Pembangkit Aliran Kunci Acak Semu – IKG

Robbi Kurniawan - 13504015
Jurusan Teknik Informatika ITB, Bandung
email: if14015@students.if.itb.ac.id, robbi.kurniawan@yahoo.com

Abstract **–** *One Time Pad* merupakan sebuah algoritma kriptografi yang sempurna apabila kunci yang digunakan acak sempurna dan panjang kunci sama dengan panjang teks. Akan tetapi, kunci acak sempurna tidak mungkin dibangkitkan secara komputasi sehingga kunci dibangkitkan dengan algoritma pembangkit kunci acak semu. Letak kekuatan algoritma pembangkit kunci acak semu adalah pada keacakan kunci yang dibangkitkan. Pada makalah ini, sebuah algoritma pembangkit kunci acak semu yang diberi nama *IKG* dirancang, diimplementasikan dalam bentuk aplikasi, dan diuji kekuatannya. *IKG* dirancang dengan menggunakan konsep dasar algoritma kriptografi simetrik *IDEA*. Setelah diuji, kunci yang dibangkitkan *IKG* dianggap sangat acak sehingga aman untuk digunakan sebagai pembangkit kunci algoritma kriptografi simetrik *One Time Pad*.

Kata Kunci*: key stream generator, IDEA, One time pad, acak semu.*

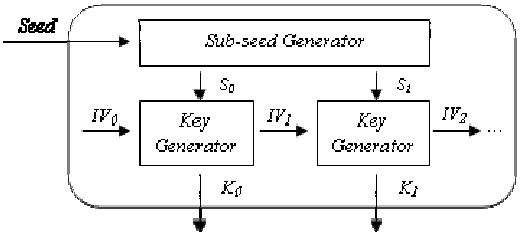
1. PENDAHULUAN

Letak kekuatan algoritma kriptografi simetrik *One time pad (OTP)* adalah pada keacakan dan panjang kunci yang dipakai [1]. Semakin acak dan panjang kunci, maka *OTP* akan menjadi semakin sempurna [1]. Oleh karena itu, perlu dikembangkan sebuah algoritma pembangkitan kunci acak semu yang dapat menghasilkan aliran kunci yang seacak dan sepanjang mungkin. Banyak cara yang telah dikembangkan dalam pembangkitan kunci acak semu, salah satunya adalah dengan memanfaatkan algoritma kriptografi baik simetrik. Salah satu algoritma kriptografi simetrik yang tergolong baru dan unik adalah *IDEA*. Algoritma ini unik karena tidak menggunakan *s-box* dan jaringan *feistel* sebagaimana algoritma simetrik lainnya sehingga dianggap lebih tahan terhadap serangan dan lebih transparan [2]. Pada makalah ini, akan dirancang dan diimplementasikan salah satu algoritma alternatif pembangkitan kunci acak semu dengan berbasiskan algoritma *IDEA* yang bernama *IKG*.

2. ANALISIS DAN PERANCANGAN

Sebagaimana algoritma pembangkitan kunci acak semu lainnya, *IKG* mendapat masukan berupa sebuah umpan (*seed*) dan menghasilkan sebuah kunci acak semu. Dalam *IKG*, umpan yang dimasukkan memiliki panjang 128-bit dan kunci acak semu yang dihasilkan memiliki panjang 8-bit.
Apabila kunci acak semu yang dibutuhkan lebih dari 8-bit ($n \times 8$ bit), *IKG* dapat dijalankan sebanyak *n* kali dengan hanya memasukkan umpan satu kali yaitu pada saat sebelum pembangkitan kunci 8-bit pertama.

IKG terdiri dari dua buah sub-modul yaitu *sub-seed generator* dan *key generator*. Kedua buah sub-modul tersebut dihubungkan sehingga membentuk arsitektur. Gambar 1 menunjukkan arsitektur IKG.



Gambar 1: Arsitektur *IKG*

Keterangan simbol:

Seed	umpan awal (*input IKG*)
S_n	umpan turunan ke-*n*
IV_n	*initialization vector* ke-*n*
K_n	kunci acak semu ke-*n* (*output IKG*)

Keterangan:
IV_0 bernilai `ffa31b6b65574ee0258a977ddfb3b36b` dalam Heksadesimal (nilai ini merupakan sebuah bilangan prima dengan panjang 64-bit).

2.1. *Key Generator*

Modul ini bertugas membangkitkan kunci acak semu (K_n) yang akan digunakan sebagai kunci pada algoritma *OTP*. Setiap kunci acak semu yang dibangkitkan memiliki panjang 8-bit. Modul ini membutuhkan masukan berupa *IV* atau *block-input (initialization vector)* sepanjang 64-bit dan kunci sepanjang 128-bit, dan menghasilkan keluaran berupa *IV* baru atau *block-output* (yang akan digunakan sebagai masukan modul ini berikutnya) dan kunci sepanjang 8-bit.

Strategi yang digunakan dalam pembangkitan kunci adalah dengan menggunakan algoritma dasar *IDEA* dengan melakukan modifikasi. Modifikasi dilakukan karena modul ini bersifat satu arah (*one-way*) dan menghasilkan dua buah keluaran (*block-output* dan kunci) sedangkan *IDEA* bersifat dua arah (*encryption-decryption*) dan hanya menghasilkan satu buah keluaran (*block-output*) [2]. Gambar 2 menunjukkan struktur *IDEA* sebelum modifikasi.

Pada struktur *IDEA*, terlihat bahwasannya *IDEA* memiliki 8+1 putaran dalam melakukan proses enkripsi/dekripsi dengan menggunakan 6 buah sub-kunci pada 8 putaran pertama dan 4 buah sub-kunci pada 1 putaran terakhir.

Modifikasi pada *IDEA* dilakukan dengan menambahkan prosedur pada setiap putaran *IDEA* dengan jumlah sub-kunci yang dibutuhkan menjadi 8 buah pada 8 putaran pertama dan 6 buah pada putaran terakhir.

Hasil modifikasi setiap putaran ke-i ($1 \le i \le 8$) IDEA (X_1, X_2, X_3, X_4) adalah sebagai berikut:

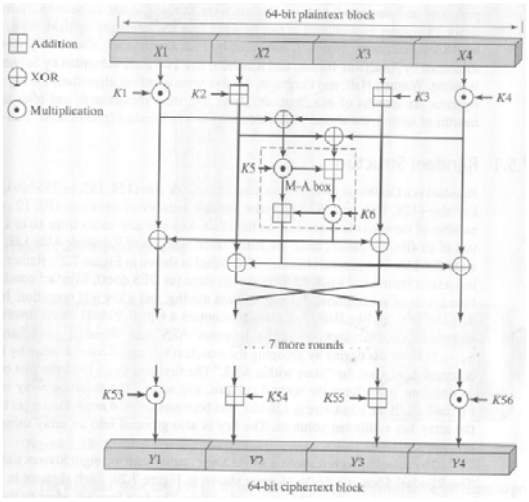
1. X_1 dikalikan dengan sub-kunci ke-1
2. X_2 dijumlahkan dengan sub-kunci ke-2
3. X_3 dijumlahkan dengan sub-kunci ke-3
4. X_4 dikalikan dengan sub-kunci ke-4
5. X_1 di-XOR-kan dengan X_3
6. X_2 di-XOR-kan dengan X_4
7. Hasil langkah 5 dikalikan dengan sub-kunci ke-5
8. Hasil langkah 6 dan 7 dijumlahkan
9. Hasil langkah 8 dikalikan dengan sub-kunci ke-6
10. Hasil langkah 7 dan 9 dijumlahkan
11. Hasil langkah 1 dan 9 di-XOR-kan
12. Hasil langkah 2 dan 10 di-XOR-kan
13. Hasil langkah 3 dan 9 di-XOR-kan
14. Hasil langkah 4 dan 10 di-XOR-kan
15. Hasil langkah 11 dan 12 di-XOR-kan
16. Hasil langkah 13 dan 14 di-XOR-kan
17. Hasil langkah 15 dijumlahkan dengan sub-kunci ke-7
18. Hasil langkah 16 dan 17 kalikan
19. Hasil langkah 18 dijumlahkan dengan sub-kunci ke-8
20. Hasil langkah 19 dan 17 dikalikan
21. Hasil langkah 11 dan 20 dijumlahkan
22. Hasil langkah 12 dan 20 dikalikan
23. Hasil langkah 13 dan 19 dijumlahkan
24. Hasil langkah 14 dan 19 dikalikan

Hasil langkah 20, 21, 22, 23 secara berurutan akan menjadi X4, X2, X3, X1 pada putaran selanjutnya (i+1). Pada putaran terakhir (ke-9), hasil modifikasi putaran adalah sebagai berikut:

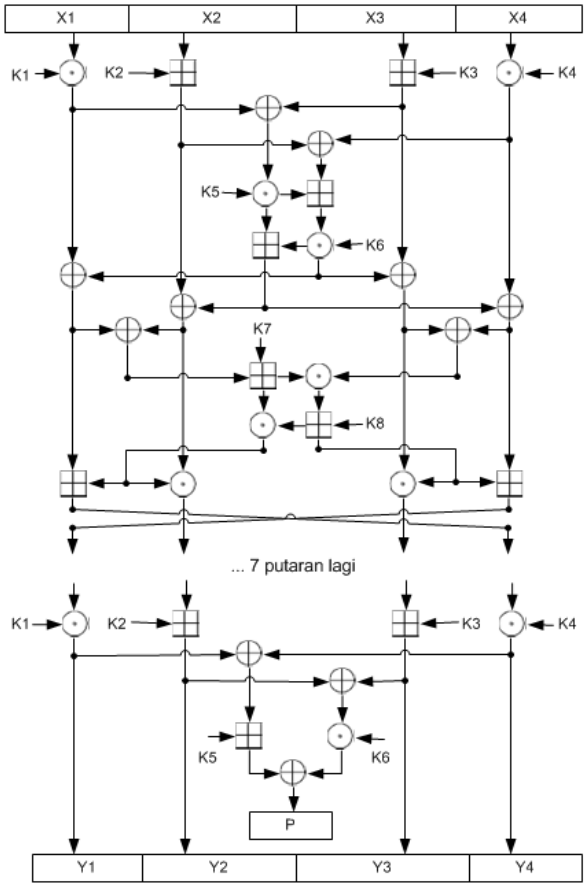
1. X_1 di-XOR-kan dengan X_4
2. X_2 di-XOR-kan dengan X_3
3. Hasil langkah 1 dijumlahkan dengan sub-kunci ke-5
4. Hasil langkah 2 dikalikan dengan sub-kunci ke-6
5. Hasil langkah 3 dan 4 di-XOR-kan

Bit ke-5 s.d. 12 dari hasil langkah 5 pada putaran terakhir akan menjadi nilai *P* (dengan panjang 8 bit). Dari seluruh putaran hasil modifikasi dapat dilihat bahwasannya algoritma ini membutuhkan (8×8) + 6 = 70 buah sub-kunci.

Pembangkitan sub-kunci dilakukan dengan cara seperti *IDEA* yaitu dengan menggeser memutar kunci awal (128 bit) sebanyak 25-bit ke kiri hingga mendapatkan 70 buah sub-kunci. Gambar 3 menggambarkan struktur *IDEA* hasil modifikasi.



Gambar 2: Struktur *IDEA* sebelum modifikasi



Gambar 3: Struktur *IDEA* hasil modifikasi

Keterangan simbol:

- ⊕ XOR 16-bit
- ⊞ Penjumlahan modulo 2^{16}
- ⊙ Perkalian modulo 2^{16}

Dengan struktur baru, dapat dilihat bahwasannya algoritma menjadi bersifat *non-reversible* dikarenakan adanya operasi penjumlahan dan pengalian (langkah 20 s.d. 24) pada putaran ke 1 s.d. 8.

Algoritma hasil modifikasi akan menghasilkan 8-bit nilai *P* dan 64-bit blok dengan masukan 64-bit blok dan 128-bit kunci. *P* pada modul ini akan menjadi K_n. Blok masukan merupakan *IV* dan blok keluaran merupakan *IV* baru.

2.2. *Sub-Seed Generator*

Modul ini bertugas membangkitkan umpan turunan (S_n) dari umpan awal yang menjadi masukan modul ini. Umpan turunan yang dibangkitkan akan menjadi masukan modul *Key Generator* berupa *key*. Setiap umpan turunan yang dihasilkan akan memiliki panjang 128-bit.

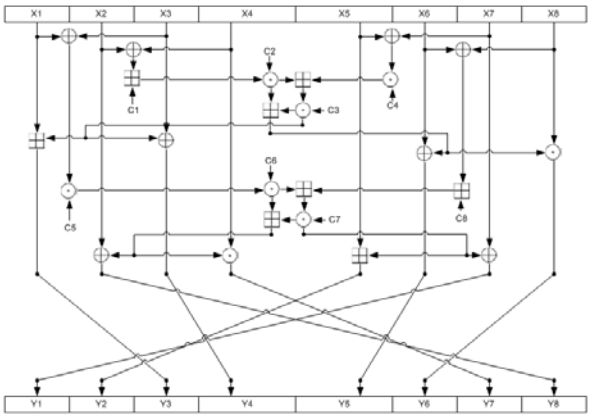
Strategi yang digunakan dalam pembangkitan umpan turunan (S_i dimana $0 \leq i$) adalah dengan menggunakan algoritma:

1. Umpan awal akan menjadi umpan turunan awal (S_{-1}).
2. Umpan turunan berikutya (S_{i+1}) didapat dengan menggeser memutar umpan turunan saat ini (S_i) sebanyak 37-bit ke kiri kemudian membaginya menjadi 8 buah blok (X_1, X_2, ..., X_8) yang masing-masing blok berukuran 16-bit. Blok tersebut selanjutnya akan diputar sebanyak 4 kali dengan masing-masing putaran memiliki prosedur:
    a. X_1 di-XOR-kan dengan X_3
    b. X_5 di-XOR-kan dengan X_7
    c. X_2 di-XOR-kan dengan X_4
    d. X_6 di-XOR-kan dengan X_8
    e. Hasil langkah *c* dan C_1 dijumlahkan
    f. Hasil langkah *b* dan C_4 dijumlahkan
    g. Hasil langkah *e* dikalikan dengan C_2
    h. Hasil langkah *f* dan *g* dijumlahkan
    i. Hasil langkah *h* dan C_3 dikalikan
    j. Hasil langkah *g* dan *i* dijumlahkan
    k. X_1 dan hasil langkah *i* dijumlahkan
    l. X_3 dan hasil langkah *i* di-XOR-kan
    m. X_6 dan hasil langkah *j* di-XOR-kan
    n. X_8 dan hasil langkah *j* dikalikan
    o. C_5 dan hasil langkah *a* dikalikan
    p. C_8 dan hasil langkah *d* dikalikan
    q. C_6 dan hasil langkah *o* dijumlahkan
    r. Hasil langkah *p* dan *q* dikalikan
    s. C_7 dan hasil langkah *r* dikalikan
    t. Hasil langkah *q* dan *s* dijumlahkan
    u. X_2 dan hasil langkah *t* di-XOR-kan
    v. X_4 dan hasil langkah *t* dikalikan
    w. X_5 dan hasil langkah *s* dijumlahkan
    x. X_7 dan hasil langkah *s* di-XOR-kan

    Hasil langkah *x, w, k, l, m, n, v, u* secara berurutan akan menjadi Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7, Y_8 untuk putaran berikutnya.

    Setelah melewati 4 putaran, blok-blok tersebut disatukan kembali menjadi sebuah blok berukuran 128-bit dan digeser memutar ke kanan sebanyak 29-bit.

    Blok ini kemudian di-XOR-kan dengan umpan turunan saat ini (S_i). Blok yang telah di-XOR-kan ini akan menjadi umpan turunan berikutnya (S_{i+1}). Gambar 4 menggambarkan satu putaran pembangkitan umpan turunan.



Gambar 4: Satu putaran pembangkitan umpan turunan

Keterangan simbol:

⊕ XOR 16-bit
⊞ Penjumlahan modulo 2^{16}
⊙ Perkalian modulo 2^{16}

Pada algoritma di atas dapat dilihat bahwasannya algoritma tersebut membutuhkan masukan berupa konstanta *C* (16-bit) sebanyak 8×4=32 buah. Tabel 1 menunjukkan nilai konstanta *C* dalam heksadesimal. Nilai-nilai konstanta *C* merupakan sebuah bilangan mungkin prima (*probable prime*) dengan panjang 512-bit.

Tabel 1: Nilai konstanta *C*

No.	Putaran	*i*	C_i (hexa)
1.	1	1	ffb8
2.	1	2	b106
3.	1	3	6f23
4.	1	4	4aa7
5.	1	5	3de2
6.	1	6	b2e8
7.	1	7	2e70
8.	1	8	628a
9.	2	1	51d3
10.	2	2	ce30
11.	2	3	c44a
12.	2	4	94cc
13.	2	5	bff9
14.	2	6	1748
15.	2	7	1d52
16.	2	8	5ea2
17.	3	1	ad88

Tabel 1 (lanjutan)

No.	Putaran	i	C_i (hexa)
18.	3	2	3e55
19.	3	3	0253
20.	3	4	ebee
21.	3	5	f78b
22.	3	6	555c
23.	3	7	cadc
24.	3	8	17f5
25.	4	1	30c3
26.	4	2	54b9
27.	4	3	6d29
28.	4	4	e8c6
29.	4	5	879a
30.	4	6	03de
31.	4	7	f8fb
32.	4	8	e595

2.3. Integrasi dengan *OTP*

Integrasi dengan *OTP* dilakukan dengan memanfaatkan kunci-kunci acak semu yang dihasilkan oleh *IKG*. Untuk setiap *text* yang ingin di-*encrypt* atau di-*decrypt*, akan dibangkitkan kunci sepanjang *text* tersebut. Hal ini dimaksudkan agar sedapat mungkin tidak ada pemakaian kunci yang berulang dikarenakan salah satu kelemahan *OTP* adalah pemakaian kunci yang berulang [1]. Selain itu perlu dilakukan penyaringan terhadap kunci acak semu yang dipakai untuk enkripsi sehingga kunci yang bernilai 0 tidak dipakai. Hal ini perlu dilakukan karena apabila kunci bernilai 0 digunakan untuk enkripsi, maka *ciphertext* sama dengan *plaintext*.

3. PENGUJIAN

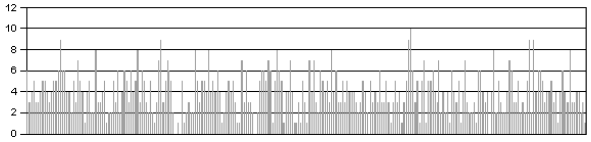
Pengujian algoritma *IKG* dilakukan dengan membangkitkan kunci sepanjang 8192-bit atau 1024 buah kunci 8-bit menggunakan umpan *"*`C>3[4R=e*BG!Q8D$`*"*. Kunci yang telah dibangkitkan akan diuji dengan menggunakan dua buah pendekatan (statistik dan perulangan). Selain kedua pendekatan tersebut, algoritma *IKG* juga akan diuji dengan uji pembangkitkan parsial. Pada makalah ini dibatasi pengujian dilakukan menggunakan metode non-formal.

Pembangkitan kunci pada pengujian dilakukan dengan mengimplementasikan algoritma *IKG* dalam bahasa *C++* dan menjalankannya pada sebuah komputer *PC* dengan *processor* berkecepatan 1,6 GHz. Program tersebut dapat membangkitkan kunci dengan kecepatan 2.451.464 $^{bit}/_{detik}$ atau 2,4 Mbps. Hasil pembangkitan kunci ditampilkan pada lampiran makalah ini.

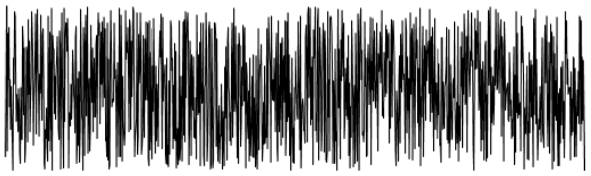
3.1. Uji Statistik

Uji statistik digunakan untuk menguji distribusi kunci yang dihasilkan. Semakin merata distribusi kunci, maka algoritma tersebut semakin baik karena semakin tahan terhadap analisis frekuensi.

Gambar 5 menggambarkan grafik distribusi kunci dan gambar 6 menggambarkan grafik kunci yang dibangkitkan (setiap kunci memiliki panjang 8-bit).



Gambar 5: Grafik Frekuensi Kunci



Gambar 6: Grafik Kunci

Dari kedua grafik tersebut dapat dilihat bahwasannya:

1. Terdapat lima buah kunci yang tidak pernah muncul yaitu 68, 70, 104, 156, 211.
2. Distribusi frekuensi kemunculan kunci masih belum merata.
3. Frekuensi kemunculan kunci acak (belum ditemukan pola)
4. Aliran kunci yang dibangkitkan acak (belum ditemukan pola)

Dari dua hal tersebut, maka algoritma *IKG* dikatakan cukup baik menurut uji statistik.

3.2. Uji Perulangan

Uji perulangan digunakan untuk menguji seberapa panjang (dalam bit) kunci yang dihasilkan mengalami perulangan. Apabila kunci yang dihasilkan tidak mengalami perulangan, maka dapat diasumsikan bahwasannya algoritma pembangkitan kunci acak semu menghasilkan kunci yang benar-benar acak (*truly random*). Selain itu, semakin panjang kunci yang dihasilkan maka semakin baik algoritma pembangkitan kunci.

Pada kasus uji ini, kunci yang dihasilkan oleh algoritma *IKG* tidak mengalami perulangan walaupun telah dibangkitkan sebanyak 8192-bit. Menurut uji perulangan, maka algoritma *IKG* dapat dikatakan baik karena tidak terjadi kunci yang berulang.

3.3. Uji Pembangkitan Parsial

Uji ini digunakan untuk menguji apakah algoritma pembangkitan tahan terhadap serangan pembangkitan parsial. Serangan ini dapat didefinisikan: dengan mengetahui satu atau beberapa buah kunci, dan IV_0 (tanpa mengetahui umpan), seorang kripanalis dapat membangkitkan kunci sebelumnya dan atau kunci sesudahnya [1].

Uji ini sangat penting untuk dilakukan karena sebuah algoritma pembangkitan bilangan acak semu yang baik tahan terhadap serangan ini dan secara intuitif, bilangan acak (*truly random*) yang satu dengan yang lainnya tidak memiliki keterikatan apapun.

3.3.1. Analisis algoritma *Key Generator*

Pada setiap putaran (1 s.d. 8), algoritma yang dipakai pada modul *key generator* memiliki persamaan-persamaan matematis:

$$A = X_1 \odot K_1 \quad \text{.....(1)}$$
$$B = X_2 \boxplus K_2 \quad \text{.....(2)}$$
$$C = X_3 \odot K_3 \quad \text{.....(3)}$$
$$D = X_4 \boxplus K_4 \quad \text{.....(4)}$$
$$G = (A \oplus C) \odot K_5 \quad \text{.....(5)}$$
$$J = \big((B \oplus D) \boxplus G\big) \odot K_6 \quad \text{.....(6)}$$
$$I = J \boxplus G \quad \text{.....(7)}$$
$$M = C \oplus J \quad \text{.....(8)}$$
$$N = D \oplus I \quad \text{.....(9)}$$
$$Q = \big((A \oplus J) \oplus (B \oplus I)\big) \boxplus K_7 \quad \text{.....(10)}$$
$$T = \big((M \oplus N) \odot Q\big) \boxplus K_8 \quad \text{.....(11)}$$
$$S = T \odot Q \quad \text{.....(12)}$$
$$Y_1 = (A \oplus J) \boxplus S \quad \text{.....(13)}$$
$$Y_2 = M \odot T \quad \text{.....(14)}$$
$$Y_3 = (B \oplus I) \odot T \quad \text{.....(15)}$$
$$Y_4 = N \boxplus S \quad \text{.....(16)}$$

Keterangan simbol:

- $\oplus$ XOR 16-bit
- $\boxplus$ Penjumlahan modulo 2^{16}
- $\odot$ Perkalian modulo 2^{16}

Persamaan-persamaan matematis tersebut (1) s.d. (16) memiliki sifat:

1. Tidak mungkin mengetahui nilai K_1, K_2, ..., K_8 walaupun nilai Y_1, Y_2, Y_3, Y_4, X_1, X_2, X_3, dan X_4 diketahui.
2. Tidak mungkin mengetahui nilai Y_1, Y_2, Y_3, dan $Y4$ tanpa mengetahui nilai K_1, K_2, ..., K_8 walaupun nilai X_1, X_2, X_3, dan X_4 diketahui.
3. Tidak mungkin mengetahui nilai X_1, X_2, X_3, dan X_4 tanpa mengetahui nilai walaupun nilai K_1, K_2, ..., K_8, $Y1$, Y_2, Y_3, dan Y_4 diketahui.

Pada putaran terakhir, nilai P yang dihasilkan merupakan potongan bit ke-5 s.d. 12 dari hasil persamaan matematis:

$$f = \big((Y_1 \oplus Y_4) \boxplus K_5\big) \oplus \big((Y_2 \oplus Y_3) \odot K_6\big) \quad \text{.....(17)}$$

Keterangan simbol:

- $\oplus$ XOR 16-bit
- $\boxplus$ Penjumlahan modulo 2^{16}
- $\odot$ Perkalian modulo 2^{16}

Persamaan (17) memiliki sifat:

1. Tidak mungkin mengetahui nilai Y_1, Y_2, Y_3, dan Y_4 walaupun nilai f, K_5, dan K_6 diketahui.
2. Tidak mungkin mengetahui nilai K_5 dan K_6 walaupun nilai f, Y_1, Y_2, Y_3, dan Y_4 diketahui.
3. Tidak mungkin mengetahui nilai $Y1$, Y_2, Y_3, Y_4, K_5, dan K_6 apabila diketahui nilai f.

Dengan ke-6 sifat di atas, dapat disimpulkan:

1. Dengan mengetahui nilai f, tidak mungkin kunci masukan dapat diketahui.
2. Dengan tidak mengetahui kunci masukan, tidak mungkin nilai f, Y_1, Y_2, Y_3, dan Y_4 dapat diketahui walaupun nilai X_1, X_2, X_3, dan X_4 diketahui.
3. Tidak mungkin nilai X_1, X_2, X_3, dan X_4 dapat diketahui walaupun nilai f, Y_1, Y_2, Y_3, Y_4, dan kunci masukan diketahui.

Dari hasil analisis algoritma *key generator* di atas, maka dapat disimpulkan bahwasannya algoritma *IKG* secara matematis tahan terhadap serangan pembangkitan parsial.

4. KESIMPULAN

Algoritma pembangkit bilangan acak semu, *IKG*, merupakan algoritma yang baik dan aman untuk digunakan sebagai alternatif pembangkitan kunci pada algoritma kriptografi simetrik *OTP*. Hal ini dikarenakan sifat *IKG* yang secara matematis tahan terhadap serangan pembangkitan parsial, distribusi kunci yang cukup merata, belum ditemukannya pola kunci yang dibangkitkan, dan tidak berdasarkan asumsi apapun (contoh: sulitnya pemfaktoran sebuah bilangan).

DAFTAR REFERENSI

[1] Stinson, Douglas, *Cryptography: Teory and Practice*, CRC Press, Maret 1995.

[2] *International Data Encryption Algorithm*, http://www.mediacrypt.com/, Media Crypt AG, Zürich, Switzerland.

LAMPIRAN
Kunci yang dibangkitkan (8192-bit)

```
0001 0111 0110 1111 1111 1111 0001 1111 1101 0101 1000 1110 1101 1110 0110 0100 1000 1110 0100
0001 1000 0011 0101 0100 0010 1011 0000 0001 1010 1111 1000 0110 1111 0111 1110 1111 1001 1010
0100 1101 0111 0110 0110 0001 0011 1110 0111 1111 0110 0010 0001 1110 0000 1100 1000 0001 0011
0101 0011 0001 1101 1111 1100 1000 0101 0111 1011 1110 0000 1110 0010 0010 0110 1010 1100 0000
1011 1001 0010 1101 1110 1010 0111 0111 1100 1000 0011 1101 0110 1100 1001 0000 1111 0111 1101
0110 1011 0010 0010 1000 0111 0100 1000 1011 1011 1001 1111 0011 0011 1001 1011 1010 1111 1001
1001 1110 0111 1110 0011 0110 1001 0110 0111 1000 1111 0001 1000 1111 0100 1111 1101 0010 1101
1100 1001 0011 0000 1101 0000 0011 0100 0000 0011 0011 0110 0010 0010 0111 0001 0000 1110 1111
0010 1111 0111 1000 1101 0111 0010 0000 0101 1101 1111 1011 0000 0001 1101 0010 1011 1100 1011
1100 0101 1001 1001 0010 0101 0001 1001 1000 1011 0000 1100 1111 1001 0100 0100 0000 0011 0001
1101 1000 1100 0101 0000 0011 1001 1000 1111 0100 1000 0010 0100 0011 0000 0110 1111 1100 0100
1101 1100 1111 1110 0101 1110 0101 1011 1010 1110 1011 1011 0100 0000 0101 0101 1011 0101 0011
0100 0101 1000 0111 0101 0000 0111 0100 1011 1011 1010 0001 0111 1000 0110 1100 0011 1100 1100
1111 1101 1111 0000 1100 0010 0101 1001 0000 0010 1000 1001 1010 0000 1110 0011 1100 1111 1010
1010 1111 0101 0101 0100 1110 0000 0010 1101 1101 1011 1111 0001 1001 1100 1000 1111 0001 0101
0110 1010 1010 1111 1110 0001 1100 0111 0010 1000 1111 0001 1111 0100 0001 1110 0010 1001 0100
0110 1011 1011 1110 0011 1101 0000 0000 1001 0011 1010 1111 0101 0011 0101 0100 1100 0011 1101
0010 0011 0011 1100 0111 1111 0100 1101 0010 1110 0100 0100 1001 1001 1001 1101 0111 1000 0110
1110 0111 0110 0011 1010 1111 1110 1101 0111 0001 1101 1100 0011 0000 0011 0101 0001 0111 1110
0101 0001 0001 1010 1101 0000 0011 0000 1000 1011 1000 1111 1000 1011 1100 0110 0100 0110 1111
0110 1011 0110 1110 1110 0101 1001 0000 1001 1101 0111 1011 0000 0111 1000 0011 1010 0111 0101
1101 0001 0110 1111 1011 1001 0110 0001 0111 1111 0001 1011 0000 0000 1111 1001 0010 0100 1101
0011 1001 0010 1001 0011 1101 1111 1000 1010 1101 1100 0111 1011 0111 1101 1101 1001 0101 0011
1101 1110 1010 1110 0000 0100 0000 1100 1000 0111 0100 1010 0100 1111 1110 1111 1000 1000 0110
0001 0111 0011 0110 0100 0010 0101 0000 1111 0101 0001 1010 0011 0010 1101 0100 1000 1000 1001
1001 1111 0100 1011 1110 0001 1011 0010 0111 0011 1011 1101 0001 0101 1110 0011 0100 1100 1100
0110 1110 1110 0111 1000 0110 0110 1110 1110 0101 0001 0100 0000 1001 1110 1001 0010 0010 1011
1000 0100 0000 0010 1101 1101 1100 0010 1010 0001 0011 1010 0011 0101 1100 1110 1100 1010 1100
0101 0011 0001 1100 1010 0010 0010 1111 1000 0111 0101 1100 0010 0111 1001 1101 1011 0011 1101
1001 1011 0001 1111 0011 1000 1011 0010 0011 1110 1101 0110 1100 1001 0010 1001 0110 1010 0100
0001 1011 0000 1100 1010 1110 0000 1010 0010 1001 0000 0111 1000 1011 0001 1011 0110 1010 1110
1000 1110 0000 1001 1011 1010 1110 0010 0011 1000 0101 0011 0011 1100 0101 0010 1111 1011 1010
0000 0110 0001 1111 0100 0010 0000 1011 0001 1001 1110 1000 0001 0010 0010 1001 0010 0101 0101
0110 0001 0000 1001 0101 1110 0110 0011 1000 1110 1010 0010 1110 1111 0110 0111 0010 1100 0101
1110 0111 1111 0101 1011 1111 0000 1010 0011 1111 1100 1000 1001 0001 0001 1000 1011 0100 0111
0001 0101 1110 1011 0000 0011 0100 0100 1111 0011 0010 0111 0010 0001 0011 1111 0001 1000 0111
0101 1000 1011 0100 0101 1011 1100 0111 0000 1000 1110 0101 1000 1001 0110 1001 0110 1100 1110
0100 0001 1000 0000 1001 1000 0001 1000 0111 1111 0100 1100 0100 0000 1111 0010 1100 0110 0000
1101 1101 1110 0111 1101 0000 0000 0011 0000 1011 1101 1100 1000 0010 0000 0110 0001 0000 1000
0001 1110 0101 1110 0100 1101 0101 0010 0110 1100 1100 0011 1100 0001 0000 0100 0111 0110 0110
0000 1101 0011 1000 0011 0111 0000 0000 1000 0110 0100 0001 0000 0100 0101 1100 0011 0101 0000
1000 1010 0001 0101 0010 1100 0000 0111 0001 0001 0010 1100 1001 1101 1101 1000 0001 0111 0110
0010 1111 0101 1011 0010 1001 1011 0101 1110 0101 0011 0101 1000 0001 1111 1100 0101 1100 0111
0100 1110 0000 0110 1011 0101 0111 0001 0101 1100 0011 0000 1100 1101 0101 1100 0010 1100 0010
0010 0111 0100 1010 1010 0001 0110 0110 1110 0010 1100 1100 0110 1101 1101 1101 0111 0100 0000
0101 1010 1000 1001 0101 0100 0111 0011 0010 0111 1100 0000 1111 1010 1111 0001 1001 0011 0000
0101 0100 1001 0011 0001 0110 1011 0100 0001 0101 0111 0010 0100 0111 0000 0111 0000 0000 1011
0010 0010 0001 0000 0011 0101 1110 1101 1100 0010 0001 1000 1100 1010 0100 1010 0101 1100 0100
0101 1101 0000 1111 0010 1001 0110 1010 0010 1010 1010 0010 0010 1011 0100 0000 0011 0100 1101
1100 0010 0011 0110 1001 1110 1110 1100 1110 1011 0000 1010 0010 0000 1110 1111 0000 1000 1010
0110 1110 1000 1101 0110 1101 1010 1110 0111 1111 0011 1110 1011 1001 1101 0101 0010 0011 0000
1111 1110 1000 0110 0100 0011 1101 1000 0100 1100 0101 0010 1001 1100 0100 1100 1110 0011 0100
1010 0111 1011 0010 0110 1011 1110 1000 0001 1000 0011 1011 1110 0110 0100 1101 1111 0101 0101
0011 1011 0100 0001 1000 0111 1001 0000 0010 0011 0100 1010 1110 0101 1111 0000 1110 0100 0010
0000 1100 1000 0010 0100 1110 1011 1010 1000 1100 0111 1000 0101 0111 0101 0001 1110 0111 1010
0001 0110 0010 0010 0010 1110 1111 0000 0110 0011 1110 0110 0101 0110 1111 0110 0100 0011 1111
0011 1111 0010 1100 1100 1111 1110 0001 1001 1101 0010 1101 1111 0011 1111 1101 0111 0010 1100
1000 1111 1011 0100 0111 1010 0000 0101 0000 1011 0001 0000 1010 1101 1010 1010 1111 1100 0001
1111 1110 0011 1111 0101 0111 0010 1101 1110 1001 0000 0000 1111 1100 0001 0001 0100 1101 1101
1101 1011 0101 1001 0001 0001 1111 0011 1101 1011 1100 1110 0100 1100 0101 0001 1110 1101 1000
1010 1110 0001 0001 0000 1110 0001 1100 1111 0111 0101 0100 1000 1101 0001 1111 1110 1100 1110
1111 1110 1110 1101 0100 1001 1110 0000 1101 1000 1011 0110 1111 0000 1101 1011 1101 0010 1111
0011 1101 0000 0010 1001 0000 1111 1001 0110 0010 1011 1100 0110 1010 0001 0100 1100 1011 1000
1011 1111 1000 1100 1010 0000 0000 1001 0010 1110 0101 0011 0010 0100 1001 1001 1111 1000 0000
0101 0101 1010 1111 0101 0001 1101 0001 0001 1001 1000 1001 0000 0101 0110 0100 0011 0100 1110
1001 1011 1000 0111 0010 0000 0111 0111 0001 1000 0100 0000 1101 1101 0101 1001 0111 0010 1110
0001 0000 1111 0010 1101 0001 0000 0001 1010 0101 1110 1010 1111 1100 1000 1101 1011 1011 0011
1000 0011 1101 0010 1110 0100 0001 1110 0111 1000 0011 1010 1110 0001 0111 0001 1000 1010 1001
1100 0011 1001 1000 0110 0111 1010 1111 1110 1101 0000 1011 1010 1001 0111 1110 1110 1011 0110
1101 1010 0111 1101 1001 0111 0010 0110 0010 1000 1011 1110 0100 0110 1101 0001 1111 1011 0101
1001 1001 1010 1000 1101 0111 1000 1001 1010 1001 1111 1000 0101 0101 0110 0010 0101 0010 1011
0111 1001 0100 0111 0011 1110 0010 0111 1001 1101 1011 1010 1110 1100 1110 1101 0101 0010 1100
1010 1010 0100 0010 1100 1101 1110 1010 0101 0011 1101 0101 1110 1001 1100 1011 1010 1000 0101
```

1101 0001 1111 0000 1000 0111 1100 0110 0100 0100 0000 0000 1111 1000 1101 0000 0000 1011 0010
0010 1100 1101 0100 1001 0100 0111 1010 1110 1001 1010 0001 1000 1110 1001 0111 0111 0011 0001
1100 1110 1011 1101 1010 0011 1100 1011 0000 0101 1110 1001 0000 0001 0011 0111 0000 1111 0101
0111 0111 1100 0010 1101 1100 0001 0001 1011 0010 1110 0011 0111 0011 0010 0011 1010 0101 1110
1110 1010 1111 1100 1010 0100 1011 1001 1111 0010 0110 1000 1000 1111 1011 0000 1001 0111 0011
0101 0011 1011 1000 1110 0000 0001 0111 0110 1110 1100 0010 1011 1001 0100 1110 1101 1011 1000
1100 1011 0101 1011 1101 1100 1001 0111 1000 1111 1010 0100 1101 1011 1110 1101 0000 0101 1010
1101 1110 0111 0110 0001 1111 1001 1110 1110 1011 0100 0000 1000 1001 1000 1000 0101 1100 0011
0010 1011 1100 0101 1000 1010 0110 0101 0111 0010 1001 0010 1100 0000 1110 1100 1110 1100 1110
0100 0001 1000 1010 0100 1101 0100 0011 1000 0011 1000 1010 1010 0000 1101 1110 0011 0001 0101
0011 0010 1011 1101 1011 0001 0101 0011 1000 1010 0100 0110 1101 0011 0001 0011 0101 1000 0110
1000 1111 0111 0001 1110 0111 0101 1111 0010 0001 0100 0010 0110 1010 0010 1100 1101 1111 0001
0000 0110 0110 0011 1111 0011 0000 1100 0000 1000 0011 1101 0111 0010 0100 1010 1111 0111 0111
1111 0011 1101 0100 0110 0111 0110 1100 1100 1111 0001 1101 1100 0011 1111 1000 1010 1110 1000
1100 0111 0011 1011 1001 1010 1100 1111 0100 1010 0011 1010 1100 0010 1000 0000 0010 0110 1011
1110 1011 1001 1101 1101 1000 1001 1111 1100 1001 1111 1000 1011 0110 0101 1110 0110 1111 1110
0111 0110 0011 0010 0000 1001 1101 0011 0001 1010 1010 1000 1011 0001 0010 0100 1101 1100 0010
1100 1000 0000 1111 0111 0111 0001 0101 1011 1001 1111 1000 1000 0101 1101 0110 1111 1101 0001
1011 1010 0001 0100 1011 0110 0011 1001 0011 0100 0111 0110 0101 1011 1011 1000 0100 1111 1001
1010 1101 0111 1110 1000 0001 1111 0100 1011 0001 0111 1001 1011 1100 1101 1011 1110 1010 0011
0110 1100 1111 1100 0110 0111 1110 0000 1111 1110 0111 1100 0010 0101 1111 0111 0010 0111 1101
1000 0011 1000 1000 0101 0000 0100 1100 1011 0111 1001 1010 0101 0110 0110 1110 0011 0010 0101
0000 0011 0111 1000 1101 1100 0000 0000 1011 1001 0101 0001 1101 1111 0110 0101 1010 1101 0101
0101 0010 0111 1000 0101 1000 0110 0101 0110 1011 0010 0000 1111 0000 1010 1000 1110 0000 1011
0111 1000 1101 0000 0100 0110 0010 0011 1010 0011 1011 0111 0011 1010 0011 0001 0011 0100 1100
0001 0001 1011 0111 0011 1100 1001 1011 1010 1011 0101 0011 1010 0100 0101 0001 0000 0111 1110
1100 0101 0111 1101 0000 1111 1010 0100 1010 1010 1101 0101 1000 0001 0001 0011 0000 0101 0001
0100 0001 1000 0011 0101 0101 0010 1101 1001 0001 1100 1110 0011 1100 0011 0010 0010 0101 1011
0101 1110 0001 1000 1001 1011 1000 1110 1111 0100 1110 0010 0111 0111 1000 1100 0111 0001 1100
0110 1110 1110 0011 1010 1000 1101 0111 0000 0100 0111 0101 1101 0000 0000 1111 1011 0101 0101
0110 1000 1011 0100 1111 1110 1110 0011 1100 0000 1001 0111 1001 0000 1000 0111 1111 0010 1011
1010 0000 0100 1000 1001 1010 1010 1010 0011 0010 0110 1100 0001 0101 0100 1110 1111 0110 1011
0101 1110 1001 1000 0111 1010 1101 0111 0110 0111 1100 1000 0001 1000 0010 0001 0010 1001 1001
0011 1000 0110 1111 0011 1101 1011 0000 1010 1110 0001 0111 1001 1111 0101 0010 0001 0110 0101
1011 0111 1110 0101 0101 1001 1001 0100 1010 1111 0000 1110 1101 0010 1111