# Iteration Functions for Pollard's Rho on Elliptic Curve Groups

by

Katie Bishop, B.S.

A Thesis

In

Mathematics

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements of
the Degree of

MASTER OF SCIENCES

Approved

Chris Monico
Chair of Committee

Lars Christensen

Arne Ledet

Mark Sheridan
Dean of the Graduate School

May, 2016

# ACKNOWLEDGMENTS

First off, I want to give a special thank-you to my advisor Chris Monico for his time and effort spent helping me. I have learned so much under his tutelage. Also, I would like to thank Scott Horras with his help in learning how to program and his never ending support and confidence in my abilities. Lastly, I would like to thank my family for their support and encouragement to reach for my dreams. In particular, I would like to thank my parents. Without their help, support, and encouragement I would not be a graduate student currently. I appreciate all that they have done for me.

CONTENTS

# ABSTRACT

Pollard's Rho method is one way to solve the discrete log problem (DLP) in Elliptic Curve Groups. It relies on the use of an iterative mapping that behaves like a random mapping. In this paper, we will discuss some alternative iteration functions to Pollard's original suggestion. We will then compare the different iteration functions to Pollard's by examining the expected value of the number of iterations, or steps, each iteration function requires for completion in elliptic curve groups of prime order $\approx 10^4,\ 10^5,$ and $10^6$. We will compare the expected value of each iteration function to the performance expected of an iteration function chosen at random. This analysis will be performed in order to identify possible iteration functions that may perform better than Pollard's original, the random case, or both most of the time. Finally, we then describe an algorithm to compute Pollard's Rho in parallel, which dramatically decreases the run time to solve the DLP in large elliptic curve groups.

# CHAPTER 1
# INTRODUCTION

## 1.1 Public Key Cryptography and Diffie Hellman Key Exchange

Throughout history, the need to keep sensitive information secure and private has been great. In the first century B.C., Julius Caesar used ciphers to relay important military strategies to his generals that he did not want the enemy to discover. During World War II, the German military used the Enigma machine in order to communicate securely across channels they knew the allied forces were eavesdropping on. Today banks allocate numerous resources to ensure that their customer's personal and bank account information stays confidential. One way that groups such as these can communicate securely is through the use of symmetric ciphers. This class of ciphers are the cheapest to use, but they require that both parties have a shared secret key to use prior to the start of communication. Public Key Cryptography provides some solutions to the problem of key exchange.

One benefit of public key cryptography is that it allows two parties to use public channels to agree upon a key for a symmetric cipher securely. Whitfield Diffie and Martin E. Hellman created the well-known Diffie-Hellman Key Exchange, which provides an algorithm for a public key exchange [1]; it goes as follows: Two parties Alice and Bob want to agree on a key to use in a symmetric cipher.

1. They first choose a large prime p. Both the key exchange and the following communication will work in the multiplicative group, $\mathbb{F}_p^*$.

2. They agree on one $\alpha \in \mathbb{F}_p^*$.

3. Alice and Bob each privately choose random numbers, $A,\ B\ \in\ \mathbb{Z}_{p-1}$ respectively.

4. Alice will publish her public key $Y_A \equiv \alpha^A \pmod{\text{p}}$ and Bob will publish his public key $Y_B \equiv \alpha^B \pmod{\text{p}}$.

5. Lastly, for Alice and Bob to finally communicate they need to compute:

$$\begin{aligned} K &\equiv (Y_B)^A \pmod{\text{p}} \equiv \alpha^{AB} \pmod{\text{p}} \text{ and} \\ K &\equiv (Y_A)^B \pmod{\text{p}} \equiv \alpha^{AB} \pmod{\text{p}} \end{aligned}$$

respectively. $K$ will be the key that both Alice and Bob will use for encryption and decryption using a symmetric cipher.

Now if an eavesdropper Emily was listening to this exchange she would know $p$, $\alpha$, $Y_A$, and $Y_B$. So, for Emily to find the secret key $K$ that Alice and Bob are using for their communication it would suffice to solve either:

$$\begin{aligned} \alpha^X &\equiv Y_A \pmod{\text{p}} \text{ for } X \text{ and then compute } (Y_B)^X \equiv K \pmod{\text{p}} \text{ or} \\ \alpha^X &\equiv Y_B \pmod{\text{p}} \text{ for } X \text{ and then compute } (Y_A)^X \equiv K \pmod{\text{p}} . \end{aligned}$$

These problems are known as Discrete Log Problems (DLP).

Note: even though this section introduced and described the Diffie-Hellman Key Exchange in $\mathbb{F}_p^*$, it will work in any arbitrary group $G$.

Let $G$ be a group of order $n$ where $n$ is composite with prime factorization $n = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$. Then one can use Pohlig-Hellman algorithm to solve the DLP in $G$ by solving $e_i$ discrete logarithms in groups of order $p_i$ for all $i = 1, \ \ldots \ , r$. This method uses $\mathcal{O}\left(\sum_{i=1}^{r} e_i \left(\log n + \sqrt{p_i}\right)\right)$ group operations, so it is very efficient if the prime factors of $n$ are small. Since $\left|\mathbb{F}_p^*\right| = p - 1$ is composite for any prime p, then Pohlig-Hellman's reduction applies, which is why people use $\mathbb{F}_p^*$ only if $p - 1$ has one very large prime factor say $p - 1 = 2q$ or $p - 1 = 6q$ where $q$ is a prime, or something similar. If $n$ is prime, then the run time of Pohlig-Hellman is $\mathcal{O}(\sqrt{n})$. This is the same run time as Pollard's Rho, which we will be focusing on in this paper.

## 1.2 Pollard's Rho Method for Cryptanalysis

Let $G$ be a multiplicative group of prime order $p$ and let $g, \ h \in G$ such that $g$ is a generator of $G$. One way to solve the equation $g^x = h$ is Pollard's Rho Method [9].

Consider the set of triples

$$\mathcal{T} = \{(a, b, \alpha) : \ a, b \in \{0, 1, \ldots, p-1\} \quad \text{and} \quad \alpha \equiv g^a h^b\}. \tag{1.1}$$

Randomly walking through the elements in $\mathcal{T}$ until two triples $(a_i, b_i, \alpha_i)$ and $(a_j, b_j, \alpha_j)$ are found such that $\alpha_i = \alpha_j$ solves the discrete log problem $g^x = h$ with high probability. Because if $\alpha_i = \alpha_j$, then $g^{a_i} h^{b_i} = g^{a_j} h^{b_j}$, which implies that $g^{(a_i - a_j)(b_j - b_i)^{-1}} = h$ where the inverse of $b_j - b_i$ is modulo p. In other words,

$$x = (a_i - a_j)(b_j - b_i)^{-1} \pmod{\text{p}}.$$

Pollard [9] suggests that the random walk through the space $\mathcal{T}$ could be done using the "reasonably random" function

$$F(a, b, \alpha) = \begin{cases} (a + 1 \bmod \text{p}, b, g\alpha), & \text{if } f(\alpha) = 0 \\ (2a \bmod \text{p}, 2b \bmod \text{p}, \alpha^2), & \text{if } f(\alpha) = 1 \\ (a, b + 1 \bmod \text{p}, h\alpha), & \text{if } f(\alpha) = 2 \end{cases} \tag{1.2}$$

where $f : G \to \{0, 1, 2\}$ is a hashing function. In practice, one does not just walk through the space using function (1.2) and compare each new $\alpha$ to the previous ones in the sequence to find an occurrence where two are the same. This would require too much storage to be practical. Rather, one uses the following algorithm, which provides an implementation of Pollard's Rho using small constant storage.

**Algorithm 1.2.1.** *Pollard's Rho algorithm for computing discrete logarithms.*
*INPUT: A group $G$ of prime order $p$, elements $g, h \in G$ where $g$ is a generator of $G$.*
*OUTPUT: The solution to the discrete logarithm. In other words, an element $x \in \{0, 1, \ \ldots \ , p-1\}$ such that $g^x = h$ in $G$.*

1. *Set $(a_0, \ b_0, \ \alpha_0) \leftarrow (1, \ 0, \ g^{a_0} h^{b_0})$.*

2. *For $i = 1, 2, \ldots$ do the following:*

*(a) Using* $(a_{i-1},\ b_{i-1},\ \alpha_{i-1})$ *and* $(a_{2i-2},\ b_{2i-2},\ \alpha_{2i-2})$ *compute* $(a_i,\ b_i,\ \alpha_i)$ *and* $(a_{2i}\ b_{2i},\ \alpha_{2i})$ *using function (1.2), where* $(a_i, b_i, \alpha_i) = F(a_{i-1}, b_{i-1}, \alpha_{i-1})$ *and* $(a_{2i}, b_{2i}, \alpha_{2i}) = F(F(a_{2i-2}, b_{2i-2}, \alpha_{2i-2}))$

*(b) If* $\alpha_i = \alpha_{2i}$*, then do the following*

*i. Set* $r \leftarrow b_i - b_{2i} \pmod p$

*ii. If* $r = 0$*, terminate the algorithm with failure. If* $r \neq 0$ *then set* $x \leftarrow r^{-1}(a_{2i} - a_i) \pmod p$

One reason for Pollard choosing the iteration function 1.2 to use in Algorithm 1.2.1 is that it appears to be "sufficiently complicated" to be regarded as a random mapping. Pollard believed that requiring a function to be as close to random as possible would improve the run time of the algorithm. And as a consequence, the discrete logarithm using this method can be solved in $\mathcal{O}(\sqrt{p})$ [9].

## 1.3 Directed Graphs for Pollard's Rho Method

The choice of $(a_0, b_0, \alpha_0)$ in Step one of Algorithm 1.2.1 is arbitrary. In fact, Pollard's Rho algorithm could have started with any randomly chosen $a_0,\ b_0 \in G$ and $\alpha_0 = g^{a_0}h^{b_0}$. For any $\alpha_0 \in G$, Algorithm 1.2.1 defines a sequence $\alpha_1, \alpha_2, \ldots$ of elements of $G$. Since $G$ is finite, then at a certain point, say $\alpha_\mu$, the points start cycling in a cycle of length $\lambda$ as can be seen in figure 1.1. This means, that after $\mu + \lambda$ iterations, Pollard Rho's Algorithm will start producing repeated $\alpha_i$ values. i.e. $\alpha_\mu = \alpha_{\mu+\lambda},\ \alpha_{\mu+1} = \alpha_{\mu+\lambda+1},\ \alpha_{\mu+2} = \alpha_{\mu+\lambda+2},\ \ldots$. This is how the algorithm received the name Pollard's Rho.

The directed graph created by Pollard's Algorithm is clearly a $\rho$-shaped graph, where the tail consists of $\alpha_0,\ \alpha_1,\ \ldots\ ,\ \alpha_{\mu-1}$ and the cycle consists of $\alpha_\mu,\ \alpha_{\mu+1},\ \ldots\ ,\ \alpha_{\mu+\lambda-1}$.

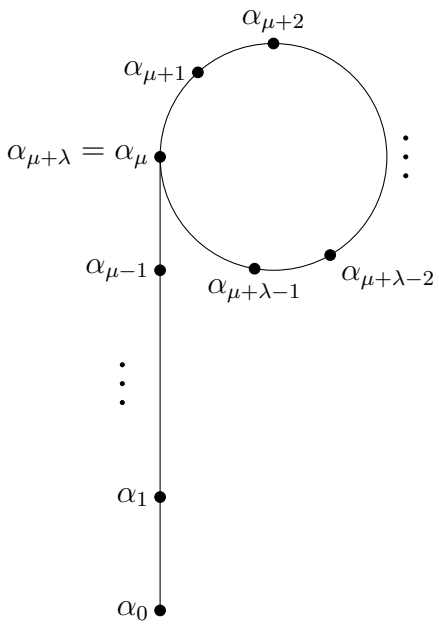


Figure 1.1: Pollard's Rho

# CHAPTER 2
# ELLIPTIC CURVE CRYPTOLOGY

Even though the Diffie-Hellman Key Exchange can be used with any group $G$ not all groups are a good choice to use in practice. The operations in $G$ should be relatively straightforward to compute since the protocol needs to be implemented in hardware or software, but complex enough for the DLP to be difficult to solve. Consider the additive group $\mathbb{Z}/p\mathbb{Z}$ whose DLP is relatively easy to solve because of the simplicity of the group operations. To solve the DLP $xg \equiv h \pmod{\text{p}}$ in $\mathbb{Z}/p\mathbb{Z}$ it is sufficient to compute $g^{-1} \pmod{\text{p}}$ and then $xgg^{-1} \equiv x \equiv hg^{-1} \pmod{\text{p}}$. Using the Binary Euclidean Algorithm to compute an inverse mod p requires at most $\frac{1}{2}(\log_2 x + 2)$ iterations where each iteration consists of additions, subtractions, and divisions by 2 (which is simply a right-shift of bits). The simplicity of a solution to the DLP in $\mathbb{Z}/p\mathbb{Z}$ shows that this is not a good candidate to use in the Diffie-Hellman Key Exchange.

The *Index Calculus* is the strongest known algorithm to solve DLP in groups, but it can only be implemented in certain groups. Let's assume $G$ is one such group. Then the Index Calculus algorithm goes as follows: Chose a relatively small subset $S$ of $G$ called a *factor base*, compute and store the logarithms of all elements of $S$ and use those values to compute the discrete logarithm of a particular element of $G$. Index Calculus can be implemented in any finite field $\mathbb{F}_q^*$ where $q$ is either a prime or a power of a prime. With an optimal choice of the subset $S$, the expected run time for $\mathbb{F}_q^*$ is $L_q\left[\frac{1}{2},\ c\right]$ where $q$ is the order of the group, and $L_q$ is defined below.

**Notation 2.0.1.** *$L_q[\alpha,\ c]$ is defined as*

$$L_q[\alpha,\ c] = exp\left((c + o(1))\,(\ln q)^{\alpha}(\ln \ln q)^{1-\alpha}\right),$$

*where $c$ and $\alpha$ are positive constants and $0 \leq \alpha \leq 1$. If $\alpha = 0$, then the function is polynomial. If $\alpha = 1$, then the algorithm has purely exponential time and if $\alpha \in (0, 1)$, then such a function is said to be sub-exponential (in $\log q$). When an algorithm has*

*a sub-exponential run time* $L_q[\alpha,\ c]$ *it implies that the algorithm is asymptotically faster than an algorithm whose run time is fully exponential in the input size, but is still slower than a polynomial time algorithm.*

Variations on Index Calculus have improved the run time for $\mathbb{F}_q^*$. The variation called *Number Field Sieve* has expected run time $L_p[\frac{1}{3},\ 1.923]$. Recent advances in [3] have shown methods to solve the DLP in finite fields of size $p^n$ with very small characteristic with complexity $L_{p^n}[\frac{1}{4},\ c]$. So, since the DLP in finite fields of the form $\mathbb{F}_q^*$ can be solved with sub-exponential time using Index Calculus, they are not optimal choices.

Elliptic Curve Groups $E(\mathbb{F}_q)$ defined over a finite field, $E(\mathbb{F}_q)$ are an attractive choice since the group operations are still relatively simple and the DLP over $E(\mathbb{F}_q)$ is significantly more difficult than the DLP over $\mathbb{F}_q^*$ of a similar size with currently known algorithms. In fact, [2] states that the key size in an Elliptic Curve Cryptosystem need only be about the cube root of the size of the key used in $\mathbb{F}_q^*$ for the same level of security. One reason for this is Index Calculus can not be implemented in well-chosen elliptic curve groups at this time. We will be focusing on Diffie-Hellman using Elliptic Curve Groups over prime fields for the rest of the paper. In principle, the ideas generalize to elliptic curves over arbitrary finite fields; we restrict to prime fields for simplicity only.

## 2.1 Elliptic Curve Groups

Let K be a field with characteristic $\neq 2$ or 3. Then an *elliptic curve group* $E(K)$ defined over a field K is the set of solutions (x,y) to an equation of the form

$$y^2 = x^3 + ax + b, \text{ where } \quad a, b \in K \quad \text{such that the discriminant} \quad 4a^3 + 27b^2 \neq 0 \tag{2.1}$$

along with an element $\mathcal{O}$ called the *point at infinity* that acts as the identity element. The group operation is defined as follows: Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two

elements in E, then

$$P+Q=\begin{cases} P, & \text{if} \quad Q=\mathcal{O} \\ Q, & \text{if} \quad P=\mathcal{O} \\ \mathcal{O}, & \text{if} \quad P=(x_1,y_1) \quad \text{and} \quad Q=(x_1,-y_1) \\ (x_3,y_3), & \text{otherwise,} \end{cases} \tag{2.2}$$

where

$$x_3=\lambda^2-x_1-x_2 \qquad \text{and} \qquad y_3=-y_1-\lambda(x_3-x_1), \tag{2.3}$$

and

$$\lambda=\begin{cases} \frac{y_2-y_1}{x_2-x_1}, & \text{if} \quad x_1\neq x_2 \\ \frac{3x^2+a}{2y_1}, & \text{otherwise.} \end{cases} \tag{2.4}$$

The operations in (2.3) and (2.4) are done in the field K. The most expensive step in the group operation is the inversions in K in (2.4) to calculate $\lambda$.

The most efficient way to compute an inverse in $K$ is using the Euclidean Algorithm as mentioned early in this chapter. An additional method to compute several inverses in parallel is Montgomery's Trick which is described in [6]. To compute $t$ inverses, Montgomery's Trick only requires $3(t-1)$ multiplications and one inversion rather than $t$ inversions separately. This is a large improvement over using the Binary Euclidean Algorithm for all $t$ inverses which requires $\frac{1}{2}(\log x+2)$ iterations per inversion. So, it would be beneficial to find a way to do the group operations in parallel in order to utilize Montgomery's trick.

Note: the order of an elliptic curve $E(\mathbb{F}_p)$ is approximately $p$ by Hasse's Theorem which states that $|E(\mathbb{F}_p)|=p+1-t$ where $|t|\leq 2\sqrt{p}$.

For the duration of this paper we will be working in the Elliptic Curve Group $E(\mathbb{F}_p)$ over a prime field $\mathbb{F}_p$ with $|E(\mathbb{F}_p)|=q$ being prime.

## 2.2 Diffie-Hellman Key Exchange in Elliptic Curve Groups

The Diffie-Hellman key exchange in Elliptic Curve Groups has a very similar protocol to what is described in Section 1.1, but for simplicity a brief overview of the key exchange using Elliptic Curve Groups will be provided here.

1. Alice and Bob agree on an Elliptic Curve Group $E(\mathbb{F}_p)$ with prime order q. They also choose a generator $P \in E(\mathbb{F}_p)$.

2. Alice and Bob each privately choose random numbers, $n_A,\ n_B\ \in\ \{0, 1,\ \ldots\ , q-1\}$ respectively.

3. Alice will publish her public key $Y_A = n_A P$ and Bob will publish his public key $Y_B = n_B P$.

4. Lastly, for Alice and Bob to finally communicate they need to compute:

$$\begin{aligned} K &\equiv n_A Y_B \equiv (n_A n_B)P \quad \text{and} \\ K &\equiv n_B Y_A \equiv (n_B n_A)P \end{aligned}$$

   respectively. $K$ will be the key that both Alice and Bob will use for encryption and decryption in a symmetric cipher.

An eavesdropper Emily would only know the values of $P$, $Y_A$, and $Y_B$. So, it would suffice to solve either $kP = Y_A$ or $kP = Y_B$ for $k$, then compute either $kY_B = K$ or $kY_A = K$ respectively.

## 2.3 Pollard's Rho for DLP in Elliptic Curve Groups

Let $P, Q \in E(\mathbb{F}_p)$, where P is a generator of $E(\mathbb{F}_p)$. Then solving the equation $kP = Q$ for k is a DLP in $E(\mathbb{F}_p)$.

The DLP $kP = Q$ can be solved with Pollard's Rho using the following iteration function:

$$F(a,b,\alpha) = \begin{cases} (a+1 \bmod \mathrm{q},\ b,\ P+\alpha), & \text{if} \quad f(\alpha)=0 \\ (2a \bmod \mathrm{q},\ 2b \bmod \mathrm{q},\ 2\alpha), & \text{if} \quad f(\alpha)=1 \\ (a,\ b+1 \bmod \mathrm{q}, Q+\alpha), & \text{if} \quad f(\alpha)=2, \end{cases} \tag{2.5}$$

where $\alpha \in E(\mathbb{F}_p)$ and $f : E(\mathbb{F}_p) \to \{0,1,2\}$ is the hashing function $f(\alpha) = x \bmod 3$ if $\alpha = (x,y)$ and $f(\mathcal{O}) = 0$.

# CHAPTER 3
# ITERATION FUNCTIONS FOR POLLARD'S RHO IN ELLIPTIC CURVE GROUPS

Pollard's Rho method described in Algorithm 1.2.1 can be generalized to use any iteration function rather than just (2.5). In fact, almost any iteration function $\psi : \mathcal{T} \to \mathcal{T}$, where $\mathcal{T}$ is the set of triples defined in (1.1) with $\alpha \in E(\mathbb{F}_p)$, will work in place of $F(a, b, \alpha)$ and the algorithm will still work.

Let $M = \{\psi : \mathcal{T} \to \mathcal{T}\}$ be the set of all possible iteration functions defined on the group $E(\mathbb{F}_p)$. Then for each pair $(\psi, \alpha_1) \in M \times E(\mathbb{F}_p)$ Algorithm 1.2.1 with iteration function $\psi$ defines a sequence $\alpha_1,\ \alpha_2,\ \ldots$ Let s be the smallest positive integer such that $\psi(\alpha_s) = \alpha_{s+1} \in \{\alpha_1,\ \alpha_2,\ \ldots\ , \alpha_s\}$. The following is motivated by Theorem 5.3.1 in [7].

**Theorem 3.1.** *Let $M$ and $s$ be defined above. For a randomly chosen $(\psi, \alpha_1) \in M \times E(\mathbb{F}_p)$ let $\mathcal{E}$ denote the expected value of $\alpha_1$. Then*

$$\limsup_{q\to\infty} \frac{\mathcal{E}}{\sqrt{2\pi q}} \leq 1.$$

*Proof.* First, for a fixed $\alpha_1 \in E(\mathbb{F}_p)$, we count how many iteration functions $\psi \in E(\mathbb{F}_p)$ there are such that $\alpha_1,\ \alpha_2,\ \ldots\ , \alpha_s$ are distinct and $\psi(\alpha_s) = \alpha_{s+1} \in \{\alpha_1,\ \alpha_2,\ \ldots\ , \alpha_s\}$. There are $q-1$ possible values for $\psi(\alpha_1) = \alpha_2$, $q-2$ values for $\psi(\alpha_2) = \alpha_3, \ldots\ , q-s+1$ possible values for $\psi(\alpha_{s-1}) = \alpha_s$. Since $\psi(\alpha_s) = \alpha_{s+1} \in \{\alpha_1,\ \alpha_2,\ \ldots\ , \alpha_s\}$, then there are s possible values for $\psi(\alpha_s) = \alpha_{s+1}$. For the other $q - s$ points $\alpha \in E(\mathbb{F}_p)$, there are q possible values for $\psi(\alpha)$. Thus, there are $(n-1)(n-2)\cdots(n-s+1)(s)(n^{n-s})$ such iteration functions for a given $\alpha_1 \in E(\mathbb{F}_p)$.

Now, since there are q possible values for $\alpha_1 \in E(\mathbb{F}_p)$, then there are $q(q-1)(q-2)\cdots(q-s+1)(s)(q^{q-s}) = \frac{q!sq^{q-s}}{(q-s)!}$ pairs $(\psi, \alpha_1)$ where s is the smallest positive integer such that $\psi(\alpha_s) = \alpha_{s+1} \in \{\alpha_1,\ \alpha_2,\ \ldots\ , \alpha_s\}$.

Since $|M| = q^q$, then $|M \times E(\mathbb{F}_p)| = q^{q+1}$. Thus, the expected value $\mathcal{E}$ of a

randomly chosen pair $(\psi, \alpha_1) \in M \times E(\mathbb{F}_p)$ is:

$$\begin{aligned}
E(s) &= \frac{1}{q^{q+1}} \sum_{s=1}^{q} s \left( \frac{q! s q^{q-s}}{(q-s)!} \right) \\
&= \frac{1}{q^{q+1}} \sum_{s=1}^{q} \frac{s^2 q! q^{q-s}}{(q-s)!} \\
&= \frac{q!}{q^{q+1}} \sum_{s=1}^{q-1} \frac{(q-s)^2 q^s}{s!} \\
&= \frac{q!}{q^{q+1}} \left[ \sum_{s=1}^{q-1} \frac{(q^2 - 2qs + s^2) q^s}{s!} \right] \\
&= \frac{q!}{q^{q+1}} \left[ q^2 \sum_{s=1}^{q-1} \frac{q^s}{s!} - 2q \sum_{s=1}^{q-1} \frac{s q^s}{s!} + \sum_{s=1}^{q-1} \frac{s^2 q^s}{s!} \right] \\
&= \frac{q!}{q^{q+1}} \left[ q^2 \sum_{s=1}^{q-1} \frac{q^s}{s!} - 2q \sum_{s=1}^{q-2} \frac{q^{s+1}}{s!} + \sum_{s=1}^{q-2} \frac{(s+1) q^{s+1}}{s!} \right] \\
&= \frac{q!}{q^{q+1}} \left[ q^2 \sum_{s=1}^{q-1} \frac{q^s}{s!} - 2q^2 \sum_{s=1}^{q-2} \frac{q^s}{s!} + q \sum_{s=1}^{q-2} \frac{s q^s}{s!} + \sum_{s=1}^{q-2} \frac{q^{s+1}}{s!} \right] \\
&= \frac{q!}{q^{q+1}} \left[ q^2 \sum_{s=1}^{q-1} \frac{q^s}{s!} - 2q^2 \sum_{s=1}^{q-2} \frac{q^s}{s!} + q \sum_{s=1}^{q-3} \frac{q^{s+1}}{s!} + q \sum_{s=1}^{q-2} \frac{q^s}{s!} \right] \\
&= \frac{q!}{q^{q+1}} \left[ q^2 \sum_{s=1}^{q-1} \frac{q^s}{s!} - 2q^2 \sum_{s=1}^{q-1} \frac{q^s}{s!} + q^2 \sum_{s=1}^{q-1} \frac{q^s}{s!} + n \sum_{s=1}^{q-1} \frac{q^s}{s!} + \frac{(2q^2 - q^2 - q) q^{q-1}}{(q-1)!} - \frac{q^2 q^{q-2}}{(q-2)!} \right] \\
&= \frac{q!}{q^{q+1}} \left[ q \sum_{s=1}^{q-1} \frac{q^s}{s!} \right] \\
&= \frac{q!}{q^q} \sum_{s=1}^{q-1} \frac{q^s}{s!} \\
&= \frac{q! e^q}{q^q} - \frac{q!}{q^q} \sum_{s=n}^{\infty} \frac{q^s}{s!} < \frac{q! e^q}{q^q}.
\end{aligned}$$

Therefore, by Stirling's Approximation,

$$\lim_{q \to \infty} \frac{\mathcal{E}}{\sqrt{2\pi q}} \leq \lim_{q \to \infty} \frac{q!}{\sqrt{2\pi q}} \left( \frac{e}{q} \right)^q = 1.$$

□

In other words, for a randomly chosen iteration function $\psi : E(\mathbb{F}_p) \to E(\mathbb{F}_p)$, the number of iterations for Pollard's Rho algorithm to find a repeated $\alpha$ is asymptotically

bounded above by $\sqrt{2\pi q}$. Furthermore, both [12] and [8] have shown that the expected value of the number of iterations could be approximated by $\sqrt{\frac{\pi q}{2}}$. In other words, $E(s)$ could be approximately $\sqrt{\frac{\pi q}{2}}$, but will not exceed the value $\sqrt{2\pi q}$.

Edyn Teske [12] proposed alternative iteration functions to Pollard's original suggestion. One is the *r-adding walk*, which is defined as follows:

**Definition 3.0.1.** *Let $r$ be a positive integer. Choose $a_1, \ldots, a_r, b_1, \ldots, b_r \in \mathbb{Z}_q$ randomly and let*

$$\mathbf{T} = \{(a_i, b_i, R_i) : a_i P + b_i Q = R_i \in E(\mathbb{F}_p) \quad for \quad 1 \leq i \leq r\}.$$

*Let $f : E(\mathbb{F}_p) \to \{0, 1, \ldots, r-1\}$ be a hash function. Then $\psi$ is an r-adding walk iteration function defined by*

$$\psi((a_j, b_j, \alpha_j)) = (a_j + a_{f(\alpha_j)},\ b_j + b_{f(\alpha_j)},\ \alpha_j + R_{f(\alpha_j)}) = (a_{j+1}, b_{j+1}, \alpha_{j+1}) \qquad (3.1)$$

*where $\alpha_j \in E(\mathbb{F}_p)$.*

For this experiment, we implemented Algorithm 1.2.1 using the following five iteration functions in Step 2 (a) with r = 15.

1. Function (2.5), which shall be denoted $\psi_P$.

2. R-adding walk with the the hash function "X mod t": For $Y \in E(\mathbb{F}_p)$, $f_M(Y) = Y_x \pmod{\text{t}}$. Denote the resulting iteration function (3.1) as $\psi_M$.

3. R-adding walk with the the hash function "X with the Minimum Difference": For $Y \in E(\mathbb{F}_p)$, $f_D(Y) = \min\limits_{i \in \{0,\ldots,r-1\}} |Y_x - (R_i)_x|$. Denote the resulting iteration function (3.1) as $\psi_D$.

4. R-adding walk with the the hash function "Minimum Slope": For $Y \in E(\mathbb{F}_p)$, $f_C(Y) = \min\limits_{i \in \{0,\ldots,r-1\}} \left|\frac{Y_y - (R_i)_y}{Y_x - (R_i)_x}\right|$. Denote the resulting iteration function (3.1) as $\psi_C$.

5. R-adding walk with the the hash function "Minimum x Value of the Sum": For $Y \in E(\mathbb{F}_p)$, $f_S(Y) = \min_{i \in \{0,\ldots,r-1\}} (Y + (R_i))_x$. Denote the resulting iteration function (3.1) as $\psi_S$.

# CHAPTER 4
# EXPERIMENT AND RESULTS

We used the C programming language to create a program that randomly finds an elliptic curve with discriminant $\neq 2, 3$ over a field $\mathbb{F}_p$, where p is specified by the user prior to running. The program returns:

1. A randomly chosen elliptic curve over $\mathbb{F}_p$.
2. Order of the elliptic curve group.
3. Number of connected components.
4. Size of each of the connected components.
5. Expected value of the number of steps until a repeated triple is found for a randomly chosen point on the elliptic curve group.

For this experiment, we used the five iteration functions defined in Chapter 3, chose three different primes p $\approx 10^4, 10^5, 10^6$, and ran the program 30 times for each p.

After gathering data 1-5 for each of the 90 curves, we calculated

$$L = \frac{\text{Expected value of number of iterations}}{\sqrt{|E(\mathbb{F}_p)|}}$$

for each curve and then averaged the values of L for the different iteration functions for each prime p. We shall denote this average as $\bar{L}$. The relevant results are located in Tables 4.1 and 4.2.

Two things that are clear in Table 4.1 and 4.2 is that $\bar{L}$ for each iteration function is $\leq \sqrt{2\pi} \approx 2.507$, just as Theorem 3.1 suggests it should be, and $\bar{L}$ and $E$ for $\psi_S$ is significantly smaller than for the other iteration functions. Pollard suggested using the function $\psi_P$ because it acts almost like a truly random mapping while still being easy to implement. For a truly random mapping, the expected value of the number

| Iteration Function | $p \approx 10^4$ (10627) | $p \approx 10^5$ (99989) | $p \approx 10^6$ (1001153) |
|---|---|---|---|
| Pollard's Rho, $\psi_P$ | 1.598 | 1.810 | 1.648 |
| x mod t, $\psi_M$ | 1.509 | 1.260 | 1.180 |
| x with the Minimum Difference, $\psi_D$ | 1.346 | 1.243 | 1.288 |
| Minimum Canonical Form, $\psi_C$ | 1.276 | 1.322 | 1.698 |
| Minimum x Value of the Sum, $\psi_S$ | 0.712 | 0.723 | 0.474 |

Table 4.1: $\bar{L}$ for each iteration function and prime p

| Iteration Function | $p \approx 10^4$ (10627) | $p \approx 10^5$ (99989) | $p \approx 10^6$ (1001153) |
|---|---|---|---|
| Pollard's Rho, $\psi_P$ | 164.8 | 572.19 | 1648.95 |
| x mod t, $\psi_M$ | 155.57 | 398.52 | 1180.16 |
| x with the Minimum Difference, $\psi_D$ | 138.81 | 393.18 | 1288.14 |
| Minimum Canonical Form, $\psi_C$ | 131.6 | 418.02 | 1698.55 |
| Minimum x Value of the Sum, $\psi_S$ | 73.36 | 228.79 | 474.15 |

Table 4.2: Average Expected Value of the Number of Iterations, $E$, for each iteration function and prime p

of iterations will be $\mathcal{O}(\sqrt{p})$, which holds true in the results of the experiment for $\psi_P$ as can be seen in 4.2. Now, maybe Pollard's suggestion of finding a function that behaves "randomly" is not the most efficient choice. $\psi_S$ seems to perform significantly better than $\psi_P$ and we suppose that this is because it behaves less randomly.

Let $(\alpha)_x$ denote the $x$-coordinate of the point $\alpha \in E(\mathbb{F}_p)$. Recall that $\psi_S$ returns a triple $(a,\ b,\ \alpha)$ such that $\alpha = (x, y)$ was the point with the smallest $x$-coordinate of the 15 sums calculated in $f_S$. Under the assumption that the $x$-coordinate of the points in $E(\mathbb{F}_p)$ are uniformly distributed, for a randomly chosen $(a_0,\ b_0,\ \alpha_0) \in \mathcal{T}$, $\psi_S(a_0,\ b_0,\ \alpha_0) = (a_1, b_1, \alpha_1)$ with $(\alpha_1)_x \leq \frac{p}{15}$ roughly 64% of the time. This is because for 15 randomly chosen points $(x, y) \in E(\mathbb{F}_p)$, the probably that at least one of them

have $x$-coordinate less than or equal to $\frac{p}{15}$ should be approximately $1 - \left(\frac{1}{15}\right)^{15} \approx .6447$; since $\psi_S$ returns the point with the smallest $x$-coordinate, than the point with the $x$ value $\leq \frac{p}{15}$, if one exists, is returned. Proceeding with a similar calculation, we discover that $(\alpha_1)_x \leq \frac{2p}{15}$ roughly 88% of the time, and that $(\alpha_1)_x \leq \frac{3p}{15} = \frac{p}{5}$ approximately 96% of the time. Now, this probability can be applied to any iteration of $\psi_S$. This means that $\psi_S$ will be iterating through the set

$$\mathbb{T} = \left\{(a, b, \alpha) : (a, b, \alpha) \in \mathcal{T} \quad \text{and} \quad (\alpha)_x \leq \frac{p}{5}\right\},$$

(which is $(\frac{1}{5})^{\text{th}}$ of the size of the set $\mathcal{T}$) 96% of the time. Now, if there exists a triple $(a_i,\ b_i\ \alpha_i)$ such that $\psi_S(a_i,\ b_i\ \alpha_i) = (a_{i+1},\ b_{i+1}\ \alpha_{i+1})$ with $(\alpha_{i+1})_x \nleq \frac{p}{5}$, then there is a 96% probability that $\psi_S(a_{i+1},\ b_{i+1}\ \alpha_{i+1}) = (a_{i+2},\ b_{i+2}\ \alpha_{i+2})$ with $(\alpha_{i+2})_x \leq \frac{p}{5}$. In other words, even if there is an iteration that maps to a triple outside of the set $\mathbb{T}$, then there is a high probability that the next iteration of $\psi_S$ will map back into the set $\mathbb{T}$. Thus, $\psi_S$ is almost solely working in a set that is $(\frac{1}{5})^{\text{th}}$ the size of the elliptic curve group $E(\mathbb{F}_p)$. Rather than the expected value being approximately $\sqrt{\frac{\pi q}{2}}$, the expected value when using the iteration function $\psi_S$ should be $\sqrt{\frac{\pi q}{10}} \approx .447\sqrt{\frac{\pi q}{2}}$.

Even though $\psi_S$ has about $\frac{1}{2}$ the expected value of the other iteration functions, each iteration is more costly. For one iteration of $\psi_S$, one needs to compute 15 point additions. Each point addition requires one modular inversion and usually only two multiplications. Montgomery's Trick can be used here to calculate the 15 inversions in parallel which would require around 75 multiplications and one inversion. So, one iteration $\psi_S$ requires roughly 1 modular inverses and 105 multiplications. Out of the iteration functions tested, $\psi_M$ is the least costly as it requires only one point addition per iteration. If the order of $E(\mathbb{F}_p)$ is around $10^6$, then this one modular inversion will require around 11 operations. So, one iteration of $\psi_M$ will require roughly 13 multiplications. So, $\psi_M$ is around 8 times faster than $\psi_S$ per iteration in an elliptic curve group with order around $10^6$, but it takes on average twice as many iterations to complete. Thus, even though $\psi_S$ decreases the number of iterations, it's clearly not the fastest choice in the traditional implementation.

## 4.1 Distinguished Points and Implementing Pollard's Rho in Parallel

Rather than solving a DLP by searching for a duplicate point in a single walk as we have been discussing so far, [6] suggests that one can solve a DLP by searching for *distinguished points* using multiple simultaneous walks. A distinguished point is a point with a distinguishing property that can be easily checked such as having certain bits of the binary representation of the point be zero. Now, to perform this collision search, there will be many, say $m$, processors used. Each processor will execute numerous walks at the same time; we will assume around 200. The following algorithm describes the protocol for a single walk on a single processor.

**Algorithm 4.1.1.** *Algorithm for implementing Pollard's Rho in Parallel.*
*INPUT: A group $G$ of prime order $p$, elements $g, h \in G$ where $g$ is a generator of $G$. An iteration function $f : \mathcal{T} \to \mathcal{T}$. A list $D$ of distinguished points in $G$. A positive integer $\theta$ that specifies how long the maximum length a walk should be.*
*OUTPUT:The solution to the discrete logarithm. In other words, an element $x \in G$ such that $g^x = h$ in $G$.*
*A single walk is described as follows:*

1. *Do while DLP is not solved*
   (a) *Randomly select $(a_0,\ b_0,\ \alpha_0) \in \mathcal{T}$.*
   (b) *For $i = 1, 2, \ \ldots\ , \theta$ do the following:*
      i. *Using $(a_{i-1},\ b_{i-1},\ \alpha_{i-1})$ compute $(a_i,\ b_i,\ \alpha_i)$ where $(a_i, b_i, \alpha_i) = f(a_{i-1}, b_{i-1}, \alpha_{i-1})$*
      ii. *If $\alpha_i \in D$, then do the following*
         A. *Send $(a_i,\ b_i,\ \alpha_i)$ to central location where the distinguished points are being stored.*
         B. *If $\alpha_i$ is a repeated distinguished point, then end - DLP is solved (central location solves the DLP using the two triples with repeated distinguished points - as described in Algorithm 1.2.1).*

*iii. If $i = \theta$ go to (a).*

For one processor to execute 200 walks at the same time, it performs the walk described in Algorithm 4.1.1 200 times simultaneously and when a modular inverse is required the processor calculates all 200 inverses at the same time using Montgomery's Trick. In this algorithm, the reason there is an upper bound on how many iterations can be done in a single walk is that there could end up being a single directed graph that has no distinguished points, so it will stop a processor from having a walk that goes on indefinitely.

Recall that [12] and [8] states that a collision will be found in approximately $\sqrt{\frac{\pi q}{2}}$ iterations. So, each of the $m$ processors should only require $\sqrt{\frac{\pi q}{2}}/(200m) \approx \frac{.005}{m}\sqrt{\frac{\pi n}{2}}$ steps in parallel. This is because if two processors (or one processor in two walks) have a collision, then they will be proceeding down the same walk there on after, so then they will both end up hitting the same distinguished point along the walk.

Now, if one ensures that some of the distinguished points have $x$-coordinate $\leq \frac{p}{5}$ and uses the iteration function $\psi_S$, then each of the $m$ processors require approximately $\frac{.002}{m}\sqrt{\frac{\pi q}{2}}$ iterations in parallel. This is because $\psi_S$ should half the number of iterations required before a collision occurs (as we saw in the previous section). Furthermore, at each iteration the processor will have to compute $15(200) = 3000$ point additions in parallel (where each point addition costs one inversion and two multiplications). One can use Montgomery's Trick to calculate the 3000 inverses modulo $p$ needed for the point additions simultaneously. Using this method, the 3000 modular inversions will cost $3(3000 - 1) = 8997$ multiplications and the 1 inversion, so each processor will require around $8997 + 2(15)(200) = 14997$ multiplications and 1 inversion to compute one iteration for all 200 walks simultaneously. If one were to implement 200 iterations on a single walk using traditional Pollard's Rho (not in parallel) with the iterating function $\psi_S$ it would require $105(200) = 21000$ multiplications and 200 inversions, so implementing a paralleled version of Pollard's Rho using $\psi_S$ is more efficient for the same number of iterations performed than traditional Pollard's

Rho.

Is implementing Pollard's Rho in parallel more efficient when using $\psi_S$ or another iterating function? As we have seen, using the function $\psi_S$ on $m$ processors requires approximately $\frac{.002}{m}\sqrt{\frac{\pi q}{2}}$ iterations where each iteration costs 14997 multiplications and 1 modular inversions per processor. Using the iteration function $\psi_M$ instead will require approximately $\frac{.005}{m}\sqrt{\frac{\pi n}{2}}$ iterations where each iteration requires one point addition per walk. So, two multiplications and one modular inversion per walk or 400 multiplications and 200 inversions per iteration for each processor. Using Montgomery's Trick, this reduces to $3(200-1) = 997$ multiplications and 1 inversion per iteration for a single processor processor. Taking into account the increased number of iterations $\psi_M$ will require, this still makes $\psi_M$ roughly 8 times faster than $\psi_S$. Thus, even when implementing Pollard's Rho in parallel, the decrease in the number of iterations for $\psi_S$ will be outweighed by the cost per iteration compared to using a iteration function cheaper to implement.

# CHAPTER 5
# CONCLUSION

We discussed and tested alternative iteration functions to the traditional Pollard's Rho. Following the experiment and analysis of a handful of iteration functions, we discovered that the iteration function $\psi_S$ reduces the number of iterations required to complete Pollard's Rho by half. After analyzing the cost per iteration, we discovered that $\psi_S$ actually behaves worse asymptotically than the other iteration functions tested. We then analyzed a parallel implementation of Pollard's Rho and again observed that $\psi_S$ still halves the number of iterations required. However, due to the number of operations each iteration requires, it is still not the most efficient iteration function to implement in practice.

# BIBLIOGRAPHY

[1] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22(6):644–654, 1976.

[2] G. Seroussi I.F. Blake and N.P. Smart. *Elliptic Curves in Cryptology*, volume 265 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 2000. Reprint of the 1999 original.

[3] Antoine Joux. A new index calculus algorithm with complexity $l(1/4 + o(1))$ in very small characteristic. Cryptology ePrint Archive, Report 2013/095, 2013.

[4] Neal Koblitz. Elliptic curve cryptosystems. *Math. Comp.*, 48(177):203–209, 1987.

[5] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. CRC Press Series on Discrete Mathematics and its Applications. CRC Press, Boca Raton, FL, 1997. With a foreword by Ronald L. Rivest.

[6] Pradeep Kumar Mishra and Palash Sarkar. Inversion of several field elements: A new parallel algorithm. Cryptology ePrint Archive, Report 2003/264, 2003.

[7] Chris Monico. Advanced problems: Cryptology. Lecture Notes, 2015.

[8] Paul C. Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 2013.

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

[10] A. Scholz. Aufgabe 253. *Jahresbericht Deutsch. Math. Verein*, 47:41–42, 1937.

[11] Harold M. Stark. *An introduction to number theory*. Markham Pub. Co., 1970.

[12] Edlyn Teske. On random walks for pollard's rho method. *Mathematics of Computation*, 70(234):809826, 2000.