Solving Discrete Logarithm Problem in an Interval Using Periodic Iterates

Jianing Liu[1,2,3] and Kewei Lv[1,2,3](✉)

[1] State Key Laboratory of Information Security,
Institute of Information Engineering, Chinese Academy of Sciences,
Beijing, China
{liujianing,lvkewei}@iie.ac.cn
[2] Data Assurance and Communication Security Research Center,
Chinese Academy of Sciences, Beijing, China
[3] University of Chinese Academy of Sciences, Beijing, China

Abstract. The Pollard's kangaroos method can solve the discrete logarithm problem in an interval. We present an improvement of the classic algorithm, which reduces the cost of kangaroos' jumps by using the sine function to implement periodic iterates and giving some pre-computation. Our experiments show that this improvement is worthy of attention.

Keywords: Discrete logarithm problem · Pollard's kangaroos method
Pollard's rho method

1 Introduction

The discrete logarithm problem (DLP) in a group G is to find the integer n such that $g^n = h$ holds given $g, h \in G$. As one of the most important mathematical primitives in modern cryptography, there are some classic algorithms to solve it, such as the Pollard's rho method, the index calculus method, and the Pollard's kangaroos method as well [1]. It's interesting to study solving discrete logarithm problem of the given interval in the practical cryptography system. The discrete logarithm problem in an interval is defined as following:

Definition 1 (DLP in an interval). Let p be a prime number and G be a cyclic subgroup of order q in F_p^*. Given the generator g and an element h of G, and an integer N less than the order of g, it is assumed that there exists an unknown integer n in the interval $[0, N]$ such that $h = g^n$ holds. To compute n.

Indeed, some instances belonging to this case had been studied, such as the DLP with c-bit exponents (c-DLSE) [2–4], Boneh-Goh-Nissim homomorphic encryption scheme [5], counting points on curves or abelian varieties over finite fields [6], the analysis of the strong Diffie-Hellman problem [1, 7], and side-channel or small subgroup attacks [8, 9] and reference therein. Pollard's rho algorithm costs time $O(\sqrt{n})$ to solve it. [4] improves Pollard's kangaroos algorithm to solve DLP in an interval of size N with expected running time $(2 + \mathrm{O}(1))\sqrt{\mathrm{N}}$ group operations and polynomial storage. Galbraith et al. improve it by increasing the number of kangaroos showing that when the number of kangaroos is four, total number of jumps is optimal and number of group operations is $(1.714 + \mathrm{O}(1))\sqrt{\mathrm{N}}$ [10]. [11] uses series of small integer multiplications to replace every multiplication of elements of a group, which reduces the cost of each jump in Pollard's rho algorithm and to determine to compute a complete multiplication according to whether some function values belong to the set of distinguished points or not. The definition of distinguished points is originally introduced in [12] for time-memory trade-off, which is some elements of group G satisfying a certain condition such that these points can be checked easily. [11] showed that when the related results meet the pre-defined distinguished point condition, we make a complete integer multiplication operation, thereby reducing the number of complete multiplications and time cost of each jump. A preprocessing storage size is $O\left((\log p)^{r+1} \cdot \log\log p\right)$ and running time is at least 10 times faster than the original algorithm.

Contribution. We use the sine-function to implement periodic iterates and give some pre-computation to reduce the cost of Pollard's kangaroos algorithm obviously. we can reduce $||p||^2$ bit operations of a complete integer multiplication to at most $d||\varepsilon||\,||\eta^2||$ bit operations, where $d = \log_\varepsilon p$. The pre-defined distinguished point condition is $[\eta - k, \eta]$, where $k = \log\eta + v,\ v$ is an integer satisfying $0 \le v < \eta - \lfloor \log\eta \rfloor$. Furthermore, we also properly increase the number of kangaroos to reduce the total number of jumps, which improve both the time cost and the total number of jumps. Compared with the classic algorithm, the efficiency is noticeably improved.

2 Pollard's Kangaroos Algorithm

The Pollard's kangaroos algorithm [13] is based on random walk and each kangaroo jumps one step at a time. The main process is: First, fix a set of small integers of k elements $S = \{s_1, s_2, \ldots, s_k\}$, which is also considered as the set of the distances of jump steps such that the mean value of the elements of S is about $\sqrt{N}$ and k is a small integer. We randomly select some elements from group G to form the distinguished set $D = \{g_1, g_2, g_{3,\ldots,} g_t\}$, such that the size of D is approximately $|D|/|G| = \frac{c}{\sqrt{N}}$, where c is a constant and $c \gg 1$. Define a random map f from G to S. mA kangaroo's jumps corresponds to a sequence in $G, g_{i+1} = g_i g_i^{f(g)},\ i = 0, 1, 2, \ldots$, starting from the given g_0. Let $d_0 = 0$, $d_{i+1} = d_i + f(g_i)$, $i = 0, 1, 2, \ldots$. Then, d_i is the sum of distances of first i jumps of kangaroo and $g_i = g_0 g_i^d,\ i = 0, 1, 2, \ldots$. The algorithm requires $2\sqrt{N}$ group operations and a small amount of additional storage space.

For each jump, we need to compute a complete integer multiplication between two group elements, which costs about $||p||^2$ bit operations. When the product belongs to the distinguished set D, the values related to this jump will be stored for collision detection; otherwise, the related values will not be stored. Thus, storage operations are not carried out every time, so it is not necessary to do a full product for each jump.

3 The Improved Pollard's Kangaroos Algorithm

In Pollard's kangaroos algorithm, the cost of each jump is complete multiplication operation of $\|p\|^2$ bits. We improve computational cost of jumpsusing pre-computation and periodic iterative functions, so that the total cost is reduced. Let p be a prime, n be an integer, and $G = <g>$ be a cyclic subgroup of order q of F_p^*. Given the generator g of G, element h and positive integer N, suppose there exists an unknown integer n in the interval [0, N] such that $h = g^n$ holds, the algorithm to compute n is given in the following.

Let η denotes a small integer, we set $\Gamma = \{t_1, t_2, \ldots, t_\eta\}$, which is also considered as the set of the distances of jump steps and the mean value of the elements of Γ is about $\sqrt{N}$. Let the index set be $S = \{1, 2, \ldots, \eta\}$ and set $\{M_s = g^{t_s} | t_s \in \Gamma\}$. Fix a small positive integer l and precompute $M_l = \{M \cup \{1\}\}^l$. Initially, both tables L_t and L_w are null. We choose small integers d and ε such that $d = \lceil \log_\varepsilon p \rceil$. In order to facilitate the calculation of the discrete logarithm, we pre-calculate the M_l before the algorithm runs and store it in the appropriate table. The size of the set M_l does not exceed $\binom{l+\eta}{\eta}$.

Next, we define label function $\tau : G \to S$ such that $\tau(g) = \frac{g \bmod p}{p} \cdot \eta + 1$.

Given $x, y \in G$, for each $0 \leq i \leq d{-}1$, write

$$x = \sum_{i=0}^{d-1} x_i \varepsilon^i, \quad \text{where} \quad 0 \leq x_i < \varepsilon \tag{1}$$

and

$$t_i = \varepsilon^i y \bmod 360, \quad \text{then } 0 \leq t_i < 360. \tag{2}$$

Define the auxiliary label function $\bar{\tau} : G \times M_l \to S$

$$\bar{\tau}(x, y) = \sum_{i=0}^{d-1} x_i \lfloor |u \sin(t_i)| \rfloor \bmod \eta + 1 \tag{3}$$

where $u \geq \eta^2$, for the convenience of computation, u is taken as η^2. For any given $g \in G$ and $M \in \mathrm{M_l}$, the time computing $\bar{\tau}(g, M)$ is much less than the cost of computing $g \cdot M$.

We define index function and auxiliary index function $\bar{s} : G \times M_l \to S$, such that $s : G \to S$ is a surjective and pre-image is approximately uniform,, and $\bar{s}(g, M) = s(gM)$. Let $\bar{s} = \bar{\tau}$. The pre-defined distinguished point condition is defined as the interval $[\eta - k, \eta]$, where $k = \lfloor \log \eta \rfloor + v$, $0 \leq v < \eta - \lfloor \log \eta \rfloor$. Generally, the value of v is a small integer.

The jumping process of a kangaroo is a sequence table of G, G, $g_{i+1} = g_i \cdot M_{s(g_i)}$ for $i \geq 0$. The algorithm starts at an initial value $g_0 \in G$, and the initial $s_0 = s(g_0))$ is randomly selected from the set S. Since $g_1{=}g_0 \cdot M_{s(g_0)}$, we can get the next index $s_1 = s(g_1) = s(g_0 M_{s(g_0)}) = \bar{s}(g_0, M_{s(g_0)})$ by calculating $\bar{s}(g_0, M_{s(g_0)})$ without computing $g_0 \cdot M_{s(g_0)}$. If the computed index value s_1 satisfies distinguished point condition, we can calculate $g_2 = g_0 \cdot M_{s_0}M_{s_1}$, $M_{s_0}M_{s_1} \in M_l$ has been pre-computed without computing $g_1 = g_0 \cdot M_{s_0}$. In the next process, for each iteration to complete a jump, the index value s_2, s_3, etc, can be calculated in the same way, and we do not compute the complete multiplication until that the pre-defined distinguished point condition is met, that is, the value of the function $\bar{s}$ falls in the interval $[\eta - k, \eta]$. At same time, we get the corresponding point $g_{\mathrm{i}} = g_0 M_{s_0} \ldots M_{s_{i-1}}$, where $\mathrm{M}_{\mathrm{s}_0} \ldots \mathrm{M}_{\mathrm{s}_{\mathrm{i}-1}} \in \mathrm{M}_\mathrm{l}$ has been basically pre-computed and can be obtained through look-up the table. If the collision has not occurred for l iterations, i.e., $\mathrm{g}_{\mathrm{l}+1} = \mathrm{g}_0 M_{\mathrm{s}_0} \ldots M_{\mathrm{s}_\mathrm{l}}$, the computed value of g_{l+1} will be stored in the table and the algorithm can be re-executed with g_{l+1} as a new starting point.

We denote two kangaroos as T and W. T and W jump respectively from the midpoint of the interval (i.e., $g_0 = g^{N/2}$) and $g'_0 = h$ to the right side of the interval. The jumping process of T and W are alternately executed. The branches T and W randomly select the initial values $s_0 = s(g_0))$ and $s'_0 = (g'_0)$ from the set S respectively and set their own initial jumping points and initial distances of the jump step of the two branches be $g_0 = M_{s_0}$, $g = M_{s'_0}$ and $\mathrm{d}_0(\mathrm{T}) = 0$, $\mathrm{d}_0(\mathrm{W}) = 0$. When $i > 1$, we have $d_i(T) = \sum_{i=0}^{i-1} t_{s_i}$, $d_j(W) = \sum_{i=0}^{j-1} t_{s'_j}$. Let T_i and W_j denote the i and j-th jump of the two branches respectively. i and j start from 0. T_0, W_0, T_1, W_1, T_2, W_2,...,T_{i-1}, $W_{i\text{-}1}$ are alternately executed. When the distinguished point condition is satisfied, the current jumping point's value of branch T or W will be computed. Then we can get the triplets that are $(g_i, d_i(T), T)$ or g, $d_j(W), W$ and store it into the corresponding table L_t or L_w at index g_i and g_j respectively. A collision occurs when g_i is accessed by a different type of branch kangaroo, so the value of $n = N/2 + d_i(T) - d_j(W)$ can be computed and the algorithm terminates.

Running Time. From Eq. (3), we can know the time of calculating $\bar{\tau}(x, y)$ includes d multiplications of modulo η, d-1 additions of modulo η, and d sinusoidal operations. Considering the time of calculating a sine operation as a constant C_0 and ignoring the relatively small cost of addition, time of computing $\bar{\tau}$ is about $d\mathrm{Mul}(||\eta||) + (\theta + 1/l)\mathrm{Mul}(||p||) + dC_0$, where θ is the probability that a point is a distinguished point, and l is the maximum number of iterations of the pre-process. Notice that the time of calculating sine function is neglected after processing in (2), the time of a jump is $d\mathrm{Mul}(||\eta||) + (\theta + 1/l)\mathrm{Mul}(||p||)$, where θ is the probability that a point is a distinguishable point, and l is the maximum number of iterations in the preprocessing table. Since the total number of jumps is $N/(2m) + 2m + 2/\theta$, the total time is $\{d\mathrm{Mul}(||\eta||) + (\theta + 1/l)\mathrm{Mul}(||p||)\} * \{N/(2m) + 2m + 2 \cdot 1/\theta\}$. From (1) and (3), we need about $d||\varepsilon||||\eta^2||$ bit operations required for a complete multiplication, obviously smaller than the $||p||^2$ bit operations required for a complete multiplication of the original algorithm. Usually we have the comparison results, seeing Table 1.

We can take the pre-defined distinguished point condition as $[\eta - k, \eta]$, the number of group operations is about $\log\eta/\eta(2 + \mathrm{O}(1))\sqrt{\mathrm{N}}$. [10] improves the classic Pollard's kangaroos algorithm by increasing the number of kangaroos such that the total number of jumps is reduced and the probability of collision is increased. When the number of kangaroos is four, the total number of jumps is optimal and number of group operations is $(1.714 + \mathrm{O}(1))\sqrt{\mathrm{N}}$.

Table 1. Time contrast in case of Two Kangaroos

Algorithm	Number of group operations	Time cost of a jump	Computation cost
Pollard's	$(2+O(1))\sqrt{N}$	$\mathrm{Mul}(\|\|p\|\|)+\|f\|$	$\|\|p\|\|^2$ bit operations of a jump
The improved	$\log\eta/\eta(2+O(1))\sqrt{N}$	$d\mathrm{Mul}(\|\|\eta\|\|)+(\theta+1/l)\mathrm{Mul}(\|\|p\|\|)$	$d\|\|\varepsilon\|\|\|\|\eta^2\|\|$ bit operations of a jump

4 Experiments on Improved Pollard's Kangaroos Algorithm

Given a 32-bit prime number p = 2147483659, g = 29, and take $\varepsilon = 8, N = 50$. Since $\sqrt{N}$ = 7, here we set η = 13 and $\Gamma = \{1, 2, 3, \ldots, 13\}$. Then $k = \lfloor \log\eta \rfloor = 3$, so the interval $[\eta - k,\ \eta] =$ [10, 13]. The distinguished point condition is [10, 13]. $M = \{g^1, g^2, \ldots, g^{13}\}$. Given h = 44895682, then our task is to seek x in the interval [1, N] such that $29^x \bmod p = 44895682$. Here, we set $l = 3$ and precompute $M_l = \{\{29^1, 29^2, \ldots, 29^{13}\} \cup \{1\}\}^3$. We show some instances of experiment for different size primes p to display advantage of the improved algorithm in Table 2.

Table 2. Experiment cost

$\|\|p\|\|$	Prime p	Total number of jumps	Number of complete multiplications
64 bits	15509012368832652833	60	37
128 bits	292087288550973971472931860508592710703	73	42
256 bits	92444955114635498485587226229817979873 64101890317331031445763518698667993782 7	115	62

5 Conclusion

In this paper, we improve Pollard's kangaroos algorithm using pre-defined distinguished point condition and periodic iterations. We reduce the cost of kangaroos' jumps by using the sine function to iterate periodically and pre- computation instead of multiplication between the elements of a group. The related function definition of the algorithm is not limited to a certain interval, the improved algorithm can be extended for the calculation of the discrete logarithm problem in the usual case.

Acknowledgements. This work is partially supported by National Key R&D Program of China (2017YFB0802502) and NSF (No. 61272039).

References

1. McCurley, K.: The discrete logarithm problem. In: Proceedings of the Symposium in Applied Mathematics, pp. 49–74. AMS (1990)
2. Gennaro, R.: An improved pseudo-random generator based on discrete log. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 469–481. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_29
3. Patel, S., Sundaram, G.S.: An efficient discrete log pseudo random generator. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 304–317. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055737
4. van Oorschot, P.C., Wiener, M.J.: On Diffie-Hellman key agreement with short exponents. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 332–343. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68339-9_29
5. Boneh, D., Goh, E.-J., Nissim, K.: Evaluating 2-DNF formulas on ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–341. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30576-7_18
6. Gaudry, P., Schost, É.: A low-memory parallel version of Matsuo, Chao, and Tsujii's Algorithm. In: Buell, D. (ed.) ANTS 2004. LNCS, vol. 3076, pp. 208–222. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24847-7_15
7. Cheon, J.H.: Security analysis of the strong Diffie-Hellman problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 1–11. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_1
8. Gopalakrishnan, K., Thériault, N., Yao, C.Z.: Solving discrete logarithms from partial knowledge of the key. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 224–237. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77026-8_17
9. Lim, C.H., Lee, P.J.: A key recovery attack on discrete log-based schemes using a prime order subgroup. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 249–263. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052240
10. Galbraith, S.D., Pollard, J.M., Ruprai, R.S.: Computing discrete logarithm in an interval. Math. Comput. **82**(282), 1181–1195 (2013)
11. Cheon, J.H., Hong, J., Kim, M.: Speeding up the Pollard rho method on prime fields. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 471–488. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89255-7_29
12. Quisquater, J.-J., Delescaille, J.-P.: How easy is collision search? Application to DES. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 429–434. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-46885-4_43
13. Pollard, J.M.: Kangaroos, Monopoly and Discrete Logarithms. J. Cryptol. **4**, 437–447 (2000)