ELLIPTIC CURVE CRYPTOGRAPHY
Improving the Pollard-Rho Algorithm

Mandy Zandra Seet

Supervisors: A/Prof. Jim Franklin and Mr. Peter Brown

School of Mathematics and Statistics,

The University of New South Wales.

2nd November 2007

Submitted in partial fulfilment of the requirements of the degree of

Bachelor of Science with Honours

Acknowledgements

The names of the people that I have listed here have had a significant role in the writing of this thesis. For it is their guidance, support, and most importantly, encouragements that have helped me survive this year.

First and foremost, my utmost appreciation goes out to my supervisors A/Prof Jim Franklin and Peter Brown, who were brave enough to undertake the job of supervising my project even though both of them are not experts in this field.

The topic and area of my thesis has been inspired by a series of lectures taught by Dr. Kohel during the 2007 AMSI Summer School program. Later, he helped me to write the program code for the Pollard-Rho algorithm in `SAGE` and clarified my many queries on elliptic curve cryptography. On this note, I also would like to thank A/Prof Ian Doust for encouraging me to attend the summer school.

Getting `SAGE` to run the Pollard-Rho program on my home computer has been frustrating. I am especially indebted to Brian Li for waking up early one morning and spending the whole day setting up VMware (virtual machine) for me to run `SAGE` in. I also would like to thank the staff members in the Computing department for installing `SAGE` in several computers in the honours room. My appreciation also goes out to Tim and Amy, whose computers I "borrowed" and used to run my program. Sorry for all the inconveniences caused.

Sometimes things do not go according to your plan, and a while ago, I had to redo my experiments. To Dr. Astrid An Huef, Joshua Capel, Lawrence Chan, Mei Li Chong, Michelle Dunbar, Amy Kwon, Koushik Panda and everyone else who helped me out of the mess I was in or just for being there, *"Thank you so much"*. I would not have recovered so quickly otherwise.

I would also like to thank those who have sacrificed an hour to sit through my presentation rehearsals. These people include my friends from the UNSW Professional Speakers club and my church youth group. Thank you for all the evaluations and suggestions to help me present better.

And to my family and friends, who form an important group of people outside of Mathematics, for their prayers, encouragements and faith in me. My parents for working hard to send me to Australia so I can fulfil my dream; my best friends, Shuxian Chang and Mei Li Chong for putting up with my mood swings and mathematical idiosyncrasies during the stressful periods; and to everyone else who have encouraged me throughout the year.

Finally, the biggest *"Thank You"* goes out to Alex Rowley, for everything he has done to help me complete this thesis.

Preface

The main aspects of Cryptography are message confidentiality, data integrity and authentication. Until modern times, cryptography was used predominantly for military purposes. Nowadays, it is mainly the security of ATM cards, computer passwords and online bank transaction that depend on cryptography.

In 1985, two separate papers written by Koblitz and Miller suggested using elliptic curves in cryptography. Since then there has been much development in this area of research. With vast improvements in technology over the last 20 years, the sizes of public keys for the various public key cryptosystems, such as RSA and the El Gamal had to be constantly increased to ensure security and privacy. However, with elliptic curves, better security can be achieved with a smaller key size. This is one of the many advantages of elliptic curve cryptosystems.

In February 2005, the National Security Agency in the United States released a document, known as Suite B, to recommend the use of Elliptic Curve cryptography as the basis for all cryptographic algorithms. This move is brought about by the fact that the current RSA and El Gamal cryptosystems are becoming more susceptible to attacks and hence are slowly becoming outdated.

Since Elliptic Curve cryptography is a relatively new phenomenon, research is still ongoing. There are many open questions which are currently being studied. Active areas of research include developing algorithms and/or modifying known ones to "break" current elliptic curve cryptosystems. One such algorithm is the Pollard-Rho algorithm that solves the Elliptic Curve Discrete Logarithm Problem, which the security of the elliptic curve cryptosystem depends on. This method is currently regarded as the best algorithm for solving the elliptic curve discrete

logarithm problem, yet it is still not efficient enough. Thus, the objective of this thesis is to study the Pollard-Rho method and suggest ways to improve the algorithm.

Chapter 1

Chapter 1 starts off the thesis with an introduction to Cryptography. The concepts of public key cryptography will be discussed, and the protocol of the El Gamal cryptosystem will be stated.

Chapter 2

Here, the arithmetic of elliptic curves and their properties will be defined. We will also discuss elliptic curves over a finite field, and introduce some algorithms to calculate the number of points on an elliptic curve over the finite field.

Chapter 3

In this chapter, we will outline the protocol for the elliptic curve analog of the El Gamal cryptosystem. Next we will see that the security of this cryptosystem depends on the Elliptic Curve Discrete Logarithm Problem (ECDLP). With that, the idea of insecure curves will also be introduced.

Chapter 4

An algorithm to solve the elliptic curve discrete logarithm problem, the Pollard-Rho method will be introduced. We will see that it uses a random walk to solve the problem, and also show how to derive the expected run-time of this algorithm.

Chapter 5

Using the computer algebra system `SAGE`, we implement the Pollard-Rho method and conduct experiments to improve the efficiency of the algorithm. We will describe the setup of these experiments in this chapter. Our experiments are modelled after Teske's and Lamb's experiments, though we will be extending them by testing more variables than they did.

Chapter 6

Findings from our experiments will be presented and discussed here. From our experiments, the best improvement of the Pollard-Rho can further reduce the run-time of the original algorithm by nearly 25%, which is slightly higher than what Teske has achieved.

Chapter 7

This chapter will introduce some concepts of linear congruential methods and probability theory to explain the results that were obtained in Chapter 6.

Chapter 8

The thesis will conclude with a brief summary of each chapter and some ending remarks.

Appendices

Appendix A will briefly introduce Schoof's Algorithm, which is a method to calculate the number of points on an elliptic curve. Detailed tables of results for each experiment will be displayed in Appendix B, and the various codes for running the Pollard-Rho algorithm in `SAGE` will be included in Appendix C.

Contents

1 Public Key Cryptography **1**

1.1 Cryptography 1

1.2 Public Key Cryptography 2

1.3 El Gamal Encryption 5

2 Elliptic Curves **7**

2.1 Weierstrass Equation 7

2.2 The Group Operation 9

2.3 Elliptic Curves over Finite Fields 13

2.4 Computing the number of $\mathbb{F}_p$-points of an Elliptic Curve 14

3 Elliptic Curve Cryptography **18**

3.1 An Analog of the El Gamal Encryption 18

3.2 Insecure Curves 21

3.2.1 Curves susceptible to Pohlig-Hellman attack 21

3.2.2 Supersingular Curves 23

3.2.3 Anomalous Curves 24

3.3 Security Features that implement Elliptic Curve Cryptography . . . 25

4 Pollard-Rho Algorithm **27**

4.1 Pollard-Rho method for solving ECDLP 28

4.2 Random Walks 31

4.3 Complexity of the Algorithm 32

5 Experimental Setup **35**

5.1 Sample size 35

5.2 Generating instances of the ECDLP 36

5.3 Basis of Measurement 37

5.4 Hash function for partitioning the group 37

5.5 Types of Random Walks 39

6 Findings about the Pollard-Rho algorithm **40**

6.1 r-Adding Walks 40

6.2 $(r + q)$-Mixed Walks 44

6.3 Summary of results 46

7 Discussion **48**

7.1 "Spikes" in the graph 48

7.2 Better results with $(r + q)$-mixed walks 50

7.3 Provably Better Walks 52

8 Conclusion **57**

8.1 Summary of Thesis 57

8.2 Avenues for future research 59

8.3 Concluding Remark 61

Appendix **61**

A Schoof's Algorithm **62**

B Tables of results **65**

B.1 Results obtained for the different values of r in r-adding walks . . . 65

B.2 Results obtained for the different values of r and q in mixed walks . 67

C SAGE code **74**

C.1 Generating curves with prime group order 74

C.2 Pollard-Rho Algorithm 76

C.3 Modified Pollard-Rho program to run 1,200 ECDLP cases 78

Bibliography **85**

Chapter 1

Public Key Cryptography

"There are two types of encryption: one that will prevent your sister from reading your diary and one that will prevent your government."

- Bruce Schneier

1.1 Cryptography

Cryptography is the study of mathematical techniques for the secure transmission of a private message over an insecure channel[28]. For many years, the concept of cryptography was used to ensure the safe transfer of messages for military purposes. For example, during World War II, the Germans used *Enigma* machines to encipher their messages before transmitting them[44]. But these days, it is the security of ATM cards, computer passwords, online bank transaction and electronic commerce that mainly depend on cryptography.

In cryptography, the message that is to be sent out is known as the *plaintext*, but it is disguised or *enciphered* to protect its contents before it is sent out, and becomes the *ciphertext*. In order to read the plaintext, the ciphertext has to be *deciphered*.

Definition 1.1.1. *A* **cipher** *is an injective map e from the plaintext space $\mathcal{M}$ to the ciphertext space $\mathcal{C}$, thus $e : \mathcal{M} \to \mathcal{C}$, and a* **deciphering** *map d is the inverse of the cipher, that is $d : \mathcal{C} \to \mathcal{M}$, with the property that*

$$d \circ e = id_{\mathcal{M}}.$$

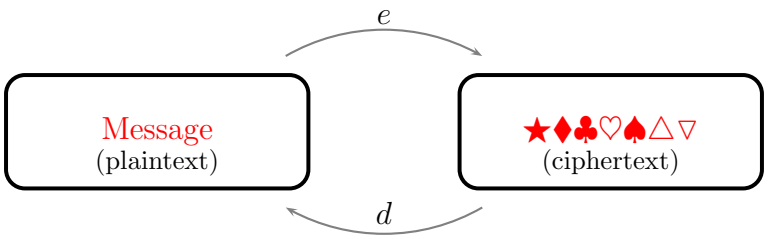


Figure 1.1: Enciphering a plaintext and deciphering a ciphertext.

A cryptosystem is the set-up of a cipher and deciphering map, which is mathematically defined as:

Definition 1.1.2. *Let $\mathcal{M}$ denote the message space, $\mathcal{C}$ denote the ciphertext space and $\mathcal{K}$ denote the key space. Then a* **cryptosystem**, *denoted $\mathcal{E}$, is defined by two maps:*

- $e_k : \mathcal{K} \times \mathcal{M} \longrightarrow \mathcal{C}$,
- $d_{k'} : \mathcal{K} \times \mathcal{C} \longrightarrow \mathcal{M}$,

such that for every $k \in \mathcal{K}$, there exists $k' \in \mathcal{K}$ with

$$d_{k'}(\mathcal{K}, e_k(\mathcal{K}, \mathcal{M})) = M,$$

for all $M \in \mathcal{M}$.

Here, k refers to the enciphering key while k' is the deciphering key. Just as a key is needed to lock and unlock doors, so in Cryptography, keys are needed to encipher a plaintext and decipher a ciphertext. If both the enciphering and deciphering keys are the same, $k = k'$ (or if k' can be easily derived from k), we say that the cryptosystem is *symmetric*. But if both keys are different, $k \neq k'$ (or if k' cannot be easily derived from k), we call this cryptosystem a *public key cryptosystem*.

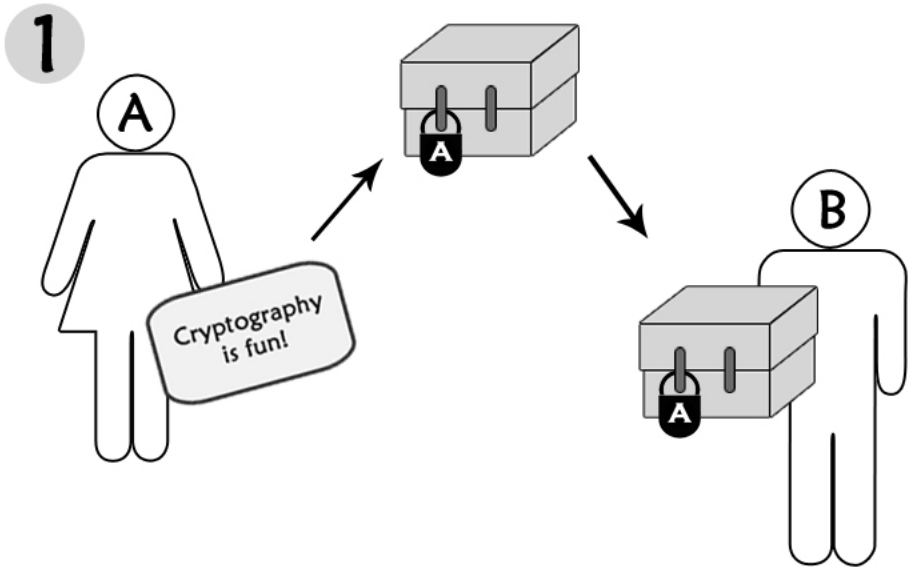
1.2 Public Key Cryptography

Suppose we have two persons, *Angelina Jolie* and *Brad Pitt*, who would like to have a private conversation. However, like all celebrities they are constantly surrounded

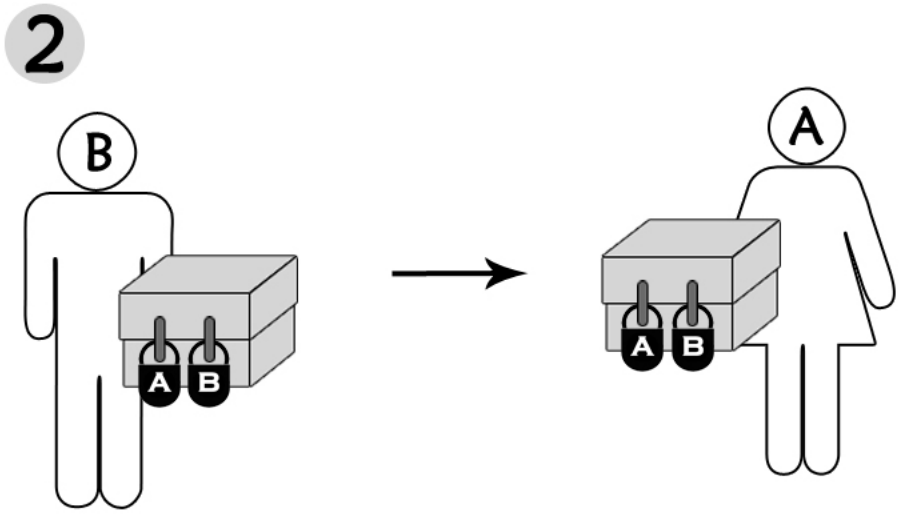
by the *Paparazzi* and their presence makes it difficult for Angelina and Brad to communicate.

To solve Angelina and Brad's problem, they can make use of the concept of Public Key Cryptography. This will allow them to converse in secret in front of the Paparazzi.

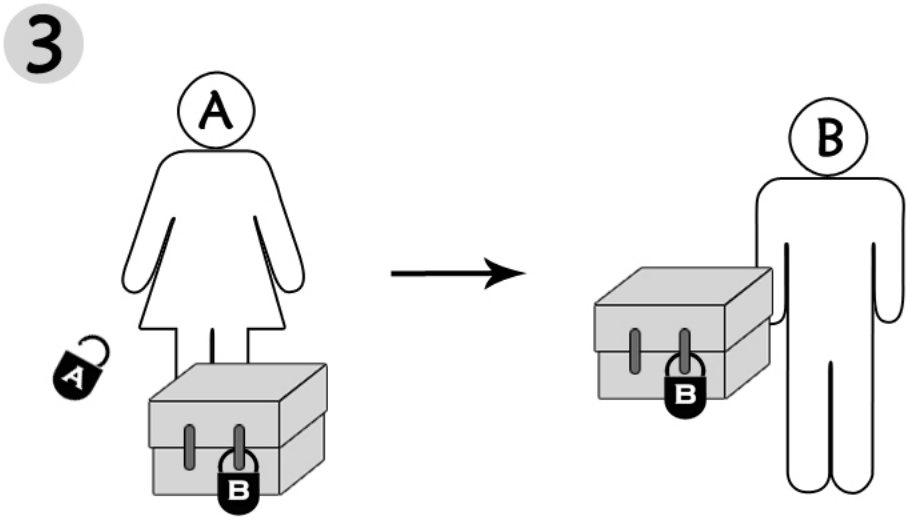
We have a diagram (below) to illustrate the protocols of a Public Key Cryptography.



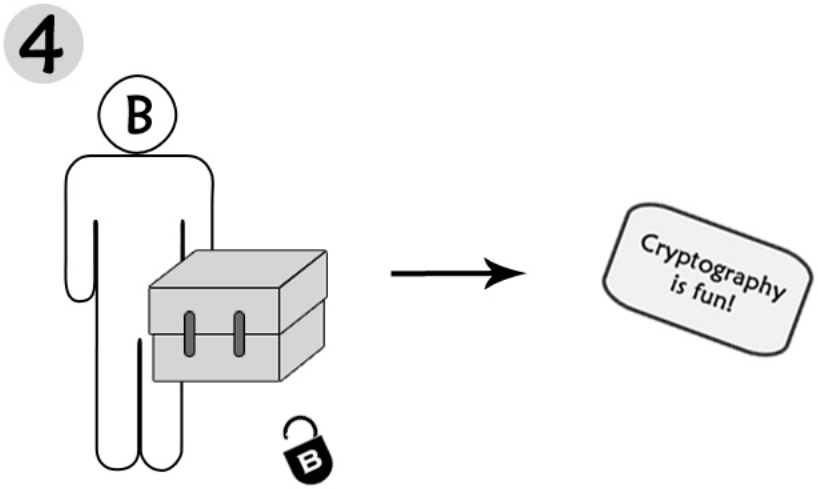
Angelina writes her message, "Cryptography is fun!", locks it in a box and sends it to Brad.



After Brad receives the box from Angelina, he adds his lock to the box and returns it to her.



Angelina then removes her lock, and the box goes back to Brad.



Now, Brad can unlock the box and read the message from Angelina.

There are several Public Key encryption algorithms. Common ones include the RSA encryption and the El Gamal encryption algorithm. The RSA encryption exploits the difficulty in factorising a large number that consists of two large prime factors. The El Gamal encryption exploits the cyclic properties of a finite field. The Elliptic Curve cryptosystem that we'll be looking at is the analog of the El Gamal encryption.

1.3 El Gamal Encryption

The El Gamal encryption was introduced by Taher El Gamal in the 1980's[65]. This cryptosystem is based on the Diffie-Hellman key exchange algorithm. It is usually defined over $\mathbb{Z}/p\mathbb{Z}$, for a prime, p, but it can be defined over any finite abelian cyclic group (such as the group defined by elliptic curves over the finite fields).

If our celebrity couple, *Angelina* and *Brad* decide to implement the El Gamal encryption, the algorithm is as follows:

Algorithm 1.3.1. El Gamal Encryption

1. *Angelina and Brad decide on a large prime p and form a finite field $\mathbb{F}_p$. They also decide on a primitive element, a.*

2. *Brad chooses a random x, where $1 < x < p-1$; and Angelina chooses a random y, where $1 < y < p-1$.*

3. *Brad computes $b = a^x$ in $\mathbb{F}_p$ and Angelina computes $r = a^y$. Brad's and Angelina's respective public keys b and r are then made public while their private keys x and y are kept in secret.*

4. *For each message, $m \in \mathbb{F}_p$ from Angelina to Brad,*

   (a) *She computes $s = m \cdot b^y = m \cdot a^{xy}$.*

   (b) *Angelina then sends the ciphertext, s to Brad.*

   (c) *To read the message, Brad calculates $m = s \cdot r^{-x} = s \cdot a^{-xy} \pmod p$.*

Note that Angelina never knows x and Brad never knows y, however, both can compute a^{xy}. The following information, p, a, a^x, a^y, s is made public, and is available to the Paparazzi. But only Angelina and Brad have the means to compute a^{xy}, and this is the key to deciphering the ciphertext.

Though the Paparazzi know what a^x and a^y are, it is difficult to compute a^{xy}. This problem that the Paparazzi faced is known as the **Diffie Hellman Problem**. And so, the security of the El Gamal cryptosystem is based on the difficulty in solving this problem.

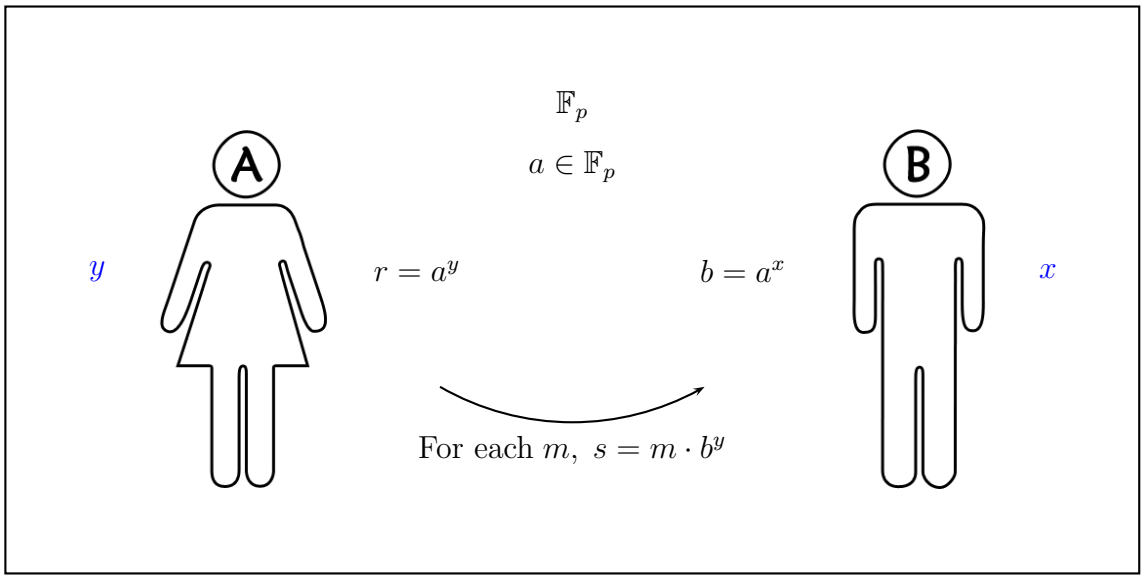


Figure 1.2: Diagram of El Gamal encryption process

Diffie Hellman Problem *Let a be a primitive element of $\mathbb{F}_p$, where p is a prime number, and let $b = a^x$ and $c = a^y$ be elements of $\mathbb{F}_p$. Find a unique d in $\mathbb{F}_p$ such that*

$$d = a^{xy}.$$

The only efficient method to solve the Diffie Hellman Problem is to solve the **Discrete Logarithm Problem**, which is stated as follows.

Discrete Logarithm Problem (DLP) *If given p, a and $b = a^x$, where a is a primitive element of the finite field $\mathbb{F}_p$ and $b \in \mathbb{F}_p$. Find the unique x, where x lies in the interval $[1, p-1]$ such that*

$$b = a^x \ (mod\ p).$$

The difficulty in solving the Discrete Logarithm Problem is exploited here in the El Gamal cryptosystem. While it can be easy to compute a^x given x, reversing the process can be hard.

However, there are several known Index Calculus methods to solve the discrete logarithm problem such as the Baby-Step-Giant-Step method and the Pollard-Rho algorithm (for finite fields). As the available algorithms are constantly improved, and as the speed of computers increases, the Discrete Logarithm Problem can be easily solved. This breaks the El Gamal cryptosystem. To combat these attacks, either a larger key size has to be used or a more secure cryptosystem has to be implemented.

Chapter 2

Elliptic Curves

Why are elliptic curves called "elliptic curves"? Here is a very brief history which begins with ellipses. In the early 18th century, an Italian mathematician, Giulio Fagnano discovered elliptic integrals while attempting to calculate the arc length of ellipses[6]. These integrals involve functions of the form:

$$I(x) = \int^{x} \frac{1}{\sqrt{t^3 + at^2 + bt + c}} dt.$$

So in order to find out the arc length of an ellipse, one must evaluate the integral and in this way curves of the form $y^2 = x^3 + ax^2 + bx + c$ came to be known as elliptic curves[39].

2.1 Weierstrass Equation

The Weierstrass form of an elliptic curve E is

$$y^2 = x^3 + Ax + B, \tag{2.1}$$

where A, B are constants. We also insist that $4A^3 + 27B^2 \neq 0$ to avoid degenerate cases.

Elliptic curves are usually defined over fields. Let $\mathbb{F}$ be a field with characteristic $\neq 2, 3$. If x, y, A, and B belong to a field $\mathbb{F}$, we say that the elliptic curve E is defined over the field $\mathbb{F}$, and is denoted by the set of points:

$$E(\mathbb{F}) = \{(x, y) \in \mathbb{F} \times \mathbb{F} \mid y^2 = x^3 + Ax + B\} \cup \{\mathcal{O}\},$$

where A, B are some fixed constants, and $\mathcal{O}$ is the *"point at infinity"*. Figure 2.1 shows elliptic curves defined over the real field, $\mathbb{R}$.

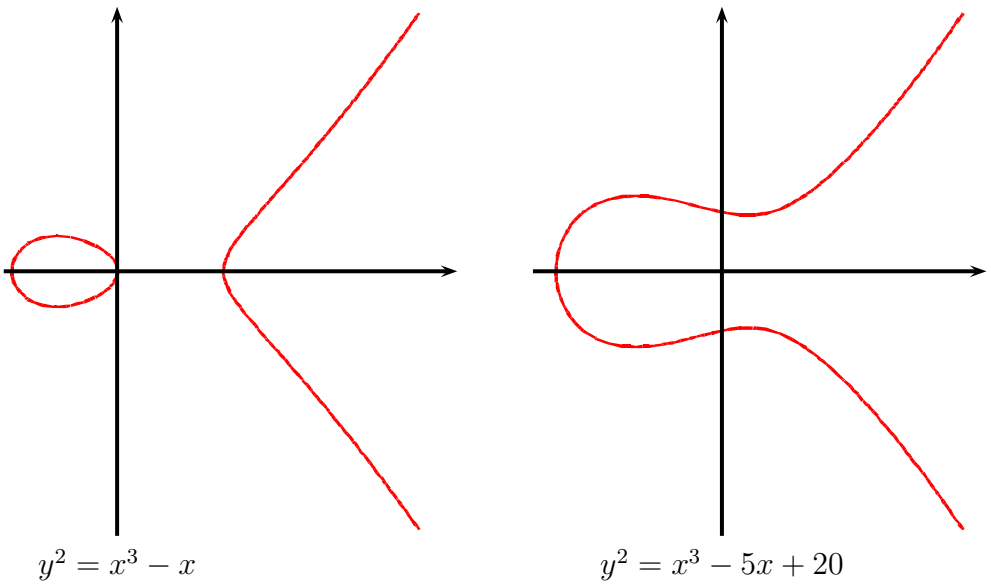


Figure 2.1: Graphs of elliptic curves over the real field

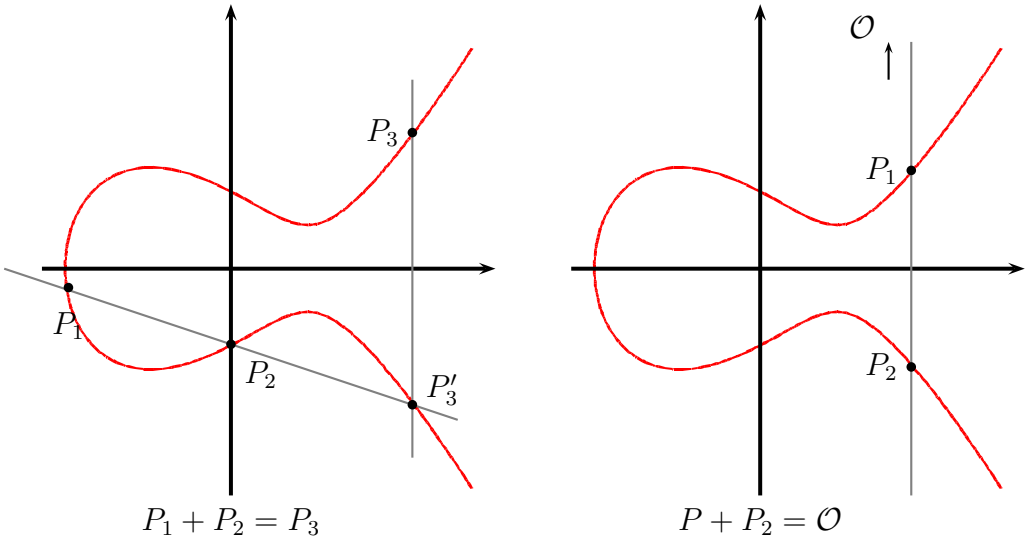


Figure 2.2: Adding points on an elliptic curve

2.2 The Group Operation

Now we would like to define the group operation, denoted by +, to add points in $E(\mathbb{F}_p)$.

Definition 2.2.1. Addition *Given two distinct points, $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ on an elliptic curve E, we define the addition $P_1 + P_2 = P_3$ by the following steps:*

- *Draw a line, L through P_1 and P_2 (neither equal to $\mathcal{O}$)*
- *If $x_1 \neq x_2$, L intersects E at a third point, P_3'*
- *Reflect P_3' in the x-axis to get P_3, and define $P_1 + P_2 = P_3$*
- *If $x_1 = x_2$, the sum of P_1 and P_2 is defined to be $\mathcal{O}$, the point at infinity. Furthermore, we define $P + \mathcal{O} = \mathcal{O} + P = P$*

Later, we will see that the addition of points and the point at infinity $\mathcal{O}$ turns $E(\mathbb{F})$ into an abelian group. Figure 2.2 on page 9 depicts the group operation graphically.

To calculate the coordinates of P_3 in terms of the coordinates of P_1 and P_2, we need to consider the different cases. We assume, initially, that our elliptic curve is over $\mathbb{R}$ to derive the formula for $P_1 + P_2$, and then use the formulae to define addition over other fields, where the "geometry" is meaningless. In particular, we can apply the formulae to elliptic curves defined over $\mathbb{F}_p$, where p is prime, provided $p \neq 2$ or 3.

Case I: P_1 and P_2 are distinct, $x_1 \neq x_2$

(See left diagram of Figure 2.2.)

We calculate the slope of L to be,

$$m = \frac{y_2 - y_1}{x_2 - x_1},$$

and hence the equation of the line, L is:

$$y = m(x - x_1) + y_1. \tag{2.2}$$

To compute the coordinates of $P_3 = (x_3, y_3)$, we substitute (2.2) into (2.1):

$$(m(x - x_1) + y_1)^2 = x^3 + Ax + B. \tag{2.3}$$

Expanding the left hand side of the equation and rearranging it we obtain

$$x^3 - m^2x^2 + \ldots = 0. \tag{2.4}$$

Note that we already know that first two roots of the equation are x_1 and x_2 and hence

$$x_3 = m^2 - (x_1 + x_2)$$

$$y_3 = m(x_3 - x_1) + y_1.$$

Thus, the coordinates for P_3 are

$$\left(m^2 - (x_1 + x_2), -m(x_3 - x_1) - y_1\right),$$

where $m = \frac{y_2 - y_1}{x_2 - x_1}$.

Case II: P_1 and P_2 are distinct but $x_1 = x_2$

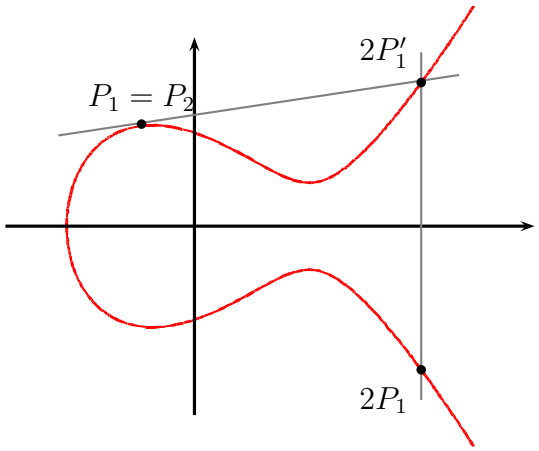
(See right diagram of Figure 2.2.)

Notice that since the line, L through both points is vertical, in this case we define

$$P_1 + P_2 = \mathcal{O}.$$

Also, if $P_1 = (x_1, y_1)$ then $P_2 = (x_1, -y_1)$.

Case III: $P_1 = P_2$, with $y_1 = y_2 \neq 0$



In this case, we draw a tangent at P_1 with gradient m. Using implicit differentiation, we find the gradient of the tangent to be,

$$2y\frac{dy}{dx} = 3x^2 + A$$
$$m = \frac{dy}{dx} = \frac{3x_1^2 + A}{2y_1}.$$

The equation of L is

$$y = m(x - x_1) + y_1,$$

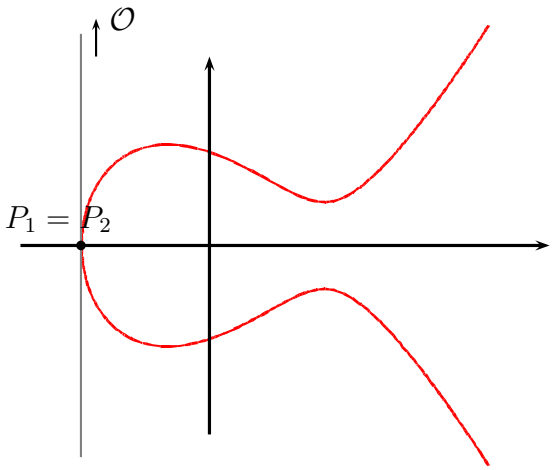
and the coordinates of P_3 are

$$\left(m^2 - (2x_1), -m(x_3 - x_1) - y_1\right),$$

where $m = \frac{3x_1^2+A}{2y_1}$. We write, in this case,

$$P_1 + P_1 = 2P_1.$$

Case IV: $P_1 = P_2$, with $y_1 = y_2 = 0$



We see that L_{tan}, the tangent line through P_1, again, intersects E at "point at infinity", $\mathcal{O}$. This is the same as case II and

$$P_1 + P_2 = 2P_1 = \mathcal{O}.$$

Since we defined $P+\mathcal{O} = P$, we see that $\mathcal{O}$ is an additive identity. In conclusion, we have the following group law.

Theorem 2.2.2. Group Law *The addition of points satisfies the following properties, for all P_1, P_2, P_3 on E over $\mathbb{F}$,*

1. *$P_1 + P_2$ also lie on E. (Closure)*
2. *$P_1 + P_2 = P_2 + P_1$. (Commutative property)*
3. *$P_1 + \mathcal{O} = P_1$. (Additive Identity)*
4. *Given P_1 on E, there exists P_2 on E such that $P_1 + P_2 = \mathcal{O}$. We denote the additive inverse of P_1 as $-P_1$. (Existence of inverses)*
5. *$(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$. (Associativity)*

NOTE: Even though we denote the additive inverse of P as $-P$, if $P = (x, y)$ then $P \neq (-x, -y)$. From Case II, we see that the coordinates of the inverse of P are $(x, -y)$.

Proof: As defined, the line through P_1 and P_2 is the same as the line through P_2 and P_1 and so the commutative property holds.

(3) is defined in the definition of the group operation. And I have already explained (4) in case II, so both properties hold.

The proof of associativity is long and tedious, and it will be omitted. A full proof is provided in Washington's book on Elliptic Curves [57], Section 2.4, page 20. ■

By theorem 2.2.2, $E(\mathbb{F})$ forms an abelian group.

Definition 2.2.3. *Suppose P lies on the elliptic curve E, then for any integer k, kP denotes*

$$\underbrace{P + P + \cdots + P}_{k \text{ times}}.$$

Because of the way addition is defined on this group, the additive structure of points is not obvious, and you can see that if given a point, $Q = kP$, it is hard to determine what the original point P is. Furthermore, if given Q and P, it is also difficult to find the number of times P is added to itself to get Q. This is the property that is exploited in cryptography, and especially in the elliptic curve analog of the El Gamal encryption algorithm (See Section 3.1).

2.3 Elliptic Curves over Finite Fields

As elliptic curves can be defined over any field, we can define them over the finite fields[53]. For Elliptic Curve Cryptography, the curves are defined over some finite field $\mathbb{F}_p$, where p is prime ($p \neq 2, 3$), and all arithmetic is done in this field. Also, for cryptographic purposes, we also require that p be a large prime.

Let $\mathbb{F}_p$ be a finite field with $p \neq 2, 3$ because we want $4A^3 + 27B^2 \neq 0$ and that we do not want division by 0 when adding points. The curve $E : y^2 = x^3 + Ax + B$ is defined over $\mathbb{F}_p$ if x, y, A, and B lie in $\mathbb{F}_p$, and the group $E(\mathbb{F}_p)$ is denoted by the set of points:

$$E(\mathbb{F}_p) = \big\{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p \mid y^2 = x^3 + Ax + B\big\} \cup \{\mathcal{O}\},$$

where A, B are fixed constants.

Then $E(\mathbb{F}_p)$ has finitely many points in $\mathbb{F}_p$. Let us illustrate this with an example.

Example 2.3.1. *Define $E(\mathbb{F}_7) : y^2 = x^3 + x + 1$. Then the group contains 5 points and they are:*

$$\{\mathcal{O}, (0, 1), (0, 6), (2, 2), (2, 5)\}.$$

□

The addition of points that we have defined in the earlier section is well defined in $\mathbb{F}_p$, since all computations are done modulo p and $p \neq 2, 3$. Let's take another example.

Example 2.3.2. *Given curve, $E : y^2 = x^3 + x + 1$ over $\mathbb{F}_{13}$. The group $E(\mathbb{F}_{13})$ has 18 elements.*

$$\left\{ \begin{array}{c} \mathcal{O}, (12, 8), (11, 2), (0, 12), (4, 11), (1, 9), (10, 6), (5, 12), (8, 12), \\ (7, 0), (8, 1), (5, 1), (10, 7), (1, 4), (4, 2), (0, 1), (11, 11), (12, 5) \end{array} \right\}$$

1. *Let us add two points, $(12, 8)$ and $(1, 9)$ on E. Firstly, we calculate m to be:*

$$m = \frac{9 - 8}{1 - 12} = \frac{1}{-11} = \frac{1}{2} \equiv 7 \ (mod\ 13).$$

Then using the formulae in Section 2.2 to calculate the coordinates, we have

$$x_3 = 7^2 - (12 + 1) \equiv 10 \ (mod\ 13)$$

$$y_3 = 7(10) + 2 \equiv 7 \ (mod\ 13).$$

So $(12, 8) + (1, 9) = (10, 7)$ *and we see that* $(10, 7)$ *lies on* E. □

2. *Now let us try to double points. We want to calculate* $2 \times (11, 2)$. *We found* m *to be:*

$$m = \frac{3(11^2) + 1}{2 \times 2} \equiv 0 \ (mod\, 13).$$

Using the formulas again, we calculate the coordinates to be

$$x_3 = 0 - (2 \times 11) \equiv 4 \ (mod\, 13)$$

$$y_3 = 0 + 2 \equiv 2 \ (mod\, 13).$$

So we have $2 \times (11, 2) = (4, 2)$ *which lies on* E. □

2.4 Computing the number of $\mathbb{F}_p$-points of an Elliptic Curve

Determining the group order is essential in Cryptanalysis. While no straightforward formula is known for calculating the group order for the elliptic curve group, there are several methods and/or algorithms available, such as Schoof's algorithm and the Baby-Step-Giant-Step (BSGS) method. In this section, we shall study the BSGS method to compute the number of $\mathbb{F}_p$-points of a given elliptic curve, E.

Let E be an elliptic curve over the finite field, $\mathbb{F}_p$, then the order of $E(\mathbb{F}_p)$ is denoted by $\#E(\mathbb{F}_p)$. We have a bound for $\#E(\mathbb{F}_p)$ given by **Hasse's Theorem**.

Theorem 2.4.1. Hasse's Theorem *The order of* $E(\mathbb{F}_p)$ *satisfies the following inequality*

$$|p + 1 - \#E(\mathbb{F}_p)| \leqslant 2\sqrt{p}. \tag{2.5}$$

Proof: The proof of this won't be covered here. For full details, please refer to Washington's book on Elliptic Curves [57], Section 4.2, pages 91-94. ■

Definition 2.4.2. *Let* P *be a point on* E, *then the* **order** *of* P *is the smallest positive integer,* k, *such that* $kP = \mathcal{O}$, *where* $\mathcal{O}$ *is the group identity. We denote the order of* P *by* $ord(P)$.

Lagrange's theorem states that the order of a point P in $E(\mathbb{F}_p)$ divides the group order $\#E(\mathbb{F}_p)$, so if we find that the order of P is greater than $(p+1-2\sqrt{p})$, then we can say that $\#E(\mathbb{F}_p)$ is an integer multiple of that order, which lies in Hasse's interval,

$$(p+1) - 2\sqrt{p} \leqslant \#E(\mathbb{F}_p) \leqslant (p+1) + 2\sqrt{p}. \tag{2.6}$$

If there is more than one integer multiple of the order of P that lies in Hasse's interval (2.6), we have to find the orders of other points, say Q and R. Then we find the lowest common multiple of these orders which lie in Hasse's interval.

With this result, we just need to determine the order of a point P, and to do that, we will use the Baby-Step-Giant-Step (BSGS) algorithm.

Baby-Step-Giant-Step method

Let P be a point on the elliptic curve, E over $\mathbb{F}_p$. Then proceed as follows:

1. Compute $Q = (p+1)P$.
2. Compute $m = \lceil p^{\frac{1}{4}} \rceil$ and create the BabyStep set:
$$\{P, 2P, 3P, \ldots, mP\}.$$
3. Compute $R = 2mP$ and create the GiantStep set:
$$\{Q - R, Q - 2R, Q - 3R, \ldots\}.$$
until a match is found between the BabyStep set and the GiantStep set. i.e.
$$Q - jR = Q - 2jmP = iP, \qquad \text{for some } j\&i.$$
4. Conclude that $(p + 1 - 2jm - i)P = kP = \mathcal{O}$.
5. Factorise $k = k_1 k_2 \cdots k_r$.
6. Compute $\frac{k}{k_q}P$ for $q = 1, \ldots, r$:
   (a) If $\frac{k}{k_q}P \neq \mathcal{O}$ for all q, then $k = ord(P)$.
   (b) If $\frac{k}{k_q}P = \mathcal{O}$ for some q, then repeat (5) with $\frac{k}{k_q}$.

7. Repeat (1) to (6) with randomly chosen points in $E(\mathbb{F}_p)$ until the lowest common multiple of orders divides only one integer n, which lies in Hasse's interval. Then $n = \#E(\mathbb{F}_p)$.

Example 2.4.3. *1. Define an elliptic curve $E : y^2 = x^3 + 4x + 10$ over $\mathbb{F}_{19}$. Compute m to be $\lceil 19^{\frac{1}{4}} \rceil = 3$ and take the point $P = (2, 8)$ on the curve. Hasse's interval is*

$$11.28 \leqslant \#E(\mathbb{F}_{19}) \leqslant 28.72. \tag{2.7}$$

Next we compute $Q = (19 + 1)P = (16, 3)$ and create the BabyStep set:

$$\mathcal{B} = \{P, 2P, 3P\} = \{(2, 8), (16, 16), (18, 10)\}.$$

Then we calculate $R = 2mP = 6P = (13, 6)$ and create the GiantStep set:

$$\mathcal{G} = \{(18, 10)\} = \{Q - R\}.$$

We stopped generating elements in $\mathcal{G}$ because we see that we have a match between $\mathcal{B}$ and $\mathcal{G}$. i.e.

$$\begin{aligned} Q - R &= 3P \\ 20P - 6P - 3P &= \mathcal{O} \\ 11P &= \mathcal{O}. \end{aligned}$$

Since 11 is prime, the order of P is 11. Also, we observe that there is only one integer multiple of $ord(P)$ that lies in (2.5) which is 22. Hence,

$$\#E(\mathbb{F}_{19}) = 22.$$

□

2. Define an elliptic curve E by the equation: $y^2 = x^3 + x + 1$. In $\mathbb{F}_{13}$, Hasse's interval will be:

$$6.8 \leqslant \#E(\mathbb{F}_{13}) \leqslant 21.2. \tag{2.8}$$

Pick a point on the curve, $P = (10, 6)$. Using the Baby-Step-Giant-Step (BSGS) method, we calculate that $ord(P) = 3$. Since 3 has several integer multiples in interval (2.6), so we have to repeat the algorithm with another point, $Q = (5, 1)$. We find that the order of Q is 18. Now we see that 18 has

only one integer multiple in (2.6) and itself is an integer multiple of $ord(P)$. *So we conclude that*

$$\#E(\mathbb{F}_{13}) = 18.$$

□

The BSGS method is a simple method to compute the number of points in $E(\mathbb{F}_p)$. One of the advantages of this method is that the algorithm requires $O(\sqrt{n})$ operations, where n is the number of points. Thus, the group order $\#E(\mathbb{F}_p)$ can be determined very quickly. However, when dealing with elliptic curves used in commercial cryptosystems, the preferred algorithm is the Schoof-Elkies-Atkins (or Schoof's) algorithm[42]. According to Washington[57], Schoof's algorithm (see Appendix A) has been known to compute elliptic curve groups with order over 200 digits.

Chapter 3

Elliptic Curve Cryptography

Since the El Gamal cryptosystem can be based on any cyclic group, we can apply the same system on the elliptic curve group, $E(\mathbb{F}_p)$[26]. As seen in Section 2.2, addition is not a straightforward process. Moreover, we saw in Section 2.4 that determining the group order of $E(\mathbb{F}_p)$ requires complex algorithms. This means that with Elliptic Curve Cryptography, smaller keys are sufficient to provide more security in message or data transmission. For example, the sizes of public and private keys of the El Gamal cryptosystem are 3072-bit and 256-bit respectively [63] while a 163-bit key size of an elliptic curve cryptosystem provides the same security level as the El Gamal keys[30]. It is for these reasons that the National Security Agency (NSA) recommends Elliptic Curve cryptographic algorithms.

There are several Elliptic Curve Cryptosystems such as the analog of the El Gamal encryption, the analog of the RSA encryption and the Weil Pairing encryption. For the purposes of this thesis, we shall look at the elliptic curve El Gamal encryption procedure.

3.1 An Analog of the El Gamal Encryption

For an Elliptic Curve El Gamal encryption, all computations are done in the finite field, $\mathbb{F}_p$. Suppose our celebrity couple, Angelina and Brad choose to implement the analog of the El Gamal cryptosystem, then the algorithm is as follows:

Algorithm 3.1.1. El Gamal Encryption

1. *Angelina and Brad decide on an elliptic curve,* $E : y^2 = x^3 + Ax + B$ *and a large prime* p *to form the finite field* $\mathbb{F}_p$*. They also pick a point,* P *that lies on* E*.*

2. *Brad chooses a random integer* x*; and Angelina chooses a random integer* y*.*

3. *Brad computes* $Q = xP$ *and Angelina computes* $R = yP$*.* Q *and* R *are then made public while Brad and Angelina respectively keep* x *and* y *private.*

4. *For each message,* $M \in E(\mathbb{F}_p)$ *from Angelina to Brad,*

   (a) She calculates $S = M + yQ = M + xyP$*.*

   (b) Angelina then sends the ciphertext S *to Brad.*

   (c) To read the message, Brad calculates $M = S - xR = S - xyP$*. (Recall that* $-P$ *denotes the inverse of* P*.)*

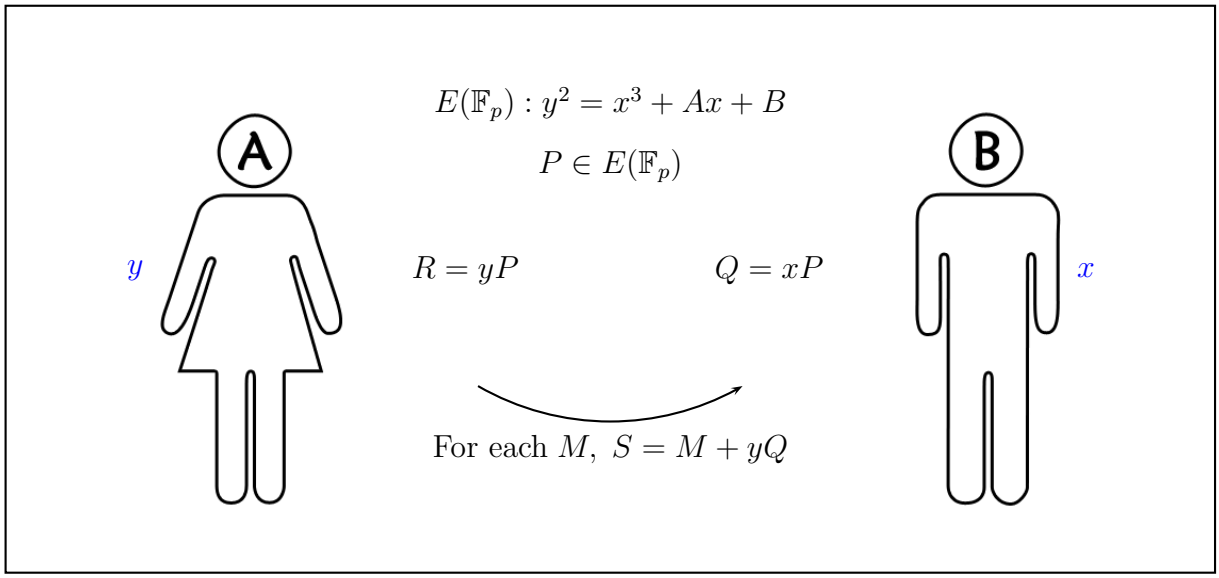


Figure 3.1: Diagram of Elliptic Curve El Gamal encryption process

Again, the following information $E(\mathbb{F}_p)$, P, yP, xP and S is made public, allowing our Paparazzi to have access to them. Let us give a numerical example to illustrate this encryption algorithm.

Example 3.1.2. *Firstly, Angelina and Brad decide on curve,* $E : y^2 = x^3+27x+1$ *over* $\mathbb{F}_{1049}$*, and a point* $P = (363, 48)$*.*

In private, Brad chooses integer $x = 153$*, and Angelina chooses* $y = 71$*. Then, Brad computes* $Q = 153P = (726, 370)$*, and Angelina computes* $R = 71P = (666, 867)$*.* Q *and* R *are then made public.*

Let the message that Angelina wants to send over to Brad be encrypted as $M = (907, 71)$*. She computes*

$$S = M + 71Q = (66, 417),$$

which she gives to Brad.

To read the message, Brad computes

$$S + 153(-R) = (907, 71) = M.$$

□

Similarly, the Diffie-Hellman Problem and the Discrete Logarithm problem exist for this Elliptic Curve cryptosystem.

Diffie Hellman Problem for Elliptic Curves *Given* $E(\mathbb{F}_p)$*,* P*,* xP *and* yP*. Determine* xyP*.*

Elliptic Curve Discrete Logarithm Problem (ECDLP) *Given* $E(\mathbb{F}_p)$*,* P *and* $Q = xP$*, find the unique* $x \in \mathbb{Z}$ *such that*

$$Q = xP,$$

where the coordinates of Q *are in* $\mathbb{F}_p$*.*

As in the finite field case, the security of this cryptosystem lies in the fact that if only P and Q (as above) are known to the Paparazzi, it is difficult to determine the number of times P has been added to itself to get Q. This property is due to the "random" additive structure of points. Koblitz[26] mentioned that the techniques developed to solve the DLP for finite fields often fail to work for the ECDLP. This fact enables this elliptic curve cryptosystem to remain secure while keeping the size of the field small. There exist several methods but without a efficient algorithm to attack the system, the difficulty in solving the ECDLP remains the key advantage of using elliptic curves in cryptography[43].

3.2 Insecure Curves

When implementing the elliptic curve cryptosystem, there are several classes of elliptic curves $E(\mathbb{F}_p)$ that Angelina and Brad should avoid if they want the maximum possible security level of the encryption system. If these curves are chosen, the elliptic curve discrete logarithm problem can be easily solved.

3.2.1 Curves susceptible to Pohlig-Hellman attack

The Pohlig-Hellman method is another algorithm that solves the ECDLP. It is most efficient on curve groups whose group order factorises into small primes. If the group order has a large prime factor, then the Pohlig-Hellman method is of little use.

To protect against this attack, the curve E and its defining finite field $\mathbb{F}_p$, over which it is defined, should be chosen such that $\#E(\mathbb{F}_p)$ is prime or *"almost prime"*, i.e. $\#E(\mathbb{F}_p) = qr$ where q is a large prime and r is a small cofactor[31].

The Pohlig-Hellman algorithm is as follows[20].

Let P, Q be elements of the elliptic curve group $E(\mathbb{F}_p)$, where $Q = xP$ for some x. Suppose the group order $\#E(\mathbb{F}_p) = n$ is known and the prime factorisation of n is:

$$n = \prod_{i}^{r} q_i^{e_i}. \tag{3.1}$$

The idea behind the attack is to calculate $x_i = x \pmod{q_i^{e_i}}$ and use the Chinese Remainder Theorem to calculate $x \pmod{n}$.

Algorithm 3.2.1. Pohlig-Hellman

Pick a prime factor, say q_j (and its exponent is e_j) for some j. The base-q_j expansion of x_j is:

$$x_j = z_0 + z_i q_j + z_2 q_j^2 + \ldots + z_{e_j - 1} q_j^{e_j - 1},$$

where $0 \leqslant z_n \leqslant q_j - 1$. The z_n's are calculated one at a time using the algorithm below:

1. *Compute $P_0 = \frac{N}{q_j} P$ and $Q_0 = \frac{N}{q_j} Q$.*

2. *Create the set,*

$$\mathcal{P} = \{P_0, 2P_0, \ldots, (q_j - 1)P_0\}.$$

Then for some $0 \leqslant \alpha \leqslant q-1$, $Q_0 = \alpha P_0$, *where* $0 \leqslant \alpha q_j - 1$, *put* $z_0 = \alpha$.

Since $ord(P_0) = q_j$, *then*

$$Q_0 = \frac{n}{q_j}Q = \frac{n}{q_j}(xP) = xP_0 = z_0P_0.$$

3. *If the exponent,* e_j *of* q_j *is 1, then stop.*

Otherwise, proceed to (4).

4. *Compute* $Q_1 = \frac{N}{q_j^2}Q$.

5. *Then for some* $0 \leqslant \beta \leqslant q-1$, $Q_1 = \beta P_0$ *where* $0 \leqslant \beta q_j - 1$, *set* $z_1 = \beta$.

6. *Repeat (4) and (5), each time increasing the power of* q_j *by 1 until all the* z_n*'s are calculated. Then compute*

$$x_j = z_0 + z_i q_j + z_2 q_j^2 + \ldots + z_{e_j-1} q_j^{e_j-1} \ (mod\ q_j^{e_j}).$$

Repeat the algorithm for each prime factor q_i and its exponent e_i. A system of congruence equations is obtained,

$$\begin{aligned} x &= x_1 \ (\text{mod}\ q_1^{e_1}), \\ x &= x_2 \ (\text{mod}\ q_2^{e_2}), \\ &\vdots \\ x &= x_r \ (\text{mod}\ q_r^{e_r}), \end{aligned} \tag{3.2}$$

and by the Chinese Remainder Theorem, the solution k, to (3.2) is unique.

We can see that the Pohlig-Hellman algorithm works best if all the prime factors of the group order, N are small. If N has a large prime factor, say q, then this method fails as it is difficult and too time-consuming to list all q elements in $\mathcal{P}$. If N is prime or *"almost prime"*, then it is impractical to use the Pohlig-Hellman algorithm. This is why curves and finite fields should be chosen such that the group order is prime or *"almost prime"*.

Example 3.2.2. *1. Let $E : y^2 = x^3 + 4x + 11$ be defined over $\mathbb{F}_{23}$. The group order is*

$$\#E(\mathbb{F}_{23}) = 32 = 2^5.$$

Since $\#E(\mathbb{F}_{23})$ factorises into small primes, this elliptic curve group is very susceptible to the Pohlig-Hellman attack. □

2. Define $E : y^2 = x^3 + 1000x + 18$ over $\mathbb{F}_{10003}$. The group order is

$$\#E(\mathbb{F}_{10003}) = 100123 = 59 \times 1697.$$

Notice that $\#E(\mathbb{F}_{10003})$ has a large prime factor, so $E(\mathbb{F}_{10003})$ is less susceptible to the Pohlig-Hellman attack. □

3.2.2 Supersingular Curves

In their paper, Menezes, Okamoto and Vanstone came up with a method to reduce the ECDLP of supersingular curves to the discrete logarithm problem of the El Gamal cryptosystem[33].

Definition 3.2.3. *Let E be an elliptic curve defined over $\mathbb{F}_p$, and $\#E(\mathbb{F}_p) = p + 1 + t$ be the group order of the curve. Then $E(\mathbb{F}_p)$ is* ***supersingular*** *if*

$$t = 0 \ (mod\, p).$$

Example 3.2.4. *Consider the curve, $E : y^2 = x^3 - x$ over $\mathbb{F}_7$, with $\#E(\mathbb{F}_7) = 8$. Then*

$$t = \#E(\mathbb{F}_7) - (7 + 1) = 0 \ (mod\, 7).$$

So this curve is supersingular.

In general, the curve $E : y^2 = x^3 - x$ over $\mathbb{F}_p$ with $p = 3 \ mod\, 4$ is supersingular[5]. □

The MOV algorithm is especially efficient for this class of curves, and as a precaution supersingular curves should be avoided when implementing the elliptic curve cryptosystem.

There are tests available to determine if a curve is supersingular and if it is susceptible to the MOV attack. The following corollary provides one such test[57].

Corollary 3.2.5. *Suppose $p \geqslant 5$ is prime. Then E is supersingular if and only if $t = 0$, if and only if $\#E(\mathbb{F}_p) = p + 1$.*

Proof: If $t = 0 \pmod{p}$, then from definition 3.2.2, E is supersingular. Suppose $p \geqslant 5$ is prime, and E is supersingular, but $t \neq 0$. Then $t \equiv 0 \pmod{p}$ implies that $|t| \geqslant p$. Since $t = (p+1) - \#E(\mathbb{F}_p)$, so by Hasse's Theorem (2.4.1) we have:

$$\begin{aligned} |t| &\leqslant 2\sqrt{p} \\ p &\leqslant 2\sqrt{p} \\ p^2 &\leqslant 4p \\ 0 &\leqslant 4p - p^2. \end{aligned}$$

Hence $0 \leqslant p \leqslant 4$, which contradicts the condition we've imposed on p. ■

According to a report by Certicom, cryptosystems that use supersingular curves are prohibited for government implementation[8].

3.2.3 Anomalous Curves

The last class of insecure elliptic curves that should be avoided is the anomalous curves. These are elliptic curves, E over $\mathbb{F}_p$ with group order,

$$\#E(\mathbb{F}_p) = p.$$

Example 3.2.6. *1. For curve $E : y^2 = x^3 + 4x + 10$ over $\mathbb{F}_{17}$, the group order is 17.*

2. For curve $E : y^2 = x^3 + x + 19$ over $\mathbb{F}_{103}$, the group order is 103.

These curves are anomalous curves. □

These curves were initially suggested to protect the cryptosystem against the MOV reduction algorithm. However, the ECDLP for the group can be solved quickly as shown by methods developed by Semaev [8]. Like supersingular curves, there are tests to determine if a chosen curve E, over $\mathbb{F}_p$ is susceptible to these attacks. This class of curves is also prohibited for use in cryptographic algorithms.

Return to Example 3.1.2 in Section 3.1, where E has the equation:

$$E(\mathbb{F}_{1049}) : y^2 = x^3 + 27x + 1.$$

Here, we find that,

1. E is not susceptible to the Pohlig-Hellman attack because

$$\#E(\mathbb{F}_{1049}) = 1059 = 3 \times 353,$$

and 353 is a large prime number.

2. The curve is not supersingular because

$$t = \#E(\mathbb{F}_{1049}) - (1050) = 9 \not\equiv 0 \pmod{1049}.$$

3. Lastly, it is not an anomalous curve because

$$\#E(\mathbb{F}_{1049}) = 1059 \neq 1049.$$

Therefore, $E(\mathbb{F}_{1049})$ is considered secure and Angelina and Brad will receive the maximum possible security level from this cryptosystem.

3.3 Security Features that implement Elliptic Curve Cryptography

One of the main advantages of using Elliptic Curve Cryptosystem is that they offer more security per bit in key sizes than either the RSA or El Gamal cryptosystems. This means that a much smaller key size is sufficient to provide the same level of security.

ECC key size	RSA key size	Ratio
163	1024	1:6
256	3072	1:12
384	7680	1:20
512	15,360	1:30

Table 3.1: Equivalent key size of the RSA cryptosystem to the Elliptic Curve cryptosystem

From Table 3.1, we see that the equivalent ECC key size is less than $\frac{1}{6}$ of the RSA system key size[7]. The small key size allows the cryptosystem to be

implemented in devices or hardware that have memory or storage constaints, such as mobile phones, PDAs (personal digital assistants) and laptops. It is easier to perform computations with smaller key size, making elliptic curve cryptosystems more computationally efficient than the RSA or the El Gamal[1]. Other benefits of using elliptic curve cryptosystems include low hardware implementation costs and high device performances due to small computations[56].

These are just some of the benefits of elliptic curve cryptography. But it is for these reasons that a number of government bodies and companies have decided to implement elliptic curve cryptosystems. Wireless technology companies such as Motorola and Qualcomm have embraced this system, while computing companies such as IBM, Sun Microsystems, Microsoft and Hewlett Packages are adopting elliptic curve technology[30]. The United States, Canada and the United Kingdom governments have already started using elliptic curve cryptography in their security systems.

One example of a security feature that implements elliptic curve cryptography is NCR's Electronic Cheque Presentation with Image Exchange, which checks that the serial numbers of cheques have not been tampered with[11]. Mozilla web browser has started a system to allow email clients to sign messages and verify the identity of the message sender using elliptic curve cryptography[19].

While the standard key size of this cryptosystem is 163-bit, the National Institute of Standards and Technology submitted a report to recommend a set of elliptic curves for federal government use with larger key sizes[37]. An example of a 192-bit curve recommended is shown in Example 3.3.1 below.

Example 3.3.1. *$E : y^2 = x^3 - 3x + B$ defined over $\mathbb{F}_p$, where*

$$B = 2455155546008943817740293915197451784769108058161191238065$$
$$p = 6277101735386680763835789423207666416083908700390324961279$$
$$\#E(\mathbb{F}_p) = n = 6277101735386680763835789423176059013767194773182842284081$$
$$P_x = 602046282375688656758213480587526111916698976636884684818$$
$$P_y = 174050332293622031404857552280219410364023488927386650641$$

Note: P_x and P_y respectively refer to the x and y coordinate of the point P on E, i.e. $P = (P_x, P_y)$.

Chapter 4

Pollard-Rho Algorithm

While the study of Cryptography seeks to develop a system or protocol for the privacy of conversation, the field of Cryptanalysis is the study of methods to break cryptographic systems.

There are two sides to Cryptanalysis. While it can be seen as antagonistic, knowing that a cryptosystem is susceptible to attacks can be an advantage. Kerckhoff's Principle says the security of a cryptosystem should depend only on the secrecy of the key space, $\mathcal{K}$; and not on the secrecy of the encryption algorithm[28]. In particular, the security of the elliptic curve cryptosystem mentioned in Chapter 3 should depend on the secrecy of the integers x and y, which are Brad's and Angelina's private keys respectively. If the integers are easily derived then the above cryptosystem will no longer be secure.

In 1978, Pollard came up with a "Monte-Carlo" method for solving the discrete logarithm problem[38]. Since then the method has been modified to solve the elliptic curve analog of the discrete logarithm problem. As the Pollard-Rho algorithm is currently the quickest algorithm to solve the Elliptic Curve Discrete Logarithm[8], so the security of the elliptic curve cryptosystem depends on the efficiency of this algorithm. Theoretically, if the Pollard-Rho algorithm is able to solve the ECDLP efficiently and in a relatively short time, then the system will be rendered insecure.

The strategy of the algorithm is to produce a sequence of randomly generated terms (R_i, a_i, b_i), where R_i is a point on the the curve E and a_i, b_i lie in $\mathbb{F}_p$, over which the elliptic curve E is defined. Since $E(\mathbb{F}_p)$ is a finite group, the sequence

eventually becomes periodic and loops back to an earlier term in the sequence. We use this periodicity to solve the ECDLP. Since the sequence does not always loop back to the first term, a diagram of the sequence looks like the Greek letter ρ (See figure 4.1). That is why this method is called the Pollard-Rho method.

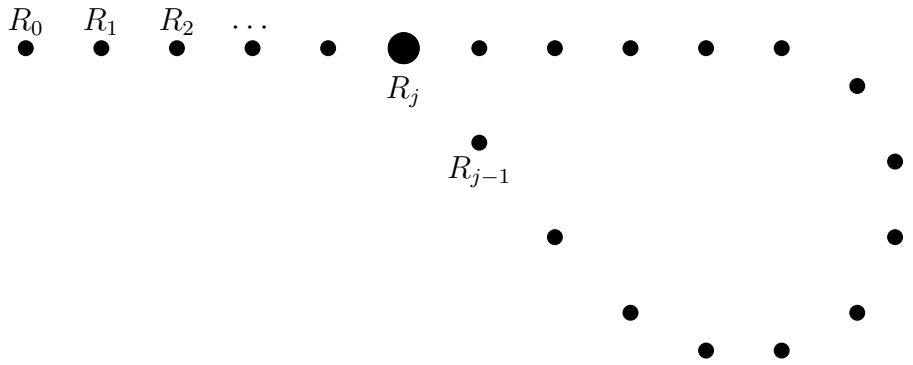


Figure 4.1: Diagram of the sequence produced by the Pollard-Rho algorithm

4.1 Pollard-Rho method for solving ECDLP

Let $G = E(\mathbb{F}_p)$, such that $|G| = n$, and P and Q such that $Q = xP$ in G. Our aim is to calculate x.

Algorithm 4.1.1. Pollard-Rho Algorithm

1. *Using a hash function, we partition G into 3 sets, S_1, S_2, S_3 of roughly the same size, but $\mathcal{O} \notin S_2$*

2. *Define an iterating function f of a random walk:*

$$R_{i+1} = f(R_i) = \begin{cases} Q + R_i, & R_i \in S_1; \\ 2R_i, & R_i \in S_2; \\ P + R_i, & R_i \in S_3 \end{cases} . \tag{4.1}$$

3. *Let $R_i = a_i P + b_i Q$, and therefore*

$$a_{i+1} = \begin{cases} a_i, & R_i \in S_1; \\ 2a_i \bmod n, & R_i \in S_2; \\ a_i + 1, & R_i \in S_3 \end{cases} \tag{4.2}$$

and

$$b_{i+1} = \begin{cases} b_i + 1, & R_i \in S_1; \\ 2b_i \ mod\ n, & R_i \in S_2; \\ b_i, & R_i \in S_3 \end{cases} \tag{4.3}$$

4. *Start with* $R_0 = P, a_0 = 1, b_0 = 0$ *and generate pairs* (R_i, R_{2i}) *until a match is found, i.e.*

$$R_m = R_{2m} \qquad for\ some\ m.$$

Once we've found a match, we have

$$R_m = a_m P + b_m Q$$
$$R_{2m} = a_{2m} P + b_{2m} Q.$$

Hence we compute x to be:

$$x = \frac{a_{2m} - a_m}{b_m - b_{2m}} \ (mod\ n). \tag{4.4}$$

Assuming that the random walk (4.1) defined in the algorithm produces random terms, the algorithm solves the elliptic curve discrete logarithm problem in $O(\sqrt{n})$ operations[38].

Now, let us illustrate how the algorithm works with an example.

Example 4.1.2. *Given*

$$E(\mathbb{F}_{47}) : y^2 = x^3 + 34x + 10$$
$$P = (30, 26)$$
$$Q = (35, 41),$$

use the Pollard-Rho algorithm to solve this elliptic curve discrete logarithm problem.

We first split $E(\mathbb{F}_{47})$ *into 3 subsets:*

$$S_1 = \{R = (x, y) \in E(\mathbb{F}_{47}) \mid 0 \leqslant y < 15\}$$
$$S_2 = \{R = (x, y) \in E(\mathbb{F}_{47}) \mid 15 \leqslant y < 30\}$$
$$S_3 = \{R = (x, y) \in E(\mathbb{F}_{47}) \mid 30 \leqslant y < 47\},$$

and calculate the group order of the curve to be $\#E(\mathbb{F}_{47}) = 41$. *When using this method of splitting the group, we find that*

$$|S_1| = 13, \qquad |S_2| = 16, \qquad |S_2| = 12.$$

The subsets are approximately equal in size and so step 1 of the algorithm is satisfied.

The initial values are $R_0 = P = (30, 26), a_0 = 1, b_0 = 0$ *and we use the mapping function (4,2) and (4.3) to generate pairs* (R_i, R_{2i}) *until a match is found.*

$$(R_1, R_2) = ((30, 26), (14, 9))$$
$$(R_2, R_4) = ((14, 9), (28, 42))$$
$$(R_3, R_6) = ((20, 18), (30, 12))$$
$$(R_4, R_8) = ((28, 42), (30, 21))$$
$$(R_5, R_{10}) = ((6, 17), (30, 21))$$
$$(R_6, R_{12}) = ((30, 21), (30, 21))$$

We have found a match between R_6 *and* R_{12} *so we stop. We have*

$$a_6 = 10,\ b_6 = 8 \qquad \textit{and} \qquad a_{12} = 5,\ b_{12} = 23$$

Let $Q = xP$ *for some integer* x. *Now using equation (4.4) to calculate* x, *we get:*

$$\frac{5 - 10}{8 - 23} \equiv \frac{-5}{-15} \equiv \frac{1}{3} \equiv 14 \ (mod\ 41).$$

Hence $Q = 14P$. □

To be able to calculate x, the denominator in (4.4) has to be invertible in $\mathbb{Z}_n$, where n is the group order of $\#E(\mathbb{F}_p)$. Unless the $\text{GCD}(b_m - b_{2m}, n) = 1$, the inverse $(b_m - b_{2m})^{-1}$ will not exist. So the Pollard-Rho algorithm does not work in some ECDLP instances. In our example above, $\#E(\mathbb{F}_{47}) = 41$ is prime, so the denominator does not equal 0 (mod 41), unless $b_m = b_{2m}$ (mod 41) allowing us to successfully solve the ECDLP.

In commercial implementations, the curve E, the underlying finite field, $\mathbb{F}_p$ and the point P are chosen such that $\#E(\mathbb{F}_p) = n$ is prime. That means that there is a high probability of success in solving the ECDLP using the Pollard-Rho

Algorithm. While choosing n to be prime increases the success of this attack, recall that the Pohlig-Hellman attack works by factoring n. Curves that are susceptible to the Pohlig-Hellman attack are deemed insecure and are unacceptable for use in commercial implementations. Therefore, for a cryptosystem to be protected against the Pohlig-Hellman attack, n should be prime.

4.2 Random Walks

As seen in Section 4.1, the Pollard-Rho method employs an iterating function of a random walk to produce a sequence of random terms. The idea behind the iterative function (4.1) in the algorithm is that the terms produced are random enough for (4.1) to be considered a random walk[38]. The coefficients a_i and b_i are chosen in a "random manner" which in turn generate a random sequence of points R_i, with the intention of finding a match[29]. For optimal performance, the walk should be a *random* random walk. In this section, we will define what a *random* random walk means, and study some theory behind a random walk on integers mod n.

We first begin with the definition of a random walk on the integers mod n.

Definition 4.2.1. *[52]* **Random walk on integers *mod* n**

Let $n \in \mathbb{N}$, and let P be a probability distribution on $\mathbb{Z}/n\mathbb{Z}$. By a random walk on the integers mod n determined by P*, we mean a sequence (X_k) given by:*

$$X_0 = 0, \quad X_{k+1} = X_k + \xi_k,\ k = 0, 1, 2, \ldots,$$

where each ξ_k is randomly chosen from $\mathbb{Z}/n\mathbb{Z}$ according to the probability distribution P.

Suppose $r \geqslant 2$ is a fixed integer. Let $p_1, p_2, \ldots, p_r$ have the properties

$$p_j > 0, \qquad j = 1, 2, \ldots, r$$

and

$$\sum_{j=1}^{r} p_j = 1.$$

Then for any set of integers $\{a_1, \ldots, a_r\} \subseteq \mathbb{Z}/n\mathbb{Z}$, let $\tilde{a} := (a_1, a_2, \ldots, a_r)$. Then for such $\tilde{a}$, we define the probability distribution $P_{\tilde{a}}$ to be

$$P_{\tilde{a}}(a) = \begin{cases} p_j, & \text{if } a = a_j \text{ for some } j \\ 0, & \text{otherwise.} \end{cases} \tag{4.5}$$

Let $\tilde{a} = (1, 2, \ldots, n)$ with probability distribution $P_{\tilde{a}}$ such that

$$P_{\tilde{a}}(a) = \begin{cases} \frac{1}{n}, & \text{if } a = a_j \text{ for some } j \\ 0, & \text{otherwise,} \end{cases}$$

i.e. $1, 2, \ldots, n$ are equally probable, then we say $P_{\tilde{a}}$ is the *uniform distribution* U.

Note: For $p_j = \frac{1}{n}$, we can easily verify that $\sum_{j=1}^{n} \frac{1}{n} = 1$.

Hence, the random walk (X_k) determined by $P_{\tilde{a}}$ is the walk obtained by

$$X_0 = 0 \qquad X_{k+1} = f(X_i), \quad k = 0, 1, 2, \ldots,$$

where $f : \mathbb{Z}/n\mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}$ is a random function.

This means that if the random walk is determined by a uniform distribution, then the walk is called a *random* random walk.

4.3 Complexity of the Algorithm

Let the group order of the elliptic curve group $E(\mathbb{F}_p)$ be n, and assume the iterative functions (4.1), (4.2) and (4.3) produce a random walk. The Pollard-Rho algorithm has a time complexity of $O(\sqrt{n})$. According to Pollard in [38], the expected running time of the algorithm is roughly

$$\sqrt{\frac{\pi n}{2}}.$$

This implies that the number of terms that must be generated until a match is found is proportional to the squareroot of the group order.

In this section, we will derive the expected running time of the Pollard-Rho algorithm.

Lemma 4.3.1. *[54] Let S be a set of n elements. Let X be the random variable for the number of elements from set S selected at random with replacement before*

any element is selected twice. Then,

$$E(X) \approx \sqrt{\frac{\pi n}{2}}.$$

Proof: Let $k = O(\sqrt{n})$. From [54], we have the probability that no match is found after selecting k elements is

$$\begin{aligned} Pr(X > k) &= \left(1 - \frac{1}{n}\right)\left(1 - \frac{2}{n}\right)\cdots\left(1 - \frac{k-1}{n}\right) \\ &\approx e^{-\frac{k^2}{2n}}. \end{aligned} \tag{4.6}$$

Using the formula for the expected value,

$$E(X) = \sum_{k=1}^{\infty} k \cdot Pr(X = k),$$

we can expand to obtain

$$\begin{aligned} E(X) &= \sum_{k=1}^{\infty} k \cdot Pr(X = k) \\ &= \sum_{k=1}^{\infty} k\left[Pr(X > k-1) - Pr(X > k)\right] \\ &= 1\left[Pr(X > 0) - Pr(X > 1)\right] + 2\left[Pr(X > 1) - Pr(X > 2)\right] \\ &\quad + 3\left[Pr(X > 2) - Pr(X > 3) + \ldots\right] \\ &= Pr(X > 0) + Pr(X > 1) + Pr(X > 2) + \ldots \\ &= \sum_{k=0}^{\infty} Pr(X > k). \end{aligned} \tag{4.7}$$

Substituting (4.5) into (4.6), we have

$$E(X) \approx \sum_{k=0}^{\infty} e^{-\frac{k^2}{2n}} \approx \int_0^{\infty} e^{-\frac{k^2}{2n}}\, dk = \sqrt{\frac{\pi n}{2}}.$$

Therefore, the expected number of elements that must be selected before any element is selected twice is

$$\sqrt{\frac{\pi n}{2}}.$$

■

How does this relate to the Pollard-Rho algorithm? The set S mentioned in the lemma is the elliptic curve group $E(\mathbb{F}_p)$ with n elements. Assuming the iterative

function (4.1) produces *random* walks, the function is "selecting a random element" from $E(\mathbb{F}_p)$ with "replacement". Furthermore, the Pollard-Rho algorithm searches for a match between the terms R_i and R_{2i}, which is the same as "selecting" an element twice.

Therefore, by this lemma, the Pollard-Rho algorithm is expected to generate $\sqrt{\frac{\pi n}{2}}$ terms before a match is found, hence solving the elliptic curve discrete logarithm problem.

Chapter 5

Experimental Setup

While the intention of the iterating function (4.1) is to produce a *random* random walk[38], however, according to Teske[50], the iterating function (4.1) of the original algorithm does not produce a *random* random walk. From Section 4.2, a *random* random walk is one that is determined by a uniform distribution. For efficiency in solving the ECDLP, we need our iterating function to produce *random* random walks. Hence, the aim of my project is to find a better iterating function that produces a more random random walk. Whilst the Pollard-Rho method can be parallelised to speed up its run-time, we are only looking for a more efficient iterative function to improve the algorithm's performance.

We choose to run the Pollard-Rho algorithm and its modified versions with better random walks on 4 classes of elliptic curve groups. Each class consists of curve groups with different numbers of digits in the group order. For practical reasons, we are not using larger groups or commercial-grade elliptic curves (See Example 3.3.1 from Section 3.3 for an example of a commercial-grade elliptic curve). The running time of the algorithm for larger curve groups gets too long to work with a sufficiently large sample size. So in our experiments, we are only using curves with a small d-digit group order, where $d = 5, 6, 7, 8$.

5.1 Sample size

Since the performance of the Pollard-Rho varies a lot over different ECDLP cases, we need a large sample size to take the variations into account. Experiments done

by Teske showed that a sample size of $N = 10,000$ for each class of curves produces relatively constant average values for the L-factor[52] (an L-factor will be defined in Section 5.3). However, it took us over 12 hours to run 1,200 ECDLP cases of the class of 7-digit prime finite field, and much longer for the class of 8-digit primes! So for practical reasons, we will only be running 1,200 ECDLP cases for each class of curves.

5.2 Generating instances of the ECDLP

Because we would like to reproduce the conditions for curves used in commercial implementations, the elliptic curves used in our experiments do not satisfy the properties of insecure curves (section 3.2) and they all have prime group orders.

One case of the Elliptic Curve Discrete Logarithm Problem consists of a curve, E, which is defined over a finite field $\mathbb{F}_p$, and two points on the curve, P and Q. We want to generate 1,200 instances of the ECDLP for the 4 classes of the curves, and the procedure for generating the cases is as follows:

1. We randomly choose an d-digit prime, p_d. Recall $d = 5, 6, 7, 8$.

2. Next, we generate A and B, with $0 \leqslant A, B < p$ such that $\#E(\mathbb{F}_{p_d})$ is prime. We also insist that E does not satisfy any of the following properties:

   (a) $4A^3 + 27B^2 \equiv 0 \pmod{p_d}$,

   (b) $\#E(\mathbb{F}_{p_d}) = p_d$ (anomalous curves), and

   (c) $\#E(\mathbb{F}_{p_d}) = p_d + 1$ (supersingular curves).

   Curves with A's and B's that satisfy any of the above properties are discarded.

3. We then generate two random points on the curve P and Q. The points P, Q and the curve group $E(\mathbb{F}_{p_m})$ form one case of the ECDLP.

4. Repeat the entire procedure till we have 1,200 cases.

5.3 Basis of Measurement

In order to compare the performance of the Pollard-Rho algorithm for the ECDLP, we need an appropriate basis of measurement. As the index of the second term in the matched pair tells us the number of iterations that the algorithm took, we will use it in our basis of measurement[29]. We measure the L-factor of the Pollard-Rho algorithm using this formula [52]:

$$L = \frac{\text{number of iterations until match is found}}{\sqrt{n}}. \tag{5.1}$$

As the expected run-time of the Pollard-Rho algorithm is $\sqrt{n}$ steps, this is a suitable number for comparison. We now need to compare it against a benchmark value, L_pr, which is the L-factor of our original Pollard-Rho algorithm.

We ran the algorithm on our 5,000 ECDLP cases, and calculated the mean L-factor to be

$$L_{pr} = 2.659.$$

This will be used as our basis of measurement. A lower L-factor will indicate good performance, while a higher L-factor will indicate otherwise. Furthermore, the lower the L-factor, the better the efficiency in solving the ECDLP.

5.4 Hash function for partitioning the group

The Pollard-Rho method requires us to split the curve group $E(\mathbb{F}_p)$ into 3 distinct sets of roughly equal sizes, although details on how to partition the group are not given in [57]. There is no straightforward method of splitting the group, and so we use a hash function.

In this context, a hash function,

$$\nu : G \longrightarrow \{1, \ldots, p\},$$

where $G = E(\mathbb{F}_p)$, is a map that sends an element of the curve group to an element of the finite field[2]. In doing so, we are just dealing with a finite field element, and hence is able to split the elliptic curve group into smaller subsets.

Using the hash function implemented by Teske in [51], we define the partition of $E(\mathbb{F}_p)$ into r subsets, $S_1, S_2, \ldots, S_r$ as follows.

1. Compute a rational approximation ϕ of $\frac{\sqrt{5}-1}{2}$ to 200 decimal places.

2. Define a function $\nu^* : E(\mathbb{F}_p) \to [0, 1)$ by

$$\nu^*(P = (x, y)) = \begin{cases} \phi y - \lfloor \phi y \rfloor, & \text{if } P \neq \mathcal{O}, \\ 0, & \text{if } P = \mathcal{O} \end{cases}$$

3. Define $\nu : E(\mathbb{F}_p) \to \{1, \ldots, r\}$ by

$$\nu(P) = \lfloor r \cdot \nu^*(P) \rfloor + 1$$

Here we have partitioned $E(\mathbb{F}_p)$ into r distinct smaller sets of roughly similar sizes. So our sets are

$$S_1 = \{P \in E(\mathbb{F}_p) \mid \nu(P) = 1\}$$
$$S_2 = \{P \in E(\mathbb{F}_p) \mid \nu(P) = 2\}$$
$$\vdots$$
$$S_r = \{P \in E(\mathbb{F}_p) \mid \nu(P) = r\},$$

where all the points with the hash value equal to 1 belong to S_1, and all the point with hash values equal to 2 belong to S_2, etc.

According to Teske[51], using $\phi = \frac{\sqrt{5}-1}{2}$, approximated to a large number of decimal places, leads to the most uniformly distributed hash values, even for non-random inputs. This is due by a theorem in [24],

Theorem 5.4.1. *Let θ be any irrational number. Let $\{x\}$ denote the fractional part of x, i.e. $\{x\} = x - \lfloor x \rfloor$. When the points $\{\theta\}$, $\{2\theta\}$, $\{3\theta\}, \ldots, \{h\theta\}$ are placed in the line segment $[0, 1]$, the points are spread out very evenly between 0 and 1.*

Therefore, since ϕ is an irrational number, the hash values calculated as above are well distributed.

Moreover, the hash function that we used here is an independent function, which means the function acts like a random function on any set of r inputs [49].

5.5 Types of Random Walks

Let us take a closer look at the original iterating function of the algorithm:

$$R_{i+1} = \begin{cases} Q + R_i, \\ 2R_i, \\ P + R_i. \end{cases}$$

Observe that it consists of two point additions and a point doubling operation. This leads to the following question: *Would the Pollard-Rho algorithm perform better, if our iterating function consists of just point additions?*

This then leads to another question: *What happens if we increase both the number of point additions and point doubling operations?*

The first type of function is defined by Teske as an r-adding walk, where r is the number of point additions. The latter is known as a $(r+q)$-mixed walk, where q is the number of point doubling operations in the iterating function.

An r-adding walk requires the elliptic curve group to be split into r smaller distinct subsets of roughly equal size while the $(r + q)$-mixed walk requires the group to be split into $r + q$ smaller subsets.

Teske implemented the algorithm for small values of r and q and Lamb ran his experiments with large values of r for the r-adding walks but with slightly larger values of r and q for the $(r + q)$-mixed walks. However, it is essential to note that only Lamb's experiments were done on elliptic curves defined over fields of the form $\mathbb{F}_{2^m}$, where m is some integer. We will be extending their experiments by running experiments for larger values of r and q. By doing so, we hope to answer the question posed by Lamb: *Do the performances of mixed walks with large r and q values improve?*

Chapter 6

Findings about the Pollard-Rho algorithm

By running the Pollard-Rho algorithm and its modified versions on the 5,000 ECDLP cases, we hope to experimentally find an iterating function that produces better random walks. This will improve the efficiency of the Pollard-Rho algorithm in solving the ECDLP.

6.1 r-Adding Walks

Definition 6.1.1. *Let $r \in \mathbb{N}$ be an even number and $P_1, \ldots, P_r$ be randomly chosen elements of our elliptic curve group $E(\mathbb{F}_p)$. Let $\nu : E(\mathbb{F}_p) \to \{1, 2, \ldots, r\}$ be our hash function. A walk (R_i) in $E(\mathbb{F}_p)$, such that $R_{i+1} = f(R_i)$ for some iterating function $f : E(\mathbb{F}_p) \to E(\mathbb{F}_p)$, is called* r**-adding** *if f is of the form:*

$$f(R_i) = \begin{cases} R_i + Q \\ \vdots \\ R_i + \frac{r}{2}Q \\ R_i + P \\ \vdots \\ R_i + \frac{r}{2}P \end{cases} \tag{6.1}$$

Adding walks use a fixed number of clauses, each of which defines a point

addition that is unique to the partition. For example, if we implement a 10-adding walk in our iterating function, the modified Pollard-Rho algorithm looks like this:

Algorithm 6.1.2. Pollard-Rho Algorithm with 10-adding walks

1. *Let $R_i = (x_i, y_i)$ and our random walk will be:*

$$R_{i+1} = f(R_i) = \begin{cases} R_i + 5Q, & \nu(P) = 1; \\ R_i + 4Q, & \nu(P) = 2; \\ \vdots & \\ R_i + Q, & \nu(P) = 5; \\ R_i + P, & \nu(P) = 6; \\ \vdots & \\ R_i + 5P, & \nu(P) = 10 \end{cases} . \tag{6.2}$$

2. *Since $R_i = a_i P + b_i Q$, then the random mappings for a_i and b_i are as follow*

$$a_{i+1} = \begin{cases} a_i, & \nu(P) = 1; \\ \vdots & \\ a_i, & \nu(P) = 5; \\ a_i + 1, & \nu(P) = 6; \\ \vdots & \\ a_i + 5, & \nu(P) = 10 \end{cases} \quad \& \quad b_{i+1} = \begin{cases} b_i + 5, & \nu(P) = 1; \\ \vdots & \\ b_i + 1, & \nu(P) = 5; \\ b_i, & \nu(P) = 6; \\ \vdots & \\ b_i, & \nu(P) = 10 \end{cases} \tag{6.3}$$

3. *Our starting values will be $R_0 = P, a_0 = 1, b_0 = 0$ and we generate pairs (R_i, R_{2i}) till a match is found.*

Note that these additions are calculated modulo n. As illustrated in the above algorithm with r-adding walks, we have r different walks with r different rules in the iterating functions, hence r different sizes of steps to take.

The aim here is to determine the value of r which gives the algorithm the best performance.

Results

Table 6.1 summarises the L-factors for Pollard-Rho method for various values of r. The percentage difference measures the performance of the r-adding walks with respect to the performance of the original algorithm. A negative number indicates improvement, while a positive number indicates otherwise. A more detailed table of the results obtained when we ran the algorithm using different values of r can be found in Appendix B.

r	**Mean L-factor**	%-difference
10	5.344	+100.98
20	3.163	+18.95
30	2.620	−1.47
40	2.413	−9.25
60	2.236	−15.91
80	2.180	−18.01
100	2.159	−18.80

Table 6.1: Performance of adding walks with the different values of r

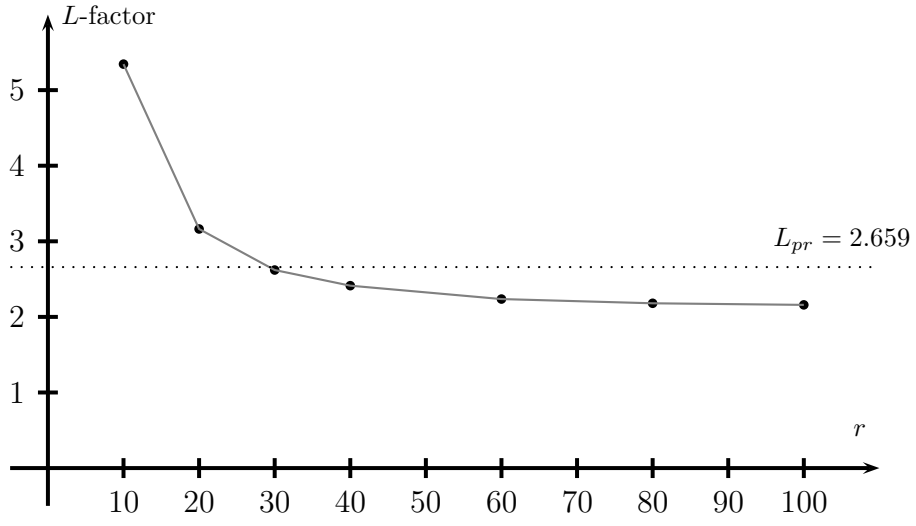


Figure 6.1: Performance of r-adding walks with different r.

Figure 6.1 shows an overall decreasing trend. Initially, the L-factors decrease sharply initially then much more slowly. This implies that the performance improves sharply at first but more gradually as r increases. This trend is also observed

in Lamb's experiments, which were performed on elliptic curves over $\mathbb{F}_{2^m}$ for some integer m. However his experiments suggest that performance of the Pollard-Rho implementation peaked at $r = 60$. This is not evidenced in our findings.

We see that the L-factors of the 10-adding walk and the 20-adding walk is higher than our benchmark value L_{pr}. As the results in Table 6.2 and Table 6.3 show, the L-factor increases as the group order gets larger for both the 10-adding walk and the 20-adding walk. This seems to imply that the Pollard-Rho algorithm using these two adding walks will perform worse than the original algorithm.

Number of digits in $E(\mathbb{F}_p)$	5	6	7	8	Mean
L-factor	3.312	4.357	5.686	8.019	5.344
% difference	+26.56	+63.86	+113.84	+201.58	+100.98

Table 6.2: Performance of modified Pollard-Rho algorithm with 10-adding walk

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L-factor	2.522	2.784	3.289	4.056	3.163
% difference	−5.15	−4.70	+23.69	+52.54	+18.95

Table 6.3: Performance of 20-adding walk

Our results seem to suggest that there will be little improvement in the performance of the algorithm if we use larger values of r. However, we can conclude that these r-adding Walks suggested by Teske behave a little like random walks when r is large enough.

6.2 $(r+q)$-Mixed Walks

Following from the previous section, we would like to investigate the performance of the algorithm when we introduce point doubling operations to our r-adding walks, where $r \geqslant 2$. The definition of a $(r+q)$-mixed walk is as follows:

Definition 6.2.1. *Let $r, q \in \mathbb{N}$ and r is an even number, and $P_1, \ldots, P_r \in E(\mathbb{F}_p)$. Let $\nu : E(\mathbb{F}_p) \to \{1, \ldots, r+q\}$ be a hash function. A walk (R_i) in $E(\mathbb{F}_p)$, with $R_{i+1} = f(R_i)$ for some iterating function $f : E(\mathbb{F}_p) \to E(\mathbb{F}_p)$, is called* $r+q$**-mixed** *if f is of the form:*

$$f(R_i) = \begin{cases} R_i + Q \\ \vdots \\ R_i + \frac{r}{2}Q \\ R_i + P \\ \vdots \\ R_i + \frac{r}{2}P \\ \left.\begin{array}{l} 2R_i \\ \vdots \\ 2R_i \end{array}\right\} q \text{ times} \end{cases} \tag{6.4}$$

The iterating function uses a mixture of point additions and doubling operations, hence the name "mixed walks"[29]. Notice that the function (4.1) of the Pollard-Rho algorithm is an example of a $(r+q)$-mixed walk, where $r = 2$ and $q = 1$.

The aim here is to find what values of r and q produce the lowest L-factor?

Results

To make sense of the results in Table 6.4, we plot a graph of the L-factor against the ratio of q to r as shown in Figure 6.2, where q is the number of point doubling operations and r is the number of point additions in our iterating function.

Generally, we found that including point doublings into our r-adding walks produces lower L-factor values. From the table, we can see that, overall, when

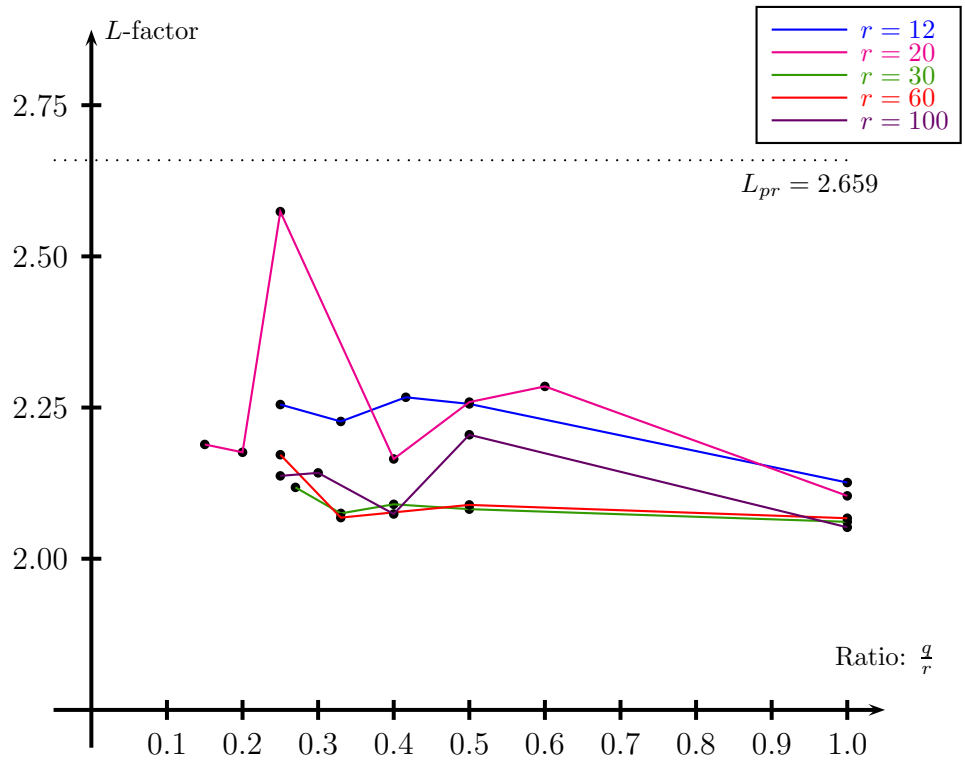


Figure 6.2: Performance of the Pollard-Rho algorithm on various values of r and q.

the ratio of q to r is 1, the L-factors tend to be lower than the L-factors of the smallest ratio. This would suggest that the more point doubling operations are introduced into the iterating function, the lower the L-factor, therefore improving the Pollard-Rho algorithm.

It appears that the lower L-factors tend to be obtained when r and q are large. So to answer Lamb's question, it would seem that larger values of r and q do improve the performance of the algorithm by a significant amount.

From the graph, we observed that the green and red line are generally lower the other coloured curves. Since the green and red line represent the $(r+q)$-mixed walk when $r = 30$ and $r = 60$ respectively, it would seem that introducing point doublings to our 30-adding walk and 60-adding walk significantly improves the performance of the Pollard-Rho algorithm.

However, the lowest L-factor occurred when $r = 100$ and $q = 100$. This means that the Pollard-Rho performs best when it has 100 point additions and 100 point doublings in its iterating function.

r	q	**Mean L-factor**	% difference
12	3	2.255	−15.19
12	4	2.227	−16.25
12	5	2.267	−14.74
12	6	2.256	−15.16
12	12	2.126	−20.05
20	3	2.189	−17.68
20	4	2.176	−18.16
20	5	2.574	−3.20
20	8	2.165	−18.58
20	10	2.259	−15.04
20	12	2.285	−14.07
20	20	2.104	−20.87
30	8	2.118	−20.35
30	10	2.075	−21.96
30	12	2.090	−21.40
30	15	2.082	−21.70
30	30	2.061	−22.49
60	15	2.172	−18.32
60	20	2.068	−22.23
60	30	2.089	−21.44
60	60	2.067	−22.26
100	25	2.137	−19.63
100	30	2.142	−19.44
100	40	2.074	−22.00
100	50	2.205	−17.07
100	100	2.052	−22.83

Table 6.4: Performance of Pollard-Rho with the various mixed walks

6.3 Summary of results

In summary, the best results from our r-adding walk occurred when $r = 100$. It would seem that the bigger the value of r, the better the performance of the

Pollard-Rho algorithm. We have also seen that when we introduce point doublings into our r-adding walk when $r = 30$ and $r = 60$, we get better random walks, although the best performance is obtained when we use our $(100 + 100)$-mixed walk.

Now if we compare the lowest L-factors from our r-adding walks and our $(r+q)$-mixed walk, we find that the L-factor of our $(100 + 100)$-mixed walk is the lowest as summarised in the table below.

	$r = 100$, $q = 0$	$r = 100$, $q = 100$
L-factor	2.159	2.052
%-difference	-18.80	-22.83

Table 6.5: Best results from our r-adding walks and from our $(r + q)$-mixed walks.

Hence, we see that our $(100 + 100)$-mixed walks improved the Pollard-Rho algorithm the most. This mixed walk is estimated to reduce the time taken to solve an ECDLP case by about 23%, while Teske's best walk is estimated to reduce the time by 21%. Thus, our mixed walk has more effect on the performance of the Pollard-Rho algorithm.

Chapter 7

Discussion

7.1 "Spikes" in the graph

As seen in Figure 6.2, there appears to be a sudden increase in the mean L-factors when we tested the mixed walk with $r = 20$ and $q = 5$. The pink line, which represents all the experiments I did on the $r = 20$ mixed walks, has a *spike*. This spike does not appear to have a theoretical explanation.

To understand why there is a spike, we look at the mean L-factors for each class of curves shown below in Table 7.1. Recall that the 5,000 ECDLP cases consists of 4 classes of curves, and each class consists of curves with a d-digit group order, where $d = 5, 6, 7, 8$.

Number of digits in $\#E(\mathbb{F}_p)$	5	6	**7**	8	Mean
L**-factor**	2.294	2.239	**3.751**	2.179	2.574

Table 7.1: Performance of $(20 + 5)$-mixed walk

From the above table, we see that the L-factor of the class of curves with 7-digit group order is much higher than the other classes, causing us to obtain a high mean L-factor. On further investigation, for one particular ECDLP case, we found that the algorithm has to generate 1,329,177 terms before a match was found, which corresponds to an L-factor of 1828.505. This is extremely high, and an obvious outlier. When we checked the L-factors of this ECDLP cases that were obtained from the other adding and mixed walks, they were similar to the mean

L-factor of the respective walks.

To investigate further, we removed the outlier and recalculated the L-factor for the 7-digit class and the mean L-factor of the $(20 + 5)$-mixed walk. The results are summarised in Table 7.2, and shown graphically in Figure 7.1

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L-factor	2.294	2.239	2.229	2.179	2.194

Table 7.2: Performance of $(20 + 5)$-mixed walk without outlier

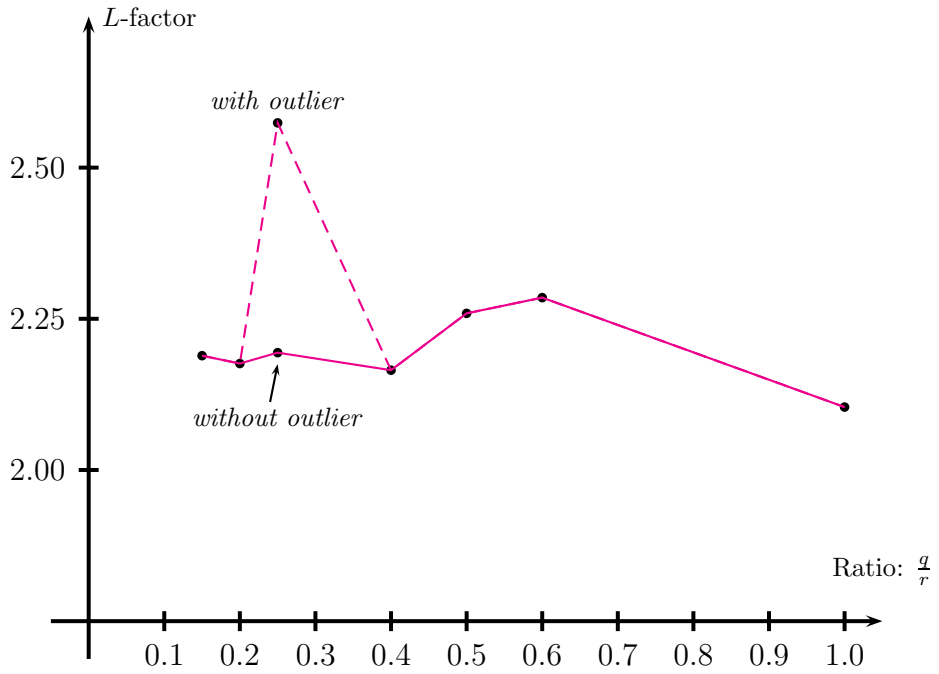


Figure 7.1: Graph of the $(20 + q)$-mixed walk for various values of q.

The resulting mean L-factor is now lower than the original L-factor, which may be more representative of the efficiency of the algorithm for this mixed walk. However, this outlier serves to remind us of the huge variation in the performance of the Pollard-Rho over different ECDLPs.

7.2 Better results with $(r+q)$-mixed walks

In this section, we will be introducing some of the theory behind *linear congruential methods* to explain the lower L-factors that we obtained from our $(r+q)$-mixed walks.

Definition 7.2.1. *A sequence (X_i) that is obtained by setting*

$$X_{i+1} = (aX_i + c) \bmod n, \qquad n \geqslant 0, \tag{7.1}$$

is called a **linear congruential sequence**, *where $X_0 \geqslant 0$ is the starting value, $a \geqslant 0$ is the multiplier, $c \geqslant 0$ is the increment, and n is the modulus such that $n > X_0$, $n > a$, $n > c$.*

Congruential sequences always have a cycle of numbers that repeats itself. The repeating cycle is known as the *period*. According to Knuth in [23], the sequence (X_i) has a shorter period when $c = 0$ than when $c \neq 0$. The relation (7.1) is a *multiplicative congruential relation* when $c = 0$, and a *mixed congruential relation* when $c \neq 0$.

Theorem 7.2.2. *The linear congruential sequence has a period of length n if and only if*

1. *c is relatively prime to n,*
2. *$b = a - 1$ is a multiple of p, for every prime p dividing n,*
3. *b is a multiple of 4, if n is a multiple of 4.*

Proof: Full proof can be found in [23], The Art of Computer Programming, Vol 2, near the bottom of page 15 after Theorem A. ■

When $c = 0$, the sequence is generated by the recurrence relations

$$X_{i+1} = aX_i \bmod n, \tag{7.2}$$

assuming $X_0 \neq 0$, otherwise we will get the zero sequence. From Theorem 7.1, we need c to be coprime to n for the period to be of length n. As $c = 0$, so the maximum period cannot be achieved. How does this apply to our $(r+q)$-mixed walks?

Recall that the iterating function of our $(r+q)$-mixed walk is of the form

$$R_{i+1} = \begin{cases} R_i + Q \\ \vdots \\ R_i + \frac{r}{2}Q \\ R_i + P \\ \vdots \\ R_i + \frac{r}{2}P \\ \left.\begin{matrix} 2R_i \\ \vdots \\ 2R_i \end{matrix}\right\} q \text{ times} \end{cases}$$

and because $R_i = a_i P + b_i Q$, so a_i and b_i satisfy the following recurrence relation

$$a_{i+1} = \begin{cases} \left.\begin{matrix} a_i \\ \vdots \\ a_i \end{matrix}\right\} r/2 \text{ times} \\ a_i + 1 \\ \vdots \\ a_i + \frac{r}{2} \\ \left.\begin{matrix} 2a_i \\ \vdots \\ 2a_i \end{matrix}\right\} q \text{ times} \end{cases} \quad \text{and} \quad b_{i+1} = \begin{cases} b_i + 1 \\ \vdots \\ b_i + \frac{r}{2} \\ \left.\begin{matrix} b_i \\ \vdots \\ b_i \end{matrix}\right\} r/2 \text{ times} \\ \left.\begin{matrix} 2b_i \\ \vdots \\ 2b_i \end{matrix}\right\} q \text{ times} \end{cases} \tag{7.3}$$

where all calculations are done modulo n, the group order of the elliptic curve group $E(\mathbb{F}_p)$.

We see that (7.3) is made up of a mixture of multiplicative and mixed congruential relations. The multiplicative relations help the walk (R_i) to get into a loop quickly since the period of the sequence is limited by Theorem 7.1. This would imply that for the Pollard-Rho algorithm to obtain a match faster, the iterating function should have more multiplicative congruential relations, i.e. the $(r+q)$-mixed walk should have a large number of point doublings.

From our experiments, we saw that we tend to obtain lower L-factors when the number of point doublings is the same as

7.3 Provably Better Walks

In this section, we will study some interesting properties of r-adding walks. The theory behind good random walks will explain the observations we made in Section 6.2.1. Furthermore, we show that we can always achieve this type of performance using r-adding walks on any elliptic curve groups.

Assume that the hash function ν is independent, recall that an independent hash function is a function that acts like a random function on any set of r inputs, we would like to generalise the random walk (R_i) of the Pollard-Rho algorithm to walks on integers *mod* n.

Firstly we generalise the r-adding walk:

$$R_{i+1} = \begin{cases} R_i + 5Q, & \nu(P) = 1; \\ R_i + 4Q, & \nu(P) = 2; \\ \qquad\vdots & \\ R_i + Q, & \nu(P) = 5; \\ R_i + P, & \nu(P) = 6; \\ \qquad\vdots & \\ R_i + 5P, & \nu(P) = 10, \end{cases} \tag{7.4}$$

where ν is our hash function that assigns a number to the point P, to

$$R_{i+1} = R_i + \alpha_{\nu(R_i)}P + \beta_{\nu(R_i)}Q, \tag{7.5}$$

where $\alpha_{\nu(R_i)}, \beta_{\nu(R_i)} = 0, 1, \ldots, n$, as the decision to add P or Q to R_i depends on $\nu(R_i)$. Also, recall that $Q = xP$ for some integer x, and so we can further reduce (7.5).

$$\begin{aligned} R_{i+1} &= R_i + \alpha_{\nu(R_i)}P + \beta_{\nu(R_i)}Q \\ &= R_i + \alpha_{\nu(R_i)}P + x\beta_{\nu(R_i)}P \\ &= R_i + (\alpha_{\nu(R_i)} + x\beta_{\nu(R_i)})P \\ &= R_i + \varepsilon_{\nu(R_i)}P, \end{aligned}$$

where $\varepsilon_{\nu(R_i)} = \alpha_{\nu(R_i)} + x\beta_{\nu(R_i)}$. Then an r-adding walk (R_i) in $E(\mathbb{F}_p)$ is given by the recurrence relation:

$$R_{i+1} = R_i + \varepsilon_{\nu(R_i)}P, \quad i = 0, 1, 2, \ldots. \tag{7.6}$$

We can further "reduce" the right side of (7.6) to just some constant times P. Recall that we defined $R_i = a_iP + b_iQ$. Then we have,

$$\begin{aligned} R_i &= a_iP + b_iQ \\ &= (a_i + xb_i)P \\ &= c_iP \end{aligned}$$

where $c_i = a_i + xb_i$, giving us

$$R_i = c_iP, \qquad \text{where } i = 0, 1, \ldots, n. \tag{7.7}$$

Now combining (7.6) and (7.7), we obtain,

$$\begin{aligned} c_{i+1} &= c_i + \varepsilon_{\nu(R_i)} \\ &= c_i + \varepsilon_{t_i}, \qquad \text{where } t_i = \nu(R_i). \end{aligned} \tag{7.8}$$

So each R_i term can be represented by (7.8), giving us the bijection

$$\mathbb{Z}/n\mathbb{Z} \ni z \longmapsto zP \in E(\mathbb{F}_p).$$

Now we have a one-to-one relation between the r-adding walks (R_i) in $E(\mathbb{F}_p)$ and the walk (c_i) on the integers *mod* n. This allows us to study the walks of the form (7.8) instead of studying r-adding walks.

In theory, these walks (c_i) that we have derived are said to be *supported by r points* if they are determined by some probability function $P_{\tilde{a}}$, where $P_{\tilde{a}}$ is as in (4.1) and r, which is the number of point additions in our r-adding walk. But our walks are determined by the hash function ν instead. However, if we add the assumption that our hash function is independent, so that it acts as a random function, then we can say that our walk (c_i) is supported by r points.

Now we would like to introduce the meaning of a walk *supported by r points*. Consider the random walk on the integers mod n as follows. Pick r values from the integers mod n, and then repeatedly choose one of these values at random.

We now have a random walk starting at 0. How long does it take for this walk to get close to being uniformly distributed on the integers mod n? This question was first posed by Diaconis in [12].

Before we answer that, we first define the convolution product P^*Q.

Definition 7.3.1. *For probability distributions P and Q on $\mathbb{Z}/n\mathbb{Z}$, we define the* **convolution product** *P^*Q to be*

$$P^*Q(a) = \sum_{b\in\mathbb{Z}/n\mathbb{Z}} P(a-b)Q(b), \quad a \in \mathbb{Z}/n\mathbb{Z}.$$

From 7.3.1, we have the following:

$$P^{*1} = P \quad \& \quad P^{*m} = P^*(P^{*(m-1)}), \quad m = 1, 2, \ldots.$$

If the random walk (X_k) is determined by the probability distribution P, then P^{*m} is the probability distribution of the walk after m steps[14]. This means P^{*m} is the probability distribution of X_m with:

$$P^{*m}(a) = P(X_m = a).$$

Now we define the distance between probability distributions P and Q on a finite group G to be

$$\|P - Q\| := \frac{1}{2}\sum_{a\in G} |P(a) - Q(a)| = \max_{A\subseteq G} |P(A) - Q(A)|.$$

This is also known as the variation distance between P and Q[52].

If Q is a uniform distribution U, then the variation distance between P and $Q = U$ is

$$\begin{aligned}\|P - U\| &= \frac{1}{2}\sum_{a\in G} |P(a) - U(a)| \\ &= \max_{A\subseteq G} |P(A) - U(A)|, \\ &= \frac{1}{2}\sum_{a\in G} \left|P(a) - \frac{1}{|G|}\right|\end{aligned}$$

as U is a uniform distribution so $U(A) = \frac{1}{|G|}$ [14].

Now we have this theorem from Hildebrand[22].

Theorem 7.3.2. *Suppose n is prime, and $r \geqslant 2$. Let $p_1, \ldots, p_r$, $\tilde{a}$ and $P_{\tilde{a}}$ be defined as above.*

Given $\varepsilon > 0$, we have

$$E\left(\|P_t^{*m} - U\|\right) < \varepsilon,$$

for $m = \lfloor \gamma n^{2/(r-1)} \rfloor$, for some constant $\gamma > 0$ and for sufficiently large n.

Proof: Please refer to Lemma 2 from [22] for the full proof of this theorem. ■

Hildebrand's theorem says that most walks supported on r points will take a constant multiple of $n^{\frac{2}{r-1}}$ steps to get close to a random random walk. Recall that a random random walk is a walk that is uniformly distributed.

For example, consider a walk on $\mathbb{Z}/n\mathbb{Z}$ that is supported on $r = 3$ points. Then by the theorem above, the walk will be close to a random random walk after γn steps for some constant γ.

To apply it to our Pollard-Rho algorithm with r-adding walks, we see that after $m = \gamma \cdot n^{\frac{2}{r-1}}$ steps, the walk is expected to be close to a *random* walk. And because it becomes a random random walk, we expect the algorithm to generate $O(\sqrt{n}) = \sqrt{\frac{\pi n}{2}}$ terms before a match is found. Let us compare these two numbers, and take the limit as n tends to infinity of the ratio

$$\frac{m}{O(\sqrt{n})} = \frac{\gamma n^{\frac{2}{r-1}}}{\sqrt{\frac{\pi n}{2}}}.$$

We find that,

$$\begin{aligned}\lim_{n\to\infty} \frac{\gamma n^{\frac{2}{r-1}}}{\sqrt{\frac{\pi n}{2}}} &= \sqrt{\frac{2}{\pi}} \lim_{n\to\infty} \gamma n^{\frac{2}{r-1} - \frac{1}{2}} \\ &= \gamma \sqrt{\frac{2}{\pi}} \lim_{n\to\infty} n^{\frac{3-r}{4r-1}}.\end{aligned}$$

Now,

$$\lim_{n\to\infty} n^{-\frac{r-3}{4r-1}} = 0,$$

if $\frac{r-3}{4r-1} > 0$, which is true if $r > 3$.

Thus, we have

$$\lim_{n\to\infty} \frac{m}{O(\sqrt{n})} = 0, \qquad \text{for } r > 3.$$

Let the part of the (c_i) walk in (7.4) that does not behave like a *random* walk refer to the walk,

$$c_i, c_2, \ldots, c_{m-1}, \qquad \text{where } m = \gamma \cdot n^{\frac{2}{r-1}},$$

since by the theorem, the walk gets close to a *random* walk after m terms. This limit tells us that with a large group order n, the number of terms generated before the walk becomes a random walk is small when compared to the number of terms that is generated before a match is found.

This implies that for sufficiently large r, the walk (c_i) produced will be a *random* random walk. Also, we will always achieve this behaviour regardless of the size of the elliptic curve group.

Hence we have shown that our r-adding walks do behave like *random* random walks, with the assumption that our hash function ν is an independent hash function.

Chapter 8

Conclusion

8.1 Summary of Thesis

Elliptic Curves

For an elliptic curve defined over a field, we have formally defined the group operation, which is the adding of points. This turns the set of points of E over $\mathbb{F}$ into an abelian group with the group operation satisfying the Group Law. The algebraic formulas used to calculate the coordinates of the sum of two points were derived by considering the different cases.

Furthermore, elliptic curves can be defined over a finite field, and because of the algebraic formulas, the group operation is also well-defined in the finite fields. We have seen that if a point is added to itself k times and we get another point on the curve, it is difficult to find out the number of times the first point was added to itself to get the second point. This is this property that is used in Elliptic Curve Cryptography.

Finally we discussed a method to calculate the number of $\mathbb{F}_p$-points on the curve E. The group order $\#E(\mathbb{F}_p)$ always lies in Hasse's interval.

Cryptography

We have introduced some of the common terminology of Cryptography, and defined a cryptosystem mathematically. Following from the definition, the procedure of a

public key cryptosystem was explained. This leads to a mathematical procedure of the El Gamal cryptosystem, whose security depends on the difficulty in solving the Discrete Logarithm Problem.

Based on the same encryption method, the Elliptic Curve analog of the El Gamal cryptosystem is then introduced. Similarly, the security is based on the difficulty in solving the Elliptic Curve Discrete Logarithm Problem. However, since the ECDLP of some classes of curves can be solved easily, those classes of curves are categorised as insecure curves. These curves should be avoided when implementing the elliptic curve cryptosystem.

Finally, some advantages of using elliptic curve cryptosystems were discussed and a list of some companies and government bodies that have started using these systems were also given.

Pollard-Rho Algorithm

One way to ensure the security of the elliptic curve cryptosystem is to test its resistance against any attack. Modifying known cryptanalytic algorithms to test its resistance is a method of doing so. The Pollard-Rho algorithm is one such algorithm that seeks to break this cryptosystem. However, it is still not effective enough, so the aim of this thesis was to improve the algorithm.

Suggestions of improving the algorithm were tested against 5,000 easier ECDLPs, due to time constraints. These suggestions included using r-adding walks and $(r + q)$-mixed walks in the iterating function of the algorithm. Teske and Lamb experimented with small values of r and q, and we have extended it by using the suggested walks with larger values of r and q. From the results obtained, it can be said that the $(r + q)$-mixed walk with $r = q = 100$ improved the algorithm the most, where r is the number of point additions and q is the number of point doublings in the algorithm's iterating function.

This modification is estimated to reduce the time needed to solve an ECDLP by about 23%, which is more than what Teske has achieved. Hence we have achieved our aim of improving the Pollard-Rho algorithm.

8.2 Avenues for future research

Effects on the performance of Pollard-Rho algorithm with q greater than r

The thesis focused on the effect of $(r+q)$-mixed walks on the algorithm. The results from the experiments suggest these mixed walks produces a more random walk, enabling the Pollard-Rho algorithm to solve the elliptic curve discrete logarithm problem more quickly. However, only the ratio of point doublings and addition ranging from 1/5 to 1 have been tested.

While Teske claimed that the performance of the algorithm gets worse if this ratio gets larger than 1, it would be of interest to see if this also applies to large values of r and q.

Effects on performance of the algorithm using doubling point operations

We have tried iterating functions using just point additions and functions that uses a mixture of point doublings and additions. Naturally, a question to ask : What kind of performance is achieved if we use an iterating function that is of the form:

$$R_{i+1} = \begin{cases} 2R_i \\ 3R_i \\ 4R_i \\ \vdots \\ vR_i \end{cases} \qquad \text{or} \qquad R_{i+1} = \begin{cases} 2R_i \\ 4R_i \\ 6R_i \\ \vdots \\ 2vR_i, \end{cases}$$

for some $v \in \mathbb{Z}$?

This iterating function contains point *"additions"* because we are adding R_i to itself more than twice.

New better random walks for the Pollard-Rho algorithm

We should consider that there is a possibility that the iterating functions suggested by Pollard and subsequently Teske do not produce random enough random walks. Perhaps a different function be used to generate these random terms, and hopefully

one that will produce *random* random walks, thus speeding up the process in solving the ECDLP.

Using the point at infinity $\mathcal{O}$ to solve the ECDLP

Suppose that while running the Pollard-Rho algorithm on an ECDLP, the i^{th} term generated is the point at infinity, i.e.

$$R_i = \mathcal{O}.$$

However, unless R_{2i} is also $\mathcal{O}$, the algorithm continues until a match is found between R_j and R_{2j} for some $j > i$.

Notice that $R_i = (a_i + b_i x)P$ and that the order of P was defined such that $kP = \mathcal{O}$. So if $R_i = \mathcal{O}$ then

$$\begin{aligned}(a_i + b_i x)P &= \mathcal{O} \\ a_i + b_i x &= k \\ x &= \frac{k - a_i}{b_i} \pmod{n},\end{aligned}$$

where n is the group order $\#E(\mathbb{F}_p)$. $b_i \neq 0 \pmod{n}$ because n is prime.

Using this, we can modify the Pollard-Rho algorithm to stop either

- after a match is found between R_j and R_{2j} for some j, or
- after the point of infinity is generated.

This approach of speeding up the Pollard-Rho only works if the point at infinity is generated before a match is obtained between the two terms. However, it may considerably reduce the number of terms that needs to be generated before the ECDLP is solved.

Parallelisation of the algorithm

A paper submitted by van Oorschot and Wiener found that by parallelising the algorithm, the run-time is reduced by a factor of b where b is the number of processors [54]. If we use our best mixed walks and perform parallel computations, we should be able to improve performance substantially.

8.3 Concluding Remark

There is speculation that once an actual quantum computer is built, the elliptic curve cryptosystem will lose its security. It has been proposed that a quantum computer is able to reduce an ECDLP from a problem whose difficulty increases exponentially to a polynomial one. This will greatly reduce the time needed to solve an ECDLP, thus breaking the cryptosystem. Despite the progress made, a working prototype is still something of the future[16].

At the moment, the Elliptic Curve Discrete Logarithm Problem has been proven to be difficult to solve. Ten years ago, Certicom issued a list of ECDLP challenges of increasing level of difficulty. The main objective of these challenges was to increase awareness and appreciation of the difficulty of the ECDLP among the cryptographic community. One of the Level I challenges, the ECCp-109 challenge, involved solving an ECDLP with a 109-bit key size. Here, this elliptic curve group $E(\mathbb{F}_p) : y^2 = x^2 + Ax + B$ has 33 digits in p and in its the group order $\#E(\mathbb{F}_p)$. The winner of this challenge used 10,000 Pentium PCs that ran continuously for 549 days[9]. It is estimated that the next challenge, the ECCp-163 is 100,000,000 times harder than the ECCp-109. The ECCp-163 challenge consists of a curve with p and the group order consisting of 50 digits each[10].

Therefore, until a quantum computer has been successfully built, it is the difficulty in solving the Elliptic Curve Discrete Logarithm Problem that gives the elliptic curve cryptosystem its high level of security.

Appendix A

Schoof's Algorithm

In this chapter, we will give a brief introduction to Schoof's Algorithm, which is a quicker algorithm to calculate the number of $\mathbb{F}_p$-points of an elliptic curve over $\mathbb{F}_p$. We need to first introduce several concepts before we summarise Schoof's algorithm.

Definition A.0.1. *The* j**-invariant** *of* E *is defined to be*

$$j = j(E) = 1728\,\frac{4A^3}{4A^3 + 27B^2},$$

where $A^3 + 27B^2 \neq 0$.

Now, we will introduce the division polynomials.

Definition A.0.2. *Define the division polynomials* $\psi_m(x, y)$ *by*

$$\begin{aligned}
\psi_0 &= 0 \\
\psi_1 &= 1 \\
\psi_2 &= 2y \\
\psi_3 &= 3x^4 + 6Ax^2 + 12Bx - A^2 \\
\psi_4 &= 4y(x^6 + 5Ax^4 + 20Bx^3 - 5A^2x^2 - 4ABx - 8B^2 - A^3) \\
\psi_{2m+1} &= \psi_{m+2}\psi_m^3 - \psi_{m-1}\psi_{m+1}^3 \\
\psi_{2m} &= \frac{\psi_m(\psi_{m+2}\psi_{m-1}^2 - \psi_{m-2}\psi_{m+1}^2)}{2y},
\end{aligned}$$

for $m \geqslant 2$.

Suppose E is an elliptic curve given by the Weierstrass equation $y^2 = x^3 + Ax + B$ and defined over a finite field $\mathbb{F}_p$, where the characteristic of the finite field is neither 2 nor 3. We also assume that E is not supersingular (i.e. $\#E(\mathbb{F}_p) \neq p+1$) and that its j-invariant $\neq 0, 1728$.

Let $\#E(\mathbb{F}_p) = (p+1) + t$, then Schoof's algorithm is as follows:

Let $S = \{2, 3, 5, \ldots, L\}$ be s set of primes such that

$$\prod_{l \in S} l > 4\sqrt{p}.$$

If $l = 2$ then $t \equiv 0 \pmod 2$ if and only if $\gcd(x^3 + Ax + B, x^p - x) \neq 1$.

For each odd prime $l \in S$,

1. Compute $p_l \equiv p \pmod l$ with $|p_l| < l/2$.

2. Compute the x-coordinate x' of

$$(x', y') = \left(x^{p^2}, y^{p^2}\right) + p_l(x, y) \bmod \psi_l,$$

where the addition here refers to the group operation of $E(\mathbb{F}_p)$, and $p_l(x, y)$ means adding (x, y) to itself p_l times.

3. For $h = 1, 2, \ldots, \frac{l-1}{2}$, compute the x-coordinate x_h of $(x_h, y_h) = j(x, y)$

(a) If $x' - x_h^p \equiv 0 \pmod{\psi_l}$, proceed to step (b). Otherwise, try the next value of h. If all values of h have been tried then proceed to step (4).

(b) Compute y' and y_h. If

$$\frac{y' - y_h}{y} \equiv 0 \pmod{\psi_l},$$

then $t \equiv h \pmod l$. Otherwise, $t \equiv -h \pmod l$.

4. If all values of h have been tried but without success, let $\omega^2 \equiv p \pmod l$. If ω doesn't exist, then $t \equiv 0 \pmod l$.

5. If $\gcd(\text{numerator}(x^p - x_\omega), \psi_l) = 1$, then $t \equiv 0 \pmod l$. If not, compute,

$$\gcd(\text{numerator}\left(\frac{y^p - y_\omega}{y}\right), \psi_l).$$

If $\gcd \neq 1$, then $t \equiv 2\omega \pmod l$, otherwise $a \equiv -2\omega \pmod l$.

Now, we have a system of congruence relations made up of $t \pmod{l}$ for each $l \in S$. By the Chinese Remainder Theorem, we can retrieve $t \pmod{\prod_{l \in S} l}$.

Hence, the group order is

$$\#E(\mathbb{F}_p) = p + 1 + t.$$

Schoof's algorithm is a quicker method than the Baby-Step-Giant-Step algorithm, and is more preferred for use in Cryptanalysis. Since then, this algorithm has been improved by Atkin and Elkies, who were Schoof's students. For more details of the SEA (Schoof-Elkies-Atkin) method, please see [42].

Appendix B

Tables of results

B.1 Results obtained for the different values of r in r-adding walks

$r = 20$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.522	2.784	3.289	4.056	3.163

Table B.1: Performance of 20-adding walk

$r = 30$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.293	2.431	2.673	3.081	2.620

Table B.2: Performance of 30-adding walk

$r = 40$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.253	2.272	2.429	2.699	2.413

Table B.3: Performance of 40-adding walk

$r = 60$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.131	2.170	2.265	2.379	2.236

Table B.4: Performance of 60-adding walk

$r = 80$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.136	2.148	2.185	2.252	2.180

Table B.5: Performance of 80-adding walk

$r = 100$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.099	2.178	2.151	2.207	2.159

Table B.6: Performance of 100-adding walk

B.2 Results obtained for the different values of r and q in mixed walks

$r = 12, q = 3$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.279	2.245	2.238	2.258	2.255

Table B.7: Performance of $(12 + 3)$-mixed walk

$r = 12, q = 4$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.210	2.218	2.238	2.242	2.227

Table B.8: Performance of $(12 + 4)$-mixed walk

$r = 12, q = 5$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.294	2.239	2.255	2.281	2.267

Table B.9: Performance of $(12 + 5)$-mixed walk

$r = 12, q = 6$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.253	2.310	2.211	2.250	2.256

Table B.10: Performance of $(12 + 6)$-mixed walk

$r = 12, q = 12$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.132	2.098	2.161	2.115	2.126

Table B.11: Performance of $(12 + 12)$-mixed walk

$r = 20, q = 3$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.237	2.159	2.204	2.156	2.189

Table B.12: Performance of $(20 + 3)$-mixed walk

$r = 20, q = 4$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.190	2.183	2.139	2.193	2.176

Table B.13: Performance of $(20 + 4)$-mixed walk

$r = 20, q = 5$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.229	2.134	3.751	2.179	2.574

Table B.14: Performance of $(20 + 5)$-mixed walk

$r = 20, q = 8$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.178	2.147	2.219	2.116	2.165

Table B.15: Performance of $(20 + 8)$-mixed walk

$r = 20, q = 10$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.243	2.281	2.269	2.244	2.259

Table B.16: Performance of $(20 + 10)$-mixed walk

$r = 20, q = 12$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.357	2.289	2.244	2.252	2.285

Table B.17: Performance of $(20 + 12)$-mixed walk

$r = 20, q = 20$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.092	2.108	2.102	2.113	2.104

Table B.18: Performance of $(20 + 20)$-mixed walk

$r = 30, q = 8$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.111	2.141	2.105	2.114	2.118

Table B.19: Performance of $(30 + 8)$-mixed walk

$r = 30, q = 10$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.081	2.087	2.020	2.111	2.075

Table B.20: Performance of $(30 + 10)$-mixed walk

$r = 30, q = 12$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.096	2.091	2.081	2.092	2.090

Table B.21: Performance of $(30 + 12)$-mixed walk

$r = 30, q = 15$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.157	2.075	2.061	2.034	2.082

Table B.22: Performance of $(30 + 15)$-mixed walk

$r = 30, q = 30$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.009	2.089	2.08	2.079	2.061

Table B.23: Performance of $(30 + 30)$-mixed walk

$r = 60, q = 15$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.077	2.038	2.471	2.105	2.172

Table B.24: Performance of $(60 + 15)$-mixed walk

$r = 60, q = 20$

Number of digits in p	5	6	7	8	Mean
L**-factor**	2.070	2.082	2.062	2.057	2.068

Table B.25: Performance of $(60 + 20)$-mixed walk

$r = 60, q = 30$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.033	2.143	2.046	2.134	2.089

Table B.26: Performance of $(60 + 30)$-mixed walk

$r = 60, q = 60$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.030	2.089	2.067	2.080	2.067

Table B.27: Performance of $(60 + 60)$-mixed walk

$r = 100, q = 25$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.202	2.098	2.117	2.130	2.137

Table B.28: Performance of $(100 + 25)$-mixed walk

$r = 100, q = 30$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.134	2.166	2.136	2.133	2.142

Table B.29: Performance of $(100 + 30)$-mixed walk

$r = 100, q = 40$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.035	2.058	2.021	2.181	2.074

Table B.30: Performance of $(100 + 40)$-mixed walk

$r = 100, q = 50$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.193	2.257	2.231	2.139	2.205

Table B.31: Performance of $(100 + 50)$-mixed walk

$r = 100, q = 100$

Number of digits in $\#E(\mathbb{F}_p)$	5	6	7	8	Mean
L**-factor**	2.050	2.019	2.092	2.047	2.052

Table B.32: Performance of $(100 + 100)$-mixed walk

Appendix C

SAGE code

SAGE, which stands for *Software for Algebra and Geometry Experimentation*, is a computer algebra software that was first started by William Stein. It is a free open souce mathematical software supports research and teaching in algebra, cryptography, geometry, number theory and other related areas.

We used this program to run our experiments in because it has customised elliptic curve and cryptography packages. This allows us to use predefined functions such as `E.cardinality()`, which computes the group order of an elliptic curve group E, and it also allows us to compute the sum of two points on the curve.

SAGE is freely available and can be downloaded from `http://www.sagemath.org`.

C.1 Generating curves with prime group order

```
#############################################################################
# Author: A. Rowley, M. Seet                                                #
#                                                                           #
# SAGE Mathematical Software, Version 2.6, http://www.sagemath.org          #
#                                                                           #
#############################################################################
```

```
def ARInputGen():

  f = open( "AR/Curves.csv", "w" )

  for Digit in range(5,9,1):   ## No of digits of p for finite field F_p ##
    for Row in range(0,10,1):
      p = next_prime(int(exp(log(10)*(random() + Digit - 1))))
      while p > 10^Digit:
        p = next_prime(int(exp(log(10)*(random() + Digit - 1))))

      order = 0
      while not(is_prime(order)) or order == p or order == p+1:
## Conditions for insecure curves ##
        A = int(random()*100)
        B = int(random()*100)
        if (4*A^3 + 27*B^2).mod(p) != 0:      ## Avoid singular curves ##
          E =EllipticCurve(GF(p),[A,B])       ## E(F_p): y^2=x^3+Ax+B  ##
          order = E.cardinality()

      ## Generating random point on curve ##

      x = FF(int((random()*p)%p))
      z = FF((x^3 + A*x + B)%p)
      while  z.is_square()!= true:
        x = FF(int((random()*p)%p))
       z = FF((x^3 + A*x + B)%p)

      y = z.sqrt()  # Return (x,y) which lies on curve #

      f.write(‘p‘+","+‘A‘+","+‘B‘+","+‘order‘+","+‘x‘+","+‘y‘+"\n")

  f.close()
  return
```

C.2 Pollard-Rho Algorithm

```
#############################################################################
# Author: Dr. D. Kohel, M. Seet                                             #
#                                                                           #
# SAGE Mathematical Software, Version 2.6, http://www.sagemath.org          #
#                                                                           #
#############################################################################

def pollard_step(Ri,ai,bi,p,P,Q,r,q,phi):

    y = ZZ(Ri[1])

    UStar = 0                           ## Hash Function
    if Ri[2] != 0:
      UStar = phi*y-floor(phi*y)
    U = int(floor(UStar * \RQ) + 1)

    m1 = int(r/2)
    m2 = int(r/2+q)

    if U <= m1:
        c = U
bi += c
Ri += c*Q                   ## same as Ri = Ri + Q
    elif U <= m2:
        c = U - m1
ai *= 2
bi *= 2
Ri *= 2                     ## same as Ri = 2*Ri
    else:
        c = U - m2
ai += c
```

```
Ri += c*P                   ## same as Ri = Ri + P
    return Ri, ai, bi

def pollard_rho(P,Q,r,q,order=0,check=True):
    phi = (RealField(2000)((sqrt(5)-1)/2))
    E = P.curve()
    if order == 0:
order = E.cardinality()
    N = order
    if check:
        assert N*P == E(0)
        assert N*Q == E(0)
    S = ZZ.quo(N)
    i = 1
    ai = S(1)
    bi = S(0)
    Ri = P
    F = E.base_field()
    p = F.cardinality()

    R2i, a2i, b2i = pollard_step(Ri,ai,bi,p,P,Q,r,q,phi)
    while True:

        ## One step for R_i (-> R_{i+1}): ##
        Ri, ai, bi = pollard_step(Ri,ai,bi,p,P,Q,r,q,phi)

        ## and two steps for R_2i (-> R_{2i+2}): ##
        R2i, a2i, b2i = pollard_step(R2i,a2i,b2i,p,P,Q,r,q,phi)

assert R2i == ZZ(a2i)*P + ZZ(b2i)*Q

        R2i, a2i, b2i = pollard_step(R2i,a2i,b2i,p,P,Q,r,q,phi)
```

```
if check:
        assert R2i == ZZ(a2i)*P + ZZ(b2i)*Q
        i += 1
if Ri[0] == R2i[0]:  ## Check if x-coords of R_i, R_2i are the same ##
    if Ri[1] != R2i[1]:   ## if y-coords are not the same,  ##
                a2i *= -1         ##    multilply R_21 by -1        ##
                b2i *= -1
            den = (bi-b2i)
            return i              ## Prints out i ##
```

C.3 Modified Pollard-Rho program to run 1,200 ECDLP cases

```
#############################################################################
# Author: A. Rowley, M. Seet                                                #
#                                                                           #
# SAGE Mathematical Software, Version 2.6, http://www.sagemath.org          #
#                                                                           #
#############################################################################

def ARPollardRho(r,q,d):

   f = open("MeLoveYou/Curves"+‘d‘+"digit.csv","r")
   o = open("MeLoveYou/Output" + ‘d‘ + "D_r" + ‘r‘ + "_q" + ‘q‘ + ".csv","w")

   line = f.readline()
   while len(line) > 0:
      items = line.split('\r')[0].split(',')

      p = int(items[0])
      A = int(items[1])
      B = int(items[2])
```

```
    x = int(items[4])
    y = int(items[5])
    u = int(items[6])
    v = int(items[7])

    FF = FiniteField(p)
    E = EllipticCurve([ FF(A), FF(B) ])

    P = E([x,y])
    Q = E([u,v])

    i = pollard_rho(P,Q,r,q)
    print(i,p)
    o.write(‘i‘ + "," + ‘p‘ + "\n")


    line = f.readline()

o.close()
f.close()
return
```

Bibliography

[1] National Security Agency. The Case for Elliptic Curves Cryptography. Webpage available at `http://www.nsa.gov/ia/industry/crypto_elliptic_curve.cfm`. Online; accessed 18-10-2007.

[2] P. Black, editor. *Hash functions*. 2007. Available from `http://www.nist.gov/dads/HTML/hash.html`. Online; accessed 31-October-2007.

[3] I. Blake, G. Seroussi, and N. Smart. *Elliptic curves in cryptography*. Cambridge University Press, 1999.

[4] I. Blake, G. Seroussi, and N. Smart, editors. *Advances in elliptic curve cryptography*. Cambridge University Press, 2005.

[5] R. Broker. Constructing Supersingular Elliptic Curves. May 2007. preprint.

[6] E. Brown. Three fermat trails to elliptic curves. *The College Mathematics Journal*, 31(3):162–172, May 2000.

[7] Certicom. The Basics of ECC. Article available at `http://www.certicom.com/index.php?action=ecc,ecc_faq`. Online; accessed 18-10-2007.

[8] Certicom. The elliptic curve cryptosystem: Remarks on the security of the elliptic curve cryptosystem. Technical report, 1997. `http://www.certicom.com`.

[9] Certicom. Certicom announces Elliptic Curve Cryptosystem challenge winner. Nov 2002. Article available at `http://www.certicom.com/index.php?action=company,press_archive&view=121`.

[10] Certicom. The Certicom ECC Challenge, 2007. Article available at `http://www.certicom.com/index.php?action=res,ecc_challenge`.

[11] Certicom. NCR cashes in with security form Certicom. 2007. Article available at `http://www.certicom.com/index.php?action=sol,success_ncr`.

[12] F. Chung, P. Diaconis, and R. Graham. Random Walks Arising in Random Number Generation. *The Annals of Probability*, 15(3):1148–1165, Jul 1987.

[13] H. Cohen and G. Frey, editors. *Handbook of elliptic and hyperelliptic curve cryptography*. Chapman and Hall/CRC, 2006.

[14] J. Dai and M. Hildebrand. Random random walks on the integers mod n. *Statistics and Probability Letters*, pages 371–379, 1997.

[15] M. Downes. Short Math Guide to LaTex, March 2002. Version 1.09 is available at `http://www.ams.org/tex/short-math-guide.html`.

[16] J. Eicher and Y. Opoku. Using the Quantum Computer to Break Elliptic Curve Cryptosystems. 1997. Summer project at University of Richmond. Article available at `www.mathcs.richmond.edu/~jad/summerwork/ellipticcurvequantum.pdf`.

[17] P. Flajolet and A. Odlyzko. Random mapping statistics. *Lecture Notes in Computer Science*, 434:329–354, 1990.

[18] K. Giuliani. Attacks on the Elliptic Curves Discrete Logarithm Problem. Master's thesis, 1999.

[19] V. Gupta, D. Stebila, and S. Shantz. Integrating elliptic curve cryptography into the web's security infrastructure.

[20] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2004.

[21] B. Harris. Probability distributions related to random mappings. *The Annals of Mathematical Statistics*, 31(4):1045–1062, Dec 1960.

[22] M. Hildebrand. Random walks supported on random points of $\mathbf{Z}/\mathbf{Z}_n$. *Probability Theory and Related Fields*, 100:191–203, 1994.

[23] D. Knuth. *The Art of Computer Programming Vol 2*. Addison-Wesley Publishing Company, 1969.

[24] D. Knuth. *The Art of Computer Programming Vol 3*. Addison-Wesley Publishing Company, 1969.

[25] N. Koblitz. *Introduction to Elliptic Curves and Modular Forms*. Springer-Verlag, 1984.

[26] N. Koblitz. Elliptic Curve Cryptosystem. *Mathematics of Computation*, 48(177):203–209, January 1987.

[27] N. Koblitz. *A course in number theory and cryptography*. Springer-Verlag, 1994.

[28] D. Kohel. Cryptography. 2007. Lecture notes are available at `http://enchidna.maths.usyd.edu.au/~kohel/res/index.html`.

[29] N. Lamb. An investigation into Pollards Rho method for attacking Elliptic Curve Cryptosystems. 2002. Article available at `http://www.cs.ualberta.ca/~nlamb/Projects.html`.

[30] K. Lauter. The Advantages of Elliptic Curve Cryptography for Wireless Security. *IEEE Wireless Communications*, Feb 2004.

[31] A. Menezes, E. Teske, and A. Weng. Weak fields for ECC.

[32] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. Electronic version available at `www.cacr.math.uwaterloo.ca/hac`.

[33] A. Menezes, S. Vanstone, and T. Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 80–89, New York, NY, USA, 1991. ACM Press.

[34] A. Miyaji. Elliptic Curves over $\mathbb{F}_p$ Suitable for Cryptosystems. In *ASIACRYPT '92: Proceedings of the Workshop on the Theory and Application*

of Cryptographic Techniques, pages 479–491, London, UK, 1993. Springer-Verlag.

[35] J. Murray and D. Smith. Getting started with PSTricks, June 2003.

[36] T. Oetiker, H. Partl, I. Hyna, and E. Schlegl. The Not So Short Introduction to LaTex 2ε, June 2007.

[37] National Institute of Standards and Technology. Recommended Elliptic Curves for Federal government use. Technical report, National Institute of Standards and Technology, Jul 1999. Report available at `http://csrc.nist.gov/csrc/fedstandards.html`.

[38] J. Pollard. Monte Carlo Methods of Index Computation (mod p). *Mathematics of Computation*, 32(143):918–924, July 1978.

[39] D. Ravenel. Elliptic Curves: what they are, why they are called elliptic, and why topologists like them, 1, Feb 2007. Wayne State University Mathematics Colloquium.

[40] H. Ruck. On the Discrete Logarithm in the Divisor Class Group of Curves. *Mathematics of Computation*, 68(226):805–806, April 1999.

[41] M. Saeki. Elliptic Curve Cryptosystem. Master's thesis, McGill University, Montreal, February 1997.

[42] R. Schoof. Elliptic Curves Over Finite Fields and the Computation of Square Roots mod p. *Mathematics of Computation*, (170):483–494, Apr 1985.

[43] R. Schoof. Counting points on elliptic curves over finite fields. *Journal de Théorie des Nombres*, 7:219–254, 1995.

[44] H. Sebag-Montefiore. *Enigma: The Battle for the Code*. Weidenfeld and Nicolson Publishing Company, 2000.

[45] J. Silverman. *The Arithmetic of Elliptic Curves, Graduate Texts in Mathematics*. Springer-Verlag, 1986.

[46] J. Silverman and J. Suzuki. Elliptic Curve Discrete Logarithms and the Index Calculus. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology*. LNCS, Springer-Verlag, 1998.

[47] N. Smart. A comparison of different finite fields for use in elliptic curve cryptosystems. Technical Report CSTR-00-007, 2000.

[48] W. Stein. SAGE mathematical software, 2007. Version 2.6, Available at `http://www.sagemath.org`.

[49] E. Teske. Personal Correspondence, May 2007, October 2007.

[50] E. Teske. Better Random Walks for Pollard's Rho Method. *Combinatorics and Optimization Research Report CORR 98-52*, 1998.

[51] E. Teske. Speeding up Pollard's Rho Method for Computing Discrete Logarithms. *Lecture Notes in Computer Science*, 1423:541–554, 1998.

[52] E. Teske. On random walks for Pollard's rho method. *Mathematics of Computation*, Feb 2000. (to appear in print).

[53] T. Thongjunthug. Elliptic Curves over $\mathbb{Q}(i)$, 2006. Honours Thesis.

[54] P. van Oorschot and M. Wiener. Parallel Collision Search with Cryptanalytic Applications. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 12(1):1–28, 1999.

[55] T. van Zandt. PSTricks: Postscript macros for generic TeX, March 1993. Available at `http://tug.org/PSTricks/`.

[56] S. Vanstone. Next generation security for wireless: elliptic curve cryptography. 2003.

[57] L. Washington. *Elliptic Curves: Number Theory and Cryptography*. Chapman and Hall/CRC, 2003.

[58] E. Weisstein. Birthday problem. From *Mathworld* – A Wolfram Web Resource. `http://mathworld.wolfram.com/BirthdayProblem.html`. Online; accessed 21-October-2007.

[59] E. Weisstein. Elliptic Curves. From *Mathworld* – A Wolfram Web Resource. http://mathworld.wolfram.com/EllipticCurve.html. Online; accessed 12-April-2007.

[60] E. Weisstein. Random Walk – 1-Dimensional. From *Mathworld* – A Wolfram Web Resource. http://mathworld.wolfram.com/RandomWalk1-Dimensional.html. Online; accessed 5-October-2007.

[61] Wikipedia. Birthday paradox, 2007. Article available at http://en.wikipedia.org/wiki/Birthday_paradox. Online; accessed 21-October-2007.

[62] Wikipedia. Cryptography, 2007. Article available at http://en.wikipedia.org/wiki/Cryptography. Online; accessed 14-April-2007.

[63] Wikipedia. Elliptic Curve Cryptography, 2007. Article available at http://en.wikipedia.org/wiki/Elliptic_Curve_Cryptography. Online; accessed 11-October-2007.

[64] Wikipedia. Kerckhoff's principle, 2007. Article available at http://en.wikipedia.org/wiki/Kerckhoffs%27_principle. Online; accessed 14-April-2007.

[65] Wikipedia. Taher Elgamal, 2007. Article available at http://en.wikipedia.org/wiki/Taher_Elgamal. Online; accessed 3-July-2007.