Algoritma Kriptografi Kunci Publik

Dengan Menggunakan Prinsip *Binary tree*

Dan Implementasinya

Hengky Budiman – NIM : 13505122

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15122@students.if.itb.ac.id

ABSTRAK

Algoritma kriptografi kunci publik merupakan algoritma kriptografi yang menggunakan sepasang kunci yang berbeda untuk mengenkripsi dan mendekripsinya. Salah satu kunci tersebut disebarkan kepada umum (kunci publik) dan kunci yang lain dijaga kerahasiaannya (kunci privat). Umumnya, proses enkripsi menggunakan kunci publik dan proses dekripsi menggunakan kunci privat, tergantung dari protokol yang digunakan. Karena menggunakan sepasang kunci yang berbeda, algoritma kriptografi kunci publik terkadang disebut juga dengan algoritma kriptografi asimetris.

Sudah banyak algoritma kriptografi kunci publik yang telah ditemukan dan dipelajari. Masing – masing algoritma tersebut memiliki kelebihan dan kekurangannya masing – masing. Pada makalah ini, akan dibahas mengenai suatu algoritma kunci publik baru yang penulis rancang sendiri dengan menggunakan prinsip pohon biner.

Kata kunci: kriptografi, kunci publik, kunci asimetris, pohon biner

1. PENDAHULUAN

Kriptografi kunci publik adalah varian baru dari metode kriptografi konvensional (yang masih menggunakan algoritma kunci simetri). Dengan menggunakan metode ini, banyak hal yang dulu tidak bisa dilakukan sekarang menjadi mungkin dilakukan, seperti misalnya autentifikasi pesan, tanda tangan digital, dan lain – lain. Kriptografi kunci publik juga merupakan prinsip dari teknologi fundamental yang banyak digunakan di seluruh dunia seperti misalnya pada standard jaringan internet TLS (*Transport Layer Security*), PGP, dan GPG.

Kriptografi kunci publik awalnya diciptakan karena pada sistem kriptografi konvensional, banyak terdapat kesulitan seperti misalnya kesulitan dalam memberikan kunci kepada orang yang hendak diberi pesan. Pencetus dari sistem kunci publik adalah James H. Ellis, Clifford Cocks, dan Malcolm Williamson di GCHQ, UK pada tahun 1970[2]. Penemuan ini dikembangkan dan dirahasiakan hingga tahun 1977 sebelum diberitakan ke publik.

2. DASAR TEORI

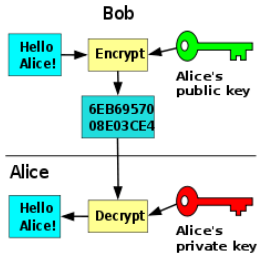
A. KRIPTOGRAFI KUNCI PUBLIK

Kriptografi kunci publik menggunakan pasangan kunci asimetris, yaitu kunci yang berbeda untuk melakukan enkripsi dan dekripsi. Kedua kunci tersebut dinamakan kunci privat (kunci yang kerahasiaannya dijaga) dan kunci publik (kunci yang disebarkan ke umum). Pesan akan dienkripsi dengan menggunakan kunci publik, dan hanya bisa dapat didekripsi dengan menggunakan kunci privat yang berkoresponden dengannya. Kekuatan dari metode ini adalah kesulitan untuk mendapatkan kunci privat bila hanya diketahui kunci publiknya saja. Hal ini juga dinyatakan oleh William Stanley Jevons pada bukunya yang berjudul *The Principles of Science : A Treatise on Logic and Scientific Method*, yang diterbitkan pada tahun 1890 [3]. Pada

bukunya tersebut, dia menjelaskan situasi – situasi di mana suatu operasi dikatakan relatif mudah, tetapi invers dari operasi tersebut sangat sulit, contohnya adalah menemukan faktor prima dari sebuah bilangan besar.

Kriptografi konvensional (kunci simetri) bisa dianalogikan seperti pada kotak pesan di tempat umum. Untuk saling bertukar pesan, Alice dan Bob memasukkan pesan mereka ke kotak tersebut. Kotak tersebut dikunci dengan menggunakan sebuah kunci gembok, dan Alice serta Bob masing – masing harus memiliki kunci tersebut. Sedangkan pada kriptografi kunci publik, harus terdapat 2 buah kunci gembok dengan masing – masing kunci untuk setiap gembok. Untuk menyampaikan pesan kepada Bob, Alice akan menggembok kotak itu dengan gembok Bob sehingga hanya Bob yang memiliki kuncinya yang dapat membukannya. Begitu pula sebaliknya bila Bob ingin menyampaikan pesan kepada Alice.

Skema dari protokol penggunaan kriptografi kunci publik untuk mengenkripsi pesan adalah sebagai berikut:



Gambar 1. Skema Enkripsi – Dekripsi Pesan

Penjelasan dari skema tersebut adalah:

1. Bila Bob ingin mengirimkan pesan kepada Alice, maka Bob akan mengenkripsi pesan tersebut dengan menggunakan kunci publik milik Alice.
2. Pesan text cipher kemudian dikirim ke Alice. Bila pesan disadap di sini, maka penyadap tetap tidak dapat mengetahui isi pesan tersebut karena penyadap tidak mengetahui kunci privat milik Alice.
3. Alice mendekripsi *ciphertext* dengan kunci privatnya dan mendapatkan kembali isi pesan semula (*plaintext*).

Keuntungan dari penggunaan kriptografi kunci publik adalah:[5]

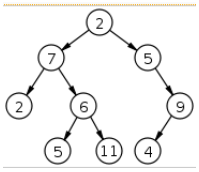
1. Tidak ada kebutuhan untuk mendistribusikan kunci privat sebagaimana pada sistem kriptografi simetri.
2. Kunci publik dapat dikirim ke penerima melalui saluran yang sama dengan saluran yang digunakan untuk mengirim pesan. Meskipun saluran tersebut tidak aman, hal ini tidak mengancam keamanan pesan karena kunci publik tidak dapat digunakan untuk mendekripsi pesan.
3. Kita bisa menggunakan skema kriptografi kunci publik untuk autentifikasi pesan.

Kerugian dari penggunaan kriptografi kunci publik adalah:[2]

1. Relatif lebih tidak aman daripada algoritma kunci simetri, karena pada algoritma kunci publik, selalu ada hubungan antara kunci privat dan kunci publik sehingga jenis serangannya rata – rata memiliki kompleksitas yang lebih kecil daripada *brute force*. Sebaliknya pada kriptografi kunci simetri, tidak ada hubungan seperti ini, sehingga serangannya relatif lebih sulit.
2. Untuk menutupi kekurangan nomor 1 tersebut, biasanya digunakan kunci dengan ukuran yang besar.
3. Umumnya algoritma kunci publik memerlukan komputasi yang besar dibandingkan dengan algoritma kunci simetri.

B. POHON BINER

Pohon biner merupakan sebuah struktur data berupa pohon, di mana setiap simpul memiliki maksimum 2 anak, yang umumnya disebut anak kiri dan anak kanan[7]. Pohon biner biasanya digunakan sebagai struktur data pada operasi aritmatika atau untuk mengoptimalkan pencarian. Pada makalah ini, penggunaan pohon biner digunakan dalam algoritma kriptografi kunci publik untuk mengoptimalkan pencarian plainteks.



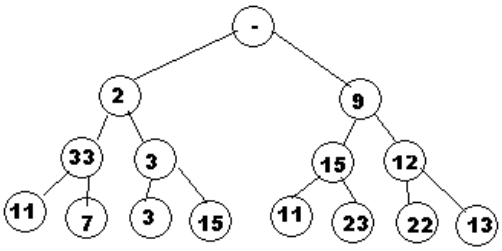
Gambar2. Contoh Pohon Biner

3. PENJELASAN KRIPTOSISTEM *BINARY TREE*

A. POHON BINER BIASA

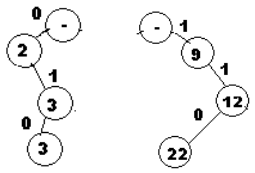
Kriptosistem *Binary tree* bekerja dengan cara seperti berikut:

1. Buat sebuah pohon biner n tingkat dengan setiap *node* memiliki nilai tertentu. Misalnya pohon dengan 3 tingkat seperti berikut, akar tidak dihitung.



Gambar 3. Contoh pohon biner yang digunakan

2. Uraikan *plaintext* yang hendak dienkirpsi menjadi bit – bit yang bersesuaian. Misalnya *plaintext* "KRP" dalam biner bersesuaian dengan 01001011 01010010 01010000
3. Lakukan operasi penjumlahan berdasarkan biner tersebut dan pohon biner yang tersedia. Bit 0 menandakan penjumlahan dengan anak sebelah kiri, bit 1 menandakan penjumlahan dengan anak sebelah kanan. Contoh:
   010 = 2 + 3 + 3 = 8
   110 = 9 + 12 + 22 = 43



Gambar 4. Contoh Enkripsi Per Bit

4. Bila langkah 3 di atas diselesaikan, maka kita akan mendapatkan *ciphertext* 8, 8, 43, 47, 42, 42, 8, 46 yang bisa dikembalikan ke karakter ASCII nya

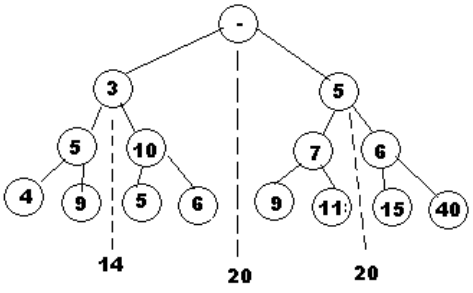
Dari cara enkripsi di atas, bisa kita perhatikan bahwa terdapat sebuah syarat di sini, yaitu tidak boleh ada kombinasi bit *plaintext* yang berbeda tetapi menghasilkan *ciphertext* yang sama, sehingga proses enkripsi tetap satu ke satu.

Bila kita ingin mendekripsinya, maka kita harus memeriksa semua kemungkinan yang ada, yang dalam kasus di atas adalah sebanyak $2^3 = 8$. Bila kita menggunakan pohon dengan kedalaman n (akar tidak dihitung), maka banyaknya kemungkinan yang ada adalah sebanyak 2^n. Hal ini berarti seiring dengan membesarnya n, solusi secara *brute force* dari kriptosistem ini akan membesar secara eksponensial dan tidak dapat dipcahkan dalam orde waktu polinomial. Hal inilah yang menjadi kekuatan utama algoritma kriptosistem *binary tree*.

Namun demikian, banyaknya jumlah kemungkinan ini juga membuat penerima sulit untuk mendekripsi *ciphertext* menjadi *plaintext*. Salah satu solusi adalah menghitung dulu semua kemungkinan yang ada dan kemudian menyimpannya di sebuah array atau di sebuah *database*. Namun, hal ini membutuhkan *resource* yang sangat mahal karena membutuhkan banyak *space memory*. Selain itu setiap kali terjadi perubahan kunci (nilai – nilai pada simpul), maka harus dilakukan perhitungan ulang semua kemungkinan. Hal ini tentunya sangat tidak efisien sehingga harus dicari jalan lain untuk mendekripsinya.

B. POHON BINER TERURUT

Salah satu solusi dari masalah ini adalah dengan membuat pohon yang memiliki sifat – sifat tertentu sehingga memudahkan proses dekripsinya. Solusi yang ditawarkan penulis adalah dengan membuat pohon biner sedemikian rupa sehingga penjumlahan *node* – *node* dari anak sebelah kiri selalu lebih kecil dari penjumlahan *node* – *node* dari anak sebelah kanan (pohon yang terurut). Contohnya adalah sebagai berikut:



Gambar 5. Contoh Pohon Biner Terurut

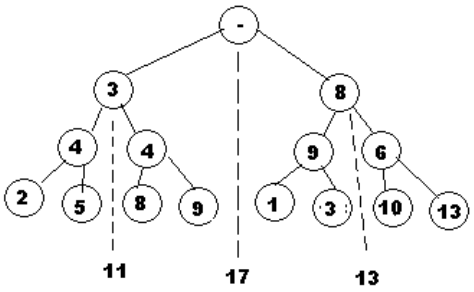
Garis putus – putus dengan nilai 20 pada gambar menunjukkan bahwa semua kemungkinan penjumlahan dari anak sebelah kanan selalu lebih besar dari 20, dan sebaliknya pada anak sebelah

kiri selalu lebih kecil dari 20. Contohnya bila *plaintext*-nya adalah 101, maka *ciphertext*-nya adalah 5+7+11 = 23 > 20. Dan bila *plaintext*-nya adalah 011, maka *ciphertext*-nya 3 + 10 + 6 = 19 ≤ 20. Dengan kata lain, semua *plaintext* dengan awalan bit 1 akan lebi besar dari 20, dan sebaliknya *plaintext* dengan awalan 0 akan lebih kecil dari 20.

Dengan pohon terurut seperti ini, proses dekripsi menjadi lebih mudah. Misalnya kita ingin mendekripsi nilai 23, prosesnya adalah sebagai berikut:

- Kita bandingkan nilai 23 dengan 20, karena 23 lebih besar, berarti bit ke 1 = "1". Kita kemudian kurangi nilai 43 dengan nilai *node* sebelah kanan, yaitu 5. Kita dapat 23 – 5 = 18.
- Kita bandingkan nilai 18 dengan 20, karena 18 lebih kecil dari 20, maka bit ke 2 = "0". Kemudian kita kurangi nilai 18 dengan 7. Kita dapatkan 18 – 7 =11.
- Nilai 11 adalah nilai dari anak *node* sebelah kanan, jadi bit ke 3 = "1".
- Kita dapatkan *plaintext* awalnya adalah "101".

Dengan metode seperti ini, proses dekripsi tentunya menjadi lebih mudah. Namun demikian, terdapat masalah lain yang harus dipecahkan, yaitu kita harus menyimpan batas – batas nilai di setiap simpul. Jumlah nilai – nilai batas ini ada sebanyak 2^{n-1} -1. Contohnya untuk n = 3 di atas, jumlah batasnya ada 2^{3-1} -1 = 3 batas. Meskipun sudah lebih sedikit daripada menyimpan semua kemungkinan nilai yang ada yaitu 2^n, hal ini tetap membutuhkan *resource* yang mahal. Salah satu solusi untuk memecahkan masalah ini adalah membuat pohon biner yang sedemikian rupa sehingga batas – batasnya memiliki pola tertentu, misalnya batas – batas tersebut mengikuti pola bilangan prima 11,13,17 seperti berikut:



Gambar 6. Contoh Pohon Biner Terurut Dengan Batas Nilai Berpola

Dengan batas yang memiliki pola seperti tersebut, kita tidak perlu menyimpan nilai – nilai dari batas tersebut. Kita dapat menghitungnya ketika sedang menjalankan proses dekripsi.

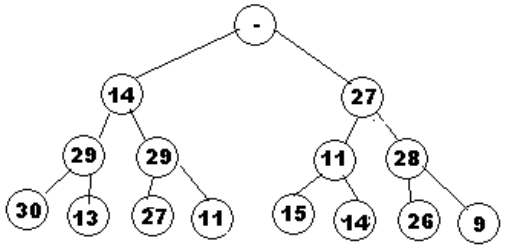
C. PENGUBAHAN POHON BINER TERURUT MENJADI POHON BINER BIASA

Proses dekripsi yang dijelaskan di atas mirip dengan proses *search* pada pohon biner. Proses ini merupakan proses yang cepat dengan kompleksitas waktu O(n). Bila kita menggunakan pohon biner ini, para penyerang tentu saja dengan mudah dapat mendekripsi *ciphertext* yang dihasilkan. Karena itulah, diperlukan sebuah mekanisme untuk mengubah pohon biner terurut menjadi pohon biner biasa.

Cara yang dilakukan adalah sebagai berikut:

- Tentukan terlebih dahulu pohon biner terurut yang hendak diubah.
- Kalikan setiap *node* di dalam pohon biner tersebut dengan n modulo m. Modulus m harus merupakan angka yang lebih besar daripada nilai terbesar *ciphertext* yang dapat dihasilkan, sedangkan pengali n harus merupakan bilangan yang relatif prima dengan m (tidak mempunyai faktor pembagi yang sama dengan m).
- Pohon yang dihasilkan merupakan pohon biner biasa. Pohon ini dijadikan sebagai kunci publik.
- Pohon biner terurut semula, nilai n, dan nilai m disimpan sebagai kunci privat untuk proses dekripsi.

Contoh pohon biner biasa yang dihasilkan dari pohon biner terurut bisa kita lihat pada gambar berikut:



Gambar 7. Pohon Biner Biasa yang Diperoleh dari Pohon Biner Terurut.

Gambar 7 merupakan pohon biner biasa yang diubah dari pohon biner terurut pada gambar 6 dengan kunci m = 31 dan kunci n = 15.

D. PROSES ENKRIPSI DAN DEKRIPSI

Misalnya kita ingin mengenkripsi *plaintext* 110 010 001, maka dengan menggunakan pohon biner biasa (kunci publik) pada gambar 7. Kita dapatkan *ciphertext*-nya adalah: 81, 70, dan 56.

Untuk mendekripsi *ciphertext* tersebut menjadi *plaintext* kembali, ada beberapa hal yang harus dilakukan, yaitu:

- Menghitung nilai invers n terhadap m, yaitu 29, karena (15 * 29) mod 31 = 1
- Mengalikan *ciphertext* dengan Inverse N mod M, didapat nilai 24, 15, dan 12
- Mendekripsinya dengan menggunakan pohon biner terurut (kunci privat), kita dapatkan:
  24 = 8 + 6 + 10 = "110"
  15 = 3 + 4 + 9 = "010"
  12 = 3 + 4 + 5 = "001"

4. ANALISIS KEAMANAN KRIPTOSISTEM *BINARY TREE* DAN PERBANDINGANNYA DENGAN ALGORITMA LAIN

Kelebihan dari kriptosistem ini adalah:

- Waktu pemrosesan yang tergolong cepat, apabila dibandingkan dengan algoritma kriptografi kunci publik lain seperti RSA. Waktu yang dibutuhkan oleh algoritma ini kurang lebih sama dengan algoritma *knapsack*, yaitu kurang lebih 100 kali lebih cepat daripada algoritma RSA [1]

- Tingkat keamanan yang cukup tinggi. Bila dibandingkan dengan algoritma *knapsack*, algoritma *Binary tree* memiliki tingkat keamanan yang lebih baik. Hal ini dikarenakan pada serangan untuk algoritma *knapsack*, meskipun tidak menggunakan teknik *brute force*, tetapi penyerang tetap harus menguji beberapa nilai yang memiliki kemungkinan tinggi sebagai kunci. Untuk mengujinya tersebut, penyerang cukup menjalankan operasi dengan kompleksitas waktu O(n) untuk setiap kunci karena barisan *knapsack* yang liniar. Sedangkan pada algoritma pohon biner, penyerang tetap harus menjalankan operasi dengan kompleksitas $O(n^2)$ untuk setiap kunci. Hal ini dikarenakan *node* pada pohon biner jumlahnya meninkat eksponensial.

Kekurangan dari kriptosistem ini adalah:

- *Memory* yang digunakan relatif besar, yaitu 2^n -1 untuk menyimpan nilai – nilai pada *node* pohon biner. Ukuran ini cukup besar bila dibandingkan dengan ukuran *memory* sebesar n pada algoritma *knapsack* biasa.

- Penyerang bisa menggunakan sifat pohon biner terurut di mana jumlah nilai anak sebelah kiri selalu lebih kecil daripada jumlah nilai anak sebelah kanan. Namun demikian, hal ini bukan merupakan celah yang sangat besar karena untuk menguji semua jumlah tersebut, maka penyerang harus menghitung semua kemungkinan kombinasi bit yang ada dulu. Hal ini sama saja dengan menyerang secara *brute force*.

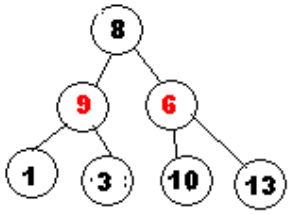
- Pembuatan pohon biner terurut sebagai kunci privat cukup sulit. Terutama untuk pohon biner dengan kedalaman yang besar. Hal ini dikarenakan jumlah *node* pada pohon biner yang meningkat secara eksponensial setiap kali kedalaman bertambah.

Beberapa hal yang bisa dilakukan untuk meningkatkan kekuatan dari algoritma kunci publik *Binary Tree* ini adalah:

- Bilangan M (modulo) yang dipakai sebaiknya merupakan bilangan yang relatif prima dengan semua bilangan pada *node* pohon biner. Hal ini dibuat agar hasil perkaliannya tidak ada yang menghasilkan bilangan 0 atau bilangan – bilangan yang memiliki kelipatan yang sama sehingga mudah ditebak. Penggunaan bilangan M yang merupakan bilangan prima lebih dianjurkan.

- Meskipun ketentuannya adalah jumlah nilai anak sebelah kiri selalu lebih kecil daripada jumlah nilai anak sebelah kanan, tetapi tidak berarti nilai *node* sebelah kiri harus lebih kecil daripada nilai *node* sebelah kanan. Hal ini dikarenakan karena penjumlahan tersebut

mempertimbangkan semua *descendant* (anak – dari anak) *node* tersebut juga. Contohnya adalah sebagai berikut:



Gambar 8. Contoh Nilai Kiri Lebih Besar

Hal ini mengakibatkan penyerang atau kriptanalis tidak bisa memanfaatkan celah ini untuk membobol kriptosistem ini.

- Bisa terdapat 2 *node* atau lebih yang memiliki nilai yang sama. Hal ini dapat menyebabkan kesulitan pada kriptanalis pada kasus tertentu. Tidak seperti pada algoritma kriptosistem *knapsack* yang melarang penggunaan bilangan yang sama lebih dari 1 kali.

5. Implementasi Program Dan Contoh Hasil

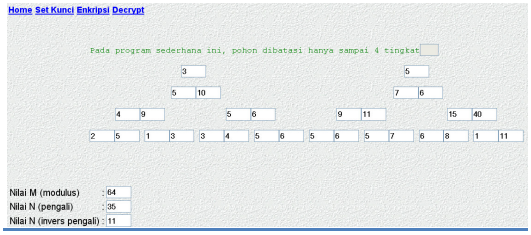
Penulis telah membuat aplikasi yang dapat mengilustrasikan cara kerja kriptosistem ini secara real. Penulis membuat aplikasi ini dalam bentuk *website*. Pemilihan format *website* didasarkan dengan alasan:

- Format *website* bisa diakses dengan gampang di mana saja dan kapan saja selama ada internet.
- Format *website* bisa dibuka dengan *browser* yang umumnya tersedia di semua komputer. Tidak seperti format program lain yang membutuhkan *set up* dan *install* program seperti aplikasi *Flash, .Net*, atau *Java*.

Website yang dibuat dapat diakses melalui alamat students.itb.ac.id/~if15122/binarytree. Berikut ini adalah *screenshot* dari aplikasi tersebut.

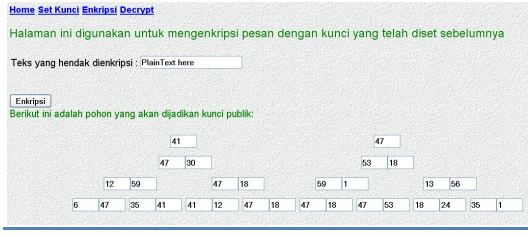
Gambar 9. *Screenshot* Halaman Muka

Halaman ini merupakan halaman muka yang diakses pertama kali. Halaman ini berisi informasi pemakaian singkat dan juga pengenalan singkat.



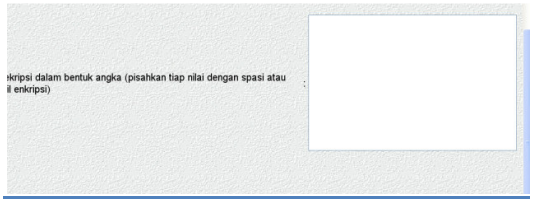
Gambar 10. *Screenshot* Halaman Mengisi Kunci

Dengan memasukkan semua kunci yang dibutuhkan, maka aplikasi akan meng-*generate* secara otomatis kunci publik (pohon biner biasa) yang dibutuhkan.



Gambar 11. *Screenshot* Halaman Untuk Mengenkripsi

Halaman ini digunakan untuk mengenkripsi pesan yang dimasukkan pengguna. Sebagai contoh, masukan "kriptografi" akan menghasilkan *ciphertext* 136 154 107 182 136 177 107 106 107 159 136 122 136 107 107 182 136 147 136 136 136 177.



Gambar 12. *Screenshot* Halaman Untuk Mendekripsi

Halaman ini digunakan untuk mendekripsi *ciphertext* menjadi *plaintext* kembali. Contoh masukan 136 154 107 130 136 156 136 188 136 177 182 106 107 106 107 130 136 182 136 96 136 177 136 154 akan menghasilkan *plaintext* "kunci publik".

6. DAFTAR PUSTAKA

[1] Esfahbod, Behdad, "Knapsack Cryptosystems", http://behdad.org/download/Presentations/knapsack/knapsack.ppt, waktu akses : 16 Mei 2009 Pukul 18.21 WIB.

[2] Menezes, Alfred J, dkk, "Handbook Of Applied Cryptography", CRC Press, 1996.

[3] MS, Anoop, "Public Key Cryptography", Tata Elxsi Ltd, India, 2007.

[4] Munir, Rinaldi, "Diktat Kuliah IF2153 Matematika Diskrit", Program Studi Teknik Informatika Institut Teknologi Bandung, 2006.

[5] Munir, Rinaldi. "*Diktat Kuliah IF5054 Kriptografi*", Program Studi Teknik Informatika Institut Teknologi Bandung, 2006.

[6] NP-Problem, http://mathworld.wolfram.com/NP-Problem.html, waktu akses: 16 Mei 2009 Pukul 17.32 WIB.

[7] What is Binary Tree, http://searchsqlserver.techtarget.com/sDefinition/0,,sid87_gci509290,00.html, waktu akses: 18 Mei 2009 Pukul 23.43 WIB.