Journal of Computer Science 10 (8): 1575-1581, 2014
ISSN: 1549-3636

doi:10.3844/jcssp.2014.1575.1581 Published Online 10 (8) 2014 (http://www.thescipub.com/jcs.toc)

A VARIANT OF POLLARD'S RHO ATTACK ON ELLIPTIC CURVE CRYPTOSYSTEMS

Siham Ezzouak, Mohammed Elamrani and Abdelmalek Azizi

Department of Mathematics and Computer Science,
Faculty of Science, University Mohammed First, Oujda, BP 60000, Morocco

Received 2014-02-18; Revised 2014-04-05; Accepted 2014-04-09

ABSTRACT

Elliptic Curve cryptosystems appear to be more secure and efficient when requiring small key size to implement than other public key cryptosystems. Its security is based upon the difficulty of solving Elliptic Curve Discrete Logarithm Problem (ECDLP). This study proposes a variant of generic algorithm Pollard's Rho for finding ECDLP using cycle detection with stack and a mixture of cycle detection and random walks. The Pollard's Rho using cycle detection with stack requires less iterations than Pollard's Rho original in reaching collision. Random walks allow the iteration function to act randomly than the original iteration function, thus, the Pollard rho method performs more efficiently. In practice, the experiment results show that the proposed methods decreases the number of iterations and speed up the computation of discrete logarithm problem on elliptic curves.

Keywords: Cycle Detection, Discrete Logarithm Problem, Elliptic Curve, Pollard Rho Method, Random Walk

1. INTRODUCTION

Elliptic curves over finite fields have been proposed by Diffie-Hellman to implement key passing scheme and elliptic curves variants for digital signature. The security of this cryptosystem is linked to the difficulty to solve elliptic curve discrete logarithm problem and if this problem is resolved the cryptosystem is broken.

Although there are several attacks against this cryptosystem such as Baby-Step Giant-Step (Shanks, 1971), Pollard's Rho method and its parallelized variant, their complexity is the square root of the prime order of the generating point used (Harrison, 2010). Up to now, Pollard's Rho method is known as the best method to resolve the discrete logarithm problem on general groups, specifically elliptic curve. Hence automorphism of the group (Duursma *et al*., 1990), parallelization (Oorschot and Wiener, 1999), iteration function (Teske, 1998; 2001) or cycle detection (Brent, 1980; Cheon *et al*., 2012) are used to improve this attack. In this study, we try to introduce a variant of Pollard's Rho attack using the new cycle detection proposed by (Nivasch, 2004) and the random walks proposed by Teske. After that, we analyze the running time and implement the new algorithm. The remainder of this study is proceded as follow: Section 2 introduces some basic definitions for the elliptic curves, Floyd's algorithm and Pollard's Rho algorithm. Section 3 describes how Pollard's Rho algorithm may be modified using Nivash's cycle detection instead of Floyd's algorithm. We explain how to introduce random walks on the modified Pollard's Rho and the algorithms are compared in section 4.

2. BACKGROUND

This section introduces the elliptic curve cryptosystem, Floyd finding cycle algorithm Floyd (1962) and Pollard's Rho method (Pollard, 1978). The Pollard's Rho method uses iteration function to build sequence of elements and it uses cycle detection to find match or collision. The match leads to the solution of ECDLP. In fact, this method is based on a

Corresponding Author: Siham Ezzouak, Department of Mathematics and Computer Science, Faculty of Science, University Mohammed First, Oujda, BP 60000, Morocco

random walk and the Birthday Paradox which states that in a a set of 23 randomly chosen people, the chance that at least two of them share the same birthday is greater than 50%. Then, if random objects are selected with replacement from n objects, one may expect $\sqrt{\pi n / 2}$ rounds before an object is picked twice.

2.1. Elliptic Curve Cryptosystem

The addition rule of the group of elliptic curves is easy to be implemented. Therefore, algebraic formulas for the group law can be derived from the geometric description. A general elliptic curve E over finite field K has the form $y^2+axy+by = x^3+cx^2+dx+e$ where a, b, c, d and e are in K. The addition operation is defined over elliptic curves with the inclusion of a point O called point at infinity or identity.

Let p be a prime with p>3. Elliptic curves can be implemented over fields of characteristic 2 and 3 and enjoy many optimizations, but suffer from specialized discrete log attacks Coppersmith (1984) and should generally be avoided. Let F_p = GF(p) the Galois Field over p and a, b $\in$ F_p and satisfy the condition $4a^3+27b^2$ mod(p) $\neq$ 0 then an elliptic curve over the Galois field $E(\mathbb{F}_p)(a,b)$ is defined by equation $y^2 = x^3+ax+b$ mod(p) where x$\in$ F_p.

Let P = (x_1, y_1) and Q = (x_2, y_2) be two points in the elliptic curve $E(\mathbb{F}_p)(a, b)$, to compute the sum R = (x_3, y_3) of points P and Q we use explicit formulas:

- If P = O then R = Q
- If Q = O then R = P
- Otherwise

- If $x1 \neq x_2$ put
$\lambda = (y_1-y_2)\,(x_1-x_2)^{-1}$ then
$x_3 = \lambda^2-x_1-x_2$
$y_3 = \lambda\,(x_1-x_3)-y_1$
-If $x_1 = x_2$ and $y_1 = -y_2$ then R = O
- If $x_1 = x_2$ and $y_1 \neq y_2$ so P = Q put
$\lambda = (3x_1^2 + A)(y_1 + y_2)^{-1}$ then
$x_3 = \lambda^2 - x_1-x_2$
$y_3 = \lambda\,(x_1-x_3)-y_1$

The most expensive step is the division in the computation of λ.

Definition 1

Hankerson *et al*. (2004) The Elliptic Curve Discrete Logarithm Problem (ECDLP) is:

Given an elliptic curve E defined over a finite field $\mathbb{F}_p$, a point P $\in$ $E(\mathbb{F}_p)$ of order n and a point Q$\in\langle$P$\rangle$, find the integer l$\in$[0, n-1] such that Q = lP. The integer d is called the discrete logarithm of Q to the base P, denoted $l = \log_P Q$.

This problem is considered as hard mathematical problem like the Integer Factorisation Problem (IFP) and the logarithm problem in multiplicative group of finite field (DLP). All methods, proposed up to now which solve ECDLP, require exponential running time.

2.2. Floyd's Cycle-finding Algorithm

Instead of comparing each new Y_i to all previous ones and stores all elements until obtaining collision, It is better to choose Floyd's algorithm Floyd (1962) in order to minimize the memory requirement and running time. In fact, one computes pairs (Y_i, Y_{2i}) of points for i = 1,2,3... until finding $Y_i = Y_{2i}$. After computing a new pair, the previous pair can be discarded, thus the storage requirements are negligible.

Theorem 1

Knuth (1969) [exercises 6-7] for a periodic sequence Y_0, Y_1, Y_2,…, there exists an i>0 such that $Y_i = Y_{2i}$ and the smallest such i lies in the range $\mu \leq i \leq \mu+\lambda$. μ and λ are the preperiod and the period of the sequence Y_i respectively.

If we suppose that the sequence is generated by random function then the expected value of μ and λ is close to $\sqrt{\pi n / 8}$. As a consequence, μ+λ is around $\sqrt{\pi n / 2}$.

2.3. Pollard's Rho Algorithm

The idea of Pollard is that three possibilities are chosen in a random manner and the resulting sequence is sufficiently complicated to be regarded as a random mapping. Let us start with random point R_0 and build the sequence R_i with the iteration function f until the collision occurs. In fact, $E(\mathbb{F}_p)$ is finite, the sequence Ri become periodic after some iterations so there will be some indices i<j such that $R_i = R_j$, j-i is the period and $R_i, R_{i+1}, R_{i+2}, \ldots, R_j$ form a loop. For cycle detection, Floyd's method is used. The original Pollard's Rho method on elliptic curves is detailed bellow:

- Split $E(\mathbb{F}_p)$ into three disjoint sets S_1, S_2 and S_3 of roughly equal size
- Let $R_0 = a_0P + b_0Q$ with a_0 and b_0 two random integers in]0, n[and the iterative function f was defined as:

$$f(R_i) = R_{i+1} = \begin{cases} P + R_i \text{ if } R_i \in S_1 \\ 2R_i \quad \text{if } R_i \in S_2 \\ Q + R_i \text{ if } R_i \in S_3 \end{cases}$$

The sequence a_i and b_i can be computed as follow:

$$(a_{i+1}, b_{i+1}) = \begin{cases} (a_i + 1, b_i) \text{if } R_i \in S_1 \\ (2a_i, 2b_i) \text{if } R_i \in S_2 \\ (a_i, b_i + 1) \text{if } R_i \in S_3 \end{cases}$$

- Compute R_j and R_{2j} and compare them until a match is found using the iteration function f
- If $R_j = R_{2j}$, then $l = \frac{a_{2j} - a_j}{b_j - b_{2j}} (\text{mod } n)$ with $b_j \neq b_{2j}$. So ECDLP is resolved

In the last step with negligible probability, we can have $b_j = b_{2j}$. In this case, we restart the process with different starting points R_0.

Algorithm 1. Iteration function
1: function f (R): R1
2: if R ∈ S_1 then
3: R1← R + P
4: else if R ∈ S_2 then
5: R1 ← 2R
6: else
7: R1← R + Q
8: end if
9: return R1
10: end function
11: function f (a, b): a_1, b_1
12: if R ∈ S_1 then
13: a1←a + 1
14: else if R ∈ S_2 then
15: a1← 2a
16: b1 ← 2b
17: else
18: b1← b + 1
19: end if
20: return a1, b1
21: end function

Algorithm 2. Pollard's Rho with Floyd's cycle finding algorithm

Require: P, Q, S_1, S_2, S_3
Ensure: Integer l where Q = lP
1: a_0 ←random∈]0; n[
2: b_0 ←random ∈]0; n[
3: j← 0
4: R_0← a_0P+b_0Q
5: for all j such that $R_j \neq R_{2j}$ do
6: $(R_{j+1}, a_{j+1}, b_{j+1}) \leftarrow f(R_j), f(a_j, b_j)$
7: $(R_{2(j+1)}, a_{2(j+1)}, b_{2(j+1)}) \leftarrow f(f(R_{2j})), f(f(a_{2j}, b_{2j}))$
8: j ← j + 1
9: if $R_j = R_{2j}$ and $b_j \neq b_{2j}$ then
10: $l \leftarrow \frac{a_{2j} - a_j}{b_j - b_{2j}} \text{mod}(n)$
11: else if $b_j = b_{2j}$ then
12: a_0 ← random ∈]0; n[
13: b_0 ← random ∈]0; n[
14: j ← 0
15: end if
16: end for
17: Return l

The Pollard's Rho algorithm is known as the best algorithm to resolve ECDLP in the generic groups. If f supposed a random map than the expected number of iterations before a collision occurs is close to $\sqrt{\pi n / 2}$. In addition, the memory requirement is negligible and the running time is exponential.

2.4. Pollard's Rho Algorithm with Random Walks

The iteration function used in Pollard's Rho algorithm is not random enough (Knuth, 1969), So Teske proposed a better iteration function by applying more arbitrary multipliers. Divide $E(\mathbf{F}_p)$ into s disjoint subsets S_1, S_2,..,S_s of approximately the same size. A good choice for s seems to be around 20 (Teske, 2001). Choose 2s random integers a_i, b_i mod n.

Let $S_i = \{R(X,Y) \in E(\mathbf{F}_p)\}$ X(mod s) = i} and

$M_i = a_iP + b_iQ$ So the iteration function is defined as bellow:

$$f(R_j) = R_{j+1} = M_i + R_j \text{ if } R_j \in S_i$$

Moreover, the pseudocode of the iteration function is:

Algorithm 3. Iteration function

1: function f (R): R1
2: if R∈ S_i then
3: R1← R +M_i
4: end if
5: return R1
6: end function

7: function f (R, a, b): a1, b1
8: if R∈ S_i then
9: a1← a_i + a
10: b1←b_i + b
11: end if
12: return a1, b1
13: end function

The Pollard Rho original can be modified just by replacing the old iteration function by the new one.

3. POLLARD'S RHO ALGORITHM USING STACK

In this section, we explain how we can improve Pollard's Rho method using stack. First, we describe Nivasch's method for cycle detection (Nivasch, 2004). Second, we outline the Pollard's Rho modified and we show the two methods with random walks. Then we compare them and select the best one. Finaly, we implement the proposed algorithm and we make a comparison with the original algorithm.

3.1. Nivasch's Cycle-finding Algorithm

The stack has been created and starts out empty. At each step j, remove all the top entries (x_i, i) (pop) from the stack where $x_i > x_j$. If $x_i = x_j$ is found in the stack, we are finished. So the cycle length is j-i. Else, add (x_j, j) to the top of the stack (push) and continue.

This algorithm run in linear time, use logarithmic space and halt in the smallest value of the sequence cycle.

3.2. Pollard's Rho Algorithm Modified

Our idea is to use stack to store elements generated by the iteration function, the main algorithm is as follow: We keep in the stack pairs (i, a_i, b_i, R_i), the Ri in the stack forms increasing sequence, so we define lexicography order on E($\mathbb{F}_p$). If Rj>R_i where 0<i<j, we push the pairs (j, a_j, b_j, R_j) in the stack, otherwise if $R_i > R_j$ we pop all pairs (i, a_i, b_i, R_i) until we find $R_i = R_j$ or $R_i < R_j$. In the first case, we halt the process and compute l. In the second case, we push (j, a_j, b_j, R_j) in the stack and we generate other points. The details of the procedure are as follow:

- E($\mathbb{F}_p$) is into three disjoint sets S_1, S_2 and S_3
- S_1, S_2 and S_3 contain points with y-coordinate value between [0, p/3[, [p/3, 2p/3[or [2p/3, p[successively
- Let $R_0 = a_0P + b_0Q$ with a_0 and b_0 two random integers ∈ [1, n-1] and we define the iterative function f

$$f(R_i) = R_{i+1} = \begin{cases} P + R_i \text{ if } R_i \in S_1 \\ 2R_i \text{ if } R_i \in S_2 \\ Q + R_i \text{ if } R_i \in S_3 \end{cases}$$

The sequence a_i and b_i can be computed as follow:

$$(a_{i+1}, b_{i+1}) = \begin{cases} (a_i + 1, b_i) \text{if } R_i \in S_1 \\ (2a_i, 2b_i) \text{if } R_i \in S_2 \\ (a_i, b_i + 1) \text{if } R_i \in S_3 \end{cases}$$

If R_j is less than R_i we pop from the stack all R_i where $R_j < R_i$ else we push R_j on the top of the stack and continue.

We halt the process when $R_i = R_j$ then we find the fixed point, the logarithm discrete is resolved and $l = \frac{a_j - a_i}{b_i - b_j} (\text{mod } n)$.

The pseudo-code is as follow:

Algorithm 4. Pollard's Rho Algorithm with stack

Require: P, Q, S_1, S_2, S_3
Ensure: Integer l such that Q = lP
1: a_0←random ∈]0; n[
2: b_0← random ∈]0; n[
3: j← 1
4: i ←0
5: R_0←a_0P + b_0Q
6: (R_j, a_j, b_j)←(f(R_0), f(a_0, b_0))
7: stack←push (i, a_i, b_i, R_i)
8: for all j such that $R_j \neq R_i$ do
9: if $R_j < R_i$ then
10: repeat
11: stack.pop()
12: (i; ai; bi;Ri) ←stack:top()
13: until $R_j \geq R_i$
14: end if
15: if $R_j > R_i$ then
16: stack←push(j, a_j, b_j, R_j)
17: (R_{j+1}, a_{j+1}, b_{j+1})←(f(R_j), f(a_j, b_j))
18: else
19: $l \leftarrow \frac{a_j - a_i}{b_i - b_j} \text{mod}(n)$
20: break
21: end if
22: j ← j + 1
23: end for
24: Return l

The stack size must not exceeded the memory allowed for this attack. In this case, this attack will be discarded and unused. For this reason, we analyse stack size at time $n<\mu+\lambda$.

Theorem 2

Nivasch (2004) Given a positive integer n, let Sn be the stack size at time n and Mn the maximum stack size up to time n. Then:

- S_n has expectation ln n + O(1)
- M_n is almost surely$<\delta$ ln n for any constant $\delta>e$

At each iteration, Nivash's cycle finding algorithm stores only one element X_i.

However, in our case, we store (i, a_i, b_i, R_i) which means that the memory required is multiplied by 5. Thus the stack size is approximately 6 ln n+O(1).

3.3. Random Walks

Random walks developed by Teske are known to do much better than the original iteration function at the cost of little more computation. Hence, we use a mixture of random walks and stack in the Pollard's Rho algorithm. The steps in Pollard's Rho modified is described as follows:

- We partitioned $E(\mathbb{F}_p)$ into 20 disjoint sets S_1, S_2,.. S_{20}
- S_1, S_2,…,S_{20} contain points with x-coordinate mod 20 equals to 0, 1, 2, …19 successively
- We generete 20 random couples (a_i, b_i) and we compute the points $M_i = a_i P + b_i Q$
- Let $R_0 = a_0P+b_0Q$ with a_0 and b_0 two random integers in [1, n-1] and we define the iterative function f:

$$f(R_i) = R_{i+1} = \{M_j + R_i \text{ if } R_i \in S_j\}$$

The sequence a_i and b_i can be computed as follow:

$$(a_i, b_i) = \{(a + a_j, b + b_j)\} \text{ if } R_i \in S_j$$

- If R_{i+1} is less than R_i we pop from the stack all R_i where R_{i+1} is the least else we push R_{i+1} on the top of the stack and continue
- We halt process when $R_i = R_k$ then we find the fixed point, the logarithm discrete is resolved and $l = \frac{a_k - a_i}{b_i - b_k} (\text{mod } n)$

3.4. Implementation

The Pollard's Rho has been tested using stack on modern PC Core Duo with the Software Software Algebra Geometry Experimentation (SAGE) (Stein, 2010). First, we produce data file containing p prime numbers and for each prime number we build a secure elliptic curve and choose arbitrary point P and Q, we add the X and Y coordinate of P and Q to the file. Second, we use our approach to compute the discrete logarithm of Q to the base P. We follow these steps to implement the algorithm:

- Generate p prime numbers with size between six and fourteen digits (ten generation for each size)
- Generate random numbers A and B such that $(4A^3 + 27B^2) \text{mod}(p)$ different from 0
- Use p prime numbers, A and B numbers to generate elliptic curve
- Choose random X coordinate for Points P and Q from $E(\mathbb{F}_p)$ and calculate Y coordinate using Weierstrass Equation
- Compute integer l such that Q = lP using the method described above

4. COMPARISON BETWEEN ALGORITHMS

Analysis algorithm is quite important in computer programming because there are usually several algorithms available for a particular application and we would like to know which is the best.

4.1. Analysis

The most expensive steps in the Pollard's Rho method is the evaluation of the iteration function, thus it's quite important to compute the number of evaluation of this last to analyze the performance of the modified method.

The number of iterations in Pollard's Rho modified is at most $\mu+2\lambda$ that is roughly $3\sqrt{\pi n/8}$. However, with the original Pollard's Rho is $\sqrt{\pi n/2}$ but at each iteration, f has been evaluated three times instead of one time with the Pollard's Rho modified. Therefore, the number of evaluations of f before the algorithm terminates is $3\sqrt{\pi n/2}$ with the original Pollard's Rho and $3\sqrt{\pi n/8}$ with the modified Pollard's Rho. We conclude that the running time will be the greatest in the Pollard's Rho original. However, the amount of

memory is increased because with the Pollard's Rho original we store only pair (Y_i, Y_{2i}) and in each step, we generate the new pairs and we discard the previous pairs, so the memory used is negligeable. However, with Pollard's Rho modified we store all generated R_i that form an increasing sequence and we delete the element from the stack only if the current R_i is less than this element.

4.2. Experiment Results

In order to validate what we claim, we made severel experiments on random elliptic curves by computing the running time and the number of evaluations of iteration function for each of them. In **Table 1**, we found that if the length of p is less than fifteen digits, the running time of Pollard's Rho modified is less than the original Pollard's Rho. In **Table 2**, we will compare the number of evaluations of the iteration function of two methods, it is the lowest in Pollard's Rho with stack. As a consequence, the Pollard Rho's modified method performs better than the original Pollard's Rho method.

In **Table 3**, we will compare the running time of Pollard's Rho with random walks and Pollard's Rho mixing stack and random walks. Our experiment used prime numbers with size between six and eleven digits and the experimental results prove that the first method is lower than the second. We found that mixed walks using stack perform better than Pollard's Rho only with random walks.

Table 1. Running time comparison of new Pollard's RHO and Original Pollard's RHO

Digit No. (p)	Pollard's Rho with stack	Pollard's Rho with Floyd's
6	0.13816063	0.09353127
7	0.52002090	0.38864080
8	1.55706340	1.10163240
9	5.01763730	3.78752410
10	12.94963120	9.72552160
11	47.00025480	36.28078460
12	173.74268710	120.13733630
13	763.50992900	458.66987140
14	1994.19594744	1499.03777833

Table 2. Number evaluation of function iteration comparison of Pollard's RHO modified and Pollard's RHO original

Digit No. (p)	Pollard's Rho with stack	Pollard's Rho with Floyd's
6	1147	550
7	4554	2310
8	13376	6208
9	38560	21502
10	147098	77176
11	456563	214430
12	796299	416246
13	2717168	1477922
14	12819430	6433273

Table 3. Running time of Pollard's Rho with random walks and mixing stack and random walks

Digit No. (p)	Pollard with Random	Random Walks Walks stack
6	3.9533380	1.3746310
7	17.4098880	11.2679040
8	44.5107820	30.9887360
9	210.4911547	108.6511900
10	638.0878780	447.9731964
11	3834.4660393	1618.8607724

5 CONCLUSION

In this study we have outlined a new algorithm to speed-up Pollard's Rho attack by make use of first Nivasch's cycle detection and second mixing Nivasch's cycle detection and random walks. However, this variant is not appropriate if the memory used is limited. For further research, we intend to improve Pollard's Rho method using at the same time cycle detection, random walks and Parallelization.

6. REFERENCES

Brent, R.P., 1980. An improved monte carlo factorization algorithm. BIT Numerical Math., 20: 176-184. DOI: 10.1007/BF01933190

Cheon, J.H., J. Hong and M. Kim, 2012. Accelerating Pollard's Rho algorithm on finite fields. J. Cryptol., 25: 195-242. DOI: 10.1007/s00145-010-9093-7

Coppersmith, D., 1984. Fast evaluation of logarithms in fields of characteristic two. IEEE Trans. Inform. Theory, 30: 587-594. DOI: 10.1109/TIT.1984.1056941

Duursma, I.M., P. Gaudry and F. Morain, 1990. Speeding up the discrete log computation on curves with automorphisms. Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Nov. 14-18, Springer Berlin Heidelberg, Singapore, pp: 103-121. DOI: 10.1007/978-3-540-48000-6_10

Floyd, R.W., 1962. Algorithm 97: Shortest path. Commun. ACM, 5: 345-345. DOI: 10.1145/367766.368168

Hankerson, D., S. Vanstone and A.J. Menezes, 2004. Guide to Elliptic Curve Cryptography. 1st Edn., Springer-Verlag, New York, ISBN-10: 038795273X, pp: 311.

Harrison, A., 2010. Square-Root Methods for the Discrete Logarithm Problem. African Institute for Mathematical Sciences.

Knuth, D.E., 1969. The art of Computer Programming. 2nd Edn., Addison-Wesley Pub. Co., Reading, pp: 634.

Nivasch, G., 2004. Cycle detection using a stack. Inform. Process. Lett., 90: 135-140. DOI: 10.1016/j.ipl.2004.01.016

Oorschot, P.C.V. and M.J. Wiener, 1999. Parallel collision search with cryptanalytic applications. J. Cryptol., 12: 1-28. DOI: 10.1007/PL00003816

Pollard, J.M., 1978. Monte carlo methods for index computation $({\rm mod}\p)$. Math. Comput., 32: 918-924. DOI: 10.1090/S0025-5718-1978-0491431-9

Shanks, D., 1971. Class Number, a Theory of Factorisation and Genera. In: Symposia in Pure Math-Ematics, Donald, J. lewis, (Eds.), Lecture Notes in Computer Science, American Mathematical Society (AMS), pp: 415-440.

Stein, W., 2010. SAGE mathematical software. Version 4.6.

Teske, E., 1998. Speeding up Pollards rho Method for Computing Discrete Logarithms. In: ANTS 1998. LNCS, Buhler, J.P., (Eds.), Springer, Heidelberg, pp: 541-554.

Teske, E., 2001. On random walks for Pollard's RHO method. Mathem. Comput., 70: 809-825.