Perbandingan Algoritma RSA dan Diffie-Hellman

Yudi Retanto 13508085
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
yudiretanto@gmail.com

Sebagian besar algoritma kriptografi menggunakan kunci untuk melakukan enkripsi ataupun dekripsi. Ketika dua buah pihak berkomunikasi menggunakan jaringan yang tidak aman maka digunakan algoritma kriptografi untuk menyembunyikan pesan. Namun dibutuhkan pertukaran kunci antara kedua pihak agar algoritma yang digunakan bisa berjalan. Agloritma RSA menggunakan kunci publik dan kunci privat untuk pertukaran kunci antarpihak yang berkomunikasi sedangkan Algoritma Diffi-Hellman menggunakan pertukaran data untuk menghasilkan suatu kunci simetris yang digunakna untuk proses enkripsi dan dekripsi.

***Indekss*—RSA, Diffie-Hellman, Pertukaran Kunci, Waktu, Keamanan, dan Efektifitas.**

TEORI DASAR

Pada awalnya algoritma-algoritma kriptografi yang ada menggunakan kunci yang sama (simetris) untuk melakukan proses enkripsi dan proses dekripsi. Jadi cukup dengan satu buah kunci proses enkripsi dan dekripsi dapat dijalankan. Namun hal ini dapat mengancam keamanan proses komunikasi jika menggunakan algoritma kriptografi dengan kunci simetris. Sebagai contoh ketika *Alice* dan *Bob* akan melakukan komunikasi yang aman dengan menggunakan proses enkripsi untuk tiap pesan yang akan dikirim maka pertama-tama *Alice* akan memberitahu *Bob* kunci yang akan digunakan untuk melakukan proses enkripsi dan dekripsi pesan. Hal inilah yang membuat algoritma kriptografi dengan knci simetri kurang aman, karena kita harus memberitahu pihak lawan komunikasi kunci yang digunakan untuk melakukan proses enkripsi dan dekripsi. Jika pada proses pemberitahuan kunci disadap atau pihak yang tidak berhak mengetahui kunci ini maka proses komunikasi dengan kunci ini menjadi tidak aman sama sekali.

Kriptografi Kunci Publik

Untuk menyikapi permasalahan tersebut maka dibuatlah algoritma kriptografi kunci public (nir-simetri) untuk menjadi solusi. Algoritma ini diterbitkan pada tahun 1976 oleh Whitfield Diffie dan Martin Hellman, yang dipengaruhi oleh pekerjaan Ralph Merkle tentang distribusi kunci publik. Metode pertukaran kunci ini kemudian dikenal sebagai pertukaran kunci Diffie-Hellman .

Tidak seperti algoritma kunci simetris, algoritma kunci public tidak memerlukan pertukaran awal yang aman dari satu atau lebih kunci rahasia antara pengirim dan penerima. Algoritma yang khusus untuk mengenkripsi dan dekripsi dirancang sedemikian rupa sehingga mudah bagi pengirim mengenkripsi pesan menggunakan kunci publik dan penerima mendekripsi menggunakan kunci privat. Karena penerima hanya mengirimkan kunci publik ke pengirim pesan, maka jika proses pengiriman kunci ini diketahui oleh orang lain maka ia tetap akan kesulitan untuk mendekripsi pesan karena tidak mengetahui kunci privat untuk mendekripsi pesan tersebut.

Pada dasarnya proses pengiriman pesan menggunakan algoritma ini adalah sebagai berikut :

1. Penerima pesan menginginkan pesan yang dikirmkan oleh pengirim aman dari pihak lain.
2. Penerima pesan membuat dua buah kunci yaitu kunci publik dan kunci private. Kunci publik digunakan untuk mengenkripsi pesan dan kunci private digunakan untuk mendekripsi pesan.
3. Maka peenerima pesan akan mengirimkan kunci publik ke pada pengirim pesan. Pada proses ini mungkin ada penyadap.
4. Pengirim menerima kunci publik dan mulai mengenkripsi pesan untuk dikirimkan menggunakan kunci publik yang didapat.
5. Pengirim mengirimkan pesan yang terenkripsi dengna kunci publik. Pada proses ini pesan mungkin dapat disadap orang lain.
6. Penerima menerima pesan yang terenkripsi. Kemudiaan mendekripsi pesan dengan kunci private yang dimiliki sehingga didapat pesan asli.

Algoritma RSA

RSA adalah salah satu jenis algoritma yang menggunakan kriptografi kunci publik. Algorimt aini adalah algoritma pertama yang cocok untuk melakukan *signing* seperti *digital signature* sebaik melakukan enkripsi dan salah satu perkembangan yang baik dalam kriptografi kunci publik.

Algoritma RSA dipublikasikan pada tahun 1978 oleh Ron Rivest, Adi Shamir, dan Leonard Adleman di Massachusetts Institute of Technology (MIT). Nama algoritma RSA didapatkan dari huruf pertama dari nama belakang mereka yaitu Rivest, Shamir dan Adleman.

Algoritma RSA memiliki tiga buah operasi yaitu pembangkitan kunci, enkripsi dan dekripsi.

Algoritma RSA memiliki kunci publik dan kunci private, dimana kunci publik dapat diketahui oleh semua orang untuk digunakan dlama mengenkripsi pesan. Pesan yang terenkripsi dengan kunci publik hanya dapat didekripsi dengan kunci private yang cocok. jadi dibutuhkan sepasang kunci untuk dapat menjalankan algoritma ini. Kunci untuk algoritma RSA dapat diperoleh dengan beberapa langkah berikut:

1. Pilih dua buah bilangan prima p dan q yang berbeda.
2. Hitung nilai n = pq
3. Hitung $\varphi(n) = (p-1)(q-1)$, dimana φ adalah fungsi totient Euler.
4. Pilih sebuah bilangan integer e dimana 1 < e < φ(*n*) dan gcd(e, φ(*n*)) = 1atau dengan kata lin e dan φ(*n*) relatif prima. Gcd (Greatest Common Divisor) adalah fpb (Faktor Pembagi Terbesar).
5. Tentukan $d = e^{-1} \bmod \varphi(n)$.

Dari proses diatas akan didapatkan tiga buah bilangan yaitu e, d dan n. Kunci publik adalah sepasang nilai e dan n sedangkan kunci privat adalah sepasang nilai d dan n.

Untuk melakukan proses enkripsi dengan kunci yang telah dihasilkan dari proses diatas maka akan diilustrasikan sbagai berikut. Ada sepasang manusia yang ingin berkomunikasi secara aman menggunakan algortima RSA misal Alice dan Bob. Alice ingin menerima pesan P dari bob maka Alice mengirimkan kunci publik (n,e) ke Bob dan menyimpan kunci private(n,d). pertama bob merubah pesan P mejadi suatu nilai bilangan integer m dimana 0 < m < n. Lalu ia melakukan perhitungan untuk mendapatkan ciphertext c dengan rumus:

$$c = P^e \pmod{n}$$

Setelah Bob mengirimkan c sebagai pesan P yang terenkripsi maka Alice akan mulai mendekripsi dengan kunci private yang dimiliki. Maka Alice akan menghitung:

$$P = c^d \pmod{n}.$$

Dengan begitu Alice mendapatkan pesan P dan dapat mulai mengolah informasi yang dikandungnya.

Algoritma Diffie-Hellman

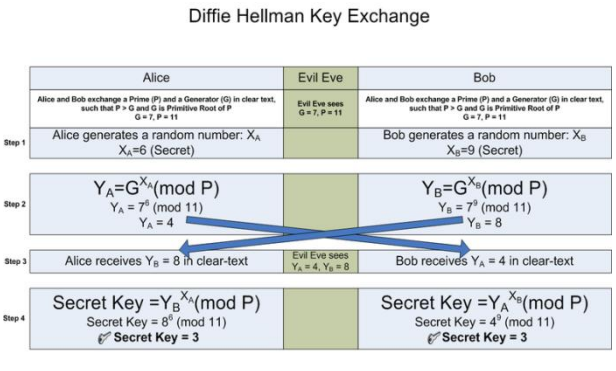
Adalah metode pertukaran kunci yang spesifik. Algoritma ini adalah salah satu contoh praktis dalam implementasi pertukaran kunci dalam bidang kriptografi. Algortima ini memungkinkan dua pihak untuk bertukar kunci simetri secara aman meskipun melalui saluran yang tidak aman.

Skema ini pertama kali diperkenalkan oleh Whitfield Diffie dan Martin Hellman pada tahun 1976. Pada tahun 2002, Hellman menyarankan algoritma ini dipanggil dengan sebutan algoritma pertukaran kunci Diffie-Hellman-Merkle karena contribusi dari Ralph Merkle dalam kriptografi kunci publik.

Tidak seperti Algoritma RSA yang menggunakan dua buah kunci yang berbeda untuk melakukan enkripsi dan dekripsi, Algoritma Diffie-Hellman meneyediakan mekanisme pertukaran kunci yang aman antara dua pihak yang nantinya digunakan untuk melakukan enkripsi dan dekripsi menggunakan Algoritma Kriptografi Kunci Simetris.

Algoritma Diffie-Hellman menyediakan pertukaran kunci yang dapat digunakan untuk melakukan enkripsi dan dekripsi dengan beberapa pertukaran data melalui jaringan publik. Berikut ini langkah-langkah yang harus dilakukan:

1. Ada dua pihak yang akan melakukan komunikasi yaitu Alice dan Bob.
2. Karena ingin pesan yang disamapaikan dienkripsi maka Alice dan Bob harus saling mengetahui kunci rahasia yang digunakan untuk mengenkripsi dan mendekripsi pesan.
3. Maka Alice dan Bob memilih bilangan prima P dan bilangan bulat G dimana P > G. kedua bilangan, P dan G, harus saling relatif prima.
4. Alice memilih satu bilangan random rahasia X_A. Lalu Alice melakukan perhitungan
$$Y_A = G^{X_A} (mod\ P)$$
5. Bob memilih satu bilangan random rahasia X_B. Lalu Bob melakukan perhitungan yang sama seperti Alice yitu
$$Y_B = G^{X_B} (mod\ P)$$
6. Lalu Alice dna Bob saling mengirimkan nilai Y_A dan Y_B hasil perhitungannya masing-masing dan menjaga kerahasiaan bilangan random masing-masing yaitu X_A dan X_B.
7. Alice mendapatkan Y_b dari Bob dan Bob mendapatkan Y_A dari Alice.
8. Alice mulai menghitung kunci rahasia yang akan digunakan untuk enkripsi dan dekripsi menggunaan perhitungan:
$$Kunci_A = {Y_B}^{X_A} (mod\ P)$$
9. Bob mulai menghitung kunci rahasia yang akan digunakan untuk enkripsi dan dekripsi menggunakan perhitungan:
$$Kunci_B = {Y_A}^{X_B} (mod\ P)$$
10. Maka nilai $Kunci_A$ = $Kunci_B$ dan Alice bersama Bob dapat mulai bertukar pesan menggunakan kunci ini.
11. Berikut ini ilustrasinya:



Jika ada pihak lain yang menyadap komunikasi antara Alice dan Bob maka ia hanya akan mengetahui nilai G, P,

Y_A dan Y_B tanpa mengetahui nilai X_A dan X_B sehingga pihak penyadap tidak akan mendapatkan nilai kunci hanya dengan nilai-nilai tersebut.

IMPLEMENTASI

Pada implementasi kali ini penulis akan menggunakan library BigInteger yang digunakan pada lingkungan pemrograman C# .NET. Penulis menggunakan library ini karena penulis akan menguji kedua Algortima yaitu Algoritma RSA dan ALgoritma Diffie-Hellman menggunakan angka-angka yang besar sehingga didapat kunci-kunci yang cukup aman.

Aplikasi yang dikembangkan berupa *Desktop Application* dengan menggunakan IDE Microsoft Visual Studio 2010 Ultimate.

Pustaka BigInteger yang digunakan memeiliki beberapa fungsi pembantu yaitu *genPseudoPrime, genCoPrime, modInverse,* dan *modPow.* Pada *genPseudoPrime* digunakan algoritma Fermat Little's Theorem untuk mendapatkan bilangna prima semu dengan nilai confidence sebagai masukan, nilai confidence ini akan menjadi jumlah looping yang dilakukan dengan pengujian Fermat.

RSA Key Generator

Pada penjelasan diatas sudah dijelaskan dasar-dasar dari Algoritma RSA sebagai slaha satu Kriptografi Kunci Publik. Karena keamanan Algoritma ini bergantung dari kunci yang digunakan maka akan dibuat implementasinya dalam bentuk program untuk membangkitkan sepasang kunci publik dan private.

Karena pada algoritma RSA dibutuhkan beberapa 6 buah variabel yang memiliki perannya masing-masing yaitu P, Q, N, TE (Totient Euler), E (Enkripsi) dan D (Dekripsi), maka pada implementasi ini dibuatlah keenam variable tersebut. Berikut ini implementasinya:

```
private BigInteger P;
private BigInteger Q;
private BigInteger N;
private BigInteger TE;
private BigInteger E;
private BigInteger D;
```

Setelah mendeklarasikan keenam variable tersebut kita dapat memulai menghitung nilai-nilai dari tiap variable agar didapatkan sepasang kunci publik dan privat yang utuh.

Pertama dipilih dua buah bilangan bulat prima untuk nilai P dan Q. nilai P dan Q harus berbeda. Pada implementasi kali ini penulis akan men-*generate* nilai P dan Q secara otomatis dan random. Pada implementasi ini, library BigInteger sudah memiliki fungsi untuk menghasilkan bilangan prima yaitu *genPseudoPrime()*. Fungsi ini menghasilkan bilangan prima semu.

Karena fungsi ini pada dasarnya membangkitkan nilai bilangan prima dengan kelas Random dari parameter yang diberikan, maka dihitung nilai seed agar nilai yang dihasilkan oleh fungsi ini akan semakin random. Nilai seed ini didapatkan dengan menghitung nilai waktu saat ini yaitu nilai jam dikalikan lima lalu ditambah nilai menit dikalikanempat lalu nilai detik dikalikan tiga dan terakhir ditambah nilai milidetik dikalikan dua. Lalu nilai ini digunakan untuk menginstantiasi kelas Random. Berikutnya kelas Random ini akan diberikan sebagai parameter pada fungsi *genPseudoPrime*.

Parameter lainnya pada fungsi *genPseudoPrime* adalah jumlah bit bilangan prima yang ingin dihasilkan dan *confidence*. Karena diinginkan bilangan prima yang cukup besar maka dimasukkan nilai 256 untuk jumlah bit bilangan prima yang akan dihasilkan untuk *confidence* dimasukkan nilai 50. Nilai ini akan menjadi nilai *loop* untuk menguji nilai yang dihasilkan dengan Fermat Little Theorem. Jadi semakin besar maka semakin baik nilai prima semu yang dihasilkan.

Setelah berhasil menghasilkan nilai P secara random menggunakan fungsi *genPseudoPrime* maka dihitung pula nilai Q menggunakan fungsi yang sama, namun sebelumnya perlu dinstantiasi kelas Random dengan nilai seed berbeda agara semakin menambah random nilai bilangan prima yang akan dihasilkan, oleh sebab itu dihiutng kembali nilai seed menggunakan metode yang sama seperti saat menghitung seed untuk menghasilkan nilai P.

Berikut ini kode implementasinya:

```
int seed = (DateTime.Now.Hour * 5) +
           (DateTime.Now.Minute * 4) +
           (DateTime.Now.Second * 3) +
           (DateTime.Now.Millisecond * 2);
P = BigInteger.genPseudoPrime(256, 50, new
           Random(seed));
seed = (DateTime.Now.Hour * 5) +
           (DateTime.Now.Minute * 4) +
           (DateTime.Now.Second * 3) +
           (DateTime.Now.Millisecond * 2);
Q = BigInteger.genPseudoPrime(256, 50, new
           Random(seed));
```

Dari implementasi diatas didapat nilai P dan Q, oleh sebab dapat dihitung ilai N dan TE (Totient Euler). Kedua nilai tersebut dapat dihitung dengan rumus berikut:

$$N = P \times Q$$

$$TE = (P - 1)(Q - 1)$$

Berikut ini kode implementasinya:

```
N = P * Q;
TE = (P - 1) * (Q - 1);
```

Karena sudah memiliki nilai P, Q, N, dan TE maka kita dapat mulai menghitung kunci publik. Kunci publik merupakan sepasang nilai N dan E. dari perhitungan sebelumnya kita sudah mendapatkan nilai N, sekarang kita akan mulai menghitung nilai E.

E adalah sebuah nilai bilangan bulat dimana 0 < E < TE serta E dan N dalah relatif prima. Nilai E bisa didapat dengan menggunakan salah satu fungsi yang dimiliki kelas BIgInteger yaitu *genCoPrime()*. Fungsi ini akan menghasilkan bilangan x dimana FPB (Faktor Pembagi Terbesar) dari kedua bilangan adalah 1. Fungsi ini

menerima dua buah parameter yaitu panjang bit nilai yang akan dihasilkan dan objek kelas Random. Panajng nilai bit yang dimasukkan oleh penulis sebagai parameter adalah 128. Nilai ini diras penulis sudah cukup panjang untuk menjadi nilai E. untuk parameter kedua penulis kembali menghitung seed dengan cara yang sama pada saat men-*generate* nilai P dan Q sebelumnya.

Berikut ini kode implementasinya:

```
seed = (DateTime.Now.Hour * 5) +
       (DateTime.Now.Minute * 4) +
       (DateTime.Now.Second * 3) +
       (DateTime.Now.Millisecond * 2);
E = TE.genCoPrime(128, new Random(seed));
```

Setelah berhasil mendapatkan nilai E maka kita dapat mulai menghitung nilai D yang nantinya digunakan untuk menjadi kunci private. Sepasang nilai N dan D adalah kunci private yang perlu dijaga kerahasiannya. Nilai D dapat dihitung dengan rumus:

$$D \equiv E^1 \, Mod \, TE$$

Atau

$$(E \times D) \, Mod \, TE = 1$$

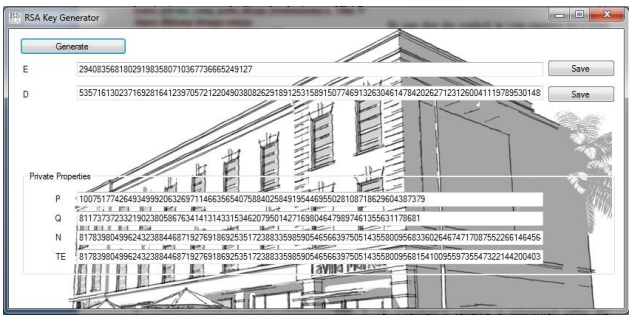
Dalam kelas BigInteger yang penulis gunakan terdapat fungsi untuk melakukan fungsi tersebut yaitu fungsi *modInverse()*. Fungsi ini menerima masukan sebuah bilangan BigInteger lalau menghitung nilai mod inversenya.

Berikut ini kode implementasinya:

```
D = E.modInverse(TE);
```

Sekarang telah didapatkan nilai untuk keenam variable, sepasang kunci publik dan private pun telah didapatkan dari perhitungan diatas. Berikut ini hasil *screenshot* jika program dijalankan

Gambar 1



Gambar 2

Dari hasil percobaan diatas didapatkan nilai-nilai sebagai berikut :

- P = 10075177426493499920632697114663565407588402584919544695502810871862960438737 9
- Q = 81173737233219023805867634141314331534620795014271698046479897461355631178681
- N = 81783980499624323884468719276918692535172388335985905465663975051435580095683360264674717087552266146456432274560776543519674706094641013360632269026709 9
- TE = 81783980499624323884468719276918692535172388335985905465663975051435580095681541009559735547322144200403552774704671495311040034644625933298833374547010 40
- E = 294083568180291983580710367736665249127
- D = 53571613023716928164123970572122049038082629189125315891507746913263046147842026271231260041119789530148914598972761469822558478560884218361817920040792 23

Diffie-Hellman Key Generator

Pada penjelasan diatas sudah dijelaskan dasar-dasar dari Algoritma pertukaran kunci Diffie-Hellman. Karena algoritm ini juga bergantung dengan besarnya angka-angka yang digunakan maka penulis akan membuat program untuk mensimulasikan algoritma ini dengan bilangan-bilangan besar.

Karena pada Algoritma Diffie-Hellman dibutuhkan 8 buah variabel yang memiliki perannya masing-masing yaitu P, G, XA, XB, YA, YB, KeyA, dan KeyB, maka pada implementasi ini dibuatlah keenam variable tersebut. Berikut ini implementasinya:

```
private BigInteger P;
private BigInteger G;
private BigInteger XA;
private BigInteger XB;
private BigInteger YA;
private BigInteger YB;
private BigInteger KeyA;
private BigInteger KeyB;
```

Setelah mendeklarasikan kedelapan variable tersebut kita dapat memulai untuk menghitung nilainya masin-masing sesuai algoritma Diffie-Hellman.

Langkah pertama yang harus dilakukan apda algoritma ini adalah menyepakati nilai P dan G pada kedua belah pihak. Nilai P dalah suatu bilangan bulat prima sedangakan G adalah suatu bilangan bulat sedemikian rupa sehingga P dan G adalah relatif prima.

Karena nilai P adalah suatu bilangan prima maka kita dapat men-*generate* nilai ini dengan fungsi *genPseudoPrime* pada kelas BigInteger. Seperti pada algoritma RSA, dihitung nilai seed dengan perhitungan nilai waktu saat ini yaitu nilai jam dikalikan lima lalu ditambah nilai menit dikalikan empat lalu nilai detik dikalikan tiga dan terakhir ditambah nilai milidetik dikalikan dua. Lalu nilai ini digunakan untuk menginstantiasi kelas Random yang akan menjadi parameter fungsi *genPseudoPrime*. Untuk parameter panajng bit dan nilai *confidence* digunakan nilai yang sama pada implementasi ALgoritma RSA diatas.

Berikutnya dihitung nilai G. karena nilai G harus relatif prima dengan nilai P yang sudah didapatkan, digunakan fungsi *genCoPrime* pada kelas BigInteger dengan parameter panjang bit 256 dan Random yang dinstantiasi dengan seed sesuai perhitungan sebelumnya.

Berikut ini kode implementasinya:

```
int seed = (DateTime.Now.Hour * 5) +
           (DateTime.Now.Minute * 4) +
           (DateTime.Now.Second * 3) +
           (DateTime.Now.Millisecond * 2);
P = BigInteger.genPseudoPrime(256, 50, new
           Random(seed));
 seed = (DateTime.Now.Hour * 5) +
           (DateTime.Now.Minute * 4) +
           (DateTime.Now.Second * 3) +
           (DateTime.Now.Millisecond * 2);
G = P.genCoPrime(256, new Random(seed));
```

Setelah mendapatkan nilai P dan G maka dimulai perhitungan untuk variabel-variabel berikutnya. Seharusnya perhitungan dilakukan masing-masing pihak A dan B namun karena program ini akan mensimulasikan bagaimana proses yang dibutuhkan untuk pertukaran kunci maka semua perhitungan akan dilakukan.

Setelah kedua pihak memiliki nilai P dan G, pihak A dan B menentukan suatu bilangan prima XA dan XB. Sma asperti sebelumnya dalam men-*generate* suatu bilangan prima maka sebelum digunakan fungsi *genPseudoPrime* dihitung nilai seed.

Berikut ini kode implementasinya:

```
seed = (DateTime.Now.Hour * 5) +
           (DateTime.Now.Minute * 4) +
           (DateTime.Now.Second * 3) +
           (DateTime.Now.Millisecond * 2);
XA = BigInteger.genPseudoPrime(256, 50, new
           Random(seed));
```

```
seed = (DateTime.Now.Hour * 5) +
           (DateTime.Now.Minute * 4) +
           (DateTime.Now.Second * 3) +
           (DateTime.Now.Millisecond * 2);
XB = BigInteger.genPseudoPrime(256, 50, new
           Random(seed));
```

Setelah pihak A menghitung nilai XA dan pihak B menghitung nilai XB mera masing-masing menghitung nilai Y. nilai XA dan XB harus dijaga kerahasiannya karena jika nilai ini diketahui pihak penyadap maka kerahasiannya kunci tidak akan terjamin lagi.

Praktisnya pihak A akan menghitung nilai YA dan pihak B akan menghitung nilai YB dengan rumus berikut:

$$YA = G^{XA} (mod\ P)$$
$$YB = G^{XB} (mod\ P)$$

Berikut kode implementasinya:

```
YA = G.modPow(XA, P);
YB =.G.modPow(XB, P);
```

Pada praktisnya setelah pihak A dan pihak B masing-masing menghitung nilai YA dan YB, mereka akan saling bertukar nilai yang nantinya digunakan untuk menghitung kunci rahasia simetri yang akan digunakan untuk melakukan proses enkripsi ataupun dekripsi.

Setelah pihak A mendapatkan nilai YB dan pihak B mendapatkan nilai YA maka dilakukan perhitungan nilai kunci rahasian dengan rumus:

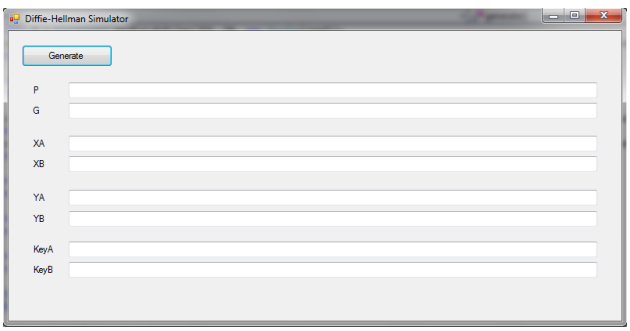
$$KeyA = YB^{XA} (mod\ P)$$
$$KeyB = YA^{XB} (mod\ P)$$

Berikut kode implementaisnya:

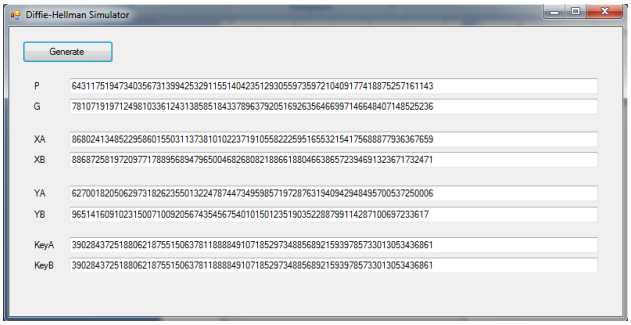
```
KeyA = YB.modPow(XA, P);
KeyB = YA.modPow(XB, P);
```

Jika proses perhitungan semua nilai berjalan dengan benar maka seharusnya nilai KeyA dan KeyB adalah sama. Karena sama maka kedua belah pihak telah memiliki sebuah nilai yang dapat digunakan untuk melakukan proses enkripsi dan dekripsi.

Dengan mengimplementasikan semua kode diatas, terbentuklah sebuah program untuk mensimulasikan Algoritma Diffie-Hellman ini. Berikut ini adalah *screenshot* dari tampilan antarmuka program.



Gambar 3



Gambar 4

Dari hasil percobaan diatas didapatkan nilai-nilai sebagai berikut :

- P = 643117519473403567313994253291155140423512930559735972104091774188752571611 43
- G = 781071919712498103361243138585184337896379205169263564669971466484071485252 36
- XA = 868024134852295860155031137381010223719105582225951655321547568887793636765 9
- XB = 886872581972097717889568947965004682680821886618804663865723946913236717324 71
- YA = 627001820506297318262355013224787447349598571972876319409429484957005372500 06
- YB = 965141609102315007100920567435456754010150123519035228879911428710069723361 7
- KeyA = 390284372518806218755150637811888849107185297348856892159397857330130534368 61
- KeyB = 390284372518806218755150637811888849107185297348856892159397857330130534368 61

PENGUJIAN DAN PERBANDINGAN

Setelah berhasil mengimplementasikan kedua algoritma diatas maka dilakukan pengujian untuk membandingkan kedua algoritma tersebut.

Kedua algoritma akan dilakukan pengujian dengan menghitung jumlah waktu yang digunakan oleh setiap algoritma untuk membangkitkan kunci. Pengujian ini dilakukan menggunakan kelas StopWatch yang ada pada pemrograman C# berikut impementasinya:

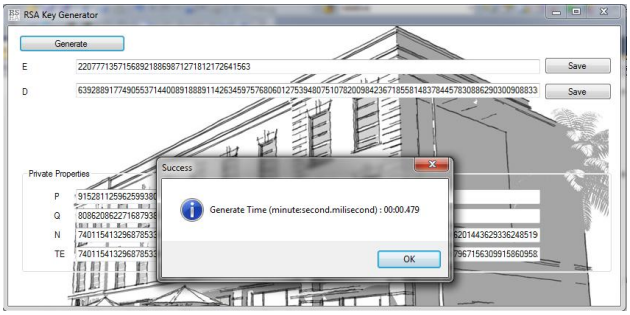
```
Stopwatch sw = new Stopwatch();
sw.Start();

//kode ada disini

sw.Stop();
TimeSpan ts = sw.Elapsed;
string elapsedTime =
        String.Format("{0:00}:{1:00}.{2:000}",
        ts.Minutes, ts.Seconds, ts.Milliseconds);
```

Dengan digunakan implementasi diatas maka dapat dihitung lama waktu yang digunakan untuk mengeksekusi kode.

Algoritma RSA

Pada Algoritma RSA untuk dapat melakukan pertukaran pesan maka dibutuhkan dua pasang kunci. Stau pasang kunci untuk pihak pertama dan satu pasang kunci lagi dimiliki oleh pihak lawan.

Maka dilakukan perhitungan waktu untuk membangkitkan dua pasang kunci, berikut *screenshot* perhitungan waktu yang digunakan:



Gambar 5

Maka dilakukan pengujian sebanyak sepuluh kali. Berikut ini hasil pengujian yang didapat:

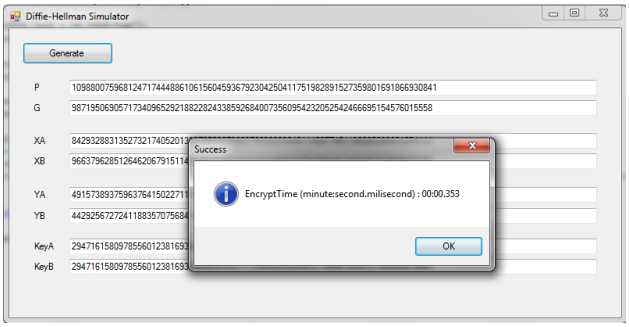
NO	WAKTU (ms)
1	365
2	431
3	451
4	407
5	396
6	579
7	472
8	378
9	432
10	428

Dari data diatas dihitung rata-rata waktu yang dibutuhkan oleh algoritma ini adalah 433.9 ms.

Algoritma Diffie-Hellman

Pada Algoritma Diffie-Hellman tidak perlu dilakukan perhitungan dua buah kunci karena kunci yang digunakan untuk proses enkripsi dan dekripsi sama.

Maka dengan menggunakan kode StopWatch seperti dijelaskan diatas maka dihitung waktu total yang digunakan untuk melakukan pertukaran kunci, berikut ini *screenshot* antarmukanya:



Gambar 6

Maka dilakukan pengujian sebanyak sepuluh kali. Berikut ini hasil pengujian yang didapat:

NO	WAKTU (ms)
1	353
2	297
3	319
4	314
5	321
6	291
7	352
8	334
9	349
10	255

Dari data diatas dihitung rata-rata waktu yang dibutuhkan oleh algoritma ini adalah 318.5 ms.

Perbandingan Waktu

Dari dua pengujian diatas didapat rata-rata waktu yang dibutuhkan oleh algoritma RSA adalah 433.9 mx sedangkan algoritma Diffie-Hellman membutuhkan rata-rata 318.5 ms. Jadi dapat dikatakan bahwa algoritma Diffie-Hellman lebih cepat dalam membangkitkan sepasang kunci.

Hal tersebut dapat terjadi karena algoritma Diffe-Hellman menggunakan perhitungan yang lebih sedikit sederhana dibandingkan algoritma RSA. Pada dasarnya setiap pihak hanya melakukan operasi *modPow* sebanyak dua kali namun pada Algoritma RSA ada dua buah perkalian bilangan BigInteger ((PxQ) dan (P-1)(Q-1)) yang membuat waktunya lebih lama sedikit dibandingkan algoritma Diffie-Hellman.

Perbandingan Keamanan

Karena pembangkitan kunci melibatkan pengiriman data antar dua pihak yang berkomunikasi maka keamanan kunci menjadi pertimbangan penting,

Keamanan kunci Algoritma RSA bergantung dari nilai P dan Q yang ditentukan pada proses pembangkitan kunci. Jika nilai N yang disebarkan ke pihak lain dapat difaktorkan menjadi nilai P dan Q maka keamanan kunci sudah tidak dapat dijamin lagi.

Keamanan kunci Algoritma Diffie-Hellman bergantung dari kerahasiaan nilai XA dan XB yang dipilih masing-masing pihak. Meskipun informasi lainnya dikirimkan melauli saluran yang tidak aman, nilai XA dan XB tidak pernah dikirimkan untuk menjamin keamanan kunci.

Perbandingan Efektifitas Kunci

Algoritma Diffie-Hellman membutuhkan pertukaran data antar dua pihak yang berkomunikasi untuk dapat membangkitkan nilai kunci. Jika ingin melakukan komunikasi dengan pihak lain maka dilakukan kembali proses pertukaran kunci dari awal.

Sedangkan Algoritma RSA, untuk membangkitkan sepasang kunci publik dan kunci privat tidak dibutuhkan pertukaran data dengan pihak lawan, setelah didapatkan kunci publik, baru kunci publik ini dikirimkan ke pihak lawan. Jika ingin berkomunikasi dengan pihak lain maka cukup dengan mengirimkan kunci publik yang telah dibuat sebelumnya tanap harus melakukan proses dari awal.

KESIMPULAN

Dari pengujian dan perbandingan diatas didapat bahwa Algortima Diffie-Hellman lebih baik dalam waktu eksekusi yang dibutuhkan. Jadi algoritma ini sangat baik dalam pertukaran komunikasi untuk dua pihak yang ingin berkomunikasi secara aman.

Pada perbandingan keamanan, kedua algoritma samam-sama memiliki kerentanan dalam algoritmanya masing-masing. Tapi Kerentananan ini dapat diatasi dengan memilih bilangan-bilangan prima yang besar sehingga sulit untuk dicari faktornya ataupun dilakukan operasi matematika terhadapnya. Pengguna juga harus mampu menjaga kunci yang telah dihasilkan dari pihak-pihak yang tidak berhak.

Pada perbandingan Efektifitas, Algoritma RSA jauh lebih baik dari algoritma Diffie-Hellman. Untuk setiap pihak yang akan melakukan komunikasi cukup sekali saja membangkitkan spesang kunci publik dan private yang nantinya kunci publik tersebut dapat disebarkan ke siapapun.

\

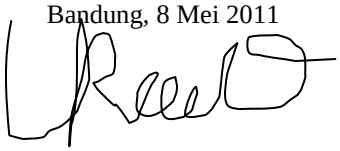
REFERENSI

- Saloma, Arto. *Public-Key Cryptography*. 1996. Springer
- Das, Abhijit., C. E. Veni Madhavan. *Public-Key Crypthography : Teroy And Practice*. 2009. Pearson Education India.
- Mollin, Richard A. *RSA and public-key cryptography*. 2003. Chapman & Hall/CRC.
- http://www.postdiluvian.org/~seven/diffie.html tanggal akses 8 Mei 2011 pukul 19.45.
- http://library.thinkquest.org/C0126342/dh.htm tanggal akses 8 Mei 2011 pukul 20.03.

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, 8 Mei 2011



Yudi Retanto 13508085