Analisis Serangan berbasis BDD pada Enkripsi E0 Bluetooth

Arief Widhiyasa (13505126)
Jurusan Teknik Informatika ITB, Bandung, email: arief@clawford.net

Abstract – *Makalah ini akan membahas cryptanalisis pada protokol bluetooth, yang menggunakan stream chiper untuk pengamanan data transmisi dan dikenal dengan nama E0. Pada dasarnya E0 akan membuat rentetan bilangan pseudorandom kemudian menggabungkannya dengan data menggunakan XOR operator. Panjang kunci bisa bervariasi, namun biasanya 128 bit. Sampai saat ini, algoritma stream chiper E0 ini masih digunakan pada protokol bluetooth, namun sampai saat ini pula, telah terdapat cukup banyak serangan pada enkripsi ini, antara lain : metoda guess & determine, algebraic attack, dan correlation attack. Sehingga keamanan transfer data melalui protokol bluetooth sulit dipastikan. Dalam makalah ini saya akan memfokuskan membahas salah satu serangan yang menggunakan Binary Decision Diagram (BDD), dan merupakan pengembangan dari Kraus attack [KA02] (termasuk pada algebraic attack) .*

Kata Kunci*: bluetooth encryption, E0, BDD.*

1. PENDAHULUAN

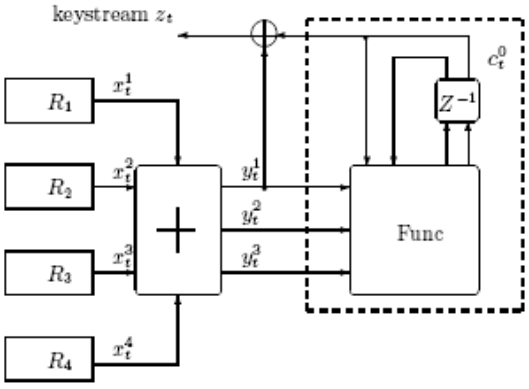
Bluetooth merupakan teknology yang digunakan untuk telekomunikasi jarak pendek yang cepat. Kemudahan penggunaannya telah menyebabkan bluetooth berkembang pesat, terutama pada mobile devices seperti handphone, PDA, notebook, kamera digital dsb.

Bluetooth menggunakan gelombang sebagai media utama komunikasinya, namun penggunaan gelombang ini menyebabkan pihap ketiga dapat melakukan penyadapan data yang dikirim. Sehingga keamanan telekomunikasi data menjadi salah satu isu penting. Salah satu teknik pengamanan yang dilakukan adalah melakukan enkripsi pada data sebelum dikirim kemudian didekripsi kembali oleh si penerima sehingga walaupun pihak ketiga mampu menyadap data tersebut dia tidak akan memahaminya.

Saat ini, bluetooth mengimplementasikan stream chiper untuk mekanisme enkripsi data sebelum dikirimkan. Chiper ini biasa disebut E0. Stream chiper ini, berbasis pada 4 LFSR's (Linear Feedback Shift Registers) dengan panjang yang berbeda-beda dan juga menggunakan 'non-linear combiner magic' (finite state machine). Keystream ini kemudian akan di XOR kan dengan plaintext untuk mendapatkan chipertextnya, dan untuk proses deskripsinya, akan dilakukan proses yang persis sama menggunakan keystream yang sama seperti yang digunakan pada saat enkripsi.

2. DESKRIPSI DASAR

2.1. E0 Keystream Generator



Gambar 1. Outline E0

Seperti yang dijelaskan pada bagian awal, keystream generator pada E0 yang digunakan bluetooth merupakan kombinasi dari 4 buah rangkaian bit memory yang dapat dirumuskan sebagai $\sigma_t = (c_{t-1}, c_t)$ pada waktu t, dimana $c_t = (c_t^1, c_t^0)$. Keseluruhan sistem menggunakan 4 LFSRs disimbolkan dengan $R_1 - R_4$, dengan panjang $L_1 = 25$, $L_2 = 31$, $L_3 = 33$, $L_4 = 39$ dan primitive feedback polynomial :

$$p_1(x) = x^{25} + x^{20} + x^{12} + x^8 + 1,$$
$$p_2(x) = x^{31} + x^{24} + x^{16} + x^{12} + 1,$$
$$p_3(x) = x^{33} + x^{28} + x^{24} + x^4 + 1,$$
$$p_4(x) = x^{39} + x^{36} + x^{28} + x^4 + 1,$$

Pada suatu clock cycle t, keempat bit output dari LFSR x^i_t, i = 1..4 akan dijumlahkan selayaknya sebuah integer. Jumlah dari sum tersebut, $y_t \in \{0,...,4\}$ dan direpresentasikan dengan biner. Misalkan y_t^i menyatakan elemen bit ke-i (i = 1,2,3). Sebuah 16-state machine (kotak yang bergaris putus-putus di gambar 1) akan mengeluarkan satu bit c_t^0 keluar dari state $\sigma_t = (c_{t-1}, c_t)$ dan mengambil input y_t untuk mengupdate σ_t menjadi σ_{t+1}. Terakhir akan didapatkan keystream z_t yang didapatkan denganc ara melakukan operasi bit XOR pada y_t^1 dengan c_t^0, yang bisa dijabarkan dengan :

$$x_t^1 \oplus x_t^2 \oplus x_t^3 \oplus x_t^4 \oplus c_t^0 = z_t.$$

Mekanisme detail dari Finite State Machine diluar lingkup bahasan kita, kecuali suatu kenyataan bahwa embeded delay cell (kotak berlabel Z^{-1} pada gambar 1) membuat c_t^0 tergantung hanya pada state inisial dari σ_0 dan juga vector-vector sebelumnya $y_{t-1}, y_{t-2},...,y_0$. Sehingga secara umum dapat dikatakan : diberikan y_t

bersama state σ_t, Finite State Machine akan maju menuju state σ_{t+1}. Tabel 1 menunjukkan transisi state Finite State Machine yang direpresentasikan dalam bilangan berbasis 4 (misalnya FSM akan mengubah $\sigma_t = 13_{(4)}$ menjadi $\sigma_{t+1} = 32_{(4)}$ dengan input $y_t = 2_{(4)}$).

		σ_t															
		00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33
y_t	0	00	11	23	32	03	12	20	31	01	10	22	33	02	13	21	30
	1	00	10	23	31	03	13	20	32	01	11	22	30	02	12	21	33
	2	01	10	20	31	02	13	23	32	00	11	21	30	03	12	22	33
	3	01	13	20	30	02	10	23	33	00	12	21	31	03	11	22	32
	4	02	13	21	30	01	10	22	33	03	12	20	31	00	11	23	32

Tabel 1. Transisi State FSM

Secara lebih formal, dengan menggunakan dua buah bit temporary $s_{t+1} = s_{t+1}^1, s_{t+1}^0$) pada setiap satuan waktu, rangkaian ekspresi di bawah ini akan dilakukan untuk mengupdate c_t

$$s_{t+1} = \left\lfloor \frac{y_t + 2 \cdot c_t^1 + c_t^0}{2} \right\rfloor,$$
$$c_{t+1}^1 = s_{t+1}^1 \oplus c_t^1 \oplus c_{t-1}^0,$$
$$c_{t+1}^0 = s_{t+1}^0 \oplus c_t^0 \oplus c_{t-1}^1 \oplus c_{t-1}^0.$$

Beberapa tahun terakhir ini, telah muncul varian E0 yang akan melakukan proses enkripsi sebanyak 2 kali yang biasa disebut dengan two-level E0, sehingga stream chiper diatas disebut dengan one-level E0.

2.2. Binary Decision Diagram

Binary Decision Diagram (BDD) merupakan sebuah struktur data yang digunakan untuk merepresentasikan fungsi-fungsi boolean. Misalkan X_n adalah kumpulan variabel boolean sebanyak n (x_0,...,x_{n-1}) dari suatu fungsi boolean. Sebuah BDD P pada Xn adalah graph berarah, berakar dan tidak memiliki cycle (dapat dikatakan pohon biner), dimana seiap node yang bukan daun akan diberi label dengan sebuah query (x_i?) dan memiliki outdegree sebesar 2, satu sisi berlabel 0 dan satu sisinya berlabel 1. Sehingga akan ada 2 jenis node daun, yaitu 0-sink dan 1-sink. Node akar dianggap sebagai node sumber atau node utama. Setiap assignment w($x_0 = w_0$, $x_1 = w_1$, ..., $x_{n-1} = w_{n-1}$) dimana $w_i \in \{0,1\}$ menunjukkan sebuah path unik pada P, yang dimulai dari node sumber menjawab w_i pada query (x_i?) yang pada akhirnya selalu akan menuju pada sink (node daun) yg unik. Node akhir merupakanhasil dari fungsi boolean yang dilakukan oleh w. Dua buah BDD dikatakan ekuivalen apabila mereka menghitung dua jenis fungsi boolean yang sama.

Sebuah BDD dikatakan Free Binary Decision Diagram (FBDD) jika dalam setiap path pada BDD, setiap variabel muncul sekali.

Sebuah BDD dikatakan Ordered Binary Decision Diagram (OBDD) jika setiap variabel akan terurut berdasarkan $x_0<x_1<...<x_{n-1}$ (dimana FBDD memperbolehkan urutan yang berbeda pada setiap path).

2.3. Cryptanalisis berbasis BDD pada E0

Framework serangan normal Krause [KA02] bekerja seperti berikut : diberikan beberapa bit keystream yang diketahui, kita ingin menghitung nilai inisial dari LFSR's. Misalkan L(x) menyatakan stream bit linear internal dari E0 keystream generator. L(x) dapat ditunjukkan dari rangkaian output dari 4 pararel LFSR's pada E0 (contoh, pada E0 berkunci 128 bit, L(x) terdiri dari 512 bit). Misalkan C(x) menyatakan komponen non-linear dari E0, C(x) merupakan unit kompresor yang termasuk hasil dari xor operation yang digunakan untuk mengambil keystreamnya. Berdasarkan deklarasi ini, K_{chiper} = C(L(x)), dimanax adalah nilai awal rahasia dari LFRS's.

Observasi Krause adalah mencari suatu kunci rahasia x dimana Kchiper = C(L(x)) untuk keystream Kchiper yang diberikan, ekuivalen dengan masalah mencari FBDD P yang minimal untuk keputusan apakah x memenuhi Kchiper = C(L(x)). Ide ini merupakan dasar dari serangan berbasis BDD terhadap system enkripsi E0.

Algoritmanya : Misalkan L(x), C(x) dan Kchiper seperti diatas. Maka dengan panjang kunci n = 128 :

1. Untuk stiap m>= 1 misalkan Q_m menyatakan FBDD minimal yang ditentukan dengan z elemen himpunan $\{0,1\}_m$ dimana C(z) adalah prefix dari K_{cipher}. Dengan lain perkataan Q_m adalah FBDD yang dibangun berdasarkan nilai dari bit keystream yang diketahui(K_{cipher}). FBDD ini menerima prefix dari bitstream internal yang digenerate dengan tiap LFSR sebagai input. Jika bitstream internal ini mengenerate sebuah prefix dari bit keystream yang telah diketahui FBDD akan menerimanya. Sebaliknya FBDD akan menolak input tersebut.
2. Untuk setiap m >= n misalkan S_m menyatakan FBDD minimal yang ditentukan dengan z =(z_0, z_1, .., z_m) dimana z_m = L (z_0, z_1, ..., z_{n-1}). Dengan lain perkataan S_m adalah FBDD yang dibangun berdasarkan polinom feedback dari LFSR. FBDD ini menerima initial value dari LFSR sebagai input. Jika initial value men-generate nilai z_m yang benar FBDD akan menerimanya. Sebaliknya FBDD akan menolak input tersebut.
3. Buatlah himpunan FBDD ketiga yang dinyatakan dengan Pm, dimana FBDD minimal yang menentukan apakah z himpunan {0,1}m adalah bitstream linear yang di-generate melalui L dan jika C(z) adalah prefix dari Kcipher. Perhatikan bahwa Pm sebenarnya adalah hasil dari perpotongan antara Qm dan Sm : Pm = SYNTH(Qm, Sm) dimana SYNTH menyatakan operasi sintesis BDD.

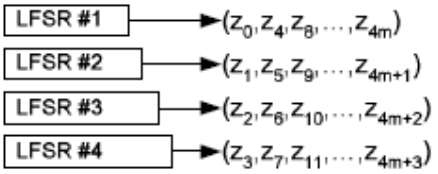
Strategi dari algoritma Krause adalah sebagai berikut, ia secara bertahap akan menghitung P_m untuk meningkatkan nilai m sampai hanya satu assignment yang diterima P_m. Assignment inilah yang merupakan initial value dari LFSR's generating K_{chiper}.

3. SERANGAN PADA E0

Pada bab ini saya akan membahas suatu algoritma serangan yang menggunakan basis BDD dan merupakan pengembangan dari algoritma Krause.

3.1. Pengurangan Algoritma

Algoritma yang dideskripsikan oleh Krause merupakan algoritma yang umum, dan perlu diadaptasikan untuk penggunaan pada E0. Dengan demikian diperlukan adanya penyederhanaan dan perubahan sebelum algoritma diimplementasikan :



Gambar 2. Metoda indexing dalam algoritma

1. Kunci observasinya adalah dengan secara rutin memeriksa E0. Setiap satu stuan waktu, satu bit dari setiap LFSR diinputkan ke compressor, dan setiap LFSR dimajukan sekali. Proses ini memberikan dua keuntungan penting. Pertama, E0 memiliki susunan natural pada bit internal Z. Pada serangan ini, pengurutan variabel yang digunakan adalah : (z0, z1, z2, ..., zj, ..., z511): dimana untuk j = 4*m+Li-1 dengan m adalah index waktu (0 <= m <= 127), dan Li adalah index dari LFSR (1 <= Li <= 4). Gambar 2 diatas mendeskripsikan metoda indexing yang digunakan pada implementasi algoritma serangan. Keuntungan yang kedua, kita bisa mengganti penggunaan FBDD menjadi OBDD. Hal ini memiliki perubahan implementasi yang kritikal, karena struktur data untuk mendukung OBDD jauh lebih mudah diimplementasikan dan lebih efisien daripada FBDD.

2. Algoritma serangan ini perlu menambahkan kenyataan bahwa LFSR pada E0 memiliki panjang yang berbeda-beda. Hal ini membuat perubahan pada detail implementasi dan perubahan pada analisis kompleksitas.

3. Seperti yang disebutkan saat penjelasan serangan Krauser [KA02], kita harus mengimplementasikan suatu operasi sintesis antara 2 buah BDD. Implementasi pada algoritma ini menggunakan algoritma dari Wegener. Namun karena semua BDD pada algoritma ini adalah OBDD yang tidak satupun memiliki loop dan semua BDD yang ada sangat kecil ukurannya, sehingga penggunaan hash table pada algoritma ini bisa diabaikan. Modifikasi ini membuat algoritma ini hanya spesifik bisa dilakukan untuk E0, namun akan meningkatkan perfomansi algoritma sangat besar.

3.2. Membangun OBDD untuk LFSR

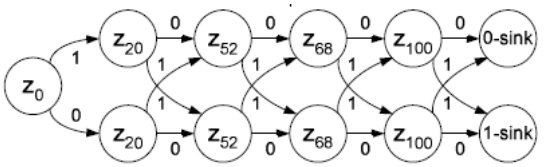
Seperti yang disebutkan pada pembahasan di bab 2, menyatakan bahwa BDD yang menghitung apakah bit internal z, konsisten dengan prefix $\{z_j\}_{j=1}^{l-1}$. Sejak setiap bit internal diproduksi oleh salah satu dari LFSR's, konsistensinya bergantung pada 4 bit lebih awal pada LFSR's tersebut. Contohnya untuk LFSR's terpendek, setiap bit harus mengikuti feedback polynomial : $t^{25} + t^{20} + t^{12} + t^8 + t^0$; yang berarti bit z_i sama dengan :

$$z_i = z_{i-8} \oplus z_{i-12} \oplus z_{i-20} \oplus z_{i-25}$$

Menggunakan bit ordering (lihat gambar 2) sehingga persamaan tesebut menjadi :

$$z_i = z_{i-32} \oplus z_{i-48} \oplus z_{i-80} \oplus z_{i-100}$$

Tabel 2 merangkum persamaan konsistensi dasar dan persamaan konsistensi yang telah dinormalisasi untuk keempat LFSR's. Perlu dicatat bahwa LFSR, memproduksi bit dengan index j sedemikian sehingga j ekuivalen dengan (i-1) mod 4.



Gambar 3. Contoh OBDD konsistensi LFSR

Sebuah OBDD yang merepresentasikan kondisi konsistensi LFSR mengandung 5 variabel dan 11 nodes (termasuk 0-sink dan 1-sink). Gambar 3 menunjukkan sebuah OBDD yang mana melakukan pemeriksaan kondisi konsistensi untuk bit nomor 100. Perlu diingat bahwa untuk setiap LFSR akan dibuat OBDD yang berbeda; hal ini karena setiap LFSR memiliki panjang sendiri-sendiri dan menghasilkan bit-bit yang berbeda untuk non-native bitnya. Jumlah non native bits yang diproduksi pada setiap LFSR sama dengan panjang kunci dikurangi ukuran LFSR. Sehingga jumlah total OBDD yang merepresentasikan kondisi konsistensi LFSR adalah 4n – 128 yang hasilnya 384.

LFSR #	Basic consistency equation	Normalized consistency equation
1	$z_i = z_{i-8} \oplus z_{i-12} \oplus z_{i-20} \oplus z_{i-25}$	$z_i = z_{i-32} \oplus z_{i-48} \oplus z_{i-80} \oplus z_{i-100}$
2	$z_i = z_{i-12} \oplus z_{i-16} \oplus z_{i-24} \oplus z_{i-31}$	$z_i = z_{i-48} \oplus z_{i-64} \oplus z_{i-96} \oplus z_{i-124}$
3	$z_i = z_{i-4} \oplus z_{i-24} \oplus z_{i-28} \oplus z_{i-33}$	$z_i = z_{i-16} \oplus z_{i-96} \oplus z_{i-112} \oplus z_{i-132}$
4	$z_i = z_{i-4} \oplus z_{i-28} \oplus z_{i-36} \oplus z_{i-39}$	$z_i = z_{i-16} \oplus z_{i-112} \oplus z_{i-144} \oplus z_{i-156}$

Tabel 2. Persamaan Konsistensi LFSR's

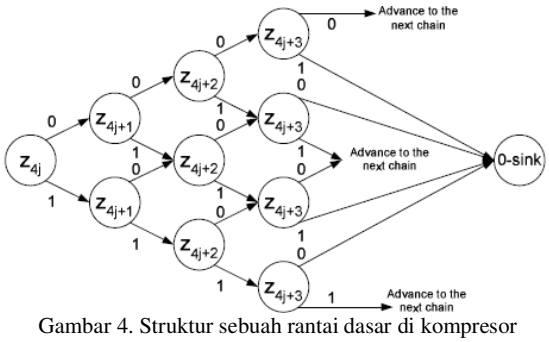
3.3. Membangun OBDD Compressor

OBDD yang merepresentasikan komponen non-linear dari E0 merupakan unit compressor. OBDD ini dibangun berdasarkan bit keystream yang diketahui, dan berdasarkan fungsi transisi dari kompresor (dapat dilihat pada tabel 3).

Current State	Input									
	0		1		2		3		4	
	Out	NS	Out	NS	Out	NS	Out	NS	Out	NS
0	0	0	1	0	0	4	1	4	0	8
1	0	12	1	12	0	8	1	8	0	4
2	0	4	1	4	0	0	1	0	0	12
3	0	8	1	8	0	12	1	12	0	0
4	1	5	0	1	1	1	0	13	1	13
5	1	9	0	13	1	13	0	1	1	1
6	1	1	0	5	1	5	0	9	1	9
7	1	13	0	9	1	9	0	5	1	5
8	0	14	1	14	0	2	1	2	0	6
9	0	2	1	2	0	14	1	14	0	10
10	0	10	1	10	0	6	1	6	0	2
11	0	6	1	6	0	10	1	10	0	14
12	1	11	0	7	1	7	0	3	1	3
13	1	7	0	11	1	11	0	15	1	15
14	1	15	0	3	1	3	0	7	1	7
15	1	3	0	15	1	15	0	11	1	11

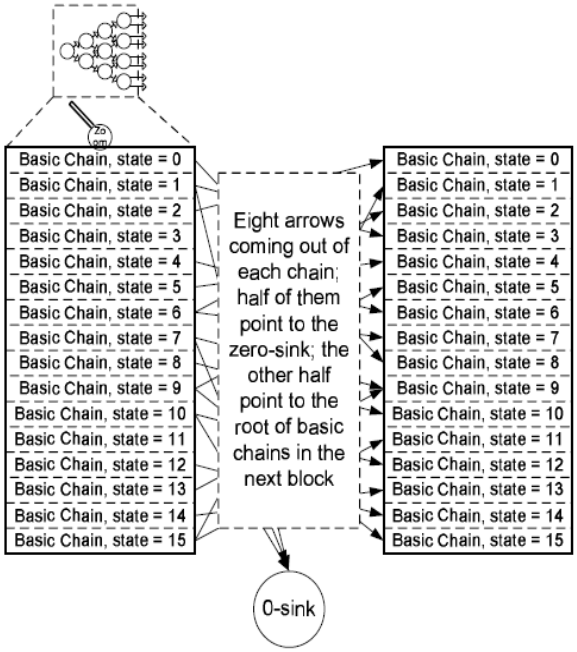
Tabel 3. Fungsi transisi FSM (kompresor)

Seperti yang telah dinyatakan sebelumnya, compressor mengupdate nilainya berdasarkan penjumlahan dari bit output LFSR. Oleh karena itu, dibutuhkan struktur BDD untuk merepresentasikan penjumlahan 4 bit. Struktur semacam itu disebut "basic chain". Untuk status dan masing-masing dari 5 kemungkinan penjumlahan, **Tabel 3** menyatakan bit output yang seharusnya. Jika sesuai dengan bit yang diberikan pada keystream yang diketahui, dapat dilanjutkan ke "chain" berikutnya, dan mengecek empat bit selanjutnya. Sebaliknya, path ini akan menuju 0-sink. Gambar 4 menunjukkan struktur dari rantai dasar.



Gambar 4. Struktur sebuah rantai dasar di kompresor

Compressor BDD dibangun dari blok-blok, masing-masing terdiri dari 16 "basic chain" (satu untuk setiap status yang mungkin pada compressor). Setengah dari path dari setiap blok menuju 0-sink, sementara setengah lainnya menuju status yang tepat pada blok berikutnya. **Gambar 5** mengilustrasikan struktur penuh dari OBDD yang merepresentasikan compressor.



Gambar 5. Dua blok konsekutif OBDD yang merepresentasikan kompresor

Sebuah blok compressor terdiri dari 160 simpul dan menggunakan 4 bit. Perhatikan bahwa masing-masing dari 4 bit menyumbang banyak simpul yang berbeda-beda pada blok. Selanjutnya, menggunakan sederetan blok dapat menghasilkan sebuah BDD non-minimal yang dapat direduksi. Sebagai contoh, untuk 128 blok, kompresor BDD tereduksi terdiri dari sekitar 14,500 simpul bukan 20,480.

4. ANALISIS (HASIL dan PEMBAHASAN)

4.1. Analisis dasar

Dapat dilihat dari penjelasan pada bab 2, bahwa E0 sebenarnya bukanlah algoritma enkripsi yang cukup baik. Mungkin dikarenakan tempat algoritma ini diimplementasikan merupakan telekomunikasi jarak dekat, jadi kemungkinan dilakukannya serangan sangat kecil, sehingga algoritma enkripsi yang digunakan masih cukup simple. Kelemahan terutama dari E0 adalah pola yang dimiliki sehingga penggunaan kunci sepanjang 128 bit pun efeknya tidak akan begitu terasa.

4.2. Analisis kompleksitas algoritma serangan

Kompleksitas waktu dari algoritma pada bab sebelumnya ditentukan berdasarkan kompleksitas ruang dari "synthesized OBDD" melalui kesulurahan sintesis proses. Pada setiap tingkat dalam proses, ukuran dari "synthesized OBDD" dibatasi dengan 2 batasan :

1. Banyaknya assignment yang memenuhi OBDD membatasi ukuran dari OBDD minimal direpresentasikan oleh fungsi boolean :

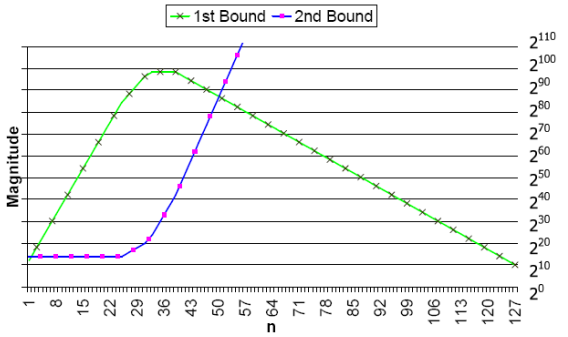
   $$|P| <= m \,.\, |One(P)| \qquad (1)$$

   dimana *One*(P) menyatakan himpunan assignment yang memenuhi dari BDD P, dan m adalah banyaknya variabel yang dimiliki BDD.
2. Setiap operasi sintesis membatasi ukuran dari hasil sintesis. Pada umumnya, batasannya adalah |SYNTH(P,Q)| <= |P|. |Q|. Namun, saat P adalah pemeriksa konsistensi LFSR pada OBDD, dapat digunakan batasan yang lebih rapat. Hal ini disebabkan karena struktur dari OBDD yang merepresentasikan LFSR diperiksa secara konsisten. OBDD ini secara efektif menyimpan sebuah bit paritas untuk mengingat jika konsistensi tercapai pada setiap poin. Hal inilah yang mendasari mengapa setiap variabel muncul sebanyak dua kali pada OBDD konsistensi LFSR. setiap simpul dalam "window" paritas antara variabel terendah dan tertinggi dalam OBDD konsistensi LFSR mengalami duplikasi, karena itu OBDD yang dihasilkan haruslah sebesar-besarnya dua kali ukuran dari ukuran OBDD terbesar. Batas tersebut dapat dijumlahkan dalam :

   $$P <= |Q(m)|.2^{(m-n)} \qquad (2)$$

   dimana |Q(m)| adalah ukuran OBDD yang mrepresentasikan compressor, m adalah banyaknya variabel (m : 4 - 512) dan n adalah jumlah keystream yang diberikan (n : 1 - 128 bit). Perhatikan bahwa batas ini masih kurang tepat (renggang) karena hanya simpul di dalam "window" paritas yang terduplikasi, sementara batasan ini mengasumsikan bahwa semua simpul OBDD terduplikasi.

Batasan pada ukuran OBDD selama proses adalah "lower envelope" dari batasan persamaan (1) dan persamaan (2). Gambar 6 menunjukkan kedua batasan tersebut.



Gambar 6. Grafik kedua batasan

Dengan menggunakan persamaan (1), diperoleh bahwa selama langkah pertama, pada setiap satuan waktu memperkenalkan 4 variabel baru, dan 1 "constraint" karena bit output diketahui. Hal ini berarti banyaknya assignment yang memenuhi dikalikan dengan 2^3 setiap satuan waktu. Setiap melewati 25 satuan waktu, seluruh native bit dari LFSR #1 akan ditentukan secara keseluruhan, sehingga banyaknya asumsi yang memenuhi bertambah sebanyak faktor dari 2^2 setiap satuan waktu. Saat native bit dari semua empat LFSR telah diset berdasarkan kondisi konsistensi dari LFSR (yaitu saat n >= 39), banyaknya assignment yang memenuhi mulai berkurang sebanyak setengah dari masing-masing satuan waktu. Batasan berdasarkan banyaknya assignment yang memenuhi untuk n >= 39 adalah $|P| <= m.2^{(128-n)}$.

Dengan menggunakan persamaan (4), kita mendapatkan bahwa selama kita tidak memulai proses synthesizing dengan OBDD konsistensi LFSR (n <= 25), maka ukuran OBDD maksimal hanya mencapai ukuran OBDD yang merepresentasikan kompresor (C_OBDD). Ketika kita memulai proses synthesis, OBDD akan mulai bertambah dengan faktor 2 untuk setiap kali operasi synthesis. Perlu dicatat bahwa jumlah operasi synthesis untuk setiap satuan waktu bergantung dari nilai n. Batasan berdasarkan hasil synthesis untuk nilai n >= 39 adalah $|P| <= |C\_OBDD| \,.\, 2^{4n-128}$ (ukuran dari C_OBDD adalah 2^{14}).

Menghitung nilai interseksi dari kedua batasan, kita akan mendapat ukuran maksimal dari OBDD yang dishyntesis melalui proses adalah |P| sekitar 2^{86}. Nilai maksimal ini tercapai pada nilai satuan waktu n = 50. Ini memberi nilai total kompleksitas waktu berupa O (2^{90}), karena kita harus melakukan algoritma ini dengan nilai yang berbeda untuk 4 bit initial dari state machine. Perlu diketahui bahwa nilai ini cukup besar bila dibandingkan dengan estimasi waktu serangan Krause [KA02]. Namun nilai ini masih merupakan nilai batasan atas. Nilai sebenarnya masih lebih rendah dari nilai ini. Untuk itu, bila dilakukan simulasi untuk menghitungnya, yang pada akhirnya menghasilkan nilai maksimal untuk OBDD dalam proses shyntesis adalah |P| sekitar $2^{82.5}$. Nilai ini mengubah nilai total kompeksitas menjadi O ($2^{86.5}$).

4.3. Analisis Heuristic

Pada algoritma serangan pada bab sebelumnya, telah diketahui kompleksitas totalnya sekitar O ($2^{86.5}$). Kompleksitas ini sudah jauh lebih baik dibandingkan dengan melakukan bruteforce search yang pastinya akan berkompleksitas O (2^{128}). Namun masih dapat dilakukan beberapa pendekatan heuristic untuk memperkecil kompleksitas algoritma.

Pendekatan terbaik untuk heuristic yang dilakukan adalah dengan menebak beberapa bit nilai pada bit initial LFSR. Pendekatan ini akan memberikan beberapa keuntungan, yang pertama, ruang kompleksitas akan menurun karena ukuran OBDD yang merepresentasikan compresor menurun dan yang lebih penting, jumlah dari OBDD yang harus diberlakukan proses synthesis menurun jauh. Pendekatan ini juga merupakan pendekatan terbaik

ketika ingin melakukan serangan pararell karena algoritma bisa dijalankan dengan menggunakan nilai tebakan yang berbeda-beda untuk setiap mesin yang melakukan penyerangan, sehingga penyerangan bisa diselesaikan lebih cepat.

Saat dites pada komputer (pentium IV dengan RAM 1 GB dan OS Windows XP), algoritma serangan tersebut hanya bisa dilakukan dengan menduga semua 56 bit dari LFSR #1 dan #2, ditambah beberapa bit dari LFSR #3 dan #4. Hasil terbaik didapatkan ketika semua bit pada LFSR #1 dan #2 ditebak ditambah 4 bit dari LFSR #3 dan #4 (masing-masing 2 bit). Dalam kasus ini ukuran maksimal OBDD yang disynthesis sekitar 2^{23} node. Karena algoritma telah menebak sebesar 60 bit, dan algoritma harus dilakukan sebanyak initial state dari kompresor, total kompleksitasnya menjadi O (2^{87}). Tabel 4 akan menunjukkan perbandingan kompleksitas ketika diujicobakan pada jumlah tebakan pada LFSR # 3 dan #4 yang berbeda-beda.

Total number of guessed bits in LFSR's #3+#4	Maximal OBDD size (# nodes)	Total time complexity
12	$2^{18.3}$	$2^{90.3}$
10	$2^{18.7}$	$2^{88.7}$
8	$2^{19.9}$	$2^{87.9}$
6	$2^{21.7}$	$2^{87.7}$
4	$2^{23.4}$	$2^{87.4}$

Tabel 4. Perbandingan nilai kompleksitas berdasarkan perbedaan jumlah bit yang ditebak.

5. KESIMPULAN

- Teknologi bluetooth yang menggunakan system E0 sebagai chiper utamanya cukup rawan akan serangan kriptanalisis akibat keberadaan pola pada E0 itu sendiri, sehingga panjang kunci 128-bit pun tidak cukup berguna untuk melindunginya.
- Serangan terhadap chiper E0 yang menggunakan metoda BDD dan merupakan pengembangan dari serangan Krause serta memanfaatkan pola-pola yang ada pada chiper E0 sangat efektif untuk melakukan penyerangan terhadap hasil enkripsi dari E0 dengan kompleksitas yang cukup singkat, yaitu O(2^{90}).
- Terdapat pendekatan heuristic yang dapat membuat penyerangan terjadi secara pararel dengan menggunakan metoda guessing (pendugaan) pada beberapa bit nilai LFSR, yang pada penggunaan terbaik akhirnya menyebabkan kompleksitas total turun menjadi O($2.^{86.5}$)

6. REFERENSI

- Munir, Ir. Rinaldi, M.T. *Diktat Kuliah IF5054 Cryptography*. Teknik Informatika ITB, 2006
- Krause Matthias. BDD-based cryptanalysis of keystream generators. In L. Knudsen, editor, *Advances in Cryptology*. Springer-Verlag, 2002.
- http://lasecwww.epfl.ch/pub/lasec/doc/YV04a.pdf
- http://www.iris.re.kr/ac04/data/Asiacrypt2004/11%20Symmetric%20Key%20Cryptanalysis/04_Yi%20Lu.pdf
- http://www.cs.unb.ca/~gdueck/courses/cs4835/bdd97.pdf
- http://www.itu.dk/people/hra/bdd97.ps
- http://eprint.iacr.org/2006/072.pdf
- http://www.tml.tkk.fi/Nordsec2004/Presentations/ritvanen.pdf
- http://www.iacr.org/conferences/crypto2005/p/16.pdf