SERANGAN BARU PADA ALGORITMA CELLULAR AUTHENTICATION AND VOICE ENCRYPTION

Brian Al Bahr – NIM : 13506093

Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if16093@students.if.itb.ac.id

Abstrak

Makalah ini menyajikan kripanalisis baru terhadap algoritma *hash* telekomunikasi yang dikenal dengan sebutan Algoritma *Cellular Authentication and Voice Encryption* (CAVE). Serangan terhadap algoritma ini sebelumnya telah diketahui membutuhkan evaluasi terhadap CAVE-4 sebanyak 2^{91} untuk menemukan satu buah *pre-image* yang memenuhi redundansi masukan. Serangan ini mencari sebuah *pre-image* dengan usaha yang ekivalen dengan mengevaluasi 2^{11} instansiasi dari CAVE 4 ronde. Metode ini diulang sebanyak 2^{80} kali untuk mendapatkan hanya sebuah contoh dari masukan data yang memiliki redundansi yang konsisten dengan tahap pemrosesan masukan dari sebuah aplikasi CAVE. Kompleksitas total untuk serangan pada CAVE 4 ronde adalah $2^{11} * 2^{80} = 2^{91}$. Sebagai perbandingan, kompleksitas dari serangan baru terhadap algoritma ini membutuhkan usaha yang ekivalen atau kurang dari 2^{72} evaluasi dari CAVE 4 ronde untuk memperoleh seluruh *pre-image*. Pada CAVE 8 ronde, metode lama membutuhkan usaha yang ekivalen dengan evaluasi sebanyak $2^{13} * 2^{80} = 2^{93}$ (8 kali lebih banyak dari pemecahan CAVE 4 ronde secara utuh) instansiasi untuk mencari sebuah *pre-image*. Sebagai perbandingan, serangan baru ini membutuhkan usaha yang ekivalen dengan 2^{72} evaluasi dari algoritma. Seiring dengan bertambahnya jumlah ronde pada CAVE, keuntungan relatif dari serangan baru terhadap serangan yang lama juga meningkat. Pengurangan usaha yang signifikan ini membuat serangan baru ini lebih mengancam bagi CAVE pada praktiknya.

Kata Kunci : *CAVE, pre-image, CDMA, IS-41C, IS-54, Cellular Authentication and Voice Encryption, wireless network.*

1. Pendahuluan

CAVE adalah sebuah primitif kriptografi yang disetujui oleh *Telecommunication International Association* (TIA) untuk digunakan untuk otentikasi , perlindungan data, dan anonimitas dari generasi kedua jaringan *Code Division Multiple Access* (CDMA). CAVE juga digunakan untuk menyediakan keamanan untuk telepon genggam Amerika IS-41C dan sistem selular IS-54. Algoritma CAVE mengotentikasi pelanggan yang sah ke jaringan CDMA dan mencegah jaringan dan pelanggan dari kekacauan pada jaringan.

Telah diketahui dalam industri telekomunikasi bahwa CAVE memiliki kelemahan. Serangan pertama yang diketahui terhadap CAVE dipresentasikan di Milan pada tahun 1998 yang mendemonstrasikan bahwa CAVE tidak bisa dianggap sebagai fungsi hash yang aman. Serangan tersebut diyakini memiliki kompleksitas waktu sebanyak 2^{91}. Keputusan untuk mengganti Algoritma CAVE dengan AKA pun disetujui pada tahun 1999. Namun, Karena proses standardisasi dan adopsi yang lambat, penggantian algoritma inipun terhambat. Serangan baru yang akan dibahas di bawah akan memiliki kompleksitas waktu yang jauh lebih kecil sehingga ancaman pun akan meningkat secara eksponensial.

2. Pembahasan

2.1. Algoritma CAVE

Komponen utama dari algoritma CAVE adalah 16 data *register* 8-bit, 2 buah *offset* 8-bit yaitu *offset_1* dan *offset_2*, serta sebuah *Linear Feedback Shift Register* (LFSR). CAVE dioperasikan dalam 4 atau 8 ronde tergantung dari aplikasi spesifik dengan setiap ronde memiliki fase update 16 *register*. LFSR 32-bit berisi 4 *byte register* yang terpisah, yaitu: $LFSR_A$, $LFSR_B$, $LFSR_C$, dan $LFSR_D$ dengan *feedback* primitif polinomial yang didefinisikan dengan fungsi :

$$L_{t+32} = L_t \oplus L_{t+1} \oplus L_{t+2} \oplus L_{t+22}$$

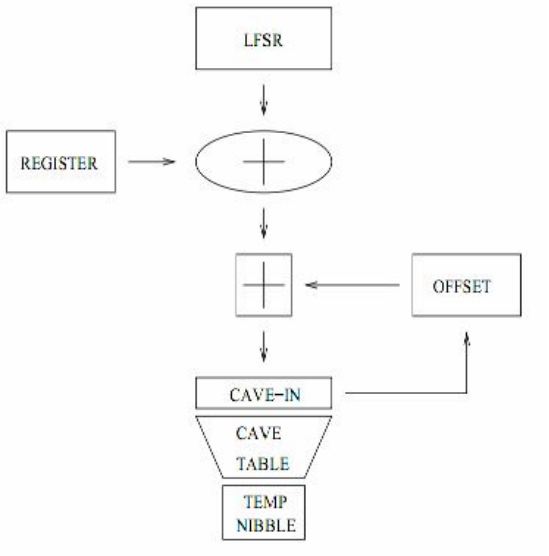
Untuk setiap fase, CAVE menggunakan *byte-byte* dari LFSR, *offset*, dan 2 buah LUT atau SBox ukuran 8 * 4 untuk memodifikasi 1 *register*. *Offset offset_1* dan *offset_2*

digunakan sebagai *pointer* pada tabel CAVE tinggi atau rendah yang direpresentasikan sebagai CT_low dan CT_high. Langkah-langkah yang terjadi dalam segmen rendah pada CAVE adalah :

$$offset\_1 = offset\_1_{prev} + (LFSR_A \oplus sreg[i]) \bmod 256 \quad (1)$$

$$temp\_low = CT\_low[offset\_1] \quad (2)$$

Langkah untuk segmen tinggi adalah sama seperti segmen rendah. Operasi segmen ditunjukkan pada gambar berikut:



Gambar 1 Operasi Segmen pada CAVE

Byte offset_1$_{prev}$ merepresentasikan nilai sebelumnya dari offset byte yang diinisialisasikan sebagai konstanta. CAVE terus melakukan memutar LFSR secara linear ke kanan ketika *nibble*-nya menjadi sama dengan urutan bit tinggi atau rendah dari *sreg*[i] yang berkorespondensi, dimana *i* adalah fase ke –*i* dari ronde. Jika *nibble* dan urutan bit tinggi atau rendah menjadi tidak sama, CAVE melakukan komputasi *byte temp* dengan melakukan konkatenasi *nibble temp_low* dan *temp_high* dan bergerak ke fase berikutnya dari ronde. Jika nilai yang dibandingkan menjadi sama, akan ada siklus ekstra dari LFSR dan kalkukasi di atas akan diulangi dengan byte LFSR paling baru dan nilai *offset*. Jika pada kasus khusus dimana siklus ekstra ini mencapai 32, maka *byte* $LFSR_D$ diinkremen modulo 256. Setelah fase selesai, LFSR akan melakukan siklus sekali yang akan menghasilkan minimum dari 16 pergeseran (*shift*) pada setiap ronde CAVE. Antara ronde, bit-bit pada *register* diacak dengan menggunakan tabel CAVE rendah untuk mendefinisikan permutasi *byte* diikuti dengan rotasi 1 bit pada blok register 128-bit.

2.2. Serangan Pada CAVE

Serangan pada CAVE menunjukkan bahwa CAVE sudah bukan lagi fungsi *hash* yang aman. Untuk 128-bit nilai *hash*, telah ditunjukkan bahwa *input* (*pre-image*) algoritma dapat ditemukan dengan usaha yang ekivalen dengan 2_{13} eksekusi dari CAVE 8 ronde atau 2_{11} eksekusi dari CAVE 4 ronde. Serangan ini bekerja dengan pertama-tama menebak 32-bit nilai LFSR dan membangkitkan sejumlah siklus LFSR yang dibutuhkan yang cukup untuk serangan dengan menggunakan polinomial *feedback* primitif. Ronde terakhir (ronde 0) direkonstruksi dengan menebak 2 *bytes offset* dan *sreg*[0] dan melakukan validasi terhadap tebakan menggunakan persamaan “cek sanitasi” *ereg[15]* □ *temp[15]* = *sreg[0]* dimana *sreg*[0] adalah nilai dari data *register* 0 pada permulaan dari ronde terakhir dan *ereg*[15] adalah nilai final dari data *register* 15 pada akhir ronde tersebut.

Ketika nilai final dari offset sudah diketahui dengan rekonstruksi ronde 0, ronde sebelumnya (ronde > 0) direkonstruksi dengan arah yang berlawanan.dari algoritma dengan menebak nilai *sreg*[15] pertama dari ronde 1 (CAVE 4 ronde beroperai dari ronde 3 ke ronde 0). Validitas dari tebakan ini dapat dicek dengan menggunakan persamaan ”cek sanitasi” *temp[15]* □ *ereg[15]* = *sreg[0]* dan ketika nilai ini valid, makan ronde tersebut telah direkonstruksi dengan benar. Dengan cara ini, ronde lain dapat direkonstruksi dengan arah yang berlawanan. Algoritma ini menghasilkan set data tunggal dimana input CAVE menghasilkan 128-bit nilai *hash* yang diberikan.

2.3. Kelemahan CAVE

Pada upabab ini, penulis akan membahas mengenai kelemahan CAVE yang kemudian dapat digunakan sebagai dasar serangan baru pada CAVE. Kelemahan berikut tidak tereksploitasi oleh serangan sebelumnya. Dengan data-data di bawah, kita dapat meningkatkan efisiensi dari serangan sebelumnya.

2.3.1. Ketidakseimbangan Tabel CAVE

Baris-baris pada tabel CAVE adalah permutasi tapi kolom-kolomnya tidak. Jadi, untuk keluaran *nibble* yang diberikan dari tabel CAVE, *offset* bit masukan urutan rendah ke tabel CAVE tidak terdistribusi secara merata. Kita akan menyebut

ketidakseimbangan pada *offset* masukan urutan rendah ini sebagai "*nibble imbalance*". Dengan data masukan atau tebakan yang kecil, properti CASE ini akan membantu untuk menentukan nilai yang tidak diketahui dengan probabilitas yang lebih besar daripada hanya menebak. Ketidakseimbangan *nibble* pada CAVE rendah dan tinggi dapat dilihat pada tabel 1 dan 2. Serangan baru akan menggunakan properti ini secara langsung.

Tabel 1 Ketikdakseimbangan pada tabel

	Frequency of low nibble inputs giving specified low nibble output						
low nibble output	0	1	2	3	4	5	6
0	0,3,6,8,A,B	1,4,5,7,D,E	2,9,F	-	C	-	-
1	1,2,4,7,9,A,B,E,F	5,6	8,D	0,C	3	-	-
2	1,2,5,6,C,D	0,3,4,A,E,F	B,7	8,9	-	-	-
3	0,3,8,9,B,F	1,4,7,A,C,D	5,E	2,6	-	-	-
4	4,8,A,C,F	0,1,2,3,7,9,D,E	5	6,B	-	-	-
5	1,2,5,6,C,F	0,3,4,9,A,D,E	7	B	8	-	-
6	4,6,7,8,9,A,B,C,F	0,2,E	3,5	D	-	-	1
7	7,9,B,D,E,F	0,2,6,8,A,C	1,3	4,5	-	-	-
8	3,6,7,8,9,C,F	0,1,4	2,5,A,D,E	B	-	-	-
9	2,4,9,A,D	1,3,5,7,8,B,C,E	F	0,6	-	-	-
A	5,6,8,A,B	1,4,9,D,E,F	0,2,3,7,C	-	-	-	-
B	1,3,5,7,B,C	0,2,4,6,A,E	D,F	8,9	-	-	-
C	0,1,2,5,8,C	3,4,7,9,B,D,E	6,F	-	-	A	-
D	0,4,5,8,C,D	1,2,3,6,B	7,9,A,F	E	-	-	-
E	1,3,9,A,D,E	0,2,4,5,6,8,B	7	C	F	-	-
F	0,3,6,D,E,F	1,2,5,7,8,B	9,A,C	-	4	-	-

CAVE Rendah

Tabel 1 menunjukkan frekuensi dari masukan bit *nibble* rendah dengan tabel CAVE rendah memberikan keluaran *nibble temp_low* tertentu. Tabel 2 serupa dengan tabel 1, hanya saja kali ini untuk merepresentasikan frekuensi bagi table CAVE yang tinggi. Rata-rata, terdapat 6 urutan bit masukan rendah ke tabel CAVE,

	Frequency of high nibble inputs giving specified high nibble output					
high nibble output	0	1	2	3	4	5
0	1,4,6,9	3,5,7,8,A,B,C,D,F	0,2	E	-	-
1	0,1,2,A,F	5,6,7,8,9,B,D	3,C,E	4	-	-
2	0,3,7,9,A,E	1,2,4,5,B,D	6,C	8,F	-	-
3	0,2,3,6,7,D,F	4,5,A,E	8,9,B,C	-	1	-
4	2,5,8,A,B,C,D,E	3,4,9	0,1,6	F	7	-
5	4,7,9,D,E,F	1,2,5,8,C	0,3,6,B	A	-	-
6	0,2,3,8,9,D,F	1,A,C	4,5,6,B,E	7	-	-
7	0,4,7,C,E,F	2,3,5,6,8,A,B	9	1	D	-
8	4,C,D,F	0,1,2,6,8,9,A,B,E	3,5	7	-	-
9	8,A,B,D,F	1,2,3,6,7,9,C	0,5,E	4	-	-
A	0,1,3,4,7,A,C,E	6,8,B,D	5	2,9	F	-
B	3,5,6,8,A,B,E	1,2,4,7,9,F	C	0	-	D
C	0,1,2,5,F	3,4,8,9,B,D,E	7,A,C	6	-	-
D	1,4,6,7,B,C	0,2,5,8,9,A	D,F	E,3	-	-
E	1,4,6,7,9,D,E	3,5,8,C	0,2,B,F	-	A	-
F	5,6,7,D,F	0,1,3,A,B,C,E	2,8,9	4	-	-

dengan 5 sebagai nilai minimum dan 9 sebagai nilai maksimum, yang tidak memberikan nilai keluaran *nibble temp_low* tertentu. Sama seperti tabel CAVE rendah, pada kasus rata-rata, terdapat 6 urutan masukan rendah ke tabel CAVE, dengan 4 sebagai nilai minimum dan 8 sebagai nilai maksimum. Observasi ini mengindikasikan distribusi kemungkinan data yang tidak seragam pada tabel CAVE ketika bekerja secara kebalikan. Serangan menggunakan informasi ini untuk memeriksa nilai kandidat yang paling mungkin sebelum kandidat lain.

2.3.2. Korelasi antara *byte* LSFR

Bit-bit dari $LFSR_A$ ($\{L_0,L_1, \ldots ,L_7\}$) sebelum fase dimulai akan digunakan kembali dalam $LFSR_B$ ($\{L_8,L_9, \ldots ,L_{15}\}$) setelah 8 siklus LFSR dimana L_7 akan berpindah ke posisi MSB dari $LFSR_B$ untuk setiap siklus. *Byte-byte* LFSR juga tidak bergantung pada *offset* dan *register*. Jadi, himpunan $\{L_0,L_1, \ldots ,L_7\}$ pada waktu *t*, himpunan $\{L_8,L_9, \ldots ,L_{15}\}$ benar-benar dispesifikasikan setelah waktu t + 8. Relasi ini dapat diekspresikan sebagai berikut.

For $\Delta t = 1$, $Ln(t) = Ln{-}1(t - 1)$

For $\Delta t = 4$, $Ln(t) = Ln{-}4(t - 4)$

For $\Delta t = 8$, $Ln(t) = Ln\ 8(t - 8)$

Serangan yang baru akan menggunakan nilai yang diketahui dari LFSR dari tabel yang telah dikomputasi sebelumnya dan memeriksa persamaan di atas untuk setiap siklus pada LFSR.

Tabel 2 Ketikdakseimbangan pada tabel CAVE Tinggi

2.4. Serangan Baru Pada CAVE

Seperti yang telah dijelaskan sebelumnya, pendekatan serangan yang baru ini akan menggunakan *look-up-table* (LUT) yang telah dihitung sebelumnya yang mendefinisikan operasi dari segmen dalam CAVE. Kemudian, untuk 128-bit keluaran *hash* (nilai final untuk *register byte*). tabel tadi akan digunakan untuk membimbing proses yang mempertahankan *list* dari semua data yang konsisten. Kita membangkitkan *list* ini dari semua segmen berurutan dalam sebuah fase, kemudian fase yang berurutan dalam sebuah ronde. *Data set* yang dihasilkan setelah hanya 2 fase dapat menentukan setengah dari bit-bit LFSR yang tidak diketahui. Dengan cara yang sama, proses dapat diteruskan pada lebih banyak fase sampai ke awal algoritma. Algoritma CAVE melakukan *hash* dari 176 bit sampai menjadi 128 bit. Jadi, dapat diduga bahwa setiap keluaran mempunyai 2^{48} *preimage*. Kita dapat membuat 24-bit tebakan setiap waktu sehingga kita memperkirakan dapat memiliki 2^{24} elemen dalam *list* pada setiap kejadian. Untuk mengurangi waktu *running* pada praktiknya, pertama-tama kita melakukan komputasi LUT yang merepresentasikan *list* dari operasi yang paling sering diulangi pada CAVE.

2.4.1. Tabel CAVE

Setiap segmen dari CAVE mengambil 24 bit masukan : menggunakan operasi eksklusif OR pada *byte* dari *register* data masukan dan *byte* LFSR diikuti dengan modulo 256. Operasi ini memberikan *byte offset* baru yang kemudian masukan ke tabel CAVE tinggi / rendah akan memberikan *temp nibble* tinggi / rendah.

LUT (1 dan 2) dapat dikonstruksi menggunakan komputasi yang melelahkan pada operasi yang esensial pada segmen CAVE. *Offset*, *byte* LFSR, dan masukan *data register*-nya dijumlahkan sampai 24 bit sehingga menghasilkan 2^{24} nilai mungkin yang berbeda. Komputasi pada ke 2^{24} nilai masukan yang mungkin ini menghasilkan *byte offset* pada setiap segmen yang merupakan masukan bagi tabel CAVE. Keluaran dari tabel CAVE ini adalah 4-bit *nibble candidate temp_low* / *nibble temp_high* dan sebuah *counter* bagi "siklus ekstra" yang bertindak sebagai *flag*. Nilai 1 pada *flag* mengindikasikan nilai dari masukan ke tabel CAVE, agar *temp nibble* tinggi / rendah mendapat padanan dengan urutan bit rendah / tinggi dari *sreg*[i] dimana *i* merepresentasikan fase tertentu dalam ronde CAVE. Nilai *temp nibble* adalah kandidat karena jika siklus ekstra terindikasi, maka CAVE akan melakukan siklus LFSR sekali, sehingga mengubah *byte* LFSR yang digunakan pada kalkulasi di atas dengan menghilangkan bit LSB, mendapatkan bit MSB dan menggeser 7 bit lainnya sebanyak 1 bit. Ketika *count* dari siklus ekstra sama dengan 0, makan *temp nibble* pada saat itu akan digunakan. Maka, kedua LUT ini terdiri dari 24-bit, sebuah keluaran *byte offset*, sebuah *nibble temp*, dan sebuah *counter* siklus ekstra.

2.4.2. Serangan yang Dilakukan

Keseluruhan serangan pada algoritma dapat dideskripsikan sebagai berikut.

Tahap Komputasi LUT

- **Inisialisasi :** Ulangi, untuk semua 2^{24} nilai dari *sreg*[15] dan pasangan dari *byte offset*

- **Langkah 1 :** Gunakan LUT untuk mencari *list* masukan yang valid untuk kedua segmen dalam 2 fase yang berurutan

- **Langkah 2 :** Untuk setiap fase, kombinasikan kedua segmen data ke dalam *list* dari data yang valid untuk fase tersebut

- **Langkah 3 :** Kombinasikan *list* fase yang berdekatan ke dalam 1 *list* untuk pasangan fase

- **Selesai :** Kombinasikan *list* yang tersisa, perhatikan konsistensi untuk menentukan *list* dari semua masukan valid yang mungkin

Berikut operasi di atas secara lebih detail.

Inisialisasi

Langkah ini melibatkan pemilihan nilai 24-bit yang membuat 2 *offset* dan *start register byte*. Untuk CAVE 4 ronde, dengan menggunakan 128-bit nilai hash yang sudah diketahui, nilai dari *register* akhir *ereg*[0, 1, 2 . . . 15] dari ronde 0 dapat ditemukan menggunakan permutasi *byte* ronde kebalikan (*reverse*) pada 128-bit nilai *hash* diikuti dengan pergeseran ke kiri secara sirkuler dari *register* yang bercampur. Dengan menebak nilai masukan data *register* dari fase 15 (*sreg*[15]) pada ronde 0, keluaran *byte* $temp_{14}$ dari fase 14 dapat ditentukan menggunakan ekspresi

$ereg[14] \square sreg[15] = temp_{14}$.

Nilai $temp_{14}$ adalah konkatenasi dari keluaran *nibble* tinggi/rendah $temp_{14}$ dari tabel CAVE. *Nibble* ini direpresentasikan sebagai $temp\_low_{14}$ dan $temp\_high_{14}$. Karena *nibble* ini adalah keluaran final dari segmen tinggi dan rendah dari fase 14 pada ronde 0 untuk algoritma CAVE, maka $offset\_1_{14}$ dan $offset\_2_{14}$ yang telah diberikan *nibble* tidak dapat menghasilkan siklus ekstra.

Hal ini mengimplikasikan bahwa

$temp\_low_{14} != sreg[14]\&0x0F$

dan

$temp\_high_{14} != sreg[14]\&0xF0$.

Ada 16 kemungkinan nilai yang mungkin untuk $offset\_1_{14}$ dan $offset\_2_{14}$ yang dapat memberikan nilai nibble ini karena adanya permutasi baris pada tabel CAVE. Serangan ini melibatkan pengetesan setiap nilai dari *byte offset*.

Langkah 1

Langkah pertama dalam analisis utama proses adalah evaluasi dari nilai 24-bit berbeda yang mungkin diakses dari LUT 1 dan 2 untuk setiap nilai pilihan data dalam segmen yang memenuhi persamaan (1). Tes ini dilakukan secara bersama-sama pada kedua segmen rendah dan tinggi. Nilai 24-bit yang dipertimbangkan pada segmen tinggi dan rendah harus memiliki *byte* yang sama dengan *sreg*[14]. Kondisi kunci ini pada seleksi nilai 24-bit menghasilkan *list* yang lebih pendek dari vektor 24-bit yang mengevaluasi persamaan (1) untuk mendapatkan $offset\_1_{14}$ dan $offset\_2_{14}$. Kondisi persamaan pada *byte register* menghasilkan *list* dengan nilai sekitar 2^{24} yang melakukan persamaan (1) dimana hal ini merupakan pengurangan sebesar 50% dari *list* semula yang membutuhkan $2 * 2^{24}$ nilai (karena ada 2 segmen CAVE).

Langkah 2

Offset yang sebelumnya digunakan untuk kalkulasi *temp nibble* sebagai berikut :

$CT\_ low[o\square set\_1_{14prev}] = temp\_low_{prev}$

$CT\_ high[o\square set\_1_{14prev}] = temp\_high_{prev}$

Temp_prev dikalkulasi dengan melakukan konkatenasi *temp nibble*. Validitas dari tebakan kita pada data 24-bit yang diperoleh dari LUT diperiksa menggunakan persamaan "*sanity check*":

$temp\_13 = ereg[13] \square sreg[14]$.

Secara umum direpresentasikan sebagai :

$temp\_i = ereg[i] \square sreg[i + 1]$.

Pada langkah ini *list* berkurang sampai sekitar 2^{16} nilai yang merupakan pengurangan yang signifikan dari list yang sebelumnya berjumlah sebanyak $2 * 2^{24}$ nilai.

Langkah 3

Langkah di atas diulang untuk 2 fase terakhir yang berdekatan dari ronde terakhir untuk mendapatkan *list* yang berkurang dari setiap fase. *List* tersebut kemudian dicek kompatibilitasnya dengan menggunakan properti korelasi antara *byte* LFSR seperti yang telah dideskripsikan di atas dan juga penggunaan dari *offset* final dari 1 fase sebagai nilai *offset* permulaan pada fase berikutnya. Rekonstruksi *backward* dari 4 segment pada 2 fase sudahlah cukup untuk membangun nilai setengah jumlah bit dari

LFSR dan 2 offset yang digunakan dalam fase tertentu. Serangan kemudian mundur (*backward*) ke fase lain dengan informasi diketahui yang lebih banyak sehingga mengurangi kompleksitas untuk iterasi berikutnya. Pada proses ini, *list* akan berkurang sampai pada akhirnya berakhir dengan *list* data yang valid yang digunakan pada permulaan ronde 3.

2.5. Analisis Kompleksitas

Untuk memperkirakan kompleksitas serangan ini sehingga bisa dibandingkan dengan serangan sebelumnya, kita menghitung kompleksitas dari setiap langkah menggunakan unit yang ekivalen dengan mengevaluasi fase lengkap dari CAVE, dengan catatan bahwa terdapat 16 fase pada setiap ronde CAVE. Karena pemrosesan *list* dengan LUT lebih tidak kompleks dibandingkan mengeksekusi fase dari CAVE, kita dapat mengembangkan *upper bound* kompleksitas dari serangan menggunakan kompleksitas yang ekivalen dengan fase sebagai unit dasarnya. Langkah 1 memiliki kompleksitas kurang dari 2^{24} unit, untuk setiap 2 fase pada setiap 2 segmen yang berurutan sehingga menghasilkan usaha total sebanyak 2^{26}. Langkah 2 membutuhkan usaha sekitar 2^{25} untuk setiap 2 fase sehingga dibutuhkan usaha sebanyak 2^{26} juga, untuk menjalankan 2^{27} unit yang ekivalen dengan fase. Hanya untuk kali pertama, langkah 3 harus mempertimbangkan setiap pasangan dari 2 *list* segmen, masing-masing dengan ukuran elemen 2^{16}, menghasilkan total 2^{32} operasi. Ongkos ini mendominasi kompleksitas dibandingkan 2 langkah sebelumnya, sehingga kita dapat menetapkan *upper bound* kompleksitas untuk mencari data yang konsisten di sepanjang 2 fase yang berurutan adalah 2^{33} unit. *List* menjadi berukuran 2^{24}, sehingga menggabungkannya membutuhkan ongkos 2^{48}. Kita menggunakan ini sebagai upper bound untuk kompleksitas untuk setiap fase pada serangan ini (64 fase padaCAVE-4). Karena setiap kalkulasi harus dilakukan 24 kali (dengan pilihan inisial yang berbeda untuk pasangan *byte offset* dan register permulaan pada tahap Inisialisasi), kita dapat memperkirakan bahwa usaha untuk mencari semua data yang valid untuk CAVE 4 ronde kurang dari $2^{48} * 2^6 * 2^{24} = 2^{78}$. Jumlah ini jauh dibandingkan serangan sebelumnya yang membutuhkan ongkos sebesar 2^{91}. Usaha untuk serangan pada CAVE 8 ronde mudah : perlu 248 * 26 usaha lainnya untuk setiap 2^{24} menghasilkan ongkos ekstra sebesar 2^{78} atau 2 kali nilai ongkos yang dibutuhkan untuk memecahkan CAVE 4 ronde. Untuk membandingkan, serangan sebelumnya membutuhkan usaha 8 kali lebih besar.

Berikut adalah kelebihan-kelebihan yang dimiliki serangan baru ini terhadap serangan yang lama :

1. Analisis yang efisien pada SBox atau tabel pencarian secara kebalikan dan operasi pada pembangunan *list* dari *data set* yang mungkin. Konsekuensinya, menggunakan LUT (*look up table*) jauh lebih cepat daripada mengkalkulasi data.
2. Serangan baru ini mengeksploitasi lebih banyak informasi / kelemahan daripada serangan sebelumnya.
3. Serangan ini mengatur pertukaran waktu / memori karena serangan ini mengumpulkan *list* dari semua data yang mungkin
4. Kompleksitas serangan ini jauh lebih kecil dibandingkan dengan serangan sebelumnya dalam mencari semua nilai masukan yang mungkin.

3. KESIMPULAN

Serangan baru ini memiliki kompleksitas yang jauh lebih rendah daripada serangan terhadap CAVE terdahulu sehingga lebih berbahaya. Serangan ini dapat mengancam keamanan dari implementasi CAVE yang sebenarnya. Otentikasi pelanggan yang sah adalah aplikasi utama dari CAVE. Jika nilai masukan berbeda yang di-*hash* ditemukan, sangat mungkin untuk memprogram ESN dan MIN ke dalam ponsel sehingga memungkinkan pelanggan yang curang mengakses jaringan nirkabel (*wireless network*). Ketika otentikasi gagal, panggilan pelanggan ke jaringan tidak dapat dilindungi bahkan dengan enkripsi suara.

Keputusan untuk mengganti CAVE dengan *Authenticated Key Agreement* (AKA) dibuat pada 1999. Proses standardisasi yang lama, ditambah pula adopsi yang lambat oleh operator membuat penggantian CAVE terlambat. Mengingat meningkatnya ancaman terhadap CAVE, maka penggantian algoritma CAVE dengan AKA harus lebih cepat dilakukan.

DAFTAR PUSTAKA

[1] Daemen, Joan, Vincent Rijmen. (2004). The *Rijndael* Specification. *http://csrc.nist.gov/encryption/AES/Rijndae*

l/Rijndael.pdf. Tanggal akses: 4 Desember 2004 pukul 20:00.

[2] Lidl & Niederreiter. (1986). Introduction to Finite Fields and Their Applications. Cambridge University Press.

[3] Munir, Rinaldi. (2004). Bahan Kuliah IF5054 Kriptografi. Departemen Teknik Informatika, Institut Teknologi Bandung.

[4] *NIST*. (2004). National Institute of Standards and Technology. *http://www.nist.gov*. Tanggal akses: 4 Desember 2004 pukul 20:00.

[5] Schneier, Bruce. (1996). Applied Cryptography 2nd. John Wiley & Sons.

[6] The Square Page. (2004), *http://www.esat.kuleuven.ac.be/~rijmen/square*. Tanggal akses: 4 Desember 2004 pukul 20:00.

[7] Tanenbaum, Andrew S. (2003). Computer Networks Fourth Edition. Pearson Education International.

[8] Trustcopy. (2004). Trustcopy - The premier provider of Brand Protection and Secured Trade Documentation Solutions. *http://www.trustcopy.com/*. Tanggal akses: 4 Desember 2004 pukul 20:00.