# Studi dan Analisis Mengenai Algoritma *Cipher* Aliran "Rabbit"

**Paramita** [1)]

1) Program Studi Teknik Informatika STEI ITB, Bandung, email: if14040@students.if.itb.ac.id

***Abstraksi*** *– Rabbit merupakan salah satu algoritma cipher aliran yang diperkenalkan pada tahun 2003. Algoritma Rabbit menggunakan 128 bit kunci rahasia dan 64 bit Initialization Vector (IV) sebagai masukan untuk membangkitkan blok keluaran yang terdiri dari 128 bit acak semu (pseudo-random,, yang merupakan kombinasi dari bit-bit pada status internal, untuk setiap iterasi. Di dalam makalah ini akan dibahas secara tuntas mengenai algoritma Rabbit. Pembahasan akan mencakup desain algoritma, analisis keamanan algoritma, dan performansi algoritma Rabbit.*

***Kata Kunci****: kriptografi, algoritma cipher aliran, Rabbit.*

## 1. PENDAHULUAN

Di dalam kriptografi, sebuah metode disebut kriptografi simetri jika kunci yang sama digunakan baik dalam proses enkripsi maupun dekripsi sebuah pesan. Salah satu algoritma kriptografi simetri adalah cipher aliran (stream cipher). Cipher aliran merupakan algoritma kriptografi yang beroperasi pada plainteks/cipherteks dalam bentuk bit tunggal, dalam hal ini rangkaian bit dienkripsikan/didekripsikan bit per bit dengan menggunakan aliran bit kunci.

Rabbit merupakan salah satu algoritma cipher aliran yang diperkenalkan pada tahun 2003. Algoritma Rabbit menggunakan 128 bit kunci rahasia dan 64 bit Initialization Vector (*IV*) sebagai masukan untuk membangkitkan blok keluaran yang terdiri dari 128 bit acak semu (pseudo-random,, yang merupakan kombinasi dari bit-bit pada status internal, untuk setiap iterasi.

Proses enkripsi/dekripsi dilakukan dengan meng-XOR-kan blok acak semu tersebut dengan plainteks/cipherteks. Ukuran dari status internal adalah 513 bit dibagi menjadi 8 variabel status dengan panjang 32 bit, 8 *counter* dengan panjang 32 bit, dan 1 bit carry untuk *counter*. Kedelapan variabel status di-update dengan 8 buah fungsi non-linear.

Rabbit dirancang sehingga lebih cepat daripada algoritma kriptografi yang biasa digunakan, dan menggunakan kunci berukuran 128 bit untuk mengenkripsi plainteks dengan jumlah blok hingga $2^{64}$. Sehingga untuk mengetahui plainteks tanpa memiliki kunci hanya dapat dilakukan dengan pencarian *exhaustive*, yaitu dengan $2^{128}$ kemungkinan kunci.

## 2. RANCANGAN ALGORITMA RABBIT

### 2.1. Notasi yang Digunakan

$\oplus$ melambangkan operator logika XOR, $\ll$ dan $\gg$ melambangkan operator pergeseran bit kiri dan kanan (*left and right bit-wise shift*), $\lll$ dan $\ggg$ melambangkan operator rotasi bit kiri dan kanan (*left and right bit-wise rotation*), $\diamond$ melambangkan konkatenasi dua rangkaian bit.

$A^{[g..h]}$ berarti nilai dari index $g$ sampai dengan $h$ dari variabel $A$. Ketika melakukan penomoran bit dari variabel, *least significant bit* dilambangkan dengan 0. Bilangan heksadesimal diawali dengan prefiks "0x". Untuk semua veriabel dan konstan digunakan notasi integer.

### 2.2. Desain Algoritma Rabbit

Status internal terdiri dari 513 bit. 512 bit dibagi menjadi 8 variabel status $x_{j,i}$ dengan panjang 32 bit, dan 8 variabel *counter* $c_{j,i}$ dengan panjang 32 bit, dimana $x_{j,i}$ melambangkan variabel status dari subsistem $j$ pada iterasi $i$, dan $c_{j,i}$ melambangkan variabel *counter* yang bersesuaian.

Selain itu digunakan 1 bit carry untuk *counter*, $\phi_{7,i}$, yang disimpan untuk setiap iterasi. *Counter* bit ini diinisialisasi dengan nilai 0. Kedelapan variabel status dan kedelapan variabel *counter* diturunkan dari kunci pada saat inisialisasi.

#### 2.2.1. Skema *Key Setup*

Algoritma diinisiasi dengan memperluas kunci berukuran 128 bit menjadi 8 variabel status dan 8 variabel *counter* sehingga diperoleh korespondensi satu-ke-satu antara kunci dan variabel status awal, $x_{j,0}$, dan variabel *counter* awal, $c_{j,0}$.

Kunci awal, $K^{[127..0]}$, dibagi menjadi 8 subkunci, yaitu:

$$k_0 = K^{[15..0]}$$
$$k_1 = K^{[31..16]}$$
$$k_2 = K^{[47..32]}$$
$$k_3 = K^{[63..48]}$$
$$k_4 = K^{[79..64]}$$
$$k_5 = K^{[95..80]}$$
$$k_6 = K^{[111..96]}$$
$$k_7 = K^{[127.112]}$$

Variabel status dan variabel *counter* diinisialisasi dari subkunci dengan aturan sebagai berikut:

$$x_{j,0} = \begin{cases} k_{(j+1 \bmod 8)} \diamond k_j & jika\ j\ genap \\ k_{(j+5 \bmod 8)} \diamond k_{(j+4 \bmod 8)} & jika\ j\ ganjil \end{cases} \quad (1)$$

dan

$$c_{j,0} = \begin{cases} k_{(j+4 \bmod 8)} \diamond k_{(j+5 \bmod 8)} & jika\ j\ genap \\ k_j \diamond k_{(j+1 \bmod 8)} & jika\ j\ ganjil \end{cases} \quad (2)$$

Sistem ini diiterasi sebanyak 4 kali sampai diperoleh variabel status $x_{j,4}$ dan variabel *counter* $c_{j,4}$, dengan $j$ dari 0 sampai dengan 7 (sebanyak 8 variabel status dan 8 variabel *counter*).

Kemudian, variabel *counter* dimodifikasi dengan:

$$c_{j,4} = c_{j,4} \oplus x_{(j+4 \bmod 8),4} \quad (3)$$

untuk semua $j$, untuk mencegah kunci diketahui dengan dilakukan pembalikan pada variabel-variabel *counter*.

### 2.2.2. Skema *IV Setup*

Yang dilakukan pada skema ini adalah modifikasi variabel status yang diperoleh pada skema *key setup*, dengan fungsi dari *IV* (iInitialization vector). Hal ini dilakukan dengan meng-XOR-kan *IV* dengan panjang 64 bit, dengan variabel *counter* sepanjang 256 bit. Variabel *counter* dimodifikasi dengan aturan sebagai berikut:

$$\begin{aligned} c_{0,4} &= c_{0,4} \oplus IV^{[31..0]} \\ c_{1,4} &= c_{1,4} \oplus (IV^{[63..48]} \diamond IV^{[31..16]}) \\ c_{2,4} &= c_{2,4} \oplus IV^{[63..32]} \\ c_{3,4} &= c_{3,4} \oplus (IV^{[47..32]} \diamond IV^{[15..0]}) \\ c_{4,4} &= c_{4,4} \oplus IV^{[31..0]} \\ c_{5,4} &= c_{5,4} \oplus (IV^{[63..48]} \diamond IV^{[31..16]}) \\ c_{6,4} &= c_{6,4} \oplus IV^{[31..0]} \\ c_{7,4} &= c_{7,4} \oplus (IV^{[47..32]} \diamond IV^{[15..0]}) \end{aligned} \quad (4)$$

Skema ini diiterasi sebanyak 4 kali untuk membuat semua bit tidak bergantung secara linear terhadap semua bit pada *IV*. Modifikasi dari variabel *counter* dengan menggunakan *IV* akan menjamin bahwa semua kemungkinan kombinasi *IV* yang berbeda sebanyak $2^{64}$ kemungkinan akan membangkitkan *keystream* yang unik.

### 2.2.3. Fungsi *Next-State*

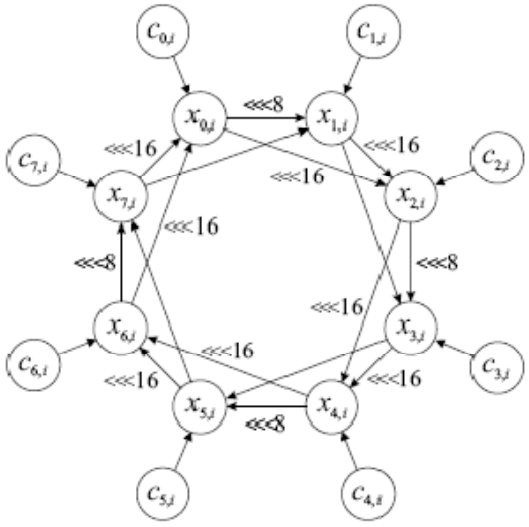
Fungsi utama dari algoritma Rabbit terletak pada persamaan berikut:

$$x_{j,i+1} = \begin{cases} g_{j,i} + (g_{j-1 \bmod 8,i} \lll 16) + (g_{j-2 \bmod 8,i} \lll 16) & jika\ j\ genap \\ g_{j,i} + (g_{j-1 \bmod 8,i} \lll 8) + (g_{j-2 \bmod 8,i}) & jika\ j\ ganjil \end{cases} \quad (5)$$

dengan:

$$g_{j,i} = \left( (x_{j,i} + c_{j,i})^2 \oplus \left( (x_{j,i} + c_{j,i})^2 \gg 32 \right) \right) mod\ 2^{32} \quad (6)$$

dimana semua penjumlahan adalah modulo $2^{32}$. Sistem ini diilustrasikan pada gambar 1.




Gambar 1. Ilustrasi fungsi *next-state*

### 2.2.4. Sistem *Counter*

Dinamika dari *counter* didefinisikan sebagai berikut:

$$c_{0,i+1} = \begin{cases} c_{0.i} + a_0 + \phi_{7,i} \bmod 2^{32} & jika\ j = 0 \\ c_{j.i} + a_j + \phi_{j-1,i+1} \bmod 2^{32} & jika\ j > 0 \end{cases} \quad (7)$$

dengan bit carry $\phi_{j,i+1}$ ditentukan dengan:

$$\phi_{j,i+1} = \begin{cases} 1 & jika\ c_{0.i} + a_0 + \phi_{7,i} \geq 2^{32} \wedge j = 0 \\ 1 & jika\ c_{j.i} + a_j + \phi_{j-1,i+1} \geq 2^{32} \wedge j > 0 \\ 0 & jika\ tidak \end{cases} \quad (8)$$

### 2.2.5. Skema Ekstraksi

Setelah setiap iterasi, 4 rangkaian bit (*word*) acak semu sepanjang 32 bit dibangkitkan dengan aturan sebagai berikut:

$$\begin{aligned} s_{j,i}^{[15..0]} &= x_{2j,i}^{[15..0]} \oplus x_{2j+3 \bmod 8,i}^{[31..16]} \\ s_{j,i}^{[31..16]} &= x_{2j,i}^{[31..16]} \oplus x_{2j+3 \bmod 8,i}^{[15..0]} \end{aligned} \quad (10)$$

dimana $s_{j,i}$ adalah rangkaian bit (*word*) ke $j$ pada iterasi ke $i$. Keempat rangkaian bit acak semu ini kemudian di-XOR-kan dengan plainteks/cipherteks untuk melakukan proses enkripsi/dekripsi.

## 3. ANALISIS KEAMANAN ALGORITMA

### 3.1. Analisis Rancangan Skema *Key Setup*

Secara umum, skema *key setup* dibagi menjadi 3 tahap, yaitu: perluasan kunci untuk memperoleh variabel status dan variabel *counter*, iterasi, dan modifikasi variabel *counter*.

#### 3.1.1. Perluasan Kunci

Dengan adanya perluasan kunci maka ada 2 akibat, yaitu:

- Adanya korespondesi satu-ke-satu antara kunci, variabel status, dan variabel *counter*, sehingga mencegah redundansi kunci.
- Setelah setiap iterasi pada fungsi *next-state*, setiap bit kunci mempengaruhi semua variabel status

#### 3.1.2. Iterasi Sistem

Skema perluasan kunci menjamin bahwa setelah 2 iterasi pada fungsi *next-state*, semua bit variabel status dipengaruhi oleh semua bit kunci dengan probabilitas 0,5. Untuk lebih amannya, maka sistem diiterasi sebanyak 4 kali.

#### 3.1.3. Modifikasi *Counter*

Walaupun variabel *counter* dapat diketahui oleh penyerang, modifikasi pada *counter* mengakibatkan kesulitan untuk memperoleh kunci yang dapat diperoleh dengan pembalikan pembentukan *counter* dari kunci. Akan tetapi, dengan adanya modifikasi *counter* tidak dapat dijamin bahwa setiap kunci yang berbeda akan menghasilkan variabel *counter* yang unik.

### 3.2. Analisis Rancangan Skema *IV Setup*

Secara umum, skema *IV Setup* dibagi menjadi 2 tahap, yaitu: penambahan dengan *IV* dan iterasi sistem.

#### 3.2.1. Penambahan *IV*

Penambahan *IV* akan memodifikasi variabel *counter* sedemikian rupa sehingga akan menjamin bahwa dengan menggunakan kunci yang sama, ada sebanyak $2^{64}$ kemungkinan *keystream* yang unik dari setiap *IV* yang berbeda.

#### 3.2.2. Iterasi Sistem

Iterasi sistem akan menjamin bahwa setelah setiap iterasi, setiap bit pada *IV* akan mempengaruhi kedelapan variabel status. Sistem diiterasi sebanyak 4 kali untuk menjadikan semua bit tidak bergantung secara linear terhadap bit-bit pada *IV*.

### 3.3. Analisis Kemanan Terhadap Serangan

Berikut ini adalah analisis keamanan algoritma Rabbit terhadap serangan-serangan kriptanalisis.

#### 3.3.1. Serangan *Divide-and-Conquer*

Serangan ini mungkin dilakukan dengan memprediksi bit-bit output dari sekumpulan kecil bit-bit status internal. Penyerang akan menebak bagian yang relevan dari status, memprediksi bit-bit output, dan membandingkannya dengan bit-bit output yang sebenarnya, dan memverifikasi apakah tebakannya benar.

Penyerang harus menebak setidaknya 2 . 12 byte input untuk fungsi $g$ untuk memverifikasi satu byte saja. Hal ini ekuivalen dengan menebak 192 bit dan oleh karena itu lebih sulit dibandingkan dengan pencarian *exhaustive* kunci. Kesimpulannya, tidak mungkin untuk menebak bit-bit output lebih sedikit daripada 128 bit.

#### 3.3.2. Serangan pada Fungsi *Key Setup*

Sesuai dengan yang disebutkan di atas, walaupun variabel *counter* dapat diketahui oleh penyerang, modifikasi pada *counter* mengakibatkan kesulitan untuk memperoleh kunci yang dapat diperoleh dengan pembalikan pembentukan *counter* dari kunci.

## 4. PERFORMANSI ALGORITMA

### 4.1. Pengukuran Performansi

Performansi diukur dengan menggunakan kode program Rabbit yang diperoleh dari sumber, dengan cara menghitung *clock tick* yang terlewat, dengan menggunakan instruksi RDTSC.

Berikut ini adalah kode program yang digunakan untuk mengukur performansi algoritma Rabbit.

```
unsigned __int64 read_clock_tick()
{
    unsigned int h,l;
    __asm
    {
        // Flush pipelines
        xor EAX, EAX
        cpuid

        // Read clock counter
        rdtsc
        mov h, EDX
        mov l, EAX
    }
    return ((unsigned __int64)h<<32) + l;
}
```

*Clock tick* dibaca sebelum dan sesudah pemanggilan fungsi enkripsi atau fungsi *setup*. Contoh penulisan dalam *pseudo-code* adalah sebagai berikut:

```
start = read_clock_tick();
end = read_clock_tick();
overhead = end - start;

start = read_clock_tick();
key_setup(...);
end = read_clock_tick();
key_setup_time = end - start - overhead;

start = read_clock_tick();
cipher(...);
end = read_clock_tick();
```

```
cipher_time = end - start - overhead;
cipher_time_pr_byte = cipher_time /
   data_size;
```

Performansi diukur dengan memanggil fungsi sebanyak 25 kali, dan yang digunakan sebagai hasil adalah hasil terbaik. Kecepatan enkripsi diperoleh dari hasil enkripsi data sebesar 16 KB yang disimpan pada memori.

### 4.2. Lingkungan Pengukuran Performansi

Evaluasi performansi dari Rabbit dilakukan pada laptop PC dengan spesifikasi:

- Prosesor Intel Core2Duo 1.83GHz
- RAM 512 MB
- Operating System Microsoft Windows XP Professional SP 2

### 4.3. Hasil Pengukuran Performansi

Berikut ini adalah hasil pengukuran performansi algoritma Rabbit.

| Fungsi | Ukuran Kode | Performansi |
|---|---|---|
| *Key Setup* | 608 byte | 446 cycle |
| *IV Setup* | 688 byte | 417 cycle |
| Enkripsi/Dekripsi | 762 byte | 4.7 cycle/byte |

## 5. KESIMPULAN

Dalam hal keamanan, algoritma Rabbit sangat baik untuk diimplementasikan, karena hingga saat ini belum ada serangan terhadap algoritma Rabbit yang lebih cepat daripada pencarian *exhaustive*. Sedangkan pencarian *exhaustive* terhadap algoritma Rabbit adalah pencarian terhadap $2^{128}$ kemungkinan kunci, karena panjang kunci yang digunakan adalah 128 bit.

Algoritma Rabbit dapat digunakan untuk mengenkripsi plainteks dengan ukuran hingga mencapai $2^{64}$ byte.

Dalam hal kecepatan, algoritma Rabbit dapat dikatakan lebih cepat daripada algoritma *cipher* aliran yang biasa digunakan.

## DAFTAR REFERENSI


[1] Munir, Rinaldi, *Kriptografi*, Institut Teknologi Bandung, 2006.

[2] Boesgaard, Martin, et al, *The Rabbit Stream Cipher – Design and Security Analysis*, Cryptico, 2003.