STUDI, IMPLEMENTASI, DAN PERBANDINGAN ALGORITMA KUNCI PUBLIK NTRU DENGAN ALGORITMA KUNCI PUBLIK RSA

Kukuh Nasrul Wicaksono – NIM : 13505097

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15097@students.if.itb.ac.id

Abstrak

Pada saat ini, algoritma kunci publik populer untuk digunakan untuk menyandikan data pada lingungan digital. Hal tersebut karena faktor keamanan dan juga kemudahan implementasinya pada lingkungan digital. Algoritma kunci publik sendiri adalah algoritma penyandian dimana kunci untuk enkripsinya berbeda dengan kunci dekripsinya.

Salah satu algoritma kunci publik yang saat ini sering digunakan adalah algoritma kunci publik RSA. Algoritma ini sangat aman dan juga mudah untuk diimplementasikan. Meskipun algoritma ini sudah berumur lebih dari 30 tahun, namun algoritma ini masih tetap relevan untuk digunakan hingga saat ini. Selain algoritma kunci publik RSA, terdapat pula algoritma kunci publik lain, salah satunya adalah algoritma kunci publik NTRU. Algoritma kunci publik ini ditemukan pada tahun 1996. Oleh penemunya algoritma ini diklaim lebih baik dari algoritma kunci publik RSA.

Pada makalah ini akan dijelaskan mengenai algoritma kunci publik NTRU dan algoritma kunci publik RSA beserta dengan implementasinya. Selain itu penulis juga akan akan memperbandingkan kedua algoritma kunci publik tersebut dari data hasil implementasi maupun dari data yang telah ada. Dengan demikian diharapkan pembaca dapat lebih memahami kedua algoritma tersebut.

***Kata kunci**:* Kunci publik, kunci privat, kriptografi kunci publik, RSA, NTRU, enkripsi, dekripsi, polinom

1. PENDAHULUAN

Sebuah pesan yang dikirimkan dari pengirim ke penerima melalui suatu media transmisi pesan memiliki resiko untuk disadap oleh pihak yang tidak berkepentingan. Untuk menjaga kerahasiaan pesan, terdapat dua macam pendekatan yang umum digunakan, yaitu dengan menggunakan jalur komunikasi yang terjamin keamanannya, atau dengan menyandikan pesan dalam bentuk yang hanya dapat dibaca oleh pihak yang berkepentingan. Pendekatan yang banyak digunakan adalah pendekatan kedua, hal ini dikarenakan untuk menjaga keamanan jalur dari jalur komunikasi yang digunakan dibutuhkan biaya yang relatif cukup besar jika dibandingkan dengan menjaga kerahasiaan pesan dengan penyandian. Oleh karena itu penyandian pesan lebih sering digunakan untuk menjaga kerahasiaan pesan.

Untuk mencapai tujuan menjaga kerahasiaan dari pesan, algoritma penyandian menggunakan kunci rahasia dalam proses enkripsi dan dekripsi dari pesan. Kunci rahasia tersebut dioperasikan dengan pesan dan menghasilkan pesan yang telah dirahasiakan. Pesan yang telah dirahasiakan tersebut dapat dikirimkan melalui media transmisi yang tidak terjamin keamanannya karena tidak dapat dibaca oleh pihak yang tidak berkepentingan selama pihak tersebut tidak memiliki kunci rahasia untuk membaca pesan yang telah menjalani proses enkripsi.

Terdapat dua jenis algoritma penyandian (kriptografi) yang sering digunakan, yaitu algoritma kriptografi dengan kunci simetri dan algoritma kriptografi dengan kunci publik. Perbedaan antara kedua algoritma kriptografi tersebut adalah pemanfaatan kunci yang digunakan dalam proses enkripsi dan dekripsi. Pada kriptografi kunci simetri, kunci yang digunakan untuk proses pengenkripsian adalah

sama dengan kunci yang digunakan untuk proses pendekripsian. Hal ini menyebabkan kerahasiaan kunci yang digunakan dalam kriptografi kunci simetri menjadi sangat penting untuk dijaga. Sedangkan pada kriptografi kunci publik, kunci yang digunakan untuk proses pendekripsian berbeda dengan kunci yang digunakan untuk proses pengenkripsian. Hal ini menyebabkan kunci pengenkripsi atau kunci pendekripsian tidak harus dijaga kerahasiaannya dan dapat dipublikasikan dengan bebas. Pada beberapa jenis kunci publik, hanya kunci pengenkripsian yang boleh dipublikasikan seperti kunci publik NTRU dan RSA.

Pada makalah ini akan dibahas secara mendalam mengenai dua buah kunci publik yang disebutkan diatas yaitu kunci publik NTRU dan kunci publik RSA. Selain penulis juga akan mengimplementasikan dua buah kunci publik tersebut kemudian akan memperbandingkan kedua buah kunci publik tersebut melalui eksperimen dan studi literatur.

2. KONSEP DASAR

2.1 Algoritma NTRU

Algoritma NTRU merupakan algoritma kriptografi kunci publik yang diciptakan oleh 3 orang yaitu Jeffrey Hoffstein, Jill Pipher, dan Joseph Silverman. Oleh penciptanya algoritma ini diklaim lebih cepat, lebih mudah, dan tidak memerlukan memori yang besar dalam prosesnya.

Pada Tahun 1996, algoritma ini pertama kali diperkenalkan ke publik dalam sebuah event Crypto ’96. Perlu diperhatikan bahwa algoritma ini dipatenkan sehingga untuk penggunaan algoritma ini khususnya untuk tujuan komersil memerlukan izin dari pembuat algoritma ini. Karena alasan tersebut, algoritma ini jarang sekali dipakai pada lingkungan software dan juga jarang diekspose di lingkungan internet.

Dasar dari algoritma ini adalah sulitnya menemukan vektor yang singkat dari sebuah *lattice* (subgroup diskrit dari sebuah kumpulan vektor yang mencakup seluruh lingkungan vektor). Dalam prosesnya NTRU menggunakan operasi terhadap polinom. Sehingga dalam prakteknya pesan akan diubah dahulu menjadi bentuk polinom agar dapat diproses melalui sistem ini.

Truncated Polynomial Rings

Algoritma NTRU menggunakan prinsip-prinsip pada sebuah polinom dengan derajat N-1 seperti berikut.

$$a = a_0 + a_1X + a_2X^2 + a_3X^3 + ... + a_{N-1}X^{N-1}$$

Koefisien dari polinom tersebut ($a_0,a_1,...,a_{N-1}$) bernilai integer. Set dari sebuah polinomial ini dalam makalah ini akan disebut sebagai *R*. Polinom dalam *R* juga dapat dilakukan operasi pertambahan dan perkalian dengan polinom lain.

$$a+b = (a_0+b_0) + (a_1+b_1)\,X + ... + (a_{N-1}+b_{N-1})\,X^{N-1}$$

$$a*b = c_0 + c_1X + c_2X^2 + c_3X^3 + ... + c_{N-1}X^{N-1}$$

dimana c_k merupakan koefisien dari hasil perkalian dua polinom *a* dan *b* tadi.

Dengan menggunakan aturan penambahan dan perkalian tersebut maka semua aturan yang terkait dapat pula dilakukan, semisal aturan distribusi pun berlaku contoh :

$$a * (b + c) = a * b + a * c$$

Proses penambahan dan perkalian diatas akan membuat R menjadi sebuah cincin yang disebut *Ring of Truncated Polynomials*. Cincin tersebut bisa dijabarkan dengan persamaan cincin $R = Z[X]/(X^N - 1)$.

Algoritma NTRU ini menggunakan *ring of truncated polynomials R* digabung dengan aritmetika modulo. Dua hal tersebut digabungkan dengan melakukan operasi pengurangan koefisien polinom *a* dengan melakukan operasi modulo dengan integer *q*, atau dapat ditulis dengan rumus :

$$a \text{ (modulo } q)$$

Rumus diatas berarti bahwa dilakukan operasi modulo *q* terhadap koefisien dari *a*, sehingga membagi setiap koefisien dengan *q* dan mengambil sisanya. Dengan begitu operasi :

$$a = b \text{ (modulo } q)$$

berarti bahwa setiap koefisien adalah hasil dari *a* - *b* adalah kelipatan dari *q*.

Catatan :

Untuk membuat penyimpanan dan perhitungan lebih mudah, biasanya penyimpanan koefisien dari polinom akan disimpan tanpa penulisan variabel X atau disimpan sebagai list dari buah bilangan sebagai berikut :

$$a = (a_0, a_1, \ldots , a_{N-2}, a_{N-1})$$

Parameter NTRU

Parameter yang digunakan pada algoritma NTRU terdiri dari 3 buah nilai integer yaitu (*N*,*p*,*q*) dimana *N* adalah derajat polinom, kemudian *p* dan *q* adalah suatu bilangan integer dimana $q > p$ dan relatif prima satu sama lain.

Pembangkitan Kunci

Untuk membangkitkan pasangan kunci publik dan kunci privat NTRU maka diperlukan langkah-langkah sebagai berikut.

1. Pilih secara acak 2 buah polinom kecil *f* dan *g* yang merupakan anggota dari *R* (polinom *N*-1) seperti yang telah dijelaskan diatas. Isi dari polinom *f* dan *g* tersebut dirahasiakan. Polinom kecil adalah sebuah polinom yang relatif kecil terhadap polinom acak mod *q*. dalam sebuah polinom yang acak maka koefisien akan secara acak terdistribusi dalam mod *q*, sehingga dalam polinom kecil koefisien akan jauh lebih kecil dari *q*.
2. Kemudian cari nilai F_q dan F_p dengan menggunakan dua persamaan berikut
$$F_q * f \equiv 1 \pmod q$$
$$F_p * f \equiv 1 \pmod p$$
3. Kemudian hitung
$$h \equiv F_q * g \pmod q$$

Kunci publik yang akan digunakan untuk enkripsi adalah polinom *h* sedangkan kunci privat untuk dekripsi adalah polinom *f* dan F_p.

Proses Enkripsi Pesan

Dalam melakukan enkripsi pesan *M* maka dilakukan langkah-langkah sebagai berikut [1].

1. Ubah pesan *M* ke dalam bentuk polinom *m* (anggota polinom *N*-1) dengan koefisien yang dipilih adalah modulo *p* yang berada pada interval $-q/2$ hingga $q/2$. (*m* adalah polinom kecil dari mod *q*).
2. Pilih secara acak polinom kecil *r*.
3. Lakukan enkripsi menggunakan kunci publik *h* dengan persamaan sebagai berikut.
$$e = r * h + m \pmod q$$
polinom e merupakan pesan yang telah terenkripsi.

Proses Dekripsi Pesan

Setelah pesan terenkripsi *e* diterima, maka dekripsi dapat dilakukan dengan langkah berikut.

1. Hitung nilai *a* dengan menggunakan kunci privat *f* sebagai berikut.
$$a \equiv f * e \pmod q$$
dimana nilai polinom *a* berada pada interval $-q/2$ hingga $q/2$.
2. Dengan menggunakan polinom *a*, kita dapat memperoleh pesan *m* dengan persamaan sebagai berikut.
$$m = F_p * a \pmod p.$$

Implementasi Algoritma NTRU

Berikut ini adalah sebagian kode implementasi untuk pembangkitan kunci, enkripsi, dan dekripsi dengan menggunakan algoritma NTRU[7].

```
void KeyGenerator::GenerateKey_HighestSecurity(int *public_key, int
*private_key, int *Fp){
// N = 503, p = 3, q = 256
   int invertable=1;
   int * g = (int *)malloc(503*sizeof(int));
   int * Fq = (int *)malloc(503*sizeof(int));
   int * temp = (int *)malloc(503*sizeof(int));
   rpg.RandPoly(g,503,72,72,3);
   while (invertable){
      rpg.RandPoly(private_key,503,216,215);
      invertable = p.InversePoly_3(Fp,private_key,503);
      if(invertable == 0){
         invertable = p.InversePoly_2(temp, private_key, 503);
         if(invertable == 0){
            p.InversePoly_qr(Fq, private_key, temp, 8, 503);
         }
      }
   }
   p.Mult(public_key, Fq, g, 256,503);
}
void Encoder::Encrypt_HighestSecurity(int* ciphertext, int* plaintext, int*
public_key){
   int *r = (int *) malloc(503*sizeof(int));
   rpgen.RandPoly(r, 503, 55, 55);
   pol.Mult(ciphertext, public_key, r,256, 503);
   free(r);
   for(int i = 0; i < 503; i++ )
   *(ciphertext+i) = (*(ciphertext+i) + *(plaintext+i)) & 255;
```

```
}

void Decoder::Decrypt_HighestSecurity(int* plaintext, int* ciphertext, int*
private_key, int* Fp)
{
    int * a = (int *) malloc(503*sizeof(int));
    po.Mult(a, private_ key, ciphertext,256, 503);
    for(int i = 0; i < 503; i++){
        if(*(a+i) < 0)  *(a+i) = *(a+i) + 256;
        if(*(a+i) > 128)  *(a+i) = *(a+i) - 256;
    }
    po.Mult(plaintext, Fp, a, 3, 503);
    free(a);
}
```

Kekuatan dan Keamanan NTRU

Kekuatan dari algoritma NTRU ini adalah sulitnya menemukan vektor yang singkat dari sebuah *lattice* (subgroup diskrit dari sebuah kumpulan vektor yang mencakup seluruh lingkungan vektor) dari sebuah polinom acak yang memiliki derajat yang besar. Pada NTRU ini terdapat 3 tingkat keamanan. Ketiga tingkat keamanan tersebut berbeda pada nilai parameter-parameter NTRU nya. Tabel berikut menggambarkan nilai parameter yang digunakan untuk masing-masing tingkat keamanannya.

Tingkat Keamanan	*N*	*p*	*q*
Menengah	107	3	64
Tinggi	167	3	128
Tertinggi	503	3	256

Tabel 1 Tingkat keamanan NTRU

Usaha yang dibutuhkan untuk membobol algoritma NTRU ini juga berbeda-beda untuk tiap tingkat keamanan yang digunakan. Berdasarkan [2] dan [3], usaha yang perlu dilakukan untuk membobol algoritma ini dengan algoritma yang paling cepat saat ini akan membutuhkan waktu sebagai berikut.

Tingkat Keamanan	Waktu
Menengah	780230 detik (9 Hari)
Tinggi	1198.10^{10} detik (380 tahun)
Tertinggi	1969.10^{35} ($6{,}2.10^{27}$ tahun)

Tabel 2 Waktu untuk pembobolan NTRU

Dengan menggunakan algoritma ini untuk mengamankan data, khususnya dengan menggunakan tingkat keamanan tinggi atau tertinggi dapat dijamin sulit untuk dibuka paksa olah orang yang tidak bertanggung jawab karena waktu yang dibutuhkan lebih dari 300 tahun. Terlebih lagi jika menggunakan tingkat keamanan tertinggi, waktu yang dibutuhkan untuk membuka paksa memerlukan waktu $6{,}2.10^{27}$ tahun.

2.2 Algoritma RSA (Rivest, Shamir, Adleman)

Algoritma RSA diperkenalkan oleh tiga peneliti dari MIT (Massachussets Institute of Technology), yaitu Ron Rivest, Adi Shamir, dan Len Adleman pada tahun 1976. Algoritma RSA ini merupakan salah satu yang paling maju dalam bidang kriptografi public key. RSA masih digunakan secara luas dalam protokol *electronic commerce*, dan dipercaya dalam mengamnkan dengan menggunakan kunci yang cukup panjang.

RSA mendasarkan proses enkripsi dan dekripsinya pada konsep bilangan prima dan aritmatika modulo. Baik kunci enkripsi maupun kunci dekripsinya keduanya berupa bilangan bulat. Kunci enkripsi tidak dirahasiakan dan diketahui umum (dinamakan kunci publik), sedangkan kunci dekripsinya dirahasiakan (kunci privat). Untuk menemukan kunci dekripsi, orang harus memfaktorkan suatu bilangan non prima menjadi faktor primanya. Kenyataannya, memfaktorkan bilangan non prima menjadi faktor primanya bukanlah pekerjaan yang mudah. Belum ada algoritma yang efisien yang ditemukan untuk pemfaktoran prima ini. Semakin besar bilangan primanya, tentu semakin sulit pula pemfaktorannya. Semakin sulit pemfaktorannya, semakin kuat pula algoritma RSA. Algoritma RSA sebenarnya sederhana sekali. Secara ringkas, algoritma RSA adalah sebagai berikut.

Proses Pembangkitan Kunci

Sebagai contoh Alice berkeinginan untuk mengizinkan Bob mengirimkan sebuah pesan pribadi (private message) kepadanya melalui media transmisi yang tidak aman (insecure). Alice melakukan langkah-langkah berikut untuk membuat pasangan kunci publik dan kunci privat:

1. Pilih dua bilangan prima p dan q secara acak dimana $p \neq q$. Kemudian hitung $n = pq$. n hasil perkalian dari p dikalikan dengan q. Nilai p dan q tersebut dirahasiakan, sedangkan n tidak dirahasiakan.
2. Hitung $\varphi = (p - 1)(q - 1)$.
3. Pilih bilangan bulat atau integer untuk kunci publik e antara satu dan φ ($1 < e < \varphi$) yang juga relatif prima terhadap φ.
4. Bangkitkan kunci dekripsi d dengan kekongruenan $ed \equiv 1 \pmod{\varphi}$.

Nilai pada kunci publik terdiri atas:

1. *n*, modulus yang digunakan.
2. *e*, eksponen publik (sering juga disebut eksponen enkripsi).

Nilai pada kunci privat terdiri atas:

1. *n*, modulus yang digunakan, digunakan pula pada kunci publik.
2. *d*, eksponen privat (sering juga disebut eksponen dekripsi), yang harus dijaga kerahasiaannya.

Setelah kunci dibangkitkan, Alice akan mengirimkan kunci publik kepada Bob untuk proses enkripsi pesan, dan tetap merahasiakan kunci privat yang nantinya akan digunakan untuk mendekripsi pesan yang telah dienkripsi oleh Bob dengan kunci publik yang telah dikirimkan sebelumnya. Selain itu nilai *p* dan *q* juga harus dirahasiakan.

Proses Enkripsi Pesan

Misalkan Bob ingin mengirim pesan *M* ke Alice, Bob harus mengubah karakter-karakter dalam pesan *M* menjadi angka $m < n$ dimana *m* biasanya merupakan bilangan ascii dari tiap karakter pesan tersebut atau dengan menggunakan protokol yang sebelumnya telah disepakati sebelumnya yang dikenal sebagai *padding scheme*.

Setelah pesan *M* diubah menjadi angka *m*, Bob dapat mengenkripsi pesan tersebut dengan menggunakan pasangan kunci publik yang terdiri dari *n* dan *e*, yang sebelumnya telah diumumkan oleh Alice. Bob kemudian menghitung cipherteks *c* yang dari *m* dengan menggunakan persamaan sebagai berikut.

$$c = m^e \bmod n$$

Perhitungan tersebut dapat diselesaikan dengan cepat menggunakan metode *exponentiation by squaring*. Setelah cipherteks *c* didapat, Bob mengirimkan *c* kepada Alice.

Proses Dekripsi Pesan

Setelah Alice menerima cipherteks *c* dari Bob, Alice dapat mendekripsi cipherteks tersebut dengan menggunakan kunci privat yang telah dirahasiakan tadi. Alice dapat mendapatkan nilai *m* dari *c* dengan menggunakan persamaan sebagai berikut.

$$m = c^d \bmod n$$

Perhitungan diatas akan menghasilkan nilai-nilai *m*. Dengan menggunakan protokol atau aturan yang sebelumnya telah digunakan untuk mengubah *M* menjadi *m* pada proses enkripsi, Alice dapat mengembalikan pesan *M* seperti semula dari nilai *m*.

Kekuatan dan Keamanan RSA

Seperti yang telah dikatakan sebelumnya bahwa kekuatan RSA terletak pada tingkat kesulitan dalam memfaktorkan bilangan non prima menjadi faktor primanya, yang dalam hal ini $n = pq$. Sekali *N* berhasil difaktorkan menjadi nilai *p* dan *q* maka $\varphi = (p - 1)(q - 1)$ dapat dihitung. Selanjutnya, karena kunci enkripsi *e* diumumkan (tidak rahasia), maka kunci dekripsi *d* yang merupakan kunci dekripsi dan bersifat rahasia dapat dihitung dengan menggunakan persamaan $ed \equiv 1 \pmod{\varphi}$. Ini berarti proses dekripsi menjadi tidak rahasia dan dapat dilakukan oleh orang lain yang tidak berhak.

Pada penggunaan algoritma RSA ini disarankan untuk menggunakan nilai *p* dan *q* yang memiliki panjang lebih dari 100 digit. Hal tersebut akan menjadikan $n = pq$ memiliki nilai lebih dari 200 digit. Nilai *n* yang sangat besar tersebut sangat sulit untuk dicari faktor primanya. Dengan menggunakan komputer sekalipun, usaha yang dibutuhkan akan sangat besar untuk mencari faktor prima dari bilangan yang memiliki 200 digit nilai. Usaha yang dibutuhkan untuk mencari faktor prima bilangan 200 digit membutuhkan waktu 4 miliyar tahun (dengan asumsi bahwa algoritma pemfaktoran yang digunakan adalah algoritma tercepat saat ini dan komputer yang dipakai mempunyai kecepatan 1 milidetik) [5].

Belum ditemukannya algoritma yang efektif dan efisien untuk mencari faktor prima dari bilangan bulat yang sangat besar membuat algoritma RSA ini tetap dipakai dan relevan hingga saat ini. Selama belum ditemukan algoritma yang efisien untuk mencari faktor prima dari bilangan bulat besar, maka algoritma RSA ini sangat direkomendasikan untuk penyandian pesan.

Implementasi Algoritma RSA

Berikut ini adalah kode dari implementasi pembangkitan kunci untuk algoritma RSA dan juga untuk enkripsi dan dekripsi dengan menggunakan algoritma RSA.

```
/*
 * Copyright (C) 2009 IF, STEI, ITB
 */
package tandatangan.key;
import java.math.BigInteger;
import java.security.interfaces.RSAPrivateCrtKey;
import java.security.interfaces.RSAPublicKey;
import java.security.KeyPair;
import java.security.SecureRandom;
import tandatangan.util.RSAAlgorithm;
/**
 *
 * @author egoz at ymail dot com
 * @author myunx_ong at yahoo dot com
 * @author persada_adi at yahoo dot com
 */
public class RSAKeyPairGenerator {

// Fields and Constants
// ..........................................................................
   private static final BigInteger
      ONE = BigInteger.valueOf(0x1),
      F4  = BigInteger.valueOf(0x10001L);
   private static final int CERTAINTY = 80;
   SecureRandom random = new SecureRandom();

// Public Methods
// ..........................................................................
   public KeyPair generateWithLength(int length){
      int pLen = length / 2;
      int qLen = length - pLen;
      BigInteger d, e, n, p, q, pMinus1, qMinus1, phi;
      e = F4;
      while(true)
      {
         try
         {
            do
            {
               p = new BigInteger(pLen, CERTAINTY, this.random);
               q = new BigInteger(qLen, CERTAINTY, this.random);
               n = p.multiply(q);
            }
            while( (p.compareTo(q) == 0) || (n.bitLength() != length) );
            pMinus1 = p.subtract(ONE);
            qMinus1 = q.subtract(ONE);
            phi     = pMinus1.multiply(qMinus1);
            d       = e.modInverse(phi);
            break;
         }
         catch(ArithmeticException ae)
         {
            // gcd(e * phi) != 1
         }
      }
      BigInteger primeExponentP = d.mod(pMinus1);
      BigInteger primeExponentQ = d.mod(qMinus1);
      BigInteger crtCoefficient = q.modInverse(p);
      BigInteger x = new BigInteger(pLen, this.random);
      BigInteger y = RSAAlgorithm.rsa(x,
                              n,
                              e);
      BigInteger z = RSAAlgorithm.rsa(y,
                              n,
                              d,
                              p,
                              q,
                              primeExponentP,
                              primeExponentQ,
                              crtCoefficient);
      if( !z.equals(x) )
         throw new RuntimeException("RSA KeyPair doesn't work");
      RSAPrivateCrtKey priv = new RSAPrivateCrtKeyImpl(
                              n,
                              e,
                              d,
                              p,
                              q,
                              primeExponentP,
                              primeExponentQ,
                              crtCoefficient);
      RSAPublicKey    pub  = new RSAPublicKeyImpl(n, e);
      return new KeyPair(pub, priv);
   }
}

/*
 * Copyright (C) 2009 IF, STEI, ITB
 */
package tandatangan.util;
import java.math.BigInteger;
/**
 *
 * @author egoz at ymail dot com
 * @author myunx_ong at yahoo dot com
 * @author persada_adi at yahoo dot com
 */
public class RSAAlgorithm {

// Static Fields
//..........................................................................
   private static final BigInteger ONE = BigInteger.valueOf(1L);

// Static Methods
//..........................................................................

   public static BigInteger rsa(BigInteger X, BigInteger n, BigInteger exp,
                        BigInteger p, BigInteger q, BigInteger u)
   {
      if (p == null)
         return rsa(X, n, exp);
      BigInteger primeExponentP = exp.mod(p.subtract(ONE));
      BigInteger primeExponentQ = exp.mod(q.subtract(ONE));
      return rsa(X, n, exp, p, q, primeExponentP, primeExponentQ, u);
   }

   public static BigInteger rsa(BigInteger X,
                        BigInteger modulus,
                        BigInteger exp,
                        BigInteger primeP,
                        BigInteger primeQ,
                        BigInteger primeExponentP,
                        BigInteger primeExponentQ,
                        BigInteger crtCoefficient)
   {
      if( !crtCoefficient.equals(primeQ.modInverse(primeP)) )
      {
         BigInteger t;
         t = primeQ; primeQ = primeP; primeP = t;
         t = primeExponentQ;
         primeExponentQ = primeExponentP;
         primeExponentP = t;
      }
      BigInteger p2 = X.mod(primeP).modPow(primeExponentP, primeP);
      BigInteger q2 = X.mod(primeQ).modPow(primeExponentQ, primeQ);
      if (p2.equals(q2))
         return q2;
      BigInteger k = (p2.subtract(q2).mod(primeP));
      BigInteger l = k.multiply(crtCoefficient).mod(primeP);
      return primeQ.multiply(l).add(q2);
   }


public static BigInteger rsa(BigInteger X, BigInteger n, BigInteger exp)
   {
      return X.modPow(exp, n);
   }
}

/*
 * Copyright (C) 2009 IF, STEI, ITB
 */
package tandatangan.signature;
import java.math.BigInteger;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
```

```
import java.security.InvalidKeyException;
import tandatangan.util.RSAAlgorithm;
/**
 *
 * @author egoz at ymail dot com
 * @author myunx_ong at yahoo dot com
 * @author persada_adi at yahoo dot com
 */
public class RSA {

// Constructors
// ..............................................................................
   public RSA() {
   }

// Implements
// ..............................................................................

   protected byte[] Encrypt(byte[] plain, PublicKey publicKey)
    throws InvalidKeyException {
      if ( !(publicKey instanceof RSAPublicKey) )
         throw new InvalidKeyException("Key isnt RSA public key");
      BigInteger m = new BigInteger(1, plain);
      RSAPublicKey priv = (RSAPublicKey) publicKey;
      BigInteger n = priv.getModulus();
      BigInteger e = priv.getPublicExponent();
      BigInteger cipher = RSAAlgorithm.rsa(m, n, e);
      return cipher.toByteArray();
   }

   protected byte[] Decrypt(byte[] cipher, PrivateKey privateKey)
    throws InvalidKeyException {
      if ( !(privateKey instanceof RSAPrivateKey) )
         throw new InvalidKeyException("Key isnt RSA private key");
      BigInteger c = new BigInteger(1, cipher);
      RSAPrivateKey priv = (RSAPrivateKey) privateKey;
      BigInteger n = priv.getModulus();
      BigInteger d = priv.getPrivateExponent();
      BigInteger plain = RSAAlgorithm.rsa(c, n, d);
      return plain.toByteArray();
   }
}
```

3. PERBANDINGAN ALGORITMA NTRU DENGAN ALGORTIMA RSA

Jika dilihat dari tingkat keamanannya, algoritma ini sebenarnya sulit untuk dibandingkan karena perbedaaan dasar algoritma yang digunakan. Persamaan dari algoritma ini hanyalah penggunaan modulusnya saja. Namun begitu, tingkat keamanan kedua algoritma ini telah terbukti handal jika kita melihat waktu yang dibutuhkan untuk memecahkan algoritma ini dengan paksa. Perbandingan yang selanjutnya akan dilakukan adalah perbandingan kecepatan dari kedua jenis algoritma ini pada pembangkitan kunci, enkripsi, dan dekripsinya.

Jika dilihat dari sumber [2], data kecepatan pada proses pembangkitan kunci, enkripsi dan dekripsi dari kedua algoritma ini adalah sebagai berikut.

Tingkat Keamanan	Enkripsi (blok/detik)	Dekripsi (blok/detik)	Pembangkitan kunci (detik)
512 bit	370	42	0.45
768 bit	189	15	1.5
1024 bit	116	7	3.8

Tabel 3 Kecepatan RSA pada 90 MHz Pentium

Tingkat Keamanan	Enkripsi (blok/detik)	Dekripsi (blok/detik)	Pembangkitan kunci (detik)
Menengah	1818	505	0.1080
Tinggi	649	164	0.1555
Tertinggi	103	19	0.8571

Tabel 4 Kecepatan NTRU pada 75 MHz Pentium

Dari data pada tabel diatas terlihat bahwa kecepatan algoritma NTRU pada proses enkripsi, dekripsi dan pembangkitan kuncinya lebih tinggi daripada algoritma RSA pada semua tingkat keamanan meskipun prosesor yang digunakan untuk algoritma RSA sedikit lebih tinggi dari yang digunakan untuk algoritma NTRU. Pada tingkat keamanan tertinggi, kecepatan algoritma NTRU pada proses enkripsi relatif sama dengan algoritma RSA dengan 1024 bit. Pada proses dekripsi, NTRU lebih cepat 2,7 kali dari RSA. Dan pada proses pembangkitan kunci NTRU lebih cepat 4.4 kali dari RSA. Hal tersebut sudah cukup membuktikan bahwa NTRU lebih baik dari RSA untuk masalah kecepatannya.

Selain memperbandingkan kecepatan kedua algoritma kunci publik dari data sumber [2], penulis juga memperbandingkan kedua program penyandi algoritma tersebut secara langsung. Pada tabel berikut dituliskan data kecepatan program penyandi RSA yang dibuat penulis, dan juga data kecepatan penyandi NTRU. Data kecepatan NTRU diperoleh dengan menggunakan kakas penghitung kecepatan penyandi dengan NTRU dari sumber [7]. Program dijalankan pada Intel Core 2 Duo 1,66 GHz.

Tingkat Keamanan	Enkripsi (detik)	Dekripsi (detik)	Pembangkitan kunci (detik)
1024 bit	0.004300	0.022900	0.359300

Tabel 5 Kecepatan RSA

Tingkat Keamanan	Enkripsi (detik)	Dekripsi (detik)	Pembangkitan kunci (detik)
Tertinggi	0.002420	0.004870	0.058500

Tabel 6 Kecepatan NTRU

Dari data percobaan diatas dapat kita lihat juga bahwa algoritma NTRU memiliki kecepatan yang lebih baik daripada algoritma RSA. Pada data tersebut, kecepatan NTRU pada proses enkripsi lebih cepat 1,7 kali daripada algoritma RSA. Pada proses dekripsi, NTRU lebih cepat 4.7 kali, sedangkan pada proses pembangkitan kunci algoritma NTRU lebih cepat 6 kali daripada algoritma RSA.

Pada percobaan ini kita melihat bahwa perbandingan kecepatan kedua algoritma tersebut berbeda dengan perbandingan kecepatan dari data pertama yang diperoleh dari sumber [2]. Hal ini kemungkinan dikarenakan program penyandi dengan algoritma RSA penulis tidak seefektif program RSA yang digunakan oleh sumber [2] pada percobaannya. Karena itu perbedaan kecepatan NTRU dan RSA menjadi lebih tinggi dari data yang diperlihatkan pembuat algoritma NTRU yang sebenarnya. Namun secara garis besar, algoritma NTRU memang memiliki kecepatan yang lebih baik dibandingkan dengan algoritma RSA dengan melihat hasil dari percobaan secara langsung ini.

4. PENUTUP

Setelah melakukan studi, implementasi, dan perbandingan dari algoritma NTRU dan RSA penulis dapat menyimpulkan hal-hal sebagai berikut.

1. Algoritma kunci publik NTRU dan RSA memiliki tingkat keamanan yang relatif sama. Hal ini dapat dilihat dari waktu yang dibutuhkan untuk membobol kedua algoritma ini yang sangat lama (milyaran tahun) dengan menggunkan algoritma tercepata yang dapat digunakan untuk masing-masing algoritma kunci publik tersebut.
2. Jika dilihat dari segi kecepatannya, algoritma kunci publik NTRU memiliki kecepatan yang lebih tinggi daripada algoritma kunci publik RSA pada proses pembangkitan kunci, proses enkripsi, maupun proses dekripsinya.
3. Meskipun algoritma NTRU lebih baik daripada algoritma RSA, algoritma NTRU ini tidak sepopuler algoritma RSA. Hal ini disebabkan karena algoritma NTRU ini dipatenkan. Dengan begitu tidak ada produk-produk open source yang menggunakan algoritma ini untuk proses penyandian datanya.
4. Algoritma NTRU sangat cocok digunakan untuk sistem keamanan nirkabel. Hal ini disebabkan karena algoritma NTRU yang cepat dan menghemat data yang dibutuhkan untuk pengiriman karena pesan dikirimkan dalam bentuk variabel polinom yang lebih menghemat tempat berdasarkan klaim yang disampaikan NTRU CryptoSystem.

DAFTAR REFERENSI

[1] J. Hoffstein, J.Pipher, J.H.Silverman. 1998. NTRU: A New High Speed Public Key Cryptosystem, Algorithmic Number Theory (ANTS III),Portland.

[2] J. Hoffstein, J.Pipher, J.H.Silverman. 1998. NTRU: A Ring-Based Public Key Cryptosystem.

[3] J. Hoffstein, J.H.Silverman, W. Whyte. 1998. NTRU: Estimated Breaking Times for NTRU Lattices.

[4] Munir, Rinaldi. 2004. Bahan Kuliah IF5054 Kriptografi. Departemen Teknik Informatika, Institut Teknologi Bandung.

[5] Munir, Rinaldi. 2006. Bahan Kuliah IF2153 Matematika Diskrit, Institut Teknologi Bandung.

[6] Penalosa, Ronald A. Studi Sistem Kriptografi Kunci Publik NTRU, Institut Teknologi Bandung.

[7] http://islab.oregonstate.edu/koc/ece575/03Project/Aciicmez/ diakses pada selasa, 18 Mei 2009 pukul 12.00 WIB