PENERAPAN KONSEP ALGORITMA GENETIK UNTUK MENINGKATKAN ASPEK KERAHASIAAN DATA PADA ALGORITMA *KNAPSACK*

Nitia Rahmi – 13504068

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesa 10, Bandung
Email : if14068@students.if.itb.ac.id

Abstraksi - *Beberapa konsep yang digunakan dalam algoritma genetik, seperti mutasi, seleksi, dan pindah silang (crossover) dapat diterapkan pada masalah kriptogtafi dengan tujuan untuk semakin meningkatkan aspek kerahasiaan data. Algoritma knapsack merupakan algoritma kriptografi yang keamanannya terletak pada sulitnya memecahkan persoalan knapsack dengan ide dasar yaitu mengkodekan pesan sebagai rangkaian solusi dari persoalan knapsack. Namun, saat ini algoritma knapsack sudah dinyatakan tidak aman. Oleh karena itu, dalam makalah ini diajukan modifikasi algoritma knapsack dengan memanfaatkan konsep algoritma genetik dengan tujuan agar algoritma knapsack menjadi lebih handal dan aspek kerahasiaan data menjadi meningkat. Masalah knapsack yang dibahas fokus pada superincreasing knapsack. Namun, penerapan algoritma genetik pada masalah knapsack lain, seperti knapsack biasa dan knapsack kunci-publik tetap akan dijelaskan dalam makalah ini. Pada akhir makalah, akan dilakukan ekperimen untuk membandingan antara algoritma knapsack murni dengan algoritma knapsack yang telah dimodifikasi dengan konsep algoritma genetik. Implementasi algoritma genetik menggunakan GAlib.*

Kata Kunci : *algoritma genetik, algoritma knapsack, evolusi, hereditas*

1. PENDAHULUAN

Algoritma *knapsack* adalah algoritma kunci publik yang keamanannya terletak pada sulitnya memecahkan persoalan *knapsack* [3]. Masalah *knapsack* dijelaskan sebagai berikut ini.

Bobot *knapsack* adalah *M*. Diketahui *n* buah objek yang masing-masing bobotnya adalah $w_1, w_2, w_3,.., w_n$ lalu tentukan nilai b_i sedemikian sehingga :

$$M = b_1w_1 + b_2w_2 + b_3w_3 + ... + b_nw_n$$

Yang dalam hal ini, b bernilai 0 atau 1. Jika $b_i = 1$, berarti objek *i* dimasukkan ke dalam *knapsack*, sebaliknya jika *bi* = 0 maka objek *i* tidak dimasukkan. Cara kerja *knapsack* :

Plainteks	1 0 1	1 1 0	1 0 0	0 0 0
Knapsack	1 7 9	1 7 9	1 7 9	1 7 9
Cipherteks	1+ 9 = 10	1+7= 8	1	0

Ide ini, menciptakan dua permasalahan *knapsack* yaitu *knapsack* yang mudah diselesaikan (*superincreasing*) dan *knapsack* yang sulit dipecahkan terkait dengan masalah komputasi yang kompleks. Persoalan *knapsack* biasa, merupakan persoalan *NP-Complete*, yaitu tidak dapat dipecahkan dalam orde waktu polinomial.

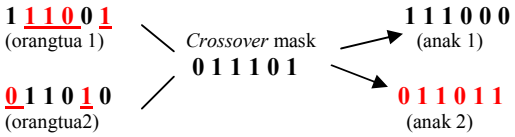
Superincreasing knapsack memiliki ciri khas pada kunci yang digunakan, yaitu harus merupakan barisan *superincreasing*, barisan di mana setiap nilai di dalam barus tersebut lebih besar daripada jumlah seluruh nilai sebelumnya. Misalnya {2, 3, 6,13,27,52}. Karena algoritma *knapsack* melibatkan operasi bit per bit, maka algoritma genetik sangat cocok diterapkan. Banyak operasi dalam konsep algritma genetik yang dapat diterapkan sehingga pada akhirnya akan meningkatkan aspek kerahasiaan data.

2. KONSEP ALGORITMA GENETIK

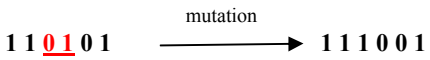
Kemunculan algoritma genetik terinspirasi dari teori-teori dalam ilmu biologi sehingga banyak istilah dan konsep biologi yang digunakan dalam algoritma genetik Sesuai dengan namanya, proses-proses yang terjadi dalam algoritma genetik sana dengan yang terjadi pada evolusi biologi [4]. Dalam ilmu biologi, sekumpulan individu (spesies) yang sama, yang hidup, bereproduksi, dan mati di suatu area disebut populasi. Konsep penting dari evolusi makhluk hidup adalah hereditas,yaitu sebuah ide yang menyatakan bahwa sifat-sifat individu dapat dikodekan dengan cara tertentu sehingga dapat diturunkan ke generasi berikutnya. Informasi yang akan diturunkan ini, tersimpan dalam sebuah *genome* yang berisi *kromosom-kromosom* dalam bentuk molekul DNA. Konsep penting lainnya dalam teori evolusi adalah *fitness* dan *selection* untuk proses reproduksi. Pada proses evolusi di dunia nyata, terdapat dua cara reproduksi, yaitu *sexual reproduction* dan *asexual reproduction*. Pada *sexual reproduction, kromosom-*

kromosom dari dua individu dikombinasikan untuk menghasilkan individu baru. Artinya kromosom individu baru berisi beberapa gen yang diambil dari orang tua pertama dan beberapa gen dari orang tua kedua. Hal ini disebut pindah silang (*crossover*). Namun, proses pengopian gen dari orang tua, tentu ada kemungkinan terjadi kesalahan. Kesalahn pengopian ini, disebut mutasi. Sedangkan pada *asexual reproduction*, hanya satu individu orang tua yang diperhatikan sehingga tidak ada *crossover*, tetapi mutasi tetap mungkin untuk terjadi. Konsep-konsep tersebut diadopsi oleh algoritma genetik. Dalam algoritma genetik juga diterapkan tiga operasi :

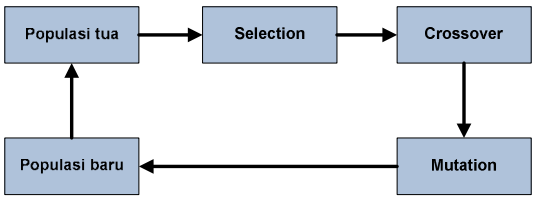
a. Seleksi (*selection*)
   Selection adalah proses penentuan individu mana yang akan menjadi parent di generasi berikutnya. Secara alamiah, individu yang lemah tidak akan bertahan hidup lama untuk bereproduksi. Dengan kata lain, probabilitas individu diturunkan, proporsional dengan nilai fittness-nya (seberapa baik individu tersebut mampu beradaptasi dengan lingkungan).
b. Pindah silang (*crossover*)
   Prinsip pindah silang adalah sebuah kromosom yan mengarah pada solusi yang bagus dapat diperoleh dengan memindahsilangkan dua buah kromosom.



c. Mutasi (*mutation*)
   Prosedur mutasi sangat sederhana, yaitu untuk semua gen yang ada, jika bilangan random yang dibangkitkan kurang dari peluang mutasi yang telah ditentukan, maka ubah nilai gen tersebut dengan nilai kebalikannya (1 menjadi 0, dan 0 menjadi 1).



Dalam algoritma genetik, terdapat siklus :



Gambar 1 Siklus Algoritma Genetik

Jadi dari suatu populasi, akan terjadi proses seleksi, yaitu pemilihan individu, kemudian, pada gen-gen individu tersebut, akan dilakukan operasi genetik, seperti *crossover* dan mutation, sehingga menghasilkan populasi baru. Populasi baru pun akan mengalami evolusi kembali. Pada akhirnya setiap populasi baru akan menjadi population tua.

Berikut ini algoritma genetik standar :

```
GA (fitness, fitness_threshold, p,r,m)
p : populasi individu terpilih
r :  selisih kenaikan populasi
m : peluang terjadiya mutasi
1. Inisialisasi : P ← p
2. Evaluasi : untuk setiap h dalam P
   hitung fitness(h)
   While [max_h fitness(h)] < fitness_threshold
   1) Select : pilih (1-r)p anggota P,tambahkan
            ke Ps
         Pr(h_i) =  Fitness(Hi) / Σ_{j=1}^{p} Fitness(Hj)

   2) Crossover : pilih rp/2 pasangan h dalam
      P,menghasilkan 2 offspring, tambahkan ke
      Ps.
   3) Mutate : balik bit (mp) anggota Ps
   4) Update : P← Ps
   5) Evaluate : untuk tiap h dalam P, hitung
            fitness(h)
3. Return : hipotesis dalam P yang
   nilai fitness-nya paling tinggi
```

3. PENERAPAN ALGORITMA GENETIK PADA *KNAPSACK*

Algoritma *knapsack* dan algoritma genetik mempunyai satu kesamaan, yaitu pemrosesannya dilakukan dalam kode bit-bit biner. Modifikasi algoritma *superincreasing knapsack* dengan penerapan algoritma genetik (sebut saja GA-*Knapsack*), dilakukan dengan menggunakan *library* algoritma genetik, GAlib dan didukung dengan MATLAB untuk melihat performansinya. Secara garis besar, cara kerja algoritma GA-*Knapsack* untuk persoalan sederhana adalah sebagai berikut.

a. Enkripsi

Enkripsi plainteks dengan algoritma GA-*Knapsack* dilakukan mengikuti langkah-langkah berikut.

1) Diketahui plainteks : 011000110101101110
2) Tentukan kunci privat : {2, 3, 6, 13, 27, 52}
3) Tentukan *crossover mask*
   Crossover mask dibangkitkan dari kunci privat, dengan persamaan berikut.
   p : kunci privat
   n : bilangan yang tidak punya faktor persekutuan dengan m
   m : bilangan yang lebih besar dari barisan superincreasing (kunci privat)
   crossover mask = p.n mod m
   misalnya, kita tetapkan n = 31 dan m = 105
   jadi dengan menggunakan persamaan di atas,

dihasilkan *crossover* mask :

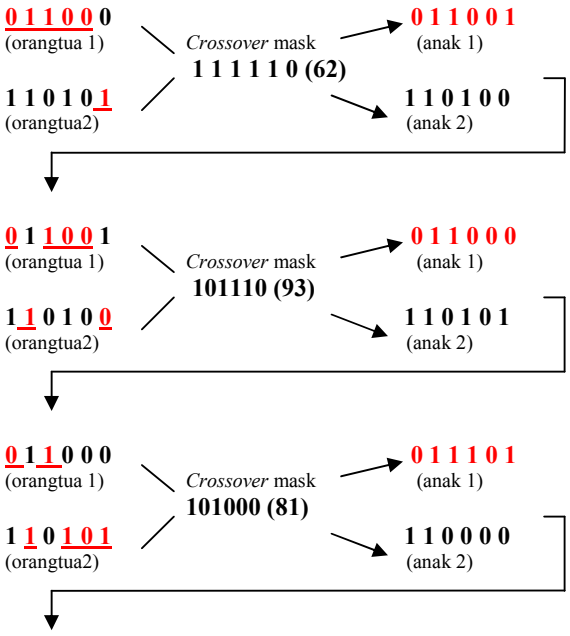
- 2 . 31 mod 105 = 62 = 111110
- 3 . 31 mod 105 = 93 = 1011101 ≈ 101110
- (karena biner dari 93 panjangnya 7 bit, maka ambil 6 bit pertama)
- 6 . 31 mod 105 = 81 = 1010001≈ 101000
- 13 . 31 mod 105 = 102 = 1100110 ≈ 110011
- 52 . 31 mod 105 = 37 = 100101

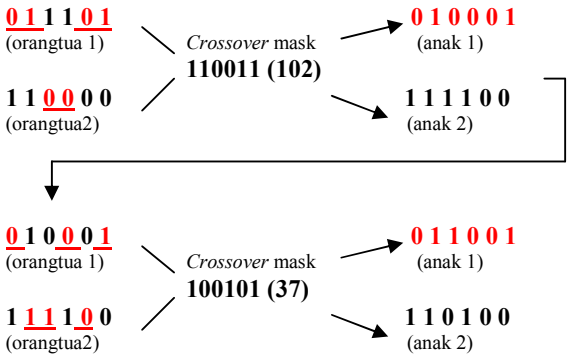
Jadi, didapatkan *crossover* mask = {62, 93, 81, 102, 37}

4) Aturan penggunakan operator genetik pada GA-*Knapsack* :
Jumlah operasi crossover = Jumlah blok plainteks / 2(dalam pasangan blok plainteks)

5) Plainteks : 011000110101101110
   - Diketahui *crossover* mask 1 = 111110 (6 bit)
   - Bagi plainteks, per 6 bit = {011000, 110101, 101110}
   - Didapatkan jumlah blok plainteks = 3 buah
   - Jumlah operasi *crossover* = 3 / 2 = 1 (1 pasangan blok plainteks)
   - Jumlah operasi mutasi = 2 mod 2 = 1 (1 buah blok plainteks)

6) Lakukan *crossover* antara blok plainteks-1 dan blok plainteks-2
   Dalam operasi ini, sebut saja :
   - Blok plainteks-1 sebagai orangtua1
   - Blok plainteks-2 sebagai orangtua2





7) Hasil akhir dari operasi *crossover* (blok plainteks-1 dan 2) lakukan mutasi
   0 1 1 0 0 1 → 1 0 0 1 1 0
   1 1 0 1 0 0 → 0 0 1 0 1 1

8) Karena jumlah blok plainteks ada 3 buah, maka untuk blok terakhir dilakukan satu kali operasi mutasi :
   1 0 1 1 1 0 → 0 1 0 0 0 1

9) Jadi terdapat 3 buah blok plainteks hasil operasi genetik (GAPlainteks) , yaitu : {100110, 001011, 010001}

   - GAPlainteks ke-1 : 100110
     Knapsack : 2, 3, 6, 13, 27, 52
     Kriptogram : 2+13+27= 42
   - GAPlainteks ke-2 : 001011
     Knapsack : 2, 3, 6, 13, 27, 52
     Kriptogram : 6+27+52=85
   - GAPlainteks ke-2 : 010001
     Knapsack : 2, 3, 6, 13, 27, 52
     Kriptogra : 3+52=55

Jadi, cipherteks yang dihasilkan : {42, 85 ,55}

b. Dekripsi

1) Diketahui cipherteks :{42, 85, 55}
2) Diketahui kunci privat : {2, 3, 6, 13, 27, 52}
3) Didapatka 3 buah persamaan *knapsack* :
   - $42 = 2b_1 + 3b_2 + 6b_3 + 13b_4 + 27b_5 + 52b_5$ (1)
   - $85 = 2b_1 + 3b_2 + 6b_3 + 13b_4 + 27b_5 + 52b_5$ (2)
   - $55 = 2b_1 + 3b_2 + 6b_3 + 13b_4 + 27b_5 + 52b_5$ (3)
4) Untuk persamaan (1) :

   $42 = 2b_1 + 3b_2 + 6b_3 + 13b_4 + 27b_5 + 52b_5$
   - Bandingkan 42 dengan nilai terbesar pada kunci publik, yaitu 52. Karena 42<52, maka 42 tidak dimasukkan ke dalam *knapsack*, diberi tanda 0.
   - Bandingkan 42 dengan bobot kedua terbesar yaitu 27. Ternyata 42 > 27,

maka 42 dimasukkan ke dalam *knapsack*, diberi tanda 1. Bobot total sekarang tinggal 42 - 27 = 15.

- Bandingkan 15 dengan 13. Ternyata 15> 13, maka 15 dimasukkan ke dalam *knapsack*, diberi tanda 1. Bobot total sekarang tinggal 15 - 13 = 2.
- Begitulah seterusnya. Prosesnya dapat dituliskan secara lebih singkat, yaitu sebaga berikut.
  42 ? 52 → 0
  42 ? 27 → 1
  15 ? 13 → 1
  2 ? 6 → 0
  2 ? 3 → 0
  2 ? 2 → 1

Didapatkan, blok GAPlainteks-1 = 100110

5) Untuk persamaan (2) :

$85 = 2b_1 + 3b_2 + 6b_3 + 13b_4 + 27b_5 + 52b_5$

- Bandingkan 85 dengan nilai terbesar pada kunci publik, yaitu 52. Karena 85>52, maka 85 dimasukkan ke dalam *knapsack*, diberi tanda 1 Bobot total sekarang menjadi 85 – 52 = 33.
- Bandingkan 33 dengan bobot kedua terbesar yaitu 27. Ternyata 33> 27, maka 33 dimasukkan ke dalam *knapsack*, diberi tanda 1. Bobot total sekarang tinggal 33 - 27 = 6.
- Bandingkan 6 dengan 13. Ternyata 6 < 13, maka 6 tidak dimasukkan ke dalam *knapsack*, diberi tanda 0.
- Begitulah seterusnya. Prosesnya dapat dituliskan secara lebih singkat, yaitu sebaga berikut.
  85 ? 52 → 1
  33 ? 27 → 1
  6 ? 13 → 0
  6 ? 6 → 1
  0 ? 3 → 0
  0 ? 2 → 0
  Didapatkan, blok GAPlainteks-2 = 001011

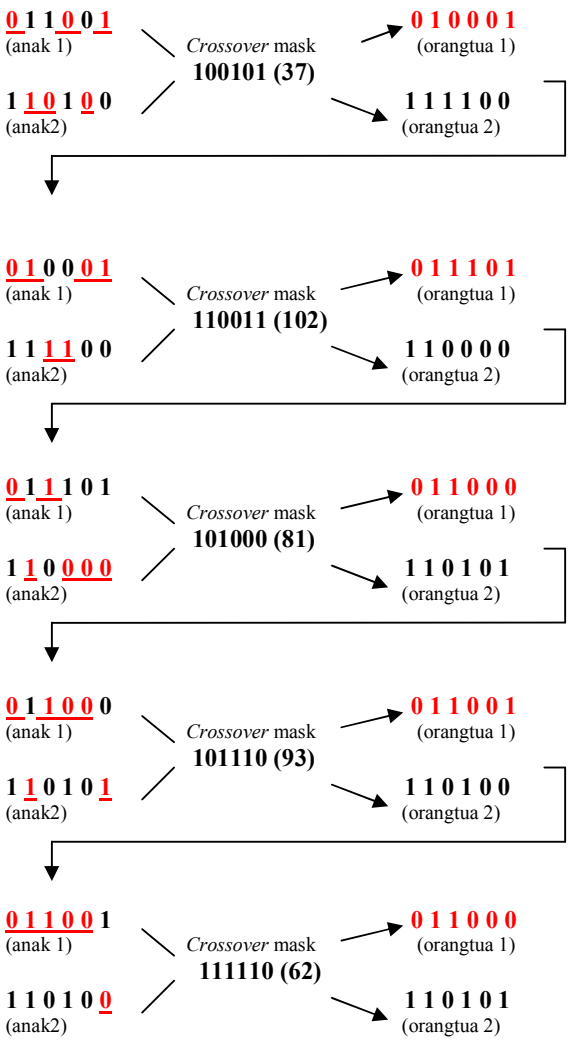
6) Untuk persamaan (2) :

$55 = 2b_1 + 3b_2 + 6b_3 + 13b_4 + 27b_5 + 52b_5$

- Bandingkan 55 dengan nilai terbesar pada kunci publik, yaitu 52. Karena 55>52, maka 55 dimasukkan ke dalam *knapsack*, diberi tanda 1 Bobot total sekarang menjadi 55 – 52 = 3.
- Bandingkan 3 dengan bobot kedua terbesar yaitu 27. Ternyata 3< 27, maka 3 tidak dimasukkan ke dalam *knapsack*, diberi tanda 0.
- Begitulah seterusnya. Prosesnya dapat dituliskan secara lebih singkat, yaitu sebaga berikut.
  55 ? 52 → 1
  3 ? 27 → 0
  3 ? 13 → 0
  3 ? 6 → 0
  3 ? 3 → 1
  0 ? 2 → 0

- Didapatkan, blok GAPlainteks-2 = 010001

7) Didapatkan blok GA-Plainteks : {100110, 001011, 010001}
8) Lakukan mutasi GA-Plainteks sehingga menjadi : {011001, 110100, 101110}
9) Karena ada 3 blok GA-Plainteks, maka :
   Jumlah operasi *crossover* = 3 / 2 = 1 (antara blok GAPlainteks-1 dan blok GAPlainteks-2)
10) *Crossover* mask yang digunakan untuk dekripsi, sama dengan *crossover* mask yang digunakan saat enkripsi. Hanya saja, saat dekripsi, urutan penggunaannya berlawanan, yaitu 37,102, 81, 93, dan 62.
11) Lakukan *invers crossover* :



Jadi, didapat subplainteks : **011000110101**

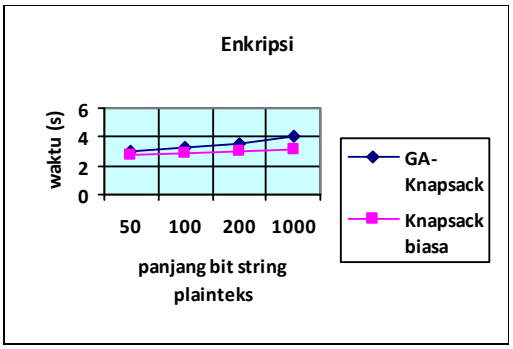
12) Gabungkan plainteks yang didapat dari *invers crossover* (langkah no.11) dengan GA-Plainteks yang ke-3 hasil langkah no. 8 sehingga didapatkan plainteks utuh, yaitu : **011000110101101110**

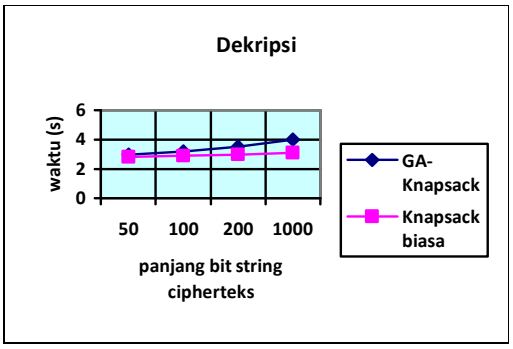
Pada bagian a dan b telah dipaparkan proses enkripsi dan dekripsi pesan dengan menerapkan algoritma genetik pada *superincreasing knapsack*. Sebenarnya, proses enkripsi pada point a, juga dapat diterapkan pada kanpsack biasa dan *knapsack* kunci publik. Namun, proses dekripsinya yang berbeda. Untuk penerapan algoritma genetik pada *knapsack* biasa dan *knapsack* kunci-publik, proses dekripsi dilakukan dengan bantuan algoritma genetik, dalam proses pencarian generasi yang tepat. Hal ini sangat tergantung pada pemilihan paraeter algoritma genetik yang digunakan. Jika tepat, maka hasilnya pun akan cepat didapat.

4. UJI COBA DAN HASIL

Dilakukan uji coba untuk melihat perbandingan hasil antara *knapsack* yang telah dimodifikasi dengan *knapsack* tanpa modifikasi. Uji coba dilakukan berdasarkan parameter panjang bit string plainteks dan waktu yang dibutuhkan kripnatalis/program untuk memecahkan cipherteks.

Dalam hal ini, kunci privat, bilangan n, dan bilangan m bersifat rahasia. Hasilnya adalah sebagai berikut.





Dari grafik di atas terlihat bahwa algoritma *knapsack* yang telah dimodifikasi dengan algoritma genetik membutuhkan waktu enkripsi dan dekripsi yang lebih lama dibandingkan dengan algoritma *knapsack* biasa. Waktu yang dibutuhkan sebanding dengan panjang bit string teks. Dapat dilihat bahwa untuk teks yang panjangnya < 200, penggunaan algoritma *knapsack* yang telah dimodifikasi tidak menunjukkan hasil yang signifikan dibandingkan dengan *knapsack* biasa. Hasilnya akan terasa jika teks yang akan dienkripsi cukup panjang (> 1000).

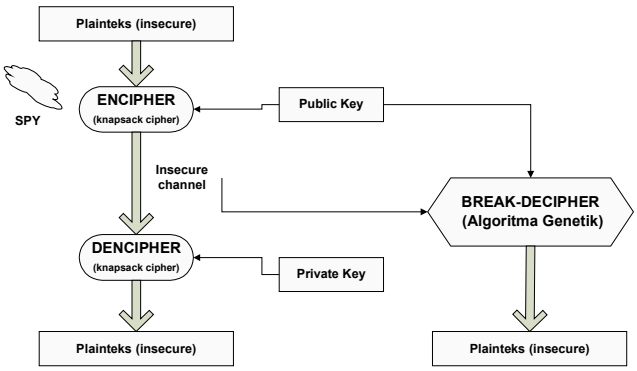
Karena waktu enkripsi dan dekripsi yang lebih lama, maka dapat dipahami bahwa algoritma *knapsack* yang dimodifikasi dengan algoritma genetik menjadi lebih kompleks, sehingga kerahasiaan data lebih terjamin jika dibandingkan dengan algoritma *knapsack* biasa.

5. ISU LAIN PENGGUNAAN ALGORITMA GENETIK PADA ALGORITMA *KNAPSACK*

Penerapan algoritma genetik pada algoritma *knapsack* memang terbukti membuat proses enkripsi dan dekripsi menjadi lebih komplek sehingga aspek kerahasiaan data dapat meningkat. Namun, di balik semua itu, seiring dengan perkembangan ilmu algoritma genetik dan semakin menurunnya biaya hardware komputer, semungkinkan hambatan di bidang komputasi bukanlah masalah besar. Paralelisasi dapat dengan mudah dilakukan untuk meningkatkan performasi perangkat keras.

Menurut penelitian yang dilakukan oleh Radomil Matousek [1], ternyata algoritma genetik menjadi kunci tentang ketidakamanan algoritma *knapsack*. Semua bentuk algoritma *knapsack* dan modifikasi apapun yang dilakukan terhadapnya, tidak membuat algoritma *knapsack* handal, bebas dari segala bentuk penyerangan.

Penerapan metode heuristik pada algoritma genetik dapat digunakan untuk memecahkan segala bentuk pesan. Ingat kembali prinsip algoritma genetik, bahwa evolusi suatu populasi ke populasi berikutnya selalu lebih baik. Dengan menggunakan perangkat keras yang *powerfull* penggunaan algoritma genetik untuk memecahkan cipherteks tanpa mengetahui apa yang menjadi rahasia (meliputi kunci privat, bilangan m, dan n) menjadi sangat mungkin. Perhatikan skema berikut.



Gambar 2 Kriptosistem dan Kriptanalisis Merkle-Hellman dengan metode heuristik. [1]

Dari skema di atas, tampak bahwa segala bentuk kehandalan algoritma *knapsack* dapat dipatahkan dengan bentuk penyerangan yang menggunakan algoritma genetik.

6. KESIMPULAN DAN SARAN

Dari uraian di atas, dapat ditarik kesimpulan sebagai berikut.

- Algoritma genetik merupakan algoritma yang powerfull untuk masalah optimisasi.
- Modifikasi *superincreasing knapsack* dengan menerapkan algoritma genetik meningkatkan aspek confussion dan diffusion pada algoritma *knapsack*.
- Proses enkripsi dan dekripsi pesan menjadi lebih kompleks, hal ini daoat dilihat dari segi waktu eksekusi. Terdapat perbedaan waktu antara *Knapsack* yang telah dimodifikasi (GA-*Knapsack*) dengan *knapsack* murni. Waktu eksekusi, sebanding dengan panjang pesan yang akan diekkripsi atau didekripsi. Semakin panjang pesan, maka perbedaan anatar GA-*Knapsack* dengan *Knapsack* murni makin terlihat.
- Penerapan konsep algoritma genetik pada *knapsack* cukup membuahkan hasil.
- Untuk mematahkan GA-*Knapsack* secara brute force hampir tidak mungkin
- Sayangnya, seiring dengan kemajuan ilmu pengetahuan, didapatkan fenomena, bahwa penggunaan algoritma genetik dapat mematahkan algoritma kriptografi yang menerapkan operasi bit per bit, salah satunya yaitu algoritma *knapsack*.

Saran penulis adalah sebagai berikut.

- Perlu dilakukan kajian lebih mendalam tentang kriptografi genetik. Mengingat banyak sekali kegunaan dan ancaman dari perkembangan algoritma genetik terhadap kriptografi.Ancamannya berupa kemampuan algoritma genetik untuk memecahkan atau mematahkan algoritma kriptografi tertentu.
- Perlu dilakukan penelitian, bagaimana karakteristik algoritma kriptografi yang aman terhadap penyerangan dengan pemanfaatan algoritma genetik.

7. REFERENSI

[1] Matousek, Radomil. *Knapsack Chipher and Cryptanalyst Using Heuristic Methods*, Institute of Automation and Computer Science, Brno University of Technology.

[2] Mitchell, Tom. *Machine Learning*, Carniege Mellon University, McGraw-Hill Companies, 1997.

[3] Munir, Rinaldi. Diktat Kuliah IF5054 Kriptografi. STEI. 2006

[4] Suyanto, Algoritma Genetikaa dalam MATLAB. ANDI Publisher, Yogyakarta, 2005.