Penambahan Permutasi pada Knapsack Cipher

Gregorius Ronny Kaluge / 13508019
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
ronny.kaluge@gmail.com, if18019@students.if.itb.ac.id, miliknya_ronny@yahoo.co.id

***Abstrak*—Algoritma kriptografi kunci publik adalah algoritma yang sangat berguna dalam kehidupan sehari-hari. Dengan algoritma ini, kunci bisa dikirimkan dengan mudah, karena tidak perlu saluran pengiriman yang aman. ada umumnya algoritma kunci publik membutuhkan waktu komputasi yang lama, baik untuk dekripsi maupun enkripsi. Untungnya, ada algoritma kunci publik yang cukup sederhana dan tidak butuh waktu komputasi yang terlalu lama. Salah satu algoritma tersebut adalah Knapsack Cipher. Sayangnya, algoritma ini sudah dinyatakan tidak aman karena bisa diserang dengan algoritma lain yang kompleksitasnya polinomial. Oleh karena itu, penulis ingin mencoba memperkuat algoritma ini dengan menambahkan langkah-langkah khusus pada tahap enkripsi dan dekripsinya.**

***Kata kunci*— Kriptografi kunci publik, Knapsack Cipher, Polinomial**

I. PENDAHULUAN

Kriptografi memiliki sejarah yang panjang. Sejak ribuan tahun silam, kriptografi telah digunakan. Sudah banyak orang yang mendalami cabang ilmu ini sampai dengan sekarang.

Pada zaman dahulu, kriptografi yang digunakan adalah kriptografi kunci simetri. Artinya kunci yang digunakan untuk enkripsi = kunci yang digunakan untuk dekripsi. Agar pesan rahasia tidak bocor, kunci rahasia tersebut harus bisa disampaikan ke pihak penerima melalui jalur yang aman.

Karena kriptografi kunci simetri ini repot digunakan (karena butuh saluran yang aman), orang-orang mulai mengembangkan algoritma kunci publik. Dengan algoritma ini, kita tidak perlu mengirim kunci publik lewat saluran yang aman. Meskipun kunci publik ini diketahui oleh penyerang, pesan rahasia tidak bisa dibobol karena perlu didekripsi dengan kunci privat.

Penemuan kriptografi kunci publik tentunya sangat membantu dalam pengiriman pesan rahasia. Namun, algoritma ternyata ini memiliki kelemahan. Kelemahan pertama adalah, cipherteks yang dihasilkan bisa menjadi beberapa kali lipat lebih panjang daripada plainteks. Yang kedua adalah pembuatan kuncinya biasanya cukup merepotkan. Yang ketiga adalah biasanya algoritma ini membutuhkan waktu yang lama untuk mengenkripsi ataupun mendekripsi pesan.

Oleh karena itu, muncullah algoritma Knapsack Cipher. Algoritma ini membutuhkan waktu komputasi yang cukup singkat dan cukup sederhana untuk diimplementasikan. Selain itu, proses pembuatan kuncinya cukup sederhana dan tidak terlalu lama.

Sayangnya algoritma ini sudah bisa dibobol berkat ditemukannya algoritma LLL (Lenstra-Lenstra-Lovász). Sejak saat itu, orang-orang sudah mencoba memperbaiki algoritma ini dan ternyata semuanya masih bisa dibobol juga, kecuali algoritma Chor-Rivest Knapsack Cipher.

II. KNAPSACK

Permasalahan Knapsack adalah permasalahan optimisasi kombinatorial. Diberikan kumpulan benda, masing-masing memiliki berat dan nilai, tentukan benda mana saja yang akan diambil sehingga total beratnya <= suatu batas nilai (biasanya kapasitas tas) dan nilai yang sebesar-besarnya. Nama dari problem ini diperoleh dari masalah yang dihadapi seseorang saat berhadapan dengan tas yang ukurannya terbatas namun harus diisi dengan benda yang paling berharga atau berguna.

III. MERKLE-HELLMAN KNAPSACK CIPHER

Ide algoritma ini berasal dari persoalan Knapsack. Diketahui himpunan angka A dan sebuah bilangan b, carilah himpunan bagian dari A yang jika dijumlahkan = b. Umumnya, persoalan ini adalah NP-Complete, namun pada algoritma ini, kunci yang digunakan adalah super-increasing knapsack. Super-increasing knapsack adalah knapsack yang setiap elemennya > jumlah elemen-elemen sebelumnya.

Karena sifat super-increasing tersebut, persoalan knapsack ini dapat diselesaikan dengan Greedy dalam kompleksitas polinomial. Untuk menyembunyikan sifat super-increasing tersebut, diperlukan 2 buah bilangan m dan n. m harus lebih besar dari total nilai knapsack dan n harus relatif prima terhadap m.

Setelah diketahui m dan n, setiap elemen knapsack dikalikan dengan n lalu di-modulus dengan m. Akibatnya, knapsack yang tadinya super-incereasing akan menjadi tidak terurut dan tidak terlihat super-increasing lagi. Knapsack hasil perkalian inilah yang disebarkan sebagai kunci publik. Sedangkan kunci privat yang perlu disimpan hanyalah X yang memenuhi $n \cdot X \equiv 1 \pmod{m}$.

Untuk mengenkripsi pesan, bagi pesan tersebut ke dalam blok-blok berukuran E bit (E adalah ukuran knapsack). Jika bit-nya kurang, tambahkan 0 sebagai bit pengganjal. Hasil enkripsi untuk suatu blok B yang elemennya $b_1..b_n$ adalah sebuah bilangan yang merupakan

$b_1.e_1 + b_2.e_2 + \ldots + b_n.e_n$. Setelah itu, bilangan-bilangan hasil enkripsi tersebut digabungkan menjadi sebuah cipherteks yang kemudian dikirimkan ke pemilik kunci privat.

Sang pemilik kunci privat akan mengalikan setiap elemen kunci publik dengan X dan di-modulo m untuk memperoleh knapsack super increasing yang asli. Selanjutnya, pesan enkripsi akan dikalikan dengan X dan di-modulo dengan m juga. Hasil pengubahan pesan tersebut adalah sebuah bilangan yang merupakan hasil penjumlahan dari satu atau beberapa elemen knapsack super-increasing. Untuk mengetahui elemen mana saja yang perlu ditambahkan untuk menghasilkan bilangan tersebut, bisa digunakan algoritma Greedy yang kompleksitasnya O(n).

IV. LENSTRA–LENSTRA–LOVÁSZ LATTICE BASIS REDUCTION ALGORITHM

Algoritma LLL yang mereduksi lattice dalam waktu polinomial. Algoritma ini ditemukan oleh Arjen Lenstra, Hendrik Lenstra dan László Lovász pada tahun 1982. Diberikan input sebuah basis $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_d\}$, dengan koordinar n dimensi, untuk lattice L pada R^n dengan d <= n, algoritma LLL akan menemukan lattice LLL-tereduksi dalam waktu $O(d^5 n \log^3 B)$. di mana nilai B adalah b_i yang terpanjang dalam norma Euclidean.

Kegunaaan asli dari algoritma ini adalah untuk memfaktorkan polinomial dengan koefisien rasional menjadi polinomial yang tidak bisa direduksi lagi, untuk mencari hampiran dari bilangan real, dan untuk menyelesaikan permasalahan program linear pada dimensi yang tetap.

Saat ini, algoritma Lenstra memiliki banyak kegunaaan, antara lain memecahkan knapsack cipher, RSA dengan syarat tertentu, NTRUEncrypt, dan sebagainya.

V. ALGORITMA ENKRIPSI

Algoritma enkripsi yang diusulkan oleh penulis adalah sebagai berikut :

1. Diketahui super-increasing knapsack dengan E elemen serta nilai m dan n.
2. Hitung nilai X yang memenuhi $n \cdot X \equiv 1 \pmod m$.
3. Acak posisi setiap elemen pada knapsack.
4. Kalikan setiap elemen knapsack dengan n lalu di-modulo dengan m.
5. Knapsack yang telah berubah tersebut akan digunakan sebagai kunci publik. Sementara itu, X dan m akan digunakan sebagai kunci privat.
6. Langkah enkripsi yang digunakan sama dengan enkripsi pada algoritma Merkle-Hellman Knapsack.
7. Untuk dekripsi, kalikan setiap elemen kunci publik dengan X lalu di-modulo dengan m.
8. Urutkan knapsack hasil perkalian tersebut namun, jangan lupa menyimpan posisi setiap elemen sebelum diurutkan.
9. Gunakan algoritma Greedy pada knapsack cuper-increasing untuk mengetahui nilai bit-bit pada pesan asli.
10. Setelah diperoleh hasil konversi cipherteks ke bit, atur susunan bit agar sesuai dengan posisi awal setiap elemen knapsack sebelum diurutkan. Setelah itu, akhirnya diperoleh pesan yang sesungguhnya.

Untuk memperjelas algoritma ini, bisa digunakan contoh berikut :
Diketahui :
Knapsack : {2, 3, 6, 13, 27, 52}
m = 105
n = 31

Proses membuat kunci publik :

1. Hitung nilai X yang memenuhi $31 \cdot X \equiv 1 \pmod{105}$. Dari situ, diperoleh X = 61.
2. Kalikan setiap elemen knapsack dengan n lalu modulo-kan dengan m menjadi :
   {62, 93, 81, 88, 102, 37}
3. Setelah diperoleh knapsack hasil perkalian tersebut, acak posisi elemen pada knapsack, misalnya menjadi :
   {37, 88, 62, 81, 102, 93}
4. Knapsack acak tersebut akan disebarkan sebagai kunci publik, sementara itu kita cukup menyimpan nilai X dan m beserta kunci publik.

Proses membuat cipherteks :

1. Misalnya pesan asli : abc
2. Ubah setiap karakter pada pesan tersebut menjadi 8 bit, diperoleh :
   01100001 01100010 01100011
3. Potong pesan tersebut menjadi blok-blok berukuran 6 bit (karena knapsack yang digunakan berukuran 6), diperoleh :
   011000 010110 001001 100011
4. Ubah setiap blok tersebut menjadi blok cipherteks.
   Blok I : 88+62 = 150
   Blok II : 88+81+102 = 271
   Blok III : 62+93 = 155
   Blok IV : 37+102+93 = 232
5. Jadi cipherteks yang dihasilkan adalah :
   150 271 155 232

Proses dekripsi :

1. Diketahui cipherteks : 150 271 155 232
2. Kalikan setiap elemen pada knapsack dengan 61 lalu modulo dengan 105 sehingga diperoleh :
   {52, 13, 2, 6, 27, 3}
3. Urutkan elemen-elemen knapsack :
   {2, 3, 6, 13, 27, 52}
   Posisi awal : {3, 6, 4, 2, 5, 1}

4. Kalikan setiap blok cipherteks dengan 61 lalu modulo dengan 105 sehingga pesan menjadi :
   15 46 5 82
5. Dengan menggunakan algoritma Greedy, diperoleh bit-bit sebagai berikut :
   100100 001110 110000 010011
6. Namun, karena knapsack yang digunakan untuk kunci publik memiliki urutan elemen yang berbeda, bit-bit di atas perlu disusun kembali. Dengan menggunakan nilai yang tersimpan di array posisi awal, kita dapat mengubah bit ke-i dalam suatu blok menjadi bit ke-posisi[i] untuk i dari 1..E. Sehingga diperoleh :
   011000 010110 001001 100011
   Hasil yang diperoleh ini merupakan plainteks yang dikirim, jika diubah menjadi karakter, akan diperoleh : abc.

VI. IMPLEMENTASI

Penulis telah mencoba mengimplementasikan algoritma ini dalam bentuk aplikasi C# dan menggunakan .NET Framework 4. Program tidak akan ditampilkan di makalah ini karena terlalu panjang, namun jika ada yang ingin memintanya, bisa menghubungi penulis melalui email.

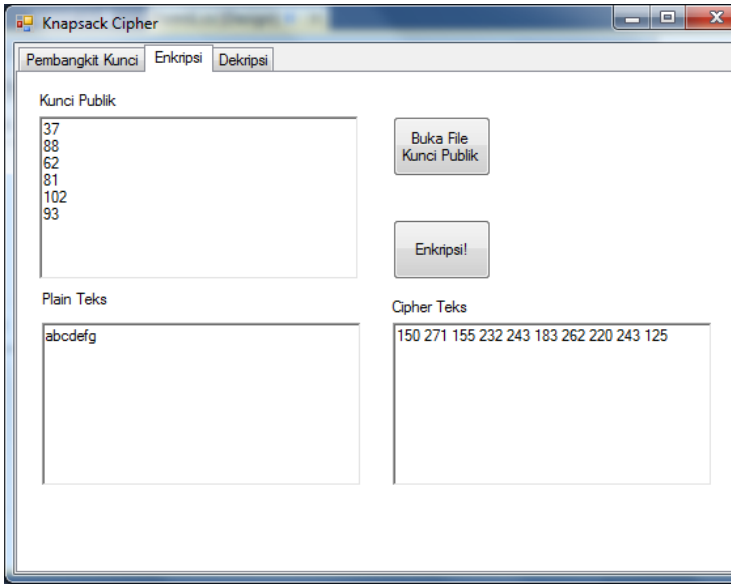
VII. PENGUJIAN

Pada bab pengujian ini, penulis akan menguji algoritma yang telah dibuat tersebut melalui aplikasi C#.

1. Pengujian Pertama

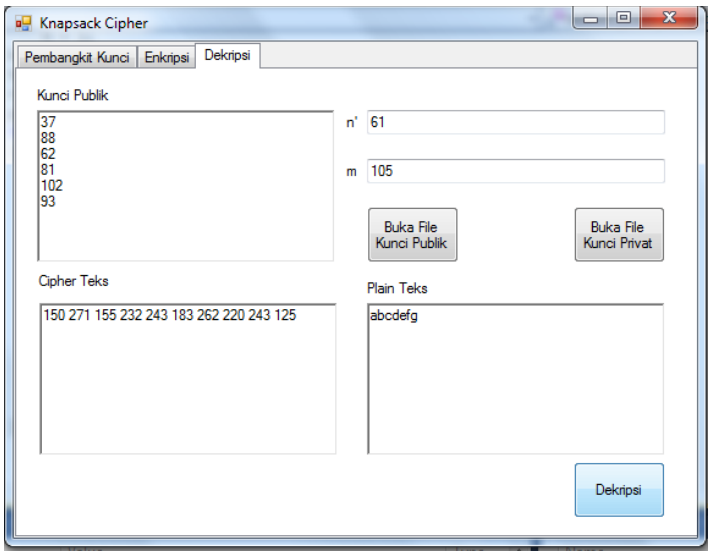
Enkripsi

Plainteks : abcdefg
Kunci publik : {37, 88, 62, 81, 102, 93}
Cipherteks : 150 271 155 232 243 183 262 220 243 125



Dekripsi

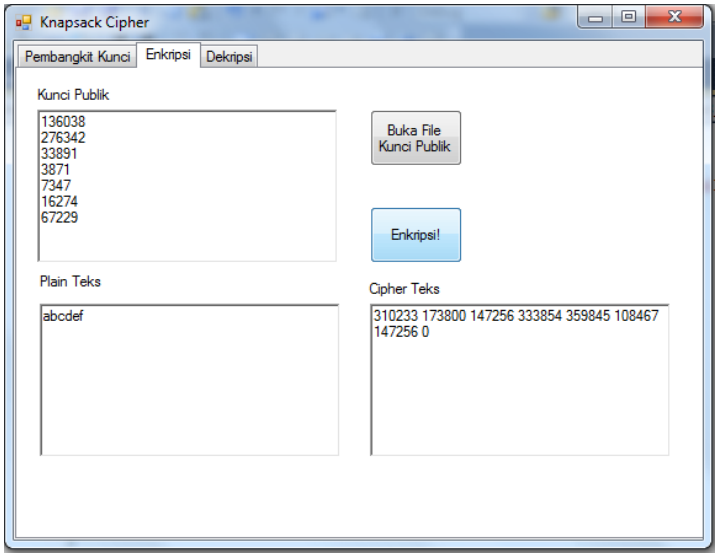
Cipherteks : 150 271 155 232 243 183 262 220 243 125
Kunci publik : {37, 88, 62, 81, 102, 93}
Kunci privat : X(atau n^{-1}) = 61, m = 105
Plainteks : abcdefg



2. Pengujian Kedua

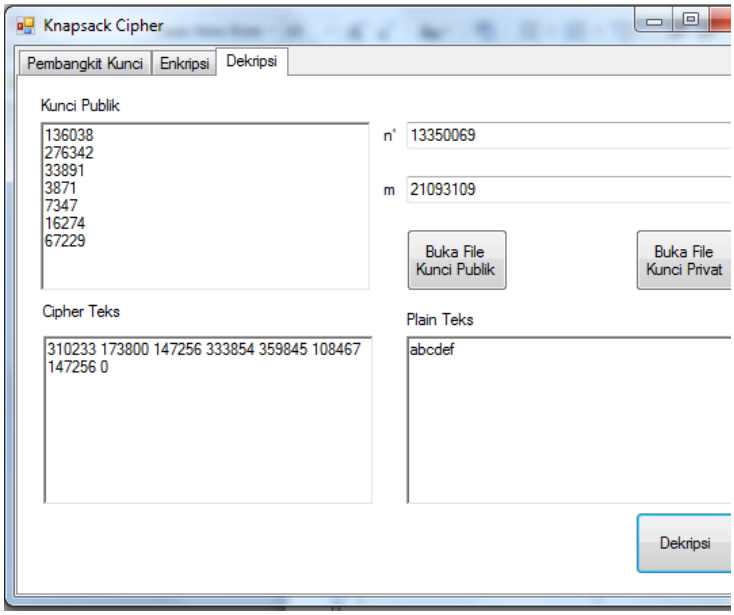
Enkripsi :

Plainteks : abcdef
Kunci publik : {136038, 276342, 33891, 3871, 7347, 16274, 67229}
Cipherteks : 310233 173800 147256 333854 359845 108467 147256 0



Dekripsi

Kunci publik : {136038, 276342, 33891, 3871, 7347, 16274, 67229}
Kunci privat : X(atau n^{-1}) = 13350069, m = 21093109
Cipherteks : 310233 173800 147256 333854 359845 108467 147256 0
Plainteks : abcdef



VIII. ANALISIS

A. Analisis Kompleksitas

Berdasarkan deskripsi algoritma pada bab 5, kita dapat menghitung kompleksitasnya. Untuk proses enkripsi, proses yang paling "berat" adalah memotong pesan dan mengalikan setiap bit yang berkoresponden dengan elemen knapsack. Oleh karena itu, untuk enkripsi, kompleksitasnya O(n).

Sementara itu, untuk dekripsi, proses yang paling "berat" adalah pengurutan elemen knapsack. Oleh karena itu, kompleksitasnya bergantung pada algoritma pengurutan yang digunakan. Jika menggunakan algoritma pengurutan seperti Quick Sort, Merge Sort, atau Heap Sort, maka kompleksitas enkripsi adalah O(n log n). Jika menggunakan algoritma pengurutan seperti Bubble Sort, Selection Sort, maka kompleksitas enkripsinya adalah $O(n^2)$

B. Analisis Keamanan

Setelah mencari referensi dari berbagai sumber, penulis memperoleh hasil bahwa algoritma ini tidak aman lagi. Cara menyerangnya adalah sebagai berikut [3]:

Asumsikan tidak ada permutasi yang digunakan, jadi f = f'. Maka untuk setiap i, terdapat :

$$\mathbf{f}_i \equiv \mathbf{f}'_i w \bmod m$$

Berdasarkan definisi kekongruenan modular, terdapat vector k, sehingga untuk setiap i :

$$u\mathbf{f}_i - m\mathbf{k}_i = \mathbf{f}'_i$$

Di mana nilai u adalah invers dari w dalam moduluo m (karena w related prima terhadap m, maka pasti terdapat inversnya). Maka dengan pembagian, diperoleh :

$$\frac{u}{m} - \frac{\mathbf{k}_i}{\mathbf{f}_i} = \frac{\mathbf{f}'_i}{\mathbf{f}_i m}$$

Karena m sangat besar (lebih besar daripada total semua elemen knapsack), maka ruas kanan nilainya sangat kecil, sehingga k_i/f_i mendekati u/m. Dengan substitusi nilai i dengan 1 dan menguranginya dengan persamaan asli, diperoleh :

$$\left|\frac{\mathbf{k}_i}{\mathbf{f}_i} - \frac{\mathbf{k}_1}{\mathbf{f}_1}\right| = \left|\frac{\mathbf{f}'_i}{m\mathbf{f}_i} - \frac{\mathbf{f}'_1}{m\mathbf{f}_1}\right|$$

Kedua elemen di ruas kanan bernilai positif dan nilai f'_1/mf_1 sangar kecil sehingga,

$$\left|\frac{\mathbf{k}_i}{\mathbf{f}_i} - \frac{\mathbf{k}_1}{\mathbf{f}_1}\right| < \frac{\mathbf{f}'_i}{m\mathbf{f}_i}$$

Lalu karena f' merupakan barisan super-increasing, setiap elemennya pasti lebih kecil daripada setengah nilai elemen setelahnya, sehingga untuk setiap nilai i,

$$\mathbf{f}'_i < m2^{i-n}$$

Dengan kata lain

$$\left|\frac{\mathbf{k}_i}{\mathbf{f}_i} - \frac{\mathbf{k}_1}{\mathbf{f}_1}\right| < \frac{2^{i-n}}{\mathbf{f}_i}$$

Setelah disusun, menjadi

$$\left|\mathbf{k}_i\mathbf{f}_1 - \mathbf{k}_1\mathbf{f}_i\right| < \mathbf{f}_1 2^{i-n}$$

Karena f adalah kunci publik, hanya beberapa pertidaksamaan (hanya 3 atau 4) yang secara unik menentukan nilai k. Pertidaksamaan ini memiliki bentuk "integer programming" sehingga dapat dengan cepat diselesaikan oleh algoritma Lenstra. Setelah diketahui nilai k, mudah sekali untuk menjebol pesan asli.

Jika kita melakukan permutasi terhadap f sebelum disebarkan, cipherteks tetap bisa dijebol dengan mudah. Karena kita hanya membutuhkan 3 atau 4 elemen pertama dari k, kita bisa mencoba semua kemungkinan ${}_nC_4$ atau ${}_nC_3$ yang masih berorde polinomial.

IX. KESIMPULAN DAN SARAN

Dari analisis yang telah dilakukan, ada beberapa kesimpulan yang diperoleh :

1. Sampai saat ini, penulis belum menemukan cara memperbaiki Merkle-Hellman Knapsack agar lolos dari serangan.
2. Secara kompleksitas, algoritma ini sangat baik karena kompleksitas enkripsi-nya hanya O(n) dan kompleksitas dekripsi-nya O(n log n) sehingga tidak terlalu lama untuk melakukan komputasi.
3. Algoritma ini tidak aman karena bisa diserang dengan algoritma LLL.

Saran dari penulis:

1. Jika ingin menggunakan algoritma Knapsack Cipher yang aman, sebaiknya gunakan algoritma Chor-Rivest Knapsack karena sampai sekarang belum ada serangan yang efisien untuk algoritma ini.
2. Jika ingin memperkuat algoritma Knapsack Cipher milik Merkle-Hellman ini. sebaiknya pelajari terlebih dahulu algoritma LLL karena memang algoritma tersebut sangat berguna untuk memecahkan masalah perhitungan bilangan bulat. Saat ini, penulis masih belum memahami cara kerja algoritma LLL sehingga masih bingung dalam menentukan cara mengatasinya.

DAFTAR PUSTAKA

[1] http://en.wikipedia.org/wiki/Lenstra%E2%80%93Lenstra%E2%80%93Lov%C3%A1sz_lattice_basis_reduction_algorithm

[2] http://en.wikipedia.org/wiki/Merkle%E2%80%93Hellman_knapsack_cryptosystem

[3] http://www.derf.net/knapsack/

[4] http://en.wikipedia.org/wiki/Knapsack_problem

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, 9 Mei 2011

Gregorius Ronny Kaluge / 13508019