Security Analysis of RSA Algorithm

For Digital Signature and Way to Improve It

Galih Andana - 13507069
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
Xugalz_@hotmail.com

***Abstract* — Nowadays, Internet are being the public network communications which can be accessed by all people around the world. If we are trying to send some messages but there is not enough or appropriate security service inside, then some message tapping or contents modification can occure without known by us. But there is some ways to keep its secrets and one of these methods is** *Digital Signature* **or in Indonesian we call it "**Tanda tangan Digital**".**

With this kind of method, we could keep our message contains and validitiy the authorizeed , whatever if it was come from the true person. But, there are still pro and contra about this kind of methods among people due to its security and ability. So in this paper, author will bring you the advantages of using RSA and also the weakness of this algorithm. And in addition, the author will bring you way how to improve its security and ability.

***Index Terms* —** *Digital Signature, security, ability, advantages, weakness, RSA Algorithm.*

I. INTRODUCTION

I.1. RSA History and Founders

The first triggered work for RSA's born was come from *Clifford Cocks*, a British mathematician, who was working for the UK intelligence agency GCHQ. At that time, he described an equivalent system to encrypt an internal document in 1973, but it needed some expensive computers to accomplish the implementation it at the time, so it was mostly just considered as a curiosity and as far as is publicly known, was never deployed. His discovery, however, was not revealed until 1998 due to its top-secret classification, and then *Rivest*, *Shamir*, and *Adleman* devised RSA independently of Cocks' work.[1]

In cryptography, RSA it self (which stands for Rivest, Shamir and Adleman who first publicly described it) is an algorithm for public-key cryptography. RSA is an algorithm which is known to be suitable for digital signing as well as encryption, and was one of the first great advances in public key cryptography. RSA is widely used in electronic commerce protocols, and is believed to be secure given sufficiently long keys and the use of up-to-date implementations.

The RSA algorithm was publicly described in 1978 by Ron Rivest, Adi Shamir, and Leonard Adleman at MIT. Since a paper describing the algorithm had been published in August 1977[2] prior to the December 1977 filing date of the patent application, regulations in much of the rest of the world precluded patents elsewhere and only the US patent was granted.

I.2. RSA Algorithm

The best security system of RSA Algorithm is depend on its dificulty to factorize the big integers to become its prime factors *p* and *q*. The factorization process is done to get the *private key* for our message. RSA will be stand as long as there has not been founded any effective algorithm to factorize a big integers to its prime factors.

Keys in RSA consists of *public key*, which can be known to everyone and is used to encrypt the message. Then the message which has been encrypted by *public key*, can only be decrypted by using *private key*. But in *digital signaturing*, we reverse the common way about its both keys. In here, we rather use our *private key* to encrypt the message as same as our hand-sign and then we publish our *public key* to everyone. So if later they get our message, they can decrypt it with our *public key* and check whatever it message was come from the true person and not has been modified by other people or not.

[1] A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.

[2] SIAM News, Volume 36, Number 5, June 2003.

II. General Ways From RSA Algorithm

II.1. RSA Key Generator Algorithm

Here it is the algorithm to generate both public and private keys for RSA :

1. Firstly, let's choose two distinct *secret* prime numbers
$$p \; and \; q.$$
We rather use p and q as same bit-length integers and should be choosen uniformly at random and as length as possible for the security reason.
2. Then let's compute :
$$n = p \times q.$$
n is treated as the next modulus for our both public and private keys. And n can be published to everyone.
3. Let Pi be the block of (plain) text to be encrypted. Actually Pi is the numerical equivalent of the text which may either be single letters or blocks of letters, just as long as
$$Pi < (p-1) \times (q-1) = \varphi(n)$$
(φ is Euler's totient function).
4. Choose an integer e such that 1 < e < φ(n), and *e* and φ(n) share no divisors other than 1 (*e* and *φ(n)* are coprime). Then,, *e* will be used togerther with *n* as our encryption key exponent (*private key* in digital signaturing).
5. Now, let's determine *d* (using modular arithmetic) which satisfies the congruence relation. Which is :
$$e \, . \, d \equiv 1 \; (mod \; \varphi(n))$$
Or in other words, we can says :
$$d = \frac{1 + k(\varphi(n))}{e}$$
Then, *d* will be used together with *n* as our decryption key and be published to everyone, so they can check whatever a message is still authentic (*public key* in digital signaturing).
6. Note : just for efficiency the following founded prime values may be precomputed and stored as part of the private key: *p* and *q*: the primes from the key generation.[3]

II.2. RSA Encryption and Decryption

There are some common variables known in RSA :

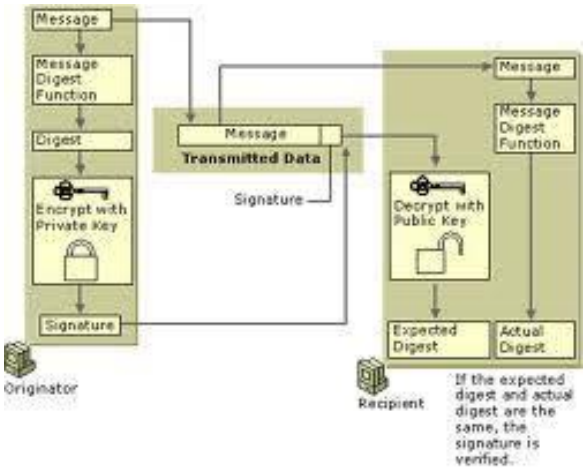
- *p* and *q* — prime integers (*secret*)
- *n* = *p* x *q* — (*not secret*)
- Φ (*n*) = (*p* – 1)(*q* – 1) — (*secret*)
- *e* — encryption key (*secret*)
- *d* — decryption key (*not secret*)
- *m* — plainteks (*secret*)
- *c* — chiperteks (*not secret*)

After both *public* and *private* keys are successfully generated, next step is to sign the plainteks message by convert its text into bit-bit of integer so that we can easily doing math operation for it. Then the encryption algorithm is :

$$E_e(m) = c \equiv m^e \; mod \; n$$

Then, after we get the ciphertext we can just append it at the bottom of our message as *Signature* from us. Next time, if some one has gotten our message, she/he can just decrypt it with our *public key* :

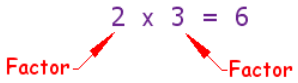
$$D_d(c) = m \equiv c^d \; mod \; n$$



Picture 1- Digital Signaturing Process[4]

II.3. The Security Rate of RSA

RSA Algorithm's security is based on its difficultness to factorized its *n* as a big integer into *p* and *q* (the prime factors), which they both are also big integers. Prime number is a whole number, greater than 1 that can be evenly divided *only* by 1 or itself, because of this specificity, and all of the others numbers can be factorized into prime numbers. "Factor" are the numbers you multiply to get another number :



And the main problem to break up the RSA algorithm is to find out both *prime factors* of *n* as *p* and *q* which is consists of up to 512 bit-integers. After *n* could be factorized into *p* and *q*, such that *n* = *p* x *q*, then Φ(n) = (p-1)(q-1) can be easily compute. Then, because *d* as the public key is released to everyone (which is *not secret* due to its functionality), the encryption key *e* also can be easily found by using formula :

$$e = \frac{1 + k(\varphi(n))}{d}$$

[3] http://www.krellinst.org/UCES/archive/modules/charlie/pke/node10.html

[4] technet.microsoft.com/en-us

It is because : e . d ≡ 1 (mod Φ(n)).

From now on, generally, it can be concluded that RSA is still secure enough as far as the *n* number as the most important key here consists of too big integers that could not be factorized as well.

III. RSA Attacks and Weaknesses Points Analysis

III.1. RSA Attacks

III.1.1. Factoring Large Integers

The first kind of attacks that implied on RSA algorithm is to factorting the modulus *n.* Given the both factorization of *n*, an attacker can easily construct *n* from the decryption exponent formula :

$$e \,.\, d \;\equiv\; 1 \,(mod\ \varphi(n))$$

In this case, we could only solves the modulus by using *brute-force attack* on it. Until now, there is no one of factoring algorithms which have been steadily improving enough to posing a threat to the RSA security, whereas factoring large integers is one of computational mathematics.

For completeness the writer notes that the current fastest factoring algorithm is *the General Number Field Sieve*. In number theory, this algorithm (GNFS) is the most efficient classical algorithm known for factoring integers which is larger than 100 digits. Heuristically, its complexity for factoring an integer n (consisting of log n bits) is of the form Its running time on n-bit integers is :

$$\left(\left(c + o(n)\right) \times n^{\frac{1}{3}} \times log^{\frac{2}{3}} \times n\right)$$

for some c < 2.[5]

This number filed of *GNFS* sieve has the same main principle (both special and general) as an simple rational sieve. Suppose *f* is an *n* - degree polynomial of rational number *Q*, and *r* is a complex root of *f*. Then, f(*r*) = 0, which can be rearranged to express r^n as a linear combination of powers of *r* less than *n*. This equation can be used to reduce away any powers of $r \geq n$. For example, if $f(x) = x^2 + 1$ and *r* is the imaginary unit *i*, then $i^2 + 1 = 0$, or $i^2 = -1$. This allows us to define the complex product :

$(a + bi)(c + di) = ac + (ad + bc)i + (bd)i^2 = (ac - bd) + (ad + bc)i.$

In general, this leads directly to the algebraic number field $Q(r)$, which can be defined as the set of real numbers given by:

$$a_{n-1}r^{n-1} + \ldots + a_1 r^1 + a_0 r^0,$$

where $a_0, \ldots, a_{n-1}$ are in Q.

The product of any two such values can be computed by taking the product as polynomials, then reducing any powers of $r \geq n$ as described above, then let *a* be value in the same form.

Then lets compute the given integers *N* and *e* which *e* satisfying $gcd(e, \varphi(N)) = 1$ by the algorithm which is implemented polynom factorizing as explained above, define the function f_e, N :

$$Z_N^* \rightarrow Z_N^* \; by \; f_{e,N}(x) = x^{\frac{1}{e}} \; mod \; N$$

III.1.2. Common Modulus Seeker

To avoid generating a different modulus $N = pq$ for each user one may wish to fix *N* once and for all. The same *N* is used by all users. A trusted central authority could provide user - *i* with an unique pair of his / hers ei_i and d_i from which user - *i* and forms a private key (N, e_i) and a public key (N, d_i).

At the first glance this may seem to work for a signature :

$$C \;=\; M^{e_{Alice}} \; mod \; N$$

intended for Alice cannot be modified by Bob since Bob does not possess e_{alice} . However, this is incorrect and the resulting system is still not insecure enough because Bob still can recover the encrytion key by Alice public key

$$d_{Alice}$$

This observation has been done by Simmons, he showed that an RSA modulus should never be used by more than one entity by looking up from its modulo.

III.1.3. Blinding Attack

Now let (N, d) be the private key and $(N,\ e)$ be the corresponded public key. Then, suppose a problem that someone wants us to give signature on a uncommon message $M \in Z_N^*$. But trully, it just the fake message which derived from *r* where created by someone :

$$M' \;=\; r^e \; M \; mod \; N$$

We may be accept to provide our signature S_0 on the true-looking M'. But then our private signature on message *M* will be revealed by :

$$S \;=\; S'/r \; mod \; N$$

Due to :

$$S^e \;=\; (S')^e/r^e \;=\; (M')^{ed}/r^e \equiv M'/r^e \;=\; M \; (mod \; N)$$

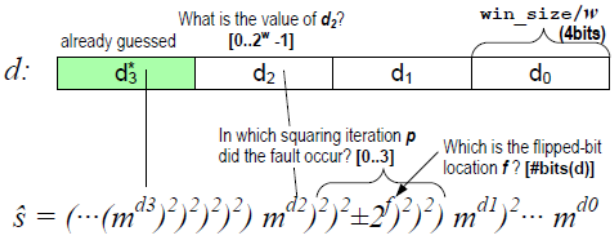
By this way, someone can obtain our valid signature on a certain message by asking us to sign a random "*blinded"* message.

[5] http://crypto.stanford.edu

III.1.4. Fault - Base Attack

In this case, we will have 3 common variables known as $\langle n, d, e \rangle$ where n and e are public known and d is private known only, then for which the signature with the private key d and length N is computed using the fixed-window exponentiation (*FWE*) algorithm with a window size w, we call k the number of windows in the private key d, that is, $k = N/w$. Let us call $\hat{s}$ a corrupted signature of the message m computed with the private key d. Assume that a single-bit binary value change has occurred at the output during its computation.

An attacker that can collect at least $S = k \cdot ln(2k)$ different pairs $\langle m, \hat{s} \rangle$ has a probability pr = 1/2 to recover the private key d of N bits in polynomial time - $O(2^w N^3 S)$.



Picture 2 - Fault Base Attack[6]

This is the pseudo-code look up he significant window from data public key and N , which are public known as the sample picture above is :

```
window search (m, s, e, win_size, win_idx)
{
     found = 0;
     for(d[win_idx] in [0..2^win size-1];
        sqr_iter in [0..win_size-1];
        fault_in [0..#bits(d)-1] )
           found += test_equation 10( m, s, e,
           win_idx,d[win_idx],sqr_iter,fault_loc)
     if (found == 1)
        return d[win_idx]
     else
        return -1
}
```

The private key then will be invoked by the given window above with this pseudo-code function :

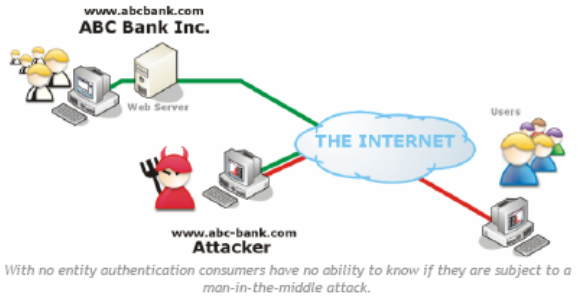
```
private key recovery ( array<m,s>, e, win_size)
{
     num win = #bits(d) / win_size
     for(win idx_in [num_win-1..0] )
        for (<m,s> in array<m,s>)
        {
           d[win_idx] = window_search(m,s,e,
                        win_size, win_idx)
           if (d[win_idx] >= 0)
                break;
        }
        if (d[win idx] < 0)
           double_win_size;
}
```

[6] www.eecs.umich.edu

III.2. RSA Weakness

III.2.1. Weakness At The Key Exchange

Such as common encryption algorithm, the main problem for being a successful sent and received a message is if it is not being excused by anyone who does not have any privilage except the true receiver. And in cryptography it is called as "*Man-In-The-Middle Attack*".



Picture 3 - Man-In-The-Middle-Attack[7]

It can be occured because the sender and ther receiver must exchange their public key to each other by any communications tools or network, this is absolutely caused an opportunity by an attacker to be the *Man-In-The-Middle-Attack*, who was explained by picture above.

In case, Alice and Bob send their own key off into communications network, and let's call Carol cut through into their communication then he pretend as one of them (Alice or Bob). Carol then pretend as Alice to Bob, and send him his public key, because of Bob's trustness, Bob will accept Carol's key and sure that it is come from true Alice. This is also happen at the other side, Carol pretend as Bob to Alice, then he sent his public key to Alice. Alice will accept it because she believes that it was come from true Bob.

Next step, Carol keep the true message from Bob to Alice and send the encrypted message, which has been encrypted by his private key in signaturing. Then Alice will believe that this message was authentic and came from Bob then accept it without knowing that was Carol in the middle of them.

III.2.1. Known Plainteks Moreover Use The Same Key

Other side of RSA weakness is in its encryption which has a common pattern to be analysed if we had known one of the plainteks sent. Then, we could choose other plainteks to be encrypted by public key in our plainteks dictionary. This dictionary was used to break or recover the private key of the true receiver and to modify the message without being known.

Without loss of generality, assume that $n = pq$ in RSA is a 512 bit number. Let e be the public exponent which is publicly known and d be the secret exponent which is

[7] www.securitydocs.com

stored inside the dictionary device. Now choose P from plaintext's storage, then the corresponding ciphertext is

$$C = P^e \quad (1)$$

(In the following, only residues modulo n are shown, e.g., we use P^e instead of P^e mod n). We denote the binary representation of the secret exponent as :

d = d511|d510| ...|di|...|d1|d0 (2)

where di, takes value 1 or 0, is the ith bit and where x|y denotes concatenation of x and y. Further, we denote

C0=C, C1=C^2, C2=C^{2^2}, ..., C511=C^{2^511} (3)

Given C and d, we can express the corresponding plaintext P as

P=(C511^d511)(C510^d510)...(Ci^di) ...(C1^d1)(C0^d0)

We assume that the attacker is in physical possession of this device and that he can repeat the experiment with the same key by applying external physical effects to obtain outputs due to single bit errors.[8]

IV. Contribution And Way To Improve RSA

IV.1. Prime Number Generator

RSA Algorithm it self, consists of 2 base cycle, first is key generator and the second is message sending, so the writer try to implement the Prime number generator by making own algorithm to generate and check whatever a big integer is prime or not.

Picture 4 - Self Key Generator

By the 25 times experimental result to generate N-bit prime number, we need at least :

bit for *p* and *q*	Average Processing (s)	Times (x)
3 – 4	0,2	7
5 – 6	0	7
7 – 8	17,2	5
9 – 10	594,2	3
10 – 12	902,1	3

[8] cryptome.quintessenz.at

```
/* Own build Prime Checking */
public bool isPrima(BigInteger x)
{
   int count = 0;
   if (x >= 2)
   {
       if (x != 2)
       {
            if (x % 2 != 0)
            {
                 for (BigInteger i = 2; i <=
                 (x.sqrt() + 1); i++)
                 {
                      if ((x % i) == 0)
                          count++;
                 }
                      if (count > 1)
                           return false;
                       else
                           return true;
             }
             else
                return false;
        }
             else
                return true;
    }
    else
        return false;
}
```

Its prime checking use the effective looking up just by square rootof the big integers checked, it is also usefull for making private key and public key for RSA algorithm, both at the same time.

IV.2. Other "Way" Computational to Generate Prime Number

Beside using own effective algorithm, the writer also included a new algorithm provided by BigInteger class in C# which is very usable to get or checked about kinds of big prime number. In this library, every prime number from 2 until 2000 are noted structurally. Its can handle at most 512 bit length for *p* and *d* within 60 sec process.

Here it is the short capture of the BigInteger class :

```
public class BigInteger{
// maximum length of the BigInteger in uint (4
bytes)
// change this to suit the required level of
precision.

private const int maxLength = 70;

// primes smaller than 2000 to test the
generated prime number

public static readonly int[] primesBelow2000 = {
        2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
89, 97,
        101, 103, 107, 109, 113, 127, 131, 137,
139, 149, 151, 157, 163, 167, 173, 179, 181,
191, 193, 197, 199,
        211, 223, 227, 229, 233, 239, 241, 251,
257, 263, 269, 271, 277, 281, 283, 293,
        307, 311, 313, 317, 331, 337, 347, 349,
353, 359, 367, 373, 379, 383, 389, 397,
        401, 409, 419, 421, 431, 433, 439, 443,
449, 457, 461, 463, 467, 479, 487, 491, 499,
```

```
        503, 509, 521, 523, 541, 547, 557, 563,
569, 571, 577, 587, 593, 599,
        601, 607, 613, 617, 619, 631, 641, 643,
647, 653, 659, 661, 673, 677, 683, 691,
        701, 709, 719, 727, 733, 739, 743, 751,
757, 761, 769, 773, 787, 797,
        809, 811, 821, 823, 827, 829, 839, 853,
857, 859, 863, 877, 881, 883, 887,
        907, 911, 919, 929, 937, 941, 947, 953,
967, 971, 977, 983, 991, 997,
        1009, 1013, 1019, 1021, 1031, 1033, 1039,
1049, 1051, 1061, 1063, 1069, 1087, 1091, 1093,
1097,
        1103, 1109, 1117, 1123, 1129, 1151, 1153,
1163, 1171, 1181, 1187, 1193,
        1201, 1213, 1217, 1223, 1229, 1231, 1237,
1249, 1259, 1277, 1279, 1283, 1289, 1291, 1297,
        1301, 1303, 1307, 1319, 1321, 1327, 1361,
1367, 1373, 1381, 1399,
        1409, 1423, 1427, 1429, 1433, 1439, 1447,
1451, 1453, 1459, 1471, 1481, 1483, 1487, 1489,
1493, 1499,
        1511, 1523, 1531, 1543, 1549, 1553, 1559,
1567, 1571, 1579, 1583, 1597,
        1601, 1607, 1609, 1613, 1619, 1621, 1627,
1637, 1657, 1663, 1667, 1669, 1693, 1697, 1699,
        1709, 1721, 1723, 1733, 1741, 1747, 1753,
1759, 1777, 1783, 1787, 1789,
        1801, 1811, 1823, 1831, 1847, 1861, 1867,
1871, 1873, 1877, 1879, 1889,
        1901, 1907, 1913, 1931, 1933, 1949, 1951,
1973, 1979, 1987, 1993, 1997, 1999 };


private uint[] data = null;              //
stores bytes from the Big Integer
public int dataLength;                  // number
of actual chars used



//*****************************************
************************
        // Constructor (Default value for
BigInteger is 0

//*****************************************
************************

public BigInteger()
{
    data = new uint[maxLength];
    dataLength = 1;
  }
```

There is a function in Big Integers called ModInverse(), this is can help us findout the true factor of prime number with extra fast time. By using BigInteger algorithm, the application builded here could operated better than before esspecially at the key.

```
public BigInteger modInverse(BigInteger modulus)
{
  BigInteger[] p = { 0, 1 };
  BigInteger[] q = new BigInteger[2];//quotients
  BigInteger[] r = { 0, 0 };        //remainders
  int step = 0;
  BigInteger a = modulus;
  BigInteger b = this;

while
(b.dataLength>1||(b.dataLength==1&&b.data[0]!=0))
{
     BigInteger quotient = new BigInteger();
     BigInteger remainder = new BigInteger();
     if (step > 1)
     {
         BigInteger pval=(p[0]-(p[1]*q[0]))%modulus;
         p[0] = p[1];
         p[1] = pval;
     }
     if (b.dataLength == 1)
       singleByteDivide(a,b,quotient,remainder);
     else
       multiByteDivide(a,b,quotient,remainder);
     q[0] = q[1];
     r[0] = r[1];
     q[1] = quotient; r[1] = remainder;
     a = b;
     b = remainder;
     step++;
 }

If(r[0].dataLength>1||(r[0].dataLength==1&&r[0].data[0]
!= 1))
  throw (new ArithmeticException("Noinverse!"));
BigInteger result = ((p[0] - (p[1] * q[0])) %
modulus);
if ((result.data[maxLength-1] & 0x80000000)!=0)
    result += modulus;//get the least + modulus
return result;
}
```

By using this class, the writer application could construct more than 10-bit length integer key in short time below :

bit for *p* and *q*	Average Processing (s)	Times (x)
8	0	3
30	0	4
128	0,6	5
256	0,8	5
512	3,1	7

Our concentration is in the key generation because it is the main strength of RSA Algorithm and it is needed to improve its security.

IV.3. RSA Modifications

Until now on, RSA algorithm is still secure enough for being one of the signature schemes with appendix for new applications. There are 2 mentioned modified RSA from RSA Laboratory :[9]

RSAES-OAEP (RSA Encryption Scheme - Optimal Asymmetric Encryption Padding) is a public-key encryption scheme combining the RSA algorithm with the OAEP method.

RSASSA-PSS which is (RSA Signature Scheme with Appendix - Probabilistic Signature Scheme) an asymmetric signature scheme with appendix combining the RSA algorithm with the PSS encoding method.

RSA-SHA which is consists of 2 step encryption algorithm, first let generate the number of seed public and private key then you can give your sign to the message with the private key, then second step is to hash our message with SHA Algorithm so our signature could not be read as ussually as common signature which use only RSA.

V. CONCLUSION

RSA security algorithm is based on the difficulty of factoring large numbers into prime factors. It can be concluded that RSA is safe only if *n* is big enough.

RSA is much slower than symmetric key cryptography algorithms such as AES and DES.

There are 2 common weakness of RSA security :

Man-In-The-Middle Attack :

A man who infiltrate into the middle of someone communications and make a modifications or changes to the message sent by other people.

Chosen-plaintext Attack :

RSA is vulnerable to the Chosen-plaintext attack. Suppose if someone has some plaintext of the messages. He can select some of the plaintext to encrypted public key premises, then save the results in the dictionary. Then analys intercepting communications channels and to compare the GCC cipherteks intercepted by cipherteks in the dictionary. If there are similarities, then kriptanalis can use the dictionary to learn the contents of the message.

VII. ACKNOWLEDGMENT

In this part of paper, the writer would like to say thanks to :

1. God,
   For His kindness and blesses to the writer so this paper could be finished and released.
2. Sir Rinaldi Munir,
   For his Cryptography lecture until now and guidance to the writer about being a great cryptanalys.
3. Parents,
   For their hope and pray to our God, so the writer stand up until now and finish this paper.
4. Google Search Engine,
   For its help to the writer to find a lot of suitable webpage and references in English and Indonesian.
5. Friends,
   For always cheer the writer up and give some advices so this paper could be finished.

And others people where the writer could not mention each one of them. The writer was very glad to be a cryptanalys, and hope someday there will appear other famous cryptographers from Indonesia just like Adi Shamir or Clifford Cocks. Thanks and God Bless You.

REFERENCES

[1] Rivest, R.; A. Shamir; L. Adleman (1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Communications of the ACM 21W.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.

[2] SIAM News, Volume 36, Number 5, June 2003, "Still Guarding Secrets after Years of Attacks, RSA Earns Accolades for its Founders", by Sara RobinsonB. Smith, "An approach to graphs of linear forms (Unpublished work style)," unpublished.

[3] http://www.krellinst.org/UCES/archive/modules/charlie/pke/node10.html.

[4] technet.microsoft.com/en-us.

[5] http://icaferina.blogspot.com/2010/02/rsa-algorithm.html.

[6] http://crypto.stanford.edu/~dabo/pubs/papers/RSA-survey.pdf

[7] http://www.eecs.umich.edu/~taustin/papers/DATE10-rsa.pdf

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 15 Mei 2010

Galih Andana - 13507069

[9] http://www.rsa.com/rsalabs/node.asp?id=2146