Pembangkit Bilangan Acak Berbasis Fungsi *Hash*

Steven Andrew / 13509061
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
andy165@students.itb.ac.id

***Abstrak*—Fungsi hash dapat dimanfaatkan sebagai pembangkit bilangan acak semu karena pada tiap pemrosesan blok pesan, dihasilkan nilai *message digest* yang jauh berbeda dari pemrosesan sebelumnya. Hasil menunjukkan bilangan acak yang dihasilkan tidak memiliki periode perubahan dan polanya sulit ditebak.**

***Indeks Istilah*—fungsi *hash*, keacakan, *message digest*, pembangkit bilangan acak semu.**

I. PENDAHULUAN

Keacakan merupakan kejadian yang tidak menentu / tidak dapat diprediksi, dan atribut yang membuat keacakan adalah bilangan acak. Keacakan dapat dibangkitkan oleh keadaan lingkungan (menggunakan fenomena fisis), keacakan dari kondisi awal (dengan *chaos theory*), dan bilangan acak yang dibangkitkan oleh sistem. Sistem membangkitkan bilangan "acak" dengan algoritma komputasi tertentu. Algoritma tersebut disebut pembangkit bilangan acak semu.

Pembangkit bilangan acak semu umumnya menggunakan fungsi-fungsi matematis tertentu. Kebanyakan algoritma menggunakan fungsi-fungsi modular (modulo), seperti *Linear Conguential Generator* (LCG), dan *Blum Blum Shub*.

Prinsip dasar pembangkit bilangan acak semu adalah menggunakan nilai awal sebagai umpan (*seed*). Lalu dari nilai sebelumnya tersebut dibangkitkan suatu bilangan acak dengan fungsi tertentu.

Dalam makalah ini, dibahas suatu algoritma pembangkit bilangan acak semu yang memanfaatkan fungsi *hash*.

II. FUNGSI HASH

Fungsi *hash* adalah suatu fungsi satu arah yang memetakan serangkaian nilai pesan ke suatu nilai yang panjangnya tetap yang disebut *message digest*.

Fungsi *hash* memiliki karakteristik sebagai berikut:

1. Fungsi *hash* dapat memroses pesan berukuran sembarang (bahkan pesan kosong)
2. Nilai *hash* yang dihasilkan berukuran tetap
3. Untuk setiap nilai h yang dihasilkan, tidak dapat dihitung nilai x sedemikian sehingga $H(x) = h$.
4. Secara ideal, tidak dapat dicari x dan y yang berbeda sedemikian sehingga $H(x) = H(y)$.

Fungsi *hash* sangat sensitif dimana perubahan pesan sedikitpun dapat menyebabkan nilai *message digest* jauh berbeda. Dengan hal ini, fungsi *hash* dimanfaatkan sebagai *checksum* untuk mengecek integritas pesan/data atau mendeteksi perubahan pesan.

Fungsi *hash* terdiri dari pemrosesan beberapa blok pesan yang menghasilkan nilai *digest* yang jauh berbeda dari pemrosesan pada blok pesan sebelumnya. Pola perbedaannya pun sulit ditebak sehingga fungsi *hash* dapat dimanfaatkan sebagai pembangkit bilangan acak semu. Dalam makalah ini, fungsi *hash* yang digunakan adalah MD5.

III. ALGORITMA MD5

MD5 adalah fungsi *hash* yang memroses blok pesan sebesar 512 bit dan menghasilkan *message digest* berukuran 128 bit (biasanya direpresentasikan dalam sebagai bilangan heksadesimal). MD5 didesain oleh Profesor Ronald Rivest dan merupakan perbaikan dari MD4.

Secara garis besar, algoritma fungsi *hash* MD5 terdiri dari langkah-langkah sebagai berikut.

1. *Pre-processing* pesan dengan penambahan bit pengganjal (*padding bits*) dan nilai panjang pesan.
2. Inisialisasi nilai *digest* (disebut penyangga/*buffer*)
3. Pemrosesan tiap blok pesan 512 bit untuk mentransformasikan nilai *digest*.

A. Pre-processing pesan

Sebelum dilakukan komputasi *hash*, pesan terlebih dahulu ditambah bit pengganjal '1' dan selanjutnya beberapa bit pengganjal '0' hingga panjang bit pesan kongruen dengan 448 modulo 512. Lalu ditambah nilai

panjang pesan (modulo 2^{64}) yang panjangnya 64 bit sehingga panjang pesan menjadi kelipatan 512. Pesan disusun dalam blok-blok 512 bit.

B. Inisialisasi nilai buffer MD

Nilai *buffer* diinisialisasi berupa 4 register 32 bit dan total panjang nilai *digest* adalah 128 bit. Register diinisialisasi dengan nilai:

```
A = 0x67452301
B = 0xefcdab89
C = 0x98badcfe
D = 0x10325476
```

C. Pemrosesan Blok Pesan 512-bit

Pemrosesan blok pesan terdiri dari 4 putaran dengan 16 operasi per putaran. Misalkan dinotasikan operasi `[abcd k s i]` dimana a,b,c,d register *buffer* dan fungsi operasi adalah

```
a <- b + ((a + g(b,c,d) + X[k] + T[i])
<<< s)
```

dimana <<< adalah operasi *circular left shift* sebanyak s bit, X[k] adalah potongan blok pesan (k = 0..15), T[i] adalah nilai yang ditentukan dengan rumus

```
T[i] = floor(abs(sin(i+1))*pow(2,32))
```

Selengkapnya nilai-nilainya adalah sebagai berikut.

```
T[ 0.. 3] = { 0xd76aa478, 0xe8c7b756,
0x242070db, 0xc1bdceee }
T[ 4.. 7] = { 0xf57c0faf, 0x4787c62a,
0xa8304613, 0xfd469501 }
T[ 8..11] = { 0x698098d8, 0x8b44f7af,
0xffff5bb1, 0x895cd7be }
T[12..15] = { 0x6b901122, 0xfd987193,
0xa679438e, 0x49b40821 }
T[16..19] = { 0xf61e2562, 0xc040b340,
0x265e5a51, 0xe9b6c7aa }
T[20..23] = { 0xd62f105d, 0x02441453,
0xd8a1e681, 0xe7d3fbc8 }
T[24..27] = { 0x21e1cde6, 0xc33707d6,
0xf4d50d87, 0x455a14ed }
T[28..31] = { 0xa9e3e905, 0xfcefa3f8,
0x676f02d9, 0x8d2a4c8a }
T[32..35] = { 0xfffa3942, 0x8771f681,
0x6d9d6122, 0xfde5380c }
T[36..39] = { 0xa4beea44, 0x4bdecfa9,
0xf6bb4b60, 0xbebfbc70 }
T[40..43] = { 0x289b7ec6, 0xeaa127fa,
0xd4ef3085, 0x04881d05 }
T[44..47] = { 0xd9d4d039, 0xe6db99e5,
0x1fa27cf8, 0xc4ac5665 }
T[48..51] = { 0xf4292244, 0x432aff97,
0xab9423a7, 0xfc93a039 }
T[52..55] = { 0x655b59c3, 0x8f0ccc92,
0xffeff47d, 0x85845dd1 }
T[56..59] = { 0x6fa87e4f, 0xfe2ce6e0,
0xa3014314, 0x4e0811a1 }
T[60..63] = { 0xf7537e82, 0xbd3af235,
0x2ad7d2bb, 0xeb86d391 }
```

g(b,c,d) adalah fungsi yang berbeda untuk tiap putaran secara berturut:

```
F(b,c,d) = (b & c) | ((~b) & d)
G(b,c,d) = (d & b) | ((~d) & c)
H(b,c,d) = b ^ c ^ d
I(b,c,d) = c ^ (b | (~d))
```

Operasi dilakukan sebanyak 64 kali, sebagai berikut: [1]

```
Round 1:
[ABCD  0  7  1]
[DABC  1 12  2]
[CDAB  2 17  3]
[BCDA  3 22  4]
[ABCD  4  7  5]
[DABC  5 12  6]
[CDAB  6 17  7]
[BCDA  7 22  8]
[ABCD  8  7  9]
[DABC  9 12 10]
[CDAB 10 17 11]
[BCDA 11 22 12]
[ABCD 12  7 13]
[DABC 13 12 14]
[CDAB 14 17 15]
[BCDA 15 22 16]

Round 2:
[ABCD  1  5 17]
[DABC  6  9 18]
[CDAB 11 14 19]
[BCDA  0 20 20]
[ABCD  5  5 21]
[DABC 10  9 22]
[CDAB 15 14 23]
[BCDA  4 20 24]
[ABCD  9  5 25]
[DABC 14  9 26]
[CDAB  3 14 27]
[BCDA  8 20 28]
[ABCD 13  5 29]
[DABC  2  9 30]
[CDAB  7 14 31]
[BCDA 12 20 32]

Round 3:
[ABCD  5  4 33]
[DABC  8 11 34]
[CDAB 11 16 35]
[BCDA 14 23 36]
[ABCD  1  4 37]
[DABC  4 11 38]
[CDAB  7 16 39]
[BCDA 10 23 40]
[ABCD 13  4 41]
[DABC  0 11 42]
[CDAB  3 16 43]
[BCDA  6 23 44]
[ABCD  9  4 45]
[DABC 12 11 46]
[CDAB 15 16 47]
[BCDA  2 23 48]

Round 4:
[ABCD  0  6 49]
[DABC  7 10 50]
[CDAB 14 15 51]
```

```
[BCDA  5 21 52]
[ABCD 12  6 53]
[DABC  3 10 54]
[CDAB 10 15 55]
[BCDA  1 21 56]
[ABCD  8  6 57]
[DABC 15 10 58]
[CDAB  6 15 59]
[BCDA 13 21 60]
[ABCD  4  6 61]
[DABC 11 10 62]
[CDAB  2 15 63]
[BCDA  9 21 64]
```

Selanjutnya hasil yang didapat ditambahkan ke keempat register.

```
A <- A + a
B <- B + b
C <- C + c
D <- D + d
```

Hasil akhir *message digest* adalah penyambungan nilai keempat register tersebut.

III. Pembangkit Bilangan Acak Semu Berbasis MD5

Seperti halnya MD5, pembangkit bilangan acak semu berbasis MD5 membutuhkan serangkaian pesan sebagai "kunci rahasia", dan nilai-nilai *buffer* awal sebagai umpan.

Berikut langkah-langkah algoritmanya:

1. *Pre-processing* pesan dengan penambahan bit pengganjal (*padding bits*) dan nilai panjang pesan.
2. Menentukan umpan/*seed* (disebut *buffer*).
3. Pemrosesan blok pesan untuk menghasilkan nilai *digest*.
4. Ambil *n* bit LSB (*least significant bit*) dari tiap keempat buffer untuk menghasilkan bilangan acak 4*n* bit.

Nilai umpan (*buffer* awal) ditentukan sesuai standar algoritma MD5 atau dapat ditentukan sendiri.

Iterasi pembangkitan bilangan acak (pemrosesan blok pesan) dapat dilakukan berulang-ulang. Jika blok pesan terakhir telah diproses, pada iterasi berikutnya dilakukan proses untuk blok pesan awal.

IV. Percobaan dan Hasilnya

Percobaan ini memperlihatkan contoh-contoh penggunaan pembangkit bilangan acak berbasis MD5, dan menunjukkan keacakannya. Deret bilangan acak yang ditampilkan adalah bilangan yang dihasilkan setelah umpan.

Contoh 1

Pesan: "" (string kosong)

Umpan:

```
A = 0x67452301
B = 0xefcdab89
C = 0x98badcfe
D = 0x10325476
```

Ukuran bilangan acak: 32-bit

Deret 100 bilangan acak 32-bit pertama:

```
d903987e ce680ea0 3b4fdb94 6723083e
21648ca8 2f63b96b a4ccf7b3 d937b8bd
4ea730c3 1594f5e0 57ebaa54 76b9bb76
42560ef9 695b352b 0f85f293 abc8aafb
ac174e40 9c5f9f44 c0125669 e1da369e
eacc390e ea295842 d2f16a2e 8e50a795
38c2bfd0 f76ae632 530bb266 88577fd5
f88cc187 98a75792 c07418a2 40305509
d8ad1a45 9c99d8af e809930c 431711f8
be6d7991 22993276 4678bc1e 82a7f25f
5967edb9 1fff29c3 521143ee 4da4409e
bb4abbae e498d3df e2f2399d 2a41ad5b
e46167ae 2b8ddcd1 cceac817 ea549a68
8728fbdc a12dba28 41129963 737350fe
38f5a2a3 5b3259cc 222e0d33 532db1f3
f21cd3d1 4583b3b7 9fd6394e 00ee2c6a
77e6ed21 76e2d47f 166bb1d0 af9610d6
d2d19c30 0ebeffd0 2c4d361d aea3329e
1ff58d4e e99bc3c6 8e4941b1 608f218a
efa433d4 abaf07ab 02b2d4f0 f13b7955
6841791e 3dd8b15d 11217249 ea8cb3ad
b401356d abc6bb17 3bd188df 585913e0
f2850518 1097b9f8 6661abad 7404ce17
5af40ba9 51d3d12e a8d35455 5ab44a66
f2e3fdce 0891de06 c2370f76 1ef160a7
```

Deret 100 bilangan acak dengan modulo 100:

```
18 84 0 18
36 83 79 45
43 8 56 38
85 67 27 7
12 88 41 30
94 14 18 53
84 98 94 33
51 62 54 21
9 75 80 0
85 70 98 39
89 67 14 78
2 59 77 91
82 93 11 96
76 52 55 14
51 44 79 3
5 27 18 38
89 23 24 10
68 52 29 10
90 26 17 70
96 99 52 41
10 57 17 81
93 59 7 0
64 0 73 59
9 42 73 66
18 86 42 99
```

Deviasi standar dari deret 100 bilangan acak dengan modulo 100 = 31.3941

Contoh 2

Pesan: "message digest"

Umpan:

```
A = 0x67452301
```

```
B = 0xefcdab89
C = 0x98badcfe
D = 0x10325476
```

Ukuran bilangan acak: 32-bit

Deret 100 bilangan acak 32-bit pertama:

```
7c8d30d0 db9aeeb9 e1a5ebac c39e0f22
f5651c5a d8e89971 aeb69458 3d9dee37
0cb0c787 56b8a3e0 363c70a9 01aae26f
f8de071c fc64e077 dbb84eb5 c4f1ff7f
68ec1c4f b61a52d2 f3d6afb2 40d58d2c
afc4bb21 4559c6d5 78715519 928f8889
86cfe469 7d2a48a6 6c973df7 ff28ed47
017ce3ce a9247565 7abb5ec4 5f1fa28b
c9542774 b4679601 1068c85e 977996f0
4e580c87 35b1beeb 96ff8ab4 42b46405
8a40c26f 47afd943 ee4bcd81 037afbcf
43a49898 929fc988 dd6c8834 1312f8e9
eb7e9c90 0dcbc98b 4eaab3fe 158bf4d5
a1f1a222 b92381fe cf9de4a0 94f417cd
fcdf69cd 49d92004 740de654 8e43d9f6
06c4e63b 20bd4b52 bc4e849b ebb171ce
93e7c2a7 53030a88 9346d1fa aebefa8a
370ee1fc cd0e29f6 d508dd2b 0da2895e
de8f0041 023dde4a 8aeb6546 fb85c790
0ebf53c6 c8d86c6c 97912603 e5c90a26
ddc88933 0c3aae48 761493a0 1387d2ae
dbab0854 50e21dc2 01471114 4a2d3ad9
b023273a c3f4561c 3c4a286e 93db48f7
aa6da4f4 a49e880a ae58f608 6a542f89
9a7b8b41 84fa746d d1d8dcdf 6f8a2afa
```

Deret 100 bilangan acak dengan modulo 100:

```
56 61 72 82
14 89 12 11
7 52 65 3
80 95 77 19
7 50 70 32
61 9 73 37
69 10 27 63
98 53 44 47
40 89 94 12
23 95 88 41
87 55 17 3
40 52 92 73
76 11 46 17
58 98 92 45
45 76 88 82
91 98 63 66
99 32 26 62
36 26 15 6
65 34 94 92
46 32 67 98
59 96 0 98
92 34 44 69
18 48 74 15
72 38 36 89
41 17 63 46
```

Deviasi standar dari deret 100 bilangan acak dengan modulo 100 = 29.6211

Contoh 3

Pesan: "MD5 is one in a series of message digest algorithms designed by Professor Ronald Rivest of MIT (Rivest, 1994). When analytic work indicated that MD5's predecessor MD4 was likely to be insecure, MD5 was designed in 1991 to be a secure replacement."

Umpan:

```
A = 0x67452301
B = 0xefcdab89
C = 0x98badcfe
D = 0x10325476
```

Ukuran bilangan acak: 32-bit

Deret 100 bilangan acak 32-bit pertama:

```
e532ee52 38d71468 543388f4 fdb5e407
c244d093 bea79fc5 eecc251e 0970b2ca
4443677d 3d59966b 0c9c692c 07cb27cc
9e3fef71 bd0c3053 542d3acb d7149bb2
eef65b5f 4d71729c 867771ed c9fe2c7b
34ac9651 71b5a0e0 e2cfcff7 9c3e377e
d2d73a8b d2fbe818 a078651c 6a270b3e
0f1e58e6 384b773c 6c05b4f9 8e6ba52a
a75465e4 141f7e49 af86ab25 059a0da6
39dc2fbe 0f35acf9 2715fac8 353885a9
5c40d0da 434c8d39 fed7b5ce 570acdf8
113af8b5 aed0f0a4 474ee522 03ef6502
920531dc 697cca2d 10098b0b bb6a17b8
1eaf6a31 26471c9e 4919e696 d92eee08
7c3447e7 a0a76a95 46aa1cd1 de09647f
e8c3bf1c 84827a2b ce8dba95 51b7310b
913bea0a 6f95c544 4eea7447 28fc4f7d
e2c3ca72 5d694590 702fa038 2ecdb357
a5a6472a 5d272c20 ed68165c 1762011a
70a4ce28 b2141ded dc965c63 4c888528
7e4c9006 049077cb 4bf0fc2b 06c100e6
3f82a99f 888f8bf1 64dac9bf 4d5eff75
88ce95bd 0ab8fee3 e42d9f08 a813068f
5749c728 24aa7df4 1199e5f7 bf9c02c2
20ffa9a8 030cc425 a94b2e3f b02ddcd9
```

Deret 100 bilangan acak dengan modulo 100:

```
74 60 40 39
47 53 54 46
93 87 32 8
93 27 15 18
55 40 93 47
5 8 7 34
87 12 64 78
78 20 37 6
80 65 37 18
54 29 80 81
42 61 46 80
29 76 2 10
80 1 75 80
65 90 98 64
63 57 17 3
56 67 37 39
58 40 39 61
62 76 0 27
34 88 60 78
72 29 71 0
22 15 95 74
11 49 95 13
25 35 80 51
4 40 51 78
24 93 43 73
```

Deviasi standar dari deret 100 bilangan acak dengan modulo 100 = 28.1365

Terlihat bahwa deret bilangan acak yang dihasilkan tidak periodik dan tidak memiliki pola yang dapat ditebak. Selain itu, nilai deviasi standarnya (atau variansi) menunjukkan bahwa bilangan-bilangan acak variatif dan memiliki fluktuasi yang baik.

V. KESIMPULAN

Pembangkit bilangan acak semu berbasis fungsi *hash* menghasilkan bilangan acak yang tidak periodik dan memiliki keacakan yang baik dengan variansi yang cukup tinggi. Dengan demikian, pembangkit bilangan acak semu berbasis fungsi *hash* memenuhi syarat sebagai *cryptographically secure pseudorandom generator* (CSPRNG).

DAFTAR PUSTAKA

[1] RFC 1321, *The MD5 Message-Digest Algorithm*.
[2] *http://en.wikipedia.org/wiki/MD5*, diakses pada 2/5/2012.
[3] *http://en.wikipedia.org/wiki/Randomness*, diakses pada 2/5/2012.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 17 Desember 2010

Steven Andrew / 13509061