The implementation of the Pollard factorization method in the cryptanalysis of the Rabin public key algorithm

M A Budiman, D Rachmawati, and S Lydia
Departemen Ilmu Komputer, Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Sumatera Utara, Jl. Universitas No. 9-A, Kampus USU, Medan 20155, Indonesia

E-mail : mandrib@usu.ac.id, dian.rachmawati@usu.ac.id, shanti.lydia8911@gmail.com

Abstract. Rabin algorithm is an asymmetric cryptography algorithm that is used to obfuscate messages by using two different keys that are used in encryption and decryption process. Rabin algorithm is considered secure since its protection depends on the hardness of factoring a very large odd number into two very large prime numbers. Various factorization methods that exist are Fermat, Kraitchik, Euler, and variants of Pollard method. In this research, we test the security of the Rabin algorithm by cryptanalyzing its public key into its two private keys using the Pollard factorization algorithm. In order to check that a very large number is prime or composite, we use the Lehmann algorithm. Our research shows that Pollard is able to factorize some Rabin public keys and the size of the public key is not directly proportional to the processing time.

1. Introduction

Rabin algorithm is one of the first asymmetric cryptography algorithms that are based on the difficulty of public key factorization [1]. The concept of asymmetric cryptography was created by Whitfield Diffie and Martin Hellman and it is so called because it has a pair of different keys for encryption and decryption [2]. Encryption and decryption process in Rabin algorithm can be done with a relatively short time since it does not have a complicated process. But decryption in Rabin algorithm has more than one correct solution that has to be chosen from four possible solutions. However, using some methods, the recipient will be able to choose where the actual message [3].

The security of cryptography will be tested by the process of cryptanalysis [4] i.e., the process to find weaknesses in the encoding system, making it possible to obtain plaintext from previously camouflaged ciphertext. Various factorization methods may be used to cryptanalyze the Rabin algorithm, including Fermat, Kraitchik, Euler, and variants of Pollard algorithms [5].

In this research, Pollard method is used be used to factorize the public key of Rabin algorithm to obtain its private key. The Pollard method [6] is a simple factorization method in terms of the calculation process, because it is only in the form of modulo operations, primality testing, and checking based on the GCD (Greatest Common Divisor) value. Today, Pollard method has evolved into some of the better variations in the process [7].

The purpose of this research is to prove that Pollard factorization method is able to factoring the public key of Rabin algorithm which is quite large and able to see how effective this method is in factorizing the public key.
2. Method
In this section we will describe the process of Rabin algorithm, Lehmann algorithm, Rabin private key criteria test and Pollard factorization method. As in addition, C# codes of cryptanalysis of Rabin’s public key with Pollard factorization method will also be displayed.

2.1. Rabin Algorithm
Rabin algorithm is one of the asymmetric cryptography algorithms that use public key and private key which is a variant of Rivest Shamir Adleman (RSA) algorithm [8]. There are three processes in Rabin algorithm. They are key generation, encryption and decryption [9]. The actors who communicate with each other are sender and recipient.

2.1.1. Key Generation. Key generation is done by recipient. The steps following:
1. Randomly, generate two distinct prime numbers p and q as private keys with the terms \( p \equiv q \equiv 3 \pmod{4} \) or \( p \mod 4 = 3 \) and \( q \mod 4 = 3 \). Save the private key securely.
2. Calculate \( n = pq \) as public key that will be published to everyone who wants to send the message to the recipient.

2.1.2. Encryption. The encryption process is done by sender who has specified a message \( m \). The steps are as follows.
1. Accept public key \( n \).
2. The plaintext / message to be encrypted is converted to the value of binary \( m_i \) based on the value of ASCII table.
3. Then add the value of binary \( m_i \), by itself so that redundant information occurs which will facilitate finding the correct solution out of the four possible solutions during the decryption process later on. The value of binary is divided into two segments and each segment should have the same value.
4. Change the value of binary \( m \) to its decimal value.
5. Calculate the value of ciphertext \( c \) by using the formula:
   \[
   c = m^2 \mod n
   \]
   with \( c \) is the ciphertext, \( n \) is the public key, and \( m \) is the decimal value obtained in step 4.
6. Send \( c \) to the recipient.

2.1.3. Decryption. The decryption process is done by the recipient. The steps are as follows.
1. Accept \( c \) from the sender.
2. Find the value of \( Yp \) and \( Yq \) which is the GCD divider (Greatest Common Divisor) of \( p \) and \( q \) using the Extended Euclidean Algorithm.
   \[
   Yp \times p + Yq \times q = \gcd (p, q) 
   \]
   \[
   \Leftrightarrow Yp \times p + Yq \times q = 1 
   \]
3. Count the square root value of ciphertext to \( p \) and \( q \) by the formula:
   \[
   m_p = c^{(p+1)/4} \mod p \\
   m_q = c^{(q+1)/4} \mod q 
   \]
   with \( m_p \) is the square root of ciphertext to \( p \) and \( m_q \) is the square root of ciphertext to \( q \).
4. Calculate the values of \( r, s, t \) and \( u \) using the Chinese Remainder Theorem [10], with the following equation:
   \[
   v = Yp \times p \times m_q \\
   w = Yq \times q \times m_p \\
   \]
   So that
   \[
   r = (v + w) \mod n 
   \]
\[ s = (v - w) \mod n \]
\[ t = (-v + w) \mod n \]
\[ u = (-v - w) \mod n \]

5. Change the decimal values of \( r, s, t, \) and \( u \) into binary form. Then the values of binary \( r, s, t, \) and \( u \) are halved into two segments. Compare the two segments. If two segments have the same binary form, then the results of decryption of ciphertext \( c \) are obtained by changing the binary form of one segment to decimal form.

2.2. Lehmann Prime Test
This algorithm is needed when the number that will be tested is already big enough and cannot be remembered exactly. The steps are as follows [2].
1. Generate a random number \( a \), \( 1 \leq a \leq p \) (\( p \) is the number that is tested for its prime)
2. Calculate \( a^{(p-1)/2} \mod p \).
3. If \( a^{(p-1)/2} \mod p \equiv 1 \) or \(-1 \mod p)\), then \( p \) is not prime.
4. If \( a^{(p-1)/2} \mod p \equiv 1 \) or \(-1 \mod p)\), then the probability \( p \) is not prime is 50%.
5. Repeat the test above as many as \( t \) times (with the different value of \( a \)). If the result of calculating of step 2 equals 1 or \(-1\), but not always equal to 1, then the probability \( p \) is prime has error not more than \( 1/2^t \).

2.3. Rabin’s Privat Key Criteria Test
The number that has already been tested its primality, next step will be tested whether it qualifies as Rabin private key ie \( p \equiv q \equiv 3 \mod 4 \) or \( p \mod 4 = 3 \) and \( q \mod 4 = 3 \) with \( p \) and \( q \) is private key.

2.4. Pollard Factorization Method
Pollard method is one of the factorization methods by factoring the value of \( n \) which produces factors in the form of primes. The steps are as follows.
1. Select the integer value of \( n \) that be factored.
2. The initial value of the seed \( x = y = 2 \).
3. If \( n \mod 2 = 0 \) (even) then enter 2 to the list of interim results, where there is one definite figures as a factor, i.e., 1.
   To determine the new value of \( n \) then do following function.
   Function \( f(n) = n / 2 \)
   \[ n_1 = f(n) \]
   \[ n_2 = f(f(n_1)) \]
   \[ \cdots \]
   \[ n_i = f(n_{i-1}) \]
   Iterate as long as condition \( n_i \mod 2 = 0 \). If it doesn't meet, then the last value of \( n \) will be used in the next step.
4. If \( n \neq 1 \) and is an odd number, then it can be seen from two conditions:
   a. If \( n \) prime number, then enter the \( n \) numbers into the final result list, then get all the results.
   b. If \( n \) is not a prime number, then count \( x = x' \mod n \), then count \( s = \gcd (x'-1, n) \). If \( I < s < n \), then add \( s \) value to result list. Repeat this step when for the new value of \( n \) value \( n_i = n_{i-1}/s \) is still fit and use the new pitch values of \( x \) and value of \( y \) plus 1. The final step of this process will end up in the process 4a, and all the results will be obtained.
   If input \( n \) is accepted as public key of Rabin and \( p \) and \( q \) as private key, then to get private key by doing this method. N will be processed when it is a composite number. This can be seen in the following C # code.
using System;
using System.Drawing;
using System.Windows.Forms;
using System.Numerics;
using System.IO;
using System.Collections;
using System.Diagnostics;

namespace CryptanalysisbyPollard {
    public partial class Cryptanalysis : Form {
        public Cryptanalysis (){
            InitializeComponent();
        }

        private BigInteger gcd(BigInteger a, BigInteger b){
            if (a == 0 || b == 0)
                return 0;
            if (a == b)
                return a;
            BigInteger r = a % b;
            while(r != 0){
                a = b;
                b = r;
                r = a % b;
            }
            return b;
        }

        private BigInteger RandomIntegerBelow(BigInteger N) {
            byte[] bytes = N.ToArray();
            BigInteger R;
            Random random = new Random();
            do {
                random.NextBytes (bytes);
                bytes [bytes.Length - 1] &= (byte)0x7F;
            } while (R >= N);
            return R;
        }

        public bool LehmannPrimeTest(BigInteger p) {
            BigInteger trial = (BigInteger)BigInteger.Log10(p) + 2;
            for (int i = 0; i < trial ; i++){
BigInteger a = RandomIntegerBelow(p);
BigInteger temp=(p-1)/2;
BigInteger L = BigInteger.ModPow(a, temp, p);
if (L != 1 && (L-p) != (-1))
    return false;
}
return true;
}

public bool ModTest(BigInteger p){
    bool result=false;
    if (p % 4==3)
        result = true;
    return result;
}

public ArrayList pollard(BigInteger n){
    BigInteger x = 2;
    int y = 2;
    ArrayList factor = new ArrayList();
    while(n % 2 == 0){
        factor.Add(2);
        n = n/2;
    }
    while( n != 1){
        if(LehmannPrimeTest(n)){
            factor.Add(n);
            return factor;
        }
        x=BigInteger.ModPow(r,j,n);
        BigInteger s = gcd(r-1,n);
        if(1<s && s<n){
            factor.Add(s);
            n = n / s;
            y = 1;
        }
        y++;
    }
    return null;
}

void BtnProsesClick(object sender, EventArgs e){
    if(tbN_krip.Text=="")
        MessageBox.Show("Please input n!");
    BigInteger n,p,q;
    n = BigInteger.Parse(tbN_krip.Text);
    if(n%2==0)
        MessageBox.Show("N is an even number");
    else if (LehmannPrimeTest (n)==true)
        MessageBox.Show("N is a prime number");
    else{
        Stopwatch sw = new Stopwatch();
```java
sw.Start();
ArrayList result = pollard(n);
p=new BigInteger(Convert.ToInt64(result[0].ToString()));
q=new BigInteger(Convert.ToInt64(result[1].ToString()));

if (LehmannPrimeTest(p)==true && LehmannPrimeTest(q)==true && ModTest(p)==true && ModTest(q)==true);
sw.Stop();
tbP_krip.Text = p.ToString();
tbQ_krip.Text = q.ToString();
timeKrip.Text = Math.Round(Convert.ToDecimal(sw.ElapsedMilliseconds),4);
MessageBox.Show("Rabin’s Private Key is found");
}
else{
    MessageBox.Show("Rabin’s Private Key isn’t found");
}
```

3. Results and Discussions
In factoring of Rabin's public key, the Pollard method will produce only two resolutions as private keys namely p and q. Some experiments on public keys by comparing different values or lengths of bit to the resulting running time can be seen in Table 1.

| No. | n (digit) | Public Key | Private Key | p & q (digit) | Running Time (ms) |
|-----|----------|------------|-------------|---------------|------------------|
| 1   | 5        | 96113      | 17 223 431  | 3             | 3                |
| 2   | 6        | 290729     | 19 307 947  | 0             | 0                |
| 3   | 7        | 428641     | 19 859 499  | 0             | 0                |
| 4   | 7        | 2825041    | 19 2719 1039 | 3             | 3                |
| 5   | 8        | 11366473   | 24 1279 8887 | 4             | 7                |
| 6   | 8        | 45432133   | 26 4799 9467 | 8             | 58               |
| 7   | 9        | 332730169  | 29 19391 17159 | 5             | 26               |
| 8   | 10       | 4776615449 | 33 73483 65003 | 5             | 50               |
| 9   | 10       | 7849068841 | 33 88667 88523 | 6             | 655              |
| 10  | 11       | 8882109749 | 34 97303 91283 | 6             | 93               |
| 11  | 12       | 16030586417| 34 117679 136223 | 6             | 40               |
| 12  | 13       | 146806688969| 38 649567 226007 | 7             | 15               |
| 13  | 13       | 559157004101| 40 884251 632351 | 7             | 17951            |
| 14  | 14       | 5849730085897| 43 3509711 1666727 | 7             | 5090             |
| 15  | 14       | 22659122405749| 45 4483543 5053843 | 7             | 19857            |

Table 1. Comparison of Public Key n in Bit to Running Time
In Figure 1, it is shown a graph of the comparison between the length of public key (in bits) and the time of the cryptanalysis process accordance with Table 1 above.

Based on test data in Table 1 and Figure 1 can be seen that the value of the public key \( n \) does not have significant effects on the running time velocity (ms), especially on the same digit. This can be seen in the value of \( n \) with 18 digits, where is the value \( n_{20} < n_{21} < n_{22} \) with running time obtained \( n_{20} = 4208 \text{ ms}, n_{21} = 3343 \text{ ms}, \) and \( n_{22} = 129 \text{ ms} \). It seems that the value of public key \( n \) does not always correlate proportionally with the running time.

The diminishing value of public key \( n \) also does not always result in faster running time. Running time may fluctuate, as seen in Figure 1 where dots show increases and decreases irregularly. As can be seen in the public key \( n \) with 16 digits, where \( n_{17} < n_{18} < n_{19} \) but the process time fluctuated irregularly: \( T_{18} < T_{17} < T_{19} = 66 < 7711 < 15765 \).

4. Conclusions
There are two conclusions of this research that can be obtained. Firstly, Pollard factorization method is proved capable to factorize the public key \( n \) of Rabin algorithm into private keys \( p \) and \( q \) with the size of public key up to 18 digits. Secondly, the length of digits or the size of public key \( n \) does not determine the running time. The length of time may fluctuate inconsistently even if the public key has the same length of digits.

Acknowledgments
We are very grateful for the cooperation of colleagues and the support of the Departemen Ilmu Komputer, Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Sumatera Utara.
References

[1] Srivastava A K and Mathur A 2013 *The Rabin Cryptosystem & Analysis in Measure of Chinese Reminder Theorem* International Journal of Scientific and Research Publications volume 3 (ISSN:2250-3153) 493-501

[2] Schneier B 1996 *Applied Cryptography* 2nd Ed (New York:John Wiley & Sons, Inc)

[3] Rabin M O 1979 *Digitalized Signatures and Public Key Functions as Intractable as Factorization* (Cambridge : Massachusetts Inst of Tech)

[4] Yang F Y 2006 *Cryptanalysis on An Algorithm for Efficient Digital Signatures* (Taiwan : Department of Computer Science and Information Engineering Chaoyang University of Technology)

[5] Mollin R A 2007 *An Introduction to Cryptography* 2nd Ed(USA:Chapman & Hall/CRC)

[6] Yan S Y 2009 *Primality Testing and Integer Factorization in Public Key Cryptography* 2nd Edition (New York : Springer)

[7] Montgomery P L 1987 *Speeding The Pollard and Elliptic Curve Methods of Factorization* California Mathematics of Computation Volume 48 Number 177 Pages 243-264

[8] Hinek M J 2010 *Cryptanalysis of RSA and Its Variants* (USA : Chapman & Hall/CRC)

[9] Wandani, Budiman M A and Sharif A 2012 *Implementasi Sistem Keamanan Data dengan Menggunakan Teknik Steganografi End of File (EOF) dan Rabin Public Key Cryptosystem* Vol 1 No 1 (Medan:Universitas Sumatera Utara)

[10] Smart N P 2016 *Cryptography Made Simple* (Switzerland : Springer International Publishing Switzerland)