Analisis Keamanan Penggunaan Algoritma AES/Rijndael untuk Penyandian pada Aplikasi Kompresi Berkas

Bernardino Madaharsa Dito Adiwidya – NIM: 13507089
Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung,
Jalan Ganesha 10, Bandung, email: if17089@students.if.itb.ac.id

Abstract – *Makalah ini membahas mengenai analisis keamanan penggunaan algoritma AES/Rijndael untuk berkas terkompresi. Selain itu, dibahas mengenai dasar algoritma AES, struktur berkas kompresi, dan format berkas kompresi terenkripsi karena sangat mempengaruhi serangan-serangan yang mungkin dilakukan. Dalam mengantisipasi serangan tersebut, baik pengguna maupun pengembang aplikasi kompresi sebaiknya selalu waspada dengan melakukan hal-hal tertentu yang perlu dilakukan. Pengguna sebaiknya selalu hati-hati dalam pemakaian sandi-lewat, sedangkan pengembang sebaiknya selalu rajin memperbaiki aplikasi dan aturan pengenkripsian berkasnya.*

Kata Kunci*:algoritma, AES/Rijndael, kompresi, enkripsi, sandi-lewat, serangan*

1. PENDAHULUAN

Pada saat ini kerahasiaan data merupakan suatu hal yang sangat penting. Data yang dimiliki tentu saja belum tentu ingin diketahui oleh masyarakat luas. Jika data yang ingin dilindungi sangat banyak dan ingin dikumpulkan dalam satu berkas saja, masyarakat dapat menggunakan aplikasi kompresi berkas untuk menyatukannya. Aplikasi tersebut memiliki banyak alternatif algoritma penyandian, tergantung merk aplikasi yang digunakan. Namun, saat ini semua aplikasi kompresi berkas memiliki sebuah opsi penyandian untuk melakukan pengenkripsian berkas yang saat ini dianggap sebagai algoritma pengenkripsi terbaik, yaitu *Advanced Encryption Standard* (AES) atau Rijndael.

Pada pertengahan tahun 1990-an, diketahui bahwa algoritma DES (*Data Encryption Standard*), yang saat itu banyak digunakan, berhasil dipecahkan. Oleh karena itu, NIST (*National Institute of Standard and Technology*) berusaha untuk menyeleksi algoritma yang pantas untuk menggantikan DES. Setelah melalui kompetisi yang dilakukan pada tahun 1997 dan proses seleksi yang amat ketat dan membutuhkan waktu yang cukup lama, pada tanggal 2 Oktober 2000 terpilihlah algoritma *Rijndael* yang dibuat oleh Rijmen dan Daemen dari Belgia untuk menggantikan DES dan terpilih menjadi AES.

Akibat terpilihnya algoritma AES/Rijndael ini, banyak aplikasi pada saat ini yang menggunakannya untuk mengenkripsi suatu data, salah satunya aplikasi untuk kompresi berkas. Saat ini terdapat berbagai macam merk aplikasi untuk melakukan kompresi berkas. Pada umumnya, aplikasi-aplikasi tersebut telah dilengkapi dengan fitur pemberian sandi-lewat (*password*) untuk mengenkripsi berkas yang telah terkompresi tersebut. Tentu saja ini mengakibatkan peningkatan kesulitan orang yang tidak berhak yang ingin mengetahui data yang ada di dalamnya. Hal ini disebabkan oleh selain untuk membuka berkas terkompresi tersebut dapat memerlukan aplikasi tertentu (sistem operasi yang terpasang belum tentu memiliki fitur untuk membukanya), berkas itu telah dienkripsi sehingga hanya pihak yang berhak saja yang dapat membukanya.

Hingga saat ini, banyak orang yang berusaha memecahkan algoritma ini dengan membuat algoritma-algoritma baru atau menggunakan berbagai macam trik, meskipun belum ada satu orang pun yang benar-benar dapat memecahkannya. Oleh karena itu, perlu diketahui kepastian keamanan penggunaan algoritma ini supaya masyarakat awam dapat merasa aman dengan data yang dimilikinya tanpa khawatir data yang dimilikinya diketahui orang lain atau tersebar tanpa sepengetahuan pemiliknya.

2. PENGGUNAAN ALGORITMA AES/RIJNDAEL

Algoritma AES/Rijndael saat ini banyak digunakan terutama untuk penyandian atau pengenkripsian. Dalam makalah ini, algoritma ini digunakan untuk melakukan pengenkripsian berkas yang telah dikompresi. Berikut ini akan dibahas terlebih dahulu tentang teori dasar algoritma ini.

2.1. Teori Dasar

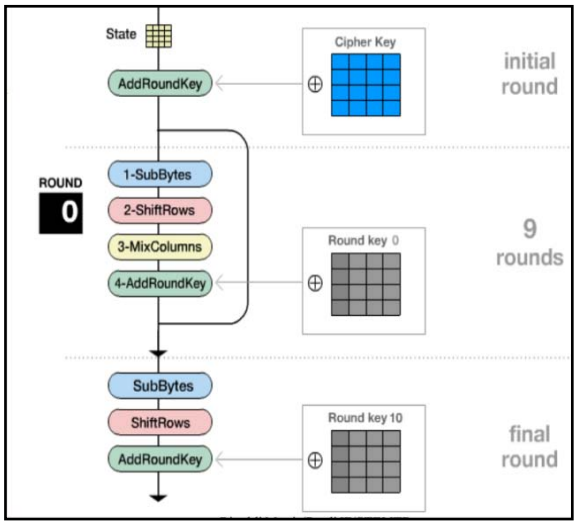
Algoritma AES/Rijndael mendukung panjang kunci dari 128 bit hingga 256 bit dengan step 32 bit. AES menetapkan bahwa panjang kunci hanya 128, 192, dan 256 sehingga dikenal AES-128, AES-192, dan AES-256. Panjang kunci dan ukuran blok dapat dipilih secara independen [1]. Setiap blok dienkripsi dalam sejumlah putaran tertentu yang jumlahnya dapat dilihat pada tabel di bawah ini.

Tabel 1 Jumlah Putaran Pengoperasian AES [2]

Tipe	Panjang Kunci	Panjang Blok Input	Jumlah Putaran (N_r)
AES-128	128 bit	128 bit	10
AES-192	192 bit	128 bit	12
AES-256	256 bit	128 bit	14

Secara umum, garis besar algoritma AES adalah terbagi dalam putaran-putaran dengan operasi masing-masing sebagai berikut.

1. *Initial Round*
   - AddRoundKey
2. *Rounds* (N_r–1 kali)
   - SubBytes
   - ShiftRows
   - MixColumns
   - AddRoundKey
3. *Final Round*
   - SubBytes
   - ShiftRows
   - AddRoundKey



Gambar 1 Skema *Round* untuk Kunci 128 bit [1]

Berikut ini penjelasan dari empat macam operasi yang dimiliki putaran-putaran di atas.

2.1.1. SubBytes (Subtitusi *Byte*)

Setiap *byte* yang akan dienkripsi pada operasi ini disubtitusikan menjadi nilai *byte* lain dengan menggunakan S-box. S-box yang digunakan diperoleh dari *multiplicative inverse* dari angka yang diberikan dalam *Rijndael's finite field* yang kemudian ditransformasikan dengan *affine transformation*:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Gambar 2 *Affine Transformation*

Hasilnya kemudian di-*xor* dengan 99_{10} atau $0x63_{16}$ atau 1100011_2. Operasi matriks dengan *xor* ini ekuivalen dengan persamaan:

$b'_i = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i$

dengan *b'*, *b*, dan *c* adalah array 8 bit dan nilai *c* adalah 01100011. Proses tersebut menghasilkan masing-masing nilai dari elemen tabel *S-box* yang hasilnya sebagai berikut.

Tabel 2 S-box

	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

Untuk proses dekripsi, diperlukan tabel S-box inversi sebagai berikut.

Tabel 3 S-box Inversi

	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

2.1.2. ShiftRows (Pergeseran Baris)

Dalam operasi ini, *byte* dari setiap baris digeser secara memutar dengan aturan pergeseran yang berbeda untuk setiap baris. Aturan pergeseran dapat berbeda untuk panjang blok yang berbeda. Aturan-aturannya adalah sebagai berikut [3].

1. Baris pertama untuk semua jenis panjang blok (128, 192, dan 256 bit) tidak digeser.
2. Baris kedua untuk semua jenis panjang blok digeser 1 ke kiri.
3. Pada panjang blok 128 dan 192 bit, baris ketiga digeser ke kiri sebanyak dua kali dan baris keempat digeser ke kiri sebanyak tiga kali.
4. Pada panjang blok 256 bit, baris ketiga digeser ke kiri sebanyak tiga kali dan baris keempat digeser ke kiri sebanyak empat kali

Untuk lebih jelasnya, aturan tersebut dapat ditunjukkan pada tabel di bawah ini.

Tabel 4 Operasi pada Blok 128 bit

S					S'			
$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$		$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	→	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,0}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$		$S_{2,2}$	$S_{2,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$		$S_{3,3}$	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$

Tabel 5 Operasi pada Blok 256 bit

S							
$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$	$S_{0,4}$	$S_{0,5}$	$S_{0,6}$	$S_{0,7}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,4}$	$S_{1,5}$	$S_{1,6}$	$S_{1,7}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$	$S_{2,4}$	$S_{2,5}$	$S_{2,6}$	$S_{2,7}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$	$S_{3,4}$	$S_{3,5}$	$S_{3,6}$	$S_{3,7}$
↓							
S'							
$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$	$S_{0,4}$	$S_{0,5}$	$S_{0,6}$	$S_{0,7}$
$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,4}$	$S_{1,5}$	$S_{1,6}$	$S_{1,7}$	$S_{1,0}$
$S_{2,3}$	$S_{2,4}$	$S_{2,5}$	$S_{2,6}$	$S_{2,7}$	$S_{2,0}$	$S_{2,1}$	$S_{2,2}$
$S_{3,4}$	$S_{3,5}$	$S_{3,6}$	$S_{3,7}$	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

2.1.3. Mixed Columns (Percampuran Kolom)

Pada operasi ini, blok ditransformasikan untuk setiap kolomnya. Setiap kolom diperlakukan sebagai *four-term polynomial* dengan cara Galois Field (GF) (2^8) dan dimodulokan dengan x^4+1 dengan polinom tetap $a(x)$ [2], yaitu $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$. Proses tersebut dapat dituliskan sebagai perkalian matriks sebagai berikut.

$$s'(x) = a(x) \otimes s(x)$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

dengan c adalah letak kolom, sehingga hasilnya

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$
$$s'_{0,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$
$$s'_{0,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$
$$s'_{0,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c})$$

Jika hasil perkalian memiliki lebih dari 8 bit, bit yang lebih tidak begitu saja dibuang. Hasil tersebut di-*xor* dengan 100011011_2 [3].

2.1.4. AddRoundKey (Penambahan Kunci)

Dalam operasi transformasi ini, digunakanlah upakunci untuk masing-masing putaran yang berasal dari kunci utama dengan menggunakan jadwal kunci Rijndael (*Rijndael's key schedule*) yang ukuran upakunci tersebut sama dengan ukuran blok yang akan diproses. Upakunci tersebut kemudian di-*xor* dengan blok input sehingga diperoleh hasilnya.

2.2. Pemakaian AES dalam Aplikasi Kompresi Berkas

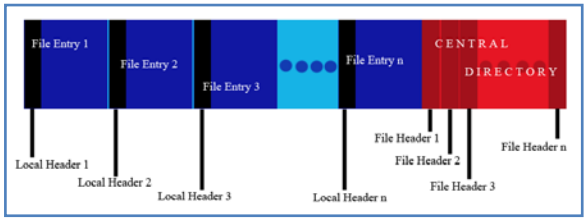
Saat ini ada berbagai macam produk pengompresi berkas. Setiap produk aplikasi berusaha untuk dapat mendukung dibukanya berkas yang telah terkompresi tersebut. Sebagai contoh, untuk mendukung enkripsi kompresi *zip* dengan AES, WinRAR dapat menangani kunci sepanjang 128 bit, IZArc 256 bit, sedangkan WinZIP mampu menangani 128 dan 256 bit [4]. Contoh lain, untuk penanganan enkripsi kompresi *7z*, aplikasi 7-Zip dan IZArc mampu menangani panjang kunci 256 bit sedangkan WinRAR 128 bit [5]. Sebenarnya tidak ada masalah untuk mengenkripsi berbagai macam jenis berkas kompresi. Hanya saja, aplikasi-aplikasi yang ada harus dapat menangani berbagai kemungkinan algoritma enkripsi.

Dalam pengenkripsian berkas dalam algoritma AES, tidak menutup kemungkinan diperlukan suatu algoritma lain untuk membantu, misalnya untuk membuat kuncinya. Sebagai contoh, untuk enkripsi *7z*, aplikasi 7-Zip tidak langsung menggunakan kunci sandi-lewat yang dimasukkan *user*. 7-Zip menjalankan suatu fungsi turunan berdasarkan algoritma *hash* SHA-256. Fungsi ini menghasilkan kunci 256 bit yang diperlukan untuk mengenkripsi dalam AES dari masukan kunci sandi-lewat yang dimasukkan *user* tadi. Untuk meningkatkan kesulitan pembobolan kunci dengan *exhaustive search*, 7-Zip juga menggunakan iterasi dalam jumlah besar ketika membuat *cipher key* tadi.

Pada pembahasan di makalah ini, dipilihlah contoh pengaplikasian berkas kompresi menggunakan format berkas *zip*. Ini karena saat ini *zip* telah dipakai oleh berbagai sistem operasi dan aplikasi-aplikasi. Akibatnya, terdapat banyak referensi yang cukup lengkap untuk membahas hal-hal yang menyangkut format berkas ini.

2.2.1. Struktur Format Berkas *zip*

Secara umum, struktur berkas *zip* terdiri atas kumpulan setiap berkas yang terkompresi di bagian depan dan *central directory* di bagian belakang. Adanya bagian *central directory* inilah yang menunjukkan bahwa suatu berkas merupakan suatu berkas *zip*. Bagian tersebut berfungsi untuk menunjukkan struktur berkas-berkas terkompresi di dalamnya seperti ketika tidak dikompresi. Selain itu, bagian tersebut menunjukkan posisi berkas pada bagian depan yang berisi berkas-berkas terkompresi. Untuk lebih jelasnya, dapat ditunjukkan pada gambar di bawah ini.



Gambar 3 Struktur Berkas *Zip*

Pada gambar di atas, kumpulan berkas-berkas terkompresi ditunjukkan oleh warna biru, sedangkan *central directory* ditunjukkan oleh warna ungu.

2.2.2. Enkripsi dalam Berkas *Zip*

Pada awalnya, *zip* mendukung enkripsi simetris berbasis *password* sederhana. Lama-kelamaan, setelah ditemukannya algoritma-algoritma baru untuk mengenkripsi berkas *zip*, akhirnya didukunglah algoritma-algoritma baru, salah satunya algoritma AES/Rijndael.

Aplikasi komersial pertama yang memperkenalkan enkripsi AES untuk berkas *zip* adalah WinZip versi 9.0 beta. Aplikasi ini mendukung panjang kunci 256 bit. Untuk membantu pengembang aplikasi kompresi *zip*, perusahaan pembuat WinZip membuat dokumentasi pengenkripsian AES-256 yang digunakan dalam makalah ini [6].

2.2.3. Pengaplikasian Enkripsi AES pada Berkas *Zip*

Setelah dipublikasikannya algoritma AES sebagai pemenang seleksi yang diadakan NIST, banyak orang yang mengaplikasikan algoritma ini. Salah satu pengaplikasinya adalah Dr. Brian Gladman. Ia membuat fungsi AES yang digunakan WinZip sebagai salah satu algoritma enkripsi berkasnya [6]. Struktur desain kode yang dibuat Gladman adalah sebagai berikut [7].

1. AES dalam mode *Counter* (CTR) untuk enkripsi
2. HMAC-SHA1 untuk autentikasi
3. Turunan kunci dari sandi-lewat dan nilai *salt* berdasarkan RFC2898

Kode yang dibuatnya memiliki 3 mode sebagai berikut (dengan ukuran panjang dalam *byte*) [7].

Tabel 6 Mode Enkripsi AES

Info	Mode 1	Mode 2	Mode 3
Panjang Sandi-lewat	8 – 31	32 – 47	48 – 63
Panjang *Salt*	8	12	16
Panjang Kunci AES	16	24	32
Panjang Kunci HMAC-SHA1	16	24	32
Panjang Verifikasi Sandi-lewat	2	2	2
Panjang Kode Autentikasi	10	10	10
Beban Panjang Berkas	20	24	28

Format berkas *zip* terenkripsinya antara lain [6]:

- Nilai *salt* (8, 12, atau 16 *byte*)
  Nilai *salt* merupakan barisan *byte* yang *random* atau *pseudo-random* yang dikombinasikan dengan sandi-lewat enkripsi untuk membuat kunci enkripsi dan autentikasi. Nilai ini dibangkitkan oleh aplikasi pengenkripsi dan disimpan tanpa terenkripsi dalam berkas.
  Nilai *salt* pada suatu berkas seharusnya berbeda dengan nilai *salt* pada berkas lain dengan *password* sama. Jika sama, penyerang yang sudah mengetahui isi salah satu berkas dapat memperkirakan isi dari berkas yang satunya.
- Nilai verifikasi sandi-lewat (2 *byte*)
  Nilai verifikasi ini diperoleh ketika dilakukan penurunan kunci enkripsi dan dekripsi dari *password*. Ketika enkripsi, nilai ini diperoleh dari sandi-lewat enkripsi dan disimpan tanpa dienkripsi dalam berkas terenkripsi. Sebelum dekripsi, nilai ini dapat diturunkan dari sandi-lewat dekripsi dan dibandingkan dengan nilai yang tersimpan dengan berkas untuk mengecek sandi-lewat yang salah.
- Isi berkas terenkripsi
  Proses enkripsi hanya diaplikasikan terhadap isi berkas saja. Hasilnya disimpan dalam bagian ini.
- Kode autentikasi (10 *byte*)

Kode ini digunakan untuk mengecek bahwa isi berkas terenkripsi belum pernah berubah dengan kurang hati-hati atau sengaja dirusak ketika dienkripsi pertama kali. Pada dasarnya, ini dapat merupakan pengecekan super-CRC setelah kompresi dan enkripsi.

3. SERANGAN-SERANGAN TERHADAP ENKRIPSI AES PADA BERKAS KOMPRESI TERENKRIPSI

3.1. Serangan melalui Percobaan Sandi-lewat

Menurut salah satu perusahaan pembuat aplikasi *password recovery*, LastBit, ada beberapa metode serangan yang digunakan untuk membobol sandi-lewat pada berkas kompresi *zip* yang terenkripsi. Metode-metode tersebut antara lain [8]:

a. *Brute Force Attack*
b. *Dictionary Attack*
c. *Password Variation*
d. *Known Plain Text Attack*
e. Metode lain yang dikembangkan sendiri

LastBit mengakui bahwa untuk melakukan pembobolan sandi-lewat dengan enkripsi AES memakan waktu yang lama. Hanya sandi-lewat dengan ukuran yang pendek saja yang bisa diperoleh dengan cepat. Perusahaan itu menyarankan untuk menggunakan *dictionary attack* untuk ukuran sandi-lewat yang panjang [8].

Berikut ini adalah penjelasan mengenai masing-masing serangan yang telah disebutkan di atas [9].

3.1.1. *Brute Force Attack*

Serangan ini merupakan cara yang paling mudah dilakukan dan paling dikenal. Caranya adalah dengan mencoba semua kombinasi karakter untuk semua kemungkinan panjang karakter yang digunakan. Sayangnya, untuk kasus terburuk, prosesnya bisa berlangsung sangat lama. Sebagai contoh, untuk menemukan kunci sepanjang 6 karakter dengan komponen hanya huruf kecil aja diperlukan 26^6 = 308915776 kali percobaan. Itu akan lebih besar lagi jika komponennya terdiri atas huruf kecil dan kapital, huruf, dan karakter lainnya. Sebagai contoh, misalnya dalam satu detik dapat dilakukan pengecekan 500.000 kunci maka diperoleh perbandingan waktu sebagai berikut.

Tabel 7 Perbandingan Waktu Percobaan Kunci

Panjang Kunci	Huruf Kecil Saja	Huruf Kecil dan Angka	Huruf Kecil dan Besar
< 4	cepat		
4	0,91 detik	3,36 detik	14,62 detik
5	23,76 detik	2,02 menit	12,67 menit
6	10,30 menit	1,21 jam	10,98 jam
7	4,46 jam	1,81 jam	23,80 jam
8	4,83 jam	2,18 bulan	3,44 tahun
9	4,19 bulan	6,53 tahun	178, 75 tahun

Pada percobaan di atas, masih belum menggunakan karakter-karakter lainnya sehingga waktu yang diperlukan dapat menjadi lebih lama lagi.

3.1.2. *Dictionary Attack*

Metode ini merupakan optimasi dari *brute force attack*. Program mencari semua kemungkinan kunci berdasarkan isi kamus. Jika ada kemungkinan kata tambahan yang dapat menjadi kunci, kata tersebut dapat ditambahkan pada kamus yang tersedia. Sayangnya, saat ini sudah banyak orang yang menghindari penggunaan kata-kata langsung dari kamus. Bisa saja mereka menggunakan tambahan beberapa karakter sebelum atau sesudah kata yang berasal dari kamus. Oleh karena itu, metode ini dapat diperbaiki dengan pencarian kata dalam kamus dengan penambahan beberapa karakter di awal atau di akhir kata dari kamus. Tentu saja cara ini memerlukan waktu yang lebih lama.

3.1.3. *Password Variation*

Metode ini sebenarnya bukan secara khusus digunakan untuk membobol, melainkan hanya digunakan untuk mencari *password* yang salah diketikkan *user*, misalnya tertekan *caps lock*, duplikasi karakter, hilangnya karakter, dan sebagainya. Tentu saja metode ini menggunakan *dictionary attack* untuk pencarian kata yang diinginkan *user*.

3.1.4. *Known Plain Text Attack*

Metode ini sangat berguna khususnya untuk berkas-berkas yang terkompresi. Pencarian dilakukan dengan perbandingan isi dari berkas kompresi terenkripsi tersebut dengan satu atau beberapa berkas yang diyakini sama dengan salah satu atau beberapa berkas yang terdapat di dalam berkas terkompresi tersebut. Jika berhasil diperoleh pola untuk satu berkas tersebut, maka dapat diperoleh pula pola untuk keseluruhan berkas yang terkompresi tersebut.

3.2. Metode Lainnya

Metode-metode lainnya pada dasarnya merupakan pengembangan dari *brute force*. Salah satunya adalah dengan menggunakan tabel statistik kata-kata yang terdapat dalam sejumlah besar teks. Metode ini mencegah digunakannya kata-kata yang jarang digunakan. Oleh karena itu, bisa saja tidak ditemukan kata yang lazim dalam kamus tetapi tidak terdapat dalam tabel tersebut. Namun, pencarian dengan metode ini tentunya lebih cepat daripada *dictionary attack*. Metode ini dikembangkan oleh LastBit dan dinamakan *smart force attack*.

Selain itu, metode pencarian *password* lainnya yang cukup lazim digunakan adalah dengan *birthday attack*. Pencarian dilakukan dengan mengkombinasikan angka-angka yang sesuai dengan kalender.

3.2. Serangan Langsung Berkas

Metode ini dilakukan dengan pencarian kelemahan algoritma enkripsi AES yang saat ini masih dikembangkan. Pencarian kelemahan tersebut tidak serta-merta menemukan berkas kompresi asli dari berkas kompresi yang terenkripsi. Oleh karena itu, hasil metode ini harus dibandingkan dengan struktur berkas kompresi yang telah disampaikan di atas.

Saat ini, algoritma AES/Rijndael dipercaya sangat kuat bagi sebagian besar orang. Ini disebabkan algoritma ini sangat kuat terhadap serangan akproksimasi klasik. Namun, karena algoritma ini sangat bersifat aljabar, serangan-serangan baru yang bersifat aljabar bermunculan [10].

Nicolas Courtois dan Josef Pieprzyg menunjukkan bahwa algoritma AES dapat ditulis sebagai sistem dari persamaan *multivariate quadratic* (MQ). Sebagai contoh, untuk AES 128 bit terdapat 8000 persamaan kuadratik dan 1600 variabel. Shamir dkk. mengembangkan algoritma bernama XL (*Extended Linearization*) untuk memperbaiki algoritma *relinearization* untuk membuat solusi sistem persamaan MQ. Mereka menganggap bahwa banyak persamaan yang dibangkitkan oleh algoritma *relinearization* bergantung secara linear dan kemudian algoritma tersebut kurang efisien sehingga dikembangkanlah XL yang lebih sederhana dan kuat daripada *relinearization* [11]. Dalam prakteknya, algoritma XL tidak berhasil untuk memecahkan algoritma AES. Namun, sistem yang dihasilkan oleh AES tidak acak dan memiliki keteraturan. Untuk itu, Courtois dan Pieprzyg melakukan pengembangan XL dan mengadaptasikannya menjadi sistem khusus, yang menjadi algoritma baru, yaitu XSL (*Extended Sparse Linearization*). Sayangnya, algoritma ini belum benar-benar dapat memecahkan AES meskipun tidak ada yang dapat membuktikan bahwa algoritma ini tidak akan bekerja [10].

Algoritma XL pada dasarnya dilakukan dengan memperbesar jumlah persamaan dengan mengalikannya dengan semua monomial pada derajat tertentu. Karena dianggap kurang mampu menyerang persamaan yang diturunkan dari *block cipher* seperti AES, algoritma ini diperbaiki dalam algoritma XSL. Algoritma ini tetap mengadopsi struktur khusus dari sistem persamaan yang dibentuk XL dengan kemudian mengalikan persamaan-persamaannya hanya dengan monomial tertentu.

Selain algoritma XL dan XSL di atas, dikembangkan pula suatu serangan lain yang dikembangkan oleh Alex Biryukov, Orr Dunkelman, Nathan Keller,

Dmitry Khovratovich, dan Adi Shamir. Metode ini mampu menyerang AES-256 hingga 10 putaran. Serangan tersebut hanya menggunakan dua kunci yang berelasi dan kompleksitas waktu 2^{39} untuk memperoleh kunci dari AES-256 dengan 9 putaran. Untuk 10 putaran, diperlukan kompleksitas waktu 2^{45}, sedangkan untuk 11 putaran diperlukan kompleksitas waktu 2^{70} dengan menggunakan jenis serangan subkunci yang lebih kuat [13]. Kompleksitas waktu ini jauh lebih baik daripada kompleksitas serangan-serangan lain sebelumnya yang sudah ada, yang ditunjukkan pada tabel di bawah ini [14].

Tabel 8 Serangan Lain Terbaik pada AES-192 dan AES-256

Serangan	Putaran	#Kunci	Data	Waktu
192				
Partial Sums	8	1	$2^{127.9}$	2^{188}
Related-key Rectangle	10	64	2^{124}	2^{183}
Related-key Amplified Boomerang	12	4	2^{123}	2^{176}
256				
Partial Sums	9	256	2^{85}	2^{226}
Related-key Rectangle	10	64	2^{114}	2^{173}
Related-key Differential	14	2^{35}	2^{96}	2^{96}
Related-key Boomerang	14	4	2^{119}	2^{119}

4. SARAN PENGGUNAAN SANDI-LEWAT DENGAN ALGORITMA AES UNTUK BERKAS TERKOMPRESI DAN PENCEGAHAN SERANGAN

Pada dasarnya, sangatlah sulit untuk memperoleh berkas asli dari suatu berkas terkompresi yang telah terenkripsi. Akan tetapi, dengan mengetahui struktur berkas kompresi dan struktur enkripsinya, cara memperolehnya dapat lebih mudah. Dari yang telah dijelaskan di atas, ada dua kemungkinan untuk memperoleh berkas asli tersebut, yaitu dengan menebak sandi-lewat yang dimasukkan pengguna seperti pada subbab 3.1 lalu melakukan proses dekripsi atau mengambil bagian isi berkas terenkripsi saja lalu melakukan percobaan dengan metode pada subbab 3.2 di atas.

4.1. Antisipasi Nilai *Salt*

Seperti yang telah dijelaskan sebelumnya, kunci enkripsi dan dekripsi bukan merupakan sandi-lewat yang dimasukkan oleh pengguna. Kunci enkripsi dan dekripsi tersebut dibangkitkan oleh nilai *salt*. Sebenarnya, hampir tidak ada masalah jika nilai ini benar-benar *random*. Akan tetapi, jika *pseudorandom*, penyerang dapat memperkirakan isi berkas lebih mudah karena kunci yang dibangkitkan memiliki pola yang sama. Apalagi nilai *salt* tidak disembunyikan dalam berkas terenkripsi.

Dalam hal ini, pengguna aplikasi kompresi tidak dapat berbuat apa-apa untuk mengatur nilai *salt*. Akan tetapi, untuk mempersulit, tentu saja pengguna dapat menggunakan sandi-lewat yang jumlah karakternya panjang. Untuk membangkitkan nilai *salt*, sebisa mungkin aplikasi kompresi membuat barisan nilai yang benar-benar acak.

4.2. Antisipasi Serangan *Brute Force*

Serangan ini sebenarnya pasti dapat memperoleh sandi-lewat yang dimasukkan pengguna. Akan tetapi, waktu yang diperlukan untuk memperolehnya sangat lama seperti yang ditunjukkan pada tabel 7. Oleh karena itu, pengguna sebaiknya memasukkan sandi-lewat sepanjang-panjangnya dengan kombinasi huruf kapital, huruf kecil, angka, dan karakter lainnya asalkan dapat diingat. Hal ini untuk mengantisipasi kemajuan teknologi perangkat keras yang bisa memproses percobaan kunci yang lebih cepat.

4.3. Antisipasi *Dictionary* dan *Birthday Attack*

Seperti yang telah diketahui, serangan ini menggunakan kata-kata dalam kamus dan tanggal dalam kalender. Oleh karena itu, pemilihan sandi-lewat sebaiknya menghindari kata-kata tersebut. Jika ingin menggunakan unsur-unsur kata dari kamus atau kalender, sebaiknya tidak mentah-mentah menggunakannya, melainkan dikaburkan dengan diubah urutan karakternya atau disisipkan karakter lain.

4.4. Antisipasi Variasi Sandi-lewat

Serangan dengan menggunakan variasi ini sebenarnya didasarkan atas kesalahan pengguna memasukkan kesalahan pengetikan untuk sandi-lewat berupa kata-kata dari kamus. Oleh karena itu, seperti antisipasi *dictionary attack* di atas, pengguna tidak menggunakan kata-kata dari kamus secara mentah-mentah dan melakukan kesalahan pengetikan dengan sengaja dan mengingatnya dengan baik.

4.5. Antisipasi Serangan *Known Plain Text*

Dalam antisipasi ini, pengguna aplikasi kompresi juga tidak dapat mencegah terjadinya serangan dengan pemilihan sandi-lewat dengan aturan tertentu. Perhatian untuk pencegahan serangan ini sebaiknya dilakukan oleh pengembang aplikasi kompresi supaya tidak terjadi kemungkinan kemunculan kesamaan pola berkas, misalnya kesamaan nilai *salt* atau kemungkinan nilai kunci enkripsi yang sama antara dua berkas yang berbeda.

4.6. Antisipasi Serangan Langsung Berkas

Untuk mengantisipasi serangan ini, pengguna juga tidak dapat berbuat apa-apa selain pemilihan sandi-lewat seperti yang telah disebutkan sebelumnya. Untunglah hingga saat ini percobaan pemecahan algoritma AES baru berhasil pada AES-256 dengan 10 putaran (untuk 11 putaran belum dapat dipraktekkan) sedangkan pada kompresi *zip* oleh WinZip dari Gladman menggunakan 14 putaran. Namun, pada suatu saat, harus ada penetapan jumlah putaran yang lebih banyak. Bruce Schneier, seorang kriptografer dan spesialis keamanan komputer, menyarankan sebaiknya saat ini AES-128 menggunakan 16 putaran, AES-192 dengan 20 putaran, dan AES-256 dengan 28

putaran. Selain itu, akibat dengan ditemukannya serangan baru oleh Biryukov dkk di atas, sebaiknya dihindari penggunaan AES-256. Akan tetapi, jika masih digunakan tidak masalah karena jumlah putarannya masih aman [12].

5. KESIMPULAN

Algoritma AES/Rijndael merupakan algoritma yang cukup kuat untuk saat ini. Banyak aplikasi yang telah menerapkan enkripsi dengan algoritma ini, salah satunya adalah aplikasi kompresi. Meskipun masih belum ada yang benar-benar dapat membongkar algoritma ini, para pengguna dan pengembang sebaiknya selalu waspada. Para pengguna enkripsi algoritma ini untuk mengompresi berkas mereka sebaiknya selalu menggunakan sandi-lewat yang berukuran panjang, setidak-tidaknya 8 karakter dengan kombinasi tidak hanya huruf kecil atau kapital saja. Selain itu, sebaiknya dihindari penggunaan kata-kata yang ada di dalam kamus atau tanggal. Meskipun rumit, sandi-lewat juga sebaiknya mudah diingat dan tidak diketahui dengan mudah oleh orang yang tidak berhak untuk keamanan data itu sendiri. Bagi pengembang, sebaiknya selalu mengikuti perkembangan serangan algoritma enkripsi saat ini dan melakukan perbaikan pada enkripsinya dan mempublikasikan pada masyarakat luas, termasuk pada pengembang lain.

DAFTAR REFERENSI

[1] Advanced Encryption Standard (AES),
URL:http://www.informatika.org/~rinaldi/Kriptografi/2006-2007/Advanced Encryption Standard (AES).ppt
Tanggal akses: 5 Februari 2010 17:24

[2] Specification for the Advanced Encryption Standard (AES),
URL:http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
Tanggal akses: 31Desember 2008 22:06

[3] The Advanced Encryption Standard (Rijndael),
URL:http://www.quadibloc.com/crypto/co040401.htm
Tanggal akses: 17 Maret 2010 01:17

[4] File Extension Zip,
URL:http://www.file-extensions.org/zip-file-extension
Tanggal akses: 19 Maret 2010 06:15

[5] File Extension 7zip,
URL:http://www.file-extensions.org/7zip-file-extension
Tanggal akses: 16 Maret 2010 15:46

[6] AES Encryption Information,
URL: http://www.winzip.com/aes_info.htm
Tanggal akses: 16 Maret 2010 16:18

[7] A Password Based File Encryption Utility,
URL:http://www.gladman.me.uk/cryptography_technology/fileencrypt/
Tanggal akses: 16 Maret 2010 16:13

[8] Zip Password,
URL: http://lastbit.com/zippsw/default.asp
Tanggal akses: 16 Maret 2010 16:15

[9] Password Recovery Method,
URL: http://lastbit.com/msohelp/prm.htm
Tanggal akses: 16 Maret 2010 16:20

[10] Is AES a Secure Cipher?
URL: http://www.cryptosystem.net/aes/
Tanggal akses: 19 Maret 2010 12:46

[11] Efficient Algorithm for Solving Overdefined Systems of Multivariate Polynomial Equation
URL:http://www.iacr.org/archive/eurocrypt2000/1807/18070398-new.pdf
Tanggal akses: 22 Maret 2010 09:02

[12] Another New AES Attack
URL:http://www.schneier.com/blog/archives/2009/07/another_new_aes.html
Tanggal akses: 19 Maret 2010 12:52

[13] Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds
URL: http://eprint.iacr.org/2009/374.pdf
Tanggal akses: 22 Maret 2010 06:33

[14] Related-key Cryptanallysis of the Full AES-192 and AES-256
URL:https://cryptolux.org/mediawiki/uploads/1/1a/Aes-192-256.pdf
Tanggal akses: 19 Maret 2010 12:50