Journal of Theoretical and Applied Information Technology
31st December 2018. Vol.96. No 24
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

PERFORMANCE COMPARISON BETWEEN BABY-STEP GIANT-STEP METHOD AND POLLARD RHO WITH BRENT CYCLE DETECTION METHOD IN CAUSING TOTAL BREAK TOWARD DIGITAL SIGNATURE ALGORITHM SCHEME

[1]RULLY SOELAIMAN, [2]MUHAMMAD GHAZIAN

[1]Lecturer, Department of Informatics, Faculty of Information Technology and Communication, Institute of Technology Sepuluh Nopember, Surabaya, Indonesia

[2]Student, Department of Informatics, Faculty of Information Technology and Communication, Institute of Technology Sepuluh Nopember, Surabaya, Indonesia

E-mail: [1]rully@if.its.ac.id, [2]muhghazian@gmail.com

ABSTRACT

This paper compares the speed of Baby-step Giant-step method and Pollard Rho with Brent Cycle Detection method to cause *Total Break*, which in turn is used to attack Digital Signature Algorithm i.e. forging a signature. This paper also finds out how the free parameter of Digital Signature Algorithm can affect the performance of the two method. This paper will present an empirical result on how the two method would perform under signature forgery scenario. Furthermore, an analysis of how close both method's performance is presented. Also, this paper provides an analysis regarding how the size of free parameter of DSA may affect the overall performance of the two methods. These analysis can provide useful basis for DSA's parameter security analysis. Moreover, the result presented in this paper can extend to other discrete logarithm problem.

Keywords: *Cryptography, DSA, Discrete Logarithm, Number Theory, Security*

1. INTRODUCTION

Digital Signature is a protocol mainly used as means to protect from source repudiation. In other words, it protects from a sender denying that he is the one who send a certain message. Digital Signature plays an important role in many practical aspect. Take for example, banking.

There has been several digital signature algorithm, such as ElGamal Digital Signature and Schnorr Digital Signature. One particular digital signature algorithm is published by NIST as a standard with the name Digital Signature Algorithm. This algorithm uses asymmetric key to work. What makes this algorithm enticing is that this algorithm uses the hardness of discrete logarithm problem as its security measure. Digital Signature Algorithm (DSA) follows a certain parameter specification.

Goldwasser et al. has made a work to determine the secureness of digital signature scheme against adaptive-chosen message attacks [1]. In the same work, Goldwasser et al stated four kind of attacks against digital signature scheme—one of them is called *total break* where the user's private key is compromised. This paper is interested to dwelve deeper into this type of attack.

The idea of *total break* is to find out the private key from the cryptosystem. Various algorithm has been developed to solve discrete logarithm. There are two general-purpose algorithm that can be utilized, namely Baby-step Giant-step and Pollard Rho. These two algorithms have very broad application due to both can be used for any algebraic structure and not only for integer domain. Due to this fact, these algorithms are very good candidate as the method to cause *total break*.

It has been noted that there are plentiful of research that revolves around Pollard Rho method. Many are interested in improving the performance either via specific hardware utilization (for example, [2], [3], and [4]) or by finding more efficient approach (for example, [5]). Combined with ever-developing hardware evolution, the improvement would be cumulative. With this fruitful development, it would be interesting to know where such improvement has taken us since it would definitely impact how security protocols, including DSA, might get compromised.

This paper is interested in finding out how free parameter in DSA can affect the performance in causing *total break*. As stated before, Baby-step Giant-step and Pollard Rho is a potential means to cause *total break*, so the two method is to be used as a control method. The two method is to be run with various values following the DSA's specification, and see how well both method can tackle the chosen parameters.

1.1. Contribution

The outcome of this research is two fold.

1.1.1. Empirical evidence of how DSA's parameter may affect two method's performance

This research will introduce the effect of various DSA parameter towards the time needed for Baby-step Giant-step and Pollard Rho method to cause *total break*. This research focuses on as many free DSA parameter as possible, following the standard protocol. While there are many works, especially student guidebook, which already explained a certain DSA parameter's effect to the two method's performance, this research is to give an empirical result of what effect they would produce. Hopefully, this can serve as a good basis for future work regarding the secureness of DSA.

1.1.2. Provide a comparison between two method on classical computer in term of causing *total break*

This paper is interested in discovering how Baby-step Giant-step and Pollard Rho would compare in term of their speed in causing *total break* when faced with Digital Signature Algorithm—one of the digital signature scheme. And as such, both method will be used to simulate signature forgery, the type of attack which Digital Signature Algorithm can fall victim into. The knowledge of how DSA can fall vulnerable to each method can serve as starting point for further development of DSA.

While this is the main objective, the finding resulted from this research can be extended to any context, as long they are related to discrete logarithm problem, ran on classical computer.

2. THEORETICAL REFERENCE

In this paper, two methods will be introduced.

1. Baby-step Giant-step.
2. Pollard Rho (using Brent Cycle Detection).

Along these two methods, a Digital Signature Algorithm explanation also will be presented.

2.1. Digital Signature Algorithm

Digital Signature Algorithm is a standard digital signature method issued by NIST as the first version of Digital Signature Standard [6]. Digital Signature Algorithm works as follow. Given a hashed message $H(M)$:

1. Determine the global public key component.
   a) A prime number p where $2^{L-1} \leq p \leq 2^L$ for $512 < L < 1024$ and L is a multiple of 64.
   b) A prime number where $q|(\mathrm{p}-1)$ and $2^{159} < q < 2^{160}$.
   c) A generator g that can be obtained using $g = h^{(p-1)/q} \ mod\ p$. h can be set arbitrarily where $1 < h < (p-1)$
2. Determine the keypair
   a) An integer x determined randomly with $0 < x < q$
   b) An integer y where $y = g^x \ mod\ p$
3. Signature generation follows this procedure.
   a) $r = (g^k mod\ p)\ mod\ q$
   b) $s = [k^{-1}(H(M) + xr)]\ mod\ q$
   c) Signature is (r, s)
4. Signature verification follows this procedure.
   a) $w = s^{-1}\ mod\ q$
   b) $u_1 = [H(M)w]\ mod\ q$
   c) $u_2 = rw\ mod\ q$
   d) $v = [g^{u_1} y^{u_2}\ mod\ p]\ mod\ q$
   e) Signature is valid if $v = r$

2.2. Baby-step Giant-step

Baby-step Giant-step is a method proposed by Shank to solve discrete logarithm problem [7]. This method is known to have time complexity of $O(\sqrt{n})$ by having time-space tradeoff with space complexity of $O(\sqrt{n})$. This method works as shown in algorithm 1

Algorithm 1 Baby-step Giant-step

```
Input: Four integers: $g, y, o$ and $m$ where $o$ is the multiplicative order modulo $m$ of $g$
Output: An integer $e$ that satisfies $g^e \equiv y\ (mod\ m)$ or "Not Exist"
1  Let $\mathbb{T}$ be a set of tuples in form of $\langle index, value \rangle$
2  for $i \leftarrow 0$ to $o - 1$ do
3      $G \leftarrow g^{i\sqrt{o}} (mod\ m)$
4      $\mathbb{T} \leftarrow \mathbb{T} \cup \langle i\sqrt{o}, G \rangle$
5  end for
6  for $j \leftarrow 0$ to $o - 1$ do
7      $Y \leftarrow y * g^j\ (mod\ m)$
8      Find a tuple $\boldsymbol{t} \in \mathbb{T}$ where $\boldsymbol{t}.value = Y$
9      if such t exist then
10         return $(\boldsymbol{t}.index - j)\ mod\ m$
11     end if
12 end for
13 return "Not Exist"
```

Implementation of this algorithm may require effective searching method to do line 8. In fact, poor choice searching algorithm may worsen the time complexity of Baby-step Giant-step. One well known good method is to use binary search, a $O(\sqrt{n})$ speed searching method, with some overhead cost to sort the table resulted from step 2–5. When this algorithm is used, the search is to be done on the index.

Another way to do the searching is using hash table, with $O(1)$ for searching. The hash table uses the value as the hash key, and the index as the hash value. Algorithm 2 illustrates how it is done. This version of Baby-step Giant-step is used during testing.

Algorithm 2 Baby-step Giant-step

Input: Four integers: g, y, o and m where o is the multiplicative order modulo m of g

Output: An integer e that satisfies $g^e \equiv y \ (mod\ m)$ or **"Not Exist"**

```
1  Let $\mathbb{H}$ be a hash table
2  for $i$ ← 0 to $o - 1$ do
3      $G \leftarrow g^{i\sqrt{o}} (mod\ m)$
4      Store $i\sqrt{o}$ to $\mathbb{H}$ with $G$ as its hash key
5  end for
6  for $j$ ← 0 to $o - 1$ do
7      $Y \leftarrow y * g^j \ (mod\ m)$
8      Check whether $\mathbb{H}$ has an entry whose its
       key is Y
9      if such t exist then
10         return $(\mathbb{H}[Y] - j)\ mod\ m$
11     end if
12 end for
13 return "Not Exist"
```

2.3. Pollard Rho

Pollard Rho method is originally designed to solve integer factorization problem [8]. A slight modification of Pollard Rho's element can be done so that this method can solve discrete logarithm. This method has expected time complexity of $O(\sqrt{n})$ [7].

2.3.1. Pollard Rho's Component

Pollard Rho consists of several element.

1. A random function $f(x)$. This function can be set arbitrarily with certain limitation so as to make sure the value produced by $f(x)$ seemed random [7].
2. Two *pointers*, p_1 and p_2. To avoid misconception, the term *pointer* used here is not the same as the term used in programming (i.e. a variable storing a memory address). When *pointer* is stated, it means a value $f(x)$ for some x.
3. A step function $step(x)$. This function determines how the *pointer* will behave for every iteration. This function is different than the random function $f(x)$. Each *pointer* will have one step function i.e. $step_{p_1}(x)$ and $step_{p_2}(x)$.

2.3.2. General Pollard Rho Algorithm

Algorithm 3 shows how Pollard Rho works in general.

Line 9 might seem undetailed. This is intentional due to how Pollard Rho can achieve various thing depending on how it is implemented. The detail of line 9 will follow later on. Also notice on line 2 that a value n is used. This value can be set to whatever value desired. It functions as the initial value for both *pointer*.

Algorithm 3 Pollard Rho – Generic

```
1  Let $p_1$ and $p_2$ be two different pointers
2  Let $n$ be an arbitrary value
3  $p_1 \leftarrow n$
4  $p_2 \leftarrow n$
5  repeat
6      $p_1 \leftarrow step_{p_1}(p_1)$
7      $p_2 \leftarrow step_{p_2}(p_2)$
8  until $p_1 = p_2$
9  Infer the desired information from $p_1$ and $p_2$
```

Algorithm 3 shows how step function and the *pointer* interacts, but not the random function. In fact, algorithm 3 does not state about the random function at all. The explanation of random function will follow because its detail relies on how Pollard Rho is implemented.

2.3.3. Random Function

Since the Pollard Rho method is going to be used to solve discrete logarithm problem, the random function used is as follow.

$$f(x) = \begin{cases} (\beta * x)\ mod\ m & x \in S_1 \\ (x * x)\ mod\ m & x \in S_2 \\ (\alpha * x)\ mod\ m & x \in S_3 \end{cases} \tag{1}$$

Consequently, $f(x)$ can be written as (2).

$$f(x) = \alpha^a * \beta^b\ mod\ m \text{ for arbitrary } a \text{ and } b \tag{2}$$

The value of α and β will be explained later on. Also as seen in (1), there are three sets stated: S_1, S_2, and S_3. These sets are the subsets of $\mathbb{Z}$ i.e. integers, and in practice, can be defined arbitrarily with limitation that the three sets must have roughly same cardinality, disjoint to each other, and $1 \notin S_2$ [7].

During implementation, the three sets are defined using the explanation in Handbook of Cryptography [7].

$$S_1 = \{x : x \equiv 1 \, mod \, 3\}$$
$$S_2 = \{x : x \equiv 0 \, mod \, 3\}$$
$$S_3 = \{x : x \equiv 2 \, mod \, 3\}$$

Therefore, (2) can be written into (3).

$$f(x) = \begin{cases} (\beta * x) \, mod \, m & x \equiv 1 \, mod \, 3 \\ (x * x) \, mod \, m & x \equiv 0 \, mod \, 3 \\ (\alpha * x) \, mod \, m & x \equiv 2 \, mod \, 3 \end{cases} \quad (3)$$

2.3.4. Brent's Cycle Detection

Pollard in his original paper uses Floyd's cycle detection as the cycle detection method [9], [8]. Later on 1980, Brent improves Pollard Rho's speed up to 24% by implementing different cycle detection [8]. This paper will utilize Brent's cycle detection. His method works as follow.

Algorithm 4 Baby-step Giant-step

```
1  Let $s$ and $t$ be two different pointers
2  Let $i$ be iteration counter
3  Let limit be current iteration limit
4  $i \leftarrow 0$
5  $limit \leftarrow q_0$
6  repeat
7      if $i = limit$ then
8          $t \leftarrow s$
9          $i \leftarrow 0$
10         $limit \leftarrow limit * q$
11     else
12         $i = i + 1$
13     end if
14     $s \leftarrow f(s)$
15 until $s = t$
```

Algorithm 4 speaks about how two *pointers* behave. Basically, it has a *pointer* (e.g. s) to do the step process for every iteration (line 14), and another *pointer* (e.g. t) to do the step process for roughly every q^x iteration with increasing x (line 7–13).

Notice the difference between the *pointers* on how they do the step process. At line 14, *pointer* s uses a random function to move, whereas at line 8 *pointer* t move by setting its value with *pointer* s. Additionally, each time t moves, the time needed for t to move next increases.

Algorithm 4 uses one additional information, namely q. Brent in his paper stated that q is a free parameter i.e. it can be chosen with any value. q is also utilized to determine the initial value (q_0) using $q_0 = q^u$ for $u \in [0,1)$, u is chosen using uniform distribution. Commonly these parameter would be set to $q = 2$ and $u = 0$ [8]. This paper will follow said parameter but with $u = 1$. That means, t will step for every 2^x iteration with x begins at 1 and x increases overtime.

2.3.5. Step function

Combining algorithm 3 and 4, the Pollard Rho's step function this paper will use is as follow.

$$step_{p_1} = f(p_1) \quad (4)$$

$$step_{p_2} = \begin{cases} p_1 & c = l \\ p_2 & \text{otherwise} \end{cases} \quad (5)$$

$step_{p_2}$ has to remember at least two states to function:

1. Current iteration, namely c.
2. $l = 2^i$ for given i and i increases overtime.

Additionally, one might want to memorize i, although it can be avoided.

2.3.6. Value of α and β

In line 9 of algorithm 3, the information that will be inferred is the discrete logarithm. To do so, α and β in (1) and (2) will be set to g and y respectively. That means (2) and (3) can be rewritten into (6) and (7)

$$f(x) = g^a * y^b \, mod \, m \text{ for arbitrary } a \text{ and } b \quad (6)$$

$$f(x) = \begin{cases} (y * x) \, mod \, m & x \equiv 1 \, mod \, 3 \\ (x * x) \, mod \, m & x \equiv 0 \, mod \, 3 \\ (g * x) \, mod \, m & x \equiv 2 \, mod \, 3 \end{cases} \quad (7)$$

Equation (6) implies that two *pointers* will be in form of (8) and (9)

$$p_1 = g^{a_{p_1}} * y^{b_{p_1}} \, mod \, m \quad (8)$$
$$p_2 = g^{a_{p_2}} * y^{b_{p_2}} \, mod \, m \quad (9)$$

Then, the inference can be done using equation (10)

$$\log_g(y) = \frac{a_{p_1} - a_{p_2}}{b_{p_2} - b_{p_1}} \left(mod \; ord_m(g)\right) \quad (10)$$

where $ord_m(g)$ is multiplicative order modulo m of g.

While there is no certain restriction on how to initialize p_1 and p_2, in this paper both *pointer* will be set to 1 at first. That means for both *pointer*,$a = 0$ and $b = 0$. It is worth of note however, that Pollard Rho method may fail when $b_{p_1} = b_{p_2}$ or $\gcd\big(b_{p_2} -$

$b_{p_1}, ord_m(g)\big) > 1$, causing (10) to be unsolvable. In such scenario, the Pollard Rho method will be redone but using different initial value. The reinitialization is done in the following manner.

$$p_1 = p_2 = g^a * y^b \; mod \; m \qquad (11)$$

for a and b are integers, $a \in \big[1, ord_m(g)\big)$ and $b \in \big[1, ord_m(g)\big)$, both chosen randomly.

2.3.7. Implementable Algorithm

Algorithm 5 sums up all stated explanation. In algorithm 5, p_2 will move at iteration $2, 4, 8, \dots$. While this is correct, in practice this paper uses a slightly modified version of algorithm 5 where p_2 moves at iteration $2, 2, 4, 8, 16, \dots$. Said modification is considered minor.

Algorithm 5 Baby-step Giant-step

```
Input: Four integers: $g, y, o$ and $m$ where $o$ is the multiplicative order modulo $m$ of $g$
Output: An integer $e$ that satisfies $g^e \equiv y\ (mod\ m)$ or "Not Exist"
1  Let $p_1$ be a pointer in form of $\langle a, b, value\rangle$
2  Let $p_2$ be a pointer in form of $\langle a, b, value\rangle$
3  Let $i$ be iteration counter
4  Let limit be current iteration limit
5  $i \leftarrow 0$
6  $limit \leftarrow 2$
7  $p_1 \leftarrow \langle 0,0,1\rangle$
8  $p_2 \leftarrow \langle 0,0,1\rangle$
9  repeat
10     if $i = limit$ then
11         $p_2 \leftarrow p_1$
12         $i \leftarrow 0$
13         $limit \leftarrow limit * 2$
14     else
15         $i = i + 1$
16     end if
17     if $p_1\ mod\ 3 \equiv 1$ then
18         $p_1.b \leftarrow p_1.b + 1$
19         $p_1.\text{value} \leftarrow (p_1.\text{value} * y)\ mod\ m$
20     else if $p_1\ mod\ 3 \equiv 0$ then
21         $p_1.a \leftarrow p_1.a * 2$
22         $p_1.b \leftarrow p_1.b * 2$
23         $p_1.\text{value} \leftarrow (p_1.\text{value} * p_1.\text{value})\ mod\ m$
24     else
25         $p_1.a \leftarrow p_1.a + 1$
26         $p_1.\text{value} \leftarrow (p_1.\text{value} * g)\ mod\ m$
27     end if
28 until $p_1 = p_2$
29 if $\gcd(p2.b - p1.b, o) > 1$ or $p_1.b = p_2.b$ then
30     Let $s \in \big[1, ord_m(g)\big)$
```

Algorithm 5 Baby-step Giant-step (cont.)

```
31     Let $t \in \big[1, ord_m(g)\big)$
32     $p_1 \leftarrow \langle s, t, (g^s * y^t)\ mod\ m\rangle$
33     $p_2 \leftarrow p_1$
34     Get back to line 9
35 end if
36 Let $b' \leftarrow (p_2.b - p_1.b)^{-1}\ mod\ o$
37 return $[(p_1.a - p_2.a) * b']\ mod\ o$
```

3. RESEARCH METHODOLOGY

This section will present a detailed explanation of how the research is done.

3.1. Dataset Preparation

Every dataset is designed to follow the parameters given in Digital Signature Algorithm. Each dataset will contain seven values; five comes from following the Digital Signature Algorithm, and the extra two are the free parameters.

1. N, a free parameter denoting how long parameter q will be in bit.
2. L, a free parameter denoting how long parameter p will be in bit.
3. q, a public key component. q must be a prime and must be a divisor to $(p - 1)$.
4. p, a public key component. p must be a prime.
5. g, a public key component. g will serve as the generator of the exponentiation.
6. y, a public key component. y is obtained by $y = g^x\ (mod\ p)$ where x is the user's private key.
7. $H(m)$, a hashed message.

Since this paper is interested in comparing the speed of Baby-step Giant-step and Pollard Rho towards achieving total break in Digital Signature Algorithm, there will be two group of datasets.

1. Dataset group which have fixed N and increasing L. There will be in total 475 datasets that fall in this group. There will be at most 10 different datasets for every distinct L. The N parameter is to be set at 10 and L is designed to not exceed 60.
2. Dataset group which have fixed L and increasing N. There will be in total 480 datasets that fall in this group. There will be 10 different datasets for every distinct N. The L parameter is to be set at 60 and N is designed not to exceed 57.

Other parameter (i.e. p, q, g, and y) in every dataset will adhere to given N and L. For $H(m)$, it will be set to have the same value for all dataset.

3.2. Testing

There will be two versions of the program that will be executed. One with Baby-step Giant-step implementation, and one with Pollard Rho with Brent cycle detection implementation. Each program is to be run with the two group of datasets as the input. Each dataset group is to be run separately. That is, there will be four executions: (1) Baby-step Giant-step with fixed *N* datasets, (2) Baby-step Giant-step with fixed *L* datasets, (3) Pollard Rho with fixed *N* datasets, and (4) Pollard Rho with fixed *L* datasets.

Each execution will be run in the following manner.

1. Receive a dataset from the corresponding group.
2. Note down the time prior to forgery.
3. Forge a signature.
4. Note down the time after forgery.
5. Output the running time by taking the difference between two times.

When handling the dataset group with fixed *N*, some adjustment needs to be done. Due to the time needed to do the forgery using dataset with fixed *N* is very small, step 3 is to be repeated 10000 times to magnify the result.

3.3. Result Evaluation

The running time for each dataset is to be aggregated according to their non-fixed free parameter. For example, in group with fixed *N*, the results which come from the dataset with the same *L* will be aggregated together. The results will be aggregated to their mean and standard deviation.

The result will be presented according to the dataset group. First, the graphic representation of each method performance is to be presented. Then the explanation regarding the graphic will follow. The evaluation will be ended with graphic comparison of both method, along with few comments concluding the explanation in the section.

4. RESULT

This section will discuss and evaluate the result of the research. This section will be mostly separated into two parts: (1) the result of methods using datasets with fixed *N* as the input, and (2) the result of methods using datasets with fixed *L* as the input.

4.1. Fixed *N* Datasets

4.1.1. Baby-step Giant-step

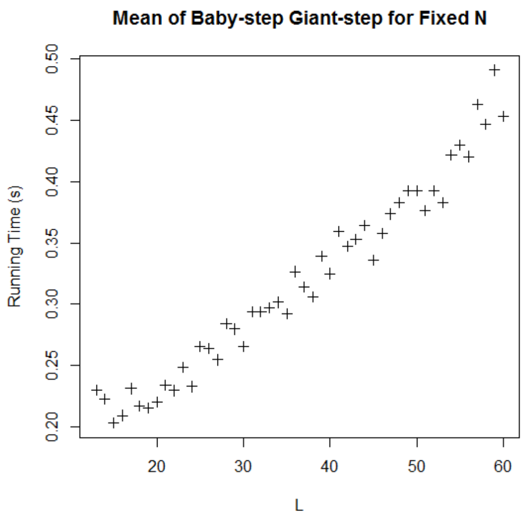

Figure 1: Running time mean of Baby-step Giant-step with fixed N datasets

Figure 1 shows that as *L* increase, so does the expected time needed. The increase happens linearly, though by minuscule factor i.e. around 0.01 second. When *L* reaches 50 and beyond however, the increase seemed to happen with slightly bigger factor.

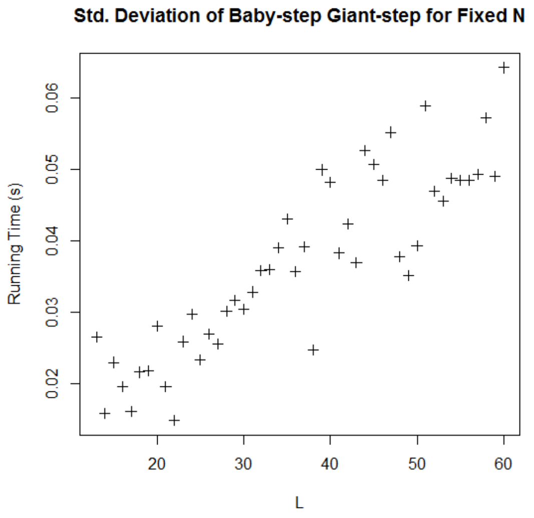

Figure 2: Running time standard deviation of Baby-step Giant-step with fixed N datasets

Similar phenomenon can be seen in figure 2. This figure shows that the uncertainty which Baby-step Giant-step method has increases as *L* increase. The increase happens linearly by factor of about 0.01 per 10 *L* increment.

4.1.2. Pollard Rho–Brent

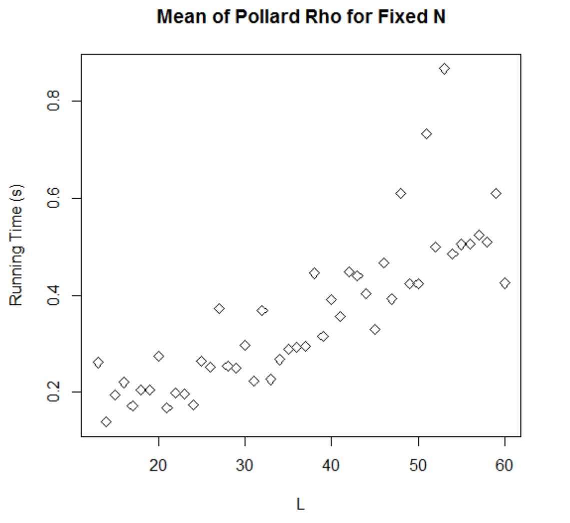

Figure 3:Running time mean of Pollard Rho with fixed N datasets

Figure 3 shows that as *L* increase, so does the Pollard Rho's expected running time. The increase happens in linearly fashion. The increase factor is higher than Baby-step Giant-step i.e. around 0.1 second per 10 *L*. Also, the increase is not really stable as can be seen when *L* reaches 50: some point peaks very high.

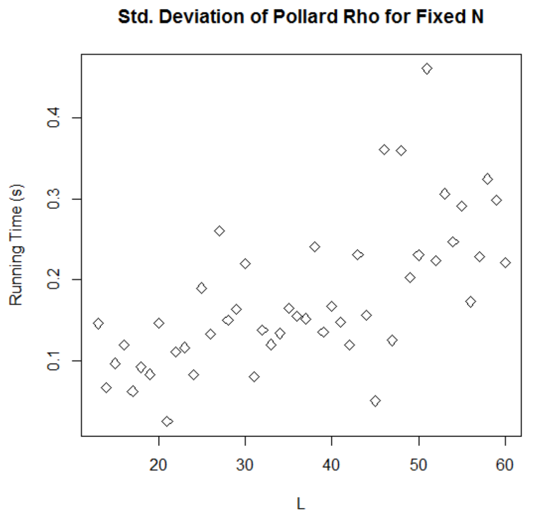

Figure 4: Running time standard deviation of Pollard Rho with fixed N datasets

When fed with fixed *N* datasets, Pollard Rho's standard deviation (figure 4) have similar property with its mean, that is, both scatters considerably although the standard deviation seems more irregular. It is difficult determine whether the standard deviation has linear growth function or quadratic growth function. As such, its growth function is yet to be determined.

4.1.3. Remark

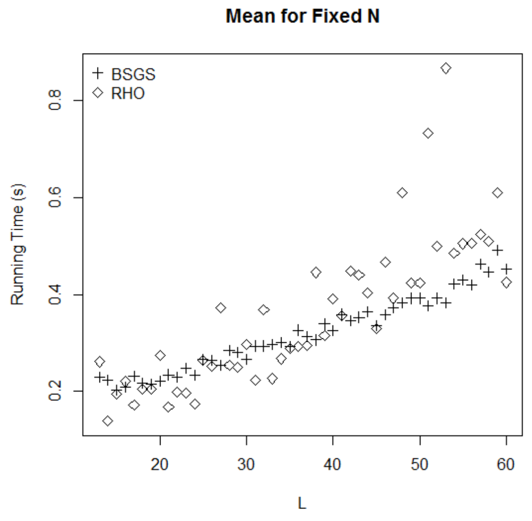

Figure 5: Running time mean of both method with fixed N datasets

Both method shows that as L increase, the time needed to solve the problem also increases in linearly fashion, albeit with different reliability. Baby-step Giant-step method has tendency to solve the problem with running time near the mean, whereas Pollard Rho has more randomness toward the time needed.

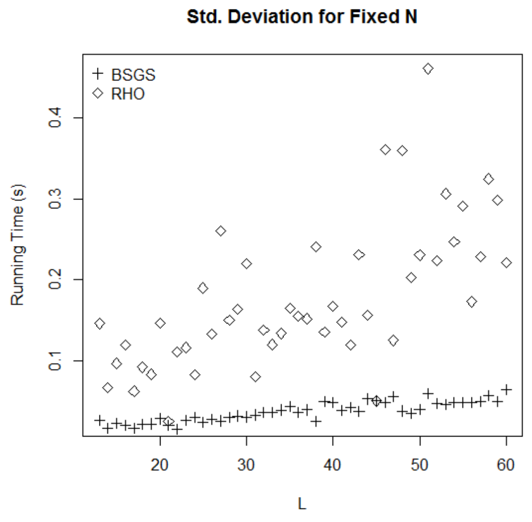

Figure 6:Running time standard deviation of both method with fixed N datasets

Figure 5 and 6 shows the mean and standard deviation comparison of both method respectively. From figure 6 it can be seen that the standard deviation of Pollard Rho method is strikingly high compared to Baby-step Giant-step.

4.2. Fixed *L* Datasets

4.2.1. Baby-step Giant-step

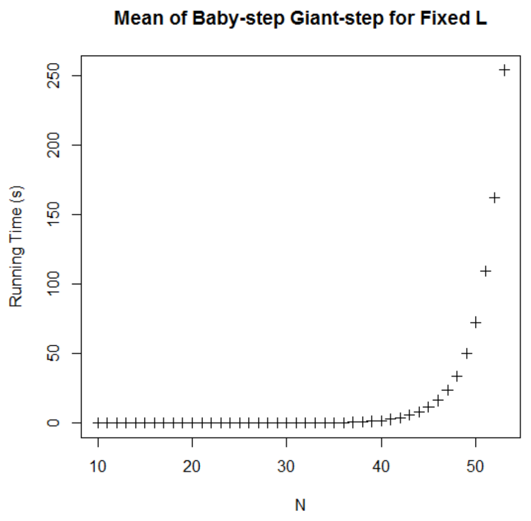

Figure 7: Running time mean of Baby-step Giant-step with fixed L datasets

Figure 7 shows that the expected running time grows with polynomial growth function. The growth becomes apparent when N goes over 40.

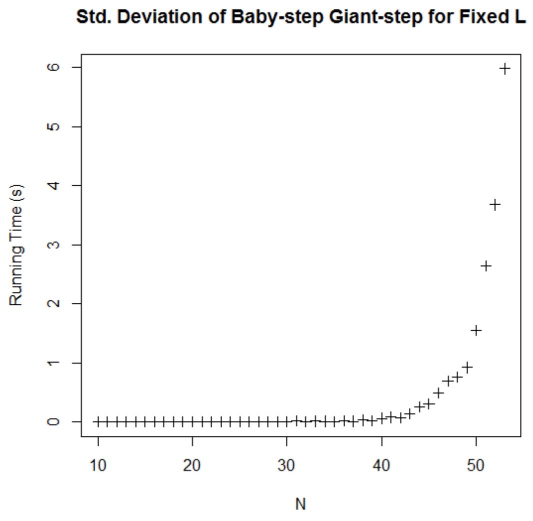

Figure 8: Running time standard deviation of Baby-step Giant-step with fixed L datasets

The standard deviation (figure 8) also follow the same growth function as the expected running time i.e. polynomial. The standard deviation starts to grow considerably when N goes over 40.

4.2.2. Pollard Rho–Brent

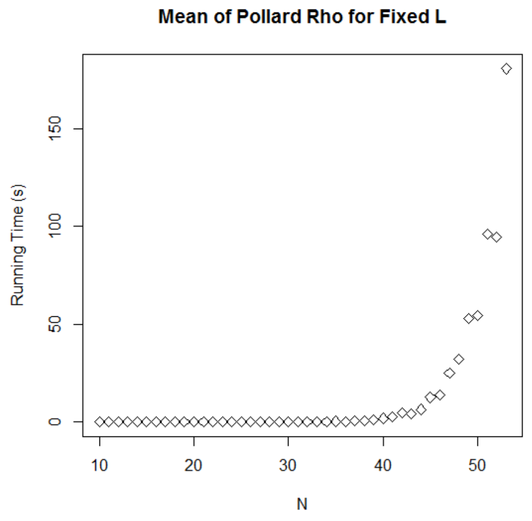

Figure 9: Running time mean of Pollard Rho with fixed L datasets

Pollard Rho with Brent cycle detection has expected running time that follows polynomial growth function as seen in figure 9. Just like its mean, its standard deviation also has polynomial growth function (figure 10).

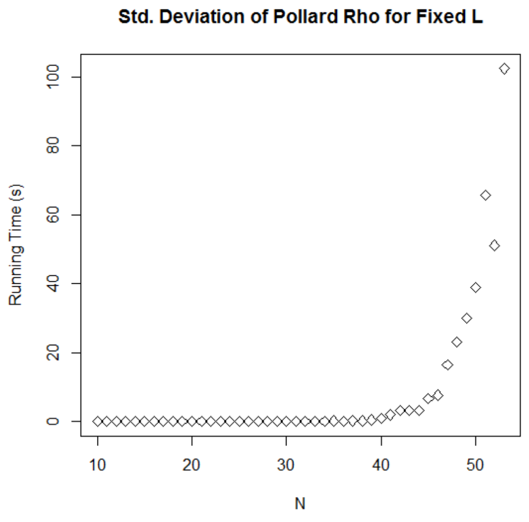

Figure 10: Running time standard deviation of Pollard Rho with fixed L datasets

On both figure, the growth starts to be noticeable once *N* reaches slightly below 40 and after that point the graphic pikes considerably.

4.2.3. Remark

It has been shown that both method has polynomial growth function for their mean and standard deviation. Even so, Baby-step Giant-step has higher expected running time than Pollard Rho, but has significantly lower standard deviation compared to Pollard Rho.

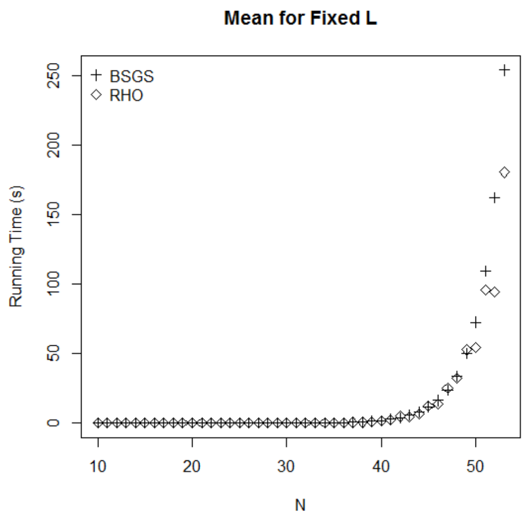

Figure 11: Running time mean of both method with fixed L datasets

Figure 11 and 12 shows the mean and standard deviation comparison of both method respectively. It can be seen that both method's mean have relatively similar growth function and that Pollard Rho's standard deviation rockets quicker than Baby-step Giant-step.

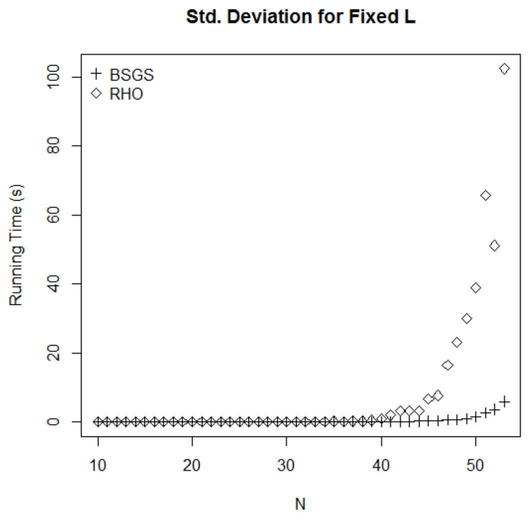

Figure 12: Running time standard deviation of both method with fixed L datasets

4.3. Coefficient of Variation's Comparison

The dispersion of running times which has been previously obtained is highly variable. It was taken into account that a different data distribution could have the same degree of dispersion, and to quantify this degree of dispersion, Coefficient of Variation is deemed to be able to handle this issue very well.

As a way to illustrate how close the performance of the two methods, Coefficient of Variations value will be calculated. This section will briefly explain about how Coefficient of Variation is calculated, and then how two method's Coefficient of Variation would compare.

4.3.1. Coefficient of Variation

The Coefficient of Variation is calculated using the following formula.

$$\widehat{c_v} = \frac{s}{\bar{x}} \tag{12}$$

where s is the data's standard deviation and $\bar{x}$ is the data's mean.

4.3.2. Coefficient of Variation for Fixed L Dataset

Table 1 provides tabular result of the $\widehat{c_v}$ for fixed L dataset.

Table 1: Coefficient of Variation - Fixed L Dataset

N	BSGS	Brent	N	BSGS	Brent
10	NA	NA	32	0.119	0.667
11	3.162	NA	33	0.117	0.448
12	NA	3.162	34	0.067	0.76
13	NA	NA	35	0.038	0.594
14	NA	NA	36	0.049	0.491
15	NA	NA	37	0.021	0.436
16	NA	NA	38	0.036	0.371
17	NA	3.162	39	0.022	0.441
18	NA	NA	40	0.033	0.453
19	3.162	3.162	41	0.029	0.759
20	NA	2.108	42	0.02	0.67
21	2.108	3.162	43	0.024	0.755
22	3.162	2.108	44	0.033	0.516
23	2.108	1.292	45	0.026	0.534
24	1.292	1.292	46	0.03	0.552
25	1.292	0.862	47	0.029	0.654
26	1.055	0.627	48	0.023	0.718
27	0.691	0.796	49	0.018	0.568
28	0.364	0.492	50	0.021	0.713
29	0.175	0.572	51	0.024	0.682
30	0.276	0.54	52	0.023	0.542
31	0.317	0.657	53	0.024	0.567

Notice there are few *NAs* in the table. This was obtained due to these entries have $s = 0$ and $\bar{x} = 0$. That means these entries have very small running time that the obtained result records the running time as 0.

Values in table 1 are presented graphically on figure 13. In the figure, the *NAs* are not plotted which explains why there are gaps on few first *N*. From the figure, both method's $\widehat{c_v}$ converges very quickly somewhere near $N > 30$. At that point, both method's $\widehat{c_v}$ is relatively stable with Pollard Rho's $\widehat{c_v}$ being slightly higher than Baby-step Giant-step's.

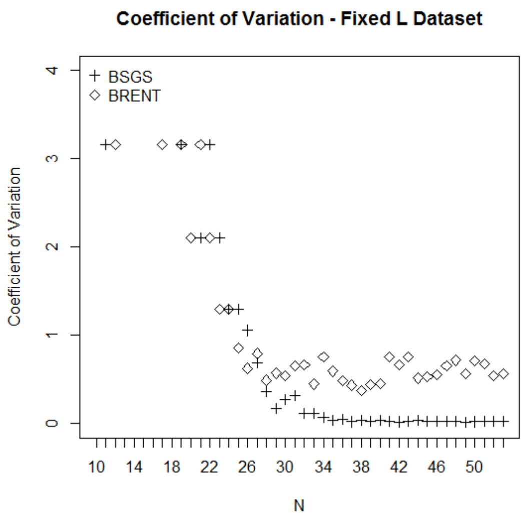

Figure 13: Coefficient of Variation plotting for fixed L dataset

The reason values with $N < 26$ have very high $\widehat{c_v}$ is that the score it derived from is very low, accompanied with the fact that the recorded result do not have enough precision. This in turn caused the score to be discretized into a score with lower precision. These values are considered as outlier.

Figure 14 shows the boxplot diagram for values in table 1. The median of Pollard Rho with Brent Cycle Detection's $\widehat{c_v}$ is slightly higher than Baby-step Giant-step's. It is as expected since Pollard Rho is a probabilistic algorithm. Also, the Pollard Rho with Brent Cycle Detection's upper hinge and lower hinge is higher compared to Baby-step Giant-step's. Overall, different *N* parameter contributes to dispersion 10 for a small factor, just about less than 1

4.3.3. Coefficient of Variation for Fixed N Dataset

Table 2 shows the $\widehat{c_v}$ for all tested *L* parameter and *N* parameter being constant. A major difference between these values with those of *L* fixed dataset's $\widehat{c_v}$ is that these values does not have *NAs*. These values then are considered to have no outlier.

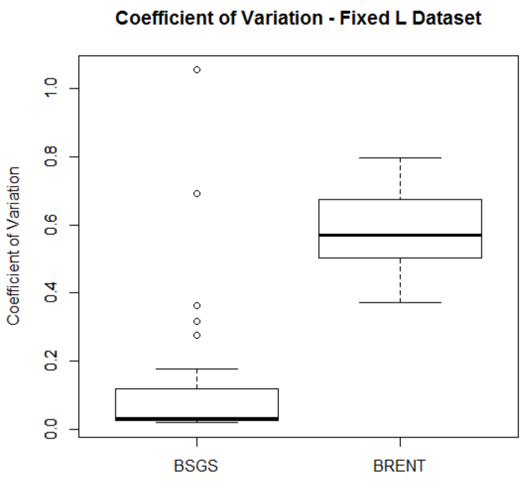

Figure 14: Coefficient of Variation boxplot for fixed L dataset

Table 2: Coefficient of Variation - Fixed N Dataset

L	BSGS	Brent	L	BSGS	Brent
13	0.116	0.558	37	0.125	0.514
14	0.072	0.483	38	0.081	0.541
15	0.113	0.494	39	0.147	0.427
16	0.094	0.542	40	0.148	0.428
17	0.07	0.363	41	0.107	0.415
18	0.1	0.449	42	0.122	0.266
19	0.101	0.406	43	0.105	0.523
20	0.127	0.531	44	0.145	0.388
21	0.084	0.147	45	0.151	0.152
22	0.065	0.56	46	0.135	0.771
23	0.104	0.591	47	0.148	0.319
24	0.128	0.477	48	0.099	0.591
25	0.088	0.719	49	0.09	0.478
26	0.102	0.527	50	0.1	0.544
27	0.1	0.696	51	0.156	0.629
28	0.106	0.591	52	0.12	0.447
29	0.113	0.653	53	0.119	0.353
30	0.115	0.739	54	0.115	0.507
31	0.112	0.358	55	0.113	0.576
32	0.122	0.374	56	0.115	0.341
33	0.121	0.53	57	0.106	0.437
34	0.13	0.5	58	0.128	0.637
35	0.147	0.568	59	0.1	0.488
36	0.109	0.531	60	0.142	0.521

The values in table 2 is presented graphically in figure 15. The figure shows that most of the time Pollard Rho with Brent Cycle Detection's $\widehat{c_v}$ is higher than Baby-step Giant-step.

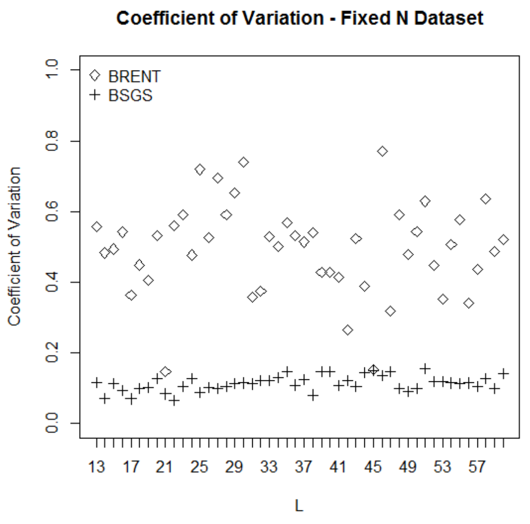

Figure 15: Coefficient of Variation plotting for fixed N dataset

Figure 16 gives insight on how the $\widehat{c_v}$ illustrated in figure 15 is distributed. The boxplot of Baby-step Giant-step's $\widehat{c_v}$ is lower than Pollard Rho with Brent Cycle Detection. It is interesting to notice that Baby-step Giant-step boxplot has significantly smaller size than Pollard Rho with Brent Cycle Detection, both the box and the whiskers. This shows how reliable Baby-step Giant-step is with regard to various L parameter

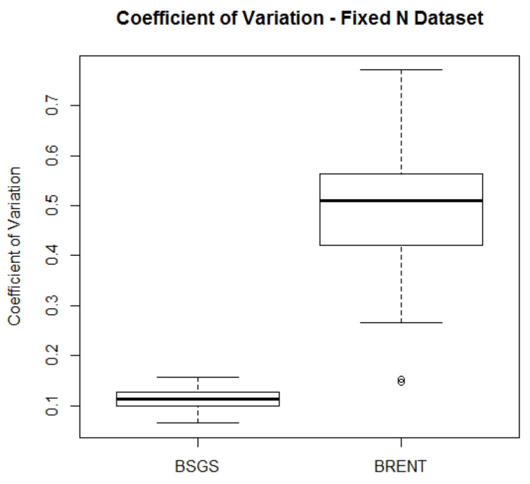

Figure 16: Coefficient of Variation plotting for fixed N dataset

4.3.4. Remark

It has been shown that both method has relatively low degree of dispersion i.e. $\widehat{c_v} < 1$. Even so, Pollard Rho with Brent Cycle Detection has higher $\widehat{c_v}$ than Baby-step Giant-step. Unfortunately, due to small sample size, deriving conclusion using these $\widehat{c_v}$ is a risky task as they could be a biased estimator. However, it can be narrowed down that on average, the Pollard Rho with Brent Cycle Detection's $\widehat{c_v}$ is higher than Baby-step Giant-step with factor of at least 0.2.

Another thing worth of note is that N parameter gives Baby-step Giant-step method minuscule degree of dispersion compared to L parameter. The N and L parameter gives a comparable degree of dispersion towards Pollard Rho with Brent Cycle Detection method with both parameter seemed to contributes equally.

5. CONCLUSION

The finding can be narrowed into two points.

5.1. The Effect of DSA Parameter

Two parameter of interest is the length of a prime modulus p, and the multiplicative order of the generator q. From the observation, the length of q contributes to both method's running time in linearly fashion with expected gradient less than 1 second for both method, whereas the length of p contributes to both method's running time in polynomial fashion.

Unsurprisingly, the length of p contributes higher than the length of q. After the length of p hits 50, the expected time difference between p and $p + 1$ is very noticable: about over 50 seconds. For DSA's security interest, increasing the size of p would be very desirable.

5.2. Two Method Comparison

From the finding, it can be concluded that both method has the same growth function. Even so, there is a high chance that Baby-step Giant-step takes time longer compared to Pollard Rho. The main factor that causes Baby-step Giant-step to be slower is the multiplicative order of the generator (i.e. q). In this experiment, it is when the multiplicative order of the generator is on around 40-bit long (i.e. $N = 40$) that Pollard Rho's expected run time begins to seemed better.

In term of their reliability, Baby-step Giant-step wins without contest. Baby-step Giant-step provides stable solution to signature forgery. Pollard Rho might seem unreliable due to how it could take significantly higher time at occasions. On the other hand, Pollard Rho also has the possibility to do the task way faster than expected.

The two method is also compared in term of how close their performance are. One characteristic that is inspected is the degree of dispersion. From observation, both method has degree of dispersion of less than 1, with Pollard Rho with Brent Cycle Detection is placed slightly higher than Baby-step Giant-step. The expected degree of dispersion's difference between the two method is over 0.2, i.e. $\Delta\widehat{c_v} > 0.2$. This degree of dispersion is consistent when the length of prime modulus p is constant, and when the length of multiplicative order of the generator q is constant.

This comparison provides an insight on how a variant of Pollard Rho would match to Baby-step Giant-step. As expected, Pollard Rho (specifically with Brent Cycle Detection) is outmatched. However, the difference between the two does not deemed to be that far. It is believed that for other variant of Pollard Rho, the difference between them and Baby-step Giant-step would be even smaller.

The author suggested that it would be interesting for future work to suppress the difference between Pollard Rho and Baby-step Giant-step as small as possible. The difference dimension used in this work can be utilized as a reference for further improvement in Pollard Rho's development.

It is worth to note however, that in this paper, the size of the value used is considered small in cryptography context. For future work, it might be interesting to see the behavior for even bigger number and find out how the growth of the execution time looks like.

6. ADDENDUM

6.1. Proof for equation (10)

It has been stated that given two *pointers*, namely p_1 and p_2 where

$$p_1 = g^{a_{p_1}} * y^{b_{p_1}} \bmod m$$
$$p_2 = g^{a_{p_2}} * y^{b_{p_2}} \bmod m$$

if $p_1 = p_2$, the discrete logarithm of y can be known. The procedure to infer the discrete logarithm is as follow.

$$p_1 \equiv p_2 \quad (mod\ m)$$
$$g^{a_{p_1}} * y^{b_{p_1}} \equiv g^{a_{p_2}} * y^{b_{p_2}} \quad (mod\ m)$$
$$\frac{y^{b_{p_1}}}{y^{b_{p_2}}} \equiv \frac{g^{a_{p_2}}}{g^{a_{p_1}}} \quad (mod\ m)$$
$$y^{b_{p_1}-b_{p_2}} \equiv g^{a_{p_2}-a_{p_1}} \quad (mod\ m)$$

Then apply log_g to both side. $mod\ m$ will turn to $mod\ ord_m(g)$ because of this.

$$\log_g\left(y^{b_{p_1}-b_{p_2}}\right) \equiv \log_g\left(g^{a_{p_2}-a_{p_1}}\right) \left(mod\ ord_m(g)\right)$$
$$\left(b_{p_1} - b_{p_2}\right)\log_g y \equiv a_{p_2} - a_{p_1} \quad \left(mod\ ord_m(g)\right)$$
$$log_g y \equiv \frac{a_{p_2} - a_{p_1}}{b_{p_1} - b_{p_2}} \quad \left(mod\ ord_m(g)\right)$$

■

According to Euler's Theorem, the following equation holds if a and n is coprime.

$$a^{\phi(n)} \equiv 1\ (mod\ n)$$

Since the multiplicative inverse of $b_{p_1} - b_{p_2}$ needs to be found, the Euler's Theorem has to apply. As the consequence, $b_{p_1} - b_{p_2}$ needs to be coprime with $ord_m(g)$. This raises an implicit constraint from equation $\log_g y \equiv \frac{a_{p_2}-a_{p_1}}{b_{p_1}-b_{p_2}}\left(mod\ ord_m(g)\right)$ that $\gcd\left(b_{p_1} - b_{p_2}, ord_m(g)\right) = 1$. Therefore, before solving the equation, it has to be checked whether or not $\gcd\left(b_{p_1} - b_{p_2}, ord_m(g)\right) = 1$ holds. In case of $b_{p_1} - b_{p_2} = 0$, the equation will also deemed unsolvable.

REFERENCES

[1] S. Goldwasser, S. Micali and R. L. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks," *Society for Industrial and Applied Mathematics,* vol. 17, 1988.

[2] Bailey et al., "The Certicom Challenges ECC2-X".

[3] Bos et al., "PlayStation 3 computing breaks 2^{60} barrier 112-bit prime ECDLP solved," [Online]. Available: https://lacal.epfl.ch/articles/112bit_prime/. [Accessed 14 November 2018].

[4] Bernstein et al., "Faster elliptic-curve discrete logarithms on FPGAs," 2016.

[5] P. C. van Oorschot and M. J. Wiener, "Parallel Collision Search with Cryptanalytic Applications," 1996.

[6] W. Stallings, "Digital Signatures," in *Cryptography and Network Security Principles and Practice, 5th Edition*, 5th ed., 2011.

[7] A. Menezes, P. var Oorschot and S. Vanstone, "Number Theoretic Reference Problems," in *Handbook of Applied Cryptography*, CRC Press, 1996.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[8] R. P. Brent, "An Improved Monte Carlo Factorization Algorithm," *BIT Numerical Mathematics,* vol. 20, 1980.

[9] J. M. Pollard, "A Monte Carlo Method for Factorization," *BIT,* vol. 15, 1975.