Studi Mengenai Algoritma ISAAC dan Metode Kriptanalitiknya

Dani – NIM : 13504060

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if14060@students.if.itb.ac.id

Abstrak

Pada makalah ini akan dibahas mengenai PRNGs (Pseudorandom Number Generator) dengan algoritma IA, IBAA, dan ISAAC yang banyak digunakan oleh system keamanan real-world untuk menggenerate kunci kriptografi, vektor insialisasi (IV), dan nilai-nilai lain yang diasumsikan acak. PRNGs dapat disebut sebagai tipe unik tersendiri dari primitif kriptografi dan harus dianalisis dengan cara-cara yang relevan. ISAAC memerlukan 18.75 instruksi untuk menghasilkan nilai 32-bit. Tidak ada cycle pada ISAAC yang lebih pendek dari 2^{40}. Panjang cycle yang diharapkan adalah 2^{8295} values.

Kata kunci: ISAAC, IBAA, IB, RC4, kriptografi, kriptanalisis, enkripsi, dekripsi, Pseudo Random Number Generator

1. Pendahuluan

Hampir tidak ada aplikasi kriptografi yang didesign dengan baik namun tidak menggunakan bilangan acak. Session keys, IV, parameter unik pada operasi digital signature, pembangkitan elemen-elemen kunci (contohnya pada algoritma OTP), dan nilai-nilai lain diasumsikan acak oleh pembuat sistem. Akan tetapi, banyak aplikasi kriptografi yang tidak memiliki sumber yang reliable untuk bit yang benar-benar acak. Tidak ada prosedur komputasi yang benar-benar menghasilkan deret bilangan acak sempurna. Bilangan acak yang dihasilkan dengan rumus-rums matematika adalah bilangan acak semu, karena bilangan acak yang dibangkitkan dapat berulang kembali. Mekanisme kriptografi yang disebut Pseudo Random Number Generator (PNRG) digunakan untuk menghasilkan nilai-nilai tersebut. PRNG mengumpulkan keteracakan dari berbagai low-entropy input stream dan mencoba menghasilkan output yang dalam prakteknya tidak dapat dibedakan dari random stream sejati.

Terdapat tiga algoritma pembangkit bilangan acak semu yang akan dibahas di sini yaitu IA, IBAA, dan ISAAC, di mana ISAAC merupakan pengembangan dari dua algoritma sebelumnya. IA (Indirection, Addition) masih memiliki bias, namun lewat pengamatan sekilas algoritma ini tampak aman. Algoritma ini tahan terhadap eliminasi Gaussian. IBAA (Indirection, Barrelshift, Accumulate and Add) mengeliminasi bias yang terdapat di IA tanpa mengurangi keamanannya. ISAAC (Indirection, Shift, Accumulate, Add and Count) lebih cepat daripada IBAA, menjamin tidak adanya bibit (*seeds*) yang buruk dan membuat state-state yang teratur menjadi tidak teratur dengan cepat.

IA didesain dengan syarat:
- state internal tidak dapat diturunkan dari hasil.
- memiliki kode yang mudah diingat.
- memiliki waktu ekseskusi yang singkat.

IBAA memiliki tujuan tambahan yaitu:
- aman secara kriptografi.
- tidak ditemukan bias untuk seluruh panjang cycle.
- cycle yang pendek harus sangat jarang ditemukan.

Sebuah pembangkit bilangan acak yang memiliki tingkat bias yang dapat diterima ditemukan. Pembangkit tersebut menggunakan accumulator dan barrelshift. IBAA dibentuk dengan mengombinasikan pembangkit bilangan acak itu dengan IA.

ISAAC mereduksi syarat mudah diingat namun menambah beberapa syarat lain:
- kode C nya harus dioptimasi untuk kecepatan.
- state-state yang teratur harus menjadi tidak teratur secara cepat.
- tidak boleh ada cycle yang pendek.

ISAAC memerlukan 18.75 instruksi mesin untuk menghasilkan nilai 32-bit. ISAAC berguna sebagai stream cipher, untuk simulasi, dan sebagai pembangkit bilangan acak semu umum. ISAAC-64 adalah mesin versi 64-bit yang

memerlukan 19 instruksi untuk menghasilkan nilai 64-bit.

Notasi yang akan digunakan dalam makalah ini : ALPHA adalah log dari jumlah bits pada array, mencari sebuah nilai dalam array memerlukan offset sebesar ALPHA bits. SIZE adalah 2^{ALPHA}, ukuran dari array.

2. Algoritma Pendukung ISAAC

2.1 RC4

Algoritma ISAAC diturunkan dari RC4, maka akan diberikan sedikit ulasan mengenai algoritma RC4 ini.

Implementasi algoritma RC4 dalam bahasa C:

```
/*
 * SIZE is (1<<ALPHA) = (1 times 2
to the 8th) = 256.

 * ind(x) is the low order 8 bits of
x, or x mod 256.
 */

#define ALPHA      (8)
#define SIZE       (1<<ALPHA)
#define ind(x)     (x&(SIZE-1))

static void rc4(m,r,aa)
int *m;
/* Memory: array of SIZE ALPHA-bit
terms */

int *r;
/* Results: the sequence, same size
as m */

int *aa;
/* Accumulator: a single value */

{
  register int a,x,y,i;
  a=*aa;

  for (i=0; i<SIZE; ++i)
  {
    x=m[i];
    a=ind(a+x);
    y=m[a];
    m[i]=y; m[a]=x;
    r[i] = m[ind(x+y)];
  }

  *aa=a;
}
```

Nilai acak dalam r dipilih dari array m, dan dua elemen dari m dipertukarkan untuk setiap byte yang dilaporkan Keamanan algoritma ini ditentukan dari kesulitan untuk mengetahui setiap nilai yang terdapat dalam m, dan dari kesulitan untuk mengetahui lokasi mana dari m yang digunakan untuk memilih setiap nilai dari urutan nilai yang dihasilkan.
Pertukaran posisi yang diketahui dengan posisi yang tidak diketahui merintangi eliminasi Gaussian. Gauss hanya dapat diterapkan jika diketahui sebuah nilai yang digunakan dalam dua persamaan. Karena setiap pertukaran mungkin merubah semua nilai, tidak pernah dapat diketahui kapan perubahan nilai pada suatu posisi terjadi, sehingga tidak ada cara untuk memastikan bahwa sebuah persamaan menggunakan nilai yang sama dengan persamaan lain.

Terdapat bias yang dapat dideteksi dari hasil RC4, di samping celah menuju internal state algoritma ini. Hubungan r[i] + x = i muncul 1/256 terlalu sering (dua kali lebih banyak dari yang diharapkan), seperti juga r[i] + y = a. Celah ini hanya memungkinkan posisi dari sebuah nilai diketahui dengan kemungkinan 1/128, dan tidak cukup untuk menurunkan internal statenya sendiri.
State dengan m[i] = 1 dan a = I seluruhnya terdapat pada cycle yang pendek. Terdapat 254! cycle pendek pada 65280 nilai. RC4 diinisialisasi sehingga tidak pernah menemukan 1/65536 dari seluruh state internal yang terdapat dalam cycle pendek tersebut.
Untuk setiap urutan nilai yang dihasilkan oleh RC4, terdapat 256 state internal yang menghasilkan urutan yang sama dengan ind(x+y). Jika (m[0..255]=m_0..m_255,a,i) adalah salah satu state tersebut, maka (m[0..255]=m_n..m_((n+255) mod 256), (a-n) mod 256, (i-n) mod 256) adalah state serupa yang lain. Meskipun ind(x+y) adalah sama untuk setiap urutan tersebut, m[ind(x+y)] akan berbeda untuk setiap urutan yang ekivalen, yang dalam kasus ini akan terdapat 256, 128, 64, 32, 16, 8, 4, 2, atau 1 cycle tersebut.

2.2 IA (Indirection, Addition)

Implementasi algoritma IA dalam bahasa C:

```
typedef  unsigned int  u4;
/* unsigned four bytes, 32 bits */

#define ALPHA      (8)
```

```
#define SIZE       (1<<ALPHA)
#define ind(x)     (x&(SIZE-1))

static void ia(m,r,bb)
u4 *m;
/* Memory: array of SIZE ALPHA-bit
terms */

u4 *r;
/* Results: the sequence, same size
as m */

u4 *bb;
/* the previous result */
{
  register u4 b,x,y,i;

  b = *bb;
  for (i=0; i<SIZE; ++i)
  {
    x = m[i];

    m[i] = y = m[ind(x)] + b;
    /* set m */

    r[i] = b = m[ind(y>>ALPHA)] + x;
    /* set r */
  }
  *bb = b;
}
```

Seperti RC4, IA beroperasi pada array rahasia dengan 256 nilai, namun setiap nilai dalam array tersebut harus mengandung sekurang-kurangnya 2 ALPHA bits. IA menggunakan indirection untuk menentukan hasilnya, namun hasil yang diberikan IA adalah jumlah semua nilai di array, bukan nilai sebenarnya dalam array tersebut. IA juga tidak mempertukarkan nilai di dalam array, akan tetapi berjalan sepanjang array dan menambahkan nilai-nilai yang dipilih secara acak semu pada nilai lama yang telah ada.

IA bersifat reversibel, setiap state internal memiliki tepat satu state pendahulu. Panjang cycle rata-rata dari setiap element dalam pemetaan reversibel dari sejumlah s state adalah s/2, dan panjang cycle rata-rata dari seluruh element dalam pemetaan ireversibel adalah sqrt(s). Sebagai tambahan dari kepemilikan setiap state internal pada beberapa cycle, pembangkit bilangan reversibel juga cenderung memiliki lebih dari setengah state pada cycle yang sama, jika diberikan urutan distribusi yang seragam.

Perlu dicatat bahwa dalam IA ketika x ditambahkan ke r[i] = b, x tidak lagi berada dalam m. Jadi x datang dari kumpulan nilai yang berbeda dengan term acak semu yang ditambahkan dengan x. Jika hal ini tidak menjadi masalah, IA akan tidak reversibel dan hasilnya akan menjadi bias dalam hal nilai genap.

Kedua indirection mengelompokkan hasil pengguna. r[i] adalah nilai yang lama dari m[i], akan tetapi dengan penjumlahan nilai yang dipilih secara acak semu. Nilai baru dari m[i] adalah hasil sebelumnya dari pengguna, dengan penjumlahan nilai berbeda yang di pilih secara acak semu. Jika nilai acak semu diasumsikan tidak diketahui, hal ini cukup untuk menangkal eliminasi Gaussian. Menebak nilai yang dipilih berarti menebak 8 bit informasi per nilai.

Terdapat celah menuju state internal IA. Hubungan ind(m[i]) = ind(r[i]-1) adalah 1/256 terlalu memungkinkan, seperti juga ind(m[i-SIZE]>>8)=ind((r[i]>>8-i). Hal tersebut muncul ketika indirection acak semu memilih dirinya sendiri. Setiap keterhubungan memegang 1/128 dari waktu.

Metode yang digunakan untuk menghindari celah ini adalah dengan membatasi setiap pilihan acak semu pada setengah dari array yang tidak mengandung nilai yang digunakan sebagai pilihan. Hal ini akan mengakibatkan hanya 128 nilai untuk setiap pilihan acak semu, jika diberikan 256 hubungan yang benar 1/128 dari waktu. Akan tetapi modifikasi yang ditawarkan membuat kode yang dibuat menjadi lebih lambat, lebih komplex, dan lebih bias, sehingga hal tersebut tidak dilakukan.

Bias dapat dideteksi dalam IA dengan menggunakan pengujian gap yang berkorelasi. Bias-bias ini secara alami serupa dengan bias yang ditemukan pada *lagged-Fibonacci* maupun pembangkit *add-carry*.

Tidak ada serangan efisien yang diketahui yang dapat dilakukan terhadap IA. Sebuah serangan *brute-force* dibuat di mana serangan tersebut memecahkan RC4 dengan ALPHA=4 dalam dua sampai sepuluh menit. Serangan *guess and generate* yang menerapkan persamaan IA pada penebakan awal memecahkan IA setelah kurang lebih 2^{13} nilai ketika ALPHA=3. Kedua serangan tersebut tidak dapat diterapkan untuk nilai ALPHA yang besar, seperti ALPHA=8.

Membuktikan keamanan IA memerlukan pembuktian bahwa tidak ada algoritma yang mampu menurunkan state internal IA secara efisien. Sejauh ini, tidak ada algoritma yang dapat melakukan hal tersebut, termasuk algoritma eliminasi Gaussian. Hal ini memang bukan menjadi bukti, namun dapat menjadi sebuah awal dari bukti.

2.3 IBAA (Indirection, Barrelshift, Accumulate and Add)

IA dikembangkan lebih lanjut dan menghasilkan algoritma IBAA, di mana sebagai tambahan cepat, mudah diingat, dan tahan terhadap eliminasi Gaussian IBAA juga tidak memiliki bias yang dapat dideteksi untuk seluruh panjang cycle. Cycle yang pendek juga harus sangat jarang ditemukan.

Implementasi algoritma IBAA dalam bahasa C:

```
/*
 * ^ means XOR, & means bitwise AND,
a<&ltb means shift a by b.

 * barrel(a) shifts a 19 bits to the
left, and bits wrap around

 * ind(x) is (x AND 255), or (x mod
256)
 */

typedef  unsigned int  u4;
/* unsigned four bytes, 32 bits */

#define ALPHA      (8)
#define SIZE       (1<<ALPHA)
#define ind(x)     ((x)&(SIZE-1))
#define barrel(a)
(((a)<<19)^((a)>>13))
/* beta=32,shift=19 */

static void ibaa(m,r,aa,bb)
u4 *m;
/* Memory: array of SIZE ALPHA-bit
terms */

u4 *r;
/* Results: the sequence, same size
as m */

u4 *aa;
/* Accumulator: a single value */

u4 *bb;
/* the previous result */
{
  register u4 a,b,x,y,i;

  a = *aa; b = *bb;
  for (i=0; i<SIZE; ++i)
  {
    x = m[i];

    a = barrel(a) +
        m[ind(i+(SIZE/2))];
    /* set a */

    m[i] = y = m[ind(x)] + a + b;
    /* set m */

    r[i] = b = m[ind(y>>ALPHA)] + x;
    /* set r */
  }
  *bb = b; *aa = a;
}
```

Tidak adanya bias pada IBAA datang dari penjumlah a. Term yang ditambahkan ke a adalah m[ind(i+(SIZE/2))]. Mungkin akan timbul pertanyaan mengapa tidak x dan y yang digunakan, mengingat bahwa nilai-nilai tersebut telah ada di register. Keputusan ini dibuat berdasarkan test-test yang telah dilakukan. Test tersebut dilakukan terhadap IBAA, kecuali nilai m[ind(x)] dan m[ind(y>>ALPHA)] diganti dengan 0. Pembangkit tersebut disesuaikan dengan hanya memiliki 8 term (bukan 256) masing-masing 3 bit (bukan 32 bit). Dengan total 30 bit dari state, pembangkit tersebut memiliki panjang cycle maksimum sebesar 2^{30}. ind(i+(SIZE/2)) diganti dengan ind(i+j), untuk setiap j dalam 0..7. Setiap pembangkit menghasilkan sekumpulan 2^{27} nilai, atau 2^{30} nilai. Tidak ada cycle yang terdeteksi. *Low-order* bit dari tiap nilai dibuang, menghasilkan sekelompok nilai yang terdiri dari 2 bit. Pengujian gap diterapkan terhadap setiap nilai-nilai yang dihasilkan, dengan pelacakan gap dengan panjang 0..63. Hasil yang diharpakan adalah 63, namun hasil pengujian memunculkan nilai-nilai 684, 412, 208, 201, 212, 203, 682, dan 13584. Perbedaan dari 63 adalah sebanding dengan nilai bias yang terdeteksi.

Tidak ada test lain yang mampu mendeteksi jumlah bias yang cukup signifikan, sehingga keputusan yang diambil hanya berdasarkan test di atas. Bias tampak berkurang seiring dengan jarak dari kedua ujung, sehingga m[ind(i+SIZE/2))] dipilih.
barrel(a) adalah permutasi dari a, dan bersifat nonlinear jika dikombinasikan dengan penjumlahan. Permutasi membantu memastikan bahwa setiap nilai adalah hampir sama. Akan tetapi, kemanan dari IBAA tidak bergantung pada kenonlinearannya. Kemanan IBAA bergantung pada indirection m[ind(x)] dan m[ind(y>>ALPHA)].
Jika m[i], m[ind(x)] dan m[ind(y>>ALPHA)] diperlakukan secara terpisah sebagai tidak diketahui, maka setiap set dari persamaan memiliki setidaknya 4/3 nilai yang tidak diketahui sebanyak persamaan. Pada set dari 3n persamaan (n menentukan a, n menentukan m, dan n menentukan r), akan dihasilkan setidaknya

4n nilai yang tidak diketahui: n dari setiap a, m[i], m[ind(x)] dan m[ind(y>>ALPHA)]. Mengeliminasi setiap subset dari persamaan ini hanya meningkatkan ratio dari nilai yang tidak diketahui untuk persamaan.
Jika pemetaan reversibel memiliki N kemungkinan nilai, maka peluang titik awal terdapat pada sebuah cycle dengan panjan N/x atau kurang adalah 1/x. Banyaknya state internal dari IBAA adalah 2^{8264}, sehingga peluang sebuah cycle yang dipilih acak lebih pendek dari 2^{40} adalah kurang lebih $2^{-8224.}$

3. ISAAC (Indirection, Shift, Accumulate, Add and Count)

3.1 Algoritma

IBAA dikembangkan agar menjadi lebih cepat, lebih berarti, lebih ramping, dan tidak memiliki cycle yang pendek sama sekali – dengan bayaran menjadi lebih sukar untuk diingat. Hasil yang diperoleh adalah ISAAC. Jika initial state adalah seluruhnya nol, maka setelah sepuluh kali pemanggilan nilai dari aa, bb, dan cc dalam heksadesimal akan berisi d4d3f473, 902c0691, dan 0000000a.

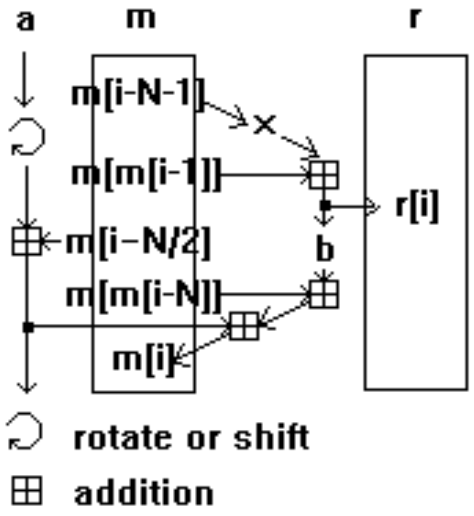
Implementasi algoritma IBAA dalam bahasa C:

```
/*
 * & is bitwise AND, ^ is bitwise
XOR, a<&ltb 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<&lt13) 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;
}
```

Internal state dari algoritma ini adalah array berukuran 256 dari nilai 32-bit, dan pada setiap round dihasilkan array 256 32-bit yang lain. Dalam hal ini, α menunjukkan initial state dan αi mengacu pada elemen ke i, sedangkan ω adalah keluaran pertamanya. Algoritma pembangkitan mengambil tiga parameter sebagai nilai awal dari tiga variabel aa, bb, dan cc. Nilai awal ini adalah

publik dan bukan merupakan bagian initial state yang rahasia.



Gambar 1
Diagram yang menunjukkan ISAAC menghasilkan sebuah nilai di r[] dan mengganti satu term di m[].

3.2 Perbedaan dari ISAAC dan IBAA

- rngstep()
Makro rngstep() sebenarnya adalah inner loop dari IBAA. Mengulangi makro ini sebanyak empat kali mengurangi overhead dari loop dan tidak mengubah hasil.

- *m++
Mengganti m[i] dengan *m++, r[i] dengan *r++, dan m[i(SIZE/2)] dengan *m2++ mengurangi harga yang harus dibayar untuk mencari term dalam posisi array yang dapat diprediksi. Hal ini juga tidak mempengaruhi hasil yang diperoleh.

- a^(mix)
Barrelshift dari IBAA digantikan dengan empat fungsi: a^(a<<13), a^(a>>6), a^(a<<2), and a^(a>>16), dengan ^ berarti XOR dan << dan >> adalah shift. Setiap pemanggilan terhadap rngstep() melakukan salah satu dari fungsi ini. Ketika mesin tidak memiliki instruksi barrelshift, hal ini menghemat satu instruksi setiap rngstep(). Fungsi-fungsi ini juga menyebabkan a untuk turun dalam 12 rngstep(). Hal ini menyebabkan state-state yang teratur menjadi tidak teratur dengan lebih cepat. Mungkin hal ini mempengaruhi bias secara keseluruhan dari pembangkit bilangan, namun tidak ada jalan untuk membuktikannya. Perlu dicatat bahwa setiap fungsi ini adalah permutasi dari a.

- cc
Sebuah counter disertakan dan digunakan (diincrement) hanya sekali per pemanggilan. Cara ini ditemukan oleh Bill Chambers. cc dan i bersama-sama menjamin panjang cycle minimunm adalah 2^{40}. Tidak ada cycle yang ditemukan sependek itu. Tidak ada initial state buruk yang terdapat dalam algoritma ini, bahkan jika initial state nya seluruhnya nol. Pengujian telah menunjukkan bahwa penambahan nilai independent terhadap b tidak mempengaruhi keamanan dan bias dari pembangkit ini.

- ind(x)
Indirection bit yang digunakan pada ISAAC adalah 2..9 untuk x dan 10..17 untuk y. (IBAA menggunakan 0..7 dan 8..15). Hal ini mengurangi satu lagi instruksi dari setiap pencarian indirect. Pengujian menunjukkan bahwa pemilihan bit indirection tidak mempengaruhi bias dan keamanan, jika tidak ada bit yang digunakan lebih dari satu kali.

Seperti telah disebutkan, ISAAC memerlukan 18.75 instruksi untuk menghasilkan nilai 32-bit (dengan optimasi yang sama, IA memerlukan 12.56 instruksi untuk menghasilkan nilai 32-bit). Tidak ada cycle pada ISAAC yang lebih kecil dari nilai 2^{40}. Tidak ada intial state yang buruk. State internal memiliki 8288 bit, sehingga panjang cycle yang diharapkan adalah 2^{8287}. Menurunkan state internal dari ISAAC tampak tidak dapat dilakukan. Hasil dari ISAAC juga tidak bias dan terdistribusi secara seragam.

3.3 Pengujian.

Terdapat dua jenis pengujian yang dilakukan, yaitu pengujian frekuensi dan pengujian jarak. Pengujian frekuensi menghitung seberapa sering suatu nilai muncul. Pengujian jarak menghitung jarak diantara tiap kemunculan nilai dalam hasil. Sebagai contoh, “abcdeaf” memiliki jarak 4 diantara kemunculan “a”.

Dilakukan 8 pengujian dengan algoritma ISAAC untuk membangkitkan 30 bilangan acak, hasil-hasil yang diperoleh adalah sebagai berikut:

- Pengujian 1

1876896121	769113277	1506746693
254040116	-296334997	1450626228
1054405739	-1528548556	-1470120982
770587355	358934438	1178439787
-1824065376	-1035967675	529483845
-1796807439	776943401	-1153872004
1468814451	1658530765	-1376256701
1560262430	1987604852	599312535
355124467	814253970	43231357
-121909755	1198612528	-964766843

- Pengujian 2

-24960013	2057366220	1975873053
43515644	-300919252	-425040335
-1893182350	-2097045963	1889396029
1902300971	-1778390481	-896456792
2082917723	-1796114394	-953706119
1511587013	486173370	-1077252652
1797209019	-1076508580	-857929914
-875941015	1772380154	-1810155236
-1024791586	-760052586	1094712948
-2067511382	-1128998633	-1509671116

- Pengujian 3

-386612371	436989268	261248082
-1021856608	179093517	1637048717
977900842	1630591764	-466982494
-129385141	440589121	-1320454580
1741805364	2068780725	-1560641846
690272540	922225562	-216926573
476319974	-1685117921	1904879699
137936568	-1589243515	389068046
594009197	-1332416797	1712583106
-1122484638	155777329	-1498829516

- Pengujian 4

148202776	2091679392	-67688055
2053050857	-1080510852	544141781
-1492351816	180316978	611213293
-26229342	-231236086	-560752149
-105995659	1946340065	-1028824467
42680088	1867635737	69815608
-1731486821	-1278740550	-1011777713
1006065875	1785474501	-2052109615
-1033002505	768852781	-117584342
-406369419	1600172188	2036660485

- Pengujian 5

2032462413	-2041652092	-2044765329
1298183243	-2008293495	102057630
-1549168143	-173570469	-176466852
-784206622	1926139865	1300372123
-1296542062	-1499486030	1236975952
500188239	-863419163	-472876976
223911320	-622870294	1543289639
-43536645	-153860867	412015193
127126789	157596404	-325651283
1241977684	-510053227	2022767544

- Pengujian 6

979616359	-1558617639	-679995306
376860872	-2017039125	-115233355
2014033531	1281684406	-1777723969
-329898388	-1751441776	-1693077587
-1265326930	-362525076	-607583831
59056059	-1044680001	-624500492
-51497786	536414994	767850906
2028676803	-938452287	-407180206
1093974052	2083211790	1265875631
-1470429265	-1249703736	470029764

- Pengujian 7

-283753665	-600019628	-1133655583
1483954431	92983654	-231784903
1297494709	-250283507	-960208217
1199221902	773716599	-1752311884
1527735374	971180165	-120816409
1717036236	2073668784	-161427071
-1877270822	509936200	-825073099
235290345	1527894443	392392293
1278767837	-1815605963	1835867461
460485895	-767290022	-537142707

- Pengujian 8

-1674560058	1323521917	-1058488509
425627896	-435114578	-842033872
-300010795	180879611	420246108
22954846	-921236128	1050573631
459185310	935807993	-691165145
-1015298430	-1742395580	-1611903631
1498603346	-985747496	1802933300
-1662634507	-1478855633	117364299
-755351645	-668724000	-851767116
676402268	1793944583	-958904316

Dari hasil pengujian dapat dilihat bahwa algoritma ISAAC menghasilkan nilai-nilai yang tampak acak dan sulit diprediksi, meskipun hal ini juga dipengaruhi oleh jumlah pengujian yang sangat sedikit (hanya 8 kali) dan jumlah bilangan dibangkitkan (hanya 30 nilai). Pengujian frekuensi dan pengujian jarak untuk data-data di atas menghasilkan nilai 0 untuk semua bilangan karena tidak terdapat nilai yang berulang. Perlu dicatat bahwa pengujian sederhana di atas tidak memeriksa keamanan algoritma ini melainkan hanya menguji kemampuan algoritma ini dalam menghasilkan barisan bilangan acak semu, di mana hal ini dapat dibuktikan dengan melihat tidak adanya keterhubungan kasat mata untuk nilai-nilai yang diperoleh.

Jika diobservasi lebih jauh lagi, terdapat beberapa kelemahan pada algoritma ISAAC yang dapat dieksploitasi oleh penyerang. Kelemahan-kelemahan tersebut akan dibahas pada bagian selanjutnya dari makalah ini.

4. Serangan terhadap ISAAC

4.1 Pembangkit Bilangan Acak yang Aman untuk Kriptografi

Pembangkit bilangan acak yang dapat menghasilkan bilangan yang tidak dapat diprediksi oleh pihak lawan cocok untuk kriptografi; pembangkit tersebut dinamakan cryptographically secure pseudo random number generator (CSPRNG).

Persyaratan CSPRNG adalah:
- Secara statistik ia mempunyai sifat-sifat yang bagus (yaitu lolos uji kelayakan statistik).
- Tahan terhadap serangan (attack) yang serius. Serangan ini bertujuan untuk memprediksi bilangan acak yang dihasilkan.

Untuk persyaratan yang kedua ini, maka CSPRNG hendaklah memenuhi dua persyaratan sebagai berikut:
- Setiap CSPRNG seharusnya memenuhi "uji bit berikutnya" (next-bit test) sebagai berikut: diberikan k buah bit barisan acak, maka tidak ada algoritma dala waktu polinomial yang dapat memprediksi bit ke (k+1) dengan peluang keberhasilan lebih besar dari ½.
- Setiap CSPRNG dapat menahan "perluasan status", yaitu jika sebagian atau semua statusnya dapat diungkap (dapat 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 kriptanalisis.

Perancangan CSPRNG dibagi ke dalam tiga kelompok:
- Perancangan berbasis primitif kriptografi

Algoritma cipher blok yang aman dapat dikonversi menjadi CPSRNG 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 (kunci dan plainteks) 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 demikian, seperti pada RC4).

- Perancangan berbasis teori bilangan

CSPRNG dapat juga dirancang berdasarkan persoalan matematika yang sulit seperti pemfaktoran bilangan menjadi faktor prima, logaritma diskrit, dan sebagainya.

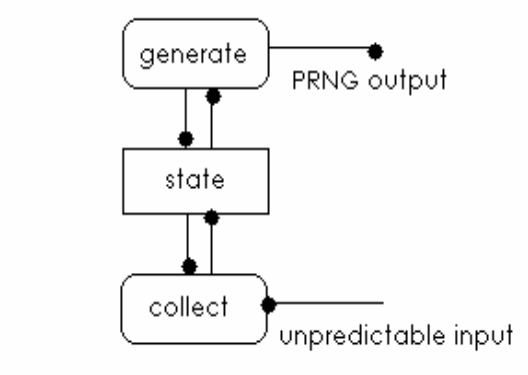
- Perancangan CSPRNG spesial

Terdapat beberapa Pseudo Random Number Generator yang secara khusus didesain agar aman secara kriptografi, dan ISAAC merupakan salah satu diantaranya.

4.2 Kelas-kelas Serangan Kriptografi

4.2.1 Direct Cryptanalytic Attack

Ketika seorang penyerang mampu secara langsung membedakan keluaran pembangkit bilangan acak semu dan keluaran acak, maka hal tersebut disebut Direct Cryptanalytic Attack. Jenis serangan ini dapat diterapkan pada sebagian besar pembangkit bilangan acak semu, meskipun tidak seluruhnya.



Gambar 2
Diagram operasi internal pada sebagian besar pembangkit bilangan acak semu

4.2.2 Input Based Attack

Serangan terhadap masukkan muncul ketika penyerang dapat menggunakan pengetahuan yang dimilikinya atau mengontrol masukkan dari pembangkit bilangan acak semu untuk

melakukan kriptanalisis terhadap pembangkit tersebut (termasuk membedakan keluaran dari PRNG dan nilai acak alami).

Input Based Attack dapat dibagi lebih jauh menjadi *known-input*, *replayed-input* dan *chosen-input* attack. Serangan chosen-input dapat diterapkan terhadap smart cards, aplikasi yang menerima pesan masuk, sandi lewat yang dipilih pengguna, statistik jaringan, dan sebagainya. Replayed-input juga dapat dipraktekan pada situasi-situasi tersebut, dan memerlukan kontrol yang lebih sedikit dari penyerang. Serangan known-input dapat digunakan dalam kondisi beberapa masukkan pembangkit bilangan acak semu, yang dianggap oleh pembuat sistem sukar ditebak, ternyata dapat diterka dalam beberapa kasus khusus.

4.2.3 State Compromise Extension Attacks

Serangan state compromise extension mencoba menggunakan keuntungan dari usaha yang telah dilakukan sebelumnya yang telah menghasilkan S sejauh mungkin. Diasumsikan bahwa, dengan alasan apapun – sebuah penetrasi temporer pada keamanan komputer, sebuah lubang keamanan, sebuah serangan kriptanalitik yang sukses – penyerang dapat mempelajari state internal dari S pada suatu titik tertentu. Serangan state compromise extension sukses jika penyerang dapat menemukan keluaran pembangkit bilangan acak semu yang belum diketahui (atau membedakan keluaran tersebut dari nilai acak alami) dari sebelum S diketahui, atau menemukan keluaran setelah pembangkit bilangan acak menerima barisan masukkan yang tidak diketahui penyerang. Serangan state compromise extension memiliki tingkat keberhasilan yang tinggi jika pembangkit bilangan acak semu dimulai dari state yang tidak aman karena entropy awal yang tidak sufisien.

Serangan ini juga dapat dilakukan jika S telah berhasil dideduksi oleh salah satu dari serangan-serangan berikut :
- Backtracking Attacks
Serangan ini menggunakan kompromi dari state S dari sebuah pembangkit bilangan acak semu pada suatu waktu t untuk mempelajari keluaran sebelumnya.
- Permanent Compromise Attacks
Serangan permanent compromise muncul jika ketika seorang penyerang berhasil menemukan state S pada waktu t, semua nilai S di masa lalu dan masa memiliki celah untuk diserang.
- Iterative Guessing Attacks
Serangan ini menggunakan pengetahuan dari S pada waktu t dan mempengaruhi keluaran dari pembangkit bilangan acak semu, untuk mempelajari S pada waktu t + ε, ketika masukkan selama rentang waktu ini dapat ditebak (meskipun tidak diketahui) oleh penyerang.
- Meet in the Middle Attacks
Serangan meet in the middle adalah kombinasi dari iterative guesseing attack dengan backtracking attack. Pengetahuan akan S pada waktu t dan t + 2ε memungkinkan penyerang menemukan S pada waktu t + ε.

Untuk menjaga ketahanan sistem, pembangkit bilangan acak semu harus mampu bertahan terhadap serangan state compromise extension sebaik mungkin.

4.3 Dalil-dalil yang Digunakan pada Deskripsi Serangan

Catatan:
z1, z2,…, zm+1 adalah barisan keluaran dari pembangkit bilangan acak semu, dan diasumsikan nilai-nilai ini telah diketahui.
at=(at,K-1, at,K-2,…,at,i,…, at,1, at,0) adalah representasi biner dari at$\in \boldsymbol{ZK}$, {at,j}$\in \boldsymbol{Z2}$
Simbol ≡ ekivalen dengan modulo 2^{32}.

4.3.1 Dalil 1

F<< (a,p)=((a<<p) ⊕ a)= (aK-1⊕ aK-p-1, aK-2⊕ aK-p-2,…, ap+i ⊕ai,…, ap⊕ a0, ap-1 , ap-2,…, a1, a0).

F>> (a,p)=((a>>p) ⊕ a)=(aK-1 , aK-2,…, ap, aK-1⊕ aK-p-1, aK-2⊕ aK-p-2,…, ap+i ⊕ai,…, ap⊕ a0).

4.3.2 Dalil 2

Jika diketahui sm[0] (mod 2^{β}), s1[1] (mod 2^{β}),…, sm-1[m-1] (mod 2^{β}) dan z1, z2,…, zm+1,

maka s0[0] (mod 2^{β}), s0[1] (mod 2^{β}),…, s0[m-1] (mod 2^{β}), dapat ditentukan untuk t=m, m-1, m-2,…, 2, 1 as sebagai berikut:

If jt=0, m-1,…,t+1, then
s0[t] (mod 2^{β}) = (zt (mod 2^{β}) – s0[jt] (mod 2^{β})) (mod 2^{β}).

If 0<jt<t+1, then

$s0[t] \pmod{2^{\beta}} = (zt \pmod{2^{\beta}} - sjt[jt] \pmod{2^{\beta}}) \pmod{2^{\beta}}$.

4.3.3 Dalil 3

Jika diketahui $sm[0] \pmod{2^{\beta}}$, $s1[1] \pmod{2^{\beta}}$,…,$sm-1[m-1] \pmod{2^{\beta}}$ and z1, z2,…, zm+1,

maka $a1 \pmod{2^{\beta}}$, $a2 \pmod{2^{\beta}}$,…., $am+1 \pmod{2^{\beta}}$, dapat ditentukan sebagai berikut:

If jt>t, then
$at \pmod{2^{\beta}} = (st[t] \pmod{2^{\beta}} - s0[\alpha t] \pmod{2^{\beta}} - zt-1 \pmod{2^{\beta}}) \pmod{2^{\beta}}$.

If t≥jt, then
$at \pmod{2^{\beta}} = (st[t] \pmod{2^{\beta}} - s\alpha t[\alpha t] \pmod{2^{\beta}} - zt-1 \pmod{2^{\beta}}) \pmod{2^{\beta}}$.

dengan t=1…m+1.

4.3.4 Dalil 4

Diberikan τ≥2n+θ2.

Jika diketahui $a1 \pmod{2^{\tau}}$, $a2 \pmod{2^{\tau}}$,…, $ai \pmod{2^{\tau}}$,…,$am \pmod{2^{\tau}}$ è $s2[2+m/2] \pmod{2^{\tau}}$, $s4[4+m/2] \pmod{2^{\tau}}$,…., $s2i[(m/2+2i) \pmod m] \pmod{2^{\tau}}$,…,$sm[m] \pmod{2^{\tau}}$,

maka $a1 \pmod{2^{\tau+q}}$, $a3 \pmod{2^{\tau+q}}$,…, $a2i+1 \pmod{2^{\tau+q}}$,…. $am-1 \pmod{2^{\tau+q}}$ dapat ditentukan sebagai berikut:

$at,q-1+\tau = bt+1,\tau-1 \oplus at,\tau-1$,

$at,q+\tau-j = bt+1,\tau-j \oplus at,\tau-j$,

$at,\tau = bt+1,\tau-q \oplus at,\tau-q$.

di mana t=1 (mod 2), $bt+1 = (at+1 \pmod{2^{\tau}} - st+1[t+1+m/2] \pmod{2^{\tau}}) \pmod{2^{\tau}}$.

4.3.5 Dalil 5

If j<p(t), then
$a2i+1,j = a2i,j \oplus s\delta(2i+1,2i+1+m/2),j[2i+1+m/2] \oplus \sigma 2i^{a}+1(j)$.

If j≥ p(t), then
$a2i+1,j = a2i,j \oplus a2i,j-p(2i+1) \oplus s\delta(2i+1,2i+1+m/2)$, $j[2i+1+m/2] \oplus \sigma 2i^{a}+1(j)$.

4.3.6 Dalil 6

Jika diketahui $st[t] \pmod{2^{j-1}}$, $s0[t] \pmod{2^{j-1}}$, αt, $jt \pmod{2^{j-1}}$, $zt-1 \pmod{2^{j-1}}$, $at-1 \pmod{2^{j-1}}$, maka $\sigma t^{s}(j)$, $\sigma t^{z}(j)$ and $\sigma 2i^{a}+1(j)$ dapat ditentukan sebagai berikut:

$\sigma t^{s}(j) =$

1 if $(s\delta(t, t\ á\ )[\alpha t] \pmod{2^{j-1}} + zt-1 \pmod{2^{j-1}} + at,j-1 \pmod{2^{j-1}}) \pmod{2^{j+1}} \geq 2^{j}$,

0 otherwise.

$\sigma t^{z}(j) =$

1 if $(s\delta(t, t\ j\ ),j-1[jt] \pmod{2^{j-1}} + s0,j - 1[t] \pmod{2^{j-1}}) \pmod{2^{j+1}} \geq 2^{j}$,

0 otherwise.

If j<p(t) and t=1 (mod 2), then

$\sigma 2i^{a}+1(j) =$

1 if $(a2i \pmod{2^{j-1}} + s2i+1[2i+1+m/2] \pmod{2^{j-1}}) \pmod{2^{j+1}} \geq 2^{j}$,

0 otherwise.

If j≥ p(t) and t=1 (mod 2), then
$\sigma ta(j) =$

1, if for k=p(t)..j-1 $(at-1,k \oplus at-1-p(t),k) . 2^{k} + at-1 \pmod{2^{p(t)-1}} + s\delta(t,t+m/2), j[t+m/2] \pmod{2^{j-1}}) \pmod{2^{j+1}} \geq 2^{j}$

0, otherwise.

4.3.7 Teorema 1

Jika diketahui αt, jt, $a2i+1 \pmod{2^{j}}$, $a2i \pmod{2^{j-1}}$, $\sigma t^{s}(j)$, $\sigma t^{a}(j)$, $\sigma t^{z}(j)$, zt,j, t=1,…,m, i=0…m/2-1 dan j≥max(p(1), p(3)),

maka st,j[t], s0,j[t], t=1,…,m, dapat ditentukan dengan menyelesaikan persamaan berikut:

$s0,j[0] \oplus s\delta(m,\ m\ j\ ),j[jm] = zm,j \oplus \sigma m^{z}(j)$,
$s0,j[1] \oplus s\delta(1,\ 1\ j\ ),j[j1] = z1,j \oplus \sigma 1^{z}(j)$,

$s0,j[t] \oplus s\delta(t,tj\ ),j[jt] = zt,j \oplus \sigma t^{z}(j)$,

$s0,j[1+m/2] = a1,j \oplus \sigma 1^{a}(j)$,

$s0,j[t] \oplus s\delta(t,tj\ ),j[jt] = zt,j \oplus \sigma t^{z}(j)$,

.....

$s0,j[m-1] \oplus s\delta(m-1,\ 1\ -\ m\ \ j\ ),j[jm-1] = zm-1,j \oplus \sigma m^{z}\text{-}1(j)$,

$s1,j[1] \oplus s\delta(1, \alpha\ 1\ ),j[\alpha 1] = a1,j \oplus \sigma 1^{s}(j)$,

$s2,j[2] \oplus s\delta(2, \alpha\ 2\ ),j[\alpha 2] \oplus s0,j[3+m/2] = z1,j \oplus a3,j \oplus a2,j - p(3) \oplus \sigma 3^{a}(j) \oplus \sigma 2^{s}(j)$,

$s3,j[3] \oplus s\delta(3, \alpha 3\ ),j[\alpha 3] = z2,j \oplus a3,j \oplus \sigma 3^{s}(j)$,

.....

$s2i,j[2i] \oplus s\delta(2i+1,2i+1+m/2),\ j[2i+1+m/2] \oplus s\delta(2i, \alpha\ i\ 2\ ),j[\alpha 2i] = z2i\text{-}1,j \oplus a2i+1,j \oplus a2i,j - p(2i+1) \oplus \sigma 2i^{a}+1(j) \oplus \sigma 2i^{s}(j)$,

$s2i+1,j[2i+1] \oplus s\delta(2i+1,\ \alpha\ 1\ i\ 2\ +\ ),j[\alpha 2i+1] = z2i,j \oplus a2i+1,j \oplus \sigma 2i^{s}+1(j)$,

.....

$sm\text{-}2,j[m-2] \oplus s\delta(m-2,\ \alpha\ 2\ -\ m\ ),j[\alpha m-2] = zm-3,j \oplus am-2,j \oplus \sigma m^{s}\text{-}2(j)$,

$sm\text{-}1,j[m-1] \oplus s\delta(m-1,m/2\ -1),j[m/2-1] \oplus s\delta(m-1,\ \alpha\ 1\ \ m\ -\ ),j[\alpha m-1] = zm,j \oplus am-1,j \oplus am-2,j - p(3) \oplus \sigma m^{a}\text{-}1(j) \oplus \sigma m^{s}\text{-}1(j)$,

$sm,j[m] \oplus s\delta(m,\ \alpha\ m\ ),j[\alpha m] = zm-1,j \oplus am,j \oplus \sigma m^{s}(j)$.

.....

$a2i,j$, $i=0\ldots m/2$ dapat ditentukan sebagai berikut:
$a2i,j = a2i+1,j \oplus a2i,j - p(2i+1) \oplus s\delta(2i+1,2i+1+m/2),j[2i+1+m/2] \oplus \sigma 2i^{a}+1(j)$.

4.3.8 Teorema 2

Jika D dan D' adalah distribusi, dan anggap event E terjadi dalam D dengan probabilitas p dan dalam D' dengan probabilitas p(1+q),
maka untuk p dan q yang kecil, $O(1/pq^2)$ dapat membedakan D dari D' dengan probabilitas sukses yang konstan.

Pembuktian dari dalil-dalil dan teorema di atas di luar lingkup bahasan makalah ini.

4.4 Serangan State Recovery pada ISAAC

Unicity distance adalah jumlah symbol barisan nilai yang perlu diobservasi dalam serangan sebelum initial state dapat diketahui.
Catatan :

jumlah state di ISAAC = $m.2^{K}.2^{Km}$ D_{ISAAC},
maka diperoleh $(2^{K})^{DISAAC} = m.2^{K}.2^{Km}$,
jadi nilai $D_{ISAAC} = m + 2$

Metode kriptanalisis yang dilakukan terdiri dari empat langkah:

- Langkah 1

Menebak $sm[0] \pmod{2^{2n+\theta 2}}$,…, $st[t] \pmod{2^{2n+\theta 2}}$,…, $sm\text{-}1[m-1] \pmod{2^{2n+\theta 2}}$.

- Langkah 2

Misalkan $\beta = 2n + \theta 2$

1. Gunakan dalil 2 untuk menghitung $s0[t]$, $t=0,1,\ldots,m-1$.
2. Gunakan dalil 3 untuk menghitung $at \pmod{2^{2n+\theta 2}}$, $t=1\ldots m+1$.
3. Misalkan $\tau = \beta$. Gunakan dalil 4 untuk menghitung $a2j+1 \pmod{2^{\tau+q}}$, $j = 0\ldots m/2-1$.
4. Untuk mendapatkan $sm[0] \pmod{2^{\tau+q}}$, $s1[1] \pmod{2^{\tau+q}}$,…, $sm\text{-}1[m-1] \pmod{2^{\tau+q}}$, $s0[0] \pmod{2^{\tau+q}}$, $s0[1] \pmod{2^{\tau+q}}$,…,$s0[m-1] \pmod{2^{\tau+q}}$, $a2i \pmod{2^{\tau+q}}$, $i=0\ldots m/2$, dilakukan:

   a) $j = \tau + 1$;
   b) while $j \le \tau+q$ do
   Gunakan dalil 1 untuk menghitung $sm[0] \pmod{2^{j}}$,…,$sm\text{-}1[m-1] \pmod{2^{j}}$, $s0[0] \pmod{2^{j}}$,…,$s0[m-1] \pmod{2^{j}}$, $a2i \pmod{2^{j}}$, $i=0\ldots m/2$,

   $j = j + 1$;

- Langkah 3

Misalkan $\tau = 2n + \theta 2 + q$

while $\tau < K$ do

1. Gunakan dalil 4 untuk menghitung $a2j+1 \pmod{2^{\tau+q}}$, $j=0\ldots m/2-1$.
2. Untuk menemukan $sm[0] \pmod{2^{\tau+q}}$, $s1[1] \pmod{2^{\tau+q}}$,…, $sm\text{-}1[m-1] \pmod{2^{\tau+q}}$, $s0[0] \pmod{2^{\tau+q}}$,$s0[1] \pmod{2^{\tau+q}}$,…, $s0[m-1] \pmod{2^{\tau+q}}$, $a2i \pmod{2^{\tau+q}}$, $i=0\ldots m/2$, dilakukan :
   a) $j=\tau+1$.
   b) while $j \le \tau+q$ do.
   gunakan teorema 1 untuk menghitung $sm[0] \pmod{2^{j}}$,…,$sm\text{-}1[m-1] \pmod{2^{j}}$, $s0[0] \pmod{2^{j}}$,…, $s0[m-1] \pmod{2^{j}}$, $a2i \pmod{2^{j}}$, $i=0\ldots m/2$.

   $j=j+1$.

- Langkah 4
Hitung L = D_{ISAAC} yang pertama dari barisan element keluaran z1*, z2*,… zL*.

Jika z1*= z1, z2*=z2,…zL*= zL maka telah ditemukan initial state yang tepat dari system, jika tidak maka ulangi dari langkah 1.

Kompleksitas dari metode ini dapat dihitung sebagai berikut:
Diasumsikan probabilitas
P{s0*[0] (mod $2^{(2n+\theta 2)}$)= s0[0] (mod $2^{(2n+\theta 2)}$),…,s0*[m-1] (mod $2^{(2n+\theta 2)}$)= s0[m-1] (mod $2^{(2n+\theta 2)}$)} =1/ $2^{(2n+\theta 2)m}$.

Maka rata-rata element tebakan sama dengan $2^{(2n+\theta 2)m-1}$. Kompleksitas dari solusi system persamaan pada langkah 2 dan 3 ialah (K-2n-θ2)·(2m)/3.

Maka dari itu, kompleksitas metode ini sama dengan Tmet=$2^{(2n+\theta 2)m-1}$·(K-2n-θ2)·(2m)/3. Perlu dicatat bahwa kompleksitas dari serangan brute force ialah Tbr=$^{2K\cdot m-1}$.

Untuk m=256, n=8, K=32, p0=13, p1=6, p2=2, p3=16, θ1=θ2=2, diperoleh Tmet =4.67·101240, dan Tbr = 5.91 102446 .

4.5 Serangan Distinguishing pada ISAAC

4.5.1 State yang Lemah

Pada dasarnya, state yang lemah terdiri dari sebagian element acak, dan sebagian lagi element yang nilainya tetap. State-state tersebut dibagi menjadi empat set yang disjoint : W1, W2, W3, W4. Bagian ini mendefinisikan tiap set tersebut dan memaparkan bias yang terkandung dalam elementnya, serta beberapa komentar penting lainnya. Notasi α digunakan untuk initial state, sedangkan ω untuk array pertama yang dikeluarkan algoritma.

- Set W1
Definisi : α ϵ W1 ←→ α0 = α1

Bias : Untuk nilai acak α ϵ W1,

$$\Pr[\omega 0 = \omega 1] \geq 254/256^2$$

Tentu saja untuk state W1, ω0 = ω1 memegang element yang sama jika index lebih besar dari 2 dipilih pada ronde pertama dan kedua (ronde pertama memiliki index ≥ 2 dengan probabilitas 254/256, lalu dipilih lagi dengan probabilitas kondisional 1/256).

State seperti ini akan ditemukan ketika membangun distinguisher D1.
Terdapat $2^{32\cdot 254} \cdot 2^{32} = 2^{8160}$ state di W1.

- Set W2
Definisi : α ϵ W2 ←→ ɜ N ϵ {2, . . . , 256}, ɜ X ϵ {0, . . . , 232 − 1}, $\alpha 0 = X, \#\{0 < i < 256, \alpha i = X\} = N - 1$.

Bias : Untuk nilai acak α ϵ W2,

$$\Pr[\omega 0 = 2X] \geq N - 1 / 256$$

Tentu saja pada ronde pertama dari algoritma ini sebuah nilai acak v dari state dipilih, yang mungkin saja X dengan probabilitas $(N - 1)/256$, lalu nilai $\omega 0 = \alpha 0 + v$ dikembalikan.

Sebuah bias yang sering muncul secara statistik muncul pada pendistribusian 32 bit pertama. Sebagai contoh, jika N adalah set sampai 6, $\Pr[\omega 0 \equiv 2X] \approx 0.02$, dan terdapat 2^{8033} state di $W2$ dengan $N = 5$.
Terdapat lebih dari $255 \cdot 2^{32254} \cdot 2^{32} \geq 2^{8167.99}$ state di $W2$.

- Set W3

Definisi : $\alpha \in W3 \leftrightarrow \exists N \in \{2, \ldots, 256\}, \exists X \in \{0, \ldots, 2^{32} - 1\}$, (for all)$i \in \{0, \ldots, N-1\}, \alpha i = X$.

Bias : Untuk nilai acak $\alpha \in W3$,

$$\Pr[\omega i \equiv 2X] \geq (N - 1 - i)/256,$$
dengan $i = 0, \ldots, N-1$

$x = X$ adalah benar, begitu juga $\alpha_{\alpha i \gg 10 \bmod 256}$ adalah sama dengan X jika $\alpha i \gg 10 \bmod 256$ lebih besar dari i dan harus lebih kecil dari N, yang muncul dengan probabilitas lebih besar dari $(N - 1 - i)/256$.

Jelas terlihat bahwa $W3$ merupakan himpunan bagian dari $W2$. Sekali lagi nilai $2X$ akan muncul dengan probabilitas yang tinggi, dibandingkan dengan aliran bit acak, namun tidak hanya di $\omega 0$.

Sebagai contoh, jika $N = 64$ dan $X = 0 : 192$ element terakhir dari α adalah acak, dan 64 yang pertama diset menjadi 0, maka $\Pr[\omega 0 = \omega 1 = 0]$

$\approx 0.06 \approx 2^{-4}$. Jika N sama kecil dengan 2, $\Pr[\omega 0 \equiv 2X] \approx 2^{-8}$, jauh lebih tinggi dari 2^{-32} yang merupakan pembangkit ideal.
Jika N lebih besar dari 216, maka $2X$ muncul rata-rata lebih dari 90 kali, sehingga X ditemukan dengan probabilitas yang tinggi dan element acak sisanya dapat dihitung dengan metode pencarian exhaustive search dalam 2^{48}.
Terdapat lebih dari $2^{32 \cdot 254} \cdot 2^{32} = 2^{8160}$ state di $W3$.

- Set W4

Definisi : $\alpha \in W4 \leftrightarrow \exists X \in \{0, \ldots, 2^{32} - 1\}$, (for all)$i \in \{0, \ldots, 255\}, \alpha i = X$.

Bias : Untuk nilai acak $\alpha \in W4$,

$$\Pr[\omega i \equiv 2X] \geq= 1 - (i + 1)/256 .$$

Hasil ini muncul sebagai kasus tertentu pada state W1. Lebih jauh lagi, nilai i yang diharapkan sehingga $\omega i \equiv 2X$ lebih besar dari

$$\sum_{i=0}^{255} (1 - (i + 1)/256) = 127.5,$$

yaitu, lebih dari setengah element yang dihasilkan pada ronde pertama adalah rata-rata $\equiv 2X$, jika $\alpha i = X$ untuk $i = 0, \ldots, 255$.

Cukup mudah membedakan bit acak asli dan bit acak yang dihasilkan oleh algoritma ISAAC yang diinisialisasi dengan state berisi nilai konstan, karena akan memiliki kurang lebih stengah ωi yang sama dengan $2X$. Seluruh state bahkan dapat ditemukan dalam beberapa detik hanya dengan bantuan kertas dan pena.
Terdapat tepat 232 state di $W4$.

4.5.2 Distinguisher D1 dan D2

Distinguisher yang kuat adalah: ”Algoritma yang dibatasi probabilistik polinomial, memiliki dua kotak hitam, masing-masing mengembalikan sampel bit dengan panjang yang tetap; untuk satu kotak sampel ini adalah benar-benar acak sedangkan pada kotak lain dihasilkan oleh pembangkit bilangan acak semu dengan initial state yang acak dan tidak diketahui. Algoritma ini mengembalikan 0 atau 1 untuk menentukan kotak mana yang dianggap sebagai pembangkit bilangan acak semu.”

Dipaparkan dua buah distinguisher D1 dan D2 dan evaluasi jumlah sampel yang diperlukan sesuai dengan teorema 2.

- Distinguisher D1

Untuk sebuah state acak dalam W1, $\Pr[\omega 0 = \omega 1] \approx 2^{-8}$, sehingga untuk sebuah state acak dari ISAAC

$$\Pr[\omega 0 = \omega 1] \geq 2^{-32}(2^{-8} + 2^{-32}) + (1 - 2^{-32})2^{-32} = 2^{-32} + 2^{-40},$$

di mana probabilitas ini adalah untuk 2^{-32} aliran bit yang benar-benar acak.

Di sini kotak-kotak akan memberikan keluaran 64-bit untuk setiap query, dan algoritmanya akan memilih sebagai kotak ISAAC salah satu yang 32 bit pertamanya paling sama secara frekuensi dengan 32 bit terakhir (yaitu waktu $\omega 0 = \omega 1$ dalam ISAAC), dan kotak acak jika terdapat kesamaan kemunculan. Menggunakan teorema 2 diperoleh $p = 2^{-32}$ dan $q = 2^{-8}$ sehingga distinguisher memerlukan sekitar 2^{48} sampel untuk mendapatkan keuntungan signifikan. Paling banyak 112 bit memori diperlukan (64 untuk membaca keluaran kotak hitam dan 48 untuk menghitung kemunculan).

- Distinguisher D2

Untuk sebuah state acak dalam W2 dengan N = 2 ($\alpha 0 = \alpha i$ for some $i > 0$), $\Pr[\omega 0 = 2X] \geq 2^{-8}$. Karena 2X adalah genap, least significant bit dari $\omega 0$ adalah 0 dengan probabilitas $½ + 2^{-8}$. Jika ζ adalah bit ini, maka untuk state acak ISAAC diperoleh

$$\Pr[\zeta = 0] \geq (1 - 2^{-25}) ½ + 2^{-25}(½ + 2^{-8}) = ½ + 2^{-33}$$

karena state acak di W2 dengan probabilitas 2^{-25}. Algoritma distinguisher harus menampung 32-bit sampel dan memilih sebagai kotak ISAAC sampel yang bit ke-32 nya paling sering bernilai 0. Dengan teorema 2 diperoleh $p = ½$ dan $q = 2^{-32}$, sehingga 2^{64} buah 32-bit sampel diperlukan oleh algoritma ini, sehingga berjalan dalam waktu sekitar 2^{64} dan memerlukan paling banyak 96 bit memori.

4.5.3 ISAAC+

Untuk memperbaiki masalah-masalah di atas, algoritma ISAAC dapat dimodifikasi. Perubahan yang dilakukan meliputi : penambahan $\oplus$ a

untuk menghindari bias yang diobservasi, melakukan rotasi untuk menggantikan pergeseran sehingga diperoleh difusi yang lebih baik dari bit state, serta mengganti addition dengan XOR untuk mengurangi kelinearan Z_2^{32}.

Implementasi algoritma ISAAC yang telah dimodifikasi dalam pseudocode

---

Input: *a*, *b*, *c*, dan state internal *s*, sebuah array berukuran 256 bertype 32-bit integer
Output: sebuah array *r* berukuran 256 bertype 32-bit integer

---

$c \leftarrow c + 1$
$b \leftarrow b + c$

for $i = 0, \ldots, 255$ do
  $x \leftarrow si$
  $a \leftarrow f^{i}(a, i) + s_{i+128 \bmod 256}$
  $si \leftarrow a \oplus b + s_{x>>>2 \bmod 256}$
  $ri \leftarrow x + a \oplus s_{si>>>10 \bmod 256}$
  $b \leftarrow ri$
end for
return *r*

ISAAC+ ini memiliki sifat sebagai berikut:

- ISAAC+ memiliki kompleksitas algoritma yang hamper sama.
- ISAAC+ berhasil melewati seluruh pengujian statistic Diehard dan NIST (hal ini menjamin kualitas statistikal minimal dari aliran bit acak semu).
- State-state lemah yang disebutkan di atas kehilangan bias yang tidak diharapkannya. Sebagai konsekuensinya, distinguisher D1 dan D2 menjadi tidak dapat diterapkan.

5. Kesimpulan

Beberapa pembangkit bilangan acak semu dibuat berurutan : IA, IBAA, dan ISAAC yang merupakan pengembangan terakhir dari dua pendahulunya. Algoritma ISAAC ini secara garis besar menggunakan array 32-bit integer berukuran 256 sebagai state internalnya, dan menuliskan hasilnya pada array lain yang sejenis, di mana dari array lain tersebut hasil yang diperoleh dibaca satu per satu untuk dikomputasi ulang. Kecepatan dan sedikitnya bias dalam algoritma-algoritma tersebut membuat algoritma-algoritma yang dibuat berguna untuk simulasi dan kriptografi. Algoritma ISAAC juga digunakan secara luas dalam kehidupan sehari-hari, seperti di kasino-kasino di mana mesin permainan memerlukan nilai-nilai acak dalam pengoperasiannya.

Serangan terhadap suatu pembangkit bilangan acak semu merupakan masalah umum yang perlu mendapat perhatian lebih. Untuk membuat suatu pembangkit bilangan acak yang baik, dapat dicantumkan beberapa syarat tambahan di samping syarat-syarat yang sudah disebutkan di bagian awal makalah ini. Pembangkit bilangan acak tersebut harus didasari oleh primitive kriptografi yang kuat, dan memiliki probabilitas sangat kecil untuk dipecahkan dengan serangan direct cryptanalysis. Selain itu seluruh state dari pembangkit bilangan acak tersebut juga harus berganti sepanjang waktu. Hal ini mencegah kompromi jika suatu state tertentu berhasil diketahui. Pembangkit bilangan acak semu sebaiknya tahan terhadap serangan backtracking, di mana keluaran *t* tidak dapat ditebak oleh penyerang jika ia berhasil mengetahui state ke *t* + 1. Masukkan untuk pembangkit bilangan acak semu harus dikombinasikan ke dalam state pembangkit bilangan ini sedemikian sehingga jika terdapat barisan masukkan yang tidak dapat ditebak, seorang penyerang yang mengetahui state-state dari pembangkit bilangan acak semu tersebut namun tidak mengetahui barisan masukkan tersebut tidak dapat menerka state akhir yang didapat (tahan terhadap serangan chosen-input). Syarat terakhir adalah pembangkit bilangan acak semu yang dibuat harus menggunakan setiap bit entropy masukkan yang diterimanya, sehingga seorang penyerang harus mengetahui keseluruhan masukkan jika ingin mempelajari state-state dari pembangkit bilangan acak semu terssebut.

Terdapat beberapa metode kriptanalisis terhadap algoritma ini, seperti serangan state recovery pada algoritma ISAAC, dengan mencoba menurunkan initial state dari algoritma ini. Namun karena kompleksitas algoritma serangan yang terlalu besar ISAAC tetap dianggap sebagi pembangkit nilai acak yang aman untuk aplikasi sehari-hari. Selain itu ISAAC juga rentan terhadap serangan distinguishing, di mana bit-bit yang dihasilkan oleh state acak dari ISAAC dapat dibedakan dari bit acak asli jika diberikan waktu yang cukup karena banyaknya set state-state lemah yang terdapat dalam algoritma ini. Serangan ini juga relevan terhadap IA dan IBAA

karena algoritmanya yang mirip dengan ISAAC. Metode yang ditawarkan sebagai pengembangan algoritma ISAAC ini juga tidak dapat menjamin keamanan yang lebih baik, sehingga perlu dilakukan studi lebih lanjut untuk membuat pembangkit bilangan acak semu dengan algoritma ISAAC yang benar-benar aman dan tidak memiliki celah.

Referensi

R. Munir, *Diktat Kuliah IF5054 Kriptografi*, Program Studi Teknik Informatika Institut Teknologi Bandung, 2006.

Robert J. Jenkins, http://www.burtleburtle.net/bob/rand/isaacafa.html.

R.J. Jenkins, "ISAAC", Fast Software Encryption –Cambridge 1996, vol. 1039, D. Gollmann ed., Springer-Verlag.

G. Marsaglia, *A New Class of Random Number Generators*. The Annals of Applied Probability, 1991.

Oded Goldreich, *Foundations of Cryptography, volume 1*, Cambridge University Press, 2001.

Georges Marsaglia, *The Diehard Battery of Tests of Randomness*, 1995.

National Institue of Standards and Technology. *Statistical Test Suite 1.8*, 2005.

Marina Pudovkina, *A Cycle Structure of the Alleged RC4 Keystream Generator*, Journal of Security of information technologies, Moscow, 4, 2000.

R.W. Baldwin, *Proper Initialization for the BSAFE Random Number Generator*, *RSA Laboratories Bulletin*, n. 3, 25 Jan 1996.

M. Santha and U.V. Vazirani, *Generating Quasi-Random Sequences from Slightly Random Sources*, *Journal of Computer and System Sciences,* v. 33, 198 6.

P. Gutmann, *Software Generation of Random Numbers for Cryptographic Purposes*, Proceedings of the 1998 Usenix Security Symposium, 1998.

D. Eastlake, S.D. Crocker, and J.I. Schiller, *Randomness Requirements for Security*, RFC 1750, Internet Engineering Task Force, Dec. 1994.

B. Schneier, *Applied Cryptrography,* John Wiley & Sons, 1996.

Jean-Philippe Aumasson, *Distinguishing attacks on ISAAC,* Cryptology ePrint archive, report 2006/438, 2006.

Souradyuti Paul, Bart Preneel, *(In)security of Stream Ciphers Based on Arrays and Modular Addition*.Asiacrypt 2006.

D. Knuth, *Seminumerical Methods, volume 2*. Addison Wesley, 1981.

S. Lloyd, *Counting Binary Functions with Certain Cryptographic Properties*. Journal of Cryptology, 1992.

Varfolomeev A.A., Zhukov A.E., Pudovkina M., *Analysis of Stream Ciphers*, Moscow, 2000.

Itsik Mantin and Adi Shamir, *A Practical Attack on Broadcast RC4*, Springer, 2001.

M. Blum and S. Micali, *How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits*. SIAM J. Comput, 1984.