Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2012, Article ID 635909, 15 pages
doi:10.1155/2012/635909

Research Article

An Efficient Collision Detection Method for Computing Discrete Logarithms with Pollard's Rho

Ping Wang and Fangguo Zhang

School of Information Science and Technology, Sun Yat-sen University, Guangzhou 510006, China

Correspondence should be addressed to Fangguo Zhang, isszhfg@mail.sysu.edu.cn

Received 7 July 2011; Revised 15 November 2011; Accepted 21 November 2011

Academic Editor: Jacek Rokicki

Pollard's rho method and its parallelized variant are at present known as the best generic algorithms for computing discrete logarithms. However, when we compute discrete logarithms in cyclic groups of large orders using Pollard's rho method, collision detection is always a high time and space consumer. In this paper, we present a new efficient collision detection algorithm for Pollard's rho method. The new algorithm is more efficient than the previous distinguished point method and can be easily adapted to other applications. However, the new algorithm does not work with the parallelized rho method, but it can be parallelized with Pollard's lambda method. Besides the theoretical analysis, we also compare the performances of the new algorithm with the distinguished point method in experiments with elliptic curve groups. The experiments show that the new algorithm can reduce the expected number of iterations before reaching a match from $1.309\sqrt{|G|}$ to $1.295\sqrt{|G|}$ under the same space requirements for the single rho method.

1. Introduction

One of the most important assumptions in modern cryptography is the hardness of the discrete logarithm problem (DLP). Many popular cryptosystems base their security on DLP. Such cryptosystems are, for example, the Diffie-Hellman key agreement protocol [1], the ElGamal signature and encryption schemes [2], the US Government Digital Signature Algorithm (DSA) [3], and the Schnorr signature scheme [4]. Originally, they worked with multiplicative groups of finite prime fields. Once elliptic curve cryptosystems were proposed by Koblitz [5] and Miller [6], analogous practical systems based on the DLP in groups of points of elliptic curves over finite fields were designed [7]. Recall the following two definitions.

Definition 1.1 (discrete logarithm problem, DLP). Let G be a cyclic group of prime order p, and let $g \in G$ be generator of G. Given $g, h \in G$, determine the integer $0 \leq k < p$ such that $h = g^k$.

Definition 1.2 (elliptic curve discrete logarithm problem, ECDLP). Let E be an elliptic curve defined over finite field $\mathbb{F}_q$. Let $P \in E$ be a point of prime order n, and let G be the subgroup of E generated by P. Given $Q \in G$, determine the integer $0 \leq k < n$ such that $Q = kP$.

For DLP on a multiplicative subgroup G of prime order p of finite field $\mathbb{F}_q$, the index calculus method determines the size of q, which is a subexponential time algorithm, while the size of p is set by Pollard's rho method [8].

Furthermore, for ECDLP, Pollard's rho method and its modifications by Gallant et al. [9] and Wiener and Zuccherato [10] are to date known as the most efficient general algorithms. van Oorschot and Wiener [11] showed that the modified Pollard's rho method can be parallelized with linear speedup.

Pollard's rho method is a randomized algorithm for computing discrete logarithms. Generally, an iteration function $F : G \to G$ is used to define a pseudorandom sequence Y_i by $Y_{i+1} = F(Y_i)$ for $i = 0, 1, 2, \ldots$, with some initial value Y_0. The sequence $Y_0, Y_1, Y_2, \ldots$ represents a walk in the group G. The basic assumption is that the walk Y_i behaves as a random walk. Because the order of the group is finite, the sequence will ultimately reach an element that has occurred before. This is called a *collision* or a *match*. The advantage of this method is that the space requirements are small if one uses a clever method of detecting a collision. The problem of efficient collision detection of a pseudo-random walk in Pollard's rho method is the central topic of this paper.

There are several collision detection algorithms for a random walk in the group G. These algorithms in general do not exploit the group structure of G. As a result, the algorithms discussed in this paper in fact apply to any set G on which an iterated function F is used to make random walks, and their utilization goes beyond discrete logarithm computation.

A simple approach to detecting a collision with Pollard's rho method is to use Floyd's cycle-finding algorithm [8], which shows that it suffices to compare Y_i and Y_{2i} for all i to find a collision. Floyd's algorithm uses only a small constant amount of storage, but needs roughly three times more work than is necessary. Brent [12] improved this approach by using an auxiliary variable. Nivasch designed an algorithm [13] for detecting periodicity in sequences using a single pointer and a small stack. This stack algorithm halts at a uniformly random point in the second loop through the sequence's cycle.

In finding DES collisions [14, 15], Quisquater and Delescaille took a different approach based on storing distinguished points, an idea noted earlier by Rivest to reduce the search time in Hellman time-memory tradeoff [16]. A distinguished point is one that has some easily checked property such as having a fixed number of leading zero bits. During the pseudo-random walk, points that satisfy the distinguishing property are stored. Collision can be detected when a distinguished point is encountered a second time. This technique can be efficiently applied to find collisions among multiple processors [11].

In this paper, we describe a new efficient collision detection algorithm for computing discrete logarithm with Pollard's rho method. It is a probabilistic algorithm and more efficient than the previous methods. With this algorithm, we can significantly reduce the space requirements and provide a better time-space trade-off approach. We also compare their performances in experiments with elliptic curve groups, and our experimental results confirmed the theoretical analysis.

The remainder of this paper is organized as follows. In Section 2, we recall Pollard's rho method for discrete logarithm computation and discuss several previous methods for collision detection. We describe and analyze the new algorithm in Section 3 and discuss its applications in Section 4. We present our experiments in Section 5 and conclude the paper in Section 6.

2. Preliminary

In this section, we describe Pollard's rho method for discrete logarithm computation and then discuss several collision detection algorithms and their performances.

2.1. Pollard's Rho Method

Pollard [8] proposed an elegant algorithm for the discrete logarithms 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*.

If D is any finite set and $F : D \to D$ is a mapping and the sequence (X_i) in D is defined by the rule:

$$X_0 \in D, \qquad X_{i+1} = F(X_i), \tag{2.1}$$

this sequence is ultimately periodic. Hence, there exist unique integers $\mu \geq 0$ and $\lambda \geq 1$ such that $X_0, \ldots, X_{\mu+\lambda-1}$ are all distinct, but $X_i = X_{i+\lambda}$ for all $i \geq \mu$. A pair (X_i, X_j) of two elements of the sequence is called a *match* if $X_i = X_j$ where, $i \neq j$. For the expected values of μ and λ, we have the following theorem.

Theorem 2.1 (see [17]). *Under the assumption that an iteration function $F : D \to D$ behaves like a truly random mapping and the initial value X_0 is a randomly chosen group element, the expected values for μ and λ are $\sqrt{\pi|D|/8}$. The expected number of evaluations before a match appears is $E(\mu + \lambda) = \sqrt{\pi|D|/2} \approx 1.25\sqrt{|D|}$.*

Now we explain how the rho method for computing discrete logarithms works. Let G be a cyclic group of prime order p, and let $g \in G$ be generator of G and $h \in G$. The discrete logarithm problem is to compute x satisfying $g^x \equiv h$. Pollard defined the iteration function $F : G \to G$ as follows:

$$F(Y) = \begin{cases} g \cdot Y & Y \in S_1, \\ Y^2 & Y \in S_2, \\ h \cdot Y & Y \in S_3. \end{cases} \tag{2.2}$$

Let the initial value $Y_0 = 1$. In each iteration of $Y_{i+1} = F(Y_i)$, the function uses one of three rules depending on the value of Y_i. The group G is partitioned into three subsets S_1, S_2, S_3 of roughly equal size. Each Y_i has the form $g^{a_i} h^{b_i}$. The sequence (a_i) (and similarly for (b_i)) can be computed as follows:

$$a_{i+1} = \begin{cases} a_i + 1 \pmod{p} & Y_i \in S_1, \\ 2a_i \pmod{p} & Y_i \in S_2, \\ a_i \pmod{p} & Y_i \in S_3. \end{cases} \tag{2.3}$$

As soon as we have a match (Y_i, Y_j), we have the equation $g^{a_i} * h^{b_i} = g^{a_j} * h^{b_j}$. Since $h = g^x$, this gives

$$a_i + b_i x \equiv a_j + b_j x \bmod p. \tag{2.4}$$

Now, if $\gcd(b_i - b_j, p) = 1$, we get that $x = (a_j - a_i)(b_i - b_j)^{-1} \bmod p$. Due to the method of Pohlig and Hellman [18], in practice applications the group order p is prime, so that it is very unlikely that $\gcd(b_i - b_j, p) > 1$ if p is large.

Theorem 2.1 makes the assumption of true randomness. However, it has been shown empirically that this assumption does not hold exactly for Pollard's iteration function [19]. The actual performance is worse than the expected value given in Theorem 2.1.

Teske [19] proposed better iteration functions by applying more arbitrary multipliers. Assume that we are using r partitions (multipliers). We generate $2r$ random numbers,

$$m_i, n_i \in_R \{0, 1, \ldots, p-1\}, \quad \text{for } i = 1, 2, \ldots, r. \tag{2.5}$$

Then we precompute r multipliers $M_1, M_2, \ldots, M_r$, where $M_i = g^{m_i} \cdot h^{n_i}$, for $i = 1, 2, \ldots, r$. Define a hash function,

$$v : G \longrightarrow \{1, 2, \ldots, r\}. \tag{2.6}$$

Then the iteration function $F : G \to G$ defined as

$$F(Y) = Y \cdot M_{v(Y)}, \quad \text{where } v(Y) \in \{1, 2, \ldots, r\}. \tag{2.7}$$

The indices are update by

$$a_{i+1} = a_i + m_{v(Y_i)}, \qquad b_{i+1} = b_i + n_{v(Y_i)}. \tag{2.8}$$

The difference in performance between Pollard's original walk and Teske's r-adding walk has been studied in [19, 20]. We summarize the results as follows. In prime order subgroups of $\mathbb{Z}_p^*$, the value of $E(\mu + \lambda)$ for Pollard's original walk and Teske's r-adding walk is $1.55\sqrt{|G|}$ and $1.27\sqrt{|G|}$, while in groups of points of elliptic curves over finite fields, the value is $1.60\sqrt{|G|}$ and $1.29\sqrt{|G|}$, respectively.

2.2. Previous Methods for Collision Detection

To find the collision in the pseudo-random walk, it always needs much storage. In order to minimize the storage requirements, a collision detection algorithm can be applied with a small penalty in the running time.

Floyd's Cycle-Finding Algorithm

In Pollard's paper, Floyd's algorithm is applied. To find $Y_i = Y_j$, the algorithm calculates $(Y_i, a_i, b_i, Y_{2i}, a_{2i}, b_{2i})$ until $Y_i = Y_{2i}$. For each iteration, we compute $Y_{i+1} = F(Y_i)$ and

$Y_{2(i+1)} = F(F(Y_{2i}))$, which means that this algorithm requires negligible storage. Floyd's algorithm is based on the following idea.

Theorem 2.2 (see [25]). *For a periodic sequence $Y_0, Y_1, Y_2, \ldots$, there exists an $i > 0$ such that $Y_i = Y_{2i}$ and the smallest such i lies in the range $\mu \leq i \leq \mu + \lambda$.*

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 $\sqrt{\pi^5|G|/288} \approx 1.03\sqrt{|G|}$ [20]. The key point for this algorithm is that we need three group operations and one comparison for each iteration, which makes it inefficient.

Brent's Algorithm

Brent proposed an algorithm [12] which is generally 25% faster than Floyd's method. It uses an auxiliary variable, say w, which at each stage of the algorithm holds $Y_{l(i)-1}$, where $l(i) = 2^{\lfloor \log i \rfloor}$. w is compared with Y_i for each iteration and is updated by $w = Y_i$ when $i = 2^k - 1$ for $k = 1, 2, \ldots$. The correctness of this algorithm depends on the following fact.

Theorem 2.3 (see [20]). *For a periodic sequence $Y_0, Y_1, Y_2, \ldots$, 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 $2^{\lceil \log \max(\mu+1,\lambda) \rceil} + \lambda - 1$.*

Under the assumption that $F : G \to G$ is a random mapping, with Brent's algorithm the first match is expected to occur after $1.98\sqrt{|G|}$ iterations [12]. The algorithm needs one group operation and one comparison for each iteration, which makes it 25%–30% faster than Floyd's algorithm. Variations of Brent's algorithm requiring slightly more storage and comparisons but less iterations can be found in [21, 22].

Stack Algorithm

In 2004, Nivasch proposed an interesting algorithm that uses logarithmic storage space and can be adapted with tradeoff for storage space versus speed. This algorithm requires that a total ordering $<$ be defined on the set G, and it works as follows. Keep a stack of pairs (Y_i, i), where, at all times, both the i's and the Y_i's in the stack form strictly increasing sequences. The stack is initially empty. At each step j, pop from the stack all entries (Y_i, i), where $Y_i > Y_j$. If a match $Y_i = Y_j$ is found with an element in the stack, the algorithm terminates successfully. Otherwise, push (Y_j, j) on top of the stack and continue. The stack algorithm depends on the following fact.

Theorem 2.4 (see [13]). *The stack algorithm always halts on the smallest value of the sequence's cycle, at some time in $[\mu + \lambda, \mu + 2\lambda)$.*

Under the assumption that $F : G \to G$ is a random mapping, the expected number of iterations before finding a match is $5/2\sqrt{\pi|G|/8} \approx 1.57\sqrt{|G|}$ [13]. The algorithm needs a little bit more than one group operation and one comparison for each iteration. Under the same assumption, Nivasch proves also that the expected size of the stack is $\ln h + O(1)$. Therefore, the algorithm only requires a logarithmic amount of memory.

Distinguished Point

The idea of the distinguished point method is to search for a *match* not among all terms of the sequence, but only among a small subset of terms that satisfy a certain distinguishing property. It works as follows. One defines a set D, a subset of G, that consists of all group elements that satisfy a certain distinguishing property. During the pseudo-random walk, points that satisfy the distinguishing property are stored. Collision can be detected when a distinguished point is encountered a second time.

Currently, the distinguished point method is the most efficient algorithm to detect collisions in pseudo-random walk when $|G|$ is large. A popular way of defining D is to fix an integer k and to define that $w \in D$ if and only if the k least significant bits in the representation of w as a binary string are zero. To break ECC2K-130, it [23] defines the distinguishing property as the Hamming weight of normal-basis representation of x-coordinate of the point less than or equal to 34. Notice that this kind of definitions allows a fast check for the distinguishing property to hold, and the size of D can be easily monitored as well. Obviously, we have the following theorem.

Theorem 2.5 (see [11]). *Let θ be the proportion of points in G which satisfy the distinguishing property, that is, $\theta = |D|/|G|$. Under the assumption that $F : G \to G$ is a random mapping and D is a uniform distribution in G, the expected number of iterations before finding a match is $\sqrt{\pi|G|/2}+1/\theta$.*

3. The New Algorithm

We are motivated by the fact that, in distinguished point method, the distinguished points may be not uniformly distributed in the pseudo-random walk, also the points in subset D may be not uniformly distributed in G, which always results in more iteration requirements. We are trying to design an algorithm which leads to a uniform distribution and also to provide a better way for time-space tradeoff rather than distinguishing property.

3.1. The Basic Algorithm

To find a collision in pseudo-random walk, which is produced by the iteration function $F : G \to G$, assuming F is a random mapping on G, our basic algorithm works as follows. We fix an integer N and use an auxiliary variable, say w, which at each N iterations keep the minimum value of N successive values produced by the iteration function F. Once getting the minimum value w from N successive values, we check whether this value has occurred before in the stored sequence, if so, we find the match and we are done. Otherwise, store this value w to the sequence. Then continue to compute the next N new values and repeat the previous procedures. Choose the integer N properly, we will find the match among the newly generated minimum value and stored minimum values.

More precisely, to find a collision in pseudo-random sequence $Y_0, Y_1, Y_2, \ldots$, which is produced by the iteration function $F : G \to G$, we have Algorithm 1.

It is obvious that the algorithm can be considered as two parts. In the first part, that is from step (3) to step (11), we seek the minimum value w from N successive values in the pseudo-random walk. The operation is very simple; if the current value Y_j is smaller than w, then just update w with the current value and continue the next iteration. Notice that steps (7), (8), and (9) can be omitted, since it is unlikely that there is a match within N iterations. Even if it happened, the algorithm ensures that we can find a match within

Input: Initial value Y_0, iteration function $F : G \to G$, fixed integer N
Output: m and n, such that $Y_m = Y_n$
(1) $w \leftarrow Y_0$, $m \leftarrow 0$, $n \leftarrow 0$
(2) **for** $i = 1$ to $\lceil |G|/N \rceil$ **do**
(3) **for** $j = (i-1)N + 1$ to $iN - 1$ **do**
(4) $Y_j \leftarrow F(Y_{j-1})$
(5) **if** $Y_j < w$ **then**
(6) $w \leftarrow Y_j$, $n \leftarrow j$
(7) **else if** $Y_j = w$ **then**
(8) $m \leftarrow j$
(9) **return** m, n
(10) **end if**
(11) **end for**
(12)
(13) **for** $k = 1$ to $i - 1$ **do**
(14) **if** $u_k = w$ **then**
(15) $m \leftarrow v_k$
(16) **return** m, n
(17) **end if**
(18) **end for**
(19) $u_i \leftarrow w$, $v_i \leftarrow n$
(20) $w \leftarrow F(Y_{iN-1})$, $n \leftarrow iN$
(21) **end for**

Algorithm 1: The new algorithm for collision detection.

the next N iterations. As a result, there is only one group operation and one comparison in the first part, which consist the main operations of the algorithm.

In the second part, that is from step (13) to step (19), once we get a minimum value w, we check whether w has appeared before in the previous stored values (u_k), which is empty at the beginning. If this is the case, the algorithm will *return* the corresponding indices and we are done. Otherwise, save the value w to the sequence (u_k) of the minimum values, and the second part is finished, continue the next N iterations. It is clear that the second part can be speeded up by using a hash table. And, more important, the second part can be independent to the first part, which means the stored sequence of minimum values can be *off line*; that is, the first part is response for generating minimum values along the random walk, while the second part searches the collision among stored minimum values independently.

3.2. Analysis

For further analysis of the algorithm, we assume that the iteration function $F : G \to G$ behaves like a truly random mapping. According to Theorem 2.1, the expected number of iterations before reaching a match is $\sqrt{\pi |G|/2}$. Let us look at some simple cases for the new algorithm. For $N = 1$, which means we store all the values in the sequence before reaching a match, and the match can be found once it appears. For $N = 2$, we need to store half of the values in the sequence before reaching a match, and always the match can be found once it appears. Obviously, the bigger the integer N, the less values we need to store. As a result, with the integer N increasing, there is a probability that we cannot detect the collision immediately when it happens. So, the new algorithm is a probabilistic algorithm. However,

with high probability, the algorithm will halt close to the beginning of the second loop. More precisely, we have the following theorem.

Theorem 3.1. *Under the assumption that an iteration function $F : G \to G$ behaves like a truly random mapping and the initial value Y_0 is a randomly chosen group element, for Algorithm 1, the expected number of iterations before finding a match is $\sqrt{\pi|G|/2} + (k + 1/2)N$ with probability $1 - (2/3)^{(k-1)}/2$, where $k = 0, 1, 2, \ldots$.*

Proof. Let I_i be the set that consists of the ith N successive values generated by iteration function F; that is,

$$I_i = \{Y_j \mid iN \le j \le (i+1)N - 1,\ j \ge 0\}, \quad \text{for } i = 0, 1, 2, \ldots, \lceil |G|/N \rceil - 1. \tag{3.1}$$

For finite group G, the sequence produced by F is eventually period; that is, for any fixed F and Y_0, there exist certain integers m and n, such that

$$\begin{aligned} I_m \bigcap I_n &\neq \emptyset, \quad m < n, \\ I_i \bigcap I_j &= \emptyset, \quad \text{for } 0 \le i, j < n,\ i \neq j, \end{aligned} \tag{3.2}$$

and also

$$I_{m+i} \bigcap I_{n+i} \neq \emptyset, \quad \text{for } i \ge 0. \tag{3.3}$$

To prove the theorem, we divide it into two cases, that is, $k = 0$ and $k = 1, 2, \ldots$. For $k = 0$, let $\min_m$ and $\min_n$ be the minimum values of I_m and I_n, then

$$\begin{aligned} \Pr\left(\min_m = \min_n\right) &= \frac{1}{N} \sum_{i=0}^{N-1} \left(\frac{1}{N} + \frac{2}{N} + \cdots + \frac{N-i}{N}\right) \frac{1}{N-i} \\ &= \frac{N^2 + 3N + 3}{4N^2} \\ &\approx \frac{1}{4}, \end{aligned} \tag{3.4}$$

that is, the probability of successfully detecting the collision within the first two intersection sets is $1/4$.

For each of $k = 1, 2, \ldots$, we notice that, for two (intersected) sets I_i and I_j, let $\min_i$ and $\min_j$ be the minimum values of I_i and I_j, respectively, we have

$$\Pr\left(\min_i = \min_j\right) = \frac{\left|I_i \bigcap I_j\right|^2}{N^2}, \quad \text{where } \left|I_i \bigcap I_j\right| \text{ denotes the cardinality of } I_i \bigcap I_j. \tag{3.5}$$

Therefore, we have

$$\begin{aligned}\Pr\left(\min_{m+k}=\min_{n+k}\right) &= \frac{1}{N}\left(\frac{1}{N^2}+\frac{2^2}{N^2}+\cdots+\frac{(N-1)^2}{N^2}\right)\\ &= \frac{2N^2-3N+1}{6N^2}\\ &\approx \frac{1}{3}.\end{aligned} \tag{3.6}$$

Under the assumption that the iteration function $F : G \to G$ is a random mapping, according to Theorem 2.1, the expected number of evaluations before a match appears is $\sqrt{\pi|D|/2}$. Combining the above two cases, using Algorithm 1, the expected number of iterations before reaching a match among minimum values is $\sqrt{\pi|G|/2} + (k + 1/2)N$ with probability $1 - (2/3)^{(k-1)}/2$, where $k = 0, 1, 2, \ldots$. □

Remark 3.2. According to the above theorem, we need to store $\sqrt{\pi|G|/2}/N + k$ terms to find the match with the probability $1 - (2/3)^{(k-1)}/2$, where k can be $0, 1, 2, \ldots$; that is, by setting parameter N, we can balance the expected number of iterations and the expected space requirements. Therefore, Algorithm 1 is a time-space trade-off algorithm.

Remark 3.3. Algorithm 1 is a probabilistic algorithm. There is a probability that we cannot detect the collision immediately when it happens. However, with high probability, the algorithm will halt close to the beginning of the second loop. For example, the probability of successfully detecting the collision $1 - (2/3)^{(k-1)}/2$ is 0.90 with $k = 5$.

Notice that, compared to the distinguished point method, the new algorithm has two advantages. First, the distinguished point method depends on the assumption that the distinguished points are uniformly distributed in the pseudo-random walk, and also the points in subset D are uniformly distributed in G. However, in practice this may not be the case, which generally results in more iterations requirement, while, for the new algorithm, each stored minimum value represents N successive values and the performance of the new algorithm independent of such assumption. Because the distinguished point method is currently the most efficient algorithm, we compare the actual performances of the new algorithm with the distinguished point method under the same expected storage requirement in experiments with elliptic curve groups in Section 5.

Second, using distinguished point method, it is possible for a random walk to fall into a loop which contains no distinguished point [11]. Then, the processor cannot find new distinguished point any more on the path. Left undetected, the processor would cease to contribute to the collision search. In this case, we can restart the random walk by choosing a different initial point. However, those points calculated by previous walk do not help for the collision search, while the new algorithm can avoid such problem, because it can always find the collision among minimum values whenever it falls into a loop.

4. Applications

The new algorithm can be combined with other algorithms, such as Pollard's lambda method, and can be adapted to a wide range of problems which can be reduced to finding collisions,

such as in Pollard's rho method for factorization [24] and in studying the behavior of random number generators [25]. In this section, we will address some of these issues.

4.1. Pollard's Lambda Method

It is clear that the new algorithm can be applied to Pollard's lambda method (also called the kangaroo method). The lambda method computes a discrete logarithm in an arbitrary cyclic group, given that the value is known to lie in a certain interval, that is, $h = g^k$, where $k \in [a, b]$ but unknown.

Generally, we have two kangaroos, one tame and one wild. Their positions are represented by group elements, the tame kangaroo T with starting point $t_0 = g^{\lfloor (a+b)/2 \rfloor}$ and the wild kangaroo W with starting point $w_0 = h$, and they travel in the cyclic group $G = \langle g \rangle$. In terms of the exponents of g, T starts at the middle of the interval $[a, b]$, while W starts at x. Since we do not know k, we do not know the exact location of the wild kangaroo, and that is why it is called wild. The two kangaroos produce two different pseudo-random walks with the same walking rules. It is obvious that, at all times, the point of tame kangaroo has the form g^i and the point of wild kangaroo has the form $h * g^j$ for some known integers i and j. The purpose is to provoke a collision between the tame and the wild kangaroos, from which we can deduce the wild kangaroo's starting point, that is, $k = (i - j) \bmod |G|$.

Similar to the case of Pollard's rho method, the new algorithm can be applied in this case to efficiently detect the collision. The advantage of the new algorithm is that we can achieve uniform distributions of minimum values in the pseudo-random walks both for the tame kangaroos and the wild kangaroos. The different performances of the new algorithm and distinguished point method in this case can refer to the case of Pollard's rho method.

4.2. Parallelization

As we have mentioned above, during the random walk, finding the minimum value from N iterations and comparing the minimum value w to all previously stored values can be separated. This feature makes the new algorithm suit for distributed computation.

However, Pollard's rho method is inherently serial in nature; one must wait for the current iteration of the function F to complete before the next can begin. Each value in the sequence totally depends on the previous value and the iteration rules. In discussing the rho method for factorization, Brent considered running many processors in parallel each producing an independent sequence and noted that "Unfortunately, parallel implementation of the "rho" method does not give linear speedup" [26]. Analogous comments apply to the rho method for computing logarithms and the generalized rho method for collision search. Notice that here each parallel processor is producing its own sequence of points independently of the others and each particular processor does not increase the probability of success of any other processor. For the corresponding picture, with high probability, each processor draws a different "rho" that never intersect with each other. There is a little chance that different processors may intersect with each other.

van Oorschot and Wiener [11] showed that the expected speedup of the direct parallelization of Pollard's rho method, using m processors, is only a factor of $\sqrt{m}$. This is a very inefficient use of parallelization. They provided a modified version of Pollard's rho method and claimed that it can be linearly parallelized with the distinguished point method; that is, the expected running time of the modified version, using m processors, is roughly $\sqrt{\pi |G|/2}/m$ group operations.

In the modified version, to perform a parallel collision search each processor proceeds as follows. Select a random starting point $Y_0 \in G$, and produce the trail of points $Y_{i+1} = F(Y_i)$, for $i = 0, 1, 2, \ldots$, until a distinguished point Y_d is reached based on some easily testable distinguished property. Store distinguished point, and start producing a new trail from a new random starting point. Unfortunately, the new algorithm is not efficient for the parallelized modified version of Pollard's rho method. The key point is that there is a probability that the new algorithm fails to detect the collision while it actually happened, which cannot be efficiently solved like the serial version.

However, for Pollard's lambda method, the new algorithm can be efficiently parallelized with linear speedup. We present here a modified version of parallelized Pollard's lambda method from [11]. Assume we have m processors with m even. Then, instead of one tame and one wild kangaroo, we work with two herds of kangaroos, one herd of $m/2$ tame kangaroos and one herd of $m/2$ wild kangaroos, with one kangaroo on each processor. Each kangaroo starts from a different point, stores a minimum value every N iterations, just like the serial version. A center server collects all the minimums, and tries to find a collision between the tame kangaroo minimums and the wild kangaroo minimums By choosing a reasonable integer N, the new algorithm provides an optimal time-space trade-off method for collision detection.

5. Experiments

We implemented Pollard's rho method with elliptic curve groups over prime fields using SAGE [27], which is an open source computer algebra software. Obviously, such experiments can also be done for DLP on a multiplicative subgroup G of finite field $\mathbb{F}_q$. We compared the different performances between distinguished point method and the new algorithm. In this section, we describe these experiments and analyse the results.

For our experiments, we briefly introduce the elliptic curve groups over prime fields and the notation we use in the following. Let q be a prime, and let $\mathbb{F}_q$ denote the field $\mathbb{Z}_q$ of integers modulo q. Let $a, b \in \mathbb{F}_q$ such that $4a^3 + 27b^2 \neq 0$. Then the elliptic curve $E_{a,b}$ over $\mathbb{F}_q$ is defined through the equation

$$E_{a,b} : y^2 = x^3 + ax + b. \tag{5.1}$$

The set of all solutions $(x, y) \in \mathbb{F}_q \times \mathbb{F}_q$ of this equation, together with the element $\mathcal{O}$ called the "point at infinity," forms a finite abelian group which we denote by $E_{a,b}(\mathbb{F}_q)$. Usually, this group is written additively. Let $P \in E_{a,b}(\mathbb{F}_q)$ be a point of prime order n, and let G denote the subgroup of E generated by P. Given $Q \in G$, determine the integer $0 \leq k < n$ such that $Q = kP$.

For the iteration function, we use Teske's r-adding walk and set $r = 20$; that is, we divide the group G into 20 subsets: $S_1, S_2, \ldots, S_{20}$. Define the iteration function as follows:

$$F(Y) = Y + (m_i P + n_i Q) \quad \text{for } Y \in S_i,\ i \in [1, 20], \tag{5.2}$$

where m_i and n_i randomly chosen from $[0, n-1]$ and $(m_i P + n_i Q)$ can be precomputed for $i = 1, 2, \ldots, 20$. This means it only needs one group operation for each iteration.

Let $W = (x, y)$ be any point of G; we define the partition of G into r subsets $S_1, S_2, \ldots, S_r$ as follows. First we compute a rational approximation A of the golden ratio $(\sqrt{5} - 1)/2$, with a precision of $2 + \lfloor \log_{10}(qr) \rfloor$ decimal places. Let

$$u^* : G \longrightarrow [0, 1), \qquad (x, y) \longrightarrow \begin{cases} Ax - \lfloor Ax \rfloor & \text{if } W \neq \mathcal{O}, \\ 0 & \text{if } W = \mathcal{O}, \end{cases} \tag{5.3}$$

where $Ax - \lfloor Ax \rfloor$ is the nonnegative fraction part of Ax. Then let

$$\begin{gathered} u : G \longrightarrow \{1, 2, \ldots, r\}, \qquad u(W) = \lfloor u^*(W) \cdot r \rfloor + 1, \\ S_i = \{W \in G : u(W) = i\}. \end{gathered} \tag{5.4}$$

This method is originally from Knuth's multiplicative hash function [28] and suggested by Teske [29]. From the theory of multiplicative hash functions, we know that, among all numbers between 0 and 1, choosing A as a rational approximation of $(\sqrt{5}-1)/2$ with a sufficiently large precision leads to the most uniformly distributed hash values, even for nonrandom inputs.

The purpose of our experiments is to evaluate the expected numbers of steps until a match is found with different collision detection methods, that is, distinguished point method and the new algorithm, under the same expected space requirement. Generally, we randomly choose a big prime number q, where q is in certain range. Then we randomly choose the parameters a and b, where $a, b \in \mathbb{F}_q$, which determine the unique elliptic curve $E_{a,b}$ over $\mathbb{F}_q$. We will check whether the order of group $E_{a,b}(\mathbb{F}_q)$ has large prime factor n in certain range. If not, repeat the above procedures until we get a prime order subgroup G of $E_{a,b}(\mathbb{F}_q)$. Then we set the generator P of G and choose a random point Q of G. When using Pollard's rho method to compute this discrete logarithm, we count the number of steps we performed until a match is found with different collision detection methods on the same case. Then we determine the ratio R of the number of steps and $\sqrt{n}$. We repeat it a couple of times with the same P but several randomly chosen Q's. Furthermore, for practical reasons, we do the above procedures with a couple of groups, where the group order p is between 2^{31} and 2^{36}. We have Algorithm 2.

More precisely, for each $i \in [31, 36]$, we generate 20 elliptic curves, where each of them has a subgroup G of prime order n, such that $n \in [2^i, 2^{i+1}]$. Then for, each group G, we generate 100 to 3200 DLPs with the same generator P but randomly generated Q. The number of elliptic curves and instances of DLPs computed is given in Table 1. For each DLP, we use Teske's r-adding walk for iteration function and find the match using distinguished point method and the new algorithm simultaneously. Once reaching a match, we compute the ratio R_l as (the number of steps until match is found)$/\sqrt{n}$. Then we compute the average ratio R_j of all DLPs over the same elliptic curve. Finally, we count the average ratio R_i of all DLPs with the same i, where $i \in [31, 36]$ and $n \in [2^i, 2^{i+1}]$.

Now, let us explain the parameters for distinguishing property and the new algorithm in more detail. In our experiments, we compute the average ratio of (number of steps until match is found)$/\sqrt{n}$ under the same space requirement. To do this, generally we first define the distinguishing property and then compute the expected storage requirements. With the same storage requirements, we can deduce the parameter N for the new algorithm.

Input: Iteration function $F : G \to G$
Output: The average ratio (number of steps)$/\sqrt{n}$: R_{i1} and R_{i2} for distinguished point method and the new algorithm, respectively
(1) **for** $i = 31$ to 36 **do**
(2) **for** $j = 1$ to 20 **do**
(3) **repeat**
(4) Choose a random prime number $q \in [2^{i+1},\ 2^{i+3}]$
(5) Choose two random numbers $a, b \in \mathbb{F}q$, where $4a^3 + 27b^2 \neq 0$
(6) $n \leftarrow$ the largest prime factor of $\#E_{a,b}$
(7) **until** $2^i \leq n \leq 2^{i+1}$
(8) Choose a random point $W \in E_{a,b}$, where the order of W equal to $\#E_{a,b}$
(9) $P \leftarrow (\#E_{a,b}/n) * W$ (the generator of G)
(10) **for** $l = 1$ to $3200/2^{i-31}$ **do**
(11) Choose a random number $c \in [0,\ n-1]$, $Q \leftarrow c * P$
(12) Choose a random point in G be the initial point Y_0
(13) $k \leftarrow 1$
(14) **repeat**
(15) $Y_k \leftarrow F(Y_{k-1})$
(16) Check whether the Hamming weight of Y_k less than certain value
(17) Check whether the x-coordinate of Y_k is a minimum value
(18) **if** there is a match among distinguished points **then**
(19) $k_1 \leftarrow k$
(20) **end if**
(21) **if** there is a match among minimum values **then**
(22) $k_2 \leftarrow k$
(23) **end if**
(24) **until** Both of two methods have found the match
(25) $R_{l1} \leftarrow k_1/\sqrt{n}$ for distinguished point method
(26) $R_{l2} \leftarrow k_2/\sqrt{n}$ for the new algorithm
(27) **end for**
(28) $R_{j1} \leftarrow (\sum R_{l1})/3200/2^{i-31}$ for distinguished point method
(29) $R_{j2} \leftarrow (\sum R_{l2})/3200/2^{i-31}$ for the new algorithm
(30) **end for**
(31) $R_{i1} \leftarrow (\sum R_{j1})/20$ for distinguished point method
(32) $R_{i2} \leftarrow (\sum R_{j2})/20$ for the new algorithm
(33) **end for**

Algorithm 2: Experiments for distinguished point method and the new algorithm.

For example, if $i = 36$, which means n, the order of G is a 36-bit prime number. According to [19], we are expected to take $1.292\sqrt{n}$ iterations before reaching a match. We define the distinguishing property as the Hamming weight of normal-basis representation of x-coordinate of the point less than or equal to 9. Each point has probability almost exactly $(\binom{36}{9} + \binom{36}{8} + \binom{36}{7} + \cdots)/2^{36} \approx 2^{-8.99}$ of being a distinguished point, that is, $\theta = 2^{-8.99}$; that is, to find a collision it is expected to compute $1.292 * 2^{-8.99}\sqrt{n}$ distinguished points. To keep the same storage requirements, we set $N = 1/\theta = 2^{8.99} \approx 508$ for the new algorithm.

The experimental results are given in Table 2. It shows that on average using the new algorithm for collision detection can reduce the number of iterations until a match is found from $1.309\sqrt{|G|}$ to $1.295\sqrt{|G|}$ under the same space requirements for the single rho method.

Under the same expected storage requirements, the main reason for the different performances of the distinguished point method and the new algorithm is that the distinguished

Table 1: Number of elliptic curves and instances of DLPs.

Bits	No. of elliptic curves	No. of DLPs per curve
31	20	3200
32	20	1600
33	20	800
34	20	400
35	20	200
36	20	100

Table 2: Different performance for distinguished point method and the new algorithm.

Bits	No. of DLPs	Ratio for distinguished point	Ratio for the new algorithm
31	64000	1.309	1.295
32	32000	1.309	1.295
33	16000	1.310	1.296
34	8000	1.310	1.297
35	4000	1.309	1.295
36	2000	1.310	1.294
Average	126000	1.309	1.295

points may be not uniformly distributed in the pseudo-random walk, also the points in subset D may be not uniformly distributed in G, which always results in more iterations requirement. while, for the new algorithm, each stored minimum value represents N successive values, which leads to an equal-interval distribution.

6. Conclusion

In this paper, we proposed an optimal time-space trade-off method for collision detection in the pseudo-random walk when computing discrete logarithms with Pollard's rho method. We discussed the new algorithm both in theoretical analysis and in practical experiments. By comparison to other methods, it shows that the new algorithm is more efficient than previous methods. Unfortunately, the only practical application of the new idea is with the parallelized lambda method and it does not work with the parallelized rho method. As a further work, we would like to explore the performances of the new algorithm in other applications.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China (no. 61070168). The authors would like to thank the reviewers for their helpful comments and suggestions.

References

[1] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[2] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[3] FIPS 186-2, "Digital signature standard," Tech. Rep. 186-2, Federal Information Processing Standards Publication, 2000.
[4] C. P. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.
[5] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987.
[6] V. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology: Proceedings of Crypto'85*, vol. 218 of *LNCS*, pp. 417–426, Springer, New York, NY, USA, 1986.
[7] A. Menezes, P. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, Fla, USA, 1996.
[8] J. M. Pollard, "Monte Carlo methods for index computation mod p," *Mathematics of Computation*, vol. 32, no. 143, pp. 918–924, 1978.
[9] R. Gallant, R. Lambert, and S. Vanstone, "Improving the parallelized Pollard lambda search on anomalous binary curves," *Mathematics of Computation*, vol. 69, no. 232, pp. 1699–1705, 2000.
[10] M. Wiener and R. Zuccherato, "Faster attacks on elliptic curve cryptosystems," in *Selected Areas in Cryptography'98*, vol. 1556 of *LNCS*, pp. 190–200, Springer, Berlin, Germany, 1998.
[11] P. van Oorschot and M. Wiener, "Parallel collision search with cryptanalytic applications," *Journal of Cryptology*, vol. 12, no. 1, pp. 1–28, 1999.
[12] R. P. Brent, "An improved Monte Carlo factorization algorithm," *BIT*, vol. 20, no. 2, pp. 176–184, 1980.
[13] G. Nivasch, "Cycle detection using a stack," *Information Processing Letters*, vol. 90, no. 3, pp. 135–140, 2004.
[14] J. J. Quisquater and J. P. Delescaille, "How easy is collision search? Application to DES," in *Proceedings of the Advances in Cryptology—Eurocrypt*, vol. 434 of *Lecture Notes in Computer Science*, pp. 429–434, Springer, New York, NY, USA, 1989.
[15] J. J. Quisquater and J. P. Delescaille, "How easy is collision search. New results and applications to DES," in *Proceedings of the Advances in Cryptology—Crypto*, vol. 435 of *Lecture Notes in Computer Science*, pp. 408–413, Springer, New York, NY, USA, 1989.
[16] M. E. Hellman, "A cryptanalytic time-memory trade-off," *IEEE Transactions on Information Theory*, vol. 26, no. 4, pp. 401–406, 1980.
[17] B. Harris, "Probability distributions related to random mappings," *Annals of Mathematical Statistics*, vol. 31, pp. 1045–1062, 1960.
[18] S. C. Pohlig and M. E. Hellman, "An improved algorithm for computing logarithms over GF(p) and its cryptographic significance," *IEEE-Transactions on Information Theory*, vol. 24, no. 1, pp. 106–110, 1978.
[19] E. Teske, "Speeding up Pollard's rho method for computing discrete logarithms," in *Algorithmic Number Theory Symposium (ANTS IV)*, vol. 1423 of *LNCS*, pp. 541–553, Springer, New York, NY, USA, 1998.
[20] S. Bai and R. P. Brent, "On the efficiency of Pollard's rho method for discrete logarithms," in *CATS 2008*, J. Harland and P. Manyem, Eds., pp. 125–131, Australian Computer Society, 2008.
[21] C.-P. Schnorr and H. W. Lenstra Jr., "A Monte Carlo factoring algorithm with linear storage," *Mathematics of Computation*, vol. 43, no. 167, pp. 289–311, 1984.
[22] E. Teske, "A space efficient algorithm for group structure computation," *Mathematics of Computation*, vol. 67, no. 224, pp. 1637–1663, 1998.
[23] D. V. Bailey, L. Batina, D. J. Bernstein et al., "Breaking ECC2K-130," Tech. Rep. 2009/541, Cryptology ePrint Archive, 2009.
[24] J. M. Pollard, "A Monte Carlo method for factorization," *BIT*, vol. 15, no. 3, pp. 331–335, 1975.
[25] D. E. Knuth, *The Art of Computer Programming*, vol. 2, Addison-Wesley, Reading, Mass, USA, 3rd edition, 1997.
[26] R. P. Brent, "Parallel algorithms for integer factorisation," in *Number Theory and Cryptography*, J. H. Loxton, Ed., vol. 154 of *London Mathematical Society Lecture Note Series*, pp. 26–37, Cambridge University, Cambridge, UK, 1990.
[27] "SAGE: an open source mathematics software," http://www.sagemath.org/.
[28] D. E. Knuth, *The Art of Computer Programming*, vol. 3, Addison-Wesley, Reading, Mass, USA, 2nd edition, 1981.
[29] E. Teske, "On random walks for Pollard's rho method," *Mathematics of Computation*, vol. 70, no. 234, pp. 809–825, 2001.