Analisis Perbandingan dan Pengujian Algoritma Kunci Publik RSA dan Paillier

I.Y.B. Aditya Eka Prabawa W. – 13507034
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
aditya_eka@students.itb.ac.id

Makalah ini membahas algoritma kriptografi kunci publik, yaitu Algoritma RSA dan analisis perbandingannya dengan Algoritma Paillier. Kedua algoritma kunci publik tersebut sama-sama menggunakan 2 buah bilangan prima yang besar sebagai parameter awal. Namun demikian, keduanya memiliki skema pembangkitan kunci, skema enkripsi, dan skema dekripsi yang berbeda. Kekuatan RSA terletak pada sulitnya memfaktorkan bilangan besar, sedangkan Algoritma Paillier memiliki decisional composite residuosity assumption sebagai kekuatan utamanya.

Dalam makalah ini dibahas secara detail skema algoritma RSA dan Paillier, properti-properti yang khusus dan unik pada masing-masing algoritma, serta analisis perbandingan antara keduanya.

Untuk mendukung analisis dan pengujian Algoritma RSA dan Paillier, telah dibuat sebuah program kecil yang mengaplikasikan kedua algoritma tersebut untuk melakukan beberapa eksperimen, seperti eksprimen perbandingan kecepatan (perbandingan kecepatan pembangkitan kunci, perbandingan kecepatan enkripsi, dan perbandingan kecepatan dekripsi) dan eksperimen modifikasi pada cipherteks. Untuk setiap hasil eksperimen akan diberikan analisisnya.

Selain itu juga dibahas kekuatan, kelemahan, dan jenis-jenis serangan (attack) yang umum pada Algoritma RSA dan Algoritma Paillier, serta aplikasi dan penerapan dari kedua algoritma tersebut pada berbagai sistem keamanan di dunia nyata.

Kata Kunci: Kunci Privat, Kunci Publik, Paillier, RSA.

I. PENDAHULUAN

Sampai pertengahan 1970-an, hanya ada sistem kriptografi kunci simeteri. Satu masalah besar dalam sistem kriptografi kunci simetri adalah pengiriman kunci rahasia kepada penerima pesan sulit untuk dilakukan dan tidak aman [1]. Sistem kriptografi kunci publik dipublikasikan pertama kali pada tahun 1976 oleh Whitfield Diffie dan Martin Hellman. Sejak saat itu, telah banyak dikembangkan berbagai algoritma kriptografii kunci publik untuk berbagai bidang kriptografi.

Terdapat tiga jenis utama kriptografi kunci publik yang berbasis pada teori bilangan komputasional. Jenis yang pertama termasuk RSA dan variannya (Rabin-Williams, LUC, Dickson, KMOV). Kekuatan kriptografi kunci publik jenis ini terletak pada sulitnya memfaktorkan bilangan bulat yang sangat besar menjadi faktor-faktor primanya. Semakin besar bilangannya, semakin sulit untuk memfaktorkan.

Jenis kriptografi kunci publik yang kedua berbasis pada skema algoritma Diffie-Hellman (El-Gamal dan variannya, DSA, McCurley, serta Cramer Shoup) yang menggunkan prinsip logaritma diskrit. Kekuatan kriptografi kunci publik jenis ini terletak pada sulitnya menghitung nilai x dari $a^x \equiv b \pmod{n}$. Semakin besar nilai a, b, dan n, semakin sulit untuk memfaktorkan.

Jenis yang terakhir dari kriptografi kunci publik yang ada berbasis pada *high degree residuosity class* (Paillier, Goldwasser-Micali, Benaloh, Naccache-Stern, Okamoto-Uchiyama dan variannya, Vanstone-Zuccherato, dan Park-Won) [4]. Kunci kekuatan pada kriptografi kunci publik dengan skema ini terletak pada kombinasi ekstraksi *residuosity classes* dari grup tertentu dengan sulitnya menghitung ordenya. Karena *residuosity classes* bersifat *additive*, kriptografi kunci publik jenis ini mirip seperti kriptografi kunci publik yang berbasis pada logaritma diskrit, tetapi kekuatannya lebih mirip dengan kriptografi kunci publik yang mengandalkan sulitnya memfaktorkan bilangan bulat besar [5].

Algoritma kriptografi kunci publik jenis pertama yang paling populer adalah algoritma RSA (paling populer di antara semua algoritma kriptografi kunci publik yang pernah dibuat). Sedangkan kriptografi kunci publik berbasis *residuosity class* yang populer, salah satunya adalah algoritma Paillier.

II. SEJARAH SINGKAT KEDUA ALGORITMA

A. Sejarah Algoritma RSA

Algoritma RSA dideskripsikan secara publik pada tahun 1978 oleh Ronald Lynn Rivest (Perancang algoritma cipher aliran RC4), Adi Shamir, dan Leonard M. Adleman, ketiganya dari Massachusetts Institute of Technology (MIT). Paper berisi deskripsi algotitma RSA yang mereka tulis berjudul "*A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*" [7]. Singkatan RSA sendiri berasal dari inisial nama belakang mereka, terurut sesuai dengan urutan yang ditulis pada paper yang mereka tulis untuk mempublikasikan algortima ini [8].

B. Sejarah Algoritma Paillier

Algoritma Kunci Publik Paillier dikemukakan oleh Pascal Paillier pada tahun 1999. Pascal Pailler adalah seorang peneliti dari Departemen Sains Komputer di ENST, Prancis. Walaupun algoritma ini tergolong masih muda dibandingkan dengan algoritma-algoritma kriptografi lainnya, algoritma Paillier telah diakui dengan baik dan digunakan dalam berbagai kegunaan.

III. ALGORITMA RSA

A. Pembangkitan Kunci

Untuk melakukan enkripsi dan dekripsi, algoritma RSA memerlukan kunci publik dan kunci privat. Kunci publik, seperti namanya, dapat diketahui oleh publik dan digunakan untuk mengenkripsi pesan. Pesan yang dienkripsi dengan kunci publik hanya dapat didekripsi dengan menggunakan kunci privat padanannya yang tentunya bersifat rahasia. Kunci publik dan kunci privat yang digunakan dalam algoritma RSA dibangkitkan dengan mekanisme sebagai berikut:

1. Memilih dua buah bilangan prima berukuran besar sembarang yang bersifat rahasia, secara umum disebut p dan q.
2. Menghitung nilai n, dimana n adalah hasil perkalian antara nilai p dan nilai q ($n = p \cdot q$).
3. Menghitung nilai $\phi(n) = (p-1)(q-1)$.
4. Memilih kunci publik yang akan digunakan untuk enkripsi, secara umum disebut e, yang relatif prima terhadap nilai $\phi(n)$ yang telah dihitung sebelumnya.
5. Membangkitkan kunci privat yang akan digunakan untuk dekripsi, menggunakan persamaan:

$$e \cdot d \equiv 1 \pmod{\phi(n)}. \quad (1)$$

Terdapat beberapa hal yang perlu diperhatikan dalam algoritma pembangkitan pasangan kunci publik dan kunci privat tersebut, antara lain:

1. Nilai bilangan prima p dan q sebaiknya tidak sama, sebab jika p sama dengan q, maka $n = p^2$. Dengan demikian p dapat diperoleh dengan menarik akar pangkat dua dari n. Hal ini tentunya membuat algoritma RSA menjadi tidak berguna. Selain itu panjang nilai p dan q disarankan lebih dari 100 digit. Dengan demikian hasil kalinya (nilai n) akan berukuran lebih dari 200 digit.
2. Nilai n tidak perlu dan tidak dapat dirahasiakan. Sedangakan nilai $\phi(n)$ bersifat rahasia.
3. Persamaan yang digunakan untuk membangkitkan kunci privat ($e \cdot d \equiv 1 \pmod{\phi(n)}$), ekivalen dengan $e \cdot d \equiv 1 + k\phi(n)$, sehingga dengan demikian d dapat dihitung menggunakan:

$$d = \frac{1+k\phi(n)}{e} \quad (2)$$

dengan mencoba bilangan bulat k yang mungkin.

Hasil dari algortima pembangkitan pasangan kunci di atas adalah sebuah kunci publik yang merupakan pasangan nilai e dan n, serta kunci privat yang merupakan pasangan antara nilai d dan n [1].

Kode yang merepresentasikan algoritma pembangkitan kunci RSA diberikan dalam bahasa C# .NET sebagai berikut:

```
BigInteger n = p * q;
BigInteger phin = (p - 1) * (q - 1);
BigInteger e = new Random();

//mencari kunci publik e dari bilangan
//random sepanjang 128 bit yang
//relatif prima dengan nilai phin.
do
{
      e = e.genCoPrime(128, new
      Random());
}
while (!(phin.gcd(e) == 1));

//mencari kunci privat d sesuai rumus
//dengan k = 0, 1, 2, ...
//hasil d harus bilangan bulat.
int k = 1;
while ((1 + k * phin) % e != 0)
{
      k++;
}
d = (1 + k * phin) / e;
```

B. Enkripsi

Mekanisme enkripsi algoritma RSA jauh lebih sederhana daripada algoritma pembangkitan kuncinya. Pada prinsipnya enkripsi RSA hanya melibatkan dua operasi aritmatika, yaitu operasi perpangkatan dan operasi modulo. Algoritma enkripsi RSA secara lengkap adalah sebagai berikut:

1. Mengambil kunci publik penerima pesan, yang secara umum disebut e, dan nilai n.
2. Plainteks yang akan dienkripsi dipecah-pecah menjadi blok-blok yang lebih kecil.
3. Setiap blok hasil pemecahan plainteks dienkripsi menjadi satu blok cipherteks dengan rumus sederhana berikut:

$$c_i = m_i^e \bmod n \quad (3)$$

dengan c_i adalah blok cipherteks hasil enkripsi, m_i adalah blok plainteks yang dienkripsi, e adalah kunci publik, dan n adalah nilai n itu sendiri.

Pada proses enkripsi ini juga terdapat beberapa hal yang perlu diperhatikan, yaitu pemecahan plainteks

menjadi blok-blok dilakukan sedemikan sehingga setiap blok merepresentasikan nilai dalam selang antara 0 sampai $n - 1$ [1].

Kode yang merepresentasikan algoritma enkripsi RSA diberikan dalam bahasa C# .NET sebagai berikut:

```
//message adalah String plainteks.
//String message diubah menjadi larik
//char untuk diambil kode ASCII-nya.
//rangkaian kode ASCII tersebut lalu
//disusun jadi String angka panjang.
String message;
char[] charArray =
message.ToCharArray();
String plain;
for (int x = 0; x < charArray.Length;
x++)
{
      plain +=
      ((byte)charArray(x)).ToString();
}

//String angka plainteks dipecah jadi
//blok2 String dengan panjang n - 1.
//sehingga dihasilkan mi.
int blockSize = n.ToString().Length -
1;
int miLength =
(int)Math.Ceiling((double)plain.Length
/ (double)blockSize);

String[] mi = new String[miLength];
int i = 0;
int j;
while (i < plain.Length)
{
      j = 0;
      while (j < blockSize && i <
      plain.Length)
      {
            mi[(i - (i % blockSize)) /
            blockSize] +=
            plain.Substring(i, 1);
            j++; i++;
      }
}

//setiap blok mi dienkripsi
//sehingga dihasilkan ci
//lalu ci dijadikan String panjang
BigInteger[] ci = new
BigInteger[miLength];
String cipher = "";
for (int k = 0; k < miLength; k++)
{
      ci[k] = (new BigInteger(mi[k],
      10)).modPow(publicKey, n);
      cipher += ci[k].ToString();
      cipher += " ";
}
```

C. Dekripsi

Mekanisme dekripsi RSA serupa dengan enkripsinya. Pada dekripsi setiap blok cipherteks didekripsi kembali menjadi blok plainteks dengan rumus yang serupa, hanya saja, perhitungan aritmatikanya tidak menggunakan kunci publik melainkan kunci privat:

$$m_i = c_i^d \bmod n \tag{4}$$

dengan m_i adalah blok plainteks yang didapatkan kembali setelah dekripsi, c_i adalah blok cipherteks yang dideksripsi, dan n adalah nilai n itu sendiri [1].

Jelas terlihat bahwa proses enkripsi dan dekripsi algoritma RSA sangatlah sederhana, proses pembangkitan kuncinya pun boleh dibilang sangat sederhana. Karena memang, kekuatan algoritma RSA bukan pada kerumitan proses pembangkitan kunci ataupun proses enkripsinya.

Kode yang merepresentasikan algoritma dekripsi RSA diberikan dalam bahasa C# .NET sebagai berikut:

```
//String angka cipherteks dipecah jadi
//blok2 String dengan panjang n - 1.
//sehingga dihasilkan si.
String cipher;
int blocksize = cipher.IndexOf(' ');
int cilength = cipher.Length /
(blocksize + 1);

String[] si = new String[cilength];
si = cipher.Split(new Char[] { ' ' });

BigInteger[] ci = new
BigInteger[cilength];
for (int i = 0; i < cilength; i++)
{
      ci[i] = new BigInteger(si[i],
      10);
}

//setiap blok ci didekripsi
//sehingga dihasilkan di
//lalu di dijadikan String panjang
BigInteger[] di = new
BigInteger[cilength];
String plain = "";
for (int i = 0; i < cilength; i++)
{
      di[i] = ci[i].modPow(privateKey,
      n);
      plain += di[i].ToString();
}
```

IV. ALGORITMA PAILLIER

A. Pembangkitan Kunci

Untuk melakukan enkripsi dan dekripsi, sama seperti algortima RSA, algoritma Paillier juga memerlukan kunci publik dan kunci privat. Kunci publik, seperti namanya, dapat diketahui oleh publik dan digunakan untuk

mengenkripsi pesan. Pesan yang dienkripsi dengan kunci publik hanya dapat didekripsi dengan menggunakan kunci privat padanannya yang tentunya bersifat rahasia. Kunci publik dan kunci privat yang digunakan dalam algoritma Paillier dibangkitkan dengan mekanisme sebagai berikut:

1. Memilih dua buah bilangan prima berukuran besar sembarang secara acak, sama seperti pada RSA, dua buah bilangan ini juga umum disebut *p* dan *q*.
2. Menghitung nilai *n*, dimana *n* adalah hasil perkalian antara nilai *p* dan nilai *q* ($n = p \cdot q$).
3. Menghitung nilai: $\lambda = \text{kpk}(p-1, q-1)$.
4. Memilih bilangan bulat acak *g*, dimana *n* membagi orde *g*. Hal ini dapat diilakukan dengan cara memeriksa persamaan berikut ini:

$$\text{fpb}\left(L\left(g^{\lambda} \bmod n^2\right), n\right) = 1 \qquad (5)$$

dimanaL didefinisikan sebagai berikut:

$$L(u) = \frac{u-1}{n} \qquad (6)$$

Yang perlu diperhatikan dalam algoritma pembangkitan kunci publik dan kunci privat algoritma Paillier, yaitu nilai bilangan prima *p* dan *q* dipilih secara acak dan independen satu sama lain sedemikian sehingga:

$$\text{fpb}(pq, (p-1)(q-1)) = 1. \qquad (7)$$

Hasil dari algortima pembangkitan pasangan kunci di atas adalah sebuah kunci publik yang merupakan pasangan nilai *n* dan *g*, serta λ sebagai kunci privat [5].

Kode yang merepresentasikan algoritma pembangkitan kunci Paillier diberikan dalam bahasa C# .NET sebagai berikut:

```
//hitung nilai n.
//hitung n2 = n * n.
//hitung nilai lambda.
//memilih bilangan bulat acak g.
n = p * q;
n2 = n * n;
lambda = ((p - 1) * (q - 1)) / ((p -
1).gcd(q - 1));

do
{
      g = randomZStarNSquare();
}
while((((g.modPow(lambda,n2) - 1) /
n).gcd(n)) != 1);
```

dengan `randomZStarNSquare()` adalah fungsi yang didefinisikan sebagai berikut:

```
private BigInteger
randomZStarNSquare()
{
BigInteger r = new BigInteger();
BigInteger prer = 999;
do
{
      prer.genRandomBits(((model * 2)-
      1), new Random());
}
while ((prer >= n2) ||
(prer.gcd(n2) != 1));
r = prer;

return r;
}
```

dengan model adalah panjang nilai *n* dalam bit.

B. Enkripsi

Mekanisme enkripsi algoritma Paillier juga jauh lebih sederhana daripada algoritma pembangkitan kuncinya. Algoritma enkripsi Paillier secara lengkap adalah sebagai berikut:

1. Mengambil kunci publik penerima pesan, yang secara umum disebut *g*, dan nilai *n*.
2. Memilih sebuah bilangan bulat acak *r*.
3. Cipherteks dihitung dengan rumus sebagai berikut:

$$c = g^m \cdot r^n \bmod n^2 \qquad (8)$$

dengan *c* adalah cipherteks hasil enkripsi dan *m* adalah plainteks yang dienkripsi [5].

Kode yang merepresentasikan algoritma enkripsi Paillier diberikan dalam bahasa C# .NET sebagai berikut:

```
//pemecahan plainteks menjadi blok
//dengan cara yang sama seperti RSA.

//memilih bilangan bulat acak r.
//enkripsi plainteks mi
//dengan kunci publik g dan n
//sehingga dihasilkan cipherteks ci.
BigInteger r = randomZStarN(n,model);
BigInteger ci;
ci = (g.modPow(new BigInteger(mi, 10),
n2) * r.modPow(n, n2)) % n2;
```

dengan `randomZStarN(n,model)` adalah fungsi yang didefinisikan sebagai berikut:

```
private BigInteger randomZStarN()
{
BigInteger r = new BigInteger();
do
{
      r.genRandomBits(model, new
      Random());
}
while ((r >= n) || (r.gcd(n) != 1));

return r;
```

```
}
```

C. Dekripsi

Mekanisme dekripsi Paillier sedikit lebih rumit daripada mekanisme enkripsinya. Cipherteks Pailler didekripsi dengan menggunakan rumus ini:

$$m = \frac{L(c^{\lambda} \bmod n^2)}{L(g^{\lambda} \bmod n^2)} \bmod n \quad (9)$$

Kode yang merepresentasikan algoritma dekripsi Paillier diberikan dalam bahasa C# .NET sebagai berikut:

```
//dari rumus dekripsi, bagian pembagi
//dihitung terlebih dahulu.
//kemudian dekripsi cipherteks ci
//dengan kunci privat lambda
//sehingga dihasilkan plainteks di.
mu = ((g.modPow(lambda, n2) - 1) /
n).modInverse(n);

di = (((ci.modPow(lambda, n2) - 1) /
n) * mu) % n;
```

V. PROPERTI DAN KEUNIKAN ALGORITMA

A. Properti dan Keunikan Algoritma RSA

Algoritma RSA memiliki beberapa properti yang digunakan dalam melakukan enkripsi atau dekripsi. Semua properti yang dimiliki oleh algoritma RSA dibangkitkan pada saat pembangkitan kunci. Berikut adalah properti-properti yang dimiliki oleh algortima RSA:

Tabel 1 Properti Algoritma RSA

Properti	Sifat Kerahasiaan
***p* (bilangan prima besar)**	privat
***q* (bilangan prima besar)**	privat
***n* = *p* · *q* (kunci)**	publik
ϕ(*n*) = (*p* – 1)(*q* – 1)	privat
***e* (kunci publik)**	publik
***d* (kunci privat)**	privat

Salah satu keunikan yang dimiliki oleh algoritma RSA adalah bahwa mekanisme enkripsi den dekripsinya identik, sehingga algoritma ini tidak hanya digunakan untuk mengenkripsi informasi tertentu, tetapi juga sangat cocok untuk digunakan sebagai algoritma tandatangan digital [1].

B. Properti dan Keunikan Algoritma Paillier

Algoritma Paillier juga memiliki beberapa properti yang digunakan dalam melakukan enkripsi atau dekripsi. Sama seperti pada algoritma RSA, semua properti yang dimiliki oleh algoritma Paillier dibangkitkan pada saat pembangkitan kunci, kecuali properti r yang dibangkitkan sebelum enkripsi dan properti μ, yang merupakan properti tambahan. Properti μ dapat dibangkitkan sebelum melakukan dekripsi. Berikut adalah properti-properti yang dimiliki oleh algortima Paillier:

Tabel 2 Properti Algoritma Paillier

Properti	Sifat Kerahasiaan
***p* (bilangan prima besar)**	privat
***q* (bilangan prima besar)**	privat
***n* = *p* · *q* (kunci)**	publik
λ = kpk(*p* – 1, *q* – 1) (kunci privat)	privat
***g* (kunci publik)**	publik
r	privat
μ	privat

Salah satu keunikan algoritma Paillier adalah sifat *Additive Homomorphic* yang dimiliki oleh algoritma enkripsinya. Sifat ini memunculkan identitas-identitas berikut:

$$D(E(m_1)E(m_2) \bmod n^2) = m_1 + m_2 \bmod n \quad (10)$$
$$D(E(m)^k \bmod n^2) = km \bmod n \quad (11)$$
$$D(E(m_1)g^{m2} \bmod n^2) = m_1 + m_2 \bmod n \quad (12)$$
$$D(E(m_1)^{m2} \bmod n^2) = m_1 m_2 \bmod n \quad (13)$$
$$D(E(m_2)^{m1} \bmod n^2) = m_1 m_2 \bmod n \quad (14)$$

Identitas-identitas ini membuat algoritma Paillier tidak hanya digunakan untuk mengenkripsi informasi tertentu, tetapi juga digunakan untuk merancang protokol voting, *threshold cryptosystem*, *watermarking*, skema *secret sharing*, dan *server-aided polynomial evaluation* [4].

Dengan demikian, hanya dengan kunci publik dan enkripsi m_1 dan m_2, dapat dihitung enkripsi $m_1 + m_2$. Berikut adalah contoh sifat *Additive Homomorphic* yang dimiliki oleh Algoritma Paillier:

Misalkan terdapat plainteks $m_1 = 10$ dan $m_2 = 20$,
dan $n = 2195455579$.
Hasil penjumlahan plainteks:

$$m_1 + m_2 \bmod n = 30.$$

Hasil penjumlahan cipherteks-nya:

$$c_1 \cdot c_2 \bmod n^2 = 4458845838505257242.$$

Dekripsi hasil penjumlahan cipherteks:

$$D(c_1 \cdot c_2 \bmod n^2) = 30.$$

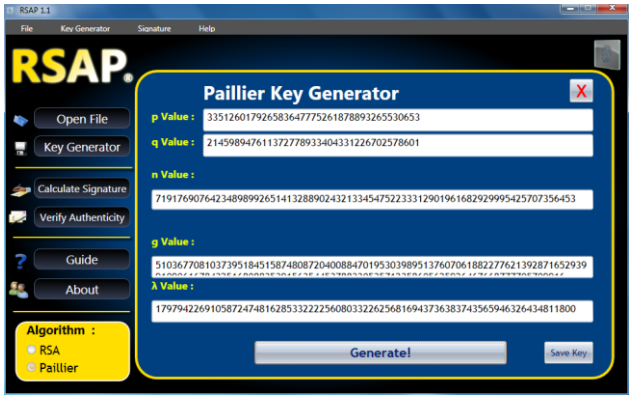
Sifat lain yang dimiliki oleh algoritma Paillier adalah sifat *self-blinding*. *Self-blinding* adalah sebuah sifat dimana semua cipherteks dapat diubah ke cipherteks lain secara random tanpa mengubah plainteks [4]. Sifat ini dapat diilustrasikan pada dua persamaan berikut ini:

$$D(E(m)) = m \text{ dan } D(E(m)r^n \bmod n^2) = m. \quad (15,16)$$

VI. IMPLEMENTASI RSA DAN PAILLIER

Bersamaan dengan penulisan makalah ini, telah dibuat sebuah program aplikasi *desktop* sederhana yang

memanfaatkan kedua algoritma, baik algoritma RSA maupun algoritma Paillier. Yang diimplementasikan dari masing-masing algoritma adalah keseluruhan tahapnya, dari tahap pembangkitan kunci, tahap enkripsi, sampai tahap dekripsi. Kedua algoritma tersebut diimplementasikan di atas *platform* Microsoft® .NET Framework 3.5 Service Pack 1, dengan bahasa pemrograman C# 3.0. Program ini digunakan untuk melakukan beberapa pengujian dengan parameter uji sederhana. Parameter-parameter yang diujikan adalah perbandingan kecepatan pembangkitan kunci, kecepatan enkripsi, dan kecepatan dekripsi. Selain itu juga diujikan modifikasi pada cipherteks untuk melihat efeknya pada plainteks hasil dekripsi. Hasil eksperimen pengujian beserta analisisnya akan diberikan pada Bab VII.



Gambar 1 Program aplikasi pengujian

Program aplikasi *desktop* yang dibuat bisa didapatkan dengan mengunduh melalui pranala berikut ini:

http://students.if.itb.ac.id/~if17034/crypto/

atau melalui e-mail penulis.

VII. Eksperimen & Analisis Pengujian Waktu

A. Eksperimen Kecepatan Pembangkitan Kunci

Dari program yang dibuat, dilakukan pengujian kecepatan pembangkitan kunci pada masing-masing algoritma. Pengujian kecepatan pembangkitan kunci dilakukan dengan panjang nilai *p* dan *q* yang berbeda-beda, yaitu: 4 bit, 8 bit, 16 bit, 32 bit, 64 bit, 128 bit, sampai 256 bit. Dan untuk setiap panjang *p* dan *q* tersebut, dilakukan 6 kali pengujian. Ketelitian waktu yang diujikan adalah dalam detik dengan 7 angka di belakang koma (batas yang didukung oleh .NET). Berikut adalah rincian pengujian waktu yang diperlukan oleh kedua algoritma untuk membangkitkan kuncinya.

Tabel 3 Hasil Eksperimen Kecepatan Pembangkitan Kunci RSA

p&q (bit)	P1 (d)	P2 (d)	P3 (d)	P4 (d)	P5 (d)	P6 (d)
4	0.0960055	0.0980056	0.1720099	0.0530030	0.0810046	1.0306018
8	0.6350363	0.7488013	0.0936002	1.1388020	0.5616010	0.0936002
16	1.5120865	0.7020013	0.2184004	0.8736015	0.6240011	1.3572024
32	0.7332013	1.7472031	0.3120005	1.1544020	0.6396011	0.3276006
64	0.6050347	0.6864012	0.5304009	2.0436036	1.2792022	0.1092002
128	1.5360879	0.1248003	2.0436036	1.9500034	0.3276005	0.3588006
256	2.6832047	4.6342082	1.7014030	1.1398202	2.9806053	2.4960043

Dengan P1 sampai P6 adalah percobaan 1 sampai 6.

Tabel 4 Hasil Eksperimen Kecepatan Pembangkitan Kunci Paillier

p&q (bit)	P1 (d)	P2 (d)	P3 (d)	P4 (d)	P5 (d)	P6 (d)
4	0.070004 0	0.1560089	0.3220189	0.1710011	0.1590091	0.2184004
8	0.030001 7	0.0600034	0.0570033	0.0936002	0.0340020	0.0310018
16	0.156000 3	0.1258002	0.1510086	0.0312000	0.0624001	0.1092002
32	0.281800 5	0.2002011	0.2840163	0.2360135	0.3990228	0.3588006
64	0.655201 1	0.2652005	0.4534009	0.6240011	0.1716003	0.0624001
128	0.702001 3	0.5148009	0.1540088	0.3530202	0.8806495	0.4080234
256	3.057605 4	1.4976026	2.3868042	2.0904036	0.5928011	3.4008060

Dengan P1 sampai P6 adalah percobaan 1 sampai 6.

Tabel 5 Perbandingan Kecepatan Rata-rata Pembangkitan Kunci RSA dan Paillier

p&q (bit)	RSA (d)	Paillier (d)
4	0.2551051	0.1827404
8	0.5452402	0.0509354
16	0.8812155	0.1059349
32	0.9412017	0.2933091
64	0.8756401	0.3719673
128	1.0568161	0.5007805
256	2.6058743	2.1710092

B.Analisis Hasil Eksperimen Kecepatan Pembangkitan Kunci

Dari hasil eksperimen dapat dilihat bahwa untuk setiap ukuran *p* dan *q* yang berbeda-beda, pembangkitan kunci RSA lebih lambat dibandingkan pembangkitan kunci Paillier. Hal ini dapat disebabkan karena pembangkitan kunci privat RSA yang dari segi komputasi lebih rumit daripada pembangkitan kunci Paillier. Pembangkit kunci privat RSA melakukan pengulangan (*looping*) untuk mencari nilai *k* yang akan memberikan kunci privat bilangan bulat, dan pada setiap kali pengulangan dilakukan proses perkalian dan modulo bilangan besar yang dari segi komputasi tergolong rumit. Sedangkan pembangkit kunci privat Paillier hanya melakukan suatu operasi untuk mencari nilai KPK (*LCM*) 2 buah bilangan bulat, tanpa melakukan pengulangan (*looping*) apapun. Dan walaupun pembangkit kunci publik Pailler lebih rumit dibanding RSA, keduanya memiliki tingkat kompleksitas komputasi yang tidak jauh berbeda.

Selain terlihat bahwa kecepatan pembangkitan kunci Paillier lebih cepat daripada RSA, terlihat juga bahwa secara umum waktu yang diperlukan untuk membangkitkan kunci meningkat seiring dengan meningkatnya panjang *p* dan *q*. Hal ini terjadi karena dengan meningkatnya panjang *p* dan *q*, maka proses komputasi yang dilakukan juga lebih rumit karena bilangan yang harus diproses lebih besar.

Terdapat satu hal lagi yang dapat diperhatikan dari hasil eksperimen. Walaupun secara umum waktu yang digunakan untuk setiap panjang *p* dan *q* secara rata-rata sama, sering terjadi komputasi yang lebih lama atau lebih cepat daripada rata-rata. Hal ini terjadi karena perhitungan mencari properti-properti kunci pada kedua algoritma sebenarnya bersifat probabilistik. Dalam dunia komputasi, untuk membangkitkan sebuah bilangan prima yang besar secara acak dilakukan dengan pendekatan probabilistik. Oleh karena itulah, pembangkitkan bilangan prima berukuran besar sebenarnya tidak dapat benar-benar dipastikan kesuksesannya. Dan bila pembangkitan bilangan prima gagal, maka program akan terus mencoba untuk membangkitkan bilangan prima yang lainnya. Hal inilah yang menyebabkan terjadinya fluktuasi kecepatan pembangkitan kunci. Hal ini juga yang menjadi alasan dilakukannya beberapa kali eksperimen untuk setiap panjang nilai *p* dan *q*.

C. Eksperimen Kecepatan Enkripsi

Sama seperti pada pengujian kecepatan pembangkitan kunci, pengujian kecepatan enkripsi juga dilakukan dengan panjang nilai *p* dan *q* yang berbeda-beda, yaitu: 4 bit, 8 bit, 16 bit, 32 bit, 64 bit, 128 bit, sampai 256 bit. Namun untuk setiap panjang *p* dan *q* tersebut, dilakukan hanya 1 kali pengujian pada ukuran plainteks 160 bit. Ketelitian waktu yang diujikan juga dalam detik dengan 7 angka di belakang koma (batas yang didukung oleh .NET). Berikut adalah rincian pengujian waktu yang diperlukan oleh kedua algoritma untuk melakukan enkripsi.

Tabel 6 Hasil Eksperimen Kecepatan Enkripsi RSA dan Paillier

p&q (bit)	RSA (d)	Paillier (d)
4	0.0010001	0.0010000
8	0.0020001	0.0030002
16	0.0020002	0.0010001
32	0.0030002	0.0020002
64	0.0050003	0.0040002
128	0.0070004	0.1872003
256	0.0090005	1.5132027

D. Analisis Hasil Eksperimen Kecepatan Enkripsi

Dari hasil eksperimen, pada awalnya belum jelas terlihat algoritma mana yang lebih cepat dalam melakukan enkripsi. Namun jelas terlihat bahwa semakin besar panjang *p* dan *q*, algoritma Paillier cenderung semakin lambat, bahkan pada panjang *p* dan *q* 256 bit, algoritma Paillier sudah jauh lebih lambat daripada algoritma RSA. Hal ini terjadi karena komputasi enkripsi algoritma Paillier lebih rumit dibandingkan algoritma RSA. Pada algoritma RSA, enkripsi hanya melibatkan satu kali perpangkatan dan satu kali operasi modulo. Sedangkan pada Paillier, diperlukan dua kali operasi perpangkatan, satu kali operasi perkalian, dan satu kali operasi modulo.

Dapat dilihat juga, bahwa seiring dengan meningkatnya panjang *p* dan *q*, maka waktu yang diperlukan kedua algoritma untuk melakukan enkripsi juga meningkat. Bila diperhatikan, terlihat juga bahwa pada algoritma RSA, peningkatan ini terjadi secara linear dan teratur.

E. Eksperimen Kecepatan Dekripsi

Eksperimen kecepatan dekripsi yang dilakukan menggunakan konfigurasi yang sama dengan eksperimen kecepatan enkripsi. Berikut adalah rincian pengujian waktu yang diperlukan oleh kedua algoritma untuk melakukan dekripsi.

Tabel 7 Hasil Eksperimen Kecepatan Dekripsi RSA dan Paillier

p&*q* (bit)	RSA (d)	Paillier (d)
4	0.0010001	0.0010000
8	0.0020001	0.0010001
16	0.0030002	0.0020001
32	0.0030002	0.0020002
64	0.0030002	0.0040002
128	0.0050003	0.0070004
256	0.0050003	0.0340018

F. Analisis Hasil Eksperimen Kecepatan Dekripsi

Dari hasil eksperimen terhadap kecepatan dekripsi, terlihat bahwa pada kedua algoritma waktu yang diperlukan untuk melakukan dekripsi secara umum meningkat seiring dengan bertambahnya panjang *p* dan *q*. Sama seperti pada eksperimen kecepatan enkripsi, hasil eksperimen kecepatan dekripsi ini juga belum secara jelas menunjukkan algoritma yang lebih cepat dalam melakukan dekripsi. Namun demikian, hasil eksperimen ini mengindikasikan akan terjadinya peningkatan waktu yang diperlukan oleh algoritma Paillier bila panjang *p* dan *q* semakin besar. Hal ini terjadi karena algoritma dekrispi Paillier secara komputasi memang lebih rumit daripada algoritma dekripsi RSA.

Pengukuran kecepatan dilakukan dengan menggunakan prosesor Intel® Core® Duo T2500 @ 2.0 GHz x 2, 533.5 MHz Rated FSB, 2MB L2 Cache, dengan 2560 MB Memori.

VIII. EKSPERIMEN & ANALISIS PENGUJIAN MODIFIKASI CIPHERTEKS

Pada kedua algoritma, baik algoritma RSA maupun algoritma Paillier, efek yang dihasilkan dari modifikasi pada cipheteks bergantung pada skema pembagian plainteks menjadi blok-blok yang dilakukan pada saat melakukan enkripsi. Modifikasi pada satu blok cipherteks hanya akan merusak satu blok tersebut pada plainteks hasil deksripsi. Bila plainteks yang dienkripsi cukup kecil sehingga tidak perlu dibagi menjadi blok-blok, maka pada saat cipherteksnya dimodifikasi, keseluruhan plainteks hasil dekripsinya akan rusak.

Hasil eksperimen ini sesuai dengan yang seharusnya memang terjadi. Pada kedua algoritma, baik algoritma RSA maupun algoritma Paillier, blok-blok plainteks dienkripsi secara independen satu sama lain, begitu juga pada saat dekripsi, blok-blok cipherteks dideksripsi secara independen satu sama lain. Oleh karena itulah, modifikasi satu blok pada cipherteks hanya merusak satu blok yang berkoresponden pada plainteks hasil dekripsinya.

IX. KEKUATAN DAN KELEMAHAN ALGORITMA

A. Kekuatan dan Kelemahan Algoritma RSA

Keamanan algoritma RSA terletak pada sulitnya memfaktorkan bilangan yang besar menjadi faktor-faktor primanya. Pemfaktoran dilakukan untuk memperoleh kunci privat yang digunakan untuk mendekripsi cipherteks. Selama belum ditemukan algoritma yang mangkus untuk pemfaktoran bilangan besar menjadi faktor-faktor primanya, maka keamanan algoritma RSA tetap terjamin [1].

Sampai makalah ini ditulis, bilangan terbesar yang dapat difaktorkan oleh algoritma pemfaktoran umum adalah sepanjang 768 bit, dengan menggunakan implementasi komputasi terdistribusi yang canggih. Enkripsi RSA dengan menggunakan nilai *n* sepanjang 300 bit atau kurang sudah dapat dipecahkan dalam beberapa jam dengan menggunakan PC [9]. Pada tahun 2003 Tromer dan Shamir sendiri mendeskripsikan sebuah perangkat keras teoritikal, TWIRL, yang juga telah mengancam kunci RSA 1024 bit. Oleh karena itu, panjang kunci RSA minimum yang disarankan saat ini adalah 2048 bit [10].

Pemilihan nilai *p* dan *q* juga mempengaruhi kekuatan algoritma RSA. Nilai *p* dan *q* sebaiknya tidak terlalu dekat apalagi sama. Kunci privat *d* juga sebaiknya besar. Michael J. Wiener menunjukkan bahwa bila *p* berada di antara *q* dan 2*q* (yang memang umum) dan $d < n^{1/4}/3$, maka *d* dapat dihitung dengan cepat dari *n* dan *e* [3].

Salah satu kelemahan RSA yang paling signifikan adalah kecepatannya. RSA adalah algoritma enkripsi yang lambat karena proses komputasinya yang melibatkan bilangan-bilangan besar. Oleh karena itu, pada prakteknya RSA biasanya hanya digunakan untuk mengenkripsi kunci dari algoritma kriptografi kunci simetri dan untuk tandatangan digital.

B. Kekuatan dan Kelemahan Algoritma Paillier

Dengan mekanisme pembangkitan kunci dan enkripsi-dekripsi seperti yang telah dipaparkan pada Bab IV, algoritma Paillier memiliki keamanan semantik terhadap *Chosen-Plaintext Attacks*. *Decisional Composite Residuosity Assumption* (DCRA) yang menjadi basis kekuatan algoritma Paillier dipercaya sulit untuk dipecahkan.

Namun demikian, karena sifat algoritma Pailler yang homomorfik, algoritma ini tidak memiliki proteksi yang tinggi terhadap *Adaptive Chosen-Chipertext Attacks*. Biasanya sifat semacam ini menjadi celah keamanan pada sistem kriptografi, tetapi dalam aplikasi tertentu, sifat ini malah diperlukan. Aplikasi algoritma Paillier yang memanfaatkan sifat homomorfiknya akan dipaparkan pada Bab XI.

X. SERANGAN TERHADAP KEDUA ALGORITMA

A. Serangan Terhadap Algoritma RSA

Pada tahun 1995 Kocher mendeskripsikan sebuah serangan terhadap RSA: Misalnya terdapat penyerang Eve yang mengetahui perangkat keras yang dimiliki oleh Alice secara cukup detail dan dapat mengukur waktu dekripsi untuk beberapa cipherteks yang diketahui, Eve dapat menghitung kunci dekripsi *d* dengan cukup cepat [14]. Serangan ini juga dapat dilakukan terhadap skema tandatangan digital yang menggunakan RSA.

Pada tahun 1998 Daniel Beichenbacher mendeskripsikan *Adaptive Chosen-Ciphertext Attack* terhadap pesan yang dienkripsi dengan RSA yang menggunakan skema padding PKCS #1 versi 1. Namun serangan ini dapat dihindari dengan menggunakan skema padding yang lebih baik, misalnya *Optimal Asymmetric Encryption Padding* (OAEP) atau PKCS #1 versi yang lebih baru [13].

Onur Aciimez dan Cetin Kaya Koc, berhasil menemukan 508 dari 512 bit kunci RSA dalam 10 iterasi dengan menggunakan *Simple Branch Prediction Analysis* (SBPA) [12]. Banyak prosesor menggunakan *branch predictor* untuk menentukan apakah cabang kondisional pada sebuah instruksi sebuah program akan diambil atau tidak. Prosesor semacam ini biasanya juga menggunakan *Simultaneous Multithreading* (SMT). Branch Prediction Analysis menyerang dengan menggunakan proses mata-mata untuk mendapatkan kunci privat yang sedang diproses dengan menggunakn prosesor semacam itu. Serangan semacam ini disebut *Side-Channel Analysis Attack*.

B. Serangan Terhadap Algoritma Paillier

Seperti yang telah dipaparkan pada Bab sebelumnya, karena sifat algoritma Pailler yang homomorfik, algoritma ini tidak memiliki proteksi yang tinggi terhadap *Adaptive Chosen-Chipertext Attacks*.

XI. APLIKASI KEDUA ALGORITMA

A. Aplikasi Algoritma RSA

Selain digunakan langsung untuk mengenkripsi pesan, algoritma RSA umum digunakan untuk mengenkripsi kunci simetris yang digunakan pada sistem kriptografi kunci simetri. Penggunaan lain dari algoritma RSA yang juga sangat populer adalah pada skema tandatangan digital. Biasanya untuk penggunaan tandatangan digital, algoritma RSA disandingkan dengan fungsi *hash* Secure Hash Algorithm-1/SHA-1.

Algoritma RSA juga digunakan secara luas pada protokol-protokol *electronic commerce* (*e-Commerce*).

B. Aplikasi Algoritma Paillier

Aplikasi algoritma Paillier yang paling populer adalah untuk voting elektronik. Alasan penggunaan algoritma Paillier untuk voting elektronik bukan hanya keamanan semantiknya. Seperti yang telah dipaparkan sebelumnya, algoritma Paillier memiliki sifat homomorfik, sifat ini dimanfaatkan dalam sistem voting elektronik yang aman.

Penggunaan lain yang cukup populer dari algoritma Paillier adalah pada uang elektronik. Sifat algoritma Paillier yang lainnya dimanfaatkan disini. *Self-blinding*, salah satu sifat algoritma Paillier dimana cipherteks Paillier dapat diubah-ubah (dengan mekanisme tertentu) tanpa mengubah makna yang terkandung di dalamnya. Ide ini dikemukakan oleh David Chaum.

Secara umum, pada prinsipnya algoritma Paillier cocok untuk sistem dengan komputasi terdistribusi karena sifat *Additive Homomorphism* yang dimilikinya [6]. Selain itu algoritma Paillier juga dapat digunakan untuk enkripsi pesan dan tandatangan digital.

XII. KESIMPULAN

Terdapat tiga jenis sistem kriptografi kunci publik, ada yang berbasis pemfaktoran, logaritma diskrit, dan *residuosity class*. RSA adalah algoritma kunci publik jenis pertama yang kekuatannya berbasis pada sulitnya memfaktorkan bilangan bulat besar menjadi faktor-faktor primanya. Dan Paillier adalah salah satu algoritma jenis ketiga yang juga populer. Kekuatan algoritma Paillier berbasis pada *Decisional Composite Residuosity Assumption*. Kedua algoritma ini sama-sama menggunakan dua buah bilangan prima besar sebagai parameter awal. Kedua algoritma ini juga sama-sama terdiri atas tiga tahap komputasi, yaitu Pembangkitan Kunci, Enkripsi, dan Dekripsi. Namun demikian, mekanisme masing-masing tahap pada kedua algoritma tersebut sangat berbeda. Masing-masing algoritma juga memiliki sifat-sifat yang unik. Algoritma RSA memiliki mekanisme enkripsi yang identik dengan mekanisme dekripsinya. Dan algoritma Paillier memiliki sifat *Additive Homomorphic* dan *Self-Blinding*.

Pembangkitan kunci algoritma RSA lebih lambat daripada Paillier. Hal ini disebabkaan oleh pembangkitan Kunci Privat RSA yang lebih rumit dari segi komputasional daripada pembangkitan Kunci Privat Paillier. Sedangkan tingkat kompleksitas komputasi pembangkitan Kunci Publik kedua algoritma tidak terlalu jauh berbeda. Semakin panjang nilai *p* dan *q* pada kedua algoritma, semakin lama pula proses pembangkitan kuncinya. Terdapat fluktuasi kecepatan yang menyimpang dari kecepatan rata-rata pembangkitan kunci pada kedua algoritma. Hal ini terjadi karena pembangkitan bilangan prima besar secara komputasi bersifat probabilistik.

Proses enkripsi algoritma Paillier lebih lambat daripada RSA. Hal ini disebabkan oleh lebih rumitnya skema enkripsi Paillier dibandingkan dengan RSA. Hal yang sama terjadi pada proses dekripsi kedua algoritma. Untuk panjang nilai *p* dan *q* yang membesar, proses enkripsi dan dekripsi kedua algoritma juga memerlukan waktu yang lebih lama.

Algoritma RSA banyak digunakan untuk mengenkripsi

pesan, tandatangan digital, dan untuk mengenkripsi kunci simetri pada sistem kriptografi kunci simetri. Algoritma RSA juga digunakan secara luas pada protokol-protokol *e-commerce*. Algoritma Paillier populer digunakan pada voting elektronik yang aman, uang elektronik, dan berbagai sistem kriptografi dengan komputasi terdistribusi.

REFERENSI

[1] Bahan kuliah Kriptografi oleh Ir. Rinaldi Munir: Landasan Matematika untuk Kriptografi, 2005; Kriptografi Kunci Publik, 2005; Algoritma RSA dan ElGamal, 2004;

[2] Bahan kuliah Struktur Diskrit oleh Ir. Rinaldi Munir: Teori Bilangan, 2005

[3] Michael J. Wiener, *Cryptanalysis of short RSA secret exponents*, IEEE, 1990

[4] Pascal Paillier, *Public-Key Cryptosystem Based on Composite Degree Residuosity Classes*, EUROCRYPT, 1999

[5] Pascal Paillier, *Efficient Public-Key Cryptosystem Provably Secure Against Active Adversaries*, ASIACRYPT, 1999

[6] Pascal Paillier, *Composite-Residuosity Based Cryptography: An Overview*, CryptoBytes, 2002

[7] Ron Rivest, Adi Shamir, Leonard Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, ACM, 1978

[8] Sara Robinson, *Still Guarding Secrets after Years of Attacks, RSA Earns Accolades for its Founders*, SIAM News, 2003

[9] "518-bit GNFS with msieve", Mersenne Forum: http://www.mersenneforum.org/

[10] "Has the RSA algorithm been compromised as a result of Bernstein's Paper?", RSA Laboratories: http://www.rsa.com/rsalabs/

[11] "FaultBased Attack of RSA Authentication", EECS - UMich: http://www.eecs.umich.edu/~valeria/research/publications/

[12] "On the Power of Simple Branch Prediction Analysis", Onur Aciicmez, Cetin Kaya Koc, Jean-Pierre Seifert: http://eprint.iacr.org/2006/351

[13] "PKCS #1: RSA Cryptography Standard", RSA Laboratories: http://www.rsa.com/rsalabs/

[14] "Timing Attacks", Kocher: http://www.cryptography.com/resources/whitepapers/TimingAttacks

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

I.Y.B. Aditya Eka Prabawa
13507034