Studi Algoritma CSPRNG ISAAC dan Perbandingannya dengan Algoritma CSPRNG Lain

Puthut Prabancono – NIM : 13506068

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if16068@students.if.itb.ac.id

Abstrak

Cryptographically Secure Pseudorandom Number Generator (CSPRNG) adalah pembangkit bilangan acak yang dapat menghasilkan bilangan yang tidak mudah diprediksi pihak lawan. Pembangkit tersebut cocok untuk kriptografi misalnya digunakan untuk pembangkitan elemen-elemen kunci. Kunci inilah yang memegang peranan sangat penting dalam masalah keamanan kriptografi. Semakin sulit kunci itu ditebak oleh pihak lawan maka kriptografi tersebut akan semakin aman dari serangan. Tidak seperti Pseudorandom Number Generator (PRNG) lainnya yang biasanya kurang aman terhadap serangan, Cryptographically Secure Pseudorandom Number Generator (CSPRNG) hadir dengan performansi lebih baik sehingga dapat mengatasi hal tersebut. Hal ini dimungkinkan karena Cryptographically Secure Pseudorandom Number Generator (CSPRNG) dirancang berdasarkan operasi matematika yang sulit seperti pemfaktoran bilangan menjadi faktor prima, logaritma diskrit dan sebagainya.

ISAAC adalah salah satu algoritma pembangkit bilangan acak dan cipher aliran yang didesain untuk benar-benar aman secara kriptografis. Pembangkit bilangan acak ini dapat menghasilkan bilangan yang tidak mudah diprediksi (secure) pihak lawan karena secara statistik memiliki sifat-sifat yang bagus seperti lolos uji keacakan statistik dan tahan terhadap serangan (attack) yang serius.

Pada makalah ini akan dibahas mengenai cara kerja algoritma ISAAC. Selanjutnya akan dilakukan perbandingan algoritma ISAAC dengan algoritma pembangkit bilangan acak lain yang banyak digunakan, seperti algoritma Blum Blum Shub dan Yarrow dari segi algoritma, kerumitan komputasi, performansi, dan keamanan bilangan-bilangan acak yang dihasilkan.

***Kata kunci**: PRNG*, *CSPRNG*, *random*, kriptografi, ISAAC.

1. Pendahuluan

1.1. Bilangan Acak

Bilangan acak (random) banyak digunakan di dalam kriptografi, misalnya untuk pembangkitan elemen-elemen kunci (contohnya pada algoritma One Time Pad), pembangkitan initialization vector(IV), pembangkitan parameter kunci di dalam sistem kriptografi kunci publik, dan sebagainya. Yang dimaksud dengan acak di sini adalah bilangannya tidak mudah diprediksi oleh pihak lawan.

Sayangnya sangat sulit memperoleh bilangan acak dalam praktek kriptografi. Tidak ada prosedur komputasi yang benar-benar menghasilkan deret bilangan acak yang sempurna. Bilangan acak yang dihasilkan dengan rumus matematika adalah bilangan acak semu (pseudo), karena bilangan acak yang dibangkitkan dapat berulang kembali secara periodik. Pembangkit deret bilangan acak semacam itu disebut *pseudo-random number generator* (PRNG).

1.2. Pembangkit Bilangan Acak

Bruce Schenier menggunakan tiga hal sebagai definisi dari pembangkit bilangan acak:

1. *Output* yang dihasilkan kelihatan acak (*looks random*), artinya lolos uji keacakan statistik.
2. Tidak mudah diprediksi (*unpredictable*), artinya walaupun algoritma dan bilangan-bilangan acak sebelumnya sudah diketahui, bilangan acak yang dihasilkan berikutnya harus tidak dapat dikomputasi dengan mudah.
3. Tidak dapat dihasilkan ulang (*cannot be reliably reproduced*), artinya jika dijalankan dua kali dengan *input* yang tepat sama, maka *output* yang dihasilkan benar-benar berbeda.

Pembangkit bilangan acak yang memenuhi tiga syarat di atas dinamakan *true random number generators* (*TRNG*). Sayangnya, bilangan acak yang benar-benar acak (*true random numbers*) sangat sulit dibangkitkan, terutama dengan menggunakan komputer yang dirancang secara *deterministic*. Syarat ketiga di atas hampir tidak mungkin dipenuhi oleh komputer. Oleh karena itu, yang bisa dilakukan adalah membangkitkan bilangan acak semu (*pseudo-random number*). Hal ini dapat dilakukan karena pembangkit bilangan acak semu termasuk *deterministic finite state machine*.

Yang dimaksud dengan acak dalam bilangan acak semu adalah bilangannya tidak mudah diprediksi. Bilangan acak semu pada umumnya dihasilkan dengan rumus-rumus matematika dan biasanya bilangan acak yang dibangkitkan dapat berulang kembali secara periodik. Pembangkit deret bilangan acak semacam itu disebut *pseudo-random number generator* (*PRNG*). Salah satu contoh *PRNG* adalah pembangkit bilangan acak kongruen lanjar atau *Linear Congruential Generator* (LCG).

Pembangkit bilangan acak kongruen lanjar (LCG) adalah salah satu pembangkit bilangan acak tertua dan sangat terkenal. LCG didefinisikan dalam relasi rekurens:

$$x_n = (ax_{n-1}+b) \bmod m$$

yang dalam hal ini,

xn = bilangan acak ke-n dari deretnya
xn-1 = bilangan acak sebelumnya
a = faktor pengali
m = modulus
(a, b, dan m semuanya konstanta)

Meskipun LCG secara teoritis mampu menghasilkan bilangan acak yang cukup baik, algoritma ini sangat sensitif terhadap pemilihan nilai-nilai a, b, dan m. Pemilihan nilai-nilai yang buruk dapat mengarah pada implementasi LCG yang tidak bagus.

Keunggulan LCG terletak pada kecepatannya dan hanya membutuhkan sedikit operasi bit. Sayangnya, LCG tidak dapat digunakan untuk kriptografi karena bilangan acaknya dapat diprediksi urutan kemunculannya. Oleh karena itu LCG tidak aman digunakan untuk kriptografi. Namun demikian, LCG tetap berguna untuk aplikasi non-kriptografi seperti simulasi, sebab LCG mangkus dan memperlihatkan sifat statistik yang bagus dan sangat tepat untuk uji-uji statistik.

1.3. Pembangkit Bilangan Acak yang Aman untuk Kriptografi

Banyak aspek dalam kriptografi yang membutuhkan bilangan acak, diantaranya:

- pembangkitan kunci
- nonces, atau *number used once* (angka yang digunakan sekali). Biasa digunakan untuk protokol autentikasi, agar komunikasi lama tidak dapat dimanfaatkan untuk *replay attack.*
- salts pada berbagai skema tanda-tangan, misal ECDSA, RSASSA-PSS
- one-time pads

Kualitas keacakan yang dibutuhkan untuk aplikasi-aplikasi di atas berbeda-beda. Misalnya pembentukan nonce pada sebuah protokol hanya membutuhkan keunikan. Di sisi lain, pembangkitan *master key* membutuhkan kualitas yang lebih tinggi, misal, entropi lebih. Pada kasus one-time pads, jaminan kerahasiaan sempurna hanya dapat terjadi jika kunci didapatkan dari sumber yang benar-benar acak dengan entropi tinggi.

Untuk itu, pembangkit bilangan acak yang digunakan dalam kriptografi harus dapat menghasilkan bilangan yang tidak mudah diprediksi oleh pihak lawan dan sulit untuk ditebak dengan mencoba semua kemungkinan (*brute force*). Pembangkit bilangan acak semacam ini dinamakan *cryptographically secure pseudo-random number generator* (*CSPRNG*). Persyaratan *CSPRNG* adalah:

1. Secara statistik ia mempunyai sifat-sifat yang bagus, yaitu lolos uji keacakan statistik.
2. Tahan terhadap serangan yang serius. Serangan ini bertujuan untuk memprediksi bilangan acak yang dihasilkan.

Untuk persyaratan yang kedua ini, maka *CSPRNG* hendaklah memenuhi dua persyaratan sebagai berikut:

1. Setiap *CSPRNG* seharusnya memenuhi "uji bit-berikutnya" (*next-bit test*) sebagai berikut: Diberikan *k* buah bit barisan acak, maka tidak ada algoritma dalam waktu polinomial yang dapat memprediksi bit ke-(*k*+1) dengan peluang keberhasilan lebih dari 1/2.
2. Setiap *CSPRNG* dapat menahan "perluasan status", yaitu jika sebagian atau semua statusnya dapat diungkap (atau diterka

dengan benar) maka tidak mungkin merekonstruksi aliran bilangan acak.

Kebanyakan *PRNG* tidak cocok digunakan untuk *CSPRNG* karena tidak memenuhi kedua persyaratan *CSPRNG* yang disebutkan di atas. *CSPRNG* dirancang untuk tahan terhadap bermacam-macam kriptanalis.

Perancangan *CSPRNG* dapat dibagi menjadi beberapa kelompok.

1. Perancangan berbasis primitif kriptografi. Algoritma *cipher* blok yang aman dapat dikonversi menjadi *CSPRNG* dengan mode cacah. Ini dilakukan dengan memilih kunci acak dan mengenkripsi 0, mengenkripsi 1, mengenkripsi 2, dan seterusnya (pencacahan juga dapat dimulai dari bilangan selain 0). Jelaslah bahwa periodenya akan menjadi 2_n untuk blok berukuran *n*-bit. Nilai-nilai awal tidak boleh diketahui oleh pihak lawan. Kebanyakan algoritma *cipher* aliran membangkitkan aliran bit kunci yang dikombinasikan dengan plainteks (umumnya di-*XOR*-kan); aliran kunci ini dapat juga digunakan sebagai *CSPRNG* yang bagus (meskipun tidak selalu demkian, seperti pada *RC4*).

   Salah satu *CSPRNG* yang berbasis *cipher* blok (menggunakan algoritma *DES*) dan diadopsi sebagai standar *FIPS* bekerja sebagai berikut: Masukan: *D* = informasi jam/ tanggal, *s* = umpan yang berukuran 64-bit, K = kunci. Proses: Hitung $I = DES_K(D)$. Luaran: bilangan acak sekarang $x = DES_K$ (I XOR s) lalu mutakhirkan umpan s yang baru untuk bilangan acak berikutnya sebagai $s = DES_K$ (x XOR I).

2. Perancangan berbasis teori bilangan. *CSPRNG* dapat juga dirancang berdasarkan persoalan matematika yang sulit, seperti pemfaktoran biilangan menjadi faktor prima, logaritma diskrit, dan sebagainya. Contoh dua *CSPRNG* yang berdasarkan teori bilangan adalah *Blum Blum Shub* dan modifikasi *RSA*.

3. Perancangan khusus
   Terdapat beberapa praktek PRNG yang telah dirancang khusus agar aman secara kriptografis, termasuk:
   a. Algoritma Yarrow yang berusaha mengevaluasi kualitas entropy dari masukkan, dan versi terbarunya, Fortuna, yang tidak melalukan evaluasi tersebut.
   b. Fungsi dari Microsoft‟s Cryptographic Application Programming Interface: CryptGenRandom.
   c. ISAAC berbasiskan varian dari RC4 chiper.

2. Algoritma ISAAC

ISAAC adalah sebuah pembangkit bilangan acak semu yang didesain oleh Robert Jenkins pada tahun 1996 sebagai pembangkit bilangan acak yang aman untuk kriptografi. Nama ISAAC merupakan akronim dari *Indirection, Shift, Accumulate, Add, and Count.*

Algoritma ISAAC memiliki kemiripan dengan RC4. Keduanya menggunakan array yang terdiri dari 256 integer 4 byte (disebut mm) sebagai state internal, menuliskan hasilnya ke dalam array 256 integer lain, yang kemudian akan dibaca satu per satu sampai kosong, kemudian dihitung kembali. Proses komputasi terdiri dari mengubah mm[i] dengan mm[i^128], dua elemen mm yang ditemukan dengan indirection, akumulator, dan counter, untuk setiap nilai i dari 0 sampai 255. Karena hanya membutuhkan sekitar 19 operasi 32 bit untuk setiap output 32 bit, algoritma ini sangat cepat jika digunakan untuk komputer 32 bit.

2.1. Cara Kerja

Berikut ini adalah implementasi algoritma ISAAC dalam bahasa C.

```
/*
 * & is bitwise AND, ^ is bitwise
XOR, a<<b shifts a by b
 * ind(mm,x) is bits 2..9 of x,
or (floor(x/4) mod 256)*4
 * in rngstep barrel(a) was
replaced with a^(a<<13) or such
 */

typedef  unsigned int  u4;
/* unsigned four bytes, 32 bits
*/
typedef  unsigned char u1;
/* unsigned one  byte,  8  bits
*/
#define ind(mm,x)  (*(u4 *)((u1
*)(mm) + ((x) & (255<<2))))
#define rngstep
(mix,a,b,mm,m,m2,r,x)
{
  x = *m;
  a = (a^(mix)) + *(m2++);
```

```
  *(m++) = y = ind(mm,x)+ a + b;
  *(r++) = b = ind(mm,y>>8) + x;
}

static void isaac
(mm,rr,aa,bb,cc)
u4 *mm;      /* Memory: array of
SIZE ALPHA-bit terms */
u4 *rr;      /* Results: the
sequence, same size as m */
u4 *aa;      /* Accumulator: a
single value */
u4 *bb;      /* the previous
result */
u4 *cc;      /* Counter: one
ALPHA-bit value */
{
  register u4
a,b,x,y,*m,*m2,*r,*mend;
  m=mm; r=rr;
  a = *aa; b = *bb + (++*cc);
  for (m = mm, mend = m2 = m
+128; m<mend; )
  {
    rngstep( a<<13, a, b, mm, m,
m2, r, x);
    rngstep( a>>6 , a, b, mm, m,
m2, r, x);
    rngstep( a<<2 , a, b, mm, m,
m2, r, x);
    rngstep( a>>16, a, b, mm, m,
m2, r, x);
  }
  for (m2 = mm; m2<mend; )
  {
    rngstep( a<<13, a, b, mm, m,
m2, r, x);
    rngstep( a>>6 , a, b, mm, m,
m2, r, x);
    rngstep( a<<2 , a, b, mm, m,
m2, r, x);
    rngstep( a>>16, a, b, mm, m,
m2, r, x);
  }
  *bb = b; *aa = a;
}
```

keterangan:
rngstep():
rngstep() pada dasarnya merupakan sebuah loop. Mengulanginya sebanyak empat kali akan mengurangi overhead pada loop.

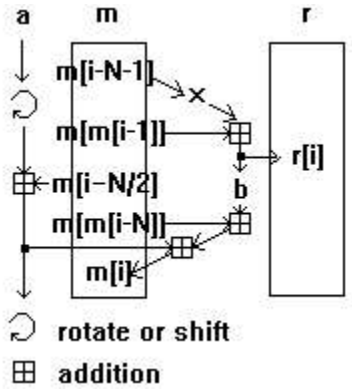
*m++
Mengganti m[i] dengan *m++, r[i] dengan *r++. dan m[i(SIZE/2)] dengan *m++ mengurangi biaya untuk mencari dalam array yang dapat diprediksi. m adalah sebuah pointer, *menunjukkan lokasi yang ditunjuknya, dan ++ memindahkan pointer ke lokasi berikutnya.
a^(mix)
^ berarti XOR dan << dan >> adalah shift (pergeseran). Setiap pemanggilan terhadap rngstep() akan melakukan fungsi ini. Rangkaian fungsi ini juga menyebabkan a untuk mencapai state yang tidak teratur dengan lebih cepat. Mungkin saja ini dapat mempengaruhi bias keseluruhan dari generator, tidak ada cara untuk mengetahuinya. Yang perlu diketahui adalah fungsi ini untuk melakukan permutasi pada a.

cc
Sebuah counter dimasukkan dan digunakan hanya satu kali setiap pemanggilan. cc dan i bersama-sama menjamin minimum panjang siklus dari 240 nilai. Tidak ada siklus yang lebih pendek dari itu. Tidak ada initial state yang buruk, bahkan state dari semua no.

ind(x)
Bit indirection yang digunakan pada ISAAC adalah 2..9 untuk x dan 10..17 untuk y.

Dapat disimpulkan sementara, ISAAC membutuhkan 18,75 instruksi untuk memproduksi setiap nilai 32-bit. Tidak ada siklus pada ISAAC yang lebih sedikit dari 240 nilai. Tidak ada initial state buruk. State internal memiliki 8.288 bit, sehingga panjang siklus yang diperkirakan adalah 28287 panggilan. Deduksi pada internal state tidak dapat dilakukan, dan hasilnya terdistribusi dengan merata.

Ilustrasi algoritma ISAAC seperti pada gambar berikut.



Gambar 1

2.2. Kelebihan dan Kekurangan Algoritma ISAAC

Algoritma ISAAC tidak menggunakan perkalian ataupun mod dalam proses pencarian hasilnya. Perkalian dan mod membutuhkan waktu pemrosesan yang lambat.

Algortitma dengan perkalian dan mod, misal x=ax+b mod p hanya menghasilkan nilai antara 0 sampai dengan p-1 untuk setiap p. Sedangkan ISAAC dapat menghasilkan nilai 32 bit apapun. Selain itu, ax+b memiliki pola yang mudah terdeteksi, misal x_{i+1} pasti ax_i+b mod p.

ISAAC dapat bekerja dengan sangat baik pada platform 32-bit. Agar dapat bekerja dengan baik pada mesin 64-bit, perlu dilakukan penyesuaian pada algortimanya. Sehingga untuk masing-masing platform () membutuhkan algoritma yang sedikit berbeda.
Kekurangan algortima ISAAC terletak pada kompleksitasnya yang cukup tinggi sehingga tidak mudah untuk dihingat. Namun kekurangan ini bukanlah hal yang sangat krusial.

2.3. Perbandingan dengan Algoritma Lain yang Sejenis

ISAAC merupakan pengembangan lebih lanjut dari algortima RC4. Sebelum ISAAC, ada dua algoritma lain hasil pengembangan RC4 yang dipublikasikan.

Yang pertama adalah IA (*Indirection, Adition*). IA bias tetapi aman. Algoritma ini kebal terhadap eliminasi Gauss. IA didesain untuk memenuhi kebutuhan berikut:
1. Deduksi internal state dari hasil keluaran harus tidak bisa dilakukan.
2. Kode dapat diingat dengan mudah.
3. Algoritma ini harus dapat berjalan secepat mungkin.

Kemudian IA dikembangkan lagi sehingga muncullah IBAA (*Indirection, Barrelshift, Accumulate and Add*). IBAA menghilangkan bias pada IA tanpa mengurangi tingkat keamanannya. Spesifikasi yang kemudian ditambahkan pada IBAA:
1. IBAA harus aman secara kriptografis.
2. Tidak boleh ada bias yang terdeteksi untuk setiap siklus.
3. Siklus pendek harus sangat jarang terjadi.

Setelah itu barulah muncul ISAAC (*Indirection, Shift, Accumulate, Add, and Count*), sebagai pengembangan lebih lanjut dari algoritma-algoritma di atas.

Pada ISAAC, kode tidak lagi mudah diingat. Sebagai gantinya, ISAAC memiliki beberapa kelebihan, yaitu:
1. State teratur dapat menjadi acak dengan cepat.
2. Tidak ada siklus pendek sama sekali.

2.4. Kriptanalisis pada Algoritma ISAAC

Kriptanalisis sudah pernah dilakukan oleh Marina Pudovkina(2001). Serangan yang ia lakukan dapat mengembalikan initial state dengan kompleksitas yang diperkirakan kurang dari waktu yang dibutuhkan untuk mencari semua akar dari initial state yang mungkin. Pada prakteknya, ini berarti serangan membutuhkan $4{,}67 \times 10^{1240}$, bukan 10^{2466}.

Walaupun begitu, hasil ini tetap bukan merupakan angka yang kecil, sehingga ISAAC masih dapat dibilang aman.

Pada tahun 2006, Jean-Philippe Aumasson menemukan beberapa state lemah. State keempat (dan terkecil) dari state-state lemah yang ditemukan mengarah pada output dengan tingkat bias tinggi dari hasil putaran pertama ISAAC dan memungkinkan penurunan dari internal state, mirip dengan kelemahan pada RC4. Masih belum jelas apakah penyerang dapat mengetahui hanya dari output apakah generator sedang dalam state lemah tersebut atau tidak. Ia juga menunjukkan bahwa kriptanalisis yang pernah dilakukan Paul-Preneel cacat, karena menggunakan algoritma erroneous, bukan ISAAC yang sebenarnya.

3. Kesimpulan

Pseudo-random number generator dibutuhkan untuk berbagai hal dalam kriptografi. Namun, tidak semua PRNG memenuhi kualifikasi untuk dapat digunakan dalam kriptografi. Pembangkit bilangan acak yang digunakan dalam kriptografi harus dapat menghasilkan bilangan yang tidak mudah diprediksi oleh pihak lawan dan sulit untuk ditebak dengan mencoba semua kemungkinan (*brute force*).

Untuk memenuhi kebutuhan akan CSPRNG, beberapa algoritma dikembangkan. Salah satunya adalah ISAAC. ISAAC merupakan pengembangan lebih lanjut dari RC4. Sebelum ISAAC, dua algoritma lain yang juga berasal dari

RC4 adalah IA dan IBAA. Kecepatan dan minimnya bias pada IA, IBAA, dan ISAAC membuat ketiganya berguna untuk simulasi dan memenuhi kualifikasi untuk digunakan dalam kriptografi.

Kelebihan ISAAC dibandingkan dengan algoritma CSPRNG lain, terutama pendahulunya (IA dan IBAA) adalah dari sisi kecepatan. Operasi pada ISAAC tidak mengandung perkalian maupun pembagian mod yang membutuhkan waktu lama.

Sejak ditemukan pada tahun 1996, sampai saat ini belum ada yang berhasil memecahkan (mengkriptanalis) algoritma ini. Beberapa orang sudah pernah mencoba memecahkan algortima ini, tetapi hasilnya masih belum cukup untuk menyatakan bahwa ISAAC adalah algoritma yang tidak cocok untuk kriptografi. Bahkan bias pada ISAAC pun masih belum dapat ditemukan.

DAFTAR PUSTAKA

[1]http://en.wikipedia.org/wiki/Cryptographically_secure_pseudorandom_number_generator

[2]http://en.wikipedia.org/wiki/ISAAC_(cipher)

[3]http://www.burtleburtle.net/bob/rand/isaac.html

[4]http://www.burtleburtle.net/bob/rand/isaacafa.html

[5]http://erngui.com/rng/

[6]http://world.std.com/~cme/P1363/ranno.html

[7]Munir, Rinaldi. Diktat Kuliah IF5054 Kriptografi. 2006.