On the Efficiency of Pollard's Rho Method for Discrete Logarithms

Shi Bai[1] **Richard P. Brent**[2] †

[1] Department of Computer Science,
Australian National University,
Canberra, ACT 0200
Email: shih.bai@gmail.com

[2] Centre for Mathematics and its Applications,
Mathematical Sciences Institute,
Australian National University,
Canberra, ACT 0200
Email: cats@rpbrent.com

Abstract

Pollard's rho method is a randomized algorithm for computing discrete logarithms. It works by defining a pseudo-random sequence and then detecting a match in the sequence. Many improvements have been proposed, while few evaluation results and efficiency suggestions have been reported. This paper is devoted to a detailed study of the efficiency issues in Pollard's rho method. We describe an empirical performance analysis of several widely applied algorithms. This should provide a better combination of algorithms and a good choice of parameters for Pollard's rho method.

Keywords: Pollard's rho method, discrete logarithm, elliptic curve discrete logarithm.

1 Introduction

The discrete logarithm is an analogue of the ordinary logarithm in a finite abelian group. Let H be a finite abelian group with the group operation $\otimes$. G is a cyclic subgroup of H generated by g, denoted as $\langle g \rangle = G$. Then an instance of the discrete logarithm problem (DLP) is stated as follow.

Definition 1.1 (DLP)**.** Given $h, g \in G$ known, DLP is to find the smallest non-negative integer x such that,

$$h = \underbrace{g \otimes g \otimes \cdots \otimes g}_{x \text{ times}}$$

As each element $h \in G$ can be expressed in the form of $h = g \otimes g \otimes \cdots \otimes g$, such x exists and is unique modulo $|G|$. By analogy to the ordinary logarithm, we write $x = \log_g h$. We also simplify the equation $h = g \otimes g \otimes \cdots \otimes g$ by writing $h = g^x$.

The discrete logarithm problem is believed to be hard, without any known efficient algorithm in the general case. Here an efficient algorithm means an algorithm with polynomial bit-complexity. The presumed hardness of DLP is relevant to many cryptosystems and cryptographic protocols such as Diffie-Hellman key exchange protocol (Diffie & Hellman 1976), ElGamal encryption (Gamal 1985), Digital Signature Algorithm (DSA) and Elliptic Curve DSA. Therefore algorithms for computing discrete logarithms are of great academic and practical importance.

Not all discrete logarithm problems are difficult. They may be trivial in some groups. The difficulty of the discrete logarithm problem depends on the representation of the group. Two popular finite groups used for discrete logarithm problems are the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^*$ of integers modulo a prime p and the group of points on an elliptic curve over a finite field, denoted by $E(\mathbb{F}_p)$. In these groups, no polynomial time algorithm for the problem has been reported in the literature.

Pollard's rho method (Pollard 1978) is a randomized algorithm for computing the discrete logarithm. It generates a pseudo-random sequence by an iteration function $Y_{i+1} = f(Y_i)$ in a finite abelian group. Because the order of the group is finite, the sequence will ultimately meet an element that has occurred before. This is called a collision or a match, which can be found by Floyd's collision-detection (cycle-finding) algorithm. Under the assumption that $f : G \to G$ behaves like a truly random mapping, the expected number of evaluations before a match appears is $\sqrt{\pi|G|/2}$, which is fully exponential in the problem size. The space requirement is negligible. In some cases, such as the elliptic curve discrete logarithm problem (ECDLP), Pollard's rho method is the fastest algorithm currently available. Although there exist sub-exponential time algorithms for discrete logarithm problems in the group $(\mathbb{Z}/p\mathbb{Z})^*$ such as the index calculus method (Coppersmith et al. 1986), Pollard's rho method is still of practical interest because of its simplicity and effectiveness for smaller groups. In addition, it does not exploit any special properties of the groups, making it potentially applicable to DLPs in other abelian groups.

The rest of the paper is organized as follows. Section 2 presents a comprehensive analysis of Pollard's rho method and its variants in two aspects: iteration functions and collision-detection algorithms. We also compare the performance of different iteration functions and collision-detection algorithms. In Section 3, we fill some gaps in the previous literature, suggest a good choice of parameters and give an empirical analysis of the performance.

2 Background

In this section we introduce Pollard's rho method and discuss the current status of research on iteration functions and cycle-finding algorithms.

† The work of the second author was supported by the Australian Research Council.

2.1 Pollard's Rho Method

Pollard proposed an elegant algorithm (Pollard 1978) for the discrete logarithm problem based on a Monte Carlo idea and called it the rho method. The rho method works by first defining a sequence of elements that will be periodically recurrent, then looking for a match in the sequence. The match will lead to a solution of the discrete logarithm problem with high probability. The two key ideas involved are the iteration function for generating the sequence and the cycle-finding algorithm for detecting a match.

2.1.1 Pollard's Iteration Function

We first introduce the definition of the iteration function applied in the rho method.

Definition 2.1 (Iteration Function)**.** An iteration function on a set X is a mapping $f : X \to X$.

In Pollard's paper, DLPs in $(\mathbb{Z}/p\mathbb{Z})^*$ are considered where p is a prime. Let g be a generator of the cyclic group $G = (\mathbb{Z}/p\mathbb{Z})^*$. Another element $h \in G$ is given. The discrete logarithm problem is to compute x satisfying $g^x \equiv h \pmod p$. Pollard's iteration function $f_P : G \to G$ is defined as follows,

$$f_P(Y) \equiv \begin{cases} g \cdot Y & (\text{mod } p) \quad Y \in G_1 \\ Y^2 & (\text{mod } p) \quad Y \in G_2 \\ h \cdot Y & (\text{mod } p) \quad Y \in G_3 \end{cases} \tag{2.1}$$

In each iteration of $Y_{i+1} = f_P(Y_i)$, the function uses one of three rules depending on the value of Y_i. The group G is partitioned into three sets G_1, G_2, G_3 with similar sizes, not necessarily subgroups. Each Y_i has the form $g^{a_i}h^{b_i}$. If it happens that $Y_k \equiv Y_j \pmod p$, then $g^{a_k}h^{b_k} \equiv g^{a_j}h^{b_j} \pmod p$. We can often solve the DLP if a_k, a_j, b_k, b_j are known. The sequence (a_i) (and similarly for (b_i)) can be computed using[1],

$$a_{i+1} \equiv \begin{cases} a_i + 1 & (\text{mod } |G|) \quad Y_i \in G_1 \\ 2a_i & (\text{mod } |G|) \quad Y_i \in G_2 \\ a_i & (\text{mod } |G|) \quad Y_i \in G_3 \end{cases} \tag{2.2}$$

Since G is finite, the sequence (Y_i) produced by the iteration function is periodic. Therefore there exist two smallest integers μ and λ ($\mu \geq 0$, $\lambda \geq 1$) such that $Y_k = Y_{k+\lambda}$ for every $k \geqslant \mu$. To analyze the performance of the rho method, we use the following theorem,

Theorem 2.2 (Harris (1960))**.** *Under the assumption that an iteration function $f : G \to G$ behaves like a truly random mapping, the expected values for μ and λ are $\sqrt{\pi|G|/8} \approx 0.63\sqrt{|G|}$. The expected number of evaluations before a match appears is $E(\mu + \lambda) = \sqrt{\pi|G|/2} \approx 1.25\sqrt{|G|}$, provided that all elements are saved, which requires $\sqrt{\pi|G|/2}$ space.*

2.1.2 Reported Performance

Theorem 2.2 makes the assumption of true randomness. However, it has been shown empirically that this assumption does not hold exactly for Pollard's iteration function (Teske 1998). The actual performance is worse than the expected value given in Theorem 2.2. As it is impractical to find the exact value of $\mu + \lambda$ for Pollard's iteration function, a collision-detection algorithm is often applied in practice, needing I iterations. To analyze the performance of the iteration function, we adopt the idea of delay factor $\delta = I/E(\mu + \lambda)$ used in (Teske 1998). The values of δ and I for Pollard's iteration function have been reported and we divide I by δ to get $E(\mu + \lambda)$. The performance is summarized as follows. In groups $(\mathbb{Z}/p\mathbb{Z})^*$, Pollard's iteration function has an average value of $E(\mu+\lambda) \approx 1.37\sqrt{|G|}$. The reported $E(\mu+\lambda)$ for prime order subgroups of $(\mathbb{Z}/p\mathbb{Z})^*$ is $1.55\sqrt{|G|}$ and $1.60\sqrt{|G|}$ for prime order subgroups of of $E(\mathbb{F}_p)$.

[1] Initially $Y_0 = 1, a_0 = 0, b_0 = 0$.

2.1.3 Floyd's Cycle-finding Algorithm

In order to minimise the storage requirement, a collision-detection algorithm can be applied with a small penalty in the running time. Collision-detection algorithms do not exploit the group structure and are generic. In Pollard's paper, Floyd's algorithm is applied. It compares each pair of Y_i and Y_{2i} for $i \geqslant 1$. Floyd's algorithm is based on the following fact.

Theorem 2.3 (Knuth (1997))**.** *For a periodic sequence $Y_0, Y_1, Y_2 \cdots$, there exists an $i > 0$ such that $Y_i = Y_{2i}$ and the smallest such i lies in the range $\mu \leqslant i \leqslant \mu + \lambda$.*

Floyd's algorithm uses only a small constant amount of storage. The best running-time requires μ iterations and the worst takes $\mu+\lambda$ iterations. Under the assumption that $f : G \to G$ behaves like a truly random mapping, the expected number of iterations before reaching a match is $\sqrt{\pi^5|G|/288} \approx 1.03\sqrt{|G|}$. In Floyd's algorithm, there are three evaluations and one comparison in each iteration. Hence on average there are $1.03\sqrt{|G|}$ comparisons and $3.09\sqrt{|G|}$ evaluations.

2.2 Advances in Iteration Functions

In this subsection, we consider some recent advances and developments in iteration functions.

2.2.1 Pollard's Generalized Function

We slightly change the rules defining the function. Let $M = g^m$ and $N = h^n$ where m, n are two random elements chosen from $[1, |G|]$, denoted as m, $n \in_R [1, |G|]$. We partition G into 3 sets G_1, G_2, G_3 with similar sizes. Let $f_{PG} : G \to G$ be a mapping,

$$f_{PG}(Y) \equiv \begin{cases} M \cdot Y & (\text{mod } p) \quad Y \in G_1 \\ Y^2 & (\text{mod } p) \quad Y \in G_2 \\ N \cdot Y & (\text{mod } p) \quad Y \in G_3 \end{cases} \tag{2.3}$$

Teske (1998) found that the variance of the performance in Pollard's generalized walk (or iteration function) is smaller than that for Pollard's original function. Therefore this function can be regarded as a controlled version of Pollard's original walk (Teske 1998). The reported $E(\mu + \lambda)$ is $1.62\sqrt{|G|}$ for subgroups of $E(\mathbb{F}_p)$. We cannot find reported results for groups $(\mathbb{Z}/p\mathbb{Z})^*$ and hence we will fill this gap in Section 3.

2.2.2 Teske's Adding-walk

Teske (1998) proposed a better iteration function by applying more arbitrary multipliers. Assume that we are using r partitions (multipliers). We generate $2r$ random numbers,

$$m_i, n_i \in_R \{1, 2, \cdots |G|\}, \text{ for } i = 1, 2, \cdots, r \quad (2.4)$$

Then we precompute r multipliers $M_1, M_2, \cdots M_r$ where,

$$M_i = g^{m_i} \cdot h^{n_i}, \text{ for } i = 1, 2, \cdots, r \quad (2.5)$$

Define a hash function,

$$v : G \to \{1, 2, \cdots r\} \quad (2.6)$$

This completes the precompute stage. Then the iteration function $f_{TA} : G \to G$ is,

$$f_{TA}(Y) = Y \cdot M_{v(Y)}, \text{ where } v(Y) \in \{1, 2, \cdots r\} \quad (2.7)$$

The indices are updated by,

$$\begin{aligned} a_{i+1} &= a_i + m_{v(Y_i)} \\ b_{i+1} &= b_i + n_{v(Y_i)} \end{aligned} \quad (2.8)$$

Based on the work of Hildebrand (1994), Horwitz & Venkatesan (2002), we have the following theorem to show that the performance of adding-walk is provably good.

Theorem 2.4 (Teske (2001)). *Let G be a finite abelian group of prime order. Assume that we work with an r adding-walk together with an independent hash function where $r \geq 16$. Then the average number of iterations before a collision occurs, divided by $\sqrt{|G|}$, is approximately independent of $|G|$. In addition, if $r > 16$ then the average number of iterations is bounded by $1.45\sqrt{|G|}$ when using Teske's modified cycle-finding algorithm.*

The reported $E(\mu + \lambda)$ is $1.29\sqrt{|G|}$ for subgroups of $E(\mathbb{F}_p)$, which is close to the theoretically optimal bound $1.25\sqrt{|G|}$ in Theorem 2.2.

2.2.3 Teske's Mixed-walk

Teske proposed another method named mixed-walk (Teske 1998) which has a similar performance to the adding-walk. It uses a mixture of the adding-walk and some squaring steps, similar to Pollard's iteration function. Assume that we are using r multipliers in the adding-walk and q squaring steps. The pseudo-random function $f_{TM} : G \to G$ is defined as follows,

$$f_{TM}(Y) = \begin{cases} Y \cdot M_{v(Y)} & v(Y) \in \{1, 2, \cdots r\} \\ Y^2 & \text{Otherwise} \end{cases} \quad (2.9)$$

Experimental results show that $r \geq 16$ plus $q/r \approx 0.25$ yields a performance comparable to that of a truly random walk. A mixed-walk of 16 multipliers and 4 squaring steps is reported to have an expected length of $E(\mu + \lambda) \approx 1.3\sqrt{|G|}$.

2.3 Advances in Collision-detection Algorithms

In Floyd's algorithm, some Y_i will be evaluated twice, which is time-consuming. There are faster algorithms. We discuss two of Brent's algorithms (Brent 1980) and a variant (Teske 1998).

2.3.1 Brent's Algorithms

Brent proposed two algorithms (Brent 1980) which are generally 25% faster than Floyd's method. A modified version of them was used in factoring the eighth Fermat number by Brent & Pollard (1981).

Brent's first algorithm (Brent 1980) uses a variable z to keep the values of $Y_{l(i)-1}$ where $l(i) = 2^{\lfloor \log i \rfloor}$. z is compared with Y_i for each iteration and is updated by $z = Y_i$ when $i = 2^x - 1$ for $x = 1, 2, \cdots$ (i is the index of iteration and the base 2 is chosen for ease of implementation). Only one sequence Y_i needs to be computed and the value of z is easily updated. The correctness of this algorithm depends on the following idea.

Theorem 2.5. *For a periodic sequence $Y_0, Y_1, Y_2 \cdots$, there exists an $i > 0$ such that $Y_i = Y_{l(i)-1}$ and $l(i) \leq i < 2l(i)$. The smallest such i is $2^{\lceil \lg \max(\mu+1, \lambda) \rceil} + \lambda - 1$.*

Under the assumption that the iteration function is truly random, an expected number of $1.98\sqrt{|G|}$ iterations for $E(\mu + \lambda)$ is reported (Brent 1980). The number of evaluations is equal to the number of comparisons, and hence the total number of operations is bounded by $3.96\sqrt{|G|}$. If cost of comparisons is insignificant, the algorithm is 30% faster in average than Floyd's algorithm. On the other hand, if comparisons are expensive, the speedup may be compromised.

A second algorithm is given in the same paper (Brent 1980). This algorithm avoids unnecessary comparisons as it is sufficient to compare only when $\frac{3}{2}l(i) \leq i < 2l(i)$. Under the assumption that the iteration function is truly random, the expected number of evaluations is $2.24\sqrt{|G|}$ with an expected number of comparison as $0.88\sqrt{|G|}$. The total number of operations is $3.12\sqrt{|G|}$.

A variation of Brent's algorithms is discussed by Teske (1998). It reduces the number of iterations by using more storage and comparisons. A chain of 8 cells is applied and each cell keeps a triplet (Y_i, a_i, b_i). Initially all the values in cells are Y_0 and is updated according to the following rules. At the i-th iteration, we compare the current value Y_i with previous values in the cells. If they are not equal, we check whether i is greater than 3 times the index of the element in the first cell. If this is true, we put current Y_i into the last cell, remove the element in first cell and then shift the other cells to the previous cell. Under the assumption that the function is truly random, the expected number of iterations is about $1.42\sqrt{|G|}$. For each iteration, there is one evaluation and eight comparisons.

2.4 Summary

We summarize the performance of collision-detection algorithms, making the assumption that the iteration function is truly random. We also compile a table including the performance of iteration functions, which is based on the reported experimental results. In the first table, the columns represent algorithms, number of expected iterations, evaluations and comparisons. In the second table, the columns denote iteration functions, multiplicative groups $(\mathbb{Z}/p\mathbb{Z})^*$, prime order subgroups of $(\mathbb{Z}/p\mathbb{Z})^*$ and prime order subgroups of $E(\mathbb{F}_p)$. $f_{TA[20]}$ denotes Teske's adding-walk with 20 multipliers and $f_{TM[16:4]}$ denotes Teske's mixed-walk with 16 multipliers plus 4 squaring steps. All the data in the table is normalised: $E(\mu + \lambda)$ is divided by $\sqrt{|G|}$.

Remark 2.6. In case where two different experimental results are reported by Teske (1998, 2001), we use first one.

Table 1: Performance of Cycle-finding Algorithms

ALGs	ITERs	EVALs	CMPs
Floyd's	1.03	3.09	1.03
Brent's 1st Alg	1.98	1.98	1.98
Brent's 2nd Alg	2.24	2.24	0.88
Teske's Modified	1.42	1.42	$8 * 1.42$

Table 2: Performance of Iteration Functions

FUNCs	$(\mathbb{Z}/p\mathbb{Z})^*$	$S \leq (\mathbb{Z}/p\mathbb{Z})^*$	$S \leq E(\mathbb{F}_p)$
f_P	1.37	1.55	1.60
f_{PG}	-	-	1.62
$f_{TA[20]}$	-	-	1.29
$f_{TM[16:4]}$	-	-	1.30

3 Experimental Investigation

We can find few comparable results for Pollard's rho method and its variants, except those reported by Teske (1998, 2001). There are some gaps in Table 2. In this section, we fill the gaps in Table 2 by an empirical investigation and give some suggestions on better parameters such as starting values and partitioning methods. In addition, we test Teske's iteration function with a comprehensive set of data and verify Teske's results.

3.1 Description of Experiments

To prepare for the experiments, random prime numbers from 3 digits to 15 digits were chosen to give finite fields $\mathbb{F}_p$. We then considered the groups $(\mathbb{Z}/p\mathbb{Z})^*$ and subgroups of $(\mathbb{Z}/p\mathbb{Z})^*$ with orders from 3 to 13 digits. We also discuss elliptic curve discrete logarithms. Let $E(\mathbb{F}_p)$ be a finite abelian group formed by the points on an elliptic curve. G is a prime order subgroup of $E(\mathbb{F}_p)$ generated by a point P. Then an instance of the elliptic curve discrete logarithm problem (ECDLP) is stated as follows. Here we write the group operation as $\oplus$.

Definition 3.1 (ECDLP)**.** Given P, $Q \in G$ known, ECDLP is to find the smallest non-negative integer x such that,

$$Q = \underbrace{P \oplus P \oplus \cdots \oplus P}_{x \text{ times}}$$

To generate subgroups of $E(\mathbb{F}_p)$, we produce random elliptic curves over $\mathbb{F}_p$ and then compute the order for each group. Due to the Pohlig-Hellman algorithm (Pohlig & Hellman 1978), we concentrate on the subgroups with largest prime orders. A generator for each subgroup is computed. The number of instances of DLPs or ECDLPs computed is given in Table 3. The first column gives the (sub)groups by the number of decimal digits in their order. The second column is the number of DLPs or ECDLPs computed for each row. The third column gives the number of different starting values Y_0 for each instance of DLP or ECDLP.

Our implementaion is based on C++ using the GNU Multiple Precision Arithmetic Library (GMP). We ran the algorithms over Gentoo Linux on a Pentium 2.4GHz platform. The whole computation took more than a month.

Table 3: Instances of DLPs or ECDLPs

DIGITs	#DLPs (ECDLPs)	STs
3 to 8	200	100
9	100	50
10	50	20
11	50	10
12	50	5
13	50	1

3.2 Iteration Functions

We first discuss the performance of different iteration functions without collision-detection algorithms. The whole sequences generated by the iteration functions were stored. Therefore the groups were restricted to be small. 2500 discrete logarithms over groups of 6-7 digits were computed. All the data in Table 4 denotes the values of $E(\mu + \lambda)$ divided by $\sqrt{|G|}$. The results fill the gaps in Table 2.

Table 4: Performance of Iteration Functions

FUNCs	$(\mathbb{Z}/p\mathbb{Z})^*$	$S \leq (\mathbb{Z}/p\mathbb{Z})^*$	$S \leq E(\mathbb{F}_p)$
f_P	1.37	1.55	1.60
f_{PG}	1.41	1.55	1.62
$f_{TA[20]}$	1.28	1.27	1.29
$f_{TM[16:4]}$	1.30	1.30	1.30

We found that Pollard's original iteration function performed worse than the truly random case (Theorem 2.2). In addition, Pollard's generalized iteration funtion is slightly worse than the original function on average. On the other hand, Teske's adding-walk and mixed-walk iteration functions behave better and mimic random walks. We discuss the choice of parameters in the rho method below.

3.3 Starting Values

The value assigned to Y_0 for the iteration function $Y_{i+1} = f(Y_i)$ is called the starting value of the sequence. We can use a fixed value for all DLPs (such as $Y_0 = 1$) or generate a random starting values using powers of g and h. We investigate the potential impacts of different types of starting values, which does not seem to have been done before. The pseudo-random functions are either Pollard's original function or Teske's adding-walk using 20 multipliers. The collision-detection algorithm is Brent's second algorithm. In addition, we adopt the partitioning method used in Pollard's original function[2]. For fixed starting values, we compute 4500 instances for DLPs and ECDLPs. The mean values of results are normalized by $\sqrt{|G|}$ in Table 5.

Table 5: Impact of Initial Values

Groups	Functions	Fixed	Random
$(\mathbb{Z}/p\mathbb{Z})^*$	f_P	2.55	2.50
	$f_{TA[20]}$	2.28	2.28
$(\mathbb{Z}/p\mathbb{Z})^*$ subgroups	f_P	2.95	2.84
	$f_{TA[20]}$	2.27	2.26
$E(\mathbb{F}_p)$ subgroups	f_P	2.88	2.92
	$f_{TA[20]}$	2.28	2.25

Although it seems to lose the advantage of randomness, choosing $Y_0 = 1$ is not significantly worse

[2] Partitioning methods will be discussed in the next part.

than choosing Y_0 at random. However, the variance is smaller in the latter case.

It seems there is no direct way to apply random initial values with Pollard's iteration function (or similarly in Teske's mixed-walk). We may need to store some auxiliary variables and update them. For example in $(\mathbb{Z}/p\mathbb{Z})^*$, we have a collision if $g^i h^j Y_0^x \equiv g^{i'} h^{j'} Y_0^y \pmod{p}$. If Y_0 is not 1, we have to update the powers of Y_0 during the procedure. A random initial value is applicable for Teske's adding-walk function. We assume random starting values in the following sections.

3.4 Partitioning Methods

An important assumption in Theorem 2.4 is that the partitioning method is independent. Here the independence means the performance of the iteration function is not affected by the the properties of the partitioning method. We will consider the potential impact of partitioning methods in this part. As we will see later, the choice may have a strong influence on the performance.

A partitioning method maps values of Y_i into different rules in the iteration function, which behaves like a hash function. In Pollard's iteration function, a partition of size three is used. This is extended to N partitions in Teske's functions. Pollard's partitioning rule is $R = \lceil N \times Y_i/|G| \rceil$ where R is the index of the rule chosen and $|G|$ is the order of the group. This method depends mainly on the high-order bits of Y_i. An alternative, the division method, uses the lower-order bits of Y_i, that is $R = (Y_i \mod N)+1$. Another more complicated method suggested by Teske (2001) is Knuth's multiplicative hash function (Knuth 1981). The principle is as follows. Assume that the partition we want to produce is $v : G \to \{1, \cdots, N\}$ where N denotes the number of partitions. Let A be a rational approximation of the golden ratio $\frac{\sqrt{5}-1}{2}$. Define $u(g) = A \cdot g - \lfloor A \cdot g \rfloor$ where g denotes an element in the group. Then the partitioning method is defined by $v(g) = \lceil u(g) \cdot N \rceil$.

We empirically investigated the impacts of different partitioning algorithms. The pseudo-random functions were either Pollard's original function or Teske's adding-walk with 20 multipliers. The collision-detection algorithms involved include Floyd's algorithm, Brent's algorithms and Teske's algorithm. As there are two iteration functions and four cycle-finding algorithms, we discuss eight combinations of them. For each combination, we index Pollard's partitioning method, the division method and Knuth's method as methods $1, 2, 3$ respectively. The Y-axis denotes the number of iterations divided by $\sqrt{|G|}$. The results in groups $(\mathbb{Z}/p\mathbb{Z})^*$ and elliptic curve subgroups are shown in Figure 1 and Figure 2. Note that the different cycle-finding algorithms have different costs per iteration (see Section 2).

For Pollard's iteration functions in groups $(\mathbb{Z}/p\mathbb{Z})^*$, it is much better to apply the original partitioning proposed by Pollard (1978), which uses high-order bits. The other two methods perform worse in this case. In other cases, such as Pollard's iteration functions in subgroups of $E(\mathbb{F}_p)$, it is slightly better to use the division method.

3.5 Choice of Parameters in Teske's Functions

We have discussed impacts of initial values and performance of different partitioning methods. In this part, we consider how the performance is affected by the parameters in Teske's adding-walk and mixed-walk functions. DLPs in prime order subgroups of

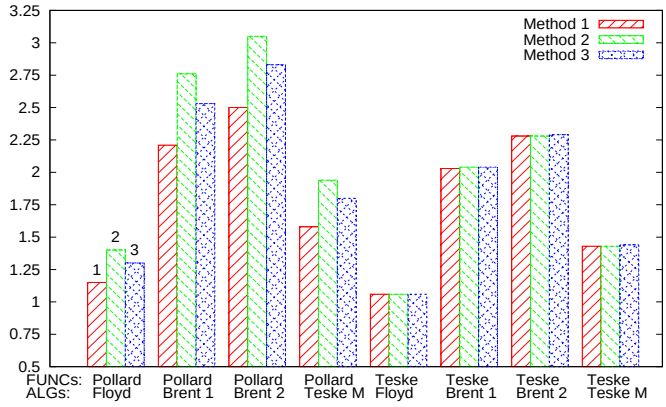

Figure 1: Partitioning Methods in Groups $(\mathbb{Z}/p\mathbb{Z})^*$

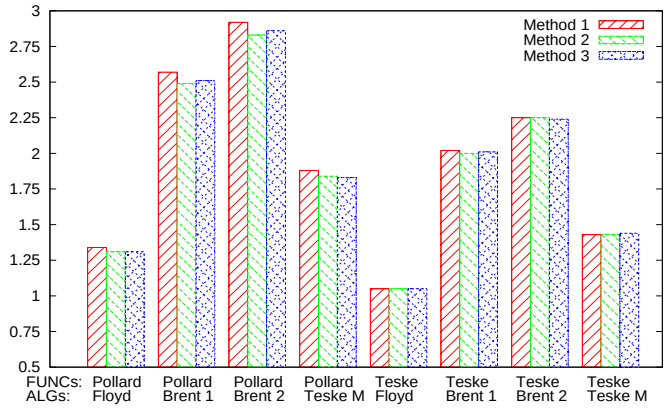

Figure 2: Partitioning Methods in Subgroups of $E(\mathbb{F}_p)$

$(\mathbb{Z}/p\mathbb{Z})^*$ can be considered as analogues of ECDLPs in prime order subgroups of $E(\mathbb{F}_p)$. The discrete logarithm problems considered are defined in groups $(\mathbb{Z}/p\mathbb{Z})^*$ and the largest prime order subgroups of $E(\mathbb{F}_p)$.

3.5.1 Groups $(\mathbb{Z}/p\mathbb{Z})^*$

We discuss the choice of parameters in Teske's function in the groups $(\mathbb{Z}/p\mathbb{Z})^*$. Teske's modified collision-detection algorithm is applied. Theorem 2.4 claims that the number of iterations is bounded by $1.45\sqrt{|G|}$ for Teske's adding-walk with $r \geq 16$ multipliers. The performance of different values of r is plotted in Figure 3. The X-axis denotes the number of multipliers used in adding-walk and the Y-axis denotes the number of iterations divided by $\sqrt{|G|}$.

The empirical results verify Theorem 2.4. We were also able to verify that the performance is generally better using a larger number of partitions. Considering the initialization cost as well, a partition number of 20-60 is a reasonable value. In addition, it has been suggested that mixed-walk with ratios q/r between $1/4$ and $1/2$ with $r \geq 16$ may yield a good performance (Teske 1998). Our experimental results do not support this suggestion. We found that mixed-

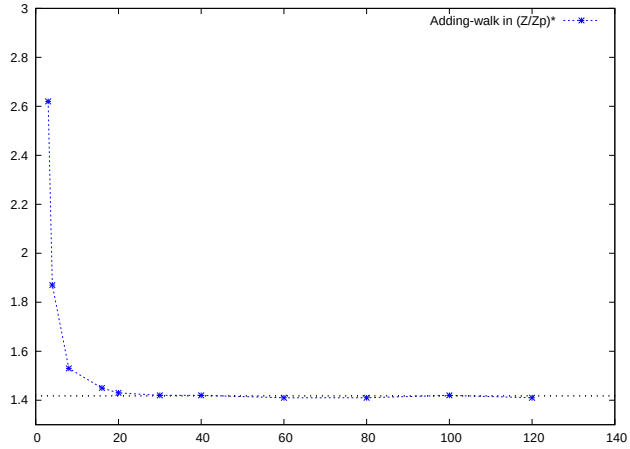

Figure 3: Performance of Adding-walk in Groups $(\mathbb{Z}/p\mathbb{Z})^*$

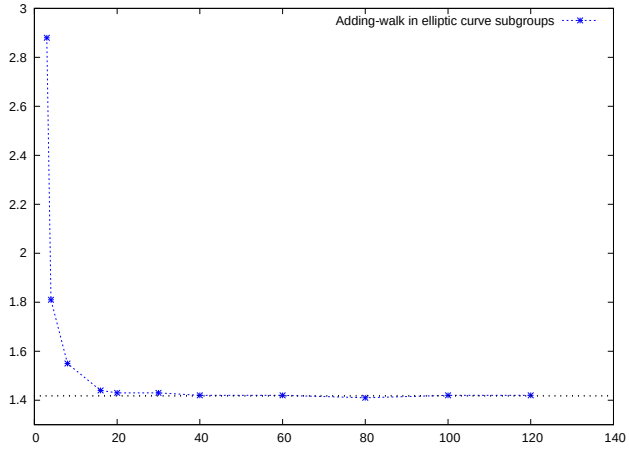

Figure 4: Performance of Adding-walk in Subgroups of $E(\mathbb{F}_p)$

walks with ratios smaller than or equal to 1/4 behave slightly better than those with ratios between 1/4 and 1/2. To illustrate this, the performance for various algorithms is tabulated in Table 6. The columns give the ratios applied, number of multipliers, squaring steps and iterations. As usual, the data in last column is normalized by $\sqrt{|G|}$.

Table 6: Performance of Mixed-walk in Groups $(\mathbb{Z}/p\mathbb{Z})^*$

RATIOs	MULTs	SQRs	ITERs
0.25	16	4	1.46
0.25	20	5	1.45
0.25	40	10	1.44
0.25	60	15	1.44
0.10	20	2	1.45
0.20	20	4	1.47
0.40	20	8	1.50
0.50	20	10	1.51
0.60	20	12	1.53
0.80	20	16	1.57

3.5.2 Prime Order Subgroups of $E(\mathbb{F}_p)$

We discuss the choice of parameters in Teske's function in the subgroups of $E(\mathbb{F}_p)$. Teske's modified collision-detection algorithm is applied. The number of iterations in Figure 4 are bounded by $1.45\sqrt{|G|}$ for Teske's adding-walk function with $r \geq 16$ multipliers. This verifies the effectiveness of Teske's function in the ECDLP case. Similarly a partition number of 20-60 is preferred. The performance of mixed-walk is obtained in the Table 7. For mixed-walk in subgroups of $E(\mathbb{F}_p)$, we arrive a similar result as before. Mixed-walks with ratios q/r smaller or equal to 1/4 with more than 16 multipliers are preferable.

Table 7: Performance of Mixed-walk in Subgroups of $E(\mathbb{F}_p)$

RATIOs	MULTs	SQRs	ITERs
0.25	16	4	1.48
0.25	20	5	1.47
0.25	40	10	1.44
0.25	60	15	1.44
0.10	20	2	1.45
0.20	20	4	1.46
0.40	20	8	1.49
0.50	20	10	1.52
0.60	20	12	1.54
0.80	20	16	1.58

4 Conclusion and Future Work

We discussed efficiency issues regarding Pollard's rho method and its variants for discrete logarithm problems and elliptic curve discrete logarithm problems. We have performed an empirical investigation to fill the current gaps in the literature, suggested better parameters for iteration functions and revisited Teske's adding-walk and mixed-walk functions.

In the previous sections, we have used the assumption that the partitioning method is independent of the iteration function. Finding a way to prove this would be an advance. In addition, the experimental results suggest that Teske's mixed-walk behaves as well as the adding-walk. While the performance of the adding-walk is supported by some theoretical results, we find no easy way to analyze the behavior of the mixed-walk. A potential way to achieve this might be based on the recent work of Miller & Venkatesan (2006) and Kim et al. (2007).

Acknowledgements

We would like to thank the anonymous referees for their helpful comments.

References

Brent, R. P. (1980), 'An improved Monte Carlo factorization algorithm', *BIT* **20**(2), 176–184.

Brent, R. P. & Pollard, J. M. (1981), 'Factorization of the eighth Fermat number', *Mathematics of Computation* **36**, 627–630.

Coppersmith, D., Odlyzko, A. M. & Schroeppel, R. (1986), 'Discrete logarithms in $GF(p)$', *Algorithmica* **1**(1), 1–16.

Diffie, W. & Hellman, M. E. (1976), 'New directions in cryptography', *IEEE Trans. Inform. Theory* **IT-22**, 644–654.

Gamal, T. E. (1985), 'A public key cryptosystem and a signature scheme based on discrete logarithms', *IEEE Trans. Inform. Theory* **31**, 469–472.

Harris, B. (1960), 'Probability Distribution Related to Random Mappings', *Ann. Math. Statist.* **31**, 1045–1062.

Hildebrand, M. (1994), 'Random walks supported on random points of Z/nZ', *Probability Theory and Related Fields* **100**(2), 191–203.

Horwitz, J. & Venkatesan, R. (2002), Random Cayley digraphs and the discrete logarithm, *in* 'Algorithmic Number Theory Symposium V, ANTS-V (LNCS 2369)', pp. 100–114.

Kim, J. H., Montenegro, R. & Tetali, P. (2007), 'A near optimal bound for Pollard's rho to solve discrete log', IEEE Proc. of the Foundations of Computer Science (FOCS), 2007, Providence, RI, to appear.

Knuth, D. E. (1981), *The Art of Computer Programming*, Vol. 3, 2nd edn, Addison-Wesley, Reading, Mass.

Knuth, D. E. (1997), *The Art of Computer Programming*, Vol. 2, 3nd edn, Addison-Wesley, Reading, Mass.

Miller, S. D. & Venkatesan, R. (2006), Spectral analysis of Pollard rho collisions, *in* 'Algorithmic Number Theory Symposium (ANTS VII), LNCS 4076, Springer-Verlag, 573-581'.

Pohlig, S. C. & Hellman, M. E. (1978), 'An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance', *IEEE Trans. Inform. Theory* **IT-24**(1), 106–110.

Pollard, J. M. (1978), 'Monte Carlo methods for index computation mod p', *Mathematics of Computation* **32**, 918–924.

Teske (1998), Speeding up Pollard's rho method for computing discrete logarithms, *in* 'Algorithmic Number Theory Symposium (ANTS IV), LNCS 1423, Springer-Verlag, 541-553'.

Teske, E. (2001), 'On random walks for Pollard's rho method', *Mathematics of Computation* **70**(234), 809–825.