STUDI TERHADAP *ADVANCED ENCRYPTION STANDARD*(*AES*) DAN ALGORITMA *KNAPSACK* DALAM PENGAMANAN DATA

Asriyanik[1]

[1]*Program Studi Teknik Informatika, Universitas Muhammadiyah Sukabumi*

ABSTRAK

Studi terhadap algoritma kriptologi sangat membantu bagi para calon cryptanalyst dalam memperkaya wawasan terhadap jenis-jenis algoritma kriptologi. Maka sebagai salah satu upaya dalam mempelajari algoritma kriptologi, maka dilakukanlah sebuah studi terhadap dua algoritma yaitu Advanced Encryption Standar (AES)dan algoritma Knapsack yang masing-masing akan dipelajari mengenai cara kerjaenkripsi dan dekripsi, kelebihan dan kelemahan juga serangan yang terjadi pada kedua algoritma tersebut. Dari kedua algoritma tersebut masing-masing memiliki cara kerja yang berbeda, AES yang merupakan algoritma simetris dengan cipherbloknya memiliki cara kerja pada enkripsi dan dekripsi dengan kunci yang sama pada sisi pengirim dan penerima data (pesan), dimana kunci tersebut dibagi ke dalam blok-blok byte yang baku. Selain itu, AES juga memiliki kelebihan dan kelemahan umum sebagai algoritma simetris ataupun kelebihan dan kelemahan yang khusus pada algoritmanya sendiri.Adapun algoritma Knapsack yang merupakan algoritma asimetris dengan memiliki banyak kunci pada proses enkripsi dan dekripsinya sehingga memiliki kelebihan dan kelemahan umum ataupun khusus.

Kata kunci:*AES, Knapsack, Enkripsi, Dekripsi, Serangan, Kelebihan, Kelemahan*

PENDAHULUAN

Keamanan data sangatlah penting dalam proses pengiriman ataupun penerimaan data yang dilakukan melalui media komputer. Pengamanan data ini dilakukan dalam rangka menjaga kerahasiaan, keutuhan, keabsahan dan ketersediaan data.Sehingga untuk memenuhi hal tersebut diperlukan adaya upaya dalam pengamanan data yakni dengan enkripsi dan dekripsi. Enkripsi yaitu melakukan manipulasi terhadap data asli/data masukan (*plaintext*) yang akan dikirim sehingga menjadi data rahasia/data keluaran (*ciphertext*). Sedangkan dekripsi yaitu mengubah data rahasia/data keluaran (*ciphertext*) yang diterima sehingga menjadi data asli/data masukan (*plaintext*).

Baik dalam proses enkripsi atau pun dekripsi banyak sekali perkembangan terhadap algoritma yang dapat digunakan dalam kedua proses tersebut. Hingga saat ini terdapat beberapa algoritma dalam kriptologi modern yang mulai dikenal sekitar tahun 1977.Di antara algoritma dalam kriptologi modern tersebut terdapat dua jenis algoritma, yaitu algoritma simestris dan algoritma asimetris. Di dalam algoritma simetris terdapat jenis algoritma*cipher*blok (*block cipher*), yaitu algoritma dengan proses enkripsi dimana data asli (*plaintext*) diubah ke dalam bentuk bit yang dibagi menjadi blok-blok dengan panjang bit yang sama, seperti *AES* (*Advanced Encryption Standard*). Adapun algoritma asimetris yang populer sebagai algoritma kunci publik, merupakan perkembangan dari algoritma simestris yang hanya memiliki satu kunci, maka di dalam algoritma asimetris memiliki banyak kunci yang digunakan, contoh dari algoritma ini adalah algoritma *Knapsack* di mana algoritma ini menggunakan analogi kantung yang memiliki kapasistas tertentu akan digunakan untuk menyimpan data sesuai dengan kapasitas kantung tersebut.

Sebagai pembelajaranterhadap*AES* (*Advanced Encryption Standard*) sebagai algoritma *cipher*blok dengan kunci simetris danalgoritma *Knapsack* sebagai algoritma kunci publik dengan kunci asismetris, maka akan diuraikan di antaranya:cara kerja dari *AES* (*Advanced Encryption Standard*) dan algoritma *Knapsack*, kekurangan serta kelebihan dari kedua algoritma tersebut juga yang lainnya.

AES (*Advanced Encryption Standard*)

Berawal dari proyek dari NIST (*National Institute of Standard and Technology*) Amerika, yang mengadakan sayembara terbuka dalam membuat standar kriptologi pengganti *DES* (*Data Encryption Standard*) yang dianggap lemah dalam faktor keamanan, kemudian algoritma baru

tersebut diberi nama*AES* (*Advanced Encryption Standard*). Dan pada tahun 2001, algoritma *Rijndael* yang ditemukan oleh Vincent Rijmen dan Daemon dari Belgia terpilih sebagai penyandang algoritma *AES* (*Advanced Encryption Standard*).

Ukuran Blok dan Panjang Kunci *Rijndael*

Rijndael memiliki ukuran blok 128 bit serta panjang kunci kunci 128, 192 dan 256 bit dengan step 32 bit. Inilah ukuran blok dan kunci yang telah ditetapkan secara independen pada AES sehingga dikenal dengan AES128, AES192 dan AES256.

Unit dasar dalam pemrosesan pada algoritma AES (Rijndael) adalah byteatau setara dengan 8 bit.Sehinggaurutan bit data masukan akan diubah terlebih dahulu menjadi urutan byte. Kemudian setelah diubah ke dalam byte, dibuat array dimensi dua atau yang biasa disebut dengan state dan setiap array memiliki 4 baris (rows), demikian pun dengan setiap panjang bit data masukan akan dibagi ke dalam blok dengan ukuran 4 blok.

Operator yang digunakan adalah operator penjumlahan, perkalian dan perkalian dengan konstanta pada medan GF(2^8) yang merupakan sebuah medan berhingga dengan koefisien derajat polinomial kurang dari 8.

Jika mengacu pada ketentuan AES tersebut, maka akan dapat disimpulkan bahwa setiap bit data masukan akan dibagi dengan 8 bit. Misalnya 128 bit/8 bit = 16 byte sehingga 128 bit setara dengan 16 byte atau 4 word (1 word = 32 bit). Begitu pula dengan data masukan 192 bit akan setara dengan 24 byte atau 6worddan data masukan 256 bit setara dengan 32 byteatau 8 word.Dan hitungan ini akan sesuai dengan penentuan panjang kunci (Nk) yang digunakan pada proses enkripsi. Sehingga jika data masukan 128 bit disimpan ke dalamstate (S) akan membentukarray dimensi dua berukuran 4 baris (rows [r]) dan 4 kolom (column[c]) di mana elemen *array* diacu sebagai $S[r,c]$, dengan $0 \leq r < 4$ dan $0 \leq c < Nc$. *Nc* adalah panjang blok dibagi 32. $Nc = 128/32 = 4$.

Proses enkripsi dari algoritma *Rijndael* itu sendiri akan dilakukan dalam sejumlah putaran tertentu bergantung pada panjang kuncinya lebih jelasnya dapat dilihat pada tabel 1 berikut.

Tabel 1. Jumlah Putaran Tiap Blok Pada*AES*

Varian *AES*	Panjang Kunci (*Nk words*)	Ukuran Blok (*Nb words*)	Jumlah Putaran (*Nr*)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Keterangan: 1 Word = 32 bit

Algoritma *Rijndael*

Seperti pada *DES*, *Rijndael* menggunakan substitusi dan permutasi, dan sejumlah putaran. Untuk setiap putarannya, *Rijndael* menggunakan kunci yang berbeda. Kunci setiap putaran disebut *round key*.

Setiap trasformasi putaran pada terdiri atas 3 buah lapisan seragam tetapi berbeda yang dapat dibalik dengan fungsinya masing-masing. Lapisan-lapisan tersebut adalah :

- Lapisan *Linear-Mixing* yang fungsinya untuk menjamin tingkat difusi dengan banyaknya putaran,
- Lapisan *Non-Linear* merupakan aplikasi paralel dari S-Box yang memiliki properti nirlanjar optimum pada kasus terburuk,
- Lapisan *Key-Addition*yaitu dengan operasi XOR sederhana kunci internal terhadap *state* sementara.

Enkripsi

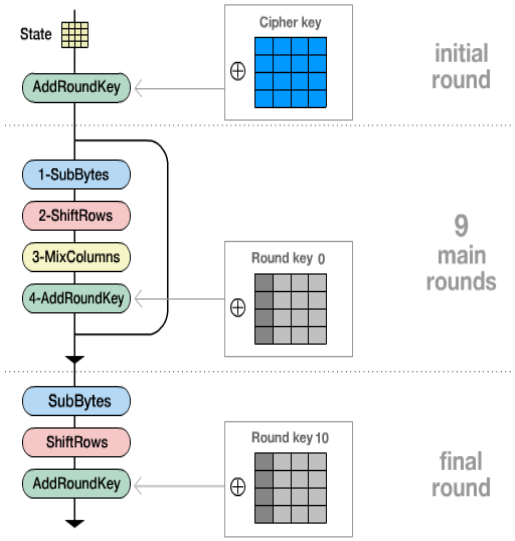
Sebagai gambaran umum proses enkripsi pada algoritma *Rijndael* yang beroperasi blok 128 bit dengan kunci 128 bit adalah sebagai berikut :

1. *Initial Round* yaitu tahapan *AddRoundKey* yang melakukan *XOR*(*Exlusive OR*) antara *state* awal dari data masukan (*plaintext*) dengan *cipher key*(kunci *cipher*).
2. Putaran sebanyak *Nr* – 1 kali (Round *Nr-1*). Proses yang dilakukan pada setiap putaran adalah:
   a. *SubByte*: substitusi *byte* dengan menggunakan tabel substitusi (*S-Box*). Tabel substitusi (*S-Box*) dapat dilihat pada tabel 2, sedangkan ilustrasi *SubByte* dapat dilihat padagambar 2.
   b. *ShiftRow:* pergeseran baris-baris *array state* secara *wrapping*. Ilustarsi *ShiftRow* dapat dilihat pada gambar 3.

c. *MixColumn*: mengacak data di masing-masing kolom *array state*. Ilustarsi *MixColumn* dapat dilihat pada gambar 4.
d. *AddRoundKey*yaitumelakukan *XOR* antara *state* sekarang dengan *round key*. Ilustarsi *AddRoundKey* dapat dilihat pada gambar 5.

3. *Final round*: proses untuk putaran terakhir:
   a. *ByteSub*,
   b. *ShiftRow*,
   c. *AddRoundKey*.

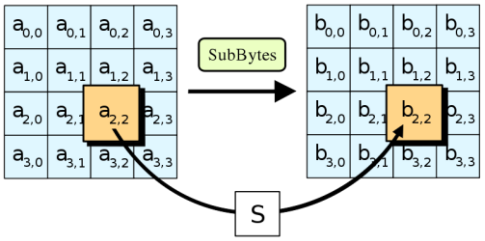
Skema proses enkripsi pada *AES* dapat dilihat pada gambar 1 berikut*:*



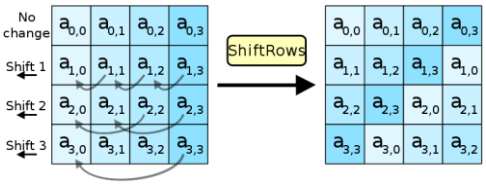
Gambar 1. Skema Enkripsi*AES*

Tabel 2. Tabel Substitusi (*S-Box*)yang Digunakan dalam Transformasi *SubByte* pada *AES*

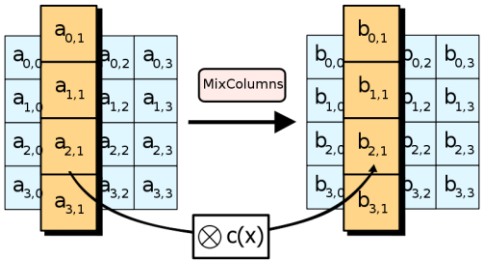
hex	y															
x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16



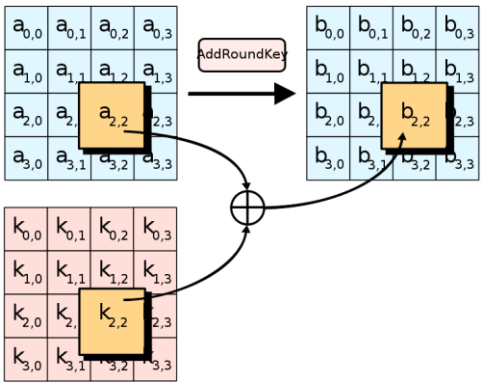
Gambar 2. Ilustrasi Transformasi *SubByte* pada *AES*



Gambar 3. Ilustrasi Transformasi *ShiftRows* pada *AES*



Gambar 4.Ilustrasi Transformasi *MixColumn* pada *AES*



Gambar 5.Ilustrasi Transformasi *AddRoundKey*pada *AES*

Dekripsi

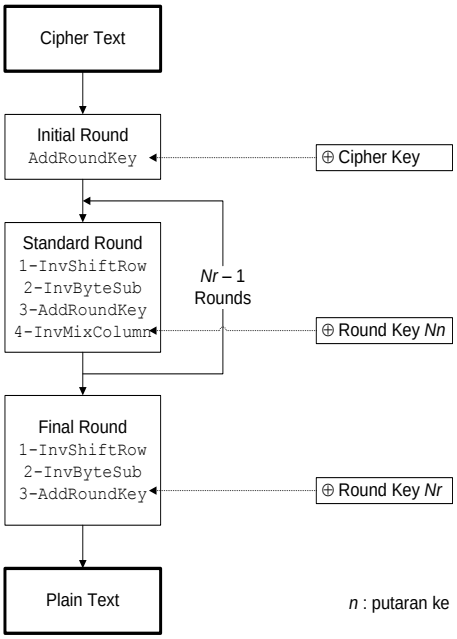
Secara garis besar, langkah dekripsi atau disebut juga dengan istilah *Invers Cipher* dari algoritma *Rijndael*yang beroperasi blok 128bit dengan kunci 128bit adalah sebagai berikut:

1. *InitialRoundyaitu*tahapan *AddRoundKey* yang melakukan *XOR* antara *state* awal (*ciphertext*)

dengan *cipher key*. Tahap ini disebut juga *InitialRound*.

2. Putaran sebanyak *Nr* – 1 kali. Proses yang dilakukan pada setiap putaran adalah:
   a. *InvShiftRow:* pergeseran baris-baris *array state* secara *wrapping*.
   b. *InvByteSub*: substitusi byte dengan menggunakan tabel substitusi kebalikan (*inverse S-box*). Tabel substitusi dapat dilihat pada tabel 3.
   c. *AddRoundKey*yaitu melakukan *XOR* antara *state* sekarang dengan *round key*.
   d. *InvMixColumn*: mengacak data di masing-masing kolom *array state*.
3. *Final Round*: proses untuk putaran terakhir:
   a. *InvShiftRow*,
   b. *InvByteSub*,
   c. *AddRoundKey*.

Skema proses dekripsi *AES* dapat dilihat pada gambar 6 berikut:



Gambar 6. Skema Dekripsi*AES*

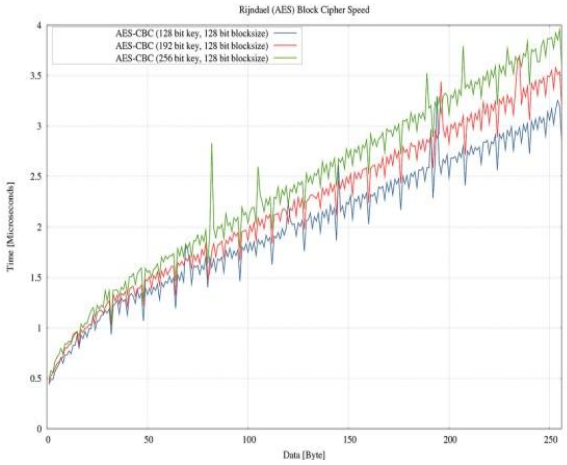
Tabel 3. Tabel *S-box* yang Digunakan dalam Transformasi *InvByteSub*pada *AES*

hex	y															
x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Kelebihan *AES*(*Advanced Encryption Standard*)

Berdasarkan analisis pada beberapa referensi dapat diketahui beberapa kelebihan dari *AES* (*Advanced Encryption Standard*) di antaranya:

a. Dilihat dari segi jenis kunci yang simetri maka kecepatan operasi (komputasi) lebih tinggi bila dibandingkan dengan algoritma asimetrik sehingga dapat digunakan pada sistem real-time seperti GSM. Grafik dari kecepatan AES pada salah satu mode operasi cipherblok yaitu CBC (*Cipher Block Chaining*)dapat dilihat pada gambar 7.



Gambar 7. Grafik Kecepatan *AES*

b. Karena *AES* mempunyai panjang kunci paling sedikit 128 bit, maka *AES* tahan terhadap serangan *exhaustive key search* dengan teknologi saat ini. Dengan panjang kunci 128-bit, maka terdapat $2^{128} \approx 3{,}4 \times 10^{38}$ kemungkinan kunci. Jika digunakan sebuah mesin dengan semilyar prosesor paralel, masing-masing dapat menghitung sebuah kunci setiap satu *pico* detik, maka akan

dibutuhkan waktu 10^{10} tahun untuk mencoba seluruh kemungkinan kunci [6].

c. Kekuatan *AES* terletak pada karakteristik sifat dari medan GF(2^8) dimana untuk setiap bilangan prima selalu terdapat sebuah medan tunggal terbatas yang unik sehingga seluruh representasi dari GF(2^8) bersifat *isomorphic* dan pemilihan polinomial biner berderajat 8 m(x) bersifat *irreducible*yakni tidak dapat dibagi oleh bilangan lain pada medan selain 1 dan dirinya sendiri, relatif prima terhadap medan.Kekuatan ini didasari oleh operasi matematis yang kompleks dan membutuhkan sumber daya yang tidak sedikit untuk melakukan komputasi. Karena didasarkan pada persamaan matematis, AES dapat dengan mudah dibuktikan keamanannya [7].

Kelemahan *AES* (*Advanced Encryption Standard*)

Beberapa kelemahan yang ditemukan pada *AES* (*Advanced Encryption Standard*) antara lain:

a. Dari segi jenis kunci yang simetris, maka akan terjadi kesulitan dalam manajemen kunci. Hal ini terjadi karena untuk setiap pengiriman dan penerimaandata dengan pengguna yang berbeda dibutuhkan kunci yang berbeda pula.
b. Masih dari segi jenis kunci yang simetris dimana pengirim dan penerima datamemiliki kunci yang sama untuk setiap proses pengiriman-penerimaan data, hal ini akan menyebabkan kunci mudah bocor meskipun dalam waktu yang lama.
c. Mengacu pada kekuatan *AES* yang ditulis pada sub bahasan 2.3poin c, dapat diketahui bahwa itu pula yang menjadi kelemahan terbesar dari AES karena dengan berhasilnya dipecahkan persamaan matematis yang mendasarinya secara otomatis seluruh sistem di dalam AES dapat ditembus dan dengan demikian barisan pertahanannya dapat dikatakan hancur berantakan, luluh lantak.

Serangan Terhadap AES

Berbagai teknik serangan pernah dilakukan terhadap AESseperti[8]:

- *Differential Cryptanalisis* dan *Linear Cryptanalysis,*
- *Truncated Differentials,*
- *The Square Attacks*dan *Interpolation Attacks*.

Dari berbagai serangan tersebut, secara matematis berbagai pembuktian telah dilakukan dan ditunjukkan bahwa AES dapat bertahan menghadapi serangan-serangan itu.

Akan tetapi pernah ada bentuk serangan XSL yaitu sebuah serangan terhadap *cipher*blok, dan serang ini tidak dapat dibuktikan tidak efektif terhadap AES. Bahkan percobaan pada *baby* Rijndael, sebuah semi cipher putaran tidak penuh dibuktikan berhasil. Dikatakan pula, dengan metoda tersebut, Rijndael terbukti menjadi yang terlemah dari kelima kandidat AES[9].

Serangan Terbaru Terhadap *AES*

Baru-baru ini terdapat serangan terhadap AES yang membuktikan cacatnya proses komputasi di dalam AES yaitu *BicliqueCryptanalysis* yang ditemukan oleh AndreyBogdanov, DmitryKhovratovich dan ChristianRechberger sebagai salah satu riset dari Microsoft. Serangan ini merupakan serangan terhadap*cipher* blokyang sudah memiliki hasil sebagai berikut:

- Kunci pertama pemulihan menyerang terhadap AES-128 secara penuhdengan kompleksitaskomputasi $2^{126,1}$.
- Kunci pertama pemulihan menyerang terhadap AES-192 secara penuh dengan kompleksitas komputasi $2^{189.7}$.
- Kunci pertama pemulihan menyerang terhadap AES-256 secara penuh dengan kompleksitas komputasi $2^{254.4}$.

Berdasarkan serangan terhadap cipherAES yang dilakukan secara penuh jumlah putarannya ternyata setiap panjang bit AESbekerja dengan pengurangan sekitar 2 bit, sehingga mereka menyebutkan bahwa AES128 adalah AES126, AES192 adalah AES190 dn AES256 adalah AES254. Hal ini sudah dapat diketahui pada serangan dengan jumlah putaran yang tidak penuh, padahal menurut para cryptanalyst inibahwa sebelumnya tidak pernah mempertimbangkan untuk menyerang pada kompleksitas yang rendah.

Algoritma *Knapsack*

Algoritma *Knapsack* adalah adalah algoritma kriptografi kunci-publik yang keamanannya terletak pada sulitnya memecahkan persoalan *Knapsack* (*Knapsack* Problem). Arti kata *knapsack* adalah kantung, karung atau ransel. Ide dari algoritma *knapsack* adalah bagaimana mengisi kapasitas kantung yang terbatas dengan

barang yang paling berguna dan berharga sampai batas kapasitas maksimum kantung. Jadi permasalahan *knapsack*adalah optimalisasi kombinatorial dalam menempatkan benda-benda agar masuk ke dalam kantung.

Knapsack Problem

Knapsack Problem merupakan masalah optimasi kombinatorial. Sebagai contoh adalah suatu kumpulan barang masing-masing memiliki berat dan nilai, kemudian akan ditentukan jumlah tiap barang untuk dimasukkan dalam koleksi sehingga total berat kurang atau sama dari batas yang diberikan dan nilai total seluas mungkin yang bisa dimasukkan. Misal terdapat barang sebanyak n yang akan dimasukkan ke dalam kantung dengan kapasitas M. Setiap barang memiliki beban w_i dan keuntungan b_i. Permasalahan tersebut dapat digambarkan dengan persamaan berikut:

$$M = b_1w_1 + b_2w_2 + \ldots + b_nw_n \quad (1)$$

Keterangan:

M = total bobot *knapsack*
w = bobot masing-masing objek, juga merupakan kunci privat
b = nilai bit *plaintext*, bernilai 0 atau 1 (0 berarti objek tidak dimasukkan, 1 berarti objek dimasukkan)
n = banyaknya barang

Dalam teori algoritma, permasalahan *knapsack* termasuk dalam kelompok NP-*complete*.Persoalan yang termasuk NP-*complete* tidak dapat dipecahkan dalam waktu orde polinomial.

Perkembangan Algoritma *Knapsack*

Algoritma *Knapsack* berkembang mulai dari algoritma yang sederhana sampai algoritma yang rumit, yaitu:

Algoritma *Knapsack* Sederhana

Ide dasar dari algoritma kriptografi *knapsack* adalah mengkodekan pesan sebagai rangkaian solusi dari persoalan *knapsack*.Setiap beban **w** dinyatakan sebagai kunci privat dan bit-bit *plaintext*dinyatakan dalam b.

Langkah-langkah untuk meng-enkripsi *plaintext*-nya adalah:

a) Bagi *plaintext* menjadi blok yang panjangnya sejumlah banyak data (**n**)
b) Kemudian setiap bit di dalam blok dikalikan dengan nilai **w** yang berkoresponden sesuai dengan persamaan (1).

Sehingga jika terdapat 6 buah barang yang akan dimasukkan ke dalam *knapsack*, dan beban barang-barang itu adalah 1, 5, 6, 11, 14, 20. Dan terdapat *plaintext* yang akan dienkripsi menggunakan algoritma *knapsack*, dengan bit sebagai berikut 111001010110000000011000 maka penyelesaian untuk masalah tersebut adalah:
n = 6
w = 1, 5, 6, 11, 14, 20
$\mathbf{M} = \mathbf{b_1w_1} + \mathbf{b_2w_2} + \ldots + \mathbf{b_nw_n}$
dengan hasil sebagai berikut:

Blok planteks 1 : 111001
Knapsack : 1, 5, 6, 11, 14, 20
Kriptogram : (1x1) + (1x5) + (1x6) + (0x11) + (0x14) + (1x20) =**32**

Blok planteks 2 : 010110
Knapsack : 1, 5, 6, 11, 14, 20
Kriptogram : (0x1) + (1x5) + (0x6) + (1x11) + (1x14) + (0x20) =**30**

Blok planteks 3 : 000000
Knapsack : 1, 5, 6, 11, 14, 20
Kriptogram : (0x1) + (0x5) + (0x6) + (0x11) + (0x14) + (0x20) =**0**

Blok planteks 4 : 011000
Knapsack : 1, 5, 6, 11, 14, 20
Kriptogram : (0x1) + (1x5) + (1x6) + (0x11) + (0x14) + (0x20) =**11**
dan*ciphertext* yang dihasilkan adalah: **32 30 0 11**.

Kelemahan algoritma sederhana ini adalah sulitnya mencari proses dekripsi *ciphertext* ke data awal. Misal, untuk mencari *plaintext* dari *ciphertext* 30(dari uraian di atas) yaitu dengan menentukan nilai b_1, b_2, b_3, b_4, b_5, b_6.Maka dengan menggunakan persamaan (1), maka didapat:

$$\mathbf{30 = b_1 + 5b_2 + 6b_3 + 11b_4 + 14b_5 + 20b_6}$$

Penyelesaian dari persamaan tersebut tidak dapat dipecahkan dalam waktu orde polinomial terutama dengan semakin banyaknya nilai **n**, hal

ini dengan asumsi bahwa barisan beban barang tidak berurut naik.Namun kelemahan ini, menjadi satu kekuatan karena dengan rumitnya pencarian dekripsi berarti keamanan data lebih terjaga.

SuperincreasingKnapsack

Superincreasingknapsack adalah permasalahan *knapsack* yang dapat dipecahkan dalam orde **O(n).** Sehingga *superincreasingknapsack* bukanlah algoritma kriptografi yang kuat karena mudah dipecahkan. Barisan beban disebut *superincreasing* jika setiap nilai di dalam barisan lebih besar daripada jumlah semua nilai sebelumnya. Misal, {1, 2, 5, 10, 20} adalah barisan *superincreasing* dan {1, 2, 3, 5, 10} bukan barisan *superincreasing*.

Dalam *superincreasingknapsack*, untuk mencari nilai b_1, b_2, … b_n dengan kata lain mendekripsikan *ciphertext*, dapat dicari dengan cara sebagai berikut:

a) Jumlahkan semua bobot di dalam barisan.
b) Bandingkan bobot total dengan bobot terbesar di dalam barisan. Jika bobot terbesar lebih kecil atau sama dengan bobot total, maka masukkan ke dalam *knapsack*, jika tidak maka tidak dimasukkan.
c) Kurangi bobot total dengan bobot yang telah dimasukkan kemudian bandingkan bobot total sekarang dengan dengan bobot terbesar selanjutnya. Demikian terus sampai seluruh bobot dalam barisan selesai dibandingkan.
d) Jika bobot total menjadi nol, maka ada penyelesaian dari permasalahan *superincreasingknapsack*, jika tidak nol, maka tidak ada penyelesaiannya.

Sehingga jika terdapat barisan *superincreasing* sebagai berikut: {2,3,6,13,27,52} dan total kapasitas *knapsack* adalah 70. Hal ini dapat digambarkan dengan persamaan:

$$\mathbf{70 = 2b_1 + 3b_2 + 6b_3 + 13b_4 + 27b_5 + 52b_6}$$

Untuk mencari nilai b_1, b_2, … b_6, dapat dicari dengan cara berikut:

1) Bandingkan 70 dengan bobot terbesar, yaitu 52. Karena $52 \leq 70$, maka 52 dimasukkan ke dalam *knapsack*
2) Bobot total sekarang menjadi 70-52=18. Bandingkan 18 dengan bobot terbesar kedua, yaitu 27. Karena 27 > 18, Maka 27 tidak dimasukkan ke dalam *knapsack*.
3) Bandingkan 18 dengan bobot terbesar ketiga, yaitu 13. $13 \leq 18$, maka 13 dimasukkan ke dalam *knapsack*
4) Bobot total sekarang menjadi 18-13=5. Bandingkan 5 dengan bobot terbesar ke 4, yaitu 6. Karena 6 > 5, maka 6 tidak dimasukkan.
5) Bandingkan 5 dengan bobot terbesar berikutnya yaitu 3. Karena $3 \leq 5$, maka 3 dimasukkan ke dalam *knapsack*.
6) Bobot total sekarang menjadi 5-3=2. Karena $2 \leq 2$, maka 2 dimasukkan ke dalam *knapsack*
7) Bobot total sekarang adalah 2-2=0.

Karena bobot total yang tersisa adalah 0, maka penyelesaian dari *superincreasingknapsack* ditemukan. Barisan bobot yang dimasukkan ke dalam *knapsack* adalah {2,3,-,13,-,52}.

Dari uraian di atas, dapat disimpulkan bahwa untuk mencari dekripsi dalam *superincreasingknapsack* dapat dilakukan dengan mudah jika nilai beban diketahui, hal inilah yang menjadi kelemahan dari *superincreasingknapsack*.

Algoritma *Knapsack* Kunci Publik

Algoritma *knapsack* kunci publik diperkenalkan oleh Ralph Merkle dan Martin Hellman pada tahun 1978.Mereka memodifikasi kunci privat dalam algoritma superincreasing. Algoritma *super increasing knapsack* memiliki kelemahan karena mudah mendekripsi cipherteks menjadi *plaintext*. Untuk mengatasi hal ini, algoritma *superincreasingknapsack* dapat diubah menjadi algoritma *non superincreasingknapsack* atau normal *knapsack* dengan menggunakan kunci publik untuk dekripsi didapat dari barisan normal *knapsack* dan kunci privat untuk enkripsi didapat dari barisan *superincreasingknapsack*.

Algoritma *non-superincreasingknapsack* atau normal *knapsack* adalah algoritma yang sangat rumit, oleh karena itu penyelesaiannya pun cukup sulit memerlukan wartu dalam orde eksponensial.

Langka-langkah untuk membangkitkan kunci publik dan kunci privat adalah sebagai berikut:

1) Tentukan barisan superincreasing
2) Kalikan setiap elemen di dalam barisan dengan n modulo m. Modulus m seharusnya angka yang lebih besar daripada jumlah semua elemen di dalam barisan, sedangkan pengali m

seharusnya tidak memiliki persekutuan dengan m.

3) Hasil perkalian akan menjadi kunci publik, sedangkan barisan superincreasing semula menjadi kunci privat.

Persamaan untuk mendapatkan kunci publik adalah:

$$\beta_i = w_i.n \bmod m \quad (2)$$

Keterangan:
β_i = kunci publik
w_i = kunci privat, didapat dari beban masing-masing barang

Langkah – langkah enkripsi dan dekripsi dalam algoritma *knapsack* Markle Hellman adalah:

a) *Enkripsi*

Proses enkripsi dilakukan hampir sama dengan algoritma *knapsack* sederhana, yaitu saat pesan dikirim, pesan diubah ke dalam bentuk bit. Bit-bit (α) yang didapat dikelompokkan sesuai dengan kardinalitas barisan kunci publik (β). Dan kalikan bit di dalam kelompok blok dengan elemen yang berkoresponden dengan kunci publik.

$$C = \sum_{i=1}^{n} \propto_i \beta_i \quad (3)$$

Keterangan:
C = kriptogram enkripsi
α = bit *plaintext*
β = kunci publik
n = kardinalitas kunci publik

b) Dekripsi

Proses dekripsi dalam algoritma *knapsack* Markle Hellman dicari dengan menggunakan kunci privat (**w**). Saat pesan diterima, maka penerima pesan menghitung nilai $\mathbf{n^{-1}}$, yaitu invers dari **n** modulo **m**, sehingga:

$$n.\ n^{-1} \equiv 1 \pmod m$$
$$n.\ n^{-1} = 1 + km$$
$$n^{-1} = \frac{(1+km)}{n},\ k \text{ sembarang bilangan bulat}$$

Setelah didapat nilai $\mathbf{n^{-1}}$, maka kalikan setiap kriptogram dengan $\mathbf{n^{-1}}$ modulo **m**, lalu nyatakan hasil kalinya sebagai penjumlahan elemen-elemen kunci privat untuk memperoleh *plaintext* dengan menggunakan algoritma pencarian solusi superincreasing *knapsack*. Jika digambarkan dalam persamaan, yaitu sebagai berikut:

$$W = C.\ n^{-1} \bmod m$$

Keterangan:
W = jumlah beban, yang akan dikonversi dalam bit
C = kriptogram enkripsi

Kelebihan dari Algoritma *Knapsack*

Kelebihan dari algoritma *knapsack* adalah:

a. Jika dilihat dari jenis kunci yang digunakan, yaitu kunci publik dan privat, maka hanya kunci privat yang perlu dijaga kerahasiaannya oleh setiap entitas yang berkomunikasi
b. Permasalahan *knapsack*termasuk ke dalam kelompok *NP-complete*. Permasalahan yang termasuk *NP-complete* tidak dapat dipecahkan dalam orde waktu polinomial. Sehingga keamanan data lebih terjaga.
c. Agar algoritma *knapsack* lebih kuat, maka kunci publik maupun kunci privat sebaiknya terdiri dari paling sedikit 250 elemen. Selain itu, nilai dari setiap elemen memiliki panjang antara 200 dan 400 bit. Dengan nilai-nilai sepanjang itu, diperkirakan dibutuhkan waktu selama 1046 tahun untuk memecahkan kunci tersebut secara *brute force*, dengan asumsi satu juta percobaan tiap detik.

Kelemahan Algoritma *Knapsack*

Kelemahan khusus pada setiap algoritma *knapsack* sudah diuraikan pada setiap sub bahasan perkembangan algoritma *knapsack*. Sehingga kelemahan secara umum dapat diuraikan sebagai berikut:

a. Dengan menggunakan kunci asimetri, enkripsi dan dekripsi data umumnya lebih lambat daripada sistem simetri, karena enkripsi dan dekripsi menggunakan bilangan yang besar dan melibatkan operasi perpangkatan yang besar.
b. Ukuran cipherteks lebih besar daripada *plaintext* (bisa dua sampai empat kali ukuran *plaintext*).
c. Sistem Merkle-Hellman dapat dipecahkan oleh Shamir pada tahun 1984 yang mempresentasikan algoritma waktu *polynomial* yang mengkalkulasikan *knapsack* yang mudah dari kunci publik. Shamir menggunakan properti *superincreasing* dari bilangan integer sebuah *knapsack* untuk memperoleh ketidaksamaan linier dari sistem.

KESIMPULAN

a. Setiap algoritma kriptografi mempunyai kelemahan dan kelebihan tersendiri.
b. Serangan akan dapat dilakukan terhadap setiap algoritma kriptografi baik dalam waktu yang cepat ataupun lama.
c. Algoritma kriptografi akan selalu mengalami perkembangan seiring dengan berkembangannya ilmu dan teknologi.

DAFTAR PUSTAKA

Joan Daemen, Rijmen Vincent. *AES Proposal: Rijndael*. Document version 2-Date 03/09/99.

Michell Setyawati H. Analisis AES Rijndael terhadap DES.Makalah IF3058 Kriptografi– Sem. II Tahun 2010/2011.

http://en.wikipedia.org/wiki/Advanced_Encryption_Standard/

Federal InformationProcessing Standards Publication 197. Announcing The Advanced Encryption Standard (AES). November 26, 2001

http://en.wikipedia.org/wiki/AES_implementations/

Renaldi Munir. Kriptografi. Informatika Bandung. 2006.

http://en.wikipedia.org/wiki/Advanced_Encryption_Standard#Known_attacks/

http://research.microsoft.com/en-us/projects/cryptanalysis/aes.aspx

Dony Ariyus. *Pengantar ilmu kriptografi, Teori, Analisis dan Implementasi*.Penerbit Andi Yogyakarta. 2008

http://en.wikipedia.org/wiki/Knapsack_problem/

http://en.wikipedia.org/wiki/Markle-Hellman_knapsack_cryptosystem/

Bayu Adi Persada. *Studi dan Implementasi Pengembangan Algoritma Superincreasing Knapsack Menjadi Algoritma Knapsack Normal*. Makalah IF ITB.

Gregorius Ronny Kaluge.*Penambahan Permutasi pada Knapsack Cipher*. Makalah IF ITB.