketch the traditional methods which are used for counting in large groups. After explaining the traditional methods, the chapter continues to the details of putting these methods into the special set-up of counting in the jacobian of hyperelliptic curves.

The reader could also see the latest methods which are employed to count in the jacobian of genus 2 hyperelliptic curves.

This chapter ends with posing the main problem which the rest of the thesis addresses.

Chapter Four: "The Solution"

This chapter is the heart of the thesis. All the chapters leading up to it make the scene for the solution explained in here. The solution is gradually developed from a naive method which are used for small groups. Each step of development is explained and has been treated with analysis and specifications of the advantages over the previous method.

At the culmination of this gradual evolution of solutions, the reader could see a method which is better than the existing methods for counting in the jacobian of genus 2 curves, and also a method which decides whether the counting is worth proceeding or would be a waste or resources.

Chapter Five: Implementation, Results and Future Prospects

The final solutions proposed in chapter 4 are implemented in Sage and Magma and were tested for neither too small nor too large hyperelliptic curves. The details of the implementation and the comparison of the results are provided in this chapter.

The author's comment about the future of the project is added to this chapter as a final note.

Appendix A: Algebra Refresher

Even though the chapter on Mathematical Background tries to address the issue of giving the background necessary for reading and understanding the thesis, most of the basic definitions and proofs are omitted in the chapter. The appendix tries to cover the minimum requirements needed for following the algebraic notations in the thesis.

Appendix B: Curve Database

As a part of testing and comparing the algorithms, we generated about 150 random hyperelliptic curves. In this appendix, the reader can look them up when they are mentioned in some of the chapters.

1.5 A note on notation

Most of the mathematical notation in chapter two follows the style in Fulton [Ful69].

For the rest of the thesis, the following table 1.2 gives a list of characters and variables used.

Variables	Description
P, Q, R	Points on an elliptic curve (chapter 3)
p	Prime numbers
l, k	Positive integers
D_i	Divisors of a hyperelliptic curve
$\mathbb{F}_q$	Field over which a hyperelliptic curve is defined
m	Characteristic of the base field
C	(hyper) Elliptic curve
J	Jacobian of a curve
w	Weil Interval
$[w_l, w_u]$	Weil Interval, limits
P, pr	Primorial (chapter 4)
$g, \#g$	Size and count of giant steps

Table 1.2: Note on Notation

Chapter 2

Mathematical Background

“Young man, in mathematics you don’t understand things,
you just get used to them.”

- John von Neumann.

The initial part of this chapter is concerned about basics of cryptosystems. In the second part, the properties of algebraic curves with special focus on hyperelliptic curves are explained.

This chapter assumes that the reader has basic knowledge of algebraic structures. A very small refresher chapter for basic algebra is provided as appendix A. For a detailed study of the necessary algebra, the reader is encouraged to peruse [Her86, Her75].

2.1 Cryptographic requirements

As it was mentioned in chapter 1, hyperelliptic curve cryptography is one way of implementing public key cryptography. In public key systems, everybody has a public key and a private key. Public key is used to encrypt and the private one is for the reverse process. The public key, as the name indicates, lies in the public domain and anyone can encrypt. The private key does the secret job and that job has to be impossible (computationally hard) without the key - so hard because the security is based on it.

These two processes can be mathematically described as two functions. Encryption being the function f taking a message from the set of messages and gives its image in the set of cipher-texts. Decryption is f^{-1} which should do the reverse. From the above, we have seen that f has to be a one-way function.

In layman terms, it should be like a door which could be easily opened from one side using the handle, where as the other side has no handle and cannot be opened without the key.

2.1.1 One way functions

One way functions are mathematical functions for which computing the inverse function is exceptionally hard. To be precise, computing the inverse of any element in the range takes exponential amount of resources (time/space).

Formally, a function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ is a one way function iff the following are true.

1. For $x \in \{0,1\}^*$, computing $f(x)$ is polynomial.
2. For $y \in \{0,1\}^*$, computing the pre-image x of y such that $f(x) = y$ is hard, non-polynomial.

A very often quoted example is "Integer Factorisation". The forward function is the multiplication of any given numbers. According to computational terms, multiplication has easy (efficient) algorithms - irrespective of the size of the inputs.

At the same time, given any integer, finding out its factors is a hard problem. The only[1] way to compute the factors is a trial and error method. If the size of the input is measured in terms of the bits in its representation, the factorisation is an exponentially hard process.

Anybody who reads this thesis would be able to calculate 63×77 in their head. But factorising a small number as small as 221 might be hard. The only way to find the factors is to try trial and error for every prime number until a factor is found.

$$63 \times 77 = (70 - 7) \times (70 + 7) = 70^2 - 7^2 = 4851$$

$$221 = 13 \times 17$$

[1] There are some index calculus methods developed to solve this problem. But they are (sub)exponential algorithms

One of the mostly used public key cryptosystems, RSA, relies on the hardness of Integer Factorisation.

Another widely used one-way function is the Discrete Logarithm function. It is usually dubbed as Discrete Logarithm Problem (DLP).

2.1.2 Discrete Logarithm Problem - DLP

Discrete logarithm is the discrete analogue of the natural (common) logarithm.

Given a, b and x, positive real numbers, x is the logarithm of a to the base b iff $a = b^x = b \times b \times \times \dots x$ times

Instead of positive real numbers, let us take the set of positive integers less than p, a prime number. This will be the set $\mathbb{Z}_p = \{0, 1, \dots, p-1\}$. We can define the operation (multiplication) just as the case above, except that the result should be $modulo\ p$.

$$b^n = (b \times b \times \dots n \text{ times })\text{mod } p$$

Such a set is called a cyclic group[2]. It is a property of cyclic groups that each element a can be generated by repetitively multiplying the generatorelements.

If G is a cyclic group, then each $a \in G$ can be written as b^k if b is a generator of G; where k is an integer not larger than n. The integer k is called the discrete logarithm of a to the base b in the group G. We can define the discrete logarithm to the base b as.

$$log_b : G \to \mathbb{Z}$$

In group theory, for every prime, there is a cyclic group as defined above.

The discrete logarithm problem is the computing of the logarithm of a given element of a group. It is exponentially hard - just like the integer factorisation problem. The only way to find it is the naive brute force[3].

The easy side of the function, the exponentiation part can be defined as given below, for g a generator of $\mathbb{Z}_p^*$ for some prime p.

$$f(p, g, x) = \langle p, g, g^x (mod\ p) \rangle$$

[2] More details of groups and cyclic groups are given in the appendix A

[3] sub-exponential

Similar to the little computation in the previous section, every reader of this thesis can compute in mind that 2^7 is 11 in $\mathbb{Z}_{13}^*$.

$$2^7 = 128 \equiv 11 \text{ mod } 13$$

But, to find the $log_2\ 5$ in the same group of $\mathbb{Z}_{13}^*$, one will keep computing 2^i mod 13 for all $i < 9$.

The hardness of DLP is leveraged in many cryptographic schemes such as Diffie-Hellman key exchange protocol [DH76].

In this thesis, we will be looking into a special case of DLP which is applicable in the (hyper)elliptic curve setting. More details are provided in 2.7

2.2 Algebraic geometry

In this section we will see the basics of algebraic geometry which are very essential for the rest of the thesis. For a better understanding of the details, the reader may refer to Fulton [Ful69]. Also a very good explanation about projective space is given in the appendix of [ST92].

For the basic algebra needed for this section, the reader is encouraged to have a quick perusal of the appendix A

Note: From here onwards until the end of the chapter, k is a field and K is its algebraic closure.

2.2.1 Affine geometry

Definition 1 (Affine Space). *$A^n(k)$ means the Cartesian product of k with itself n times. $A^n(k)$ is the set of n-tuples of elements of k. $A^n(k)$ is called n-dimensional affine space over k. Its elements are called points. Simply A^n means $A^n(K)$ where k is understood and K is its closure.*

$A^1(k)$ is the affine line and A^2 is the affine plane where k is understood from the context.

The points in $A^n(k)$ are called the rational points of A^n.

Definition 2 (Zero of a polynomial). *If $F \in k[x_1, x_2, \ldots, x_n]$, a point $P = (a_1, \ldots, a_n) \in A^n$ is a zero of F if $F(P) = F(a_1, \ldots, a_n) = 0$. The set of zeros of F is called the hyper-surface generated by F and is denoted by $V(F)$.*

Definition 3 (Affine algebraic set). *If S is any set of polynomials in $k[x_1, x_2, \dots, x_n]$, then*

$$V(S) = \{P \in A^n \mid F(P) = 0 \textit{ for all } F \in S\}$$

$$V(S) = \bigcap_{F \in S} V(F)$$

A subset $X \in A^n$ is an affine algebraic set if $X = V(S)$ for some S.

Definition 4 (Affine variety). *An affine algebraic set is called an affine variety if it cannot be written as a union of two smaller affine algebraic sets.*

Or

It is an irreducible affine algebraic set. i.e, if $X = V(S)$ and the ideal generated by S is a prime ideal in $k[x_1, x_2, \dots, x_n]$, then X is an affine variety.

2.2.2 Projective geometry

Two lines intersect at exactly one point. Is it always true? What happens if they are parallel?

In our case, we need any two lines to intersect - whether they are parallel or not. So we are trying to enlarge the plane so that they will intersect at infinity. For the same, we identify each point[4] $(x, y) \in A^2$ with points $(x, y, 1) \in A^3$. Every point $(x, y, 1)$ determines a unique line which passes through the origin and the point $(x, y, 1)$. Every line through $(0, 0, 0)$ which are in the place $z = 0$ are the points at infinity.

Definition 5 (Projective Space). *Projective Space over k, written as $P^n(k)$ or simply P^n is defined to be the set of all lines through $(0, 0, \dots, 0)$ in $A^{n+1}(k)$. Any point $x = (x_1, \dots, x_{n+1}) \neq (0, \dots, 0)$ determines a unique line namely $\{(\lambda x_1, \dots, \lambda x_{n+1}) \mid \lambda \in k\}$.*

Two points (x) and (y) are defined to be equivalent iff there is a nonzero $\lambda \in k$ such that:

$$y_i = \lambda x_i \textit{ for } i = 1, \dots, n + 1$$

One other way to identify P^n is as the set of equivalence classes of points in $A^{n+1} - \{0, \dots, 0\}$.

Definition 6 (Homogeneous coordinates). *Elements of P^n[5] will be called points. The equivalence classes of points are given by:*

[4] For ease of understanding we take A^2 and A^3. In general, we can take A^n and A^{n+1} with $(n + 1)^{th}$ co-ordinate to be one.

[5] P^n means $P^n(K)$ when k is known and K is its closure

$$(x_1, \ldots, x_{n+1}) \sim (\lambda x_1, \ldots, \lambda x_{n+1});\ \lambda \neq 0, x_i \in k.$$

If a point $P \in P^n$ is determined by some $(x_1, \ldots, x_{n+1}) \in A^{n+1}$, we say that $(x_1, \ldots, x_{n+1})$ is a set of homogeneous coordinates for P. In fact, $(x_1, \ldots, x_{n+1})$ stands for an equivalence

Definition 7 (Homogeneous Polynomial)**.** *A homogeneous polynomial is a polynomial with all its terms having the same degree.*

Or formally,

$$F(\lambda x_1, \ldots, \lambda x_{n+1}) = \lambda^{deg(f)} F(x_1, \ldots, x_{n+1}) \textit{ for all } \lambda \in K.$$

Definition 8 (Projective algebraic set, Projective variety)**.** *If S is any set of homogeneous polynomials in $k[x_1, x_2, \ldots, x_{n+1}]$,*

$$V(S) = \{P \in P^n \mid F(P) = 0 \textit{ for all } F \in S\}$$

$$V(S) = \bigcap_{F \in S} V(F)$$

A subset $X \subseteq P^n$ is a projective algebraic set if $X = V(S)$ for some S.

A projective algebraic set is called a projective variety if it cannot be written as a union of two smaller projective algebraic sets.

Or

It is an irreducible projective algebraic set. i.e, if $X = V(S)$ and the ideal generated by S, represented by $I(V)$, is a prime ideal in $k[x_1, x_2, \ldots, x_{n+1}]$, then X is a projective variety.

When $V(I)$ is a projective or affine variety then the generator polynomials of $I(V)$ are irreducible. Otherwise, the union of the factors of these polynomials form the same ideal, as the roots of the polynomials are the same. Now, the ideal formed is prime ideal. Suppose, for example that the ideal is generated by only one irreducible polynomial: $I(V) = F$. Then $GH \in (F) \Rightarrow G \in (F)$ or $H \in (F)$. In other words: F divides G or H.

2.2.3 Affine and projective spaces - relation

We came up with the projective space, in the beginning of this section, to enable any two lines to intersect at exactly one point. Now we should see how it happens. From definition 5, we know that a point $(x, y, z) \in P^2$ is equivalent to $(x/z, y/z, 1) \in P^2$ which is the equivalent of $(x, y) \in A^2$. Now, we should try to get the points at infinity by putting $z = 0$ in $(x, y, z) \in P^2$. We get a point at infinity of A^2. Using homogeneous coordinates and polynomials we can find the intersection of the lines (x, y, z) in the projective plane $z = 0$. The intersection represents the point at infinity of A^2 which will be the point of intersection of the lines under consideration. This sounds really absurd. But we define that all lines in the plane $z = 0$ and passing through origin corresponds to directions in the affine space. So we can define the projective space as follows.

$$P^2 = A^2 \cup \{\text{set of directions in } A^2\}$$

So, all lines in the same direction will intersect at one of these points.

Now, how do we define it formally? We can define the set of direction in A^2 by P^1. So, $P^2 = A^2 \cup P^1$. Following represents the mapping between the two spaces.

$$\frac{\{[a, b, c] : a, b, c \text{ not all zero}\}}{\sim} \leftrightarrow A^2 \cup P^1$$

$$\begin{aligned}
[a, b, c] &\rightarrow \begin{cases} (a, b) \in A^2 & \text{if } c \neq 0 \\ [a, b] \in P^1 & \text{if } c = 0 \end{cases} && (2.1) \\
[x, y, 1] &\leftarrow (x, y) \in A^2 && (2.2) \\
[A, B, 0] &\leftarrow [A, B] \in P^1 && (2.3)
\end{aligned}$$

Definition 9 (Homogenisation and Dehomogenisation)**.** *$F \in k[x_1, \ldots, x_{n+1}]$ is called a form if it is a homogeneous polynomial and we define $F_* = F(x_1, \ldots, x_n, 1)$. This is called de-homogenisation.*

If we have a polynomial in n variables, we can replace x_i by x_i/x_{n+1}. This transformation gives F^* from F. This is called homogenisation.

Definition 10 (Algebraic Curve)**.** *An algebraic curve is always an algebraic variety of dimension equal to one. In two dimensional plane (P^2), a projective variety C is called an algebraic curve when $I(C)$, the ideal of $k[x_1, \ldots, x_{n+1}]$ which generates C, is generated by a single polynomial $\in k[x_1, \ldots, x_{n+1}]$ which is irreducible by definition.*

We denote $V(I)$, [curve generated by ideal I] by C

$$C : F(x_1, \ldots, x_{n+1}) = 0 \in k[x_1, \ldots, x_{n+1}]$$

and $I(C) = \langle F \rangle$.

From here onwards, C is an algebraic curve. Let it be $C : F(x_1, \ldots, x_n)$.

Definition 11 (Coordinate ring). *Coordinate ring of C over k is the quotient ring given by*

$$k[C] = k[x_1, \dots, x_n]/I(C)$$

Similarly, the coordinate ring of C over K is defined as

$$K[C] = K[x_1, \dots, x_n]/I(C)$$

An element of $K[C]$ is called polynomial function on C. They are polynomials modulo C.

Definition 12 (Function field and rational functions). *The function field $k(C)$ of C over k is the field of fractions of $k[C]$. Similarly, $K(C)$ the function field of C over k is the field of fractions of $K[C]$.*

$$K(C) = \left\{ \frac{G}{H} \mid G, H \in K[C], deg(G) = deg(H) \right\}$$

An element of $K[C]$ is called a rational function.

Definition 13 (Zeros and Poles). *Let $R \in K(C)^*$ and $P \in C$. If $R(P) = 0$, then R is said to have a zero at P. If R is not defined at P, then R has a pole at P. (We write $R(P) = \infty$)*

Definition 14 (Uniformising parameter). *Let $P \in C$. For all $G \in K(C)^*$, there exist $T, S \in K(C)^*, m_P \in \mathbb{Z}$ such that,*
$G = T^{m_P} S$ and $T(P) = 0$ and $S(P) \neq 0, \infty$.

The function T is called a uniformising parameter for P.

Definition 15 (Intersection multiplicity). *Let $G, S \in K(C)$ and $P \in C$. Let $T \in K(C)$ be the uniformising parameter for P:*
$G = T^{m_P} S$ and $T(P) = 0$ and $S(P) \neq 0, \infty$. Then m_P is the intersection multiplicity of G at P.

Theorem 16 (Bezout's Theorem). *Let F and G be projective plane curves with degrees m an n respectively. Assume F and G have no common components. Then :*

$$\sum_{P \in F \cap G} I(P) = mn$$

Where $I(P)$ is the intersection multiplicity at point P and $P \in F \cap G$ are the common points of F and G. i.e, the points of intersections.

Definition 17 (Order of Polynomial functions). *The order of a polynomial function $G \in K[C]$ at a point $P \in C$ is the intersection multiplicity at that point and denoted by $ord_P(G)$.*

Definition 18 (Order of rational functions). *The order of a rational function $R = G/H \in K(C)$ at a point $P \in C$ is defined as: $ord_P(R) = ord_P(G) - ord_P(H)$.*

2.2.4 Divisors

The ideals generated by the polynomial in function field of C are sub-varieties of C. i.e, the intersection of roots of $I(C)$ and a rational function. We name them as Divisor.

Definition 19 (Divisor). *A divisor D is a formal sum of points $P \in C$:*

$$D = \sum_{P \in C} m_P P$$

with $m_P \in \mathbf{Z}$ and for all but finitely many $m_P = 0$.

The degree of D is the integer $deg(D) = \sum_{P \in C} m_P$.

The order of D at P is the integer $ord_P(D) = m_P$.

Definition 20 (support of a divisor). *Let $D = \sum_{P \in C} m_P P$ be a divisor. The support of D is the set:*

$$supp(D) = P \text{ in } C : m_P \neq 0$$

Definition 21 (Addition of divisors). *The divisors form a group under addition. The group of divisors of C are denoted by $Div(C)$. We can add two divisors as follows.*

$$\sum_{P \in C} m_P P + \sum_{P \in C} n_P P = \sum_{P \in C} (m_P + n_P) P$$

The subgroup of $Div(C)$ with divisors of degree 0 is $Div^0(C)$.

Definition 22 (GCD of divisors.). *Let $D_1 = \sum_{P \in C} m_P P$ and $D_2 = \sum_{P \in C} n_P P$. Then the $gcd(D_1, D_2)$ is defined by*

$$gcd\left(\sum_{P \in C} m_P P, \sum_{P \in C} n_P P\right) = \sum_{P \in C} min(m_P, n_P) P$$

Definition 23 (Principal Divisor). *Let $R = G/H \in K(C)$ and $G, H \in K[C]$. The divisor of a rational function R is called a principal divisor and defined as:*

$$div(R) = \sum_{P \in C} ord_P(R) P$$

By applying definition 18 at every points of G and H, we know that $div(R) = div(G) - div(H)$. We can see that $div(R) \in D^0$.

Definition 24 (Principal Divisor Group)**.** *The principal divisor group is defined by:*

$$P = \{Div(R) \mid R \in K(C)\}$$

We have,

$$P \subset Div^0(C) \subset Div(C).$$

Definition 25 (Jacobian)**.** *The Jacobian of the curve C is defined by the quotient group:*

$$J = J(C) = Div^0(C)/P$$

Let $D_1, D_2 \in Div(C)$. We have the following equivalence relation on $Div(C)$:

$$D_1 \sim D_2 \Leftrightarrow D_1 - D_2 \in P$$

Or equivalently:

$$D_1 \sim D_2 \Rightarrow \exists R \in K(C) : D_1 = D_2 + div(R)$$

2.2.5 Genus of a curve

The genus of a curve is better explained with the help of a well celebrated theorem in mathematics. For any given curve, the problem which is given below gives the value of genus of the curve and the theorem stated below gives a definition to the genus.

For any divisor D, the set

$$L[D] = \{f \in K(C) \mid (f) + D \geq 0\} \cup 0$$

Where (f) is the principal divisor formed by f. $L(D)$ is the space of all rational functions with poles no worse than D^+ (points having positive order) and zeros of multiplicity at least as specified by D^-.

$L(D)$ is a vector space over K The dimension of $L(D)$ is defined to be $\ell(D)$. The problem of finding the dimension of the vector space is the Riemann-Roch problem.

Theorem 26 (Riemann's Theorem)**.** *There is a constant g such that $\ell(D) \geq deg(D) + 1 - g$ for all divisors D. The smallest such g is called the genus of C. g is always a non-negative integer.*

The theorem stands as a definition for genus of a curve.

2.3 Hyperelliptic curves

With the preparation made till here in this chapter, we are ready to see the definition and properties of Hyperelliptic curves. Hyperelliptic curves are a class of algebraic curves. They can be seen as generalisations of elliptic curves. We classify them depending on the genus of the curve. For all genus, $g \geq 1$ we have hyperelliptic curves. A detailed, simple and beautiful tutorial on Hyperelliptic Curves is available in [MWZ96, Kob94].

Definition 27 (Hyperelliptic Curves). *Let k be a field and K be the algebraic closure of k. A hyperelliptic curve C of genus g over k is defined by an equation of the form.*

$$C : y^2 + h(x)y = f(x) \text{ in } k[x, y] \tag{2.4}$$

Where $h(x) \in k[x]$ is a polynomial of degree at most g and $f(x)$ is a monic polynomial of degree $2g+1$ and there are no solutions $(x, y) \in K^2$ which simultaneously satisfy $y^2 + h(x)y = f(x)$ and the partial derivatives $2y + h(x) = 0$ and $h'(x)y - f'(x) = 0$. A singular point on C is a solution $(x, y) \in K^2$ which simultaneously satisfies all these conditions.

So, in other words, a hyperelliptic curve does not have any singular points.

Definition 28 (Rational points, Points at infinity, finite points). *Let L be an extension field of k. The set of $L - rational$ points on C are denoted $C(L)$ and is the set of points $P = (x, y) \in L \times L$ which satisfy the equation 2.4 of curve C together with a special point* at infinity [6] *denoted by ∞. The set of points $C(K)$ is simply denoted by C. The points in C other than ∞ are finite points.*

Definition 29 (Opposite, special and ordinary points). *Let $P = (x, y)$ be a finite point on C. The opposite point of P is the point $\widetilde{P} = (x, -y - h(x))$ (Note that $\widetilde{P}$ is indeed on C.). We also define the opposite of ∞ by $\widetilde{\infty} = \infty$ itself. If a finite point P satisfies $\widetilde{P} = P$, then it is called a special point. Otherwise P is an ordinary point.*

2.3.1 Examples

1. The figure shows a hyperelliptic curve over $\mathbb{R}$.

$$y^2 = (x-2)(x-1)x(x+1)(x+2)$$

In this example the genus of the curve is 2. The curve can be seen as the first curve in Figure 2.1.

[6] point at infinity is in the projective plane $P^2(L)$.It is the only projective point lying on the line at ∞ that satisfies the homogenised hyperelliptic curves equation. If $g \geq two$ then ∞ is a singular point but allowed since $\infty \notin L \times L$

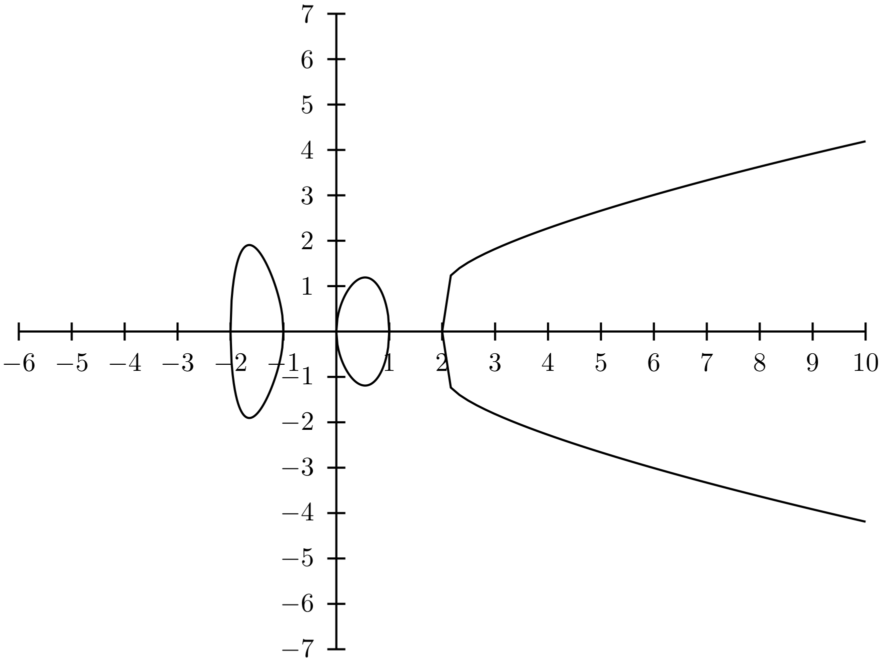

Figure 1: $y^2 = (x-2)(x-1)x(x+1)(x+2)$

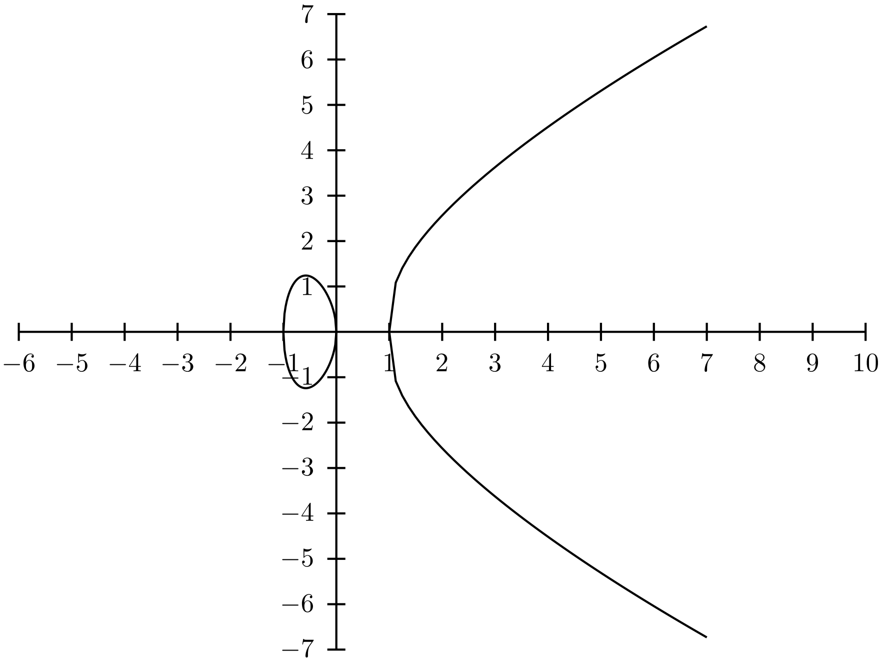

Figure 2: $y^2 = (x-1)x(x+1)$

2. The following figure shows a hyperelliptic curve of genus 1. In other words, this is an elliptic curve. The curve can be seen as the second curve in Figure 2.1.

$$y^2 = (x-1)x(x+1)$$

3. The hyperelliptic curve given by the equation:

$$y^2 + xy = x^5 + 2x^4 + x^3 - 5x^2 + 10$$

   (a) When the curve is defined over the finite field $\mathbb{Z}_{11}$, the valid points are:

$$(1,4),(1,6),(4,2),(4,5),(5,7),(5,10),(8,0)^*,(8,3),(9,5),(9,8)$$

   The point which is starred is a special point.

   (b) When the curve is defined over the finite field $\mathbb{Z}_7$, the valid points are:

$$(1,1),(1,5),(2,6),(3,5),(3,6),(4,4),(4,6),(5,3),(5,6),(6,4)$$

   Here we can see that there are no special points.

Definition 30 (Coordinate ring and polynomial functions.). *The definitions are same as the earlier ones.*

Coordinate ring of C over k.

$$k[C] = k[x,y]/(y^2 + h(x)y - f(x))$$

Coordinate ring of C over K.

$$K[C] = K[x,y]/(y^2 + h(x)y - f(x))$$

Elements of $K[C]$ are called polynomial functions.

Definition 31 (Function field and rational functions). *The function field $k(C)$ of C over k is the field of fractions of $k[C]$. Similarly, $K(C)$ the function field of C over k is the field of fractions of $K[C]$.*

$$K(C) = \left\{ \frac{G}{H} \mid G, H \in K[C], deg(G) = deg(H) \right\}^7$$

An element of $K[C]$ is called a rational function.

[7] the condition for degree is necessary iff G, H are from the homogeneous coordinate ring [Ful69]

2.3.2 Divisors

In the last section we saw all the definitions and primary details of divisors of an algebraic curve. Those are applicable for a divisor of hyperelliptic curves also.

An example would make it clearer with the case in hand.

Example 32. *Let $P = (x_1, y_1)$ be a point on C. Then,*

$$div(x - x_1) = \begin{cases} P + \widetilde{P} - 2\infty & P \text{ is ordinary },\\ 2P - 2\infty & P \text{ is special }. \end{cases}$$

$(x - x_1)$ is the line which is parallel to y axis and passes through the point $(x_1, 0)$. The y value of C for the value x_1 is y_1. The line passes through this point of the curve C. If the point P is an ordinary point, then there are two y values for the same x. These points correspond to P and $\widetilde{P}$.

Or

If P is a special point, its opposite also is the same point P and the line passes through it. So is the divisor.

2.3.3 Semi-reduced divisors

A semi-reduced divisor is a divisor of the form:

$$D = \sum_i m_i P_i - \left(\sum_i m_i\right) \infty$$

Where each $m_i \geq 0$ and all the P_i's are finite points such that if $P \in supp(D)$, then $\widetilde{P} \notin supp(D)$ unless P is special in which case $m_i = 1$.

Fact 33. *For each divisor $D \in D^0$ there exists a semi-reduced divisor $D_1 (D_1 \in D^0)$ such that $D \sim D_1$.*

2.3.4 Reduced divisors

Definition 34 (Reduced Divisor). *Let $D = \sum_i m_i P_i - (\sum_i m_i) \infty$ be a semi-reduced divisor. We call D to be a reduced divisor, if it satisfies the following property.*

$$\sum m_i \leq g$$

Fact 35. *For every divisor $D \in D^0$, there exists a unique reduced divisor D_1 such that $D \sim D_1$.*

2.3.5 Representations of divisors

Divisors are combinations of points on the curves. There are at least three different ways of representing divisors. For the purposes of this thesis, only two of them are needed.

Point or explicit representation

This is the simplest form of representation. This is the representation which directly follows from the definition of the divisor word by word. Here, we represent the divisors just as the formal sum of points along with the order of points. If $P = (x_i, y_i)$ are the points in the support of the divisor and m_i's are the order of point P_i's respectively:

$$D = \sum_i m_i P_i$$

For computational purposes this representation is not advisable. One drawback of this form is that the values of x_i's and y_i's are in K which is the closure of the field k on which we have defined our curve.

Mumford representation

This is the representation which is mostly used for the computing. The reasons for the ease of use in computing will be clear, once the representation is described.

A semi-reduced divisor can be represented with two polynomials. Let D be the divisor.

$$D = \sum_i m_i P_i - (\sum m_i)\infty$$

The two polynomials are:

1. $U(x) = \Pi(x - x_i)^{m_i}$: This is a monic polynomial of degree $\sum m_i$.

   In fact, this is a polynomial which has roots, which have the same x-coordinate as the points in the support of the divisor. The multiplicities of the roots also is the same as the order of the corresponding point.

2. $V(x)$ - for the representation of y coordinates. There are two cases here.

   (a) If all the points P_is are distinct.

$$V(x) = \sum_i y_i \left(\frac{\Pi_{j \neq i} (x - x_i)}{\Pi_{j \neq i} (x_i - x_j)} \right)$$

$V(x)$ is the unique polynomial of maximum degree one less than degree of U. i.e, $deg(V) \leq deg(U) - 1$. Also, that $V(x_i) = y_i$ for all x_i.

(b) If all the points are not distinct.

We have to find out a V which satisfies the following condition along with the condition $V(x_i) = y_i$.

$$V(x) = \left[\begin{array}{c} \text{The unique polynomial of degree smaller than} \\ \sum_i m_i - 1 \text{ such that if multiplicity of } P_i = m_i \\ \left(\frac{d}{dx}\right)^j \left[V(x)^2 + V(x)h(x) - f(x)\right]_{x=x_i} = 0 \\ \text{for } 0 \leq j \leq m_i - 1 \end{array} \right]$$

In other words, $V(x)$ is the unique polynomial such that:

$$(x - x_i)^{m_i} \mid (V(x)^2 + V(x)h(x) - f(x))$$

This, what is seen above is given by the following theorem [MWZ96, Mum84].

Theorem 36. *Let $D = \sum_i m_i P_i - (\sum m_i)\infty$ be a semi-reduced divisor. Where $P = (x_i, y_i)$ are the points and m_i are the order of the points respectively.*

Let $a(x) = \Pi(x - x_i)^{m_i}$ and $b(x)$ be a unique polynomial which satisfies:

1. *$deg(b(x)) \leq deg(a(x))$*
2. *$b(x_i) = y_i$ for all i for which $m_i \neq 0$*
3. *$a(x)$ divides $(b(x)^2 + b(x)h(x) - f(x))$*

Then $D = gcd(div(a(x)), div(b(x) - y))$.

For proof of the theorem [MWZ96] and for more details of Mumford representation refer [Mum84].

Now we can see whether these polynomials $a(x)$ and $b(x)$ are constructable. $a(x)$ is easy. For $b(x)$:

1. $P = (x_i, y_i)$ is ordinary.

   Let $b(x) = \sum_i c_i(x - x_i)^{m_i}$ be the polynomial we need. It is easy to see that $b(x)$ satisfies the conditions. We have to find out the constants c_i's. From $b_i(x_i) = y_i$ we get c_0. Then $(b_i(x)^2 + b_i(x)h(x) - f(x)) = 0$ for $x = x_i$ and for all the $m_i - 1$ derivatives. This gives us enough equations to find out the constants.

2. $P = (x_i, y_i)$ is special. ($m_i = 1$) Here we can directly see that $b_i(x_i) = y_i$ satisfies all the conditions.

Now, using Chinese Remainder Theorem [Knu97] for polynomials, we can find a unique polynomial $b(x_i) \in k[x]$ which can represent the divisor along with $a(x)$.

$$b(x) \equiv b_i(x)(mod(x - x_i)^{m_i}) \text{ for all } i$$

The polynomials $a(x)$ and $b(x)$ together will represent the divisor.

The main advantage of this representation is that we can have the representation in $k[x]$. We need not bother about the closure of k. All the calculations also can be done in $k[x]$. This we will see later.

2.4 Jacobian of a curve

In the last section we saw the definition of jacobian of a curve. In this section the details of Jacobian is further explored.

Let $D^0 = \{\text{set of all divisors of degree } 0\}$. The set of divisors of rational functions form the principal divisors, $f\{P\} \subset D^0$. And the jacobian, J is the quotient group D^0/P.

Let $D_1, D_2 \in D^0$; $D_1 \sim D_2$ if $D_1 - D_2 \in P$. That is, $D_1 = D_2 + (f)$ for some $f \in K(C)$.

By definition itself J is a group.

2.4.1 Group operation in Jacobian

Jacobian is a group by its definition itself and the operation of divisor addition satisfies all the group axioms. But for the sake of a formal-ness, one could say that it is the addition of two reduced divisors.

J is a group of equivalence classes (see facts 33 and 35). Every divisor $\in J$ has an equivalent reduced divisor. In every class of J, there will be a unique reduced divisor. Hence, addition in the jacobian is the addition of two equivalence-classes in the Jacobian. The classes can be represented by the unique reduced divisor present in each class[8]. So, essentially, the addition of two classes boils down to adding two reduced divisors.

Once two reduced divisors are added, the result is either a reduced divisor or a semi-reduced divisor. If it is a reduced divisor it already represents the resulting equivalence-class in which it is a member. On the other hand, if it is a semi-reduced divisor, the

[8] this is similar to representing $\mathbb{Z}_p$ by the smallest positive numbers which can represent the class

reduced divisor equivalent to the semi-reduced divisor to represent the resulting class (there are algorithms to compute the representative reduced divisor of an equivalence class).

Let $[D_1]$ and $[D_2]$ be the two classes to be added. And the result be $[D_3]$.

$$[D_1] + [D_2] = [D_3]$$

This is done by:

$$D_{1r} \oplus D_{2r} = D_{3r}$$

Where D_{1r} represents $[D_1]$, D_{2r} represents $[D_2]$ and D_{3r} represents $[D_3]$ and all D_{ir} are reduced divisors. And $\oplus$ stands for the addition of the divisors and then the reduction.

How do we do this in practice? For the addition in the jacobian, we have many algorithms available. They use different types of representations of the divisors, which are explained in one of the following sections (Section 2.6). Before looking into the details of the algorithms, let us have a look to the different representations of divisors.

Definition 37 (k-rational Divisor). *Let $P(x, y)$ be a point on C and σ be an automorphism of K over k. Then $P^\sigma = (x^\sigma, y^\sigma)$ also is a point on C.*

A divisor $D = \sum m_P P$ is a k-rational divisor if $D^\sigma = \sum m_P P$ is equal to D for all automorphisms σ of K over k.

A divisor is k-rational does not mean that all the points are k-rational.

2.5 Frobenius endomorphism

Let g be a positive integer and let $\mathbb{F}_q$ be a finite field of $q = p^n$ elements. Let C be the hyperelliptic curve defined by $y^2 = f(x)$ where $f(x)$ is a monic polynomial of degree $2g + 1$ with coefficients in $\mathbb{F}_q$ and with distinct roots. The roots (coordinates) may be in the base-field or in an extension field. Let J represent the Jacobian of the curve.

Let us now consider a q-power Frobenius endomorphism $\phi(x) = x^q$. The elements of $\mathbb{F}_q$ are not affected by ϕ, but in the extension fields, it is non-trivial. The map transforms the points on the curve by transforming their coordinates. For the same reason, the mapping extends to divisors, by changing the points.

But on closer look, the action of changing a divisor is also the changes on the coefficients of the Mumford representation of the divisor. Since the divisor is defined over $\mathbb{F}_q$, the mapping ϕ may permute its points but the divisor is left without any change.

2.5.1 Characteristic polynomial

The Frobenius operator ϕ acts linearly and it has a characteristic polynomial of degree $2g$ with integer coefficients. In genus two, the characteristic polynomial has the following form.

$$\chi(X) = X^4 - s_1X^3 + s_2X^2 - s_1qX + q^2$$

so that $\chi(\phi)$ is the identity map on all of J.

According to "Riemann hypothesis for curves", on the roots of the zeta functions of curves, which was proved by Weil, the complex roots of χ have absolute value $\sqrt{q}$. In genus 2, the bounds for it are $\mid s_1 \mid \leq 4\sqrt{q}$ and $\mid s_2 \mid \leq 6q$.

2.5.2 Bounds on cardinality - Weil Interval

The Frobenius is very closely related to the number of points on the curve and also to the divisors in J, over the base field and its extensions.

Theorem 38. *Let J be the Jacobian variety of a hyperelliptic curve C. The group of $\mathbb{F}_q$ rational points on J is denoted by $J(\mathbb{F}_q)$. Let $\chi(t)$ be the characteristic polynomial of the q^{th}-power Frobenius endomorphism of C, then the order of the Jacobian is given by $\#J(\mathbb{F}_q) = \chi(1)$.*

From the theorem it is clear that the knowledge of χ is equivalent of counting the jacobian.

There are two additional results in the case of genus two curves. If $\#C(\mathbb{F}_{q^i})$ is the number of points of C in $\mathbb{F}_{q^i}$, the following are true.

$$\begin{aligned}\#C(\mathbb{F}_q) &= q - s_1\\ \#C(\mathbb{F}_{q^2}) &= q^2 - {s_1}^2 + 2 \cdot s_2\end{aligned}$$

These details help in fixing an interval for the order of the Jacobian. The small interval is called the Hasse-Weil interval.

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

It is also clear, if the order of jacobian is known, determining the characteristic polynomial becomes easier.

For genus 2

The width of the Hasse-Weil interval is close to $4gq^{g-\frac{1}{2}}$. In case of genus 2, $w = 2\lfloor 4(q+1)\sqrt{q}\rfloor$.

2.6 Addition algorithms for divisors

In section 2.4.1, we saw how the Jacobian of a hyperelliptic curve acts as a group and how the group law functions. In this section we will see the various methods for group addition.

2.6.1 Geometrically what is it?

We are concerned about the addition of reduced divisors of a genus g hyperelliptic curve. Consider that we have two reduced divisors, D_1 and D_2.

$$D_1 = \sum_i m_i P_i - \left(\sum_i m_i\right)\infty \quad \text{and} \quad D_2 = \sum_i m_i Q_i - \left(\sum_i m_i\right)\infty$$

where the P_i and Q_i are points on C.

The idea is to find out a curve which passes through the points P_i and Q_i with corresponding intersection multiplicities so that the intersection cycle of the curve will give $D_1 + D_2$. Let us draw that curve. We can see that this new curve will intersect with a few more points of C. Now we have to draw a new curve which passes through the opposites of the new intersections with the same multiplicities of their opposites[9]. This new intersection cycle is the resulting reduced divisor which is the desired sum $D_1 + D_2$

For ease of explanation, we will consider curve of genus 2. So we have:

$$D_1 = P_1 + P_2 - 2\infty \quad \text{and } D_2 = Q_1 + Q_2 - 2\infty$$

From geometry, we know that these points determine a unique cubic polynomial $b(x)$ which passes through them with respective multiplicities (in this case all multiplicities are 1). Substituting $b(x)$ for y in the equation of the hyperelliptic curve, we get:

$$b(x)^2 + b(x)h(x) = f(x) \tag{2.5}$$

Solving the equation gives us 6 solutions (points on the curve) of which 4 of them are

[9] The reason for this is to have an identity element [ST92, Sil86]

known to us. Let the new points be R_1 and R_2. Then the new divisor $D_3 = \widetilde{R_1} + \widetilde{R_2} - 2\infty$ is the sum of D_1 and D_2.

2.6.2 Examples

Example 39. A more visual representation of the aforementioned group law can be seen in the first picture in Figure 2.2.

Blue and red lines are divisors D_1 and D_2 respectively. The new polynomial $b(x)$ is represented by the green curve. It intersects the hyperelliptic curve at six points including P_1 and P_2 of D_1, Q_1 and Q_2 of D_2 and the new points R_1 and R_2. Then we can calculate the opposites of these new points and get $\widetilde{R_1}$ and $\widetilde{R_2}$ respectively. The new curve (magenta) is the resultant divisor.

Example 40. We will see a more concrete example. Let us take a hyperelliptic curve of $g = 1$ which is an elliptic curve. How is the addition done in that case. As the genus is one, the reduced divisor is nothing but a single point P and also the multiplicity cannot exceed one. So our D_1 and D_2 are nothing but two points P and Q.

$$y^2 = (x-2)x(x+4)$$

We have the elliptic curve with points P and Q to be added together. We find out a line(curve) passing through them. This new line passes through the point R. Now we take the reflection of R on x-axis. The reflection $\widetilde{R}$ is the sum of P and Q.

The group law is depicted in the second picture in Figure2.2.

2.6.3 Algebraic methods

In the last section we got a feel of how to add divisors. In the case of elliptic curves it was very easy. But in the general case, drawing a curve is not an easy thing. What we can do is that algebraically find out the equation of the curves/divisors. This is not very difficult. In fact, we can use the representations we saw in the last section. The one which is most popular is Mumford representation. The polynomials $a(x)$ and $b(x)$ have all information about the divisors. Also there is a method which uses Chow forms.

Cantor's Algorithm

In 1987 Cantor [Can87] came up with an algorithm for the addition of reduced divisors of hyperelliptic curves. The algorithm is known as Cantor's algorithm. As we have two

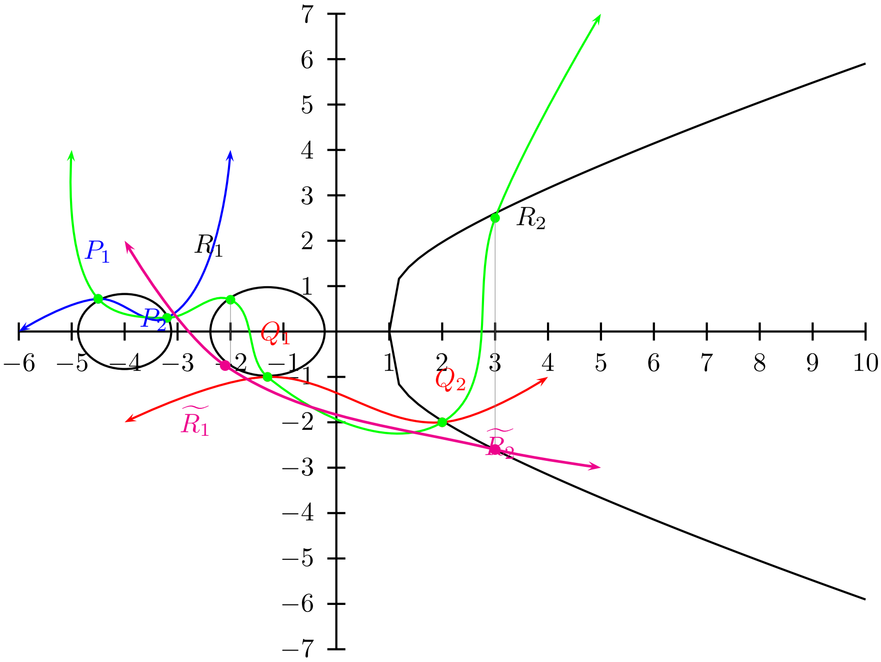

Figure 1: Addition of divisors in hyperelliptic curves

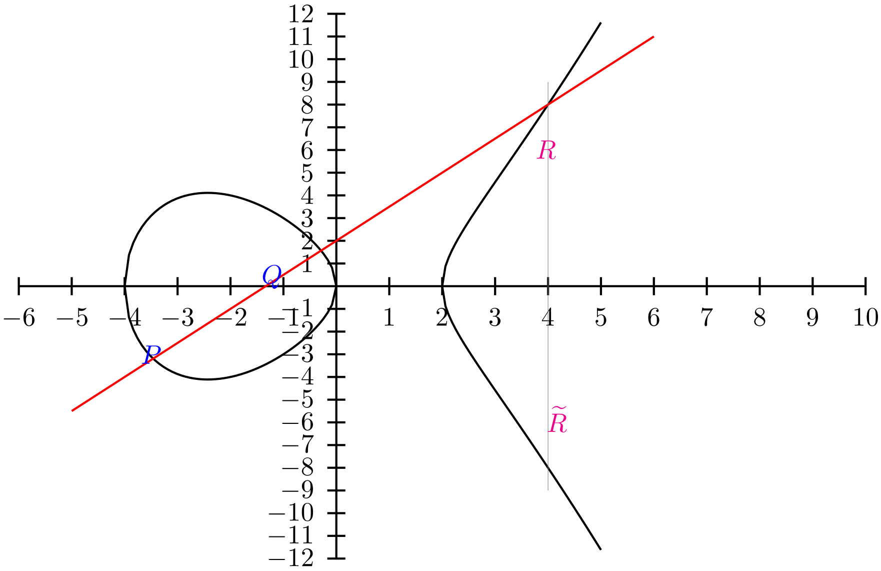

Figure 2: Group law in group of points of elliptic curves

phases for addition of divisors, the algorithm also has two phases. Cantor's method uses Mumford representation.

1. Composition : This is the phase in which we find out the new divisor which is the sum of the input divisors.

2. Reduction : In this phase we prune out the parts which are not needed and take the inverse of the resulting one to get the result.

Composition

- **Input**: Reduced divisors $D_1 = div(a_1, b_1)$ and $D_2 = div(a_2, b_2)$ both defined over k.

- **Output**: A semi-reduced divisor $D = div(a, b)$ defined over k such that $D \sim D_1 + D_2$.

1. Use the extended Euclidean algorithm to find polynomials $d_1, e_1, e_2 \in k[u]$ where

$$d_1 = gcd(a_1, a_2) \text{ and } d_1 = e_1a_1 + e_2a_2$$

2. Use the extended Euclidean algorithm to find polynomials $d, c_1, c_2 \in K[u]$ where

$$d = gcd(d_1;\ b_1 + b_2 + h) \text{ and } d = c_1d_1 + c_2(b_1 + b_2 + h)$$

3. Let $s_1 = c_1e_1, s_2 = c_1e_2$ and $s_3 = c_2$, so that

$$d = s_1a_1 + s_2a_2 + s_3(b_1 + b_2 + h) \tag{2.6}$$

4. Set

$$a = a_1a_2 = d_2 \tag{2.7}$$

$$b = \frac{s_1a_1b_2 + s_2a_2b_1 + s_3(b_1b_2 + f)}{d} \text{ mod } a \tag{2.8}$$

Again, I am not providing the proof. In fact, the proof given by Cantor contained a few errors and later when Koblitz gave the algorithm, he did not give any proof. But for the proof, readers can refer [Kob98, Can87, MWZ96].

Reduction

Reduction part is comparatively simpler and easy to understand. Here also we skip the proof - for details [Kob98, Can87, MWZ96]. But here, the algorithm is pretty clear that no one needs a proof to see that it is indeed correct.

Input: A semi-reduced divisor $D = div(a, b)$ defined over k.

Output: The (unique) reduced divisor $D' = div(a', b')$ such that $D' \sim D$.

1. Set

$$a' = (f - bh - b^2)/a \tag{2.9}$$

$$b' = (-h - b) \bmod a' \tag{2.10}$$

2. If $deg_x(a') \geq g$ then set $a \leftarrow a'$, $b \leftarrow b'$ and go to step 1.

3. Let c be the leading coefficient of a', and set $a' \leftarrow c^{-1}a'$

4. Output(a', b').

Other versions

As we told in the introduction, we are looking for faster addition in the jacobian of hyperelliptic curve. The method given above is a polynomial time algorithm. But when it comes to the number of micro-instructions needed in a processor, this version of the algorithm is too general to implement. So we have many different refined version of this algorithm or slight variants of this algorithm which is tailor made for different genus hyperelliptic curves.

Harley's Algorithm

In the year 2000, Rob Harley [Har00] came up with an algorithm which is very similar to the original Cantor's algorithm. The algorithm was optimised and made for genus two curves [GH00]. In the tailor made method for genus two curves, addition and doubling are handled separately. Finally the number of operations comes up to two inversions, three squarings and 24 multiplications for the genus 2 case.

Explicit formulae by Tanja Lange

In her paper in 2002, Tanja Lange [Lan02] gives explicit formulae for arithmetic on genus two curves over fields of even characteristic and for arbitrary curves. The formula is faster than all the methods which existed before that. It allows to obtain fast arithmetic on hyperelliptic curves of genus 2. The algorithm is a case by case analysis of different situations which can arise.

2.7 ECDLP and HECDLP

In the earlier section 2.1.2, we saw that, in the multiplicative group $\mathbb{Z}_p^*$, the discrete logarithm problem is: given elements r and q of the group, and a prime p, find a number k such that $r = q^k \text{mod } p$.

In the previous sections we also introduced new types of groups: the groups of points on the Jacobian of hyperelliptic curves. It is also to be noted that elliptic curves are hyperelliptic curves of genus 1. They also have a jacobian and the jacobian is simply the group of points on an elliptic curve. The main advantage of these groups - the points on an elliptic curve or the jacobians of hyperelliptic curves - compared to the multiplicative groups of finite fields, is that under certain conditions, there is no method like index calculus known to solve the DLP. If the groups are chosen with care, then the most efficient way to solve the DLP is by means of Pollard's rho method [Pol78].

For this method, one has to perform roughly $\sqrt{\#J}$ group operations. This means that its running time is exponential in $\lg \#J$, and one can use smaller groups for achieving the same level of security.

2.7.1 Elliptic Curve Discrete Logarithm Problem

The points on an elliptic curve form a group under addition. The elliptic curve discrete logarithm problem could be defined as: given points P and Q in the group, find a number that $k \cdot P = Q$; k is called the discrete logarithm of Q to the base P.

Example:
In the elliptic curve group defined by

$$y^2 = x^3 + 9x + 17 \text{ over } F_{23}$$

What is the discrete logarithm k of $Q = (4, 5)$ to the base $P = (16, 5)$?

In comparison with the example given in section 2.1.2, one can already see that this question is very hard.

The (naive) way to find k is to compute multiples of P until Q is found. The first few multiples of P are:

$P = (16, 5);\ 2P = (20, 20);\ 3P = (14, 14);\ 4P = (19, 20);\ 5P = (13, 10);\ 6P = (7, 3);\ 7P = (8, 7);\ 8P = (12, 17);\ 9P = (4, 5)$

Since $9P = (4, 5) = Q$, the discrete logarithm of Q to the base P is $k = 9$.

In a real application, k would be large enough such that it would be infeasible to determine k in this manner.

2.7.2 Hyperelliptic Curve DLP

As the name itself indicates, HECDLP is the general version of ECDLP. Unlike having the points on the curve, which is used in ECDLP, HECDLP depends on the group of reduced divisors for defining the DLP.

The hyper elliptic curve discrete logarithm problem could be defined as: given divisors D_1 and D_2 in the group, find a number that $k \cdot D_1 = D_2$; k is called the discrete logarithm of D_2 to the base D_1.

As mentioned earlier, the main advantage of HECDLP is the lack of existence of any sub-exponential algorithms to solve it. With today's computers it is reasonable to assume that it is unfeasible to perform 2^{80} operations in a reasonable amount of time. Under this assumption, it follows that cryptosystems based on elliptic or hyperelliptic curves are secure if $\#J \sim 2^{160}$. As a consequence, these systems are more efficient, and allow shorter key sizes than their multiplicative group counterparts. The details of the sizes needed were explained in Table 1.1.

Chapter 3

The State Of The Art

"Do not worry about your difficulties in Mathematics,
I can assure you, mine are still greater.

- Albert Einstein.

In the previous chapter, we saw how the points (or divisors) of a hyperelliptic can form a group, which is called the jacobian of the curve. We also did see that the count (cardinality) of this group is vital in deciding the level of security the curve can offer. In this chapter, the methods which already exist for calculating the order of groups are sketched.

We will start by examining the algorithms which can be used for generic groups and then observe how these algorithms are tailor suited for the special cases of jacobian. Counting methods which leverage the properties of hyperelliptic curves also are explained.

A note on notation: All through this chapter, G is the group of which the order is to be computed; M is the upper-limit of the order and α is a random element which has the order N.

3.1 Existing methods for counting

Owing to the property of groups that the order of the group is a multiple of the order of the elements of the group (or even the same sometimes), the very naive method starts by finding the order of an element α.

The order of an element is the number for which the exponent of the elements produces the unit element of the group. So, one starts by finding the exponents of α until the unit element of the group is produced.

$$\alpha, \alpha^2, \alpha^3, \ldots \alpha^N$$

Since the order o of the group is the number for which the the N^{th} power of every element in the group is same as the unit element, quite often the order of an element is the same as that of a group.

If they are not the same, one could find out the orders of different elements of the group and find out the least common multiple of them which is either the order of the group or a large factor of it.

Determining the order of a group (or α) using the above outlined method is very expensive. The larger the order of α, the more expensive the computation becomes. The complexity of the algorithm is $O(N)$.

There are two methods which are based on the aforementioned idea, but better in complexity. The goal of the algorithm is to find an n such that $\alpha^n = 1_G$ where 1_G is the unit element. So it suffices to find two powers of α which are equal.

If i and j are two numbers for which $\alpha^i = \alpha^j$, then we know, $\alpha^{j-i} = 1_G$ and that $j - i$ is the number being sought. It is possible that $j - i$ is not n, but a multiple of n.

3.1.1 Baby Step Giant Step - BSGS

One of the methods which leverages the idea of finding a multiple of α is the baby step giant step method. It was proposed by Shanks [Sha71] and it finds the order of the element.

All what is needed is an upper limit of the group order.

Assuming the order of the group is M, the algorithm takes a random α from the group and computes two sets of its exponents. As the name of the method indicates, one set is created using baby steps and the other using giant steps.

Baby Steps

Every exponent of α is calculated and stored in a list up to $\sqrt{M}$. Along with the values, the exponent-value of α is also stored.

This set can otherwise be seen as the set of α^k for $k <= \sqrt{M}$.

Giant Steps

Exponents of α is computed for every multiples of $\sqrt{M}$ until M. This set is α^j for $j = i \times \sqrt{M}$

After computing each exponent, it is checked whether the value is present in the first list created by baby steps. If it is present there, then the result is at hand.

If one of the giant steps is present in the baby steps, the number j and k mentioned in the previous section is found out.

The algorithm can proceed to compute the N (or a multiple of it) by finding the difference of the exponents which produced the same value.

Example:

Consider a group, G which has an order close to one million. Using the initial naive method, one has to compute every exponent of α for all values $1, 2, \ldots, 1000000$.
In baby-step-giant-step method, one can compute the powers of α as below.

$$\alpha, \alpha^2, \alpha^3, \ldots \alpha^{1000}$$

Then, compute and check each of the following until a match is found. The smallest difference in the powers which give the same exponent would be N.

$$\alpha^{1000}, \alpha^{2000}, \alpha^{3000}, \ldots \alpha^{1000000}$$

The complexity of the algorithm is $O(\sqrt{N})$ to calculate N.

3.1.2 Birthday paradox method

This method, which is also known as Pollard's Rho method, is based on the well known birthday paradox. The idea of birthday paradox is that if we randomly take $\sqrt{k}$ elements from a set of size k, there is a good chance we will pick an element twice[1]. For more details, please see [Pol78].

While putting the birthday paradox into use for order computation, we guess a random sequence of powers of α and we are bound to have a repetition sooner or later.

The algorithm starts by randomly guessing powers of α, check whether that value was computed earlier. If it was, the j and k of the previous section are available and their difference is the order of α. If the power was not computed earlier, it is added to the list and the algorithm continues.

Just as in the baby-step-giant-step method, the complexity of the algorithm is $O(\sqrt{N})$.

[1] If there are at least 23 people in a room, the chance that two of them share the same birthday is high

3.1.3 Order counting with Primorial

In [Sut07], A. Sutherland proposed a method of making the counting of the order of a group easier, by using some rudimentary facts about groups. In effect, computing the order relies on Pollard's methods. But the rudimentary methods help to save resources.

The order of an element of a group is always a factor of the order of the group. If a rough idea about the upper limit of group order is available, one can carefully choose an element for which the order has no factors smaller than prime P.

The idea is to start with a random element α, and with M being the upper limit of the group order. According to the Primorial method, one finds the largest power of 2, smaller than M - let's call it k and computes $\beta = \alpha^{2^k}$. From the properties of the group, β has an order which is co-prime to 2.

One can do the same with all prime numbers - $2, 3, 5, \ldots$.

$$\beta = \alpha^{\Pi p^{k_p}}$$

By computing a β with order coprime to the small primes, in the traditional group order algorithms, the numbers which are coprime to the product of all the primes can be avoided and a great saving in time/space is achieved.

3.2 Counting in the Jacobian

If we consider a hyperelliptic curve over $\mathbb{R}$, or even over the complex numbers, it is meaningless to talk about counting the number of points, because it is infinite.

But once we define a curve over a finite field, the number of points is not infinite anymore and becomes important because of the application of the curves and the significance played by the number of points (order of jacobian).

One could imagine any kind of finite field over which a hyperelliptic curve can be defined. Let us broadly put them into two classes - fields of characteristic 2 and fields with an odd prime as the characteristic.

3.2.1 Fields with small characteristic

Many results are available for counting points over finite fields of characteristics p, mainly with the use of p-adic liftings. The complexity of these methods is exponential in $\ln p$. Hence for small primes (especially for $p = 2$) these methods are good. Using these algorithms, it is possible to compute orders of Jacobian varieties in sizes of cryptographic usage over fields with small characteristic.

The main result with small fields is available at [Ked01, GG01].

3.2.2 Fields with larger/arbitrary characteristic

The story takes a not-so-happy turn when the counting has to be done over fields of arbitrary characteristic. There are a few results of which only one is considered to be of practical use: Gaudry-Harley [GH00, Gau05].

3.2.3 Gaudry-Harley method

The Gaudry-Harley algorithm consists of two main parts. In the first part, details about the order of the Jacobian is computed based on different results.

The detail which is computed is the residue modulo $m, m \in \mathbb{Z}_{>0}$ of the order of the given jacobian variety. The details of extracting this information is described in the following section.

Once the residue modulo m is calculated, in the second phase of the algorithm, a parallelised Pollard's lambda method is used in finding the number which matches the acquired information and also the properties of the group order. This square root algorithm was taking the lion's share of computing resources in the Gaudry-Harley algorithm.

The initial part of the algorithm, where the residue modulo m details are computed, has two smaller sections.

Cartier Manin

For hyperelliptic curves, the characteristic polynomial of the Frobenius modulo p is connected with the Hasse-Witt matrix of the curve. The $g \times g$ matrix (Hasse-Witt) can be computed using the following result.

Theorem 41. *Let $y^2 = f(x)$ represent a hyperelliptic curve with genus g. Let the coefficient of x^i in the polynomial $f(x)^{\frac{(p-1)}{2}}$ be represented by c_i. Then the Hasse-Witt matrix is given by:*

$$A = (c_{ip-j})_{1 \leq i,j \leq g}$$

In [Man65, GH00], the above result is connected to the residue modulo p of Frobenius. For a matrix $A = (a_{ij})$, let A^p be the element-wise p^{th}-power. i.e., (a_{ij}^p).

Theorem 42. *Let C be a curve of genus g defined over a finite field F_{p^n}. Let A be the Hasse-Witt matrix of C, and let $A_\phi = AA^p \dots A^{(p^{n-1})}$. Let $k(t)$ be the characteristic polynomial of the* *matrix A_ϕ, and $\chi(t)$ the characteristic polynomial of the Frobenius endomorphism. Then*

$$\chi(t) \equiv -1^g t^g k(t) \; mod \; p$$

Armed with this knowledge, it is straight forward to compute $\chi(t)$ modulo the characteristic p and hence $\#J(\mathbb{F}_q) \; mod \, p$. One must notice that this method would be feasible only for not too large p. In practice, any $p > 1000000$ is too expensive.

This is a very efficient way of knowing about the Jacobian order, for moderately large p.

Schoof's Improvement

A detailed explanation of Schoof's method (extended by Gaudry) is beyond the scope of this thesis. A more interested reader is recommended to read [Kam91, Pil90].

In a nutshell, the hyperelliptic analogue of Schoof's algorithm[2] is about computing χ modulo l for small primes. Once this has been done for sufficiently many primes, Chinese remainder theorem [Knu97] is used to compute χ. From the bounds which were mentioned above, the primes are bounded by $l = O(\lg q)$.

For the methods of computing χ mod l, the reader is encouraged to peruse [Kam91, Can94].

Modified Pollard's Lambda

From the Cartier Manin operator and Schoof's method, different details of $o(J)$ is computed as $o(J) \equiv n_1 \bmod p$, $o(J) \equiv n_i \bmod l_i^k$, for $l = 2, 3, 5, \ldots$, $k \in \mathbb{Z}$.

Using Chinese remainder theorem the aforementioned information is combined to $o(J) \equiv n \bmod m$ where m is the product of p and all values of l^k.

The birthday paradox algorithm which was mentioned earlier in the chapter can be modified to avoid computing the values which are not in the form of n mod m. Using the modified algorithm, computing resources can be saved up to an order of $\sqrt{m}$.

For further details, see [GH00]

3.3 Can this be improved?

So far in the thesis, we discussed about the counting in the Jacobian of hyperelliptic curves of genus 2. In this section, we give a more clear description of our goal.

[2] Originally designed for elliptic curves

In section 3.2.3, we saw the state of the art methods for generic curves. It was also seen that the running time of the algorithm is too long to have any practical use in cryptosystems. We also have seen a few of the generic methods of group order counting. The goal of this thesis is to bring the best out of the both the worlds. i.e., to try to apply the advances of generic group order counting in the special setting of hyperelliptic curves, where one has the advantage of

- Weil Interval
- Cartier Manin operator
- (modified) Schoof's method

The details of how they can be applied together are explained in the next chapter. The next chapter also shows what an improvement is achievable.

Chapter 4

The Solution

> "The art of doing mathematics consists in finding that special case which contains all the germs of generality."
>
> - David Hilbert

In this chapter, we develop a faster method for counting in the jacobian of hyperelliptic curves (of genus 2). Starting with the naive solution which tries to apply the Primorial method [Sut07] in the jacobian, the chapter witnesses the gradual development of an algorithm which combines all the special knowledge available (about the jacobian).

The final algorithm has an improvement factor of $\frac{\varphi(P)}{P}$[1] over the conventional method. Even though the computational complexity of the algorithm is the same as the traditional one, the improvement is a quite good, in computing in the jacobian of practically useful hyperelliptic curves[2].

A note on notation: In this chapter, P is the Primorial (multiple of primes up to p), and T is the totient of P, m is the characteristic of the field on which the curve is defined. The variables w, l and u represent the Weil interval, its lower and upper boundaries - respectively. $\#J$ stands for the size of the jacobian.

[1] where P is the product of the first p primes. The value of p depends on the size of the jacobian

[2] In this chapter, a hyperelliptic curve (of genus two) which is practically applicable is assumed to be on a field of size 2^{80} which in turn means, the size of the jacobian is $\sim 2^{160}$ and size of Weil interval is $\sim 2^{120}$.

Size(crv)	Size (Jac)	P	Reduction $\frac{P}{\varphi(P)}$	Speed-Up $\sqrt{\frac{PP}{\varphi(PP)}}$	Work %	Days ($7 \times \%$)
10	20	11	4.812	2.193	45.584	3.190
20	40	17	5.539	2.353	42.488	2.974
30	60	29	6.331	2.516	39.742	2.781
40	80	31	6.542	2.557	39.096	2.736
50	100	41	6.892	2.625	38.091	2.666
60	120	47	7.209	2.685	37.242	2.607
70	140	59	7.474	2.734	36.576	2.560
80	160	67	7.714	2.777	36.003	2.520
90	180	71	7.824	2.797	35.748	2.502

Table 4.1: Improvements for different curves.

4.1 Primorial Vs Weil

Counting in the jacobian (of hyperelliptic curves) is easier than the counting in any other group of similar size, primarily owing to the knowledge of the search intervals.

But the traditional algorithms and the Primorial method, which we have in hand, are not fine tuned for searching in intervals. So, the naive attempt could be the application of the Primorial method to see whether the improvements promised by it would be good enough to surpass the benefit of Weil's interval.

A hyperelliptic curve defined over a field of size q has a jacobian of the size comparable to $N = q^2$. Using the traditional methods of counting implies that the search has a complexity $\sqrt{N} = q$. See [GH00].

Using the Primorial method, one can avoid a large share of the search space. Table 4.1 gives an idea of how much improvement in search can be achieved for different sizes of groups. The sizes (represented in bit-size) are of the field over which the curve is defined and the jacobian respectively.

The last column is the number of days needed for the algorithm to finish running, assuming that the non-Primorial version would need a week (7 days) to finish.

The size of the Weil interval is of the order of $q^{1.5}$ for genus 2[3]. So, Weil interval is a very small fraction ($q^{\frac{-1}{2}} \sim 2^{-40}$for a practically useful curve) of the whole jacobian of a hyperelliptic curve, where as the Primorial savings is of the order of less than 65%.

Naive Primorial reduces the search space to two thirds, where as the whole Weil-interval itself is one in 2^{40}th of the Jacobian. Therefore, the naive method of trying to apply the Primorial method directly on the whole jacobian of a curve is extremely more expensive than using the Weil knowledge.

[3] The weil interval is specified as $\lceil(\sqrt{q}-1)^{2g}\rceil \leq \#J(F_q) \leq \lfloor(\sqrt{q}+1)^{2g}\rfloor$. For more details see 2.5.2

4.2 Primorial and Weil

Since the naive method of applying the Primorial directly on the Jacobian is not very promising, the advantage of Weil's interval has to be taken into consideration.

Note: Primorial method only guarantees that the order of the chosen (created) element is co-prime to the selected primes (Please see 3.1.3). The order of the group is a multiple of this order and need not be co-prime to the primes chosen.

Instead of the naive method of searching in all over the jacobian, the search has to be modified to specific intervals inside the jacobian.

1. The Weil interval $[l, u]$ is initially searched for the element order. There search would be successful only if the group order is the same as the order of the element, or the group order is a product of the order of the element and a number which is co-prime to the Primorial[4].

2. Since the order of the Jacobian is sure to be within the Weil interval and since the order of the divisor has to be a factor of it, if the order was not found in the case above, we can avoid the searches is such ranges where the order cannot be present and move on to the intervals where the probability of finding the order is non-zero.

   For example, the range $[\frac{w_u}{2}, w_l]$ is sure not to have the order - because no multiple of any number in that interval is within the Weil interval.

   So, if the order was not found in Weil interval, the search has to continue to $\frac{w}{2}, \frac{w}{3}, \ldots = [l, u], [\frac{l}{2}, \frac{u}{2}], \ldots$. until the order of the element is found.

Considering the search spaces as described above, the counting algorithm has to look in a set of special intervals (defined by Weil interval). Which in other words mean that set of intervals could be skipped while doing BSGS. The following set contains all such intervals.

$$\left\{ \left[\frac{l}{i}, \frac{u}{i+1} \right] \mid 1 \leq i < r = \frac{u}{w} \right\}$$

The remaining search space - from 1 to w_u minus the above set, will provide us with the size of the real search space.

[4] While using the primorial method, the orders computed are coprime to the primorial. Order of the element is assured to be coprime to the primorial. If the order of the group is a multiple of the order of the element, the multiplier has to be coprime to the primorial, otherwise it would never have been computed.

$$\begin{aligned}
S &= \frac{u}{1} - \frac{l}{1} + \frac{u}{2} - \frac{l}{2} + \ldots + \frac{u}{r} - \frac{l}{r} + \frac{u}{r+1} \\
&= \frac{w}{1} + \frac{w}{2} + \ldots + \frac{w}{r} + \frac{u}{r+1} \\
&= S_1 + S_2 \\
&> w\left[1 + \frac{\lg r}{2}\right] + S_2 \\
&= w\left[1 + \frac{\lg q^{\frac{1}{2}}}{2} - 3\right] + S_2
\end{aligned}$$

4.2.1 Put in perspective

From the above equation, if we calculate the search space S, that will be $S = 19w + S_2$. As mentioned in the previous section, even for large primes (the ones used for Primorial at cryptographic sizes, usually less than 67), the factor $c = \frac{\varphi(p)}{PP}$ is about than 0.25

Using the Primorial advantage along with the reduced search space, the resulting search space would be $f \cdot S = 4.95 \cdot w + 0.25 \cdot S_2$. This value is definitely larger than w, the Weil interval.

From this, it is clear that there is no advantage of using the Primorial method, if we know a small interval where we expect the order to be present.

Using Primorial method along with Weil interval is actually more expensive than the method without Primorial. This arises from the fact that a non-Primorial method has to do the search only inside the Weil interval, where as Primorial method needs a search outside the interval.

4.3 Including Cartier Manin operator

Using the Cartier-Manin operator (see 3.2.3 and [Man65]), some more details of the group order can be obtained, provided the characteristic of the underlying field is not a very large prime number[5].

The group order can be represented in terms of m, the characteristic of the field as follows:

$$\#J = N_r \; mod \; m$$

[5] Any prime larger than 10^6 is considered too large.

As the original Primorial method avoids making the (computing the) unnecessary babysteps or giantsteps based on the knowledge of co-primeness of the order to initial primes, a new method could incorporate this special knowledge above for avoiding the computation of baby/giant-steps which are not of the form as in the equation above.

The method to find out the jacobian using baby-step-giant-step method is sketched as Algorithm 1.

Algorithm 1: Compute Jacobian order: Cartier Manin and Primorial BSGS

```
Data: $J$, $w$, $w_l$, $w_u$, $m$, $N_r$, $\alpha$ - random element of $J$
Result: $\#J$ - Order of the jacobian
begin
    $N_r \longleftarrow CartierManinOn(J, m)$;
    $p, P \longleftarrow BestPrimorial(w, N_r)$;
    $T \longleftarrow BestLimitBabies(w, N_r, P)$;
    $\beta \longleftarrow DivisorWithSpecialOrder(\alpha, p)$;

    * Generating the babysteps *;
    $B_0 \longleftarrow \{k \cdot m - N_r \mid 0 < k \leq (1 + T/m)\}$;
    $B_1 \longleftarrow \{j \perp pp \mid j \leq T, j \in \mathbb{Z}^+\}$;
    $B \longleftarrow B_0 \cap B_1$;
    $Babysteps \longleftarrow \emptyset$;
    for $x \in B$ do
        $Babysteps \longleftarrow Babysteps \cup \{x \cdot \beta\}$;
    end

    * Generating giants and checking against babysteps *;
    $Giants \longleftarrow \{g \mid w_l \leq g \leq w_u;\ m, T \text{ divides } g\}$;
    for $x \in Giants$ do
        if $x \cdot \beta \in Babysteps$ then
            $\#J \longleftarrow ComputeDifference(x, Babysteps)$ ;
        end
    end
    return $\#J$
end
```

Details of Functions

In the algorithm 1, there are some functions which are not yet been well defined. A description of each of them is given below. The functions being standard and straightforeward, the pseudo-code for the same are avoided.

$CartierManinOn(J, m)$**:** As described in section 3.2.3, the CartierManin operator can be used to find the remainder of $\#J$ when divided by m, the characteristic of the field. Please see [Man65] for further details.

$BestPrimorial(w, N_r)$: Once the values of Weil interval and the special knowledge of $\#J$ are obtained, the best prime number for the algorithm can be decided. This function is explained below.

$BestLimitBabies(w, N_r, P)$: The number of babysteps and giantsteps are defined by upper limit of the babystep. This function finds out the optimal value of the limit, based on the available inputs.

$DivisorWithSpecialOrder(\alpha, p)$: The element which is to be used for baby-steps and giant-steps has to meet the requirement that its order should be co-prime to the Primorial (i.e., co-prime to each of the prime numbers $\leq p$). To compute such a divisor, β, a random divisor is chosen and it is exponentiated to all the powers of the prime numbers in question. For more details, see section 3.1.3 or [Sut07].

4.3.1 The Cost - Reduction and Estimation

Like the traditional BSGS, the cost is based on the number or babysteps and giantsteps to be made before the order can be computed. The best result is achieved when these two numbers are roughly the same. See [Coh96].

The cost can be found out by looking at the two functions which are mentioned above.

$BestPrimorial(w, N_r)$: It would seem that the larger the size of the Primorial, the lesser the number of computations needed. But the increase in the size of the Primorial is advantageous to the computation only up to a limit. And this limit depends on the size of the search space. In the normal scenario, where the restriction of N_r is unknown, the optimal value of Primorial is the one closest to $\sqrt{w \cdot \frac{P}{\varphi(P)}}$, which is also used as the upper bound of baby steps. See [Sut07].

But with the introduction of N_r, the number of babysteps in the same interval is reduced by a factor of N_r, where as the number of giantsteps remain the same. To share the advantage of N_r between the babysteps and giantsteps equally, the optimum value of the Primorial moves higher by a factor of $\sqrt{N_r}$.

Hence, the optimal value of Primorial would be the Primorial closest to $\sqrt{w \cdot \frac{P}{\varphi(P)} \cdot N_r}$

$BestLimitBabies(w, N_r, P)$: Once the best Primorial has been established, computing the upper-bound is all what is needed for starting babysteps. Looking at the giantsteps, one can see that the giantsteps has to be divisible by m because of the properties of babysteps. Each giantstep has to be also divisible by the upper-bound. Hence, the upper-bound shall be a multiple of m. Now with this knowledge, the upper-bound can be fixed as the multiple of m closest to the optimal Primorial which was computed in the previous step.

Improvement from traditional methods

In terms of complexity of the methods, the use of CartierManin or Primorial has not reduced the complexity from $O(N^{\frac{1}{2}})$. But by the reduction in the number of steps, the employment of both Cartier Manin and Primorial methods provides a reduction of $\sqrt{m \cdot \frac{P}{\varphi(P)}}$ in the computations.

Proof

Using the Cartier Manin operator reduces the size of search space by a factor of m, the characteristic of the field over which the curve is defined. As the complexity of the algorithm is $O(\sqrt{n})$, an speed up of $\sqrt{m}$ is attained.

While using Primorial, only those numbers which are co-prime to P are used for the baby-steps or for giant-steps.

The number of numbers co-prime in every P numbers is $\varphi(P)$. In the Weil interval, there fore the total number of giant-steps + baby-steps is reduced by another factor of $\sqrt{\frac{\varphi(P)}{P}}$.

Therefore, both the methods of CM and Primorial gives a speed increase of $\sqrt{m \cdot \frac{P}{\varphi(P)}}$

4.4 Including Schoof's improvement

In this section and the following section, we will see the final products of this thesis. In this section, we work on improving the counting method by incorporating the details obtained by Schoof's method in knowing some more information about the Jacobian order.

For further details on Schoof's method, please see [Sch95, Can94].

Schoof's method provides information about the jacobian in a similar fashion like Cartier Manin.

$$\#J = n_s \, mod \, 2^k \cdot l$$

Where l is the product of initial primes up to p_1. Usually, in practice, the value of p_1 is less than 17, for the reason that computation of modulo p becomes expensive exponentially. Details of the computation could be seen at [GH00].

So, the details of the order of the jacobian available are as below.

$$\begin{aligned}\#J &= N_r \, mod \, m \\ \#J &= n_s \, mod \, 2^k \cdot l \\ \#J &\perp P\end{aligned}$$

Using Chinese Remainder Theorem [Knu97], the first two of the modulos can be combined as follows.

$$\begin{aligned}\#J &= n \, mod \, m \cdot 2^k \cdot l \\ \#J &= NmodM\end{aligned}$$

Even with the additional Schoof's method, one can re-formulate the counting problem in a way similar to how it was when only Cartier Manin operator was used, which is helpful in using the same (or slightly modified) method, provided the new problems are ironed out.

4.4.1 The trouble introduced

Before the introduction of modulo by Schoof, the modulo and the Primorial were coprime[6]. In fact, the common factor of m, N_r and P was 1, before the introduction of Schoof's method.

But if the common factor d is not 1, then no single integer in the Weil interval which satisfies the modulo would be coprime to the Primorial, P. In the last section, where CartierManin operator was used, there is a subset of integers which satisfy the modulo which are not coprime to P.

[6] m is a large prime number and P is the product on initial primes

4.4.2 The algorithm

In a nutshell, the solution tries to generate babies and giants which satisfy the conditions and produce the results. But the names shall not confuse the reader, and make her think that babysteps always have to be first. In our solution, we are generating the giants steps initially and the babysteps are made to match the giantsteps stored.

An optimal value for the size of the giant step is computed using the available values of Primorial and the m. Using this value, the giantsteps are made in the Weil-interval and are saved. As the next step, the babysteps are produced. Unlike in the earlier cases, where we suggested searching in the small intervals if the searched value was not found in the original one, here we suggest to adjust the value of the Primorial to accommodate the generation of more babysteps - which is the equivalent of going to lower-intervals[7].

The method mentioned above is sketched in Algorithm 2.

Details of Sections of the algorithm

As mentioned above, the algorithm has 3 sections which are explained here one by one.

Choosing the optimal giantstep size

Similar to what we saw in the previous algorithm, the introduction of Cartier Manin results and the Primorial values reduces the search space by a large factor. Schoof's method throws new light to the Jacobian order, thereby reducing the search space further[8].

The reduction of the space by a factor k directly implies that the size of the giantstep would be $\sqrt{k}$ times larger than the traditionally optimal size.

But, the size of the giantstep has to be a multiple of P and also m (m in $\#J = n \bmod m$), for the babysteps are going to be of the form $P \perp b = l \cdot m - n$, for $l \in \mathbb{Z}$.

Considering these two restrictions, one has to choose, g, the size of a giantstep as close as it can be to $\sqrt{w \cdot k}$, where w is the size of Weil-Interval; where g is a multiple of m and P.

As shown in the code, we choose the Primorial chosen for a case without Schoof's and CM information available, and remove the larger primes until the size of the giantstep is in the neighborhood of the optimum size.

[7] The reader would note that, since the giantsteps are made initially and stored, it would be a waste of resources to make another set of giantsteps for every "miss" in the search interval

[8] Reduction in space does not imply that the size of the interval has become less, but that many of the integers in the interval can be excluded from the potential candidates for group order

Algorithm 2: Compute Jacobian order: Cartier Manin, Schoof and Primorial BSGS

```
Data: $J$, $w$, $w_l$, $w_u$, $m$, $N_r$, $p$, $P$, $\alpha$ - random element of $J$
Result: $\#J$ - Order of the jacobian
begin
    $N_r \longleftarrow CartierManinOn(J, m)$;
    $p, P \longleftarrow BestPrimorial(w, N_r)$;
    $T \longleftarrow BestLimitBabies(w, N_r, P)$;
    $\beta \longleftarrow DivisorWithSpecialOrder(\alpha, p)$;
    * Find the optimal Giant Size *;
    while True do
        $lpm \longleftarrow lcm(pr, m)$;
        $g \longleftarrow$ Multiple of lpm after$\sqrt{w}$;
        $opt \longleftarrow \sqrt{\frac{w \cdot pr \cdot m}{\varphi pr}}$;
        if $g$ too larger than $opt$ then
            Remove the larger primes one by one;
        else
            Be satisfied with it, BREAK;
            break;
        end
    end
    * Generating giants steps *;
    $Giants \longleftarrow \{g_i = g \cdot i \mid w_l \leq g_i \leq w_u;\ i \in \mathbb{Z}\}$;
    * Generating the babysteps *;
    $Primes = \{$All primes in$pr\}$;
    foreach $i < g$ do
        if $i \perp pr$ and $i = nmodm$ then
            $Result \longleftarrow CheckBabyStep(i, Giants)$;
            if Result then
                return $Result$
            end
        end
    end
    foreach $p_i$ in $Primes$ do
        $pr_x \longleftarrow \frac{pr}{p_i}$;
        foreach $i < g$ do
            if $i$ not checked yet & $i \perp pr_x$ & $i = nmodm$ then
                $Result \longleftarrow CheckBabyStep(i, Giants)$;
                if Result then
                    return $Result$
                end
            end
        end
    end
end
```

Loop Hole: The while loop could be infinite.

The algorithm has a while loop which seemingly runs into an infinite loop. But here is the proof that it will terminate, where the worst case is P being reduced to 1.
As the primes are removed one by one, the size of the suggested value would get closer to the optimal value and finally up to the next multiple of m larger than the optimal value which has to be set as the safe-neighborhood for satisfying the condition to break out of the loop.

Generation of Giantsteps.

This section of the algorithm is the easiest but the more space consuming one. Once the size of the giantstep has been fixed up on, the algorithm can compute the giantsteps between the Weil-Interval.

It is worth noting that unlike the traditional babystep-giantstep algorithm, giants are generated and stored for further searches. This of course comes with the disadvantages of having multiplications with larger multipliers.

All the giantsteps are multiples of P and also m.

Generation of Babysteps.

The babystep generation is worth noting because of the different sweeps through the giantstep space for completely generating them, instead of doing them in a single sweep.

In the initial sweep, the steps which are coprime to the Primorial are chosen to be generated and compared against the giantsteps in the store. This is the equivalent of searching for the the order of α, the random element inside the Weil-interval. To be precise, the babysteps which are of the form $l \cdot m - n$, which are coprime to the Primorial P are computed. Since the giants are all multiples of P and m, if the order happens to be in Weil-interval, there will be a match between one of the babysteps and one of the giantsteps.

It is worth observing that, a match in the first sweep means the order of the group is the same as the order of α, the element in question.

Once all the potential candidates, which are coprime-multiples of α, are generated and checked against the giantsteps, and if it leads to no match, the implication is that the order of the element is a factor of the group order.

Then the algorithm proceeds to explore the intervals where the order of the element is in the intervals $\frac{w}{2}, \frac{w}{3}, \ldots$

This is achieved by taking off the primes from the Primorial one by one and generating the babysteps which are coprime to the new Primorial, and which are not generated in the earlier sweeps.

This process continues until the Primorial becomes one - which is falling back to the normal babystep-giantstep algorithm.

4.5 Proof

The proof of the algorithm explained above needs two sections: (i) It terminates and (ii) It finds the right solution.

4.5.1 The algorithm terminates

The algorithm has three loops of which one is seemingly an infinite loop. In the above section, we saw that the loop which seems an infinite loop does terminate in all cases.

The second loop in the algorithm is for the generation of the giantsteps. It executes exactly $\frac{w}{g}$, where w is the size of the Weil-interval and g is the size of the optimal giantstep-size determined.

The third section of the algorithm makes l sweeps through the whole interval $[1, g]$ where g is the size of the optimal giantstep-size, which guarantees that the algorithm terminates. (Where l is the number of primes in the Primorial)

4.5.2 It gives the deemed result

To prove that the algorithm gives the deemed result, two points are to be noted.

- Irrespective of the value of the optimal giantstep-size, the algorithm creates $\frac{w}{g}$ giantsteps in the Weil-interval.
- Through the l sweeps through the interval $[1, g]$, the algorithm generates all the babysteps which satisfy $\alpha \times (k \cdot m - n)$.

Considering that all the babysteps up to g are checked against all the giantsteps which are separated by g in the Weil-interval, a match is sure to occur between one of the babysteps and one of the giantsteps.

The details of the proof is the same as in section 3.1.1.

4.6 Cost analysis

The cost of the algorithm is the number of group operations required for computing the size of the group order. The number of operations required is the sum of the number of giantsteps and of babysteps made during the computation.

Let the size of the giantstep be:

$$g = c \cdot \sqrt{\frac{w \cdot P \cdot m}{\varphi(P)}}$$

Where c is the constant which decides the optimum value.

This implies that the number of giantsteps be:

$$\#g = \frac{w}{g} = \frac{1}{c} \cdot \sqrt{\frac{w \cdot \varphi(P)}{P \cdot m}}$$

For the number of babysteps, we could consider the best and worst cases.

Best Case

In the best case, the order of the element is a non-trivial factor of the group order, which means the first sweep of babysteps would get a match from the giantsteps.

Which means, the number of operations $= \frac{g \cdot \varphi(P)}{P}$.

Therefore, the number of operations would be of the order of $O(\sqrt{\frac{w \cdot \varphi(P)}{P \cdot m}})$.

Worst Case

In the worst case, the order of the element is a very small factor of the group order, which means that the number of sweeps of babysteps is l, the number of primes in the Primorial.

This implies that the number of operations $= g$ which is of the order of $O(c \cdot \sqrt{\frac{w \cdot P \cdot m}{\varphi(P)}})$. The constant c could be leveraged to adjust the value of g.

A value $c = \frac{\varphi(P)}{P}$ makes sure that the worst case is always better than the Schoof-CartierManin method without the use of Primorials.

4.6.1 In practice

In the case of jacobians and their elements, an average case could not make sense. But from the above discussion confirms that the algorithm2 has a better running time than the Schoof-CartierManin algorithm. Please see [GH00].

Where as the best case saves up to 60% of the work (For details of percentage savings, please refer to table 4.1).

In the next section, the above algorithm will be applied in the cryptographic setting where one could make sure that the best case of the algorithm would only be put to use.

4.6.2 In theory

The improvement which algorithm promises is in terms of the order P, but not in terms of the size of the field on which the hyperelliptic curve is defined. Even though the size of the Primorial depends on the size of q, it is very weakly related and is not strongly coupled to make a polynomial difference in the cost.

At the same time, the introduction of the Primorial method provides with an improvement factor of $\sqrt{\frac{P}{\varphi(P)}}$, which ranges from 2 to 3. To know what percentage of change it would make, please see table 4.1.

4.7 Cryptographic restrictions

All through the previous sections in this chapter, we examined the different cases of using the available information about the jacobian order, in finding out the order of any arbitrary hyperelliptic curve.

In the last section (Section 4.4), it was even shown that the combination of the available information can be leveraged to design an algorithm (Algorithm 2) which substantially reduces the cost.

In this section, we will look at the requirements of a cryptosystem, for guaranteeing security on the forms of protocols/methods it offers. In section 2.7, we saw that the jacobian offers no sub-exponential algorithms for the solution of DLP defined on it. This statement is true only for those curves which are of genus less than 3. Please see [SV].

Even though there aren't any sub-exponential algorithms, there are some attacks which have been developed for breaking the cryptosystems based on hyperelliptic curves. Here in the rest of the section, we shall see the effective ones among them, which are related to the order of the group and we shall see what kind of countermeasure is applied to get around these problems.

Pohlig-Hellman

If the group order $\#G$ of G factors as $\Pi {r_i}^{e_i}$ then it is possible to reduce the DLP in G to a DLP in subgroups of order r_i, see [PH78]. So if r is the largest prime divisor of $\#G$, then the DLP in G is as hard as the DLP in the subgroup of order r. For this reason it is important to choose G such that its order is almost prime, i.e. such that $\frac{\#G}{r}$ is small. A typical choice is to require that this quotient is ≤ 4. Ideally, one would like $\#G$ to be prime, but it is not always possible to achieve that. For example, an elliptic curve, or hyperelliptic curve of genus 2 over a field F_{2^l}, that is not super-singular always yields a group of even order.

MOV and Frey-Rück Attacks

Let r be the largest prime divisor of the group order $\#G$. Let k be the smallest positive integer such that $r \mid q^k - 1$. Then there is a computable injective group homomorphism from the order-r subgroup of G to ${F^*}_{q^k}$, see [9] and [31]. If k is too small, then one can solve the DLP in G by first mapping it to ${F^*}_{q^k}$, and use index calculus there. So one should avoid groups for which k is small. Typically, $k > 20$ is a safe choice. A random curve is very unlikely to yield a group for which k is small. There is however a special class of curves, super-singular curves, for which k is always small.

Anomalous curves

If the largest prime divisor r of $\#G$ is equal to the characteristic of F_q, then one can transform the DLP to a DLP in the additive group of F_q, where it is trivial to solve. See [42] and [38]. So this should be avoided.

Weil restriction and cover attacks

Let C be an elliptic curve or hyperelliptic curve of genus g defined over an extension field F_{q^e}. Let G be the group $C(F_q)$ if C is elliptic, or $J_C(F_q)$ if C is hyperelliptic. Then sometimes it is possible to find a curve X defined over F_q such that there is a homomorphism from G to $J_X(F_q)$ that transfers the DLP from G to $J_X(F_q)$. If the genus of such an X is not much bigger than $e \cdot g$, then index calculus methods on X enable one to compute the DLP faster than with Pollard's rho method. The original idea behind this construction goes back to Frey, and it has been applied successfully to attack several curves, for the first time in [10].

4.7.1 Good curves

To summarise which curves are safe to use, here is the list of the required properties. Let q be either a prime, or $q = 2^l$, with l prime. Let C be an elliptic curve or a hyperelliptic curve of genus 2 over F_q. Let G be the associated group, i.e. $G = C(F_q)$ or $G = J_C(F_q)$. Let r be the largest prime divisor of $\#G$. Then we require

1. r does not divide q
2. If $k > 0$ is the smallest integer such that $r \mid q^k - 1$ then $k > 20$.
3. $r > 2^{160}$
4. $\frac{\#G}{r} \leq 4$.

In order to check these requirements, one needs to be able to determine $\#G$. For the employment of a cryptographic system, one only has to find one good curve. The point counting only needs to be done in the initial set-up of the system. Once one has a safe curve, it can be used as long as no attack is known on that specific curve. The curve, and the size of the group are not part of the keys of the cryptosystem, so even if a number of secret keys are revealed, it does not jeopardise other secret keys. Therefore, one can use standard curves, for which the cardinality of G has already been determined.

4.8 Applying the restrictions

In the last section we saw the restrictions which are put on the order of a curve, if it has to be suitable to be used for building a cryptosystem. Looking in that light, there is one of the restrictions/conditions which can be used directly in improving the order counting algorithm we saw earlier.

Let us recap the restriction/condition 3: The largest divisor of the group order has to be larger than 2^{160}. This extra knowledge can be made use by searching only above this limit.

In the algorithm 2, the search is initially for the order of the random element, α which is the same as the order of the group. Once the whole Weil-interval is searched for a value satisfying that, and if it turns out to be failure, then the algorithm knows that the order of the element is a non-trivial factor of the order of the group and proceeds to search for $o(\alpha)$ in the interval $\frac{w}{2}, \frac{w}{3}, \ldots$, by removing the prime numbers one by one from the Primorial.

But with the new light of restriction, the algorithm could search in the Weil-interval and its subsequent fractions, only until the Weil-interval fraction is still above 2^{160}.

4.8.1 Some facts

A hyperelliptic curve which could be used for cryptographic purposes satisfies the following.

- The field $\mathbb{F}_q$ over which the curve is defined is of the order $\sim 2^{80}$
- The order of the jacobian would be closer to $\sim 2^{160}$
- The largest prime divisor of $\#J$ has to be larger than $\sim 2^{160}$
- Using the Primorial method requires the Primorial to contain primes up to $67 < 2^6$ (please see 4.1)

If the choice of $\mathbb{F}_q$ is wisely made, the algorithm from the previous section could be modified to leverage the facts mentioned above, to decide whether a hyperelliptic curve is safe for cryptography or not, and if safe, to compute the order of the jacobian.

The tailor-made jacobian counting algorithm can be seen in Algorithm 3.

Details of Sections of the algorithm

The algorithm 3 is the same as the algorithm 2 except for the generation of babysteps. As in the previous method, the babysteps which are all of the form $k \cdot m - n$ for k a positive integer and the steps being coprime to the Primorial are generated first.

In the next step, where each prime number is taken away from the Primorial, so as to check for the order of the element in lower intervals, the algorithm checks whether the upper limit of the interval is smaller than the restriction set by security measures.

And if it is, the algorithm can specify that the curve is not suited for cryptography and quits.

4.8.2 Analysis and Advantages

Instead of making l sweeps through the giantstep size, the new algorithm makes only k sweeps until the k^{th} prime divides the Weil-interval to get an interval which has upper limit smaller than 2^{160}.

By wisely choosing the field over which the curve is defined, and also choosing the optimal giantstep size as $O(c \cdot \sqrt{\frac{w \cdot P \cdot m}{\varphi(P)}})$, the number of giantsteps would be:

$$\#g = \frac{w}{g} = \frac{1}{c} \cdot \sqrt{\frac{w \cdot \varphi(P)}{P \cdot m}}$$

Algorithm 3: Compute Jacobian order: Cartier Manin, Schoof and Primorial BSGS - Tailor made for hyperelliptic curves for cryptographic purposes

```
Data: $J$, $w$, $w_l$, $w_u$, $m$, $P$, $N_r$, $\alpha$ - random element of $J$
Result: $\#J$ - Order of the jacobian
begin
    $N_r \longleftarrow CartierManinOn(J, m)$;
    $p, P \longleftarrow BestPrimorial(w, N_r)$;
    $T \longleftarrow BestLimitBabies(w, N_r, P)$;
    $\beta \longleftarrow DivisorWithSpecialOrder(\alpha, p)$;
    * Find the optimal Giant Size: See 2*;
    * Generating giants steps: see 2*;
    * Generating the babysteps *;
    $Primes = \{$All primes in$pr\}$;
    foreach $i < g$ do
        if $i \perp pr$ and $i = nmodm$ then
            $Result \longleftarrow CheckBabyStep(i, Giants)$;
            if Result then
                return $Result$
            end
        end
    end
    foreach $p_i$ in $Primes$ do
        if $\frac{w_u}{p_i} lt2^{160}$ then
            The curve is not suited for cryptography;
            Exit the algorithm and choose another Curve;
        end
        $pr_x \longleftarrow \frac{pr}{p_i}$;
        foreach $i < g$ do
            if $i$ not checked yet & $i \perp pr_x$ & $i = nmodm$ then
                $Result \longleftarrow CheckBabyStep(i, Giants)$;
                if Result then
                    return $Result$
                end
            end
        end
    end
end
```

and the number of babysteps would be:

$$\#b = k \cdot g \cdot \frac{\varphi(P)}{P \cdot m} = c_1 \cdot \sqrt{\frac{w \cdot \varphi(P)}{P \cdot m}}$$

And the cost of the algorithm in terms of the operations needed is $\#b + \#g = O(\sqrt{\frac{w \cdot \varphi(P)}{P \cdot m}})$.

4.9 Summary

From the beginning of the chapter, where we started with the naive method of counting in the jacobian of a hyperelliptic curve, we have developed a solution which reduces the work to a fraction of the search needed by the naive solution.

In our pursuit to better solutions, we did see some solutions which pointed that the Primorial method cannot be better than the Weil-interval when there are no other extra information available.

But, the combination of all the information, has made it possible to devise a method which is better than the solutions which we saw in the chapter 3 "State of the Art", where the best solutions which exist till date are mentioned.

Also, by taking into consideration, the restrictions on a group for guaranteeing security, we designed a tailor-made method for hyperelliptic curves of genus two, for deciding whether the curve is good for computation or not. That solution has a silver lining that, the solution gives the jacobian order, only if the curve fits the required prerequisites and a large portion of the unnecessary computation is avoided if the curve is not good enough.

After seeing the theoretical improvement of $\sqrt{\frac{P}{\varphi(P)}}$, the details of implementation and results are given in the next chapter.

Chapter 5

Implementation, Results and Future Prospects

> "This principle is so perfectly general that,
> no particular application of it is possilbe"
>
> - George Polya.

The algorithms which were discussed in the chapter 4 were implemented and were run against the previously-existing algorithms.

In this chapter, we will see the details of the implementation and the results obtained. We also would look into the future of the research in the direction we are following.

5.1 Implementation

Even though the chapter 4 talks only about the search algorithms, the implementation of the jacobian counting system required more than just the search algorithms. The complete implementation was divided into two phases and were run in two different locations owing to some technical restrictions.

5.1.1 Generating the Curve

The initial phase of the curve consisted of generating enough random curves on random fields. This phase also consisted of computing the Cartier Manin and Schoof numbers of the curve.

Since MAGMA [MAG] has many of the routines needed for the aforementioned operations, this part of the implementation was done with Magma on a Sun Sparc machine with shared 16GB memory.

Even though the first phase has no direct connection with the solutions we discussed earlier, this was an integral part of the project.

Details of implementation (including MAGMA code and BASH scripts) are available at [Sad].

5.1.2 Counting Algorithms

For the implementation of the counting algorithms, we used SAGE [Ste07] and Python^{TM}. Sage provides many useful methods for hyperelliptic curve implementation. The inclination towards open source software acted as an additional reason behind the implementation of second phase in the open-source mathematical software.

The details of implementation, including the scripts and SAGE code, are available at [Sad].

5.2 Results: Old Vs. New - Comparison Tables

As a part of testing the implementation, we generated a collection of hyperelliptic curves (randomly). Potential (random) prime numbers in a valid range[1] was chosen and depending on the size (bit size) of the prime numbers, a suitable extension was selected and the curves were built with random coefficients.

The details of these curves are given in the Curve Database: B.1, B.2, B.3, B.4, B.5.

In this section, we provide the reader with the details and results of running of the best algorithm from chapter 4 and the algorithms which were designed in the previous chapter.

The results are widely categorised into two parts -
(i) Based on the advantage of the new methods over the old method.
(ii) Based on the size of the prime numbers and extensions.

If the reader wishes to have more information about the curve, she is kindly requested to look up the serial number (Column #) of the curve in the Curve Database: B.1, B.2, B.3, B.4, B.5.

[1] The characteristic of the base field has to be less that 10^6 for viable computation of the Cartier Manin operator

5.2.1 Most of the curves

The table 5.1 compares the running times of the CMSchoof algorithm and the algorithm 2 explained in the previous chapter.

The columns of the table are:

- The serial number of the curve
- The characteristic of the field
- The time required for the CMSchoof algorithm in section 3.2.3
- The time taken by the Primorial method 2
- The order of the curve computed by the two methods

The reader could see that the total time (last row) taken by the Primorial method is only about 60% of the time required by the CMSchoof method from section 3.2.3.

In table 5.2, the same computation of the group order is given with the break down of number of babysteps and giantsteps which were computed for the CMSchoof and Primorial methods respectively[2].

As in the previous tables, the last row show the total of the computations needed for all the curves in the table. The total computation required by the Primorial-Schoof combination 2 is only 75% of the methods without the Primorial advantage.

5.2.2 The curves which faired well

With some curves, the new method was so fast that we considered it worth mentioning here. The tables 5.3, 5.4 show the time and operation comparisons between the two different methods which are the main theme of this chapter.

5.2.3 Curves on non-prime fields

In the initial parts of this section, we saw the comparison of algorithms when they were applied on curves which were defined on prime-fields.

In the tables 5.5, 5.6, 5.7, 5.8, the reader could see the comparison of algorithms when the curves are of fields of prime power.

[2] It is to be observed that the giantsteps and babysteps are interchanged for the Primorial-Schoof combination

	Characteristic	CM-Schoof	Primorial	Group Order
#		(in seconds)	(in seconds)	
4	950611	1.92	0.69	905310495679
7	914701	1.77	0.74	836139440502
11	949733	1.27	0.81	900165652552
12	915251	1.77	0.74	837793416944
16	931597	1.86	0.70	869982361562
20	943153	0.95	0.90	890142900928
36	935971	1.74	0.71	875243674426
41	910471	1.32	0.72	829338545904
43	941557	1.31	0.69	886140139008
48	976471	1.94	1.13	954445984839
49	910421	1.32	1.22	829112748104
52	950329	1.30	0.64	902394479472
54	953773	0.98	0.99	908944982496
57	983513	1.30	0.85	967276075676
66	907363	1.76	0.78	823137443386
67	910177	1.90	1.36	830364235233
71	926701	1.30	0.96	857654014544
78	994927	1.32	0.88	988930781928
80	960931	1.35	1.21	925158161432
83	968467	1.30	0.56	936743525792
85	969421	1.82	0.71	939710333262
88	970859	1.01	0.54	943450974432
89	944491	1.31	0.74	891264709896
91	961871	1.29	0.86	924578774816
101	976991	1.31	0.64	953972995396
104	995833	1.88	1.22	992188599049
105	930779	1.36	1.07	867626285192
106	945289	1.34	0.55	894471514876
107	979651	1.89	1.02	960456115583
109	980887	1.34	0.60	962324055304
113	978683	1.89	0.80	959357530602
114	994501	1.86	1.10	988997216327
115	968729	1.85	1.00	939179183518
116	994751	1.86	1.26	990686610702
117	931003	1.31	1.22	867403320348
118	916291	1.81	1.09	840590700384
120	989557	0.99	0.54	978269804256
		55.8	32.3	

Table 5.1: Comparison old vs. new methods: Time and Group Order

#	Base	Ext	Schoof Baby	Primorial Giant	Schoof Giants	Primorial Baby	Schoof Total	Primorial Total
4	950611	1	37	1	26	38	63	39
7	914701	1	36	3	15	37	51	40
11	949733	1	26	8	7	19	33	27
12	915251	1	36	3	19	37	55	40
16	931597	1	36	2	29	37	65	39
20	943153	1	18	11	11	10	29	21
36	935971	1	36	2	14	37	50	39
41	910471	1	26	6	14	19	40	25
43	941557	1	26	5	12	19	38	24
48	976471	1	37	11	23	38	60	49
49	910421	1	26	17	14	19	40	36
52	950329	1	26	4	11	19	37	23
54	953773	1	19	13	7	10	26	23
57	983513	1	26	9	13	19	39	28
66	907363	1	36	4	17	37	53	41
67	910177	1	36	17	28	37	64	54
71	926701	1	26	11	9	19	35	30
78	994927	1	26	9	10	19	36	28
80	960931	1	26	17	19	19	45	36
83	968467	1	26	2	9	19	35	21
85	969421	1	37	2	18	38	55	40
88	970859	1	19	3	11	10	30	13
89	944491	1	26	6	10	19	36	25
91	961871	1	26	9	11	19	37	28
101	976991	1	26	4	11	19	37	23
104	995833	1	37	13	21	39	58	52
105	930779	1	26	14	17	19	43	33
106	945289	1	26	2	16	19	42	21
107	979651	1	37	9	22	38	59	47
109	980887	1	26	3	14	19	40	22
113	978683	1	37	4	25	38	62	42
114	994501	1	37	11	18	38	55	49
115	968729	1	37	9	22	38	59	47
116	994751	1	37	15	24	38	61	53
117	931003	1	26	17	15	19	41	36
118	916291	1	36	11	23	37	59	48
120	989557	1	19	3	7	10	26	13
			1102	290	592	965	1694	1255

Table 5.2: Comparison old vs. new methods: Number of operations

	Characteristic	CM-Schoof	Primorial	Group Order
#		(in seconds)	(in seconds)	
101	976991	1.31	0.64	953972995396
52	950329	1.30	0.64	902394479472
43	941557	1.31	0.69	886140139008
120	989557	0.99	0.54	978269804256
4	950611	1.92	0.69	905310495679
106	945289	1.34	0.55	894471514876
41	910471	1.32	0.72	829338545904
109	980887	1.34	0.60	962324055304
89	944491	1.31	0.74	891264709896
83	968467	1.30	0.56	936743525792
16	931597	1.86	0.70	869982361562
88	970859	1.01	0.54	943450974432
113	978683	1.89	0.80	959357530602

Table 5.3: Comparison old vs. new: Time and Group Order (Better Curves)

			Schoof	Primorial	Schoof	Primorial	Schoof	Primorial
#	Base	Ext	Baby	Giant	Giants	Baby	Total	Total
101	976991	1	26	4	11	19	37	23
52	950329	1	26	4	11	19	37	23
43	941557	1	26	5	12	19	38	24
106	945289	1	26	2	16	19	42	21
120	989557	1	19	3	7	10	26	13
4	950611	1	37	1	26	38	63	39
41	910471	1	26	6	14	19	40	25
109	980887	1	26	3	14	19	40	22
89	944491	1	26	6	10	19	36	25
83	968467	1	26	2	9	19	35	21
16	931597	1	36	2	29	37	65	39
88	970859	1	19	3	11	10	30	13
113	978683	1	37	4	25	38	62	42
			356	45	195	285	551	330

Table 5.4: Comparison old vs. new: Number of operations(Better Curves)

#	Characteristic	CM-Schoof (in seconds)	Primorial (in seconds)	Group Order
124	1009	1.03	4.18	1040604009999
134	647	0.93	3.84	176319369215
140	691	1.00	3.88	229310730495

Table 5.5: Curves : Time and Group Order (Better Curves)

#	Base	Ext	Schoof Baby	Pri. Giant	Schoof Giants	Pri. Baby	Pri. Restr.	Schoof Total	Pri. Total	Pri.R Total
134	647	2	1295	1451	1294	1156	165	2589	2607	1616
124	1009	2	1427	1764	1427	1156	165	2854	2920	1929
140	691	2	978	828	977	1156	165	1955	1984	993
			3700	4043	3698	3468	495	7398	7511	4538

Table 5.6: Comparison old vs. new: Number of operations(Better Curves)

In the tables where the operations are compared for curves on non-prime fields, it is seen that the operations needed by Primorial methods are more than the naive CMSchoof method without Primorial.
But the tables which compare the non-prime based curves have two extra columns which specify, how the algorithm 3 comes to help from saving the extra computation by making use of the known cryptographic restrictions.
It is worth noting that the saving done by the restricted (Pri.R) algorithm is about 50% of the CMSchoof algorithm which is the best one available.

#	Characteristic	CM-Schoof (in seconds)	Primorial (in seconds)	Group Order
136	197	1.32	5.90	58536332836258
130	109	1.15	5.14	1683002823435
123	211	1.21	5.93	88361163800489
143	149	1.18	5.50	10966612124981

Table 5.7: Comparison old vs. new: Time and Group Order (Better Curves)

#	Base	Ext	Schoof Baby	Pri. Giant	Schoof Giants	Pri. Baby	Pri. Restr.	Schoof Total	Pri. Total	Pri.R Total
136	197	3	5981	7148	5981	5006	715	11962	12154	7863
130	109	3	4246	3603	4246	5006	715	8492	8609	4318
123	211	3	6745	9089	6744	5006	715	13489	14095	9804
143	149	3	5189	5380	5188	5006	715	10377	10386	6095
			22161	25220	22159	20024	2860	44320	45244	28080

Table 5.8: Comparison old vs. new: Number of operations(Better Curves)

5.3 Summary and Future Prospects

In the last two chapters, we saw the improvements which are offered by the combination of the traditional methods and the primorial counting method.

Even with the improvements, the counting in the jacobian of hyperelliptic curves are still $O(N^{\frac{1}{2}})$ where N is the size of the Weil interval, where $N = q^{\frac{3}{2}}$ and $q = 2^n$ where q is the size of the field on which the curve is defined and n is the size in number of bits. The algorithms still need exponential space and time for the computation in the jacobian.

But on the happy note, even when the theoretical improvements aren't too satisfying, on the practical side, we saw that the computation requirement reduces up to 60% of the original requirement.

5.3.1 Bringing in Birthday Paradox

All the algorithms discussed in this thesis were focusing on the BSGS version of the counting. The same methods can be used in a setting where the theme is Birthday paradox algorithms (Please see 3.1.2).

Since BSGS is better suited for the curves (and the fields) we have discussed in this thesis, Birthday paradox algorithms weren't explored much. But for curves which need much more computation, when the memory-space available is limited, Birthday paradox algorithms would make more sense and should be explored.

5.3.2 Reverse Engineering

For the purpose of guaranteeing security, counting is not the only way. A different approach is to construct curves using the CM method[AM93]. Even though nowadays counting points is feasible for elliptic curves of cryptographic size, this method is still of interest, e.g. it is the main way of constructing non-supersingular curves with low embedding degree which can be useful in pairing based protocols if one wants to avoid supersingular curves for some reason or if a larger embedding degree is desired.

While our interests are in creating curves suitable for cryptography, the future research could follow this lead too. The readers who are interested are encouraged to peruse [Wen03] and [MKS].

Appendix A

Algebra Refresher

This chapter has been provided as an easy reference to the terms used in the thesis. For detailed references for group theory Herstein [Her86, Her75] and for field theory Roman [Rom] are recommended. Set theory is assumed to be known.

A.1 Groups

Definition 43 (Group)**.** *A nonempty set G is said to be a group if in G there is defined an operation $\oplus$ such that it satisfies the following.*

1. *Closure: $a, b \in G$ implies $a \oplus b \in G$*

2. *Associativity: Given $a, b, c \in G$ then $a \oplus (b \oplus c) = (a \oplus b) \oplus c$*

3. *Existence of Identity: There exists a special element $\epsilon \in G$ such that $a \oplus \epsilon = \epsilon \oplus a = a$ for all $a \in G$. ϵ is called the* identity *element of G.*

4. *Inverse Element: For every $a \in G$ there exists an element $b \in G$ such that $a \oplus b = b \oplus a = \epsilon$. (We write b as a^{-1} and call it the* inverse *of a in G)*

These four conditions are called *group axioms*.

We usually represent a group by $(G, *)$; where G is the group and $*$ is the group operation.

Example 44. *$(\mathbb{Z}, +)$ is a group where $\mathbb{Z}$ is the integers and $+$ is the ordinary addition.*

Definition 45 (Order)**.** *The number of elements of a group is called the* order *of the group. If the order is finite, the group is said to be a finite group. Order of a group G is denoted as $\|G\|$.*

But, for an element $a \in G$, the least positive number m such that $a^m = \epsilon$ is called the order *of a in G, represented as $o(a)$.*

Definition 46 (Abelian/Commutative Group). *A group is an abelian*[1] *group if $a \oplus b = b \oplus a$ for all $a, b \in G$.*

Definition 47 (Subgroup). *A nonempty subset H of a group G is called a subgroup of G, if relative to the operation in G, H itself satisfies all the group axioms.*

Example 48. *H = {Even numbers} $\subset \mathbb{Z}$ $(H, +)$ is a group under ordinary addition.*

Definition 49. *A relation $\sim$ of a set G is called an equivalence relation if for all $a, b, c \in G$:*

1. *$a \sim a$.(Reflexive)*

2. *$a \sim b$ implies $b \sim a$.(Symmetric)*

3. *$a \sim b$ and $b \sim c$ implies $a \sim c$.(Transitive)*

Definition 50 (Equivalence Class). *If $\sim$ is an equivalence relation on G, then $[a]$, the equivalence class of a is defined by $[a] = \{b \in G \mid b \sim a\}$.*

Lemma 51. *If $\sim$ is an equivalence relation on G, then*

1. *G = $\bigcup_{a \in G}[a]$*

2. *$[a] \cap [b] \neq \phi$ equivalent to $[a] = [b]$*

Definition 52 (Coset). *For a group G, for which H is a subgroup, and an element of G, then $gH = \{gh \mid h$ an element of $H\}$ is called the left coset of H in G, $Hg = \{hg \mid h$ an element of $H\}$ is called the right coset of H in G.*

Theorem 53 (Lagrange's[2] Theorem). *If G is a finite group and H is a subgroup of G, then $\|H\|$ divides $\|G\|$.*

Fact 54. *If G is finite and $a \in G$, then $o(a) \mid \|G\|$. We can see this directly from definition 45 and Theorem 53.*

Definition 55 (Homomorphism). *Let G and G′ be two groups then a mapping $\phi : G \rightarrow G'$ is called a homomorphism if $\phi(ab) = \phi(a)\phi(b)$ for all $a, b \in G$.*

If the homomorphism is a bijection, then it is called an isomorphism.

Definition 56 (Kernel). *If ϕ is a homomorphism from G to G', then the kernel of ϕ is defined by $k\phi = \{a \in G \mid \phi(a) = \epsilon'\}$, and ϵ' is the identity element of G'.*

[1] The name comes from the great Norwegian mathematician Niels Henrik Abel

[2] The name of the theorem comes from famous mathematician J L Lagrange

Definition 57. *Image of a subgroup H of G under ϕ is defined as $Im(H) = \{b \in G' \mid \exists a \in G$ such that $\phi(a) = b\}$.*

Definition 58 (Normal Subgroup)**.** *A subgroup N of G is said to be normal subgroup iff $a^{-1}Na \subset N$ for every $a \in G$.*

We denote this by $N \lhd G$.

Definition 59 (Factor group)**.** *If $N \lhd G$ and we define $a \sim b$ iff $ab^{-1} \in N$, we get a new set of equivalence classes. This set of equivalence classes is called the* factor group or quotient group *of G by N.*

We have a symbol for this factor group: G/N.

Theorem 60. *If $N \lhd G$, and*

$G/N = \{[a] \mid a \in G\} = \{Na \mid a \in G\}$

Then G/N is a group relative to $[a][b] = [ab]$.

Theorem 61 (Homomorphism)**.** *Let $f : G \to G'$ be a homomorphism and N be a sub group of G with $N \subseteq ker(f)$. We then have a unique homomorphism $h : G/N \to G'$ such that $h \circ \phi = f$.*

i.e.,

$$\begin{tikzcd}
G \arrow[r, "f"] \arrow[d, "\phi"'] & G' \\
G/N \arrow[ur, "h"] &
\end{tikzcd}$$

Now we have come to a point where we can discuss the other three homomorphism theorems. I will state them one by one. I shall not give the proofs. For the proofs the reader can refer to any of these books by Herstein [Her86, Her75].

Theorem 62 (First Homomorphism Theorem)**.** *Let ϕ be a homomorphism of G* onto *G' with kernel K. Then $G' \simeq G/K$, the isomorphism between these effected by the map.*

$$\psi : G/K \to G'$$

defined by $\psi(Ka) = \phi(a)$.

Theorem 63 (Second Homomorphism Theorem)**.** *Let the map $\phi : G \to G'$ be a homomorphism of G* onto *G' with kernel K. If H' is a subgroup of G' and if*

$$H = \{a \in G \mid \phi(a) \in H'\}$$

Then H is a subgroup of G, $H \supset K$ and $H/K \simeq H'$. Finally, if $H' \lhd G'$ then $H \lhd G$.

Theorem 64 (Third Homomorphism Theorem). *If the map $\phi : G \to G'$ is a homomorphism of G* onto *G' with kernel K, then if $N' \triangleleft G'$ and $N = \{a \in G \mid \phi(a) \in N'\}$, we conclude that $G/N \simeq G'/N'$ and $G/N \simeq (G/K)/(N/K)$*

A.2 Rings and Fields

Definition 65 (Ring). *Let R be a set on which two binary operations are defined, called addition and multiplication, and denoted by $+$ and $\cdot$. Then R is called a* ring *with respect to these operations if the following properties hold:*

1. *Closure: If $a, b \in R$, then the sum $a + b$ and the product $a \cdot b$ are uniquely defined and belong to R.*
2. *Associative law: For all $a, b, c \in R$, $a + (b + c) = (a + b) + c$ and $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.*
3. *Commutative law: For all $a, b \in R$, $a + b = b + a$.*
4. *Distributive law: For all $a, b, c \in R$, $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(a + b) \cdot c = a \cdot c + b \cdot c$.*
5. *Additive identity: The set R contains an additive identity element, denoted by 0, such that for all $a \in R$, $a + 0 = a$ and $0 + a = a$.*
6. *Additive inverse: For each $a \in R$, there exists an element $b \in R$ such that $a + b = 0$ and $b + a = 0$. The element b is called the additive inverse of a in R, and denoted by $-a$.*

If $\cdot$ is also commutative, the ring is called a commutative ring. Otherwise an associative ring.

Definition 66 (Integral Domain). *A commutative ring R is called an Integral Domain if $a \cdot b = 0$ implies $a = 0$ or $b = 0$. In other words, an integral domain is a commutative ring with NO zero divisors.*

An element $0 \neq a \in R$ is called a zero divisors of there exists an element $b \neq 0 \in R$ such that $a \cdot b = 0$.

If there is an element $1 \in R$ such that for all $a \in R$, $1 \cdot a = a \cdot 1 = a$, we call R to be a ring with unit. It is necessary that $1 \neq 0$.

Definition 67 (Ideal). *For a group we have subgroup. The same way, for a ring we have an Ideal. It is defined as below.*

Let R be a ring, a non-empty subset I of R is called an ideal (two sided) if

1. *I is an additive subgroup of R.*

2. *Given $r \in R$, $a \in I$, then $ra \in I$ and $ar \in I$.*

Definition 68 (Homomorphism)**.** *Similar to group homomorphisms, we have homomorphisms in rings also.*

A mapping $\phi : R \rightarrow R'$ of the ring R into the ring R' is a homomorphism if

1. $\phi\,(a+b) = \phi(a) + \phi(b)$

2. $\phi(ab) = \phi(a)\phi(b)$

Definition 69 (Field)**.** *A ring R is called a field iff the following conditions are satisfied.*

1. *R is a ring with unit.*

2. *For all $a \neq 0 \in R$ there exists a $b \in R$ such that $a \cdot b = b \cdot a = 1$. This b is denoted as b^{-1}*

3. *R is commutative.*

Or, we can say the same in other words.

1. *$(R, +)$ is a commutative ring.*

2. *$(R^*, \cdot)$ is a commutative ring.($R^* = R \backslash \{0\}$).*

3. *$\cdot$ is distributive over $+$.*

A.3 Extension Field

Definition 70 (sub-field, extension field)**.** *If a field F is a subset of another field K with respect to the same operations in K, then F is called a sub-field of K. And, K is an extension field of F.*

Definition 71 (algebraic, minimal polynomial)**.** *Let K be an extension field of F. An element $a \in K$ is said to be algebraic over F if there exists a polynomial $f \in F[x]$ with $f(a) = 0$. The monic polynomial with minimal degree so that $f(a) = 0$ is called the minimal polynomial of a over F and denoted by f^a_{min}.*

Definition 72 (Algebraic closure)**.** *A field K is said to be algebraically closed if every polynomial $f(x) \in K[x]$ has a zero in K. Such a polynomial splits into linear factors.*

Sometimes, a field F may not be algebraically closed, but all the polynomials in $F[x]$ have their zeros in K, an algebraict extension of F. Then K is called the algebraic closure of F.

From the cryptographic perspective, all the details of algebra are not needed. In [Kob98] Koblitz gives an excellent tutorial of what is needed for our purpose.

Appendix B

Curve Database

In this appendix-chapter, the reader may look up the details of the hyperelliptic curves which were used for the testing and comparison of algorithms mentioned in this thesis. The database has 150 curves, satisfying various parameters and they can be looked up in the 5 tables below.

#	Field	Curve $y^2 =$
1	952199	$x^5 + 308474 * x^4 + 925695 * x^3 + 135573 * x^2 + 503860 * x + 177405$
2	923917	$x^5 + 845747 * x^4 + 737151 * x^3 + 467122 * x^2 + 835737 * x + 141596$
3	965177	$x^5 + 696632 * x^4 + 964977 * x^3 + 703401 * x^2 + 861456 * x + 191726$
4	950611	$x^5 + 472292 * x^4 + 905508 * x^3 + 888057 * x^2 + 405833 * x + 856022$
5	905381	$x^5 + 381370 * x^4 + 91086 * x^3 + 291930 * x^2 + 223972 * x + 137727$
6	920497	$x^5 + 221340 * x^4 + 267870 * x^3 + 914875 * x^2 + 60119 * x + 116978$
7	914701	$x^5 + 488913 * x^4 + 477726 * x^3 + 62176 * x^2 + 171770 * x + 201159$
8	906259	$x^5 + 235896 * x^4 + 628781 * x^3 + 727368 * x^2 + 580885 * x + 350981$
9	986053	$x^5 + 16163 * x^4 + 385124 * x^3 + 140238 * x^2 + 365002 * x + 118442$
10	904147	$x^5 + 802370 * x^4 + 581623 * x^3 + 852960 * x^2 + 334067 * x + 753980$
11	949733	$x^5 + 589161 * x^4 + 224726 * x^3 + 818462 * x^2 + 926603 * x + 191471$
12	915251	$x^5 + 323219 * x^4 + 51552 * x^3 + 403315 * x^2 + 364495 * x + 758611$
13	994337	$x^5 + 348867 * x^4 + 484819 * x^3 + 583458 * x^2 + 773998 * x + 118484$
14	966463	$x^5 + 87284 * x^4 + 292742 * x^3 + 413076 * x^2 + 807098 * x + 537581$
15	977239	$x^5 + 556097 * x^4 + 863815 * x^3 + 448222 * x^2 + 558078 * x + 548796$
16	931597	$x^5 + 613085 * x^4 + 53709 * x^3 + 722125 * x^2 + 663963 * x + 32753$
17	918431	$x^5 + 732804 * x^4 + 909857 * x^3 + 713375 * x^2 + 350137 * x + 525702$
18	911737	$x^5 + 5326 * x^4 + 589802 * x^3 + 505339 * x^2 + 319828 * x + 498167$
19	980717	$x^5 + 579306 * x^4 + 335321 * x^3 + 696098 * x^2 + 865945 * x + 363890$
20	943153	$x^5 + 625768 * x^4 + 845048 * x^3 + 631711 * x^2 + 558178 * x + 929066$

Table B.1: Curve Database I

#	Field	Curve $y^2 =$
21	935243	$x^5 + 146124 * x^4 + 580923 * x^3 + 345113 * x^2 + 646258 * x + 875693$
22	927961	$x^5 + 302032 * x^4 + 770093 * x^3 + 599324 * x^2 + 340845 * x + 876678$
23	960763	$x^5 + 334256 * x^4 + 155264 * x^3 + 523903 * x^2 + 841394 * x + 684258$
24	911861	$x^5 + 335815 * x^4 + 327802 * x^3 + 551070 * x^2 + 414920 * x + 127089$
25	998813	$x^5 + 382474 * x^4 + 835628 * x^3 + 627490 * x^2 + 476589 * x + 448891$
26	928043	$x^5 + 319134 * x^4 + 545185 * x^3 + 127316 * x^2 + 267040 * x + 701437$
27	952163	$x^5 + 400477 * x^4 + 432850 * x^3 + 426697 * x^2 + 115640 * x + 282213$
28	964253	$x^5 + 929637 * x^4 + 160994 * x^3 + 522148 * x^2 + 514553 * x + 394123$
29	999181	$x^5 + 516964 * x^4 + 302284 * x^3 + 555897 * x^2 + 360724 * x + 467363$
30	943541	$x^5 + 291420 * x^4 + 117431 * x^3 + 570268 * x^2 + 566063 * x + 239351$
31	944773	$x^5 + 112164 * x^4 + 730068 * x^3 + 36016 * x^2 + 267839 * x + 317481$
32	907637	$x^5 + 506229 * x^4 + 455054 * x^3 + 541174 * x^2 + 196449 * x + 85008$
33	906461	$x^5 + 691669 * x^4 + 873182 * x^3 + 261543 * x^2 + 24711 * x + 559877$
34	989119	$x^5 + 60211 * x^4 + 79917 * x^3 + 957707 * x^2 + 579049 * x + 301535$
35	916213	$x^5 + 866080 * x^4 + 470448 * x^3 + 739836 * x^2 + 371418 * x + 776546$
36	935971	$x^5 + 738164 * x^4 + 886781 * x^3 + 659990 * x^2 + 879837 * x + 89867$
37	982703	$x^5 + 927946 * x^4 + 20961 * x^3 + 248970 * x^2 + 71513 * x + 456256$
38	969037	$x^5 + 323305 * x^4 + 159255 * x^3 + 675683 * x^2 + 268273 * x + 659947$
39	900563	$x^5 + 364811 * x^4 + 256616 * x^3 + 743231 * x^2 + 61078 * x + 175424$
40	904103	$x^5 + 878771 * x^4 + 392107 * x^3 + 260651 * x^2 + 346109 * x + 132175$
41	910471	$x^5 + 512853 * x^4 + 81118 * x^3 + 715561 * x^2 + 565884 * x + 557944$
42	955987	$x^5 + 55414 * x^4 + 119075 * x^3 + 68170 * x^2 + 632193 * x + 446733$
43	941557	$x^5 + 443188 * x^4 + 75279 * x^3 + 400529 * x^2 + 412056 * x + 826719$
44	916831	$x^5 + 559972 * x^4 + 142069 * x^3 + 147863 * x^2 + 219642 * x + 835352$
45	942433	$x^5 + 187062 * x^4 + 207093 * x^3 + 16167 * x^2 + 730536 * x + 352700$
46	970747	$x^5 + 761987 * x^4 + 438683 * x^3 + 581133 * x^2 + 32785 * x + 223148$
47	926633	$x^5 + 456145 * x^4 + 678919 * x^3 + 714671 * x^2 + 363252 * x + 834934$
48	976471	$x^5 + 85490 * x^4 + 77860 * x^3 + 434449 * x^2 + 313872 * x + 353471$
49	910421	$x^5 + 295787 * x^4 + 725416 * x^3 + 194139 * x^2 + 777672 * x + 40676$
50	993647	$x^5 + 229170 * x^4 + 376738 * x^3 + 107663 * x^2 + 613338 * x + 440009$
51	901891	$x^5 + 567807 * x^4 + 418521 * x^3 + 285982 * x^2 + 486775 * x + 299870$
52	950329	$x^5 + 767708 * x^4 + 195523 * x^3 + 80165 * x^2 + 180655 * x + 27641$

Table B.2: Curve Database II

#	Field	Curve $y^2 =$
53	918641	$x^5 + 915888 * x^4 + 259085 * x^3 + 305820 * x^2 + 342520 * x + 902511$
54	953773	$x^5 + 346600 * x^4 + 774616 * x^3 + 452647 * x^2 + 609948 * x + 33581$
55	972661	$x^5 + 466991 * x^4 + 605161 * x^3 + 705390 * x^2 + 883726 * x + 916168$
56	903541	$x^5 + 112643 * x^4 + 770036 * x^3 + 804706 * x^2 + 588768 * x + 835379$
57	983513	$x^5 + 642288 * x^4 + 143187 * x^3 + 20843 * x^2 + 4649 * x + 667765$
58	966727	$x^5 + 320445 * x^4 + 207163 * x^3 + 124293 * x^2 + 507447 * x + 238255$
59	979831	$x^5 + 839384 * x^4 + 222311 * x^3 + 462038 * x^2 + 337295 * x + 33443$
60	910229	$x^5 + 267073 * x^4 + 746452 * x^3 + 140984 * x^2 + 622670 * x + 528420$
61	964289	$x^5 + 610447 * x^4 + 687677 * x^3 + 69989 * x^2 + 618993 * x + 240518$
62	954221	$x^5 + 258797 * x^4 + 684417 * x^3 + 621193 * x^2 + 324361 * x + 687141$
63	978389	$x^5 + 158134 * x^4 + 252056 * x^3 + 560781 * x^2 + 516362 * x + 705129$
64	908363	$x^5 + 201772 * x^4 + 347360 * x^3 + 146106 * x^2 + 525564 * x + 291647$
65	958921	$x^5 + 236201 * x^4 + 914770 * x^3 + 813276 * x^2 + 250122 * x + 162319$
66	907363	$x^5 + 513615 * x^4 + 52316 * x^3 + 20406 * x^2 + 190287 * x + 284444$
67	910177	$x^5 + 629253 * x^4 + 460689 * x^3 + 579116 * x^2 + 826627 * x + 340533$
68	977591	$x^5 + 583540 * x^4 + 124590 * x^3 + 169167 * x^2 + 811997 * x + 417768$
69	938437	$x^5 + 654862 * x^4 + 453935 * x^3 + 420859 * x^2 + 724659 * x + 81669$
70	959263	$x^5 + 387097 * x^4 + 720977 * x^3 + 327083 * x^2 + 476228 * x + 379088$
71	926701	$x^5 + 710026 * x^4 + 287242 * x^3 + 155133 * x^2 + 486130 * x + 43680$
72	920197	$x^5 + 24682 * x^4 + 281421 * x^3 + 539839 * x^2 + 11885 * x + 164240$
73	919013	$x^5 + 705910 * x^4 + 199623 * x^3 + 572806 * x^2 + 167812 * x + 805401$
74	923539	$x^5 + 126109 * x^4 + 41294 * x^3 + 791372 * x^2 + 371138 * x + 532177$
75	949733	$x^5 + 281600 * x^4 + 227997 * x^3 + 875928 * x^2 + 458945 * x + 80809$
76	930737	$x^5 + 26153 * x^4 + 722274 * x^3 + 566502 * x^2 + 46356 * x + 594326$
77	976853	$x^5 + 260704 * x^4 + 911156 * x^3 + 369606 * x^2 + 269146 * x + 528503$
78	994927	$x^5 + 880424 * x^4 + 408335 * x^3 + 137512 * x^2 + 615796 * x + 513052$
79	918733	$x^5 + 580469 * x^4 + 204042 * x^3 + 372134 * x^2 + 392585 * x + 358505$
80	960931	$x^5 + 513818 * x^4 + 680267 * x^3 + 11661 * x^2 + 847635 * x + 939420$
81	943471	$x^5 + 467547 * x^4 + 358411 * x^3 + 740854 * x^2 + 116245 * x + 42350$
82	923617	$x^5 + 558081 * x^4 + 390611 * x^3 + 688817 * x^2 + 561472 * x + 777871$
83	968467	$x^5 + 575258 * x^4 + 686641 * x^3 + 820067 * x^2 + 377482 * x + 15313$
84	978001	$x^5 + 609724 * x^4 + 595441 * x^3 + 355992 * x^2 + 503159 * x + 659553$
85	969421	$x^5 + 465880 * x^4 + 161125 * x^3 + 636205 * x^2 + 914646 * x + 548369$

Table B.3: Curve Database III

#	Field	Curve $y^2 =$
86	979807	$x^5 + 849749 * x^4 + 357177 * x^3 + 777899 * x^2 + 627546 * x + 749719$
87	965491	$x^5 + 964855 * x^4 + 311539 * x^3 + 696336 * x^2 + 612648 * x + 541082$
88	970859	$x^5 + 232623 * x^4 + 2761 * x^3 + 665717 * x^2 + 914105 * x + 445769$
89	944491	$x^5 + 882817 * x^4 + 16255 * x^3 + 483054 * x^2 + 602580 * x + 521030$
90	974167	$x^5 + 268719 * x^4 + 92088 * x^3 + 639935 * x^2 + 500236 * x + 152777$
91	961871	$x^5 + 749032 * x^4 + 466051 * x^3 + 354382 * x^2 + 643823 * x + 785668$
92	950221	$x^5 + 264068 * x^4 + 734527 * x^3 + 524677 * x^2 + 198317 * x + 256834$
93	971491	$x^5 + 181513 * x^4 + 474682 * x^3 + 731641 * x^2 + 163741 * x + 667060$
94	909071	$x^5 + 643387 * x^4 + 119279 * x^3 + 187103 * x^2 + 378285 * x + 816769$
95	912763	$x^5 + 287627 * x^4 + 349824 * x^3 + 502493 * x^2 + 39028 * x + 656690$
96	997013	$x^5 + 885968 * x^4 + 475473 * x^3 + 193991 * x^2 + 873436 * x + 206068$
97	982271	$x^5 + 684362 * x^4 + 527866 * x^3 + 462578 * x^2 + 368291 * x + 777162$
98	918431	$x^5 + 250073 * x^4 + 267946 * x^3 + 881682 * x^2 + 729117 * x + 830045$
99	915437	$x^5 + 792764 * x^4 + 368799 * x^3 + 874837 * x^2 + 573052 * x + 632498$
100	950921	$x^5 + 310381 * x^4 + 483901 * x^3 + 894620 * x^2 + 117339 * x + 573050$
101	976991	$x^5 + 606644 * x^4 + 644314 * x^3 + 339718 * x^2 + 159335 * x + 261407$
102	953431	$x^5 + 552110 * x^4 + 540686 * x^3 + 747278 * x^2 + 87804 * x + 737276$
103	981713	$x^5 + 293911 * x^4 + 307274 * x^3 + 555581 * x^2 + 801463 * x + 186455$
104	995833	$x^5 + 841079 * x^4 + 105371 * x^3 + 772157 * x^2 + 882275 * x + 279833$
105	930779	$x^5 + 836677 * x^4 + 461360 * x^3 + 311102 * x^2 + 683901 * x + 325193$
106	945289	$x^5 + 828704 * x^4 + 490383 * x^3 + 287874 * x^2 + 546724 * x + 613626$
107	979651	$x^5 + 865447 * x^4 + 824291 * x^3 + 465559 * x^2 + 90588 * x + 847886$
108	985013	$x^5 + 52954 * x^4 + 34816 * x^3 + 480083 * x^2 + 574302 * x + 526033$
109	980887	$x^5 + 946362 * x^4 + 860833 * x^3 + 639467 * x^2 + 695877 * x + 712571$
110	901751	$x^5 + 838525 * x^4 + 204498 * x^3 + 732915 * x^2 + 809393 * x + 889067$
111	998687	$x^5 + 134067 * x^4 + 735442 * x^3 + 424012 * x^2 + 38532 * x + 997697$
112	974747	$x^5 + 103225 * x^4 + 728227 * x^3 + 114167 * x^2 + 100151 * x + 856138$
113	978683	$x^5 + 607590 * x^4 + 549956 * x^3 + 628699 * x^2 + 485486 * x + 374205$
114	994501	$x^5 + 569473 * x^4 + 374412 * x^3 + 407717 * x^2 + 388451 * x + 928747$
115	968729	$x^5 + 945772 * x^4 + 906436 * x^3 + 248262 * x^2 + 75456 * x + 296993$
116	994751	$x^5 + 578119 * x^4 + 269463 * x^3 + 151055 * x^2 + 559920 * x + 712056$
117	931003	$x^5 + 456756 * x^4 + 580152 * x^3 + 704506 * x^2 + 528293 * x + 650869$
118	916291	$x^5 + 832260 * x^4 + 834520 * x^3 + 325664 * x^2 + 707708 * x + 719909$

Table B.4: Curve Database IV

#	Field	Curve $y^2 =$
119	979747	$x^5 + 748096 * x^4 + 292993 * x^3 + 663979 * x^2 + 875484 * x + 785966$
120	989557	$x^5 + 263262 * x^4 + 40556 * x^3 + 19334 * x^2 + 775109 * x + 342663$
121	in x 113^3	$x^5 + 53 * x^4 + 67 * x^3 + 20 * x^2 + 102 * x + 32$
122	in x 227^3	$x^5 + 86 * x^4 + 154 * x^3 + 95 * x^2 + 59 * x + 183$
123	in x 211^3	$x^5 + 104 * x^4 + 107 * x^3 + 52 * x^2 + 44 * x + 191$
124	in x 1009^2	$x^5 + 288 * x^4 + 718 * x^3 + 382 * x^2 + 835 * x + 499$
125	in x 1783^2	$x^5 + 542 * x^4 + 1205 * x^3 + 1648 * x^2 + 568 * x + 81$
126	in x 1693^2	$x^5 + 779 * x^4 + 1429 * x^3 + 1312 * x^2 + 1133 * x + 808$
127	8262101	$x^5 + 4196304 * x^4 + 1137894 * x^3 + 3506813 * x^2 + 5164856 * x + 5396020$
128	in x 97^3	$x^5 + 25 * x^4 + 39 * x^3 + 63 * x^2 + 11 * x + 15$
129	4168933	$x^5 + 715768 * x^4 + 2865808 * x^3 + 3908416 * x^2 + 2123323 * x + 2058929$
130	in x 109^3	$x^5 + 38 * x^4 + 76 * x^3 + 93 * x^2 + 79 * x + 9$
131	7643731	$x^5 + 234475 * x^4 + 5576477 * x^3 + 7263848 * x^2 + 970251 * x + 3445515$
132	in x 1907^2	$x^5 + 1862 * x^4 + 1499 * x^3 + 89 * x^2 + 733 * x + 1476$
133	in x 3181^2	$x^5 + 2472 * x^4 + 1191 * x^3 + 1930 * x^2 + 856 * x + 330$
134	in x 647^2	$x^5 + 574 * x^4 + 423 * x^3 + 379 * x^2 + 464 * x + 252$
135	in x 3187^2	$x^5 + 1921 * x^4 + 758 * x^3 + 1546 * x^2 + 426 * x + 1818$
136	in x 197^3	$x^5 + 13 * x^4 + 185 * x^3 + 80 * x^2 + 34 * x + 2$
137	in x 1553^2	$x^5 + 173 * x^4 + 885 * x^3 + 379 * x^2 + 1309 * x + 1344$
138	8237839	$x^5 + 7185432 * x^4 + 3379093 * x^3 + 6878452 * x^2 + 6893411 * x + 6020685$
139	2474039	$x^5 + 1879790 * x^4 + 2268255 * x^3 + 827250 * x^2 + 694767 * x + 596778$
140	in x 691^2	$x^5 + 602 * x^4 + 392 * x^3 + 56 * x^2 + 386 * x + 388$
141	in x 4057^2	$x^5 + 296 * x^4 + 3613 * x^3 + 3857 * x^2 + 407 * x + 1977$
142	in x 2719^2	$x^5 + 2399 * x^4 + 2506 * x^3 + 1661 * x^2 + 1695 * x + 1455$
143	in x 149^3	$x^5 + 44 * x^4 + 18 * x^3 + 21 * x^2 + 136 * x + 16$
144	in x 883^2	$x^5 + 423 * x^4 + 411 * x^3 + 103 * x^2 + 373 * x + 87$
145	361723	$x^5 + 332038 * x^4 + 24939 * x^3 + 66156 * x^2 + 259821 * x + 90002$
146	1180477	$x^5 + 209775 * x^4 + 1151223 * x^3 + 614088 * x^2 + 105825 * x + 1011857$
147	in x 3163^2	$x^5 + 843 * x^4 + 1649 * x^3 + 893 * x^2 + 2392 * x + 2450$
148	in x 163^3	$x^5 + 135 * x^4 + 139 * x^3 + 57 * x^2 + 10 * x + 87$
149	in x 607^2	$x^5 + 471 * x^4 + 274 * x^3 + 80 * x^2 + 124 * x + 474$
150	5369183	$x^5 + 1143762 * x^4 + 2949093 * x^3 + 491258 * x^2 + 5057777 * x + 1189300$

Table B.5: Curve Database V

Bibliography

[AM93] A. O. L. Atkin and F. Morain. Elliptic curves and primality proving. *Mathematics of Computation*, 61:29–68, 1993.

[Atk92] A. O. L. Atkin. Several public email messages, 1992.

[BSS99] Ian F. Blake, G. Seroussi, and N. P. Smart. *Elliptic curves in cryptography*. Cambridge University Press, New York, NY, USA, 1999.

[Can87] D. G. Cantor. Computing in the jacobian of hyperelliptic curve. *Math. Comp.*, 48:95–101, 1987.

[Can94] D G Cantor. On the analogue of the division polynomials for hyperelliptic curves. *Journal für die reine und angewandte Mathematik*, 447:91–146, 1994.

[Coh96] H Cohen. *A course in computational algebraic number theory*. Springer, 1996.

[DH76] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[EG85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. pages 10–18, 1985.

[Ful69] B Fulton. *Algebraic curves*. Benjamin, 1969.

[Gau05] P. Gaudry. Fast genus 2 arithmetic based on theta functions. 2005. http://eprint.iacr.org/.

[GG01] Pierrick Gaudry and Nicolas Gürel. An extension of kedlayaś point-counting algorithm to superelliptic curves. *Lecture Notes in Comput. Sci.*, 2248:480–494, 2001.

[GH00] Pierrick Gaudry and Robert Harley. Counting points on hyperelliptic curves over finite fields. *ANTS IV ; LNCS 1838*, pages 297–312, 2000.

[Har00] R. Harley. adding.text, doubling.c, 2000.

[Her75] I N Herstein. *Topics in Algebra*. Wiley: New York, 2nd edition edition, 1975.

[Her86] I N Herstein. *Abstract Algebra*. Macmillan, 1986.

[Kam91] W Kampkötter. *Explizite Gleichungen für Jacobische Varietäten hyperelliptischer Kurven*. PhD thesis, Universitẗ Gesamthochschule Essen, 1991.

[Ked01] Kiran Sridhara Kedlaya. Counting points on hyperelliptic curves using monsky-washnitzer cohomology. *Journal of the Ramanujan Mathematical Society*, 16:323–338, 2001.

[Knu97] D E Knuth. *The Art Of Computer Programming - 2, Seminumerical Algorithms*. Addison-Wesley, third edition edition, 1997.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[Kob89] Neal Koblitz. Hyperelliptic cryptosystems. *J. Cryptology*, 1(3):139–150, 1989.

[Kob94] N Koblitz. *A Course in Number Theory and Cryptography*. Springer-Verlag, Berlin, 1994.

[Kob98] N Koblitz. *Algebraic Aspects of Cryptography*, volume Vol. 3 of *Algorithms and Computation in Mathematics*. Springer-Verlag, New York, 1998.

[Lan02] Tanja Lange. Efficient arithmentic on genus 2 hyperelliptic curves over finite fields via explicit formulae. *Cryptology ePrint archive*, 121, 2002.

[MAG] MAGMA.

[Man65] J I Manin. The hasse-witt matrix of an algebraic curve. *Transactions of American Mathematical Society*, 45:245–264, 1965.

[MKS] CHAO J MATSUO KAZUTO, HAGA TOMOYUKI and TSUJII SHIGEO. On construction of secure hyperelliptic curve cryptosystems using igusa invariants. *Transactions of the Institute of Electronics, Information and Communication Engineers*, J84A:1045–1053.

[Mum84] D. Mumford. *Tata Lectures on Theta II*. Birkhauser, 1984.

[MWZ96] Alfred J Menezes, Yi-Hong Wu, and Robert J Zuccerato. An elementary introduction to hyperelliptic curves. 1996.

[PH78] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, 24:106–110, 1978.

[Pil90] J. Pila. Frobenius maps of abelian varieties and finding roots of unity in finite fields. *Mathematics of Computation*, 55(192):745–763, 1990.

[Pol78] J. M. Pollard. Monte carlo methods for index computation mod p. *Mathematics of Computation*, 32:918–924, 1978.

[Rom] S. Roman. *Field Theory*, volume 158 of *Graduate Texts in Mathematics*.

[Sad] Sandeep Sadanandan. Implementation details, code and results.

[Sch95] René Schoof. Counting points on elliptic curves over finite fields. *Journal de Théorie des Nombres; de Bordeaux 7*, pages 219–254, 1995.

[Sha71] D. Shanks. Class number, a theory of factorization and genera. *Proc. Symp. Pure Mathematics*, 20:415–440, 1971.

[Sil86] Joseph H Silverman. *The Arithmetic of Elliptic Curves*. Springer-Verlag, 1986.

[ST92] Joseph H Silverman and John Tate. *Rational Points on Elliptic Curves*. Springer-Verlag, 1992.

[Ste07] William Stein. *SAGE Regerence Manual*, 2007.

[Sut07] Andrew V Sutherland. *Order Computations in Generic Groups*. PhD thesis, Massachusetts Institute of Technology, 2007.

[SV] Jasper Scholten and Frederik Vercauteren. An introduction to elliptic and hyperelliptic curve cryptography and the ntru cryptosystem.

[Wen03] Annegret Weng. Constructing hyperelliptic curves of genus 2 suitable for cryptography. *Math. Comput.*, 72(241):435–458, 2003.

Index