Implementation of Pollard Rho over binary fields using Brent Cycle Detection Algorithm

Intan Muchtadi-Alamsyah, Taufiq Akbari Utomo
Faculty of Mathematics and Natural Sciences, Institut Teknologi Bandung, Jalan Ganesha No. 10 Bandung 40132, Indonesia
E-mail: ntan@math.itb.ac.id

Abstract. The security of Elliptic Curve Cryptography depends on how to solve the Elliptic Curve Cryptography Discrete Logarithm Problem (ECDLP). In this paper we propose the use of modified Pollard Rho Algorithm by using Brent Cycle Detection Algorithm to solve the ECDLP. We give performance comparison on time and the number of iterations between Pollard Rho with Brent Cycle Detection and Pollard Rho with Negation map. In particular, for Koblitz curve, we also give comparison between Pollard Rho with Brent Cycle Detection and Pollard Rho with Negation and Frobenius maps.

1. Introduction
Some public key cryptosystems that is widely used today include RSA and the elliptic curve cryptosystem. Elliptic curve cryptosystem can provide the same level of security with RSA with smaller key sizes. As a result, the elliptic curve cryptosystem consumes less energy, memory, bandwidth, and time to perform encryption and decryption than RSA cryptosystem.

The best way to test the security of a cryptosystem is to position themselves as the attacker. The security of elliptic curve cryptosystem relies on the solution of the elliptic curve discrete logarithm problem. One of the algorithm used to resolve such problems is the Pollard Rho Algorithm.

In previous research we have implemented the Pollard Rho algorithm using the Frobenius and Negation maps [5] and also Basis Conversion [4]. In this research we explore the use of Brent Cycle Detection Algorithm to detect collisions in Pollard Rho Algorithm. We give comparison on time and the number of iterations between Pollard Rho with Brent Cycle Detection and Pollard Rho with Negation map. In particular, for Koblitz curve, we also give comparison between Pollard Rho with Brent Cycle Detection and Pollard Rho with Negation and Frobenius maps.

2. Basic Definition
Let $F$ be binary field. An elliptic curve is an equation of the form

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

plus certain conditions with $a_1, a_2, a_3, a_4, a_6 \in F$. For binary field $GF(2^n)$ the above equation can be simplified to

$$y^2 + xy = x^3 + ax^2 + b,$$
with \(a, b \in GF(2^m)\).

If \(E\) is an elliptic curve over \(\mathbb{R}\) then the addition operation of two points can be illustrated as follows.

Let \(P = (x_1, y_1), Q = (x_2, y_2) \in E\), and \(R = P + Q\) with \(P \neq Q\). To obtain \(R\), draw a line through \(P\) and \(Q\). This line will intersect the curve at three points, two of them are \(P\) and \(Q\). Let \(R'\) be the third intersection point. Then \(R\) is the reflection of \(R'\) through the \(x\)-axis. Note that if \(Q\) is the reflection of \(P\) through the \(x\)-axis, the line formed with intersect the curve at \(P, Q\) and another point called the point at infinity, which we denote by \(O\). Reflecting \(O\) through the \(x\)-axis gives \(O\) itself. So \(P + Q = O\) when \(Q\) is the reflection of \(P\) through the \(x\)-axis.

Now let \(S = P + P\). To obtain \(S\), draw the tangent line of \(E\) at \(P\). This line will intersect the curve at two points, one of them is \(P\). Let \(R'\) be the second intersection point. Then \(S\) is the reflection of \(S'\) through the \(x\)-axis.

For \(E\) elliptic curve over \(GF(2^m)\), the addition rule is as follows.

(i) For all \(P \in E\), \(P + O = O + P = P\), with \(O\) as the point at infinity.

(ii) If \(P = (x, y)\), then the negative of \(P\), denotes by \(-P\) is \((x, x + y)\).

(iii) If \(P = (x_1, y_1)\) and \(Q = (x_2, y_2)\) with \(Q \neq \pm P\), then \(P + Q = (x_3, y_3)\), where 
\[
x_3 = \lambda^2 + \lambda + x_1 + x_2 + a, \quad y_3 = \lambda(x_1 + x_3) + x_3 + y_1, \quad \text{and} \quad \lambda = \frac{y_1 + y_2}{x_1 + x_2}.
\]

(iv) If \(P = (x_1, y_1)\) then \(P + P = (x_4, y_4)\), where \(x_4 = \mu^2 + \mu + a = x_1^2 + \frac{b}{4a}, \quad y_4 = x_4^2 + \mu x_4 + x_4, \quad \text{and} \quad \mu = x_1 + \frac{y_1}{x_1}.
\]

The points in \(E\) together with \(O\) form an abelian group with the above addition operation [2].

3. Pollard Rho Algorithm

Let \(P \in E\) and \(k\) be a positive integer. We define \(kP = P + P + \cdots + P\). Moreover if \(k = 0\) then \(kP = O\), and if \(k\) is a negative integer then \(kP = (-P) + (-P) + \cdots + (-P)\). We define the order of \(P\) as the smallest positive integer \(m\) such that \(mP = O\).

Sending a message from Alice to Bob using Elliptic Curve Cryptography is done in the following way:

(i) Alice and Bob agree on the elliptic curve \(E\) and a point \(P\) of order \(m\).

(ii) Bob chooses a positive integer \(k \in \{1, 2, \cdots, m - 1\}\), which is confidential. After that, Bob computes \(Q = kP\) and announces \(Q\).

(iii) Let \(u\) be the message that Alice will send. Alice converts \(u\) into a point \(M\) on the curve.

(iv) Alice chooses a positive integer \(j \in \{1, 2, \cdots, m - 1\}\), which is confidential. After that, Alice computes \(C_1 = jP\) and \(C_2 = M + jQ\).

(v) Alice sends \((C_1, C_2)\) to Bob.

(vi) Bob can get \(M\) from \((C_1, C_2)\) by computing \(M = C_2 - jQ = C_2 - j(kP) = C_2 - k(jP) = C_2 - kC_1\).

(vii) Bob converts \(M\) back into the message \(u\).

Eve as a third party only knows \(E, P, Q, n, C_1\) and \(C_2\). If she wants to know \(M\), then she must know \(j\) or \(k\). This is the discrete logarithm problem.
More formally, given an elliptic curve $E$ and a point $P \in E$ of order $m$, the Elliptic Curve Discrete Logarithm Problem (ECDLP) is to find $k \in \{1, \ldots, m - 1\}$, if $Q = kP$ is known. The Pollard Rho Algorithm is one of the algorithms for solving the discrete logarithm problem. The main idea of this algorithm is to generate a sequence $\{X_i\}_{i=1}^{\infty}$, with $X_i \in E$, until the collision occurred, i.e. when $X_i = X_j$ for some $i \neq j$. The computation of each term in the sequence is deterministic so that we can obtain information on $X_i$ such that ECDLP can be solved. However, the sequence $\{X_i\}_{i=0}^{\infty}$ is generated such that the sequence behaves like a random sequence. Thus, based on the birthday paradox, expectations of the number of terms prior to the collision is $\sqrt{\frac{\pi m}{2}}$, with $m$ is the order of $P$ [3].

Let $E$ be an elliptic curve over $GF(2^n)$, $P \in E$ of order $m$, and $Q = kP$, with $k$ is unknown. The Pollard Rho Algorithm can be used to find $k$ and is given as follows:

(i) Divide $E$ into three sets with almost equal cardinalities, namely $S_1, S_2,$ and $S_3$.
(ii) Set $X_0 = P$, and define the iteration function $f$:

$$X_{i+1} = f(X_i) = \begin{cases} X_i + P, & X_i \in S_1 \\ 2X_i, & X_i \in S_2 \\ X_i + Q, & X_i \in S_3 \end{cases}$$

(iii) Write $X_i = s_iP + t_iQ$ for all $i$.
(iv) We get $s_0 = 1, t_0 = 0$ and

$$s_{i+1} = \begin{cases} s_i + 1, & X_i \in S_1 \\ 2s_i, & X_i \in S_2 \\ s_i, & X_i \in S_3 \end{cases}$$

$$t_{i+1} = \begin{cases} t_i, & X_i \in S_1 \\ 2t_i, & X_i \in S_2 \\ t_i + 1, & X_i \in S_3. \end{cases}$$

(v) Continue the iterations until we obtain $j, l$ with $j \neq l$ but $X_j = X_l$.
(vi) For $j$ and $l$ as in 5, we get

$$s_jP + t_jQ = s_lP + t_lQ.$$ 

Since $Q = kP$ and the order of $P$ is $m$, we get $s_j + t_jk \equiv s_l + t_lk \mod m$. If $gcd(t_j - t_l, m) = 1$, then

$$k = \frac{s_l - s_j}{t_j - t_l} \mod m.$$ 

Notice that if $gcd(t_j - t_l, m) \neq 1$, then $t_j - t_l$ has no inverse over multiplication modulo $m$. However in this research we choose $P$ such that $m$ is a prime number, so that $k$ can always be found except if $t_j = t_l$.

4. Modified Pollard Rho

In Pollard Rho Algorithm, the detection is done by checking whether the point generated on the latest iteration is the same as one of the points generated in the previous iteration. In other words, we run iterations until we obtain $j, l$ with $j \neq l$ but $X_j = X_l$. After that, we obtain $X_j = s_jP + t_jQ$ and $X_l = s_lP + t_lP$. Since $Q = kP$ and the order of $P$ is $m$, we get

$$s_j + t_jk \equiv s_l + t_lk \mod m.$$
Then we solve that equation to get $k$.

If this is done, then we need to store every $X_i$ generated through iterations. In addition, we also need to store $s_i$ and $t_i$ on each iteration. Thus, although Pollard Rho Algorithm can solve ECDLP within $O(\sqrt{m})$ time, we need to store $O(\sqrt{m})$ points.

However, storing one point $X_j$ requires minimal $4n$ bits, i.e. for each $n$ bit we store the abscissa of $X_j$, the ordinate of $X_j$, $s_j$ and $t_j$. For a large $n$, for example $n > 60$, keeping all points will require big memory. To decrease memory usage, cycle detection can be done by using the Brent Cycle Detection Algorithm. In detecting the iteration cycle performed in Pollard Rho Algorithm, the Brent Cycle Detection Algorithm goes as follows:

(i) Suppose the sequence that we want to generate is $\{X_i\}_{i=0}^{\infty}$ with initial value $X_0$ and iteration function $f$ such that $X_{i+1} = f(X_i)$. Set $j = 0$, $k = 0$, and $l = 1$.

(ii) Replace $k$ with $k + 1$, then check whether $X_j = X_k$. If $X_j = X_k$, then we get a collision.

(iii) If the collision has not occurred, repeat step 2 until a collision occurs or $k = 2l - 1$.

(iv) If $k = 2l - 1$, replace $j$ with $k$ and replace $l$ with $2l$.

(v) Repeat steps 2-4 until a collision occurred.

The expectation of the number of iterations performed is $1.9828\sqrt{m}$, where $m$ is the order of $P$ [1]. It is more than $\sqrt{\frac{\pi m}{2}} \approx 1.2533\sqrt{m}$, i.e. the expectation of the number of iterations in Pollard Rho Algorithm unmodified. However, we only need to store two points, instead of $O(\sqrt{m})$ points.

5. Experimental Results
In previous research we gave a comparison between standard Pollard Rho and Pollard Rho Algorithm with Negation and Frobenius maps for Koblitz curves (see [4] [5]).

In this section we will give a comparison between three variants Pollard Rho Algorithm on elliptic curves over binary field: the standard Pollard Rho Algorithm, Pollard Rho Algorithm with Negation map, and Pollard Rho using Brent Cycle Detection Algorithm. In addition, the possibility of using Frobenius on Koblitz curves gives more comparison between variants of Pollard Rho Algorithm.

First, the following is the list of curves used in this research.

| $n$ | $a$ | $b$ | $P_x$ | $P_y$ | $m$ | $k$ |
|-----|-----|-----|-------|-------|-----|-----|
| 7   | 1   | 1   | 125   | 17    | 71  | 10  |
| 11  | 1   | 1   | 317   | 1892  | 591 | 62  |
| 13  | 0   | 1   | 3164  | 5281  | 2003| 612 |
| 17  | 1   | 1   | 18909 | 120804| 65587| 13690|
| 17  | 90907| 8300 | 119691| 66462 | 65490| 43810|
| 19  | 1   | 1   | 82773 | 3169 | 262543| 224885|
| 19  | 34563| 310103| 253684| 3177725| 261651| 74551|
| 23  | 1   | 1   | 718599| 1674296| 4196603| 852098|
| 23  | 345761| 8316424| 1605957| 6156578| 4156353| 448137|
| 29  | 516948084| 310755777| 253272920| 62478750| 129419847| 814995421|
| 31  | 845199535| 167506850| 15805131681| 910355105| 1073784797| 972658180|
| 37  | 94834870882| 7148293082| 123023828246| 21593014607| 68719385843| 207711372|
| 41  | 0   | 1   | 1135561933198| 312645604389| 549758909843| 439866657162|
| 41  | 312605682591| 483991066662| 1585161278299| 85987920702| 16099514048257| 875103879016|
In Table 2 we give comparison between standard Pollard Rho and Pollard Rho with Brent Cycle Detection Algorithm. In Table 3 we give the same comparison for Koblitz curves without Frobenius map, meanwhile the same comparison for Koblitz curves with Frobenius map is given in Table 4.

**Table 2** Comparison between standard Pollard Rho and Pollard Rho with Brent Cycle Detection Algorithm

| Bit | Standard Iteration | Standard Time (second) | Brent Iteration | Brent Time (second) |
|-----|--------------------|------------------------|-----------------|---------------------|
| 17  | 352                | 0.156248               | 473             | 0.203124            |
| 19  | 895                | 0.453124               | 1571            | 0.795614            |
| 23  | 6098               | 3.765658               | 12555           | 7.656267            |
| 29  | 15689              | 12.687533              | 30471           | 23.515687           |
| 31  | 58678              | 51.593854              | 99508           | 83.578335           |
| 37  | 251950             | 290.188130             | 351431          | 364.672700          |
| 41  | 2310322            | 4993.755471            | 3869603         | 4943.581061         |

**Table 3** Comparison between std Pollard Rho and Pollard Rho with Brent Alg (Koblitz)

| Bit | Standard Iteration | Standard Time (second) | Brent Iteration | Brent Time (second) |
|-----|--------------------|------------------------|-----------------|---------------------|
| 7   | 16                 | 0.015624               | 30              | 0.015626            |
| 11  | 43                 | 0.015637               | 51              | 0.015624            |
| 13  | 130                | 0.062496               | 158             | 0.062498            |
| 17  | 89                 | 0.046876               | 97              | 0.046874            |
| 19  | 1819               | 0.966245               | 3479            | 1.703126            |
| 23  | 398                | 0.249999               | 636             | 0.390642            |
| 41  | 1125258            | 2098.741221            | 3165644         | 4122.895303         |

**Table 4** Comparison between standard Pollard Rho and Pollard Rho with Brent Cycle Detection Algorithm for Koblitz curves, both use Frobenius map

| Bit | Standard Iteration | Standard Time (second) | Brent Iteration | Brent Time (second) |
|-----|--------------------|------------------------|-----------------|---------------------|
| 7   | 3                  | 0.015641               | 6               | 0.015625            |
| 11  | 21                 | 0.046877               | 25              | 0.031263            |
| 13  | 12                 | 0.031254               | 24              | 0.031230            |
| 17  | 131                | 0.218761               | 162             | 0.234359            |
| 19  | 123                | 0.256540               | 207             | 0.375001            |
| 23  | 861                | 2.125028               | 1065            | 2.359400            |
| 41  | 87869              | 44.544495              | 90320           | 538.360148          |

From the experiment, we see that the use of Frobenius map generally reduces the number of iterations required. However, the time required is not always shorter. This is due to the additional time required to generate equivalence class of each iteration.

In Table 5 we give comparison between Pollard Rho with Frobenius map and Pollard Rho without Frobenius map for Koblitz curves, both use Brent Cycle Detection Algorithm. In Table 6 we give the same comparison but without Brent Cycle Detection Algorithm and without Negation map. Meanwhile the same comparison without Brent Cycle Detection Algorithm but with Negation map is given in Table 7.
Table 5 Comparison between Pollard Rho with Frobenius map and Pollard Rho without Frobenius map for Koblitz curves, both use Brent Cycle Detection Algorithm

| Bit | With Frobenius | Without Frobenius |
|-----|----------------|-------------------|
|     | Iteration | Time (second) | Iteration | Time (second) |
| 7   | 30       | 0.015626 | 6         | 0.015625 |
| 11  | 51       | 0.015624 | 25        | 0.031253 |
| 13  | 168      | 0.062498 | 24        | 0.031230 |
| 17  | 97       | 0.046874 | 162       | 0.234359 |
| 19  | 3479     | 1.703126 | 207       | 0.375001 |
| 23  | 636      | 0.390942 | 1065      | 2.359400 |
| 41  | 3165044 | 4122.895303 | 90326 | 558.300148 |

Table 6 Comparison between Pollard Rho with Frobenius map and Pollard Rho without Frobenius map for Koblitz curves, both without Brent Cycle Detection Algorithm and without Negation map

| Bit | With Frobenius | Without Frobenius |
|-----|----------------|-------------------|
|     | Iteration | Time (second) | Iteration | Time (second) |
| 7   | 16       | 0.015624 | 3         | 0.015641 |
| 11  | 43       | 0.015637 | 21        | 0.046877 |
| 13  | 130      | 0.062496 | 12        | 0.031254 |
| 17  | 89       | 0.046876 | 131       | 0.218761 |
| 19  | 1819     | 0.906245 | 123       | 0.265640 |
| 23  | 398      | 0.249999 | 861       | 2.125028 |
| 41  | 1125258 | 2098.714221 | 87869 | 648.544495 |

Table 7 Comparison between Pollard Rho with Frobenius map and Pollard Rho without Frobenius map for Koblitz curves, both without Brent Cycle Detection Algorithm but with Negation map

| Bit | With Frobenius | Without Frobenius |
|-----|----------------|-------------------|
|     | Iteration | Time (second) | Iteration | Time (second) |
| 7   | 11       | 0.015616 | 2         | 0.031252 |
| 11  | 25       | 0.015626 | 13        | 0.046874 |
| 13  | 113      | 0.062489 | 17        | 0.046894 |
| 17  | 411      | 0.218751 | 63        | 0.171890 |
| 19  | 495      | 0.296888 | 102       | 0.328139 |
| 23  | 3959     | 2.906526 | 696       | 2.609399 |
| 41  | 1142980 | 2449.223652 | 66879 | 666.827368 |

From the experiment, we see that the use of Negation map generally reduce the number of iterations required. However, if Negation map is used without Frobenius map, almost 10 percent (276,344 out of 2,773,726) iterations are repeated due to fruitless cycles. As a result, the acceleration factor of $\sqrt{2}$ which was originally predicted is not achieved.

6. Conclusion and Further Research
The use of Brent Cycle Detection Algorithm to detect collisions in Pollard Rho Algorithm needs more iterations and generally takes longer than storing every point and check it out.
Nevertheless, we can not store all points for Pollard Rho Algorithm with large binary field, for example \( GF(2^n) \) with \( n > 60 \).

The use of Frobenius map generally reduces the number of iterations required. However, the time required is not always shorter. This is due to the additional time required to generate equivalence class of each iteration. The use of Negation map generally reduce the number of iterations required. However, if Negation map is used without Frobenius map, almost 10 percent (276,344 out of 2,773,726) iterations are repeated due to fruitless cycles. As a result, the acceleration factor of \( \sqrt{2} \) which was originally predicted is not achieved.

For further research, one can investigate Nivash Cycle Detection algorithm. Theoretically, Nivash Cycle Detection algorithm need \( \frac{5}{2} \frac{\sqrt{\pi m}}{8} \) iterations, less than Brent Cycle Detection Algorithm in detecting collisions. However, expectations for the number of points that need to be stored is \( \ln h + O(1) \), with \( h \) is the number of iterations [1].

Acknowledgment
This research is supported by Asahi Glass Foundation research grant 2014.

Reference
[1] Cohen H and Frey G 2006 Handbook of Elliptic and Hyperelliptic Curve Cryptography (Chapman and Hall/CRC)
[2] Fulton, W 1969 Algebraic Curves: An Introduction to Algebraic Geometry (W. A. Benjamin)
[3] Hankerson D, Menezes A, and Vanstone S 2004 Guide to Elliptic Curve Cryptography (Springer)
[4] Muchtadi-Alamsyah I, Ardiansyah T, Carita S S 2014 Adv. Sc. Let. 20(1) 340-343.
[5] Muchtadi-Alamsyah I, Ardiansyah T, Carita S S 2013 Far East J. Math Sc. Special Volume, No 4 385-402.