Studi Perancangan Algoritma Fungsi Hash

Kevin Chandra Irwanto – 13508063
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
synolyn@yahoo.com

ABSTRAK

Fungsi hash(*hash function*) adalah fungsi yang menerima masukan string yang panjangnya sembarang lalu mentransformasikannya menjadi string keluaran yang panjangnya tetap(*fixed*), umumnya berukuran jauh lebih kecil dibandingkan dengan ukuran string awalnya.

Fungsi hash ini banyak diaplikasikan pada sistem penyamarataan panjang password pada suatu basis data, atau digunakan pada tanda tangan digital (*digital signature*).

Penulis mengambil topik ini karena tertarik oleh salah satu sifat fungsi hash yaitu perbedaannya yang sangat jauh ketika isi pesan diubah 1 bit atau 1 karakter saja.

Selanjutnya akan dibahas tentang fungsi hash itu sendiri dan suatu algoritma fungsi *one-way-hash* baru yang dirancang oleh penulis.

Kata kunci : *hash function, digital signature, fixed output length, message digest, message integrity check,*

1. PENDAHULUAN

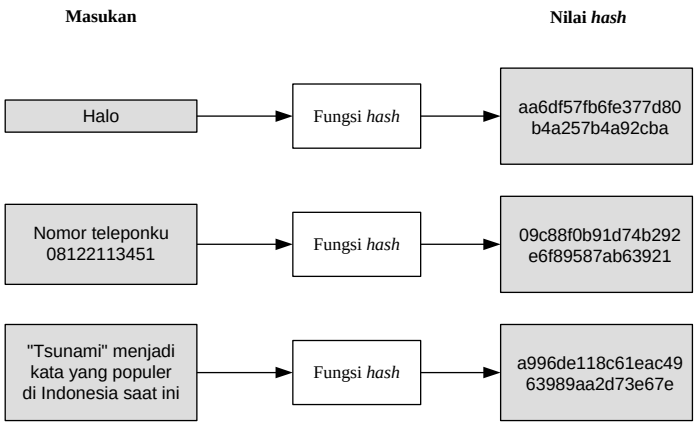
Jaman sekarang, teknologi berkembang sangat cepat. Begitu pula dengan kriptografi yang nantinya akan ada banyak sekali algoritma-algoritma kriptografi dengan cara dan sistem yang berbeda-beda. Fungsi hash adalah salah satu algoritma kriptografi yang membuat suatu data menjadi berbeda satu dengan yang lainnya.

Fungsi hash memiliki satu sifat yaitu menghasilkan *output* yang sangat jauh berbeda ketika isi data diubah 1 bit atau 1 karakter. *Output* dalam fungsi hash biasa disebut dengan *message digest*.

Algoritma-algoritma fungsi hash yang telah ada sekarang antara lain SHA, MD5, GOST, HAVAL, dll. Yang banyak dikenal adalah SHA dan MD5.

2. FUNGSI HASH

Fungsi hash seperti telah disebutkan diatas, adalah fungsi yang menerima masukan string yang panjangnya sembarang(bisa panjang, bisa pendek) lalu mentransformasikannya menjadi string keluaran yang panjangnya tetap(*fixed*), yang umumnya berukuran jauh lebih kecil dibandingkan dengan ukuran string awalnya.



Pada gambar diatas adalah salah satu gambaran tentang *input output* dari suatu fungsi hash. *Output* diekspresikan ke dalam bentuk hexadesimal.

Fungsi hash yang akan dibahas adalah fungsi hash satu arah(*one-way-function hash*). Fungsi ini memiliki beberapa sifat :

- Bisa diterapkan dengan ukuran berapapun.
- Hasil ukuran keluarannya tetap(*fixed output length*).
- H(x) mudah dihitung untuk setiap x.
- Untuk setiap hasil H(x) = h, tidak bisa lagi dikembalikan menjadi x.
- Tidak mungkin bisa mencari nilai $y \neq x$ dengan H(y) = H(x).

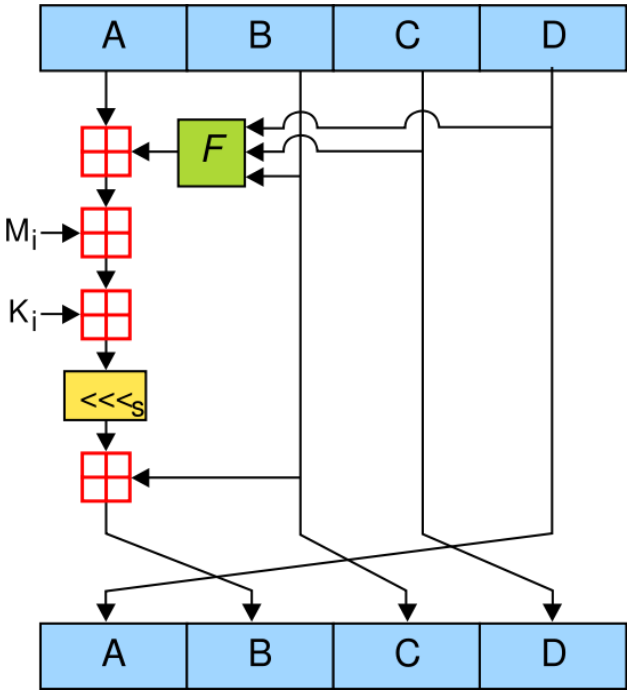
Jadi pada intinya, fungsi hash satu arah ini membuat setiap data menjadi unik, dan juga ketika salah satu isinya diubah, keluarannya berubah menjadi sesuatu yang berbeda sangat jauh dengan keluaran sebelumnya.

Oleh karena itu, fungsi hash satu arah ini dipakai pada aplikasi penyamarataan panjang password dalam suatu tabel basis data. Digunakan juga dalam pemverifikasian sebuah salinan arsip dengan arsip aslinya. Yang dilakukan adalah mengirimkan *message digest* dari arsip aslinya, jika *message digest* tersebut sama dengan *message digest* dari salinan arsipnya, maka arsip tersebut sudah tersalin seluruhnya.

Sebagai perbandingan, akan dibahas sedikit mengenai algoritma SHA dan MD5.

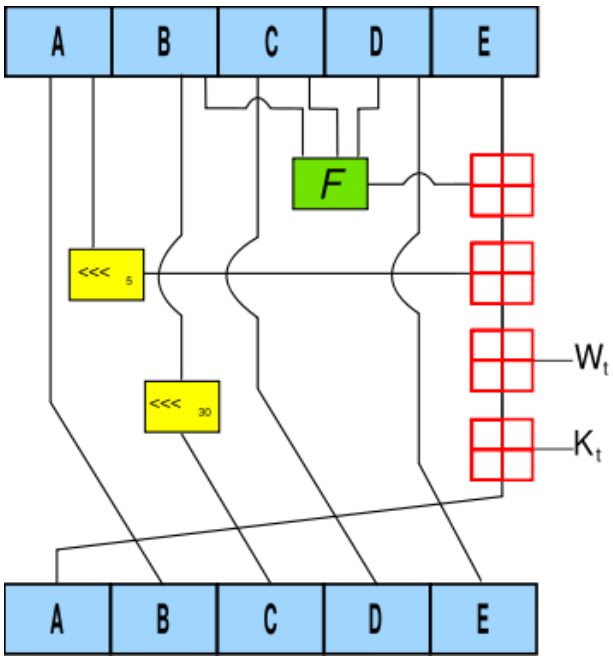
2.1. SHA

SHA adalah fungsi hash satu arah yang dibuat oleh NIST(*National Institute of Standards and Technology*) dan digunakan bersama DSS(*Digital Signature Standard*). SHA didasarkan oleh algoritma MD4 yang sebelumnya sudah ada. Hasil *message digest* yang didapat SHA-1 mempunyai panjang 160 bit. Gambar di bawah adalah ilustrasi dari proses algoritma SHA-1.



2.2. MD5

MD5 juga merupakan fungsi hash satu arah yang dibuat oleh Ron Rivest dari MIT. MD5 merupakan hasil perbaikan dari MD4 karena algoritma tersebut berhasil diserang kriptanalis. Hasil *message digest* yang didapat MD5 mempunyai panjang 128 bit.



Gambar di atas adalah ilustrasi dari algoritma MD5.

3. STUDI PERANCANGAN ALGORITMA FUNGSI HASH

Berdasar algoritma SHA-1 dan MD5, terdapat persamaan di kedua algoritma tersebut. Yaitu pada *buffer-buffer* yang masing-masing panjangnya 32 bit, tahap *pre-processing* dan pada tahap *processing*-nya.

Dalam tahap *pre-process* ini, pesan asli di-*append* oleh sebuah bit ‘1’, lalu dilanjutkan dengan bit ‘0’ sampai pesan tersebut menghasilkan kelipatan 512 dikurangi oleh 64. Kemudian untuk 64bit terakhir diisi oleh panjang dari pesan asli sebelum di-*append*.

Kemudian untuk *buffer*, ada kemungkinan nilai tersebut bisa diisi oleh berapapun yang nantinya akan dibahas pada bagian analisis. *Buffer* ini merupakan ‘bahan baku’ penghasil *output* atau dikenal sebagai *message digest* setelah melalui tahap *processing*.

```
for i from 0 to 79
        if 0 ≤ i ≤ 19 then
            f = (b and c) or ((not
b) and d)
            k = 0x5A827999
        else if 20 ≤ i ≤ 39
            f = b xor c xor d
            k = 0x6ED9EBA1
        else if 40 ≤ i ≤ 59
            f = (b and c) or (b and
d) or (c and d)
            k = 0x8F1BBCDC
        else if 60 ≤ i ≤ 79
            f = b xor c xor d
            k = 0xCA62C1D6

        temp = (a leftrotate 5) + f
+ e + k + w[i]
        e = d
        d = c
        c = b leftrotate 30
        b = a
        a = temp
```

Kode diatas adalah sebagian dari algoritma fungsi SHA-1. Persamaannya dengan MD5 adalah pada bagian fungsi f dan cara pemrosesannya yaitu proses dilakukan untuk setiap blok yang berukuran 512 bit. Kemudian *message digest* didapat dari menggabungkan seluruh *buffer* yang ada.

4. FUNGSI HASH YANG DIBUAT

Diawali dengan inisialisasi 5 buah *buffer* 32 bit seperti pada algoritma SHA-1 dan MD5. Untuk nilai inisialisasinya dicoba dengan nilai yang berbeda dengan SHA-1.

```
/********TAHAP 1********/
/******* My Hash *******/

h0 = 0x70615243
h1 = 0xABF89EDC
h2 = 0xCD8FE9BA
h3 = 0x34162507
h4 = 0xF0D2B498
```

Setelah itu dilakukan *padding* yang sama dengan algoritma SHA-1 dan MD5. Yaitu *padding* sebuah bit '1' dilanjutkan bit '0' sebanyak kelipatan 512 dikurangi 64, dan 64 bit terakhir diisi dengan panjang data.

Kemudian blok-blok yang berisi 512 bit tersebut dipecah menjadi 16 buah 32 bit. Pada algoritma ini, pengulangan yang dilakukan sebanyak 32 kali. Untuk mendapatkan 32 buah 32 bit dilakukan pengulangan seperti *pseudocode* dibawah.

```
/********TAHAP 2********/
/******* My Hash *******/

//  w[i],0≤i≤15  sebagai  pecahan-
pecahan 32bit

for i from 16 to 47
       w[i] = w[i-1] xor w[i-10]
xor w[i-11] xor w[i-13]
```

Setelah didapat 64 buah 32 bit, lalu masuk ke tahap pengulangan utama.

```
/********TAHAP 3********/
/******* My Hash *******/

//  w[i],15≤i≤47  sebagai  pecahan-
pecahan 32bit yang digunakan dalam
looping
// variabel a, b, c, d, dan e
diinisialisasikan dengan nilai h0,
h1, h2, h3, dan h4

for i from 16 to 47
        if i mod 4 = 0 then
            f = a xor c or (b and d)
        else if i mod 4 = 1 then
            f = (e and c) xor not(a)
        else if i mod 4 = 2 then
            f = (c and b) xor (a and
c) xor (b and e)
        else if i mod 4 = 3 then
            f = not(a or e)

        temp = e + w[i]
        e = c + f + f
        c = a
        a = d + f + w[i]
        d = b
        b = temp
```

Pengulangan diatas dilakukan untuk setiap blok 512 bit. Jadi didapat nilai a, b, c, d, dan e untuk setiap pengulangan proses 512 bit data. Setelah didapat nilai-nilai tersebut, dilakukan penjumlahan dengan aturan dibawah ini.

```
/********TAHAP 4********/
/******* My Hash *******/

h0 = h0 + a
h1 = h1 + b
h2 = h2 + c
h3 = h3 + d
h4 = h4 + e
```

Terakhir, menggabungkan h0, h1, h2, h3, dan h4 sehingga didapat *message digest* untuk data tersebut.

5. ANALISIS

Penulis sempat membuat program dengan mengganti *source code* tugas kecil 4 yang telah dibuat sebelumnya. Akan dibahas mengenai hasil yang diperoleh, namun sebelum itu akan dibahas alasan pengambilan algoritma pada fungsi hash yang dibuat.

Pertama-tama, penulis mengambil contoh algoritma dari SHA-1 dan MD5. Tidak heran jika algoritmanya terasa mirip.

Pada tahap 1, penulis mencoba mengganti angka-angka yang ada didasarkan pada pemikiran penulis bahwa penggantian angka-angka tersebut tidak berpengaruh pada sifat-sifat yang ada pada fungsi hash. Namun pemilihan angka tersebut berpengaruh pada kriptanalis yang sedang menyerang fungsi hash tersebut.

Kemudian diantara tahap 1 dan tahap 2, terdapat proses *padding bits*. Penulis hanya mengikuti algoritma SHA-1 dan MD5.

Pada tahap 2 juga hanya mengubah sedikit dari algoritma SHA-1 dan MD5. Namun, pada fungsi hash yang dibuat, tidak menggunakan bit-bit 'data asli' secara langsung, yang digunakan hanya w[16] sampai dengan w[47]. Hal ini dilakukan sebagai salah satu usaha memproteksi data asli.

Kemudian masuk ke pengulangan utama, yaitu pengulangan sebanyak 32 kali yang dilakukan untuk memroses variable a, b, c, d, dan e. Pada fungsi hash yang dibuat, melakukan pengulangan sebanyak 32 kali. Hal ini didasarkan pada SHA-1 yang terlalu banyak melakukan pengulangan. Penulis juga berpikir jika pengulangan semakin banyak, maka pola pengulangan tersebut akan lebih mudah terlihat daripada yang melakukan pengulangan lebih sedikit. Walaupun hal tersebut nantinya akan menjadikan hasil yang dikeluarkan menjadi kurang 'acak' dibandingkan dengan yang melakukan pengulangan lebih banyak.

Di tahap 3 ini pula, penulis membedakan fungsi hash yang dibuat dengan SHA-1 atau MD5 yakni dengan cara pembagian fungsi yang berbeda. Pada fungsi hash yang dibuat, penulis secara cepat mengacak seluruh variabel a, b, c, d, dan e untuk setiap putaran yang dilakukan.

Seperti pada *code* yang telah dilampirkan pada bab 3 di atas, algoritma SHA-1 membagi fungsinya secara berurutan yakni dari 0 s.d. 19 berbeda dengan 20 s.d. 39, dst. Sedangkan pada fungsi yang dibuat, pengulangan fungsi dilakukan pada setiap kelipatan 4.

Kemudian kekuatan algoritma SHA-1 terletak pada banyaknya putaran. Karena setelah menghitung fungsi f, nilai a, b, c, d, dan e hanya digeser nilai-nilainya ketika hanya nilai a saja yang diproses. Sedangkan dalam fungsi hash yang dibuat, kekuatannya tidak terletak pada banyaknya putaran, namun pada algoritmanya itu sendiri.

Terakhir, pada tahap 4 juga mengikuti SHA-1. Lalu *message digest* fungsi hash yang dibuat didapat dari menggabungkan h0, h1, h2, h3, dan h4 secara berurutan.

Kemudian, akan dibahas mengenai hasil yang diperoleh ketika mengimplementasikan fungsi hash yang telah dirancang di atas.

Ditemukan kelemahan dalam algoritma fungsi hash yang dibuat, yakni untuk panjang data 0 s.d. 12 karakter atau dengan kata lain 0 s.d. 96 bit panjang data, tidak bisa mengimplementasikan salah satu sifat fungsi hash yang menyebutkan penggantian 1 bit dapat membedakan jauh dengan data sebelum diganti. Jadi hasil yang didapat untuk setiap data dengan panjang 0 s.d. 96 bit, hasilnya akan sama dengan data dengan panjang yang sama pula. Misalnya string 'awewa' dengan 'awewe' akan menghasilkan fungsi hash yang sama.

Masalah tersebut terpecahkan ketika penulis mencoba mengubah *code* pada tahap 3 menjadi seperti di bawah ini.

```
/********TAHAP 3********/
/******* My Hash *******/

//  dengan mengubah inisialisasi
looping yang dilakukan, masalah yang
disebutkan sebelumnya menjadi
terselesaikan

for i from 0 to 47
        if i mod 4 = 0 then
            f = a xor c or (b and d)
        else if i mod 4 = 1 then
            f = (e and c) xor not(a)
        else if i mod 4 = 2 then
            f = (c and b) xor (a and
c) xor (b and e)
        else if i mod 4 = 3 then
            f = not(a or e)

        temp = e + w[i]
        e = c + f + f
        c = a
        a = d + f + w[i]
        d = b
        b = temp
```

Namun, sampai sekarang penulis masih bingung mengapa masalah tersebut dapat diselesaikan ketika mengikutkan 'data asli' ke dalam perhitungan pada tahap 3.

Berdasarkan analisis dari hasil yang dibuat oleh penulis, penulis mencoba membuat fungsi hash sederhana yang baru yang tidak mengikuti struktur dari SHA-1 maupun MD5, walaupun tetap menggunakan teknik *padding* dan menggunakan *buffer*.

Proses diawali dengan inisialisasi buffer dengan spesifikasi sama persis dengan tahap 1 pada penjelasan bab 4 di atas. Kemudian tetap menggunakan spesifikasi *padding* yang sudah ada agar panjang pesan menjadi kelipatan 512 bit. Proses pemecahan data menjadi 32 bit tetap dilakukan. Tahap berikutnya akan dijelaskan dalam pseudocode di bawah ini

```
/********TAHAP 2********/
/*Algoritma Fungsi Hash Sederhana*/

//  w[i],0≤i≤15  sebagai  pecahan-
pecahan 32bit

var tes = w[0]
for i from 1 to 15
      tes = tes xor w[i]
```

Kemudian setelah didapat angka di dalam variabel ‘tes’ tersebut. Dilakukan tahap 3 dengan spesifikasi di bawah ini.

```
/********TAHAP 3********/
/*Algoritma Fungsi Hash Sederhana*/

// variabel a, b, c, d, dan e
diinisialisasikan dengan nilai h0,
h1, h2, h3, dan h4 ditambahkan
dengan contoh
//blok512  adalah  pecahan-pecahan
data yang berukuran 512 bit

a = h0 + tes //lihat tahap 2 diatas
b = h1 + tes
c = h2 + tes
d = h3 + tes
e = h4 + tes

for i from 0 to blok512.length
      a = a*(i+1)
      b = b*(i+1)
      c = c*(i+1)
      d = d*(i+1)
      e = e*(i+1)
```

Setelah itu dilakukan tahap 4 seperti biasa, lalu digabungkan h0, h1, h2, h3 dan h4 untuk mendapatkan *message digest*-nya.

Hasilnya cukup memuaskan, malah masalah yang ditemui pada percobaan pertama tadi tidak ditemui. Namun, fungsi hash sederhana yang dibuat ini akan lebih mudah memberikan 2 data dengan *message digest* yang sama persis ketimbang algoritma SHA-1, MD5 maupun yang telah dibuat penulis sebelumnya.

Fungsi hash sederhana ini telah dicoba dengan sejumlah data uji berikut.

Data Uji	Hasil Hash
asdwe	A5E9A0A7D3B506532559EF8DE6421EF1036D0E18F
asdww	C6F92FD7DDB051DB36F3A770A3781E51848D0E18F
asdee	A5E9A086C425D6492A0E677C4C8A3C33CD6DAD97D
asdew	C6F92FB6CE2121D13BA81F5F09C03B944E8DAD97D
mau panjang	57A844BF71CA7CA4624FF9045E
cobain gimana yah enaknya panjangnya wah pusing lama lama	30BCEB118463DA
mau panjang cobain gimana yah enaknya panjangnya wah pusing lama lamm	9E2FDC9D64F3A64F3286D6A401FF9A4B298463DA

Dapat dilihat dengan keenam data uji tersebut bahwa fungsi hash sederhana yang telah dibuat penulis menghasilkan hasil yang berbeda satu dengan yang lainnya. Walaupun memang terlihat kemiripan hasil antara data uji pertama dengan ketiga, dan data uji kedua dengan keempat. Hal mungkin terjadi karena memang algoritma yang dibuat sederhana. Dan juga perbedaan yang terletak pada karakter ke-4 atau lebih spesifik bit ke 24 s.d. 31 menghasilkan *message digest* yang hampir mirip.

Jadi dalam fungsi hash sederhana yang dibuat, data asli hanya digunakan bit-bitnya untuk di xor-kan satu dengan yang lainnya. Lalu hasil operasi xor tersebut dijumlahkan dengan *buffer*, lalu dikalikan sebanyak iterasi yang dilakukan, kemudian dijumlahkan kembali dengan *buffer* awal sebelum diproses.

Tentang batasan program yang dibuat penulis, penulis tidak memasukkan operator *shift-left* ataupun *right* karena keterbatasan kemampuan penulis dalam meng-*coding* suatu program. Lalu, hasil *message digest* yang diperoleh seharusnya memiliki panjang 160 bit(32bit * 5), namun ada kalanya berukuran kurang dari 160 bit karena waktu menggabungkan kelima *buffer*, belum di cek apakah angka-angka awal pada *buffer* tersebut ‘0’ atau bukan.

6. KESIMPULAN

- Fungsi hash dapat dibuat dengan mengubah-ubah fungsi yang terdapat pada SHA-1 maupun MD5, karena struktur fungsi tersebut hanya berpengaruh pada ‘ke-*random*-an’ *message digest* yang didapat. Baik itu fungsi dalam tahap 2 maupun tahap 3.
- Begitu pula dengan inisialisasi dari angka-angka yang ada pada *buffer* dalam algoritma SHA-1 dan MD5, dapat diubah sesuka hati karena hanya berpengaruh pada hasil *message digest* yang didapat, tidak mempengaruhi sifat-sifat yang ada pada fungsi hash.
- Jika ada struktur dalam algoritma SHA-1 yang ikut diubah dalam pembuatan algoritma fungsi hash baru, akan menimbulkan masalah. Seperti yang sudah dijelaskan sebelumnya pada bab analisis.
- Fungsi hash juga dapat dibuat hanya dengan menggunakan operasi tambah, kali, dan xor. Seperti yang sudah dibuat pada algoritma fungsi hash sederhana di atas. Namun sudah pasti mempunyai lebih banyak kelemahan.

7. REFERENSI

[1] Munir, Rinaldi. *Diktat Kuliah IF5054*, Departemen Teknik Informatika, Institut Teknologi Bandung, 2005.

[2] http://en.wikipedia.org/wiki/SHA-1
Diakses tanggal 8 Mei 2011

[3] http://en.wikipedia.org/wiki/MD5
Diakses tanggal 8 Mei 2011

[4] http://stsn6.wordpress.com/2009/10/15/konsep-fungsi-hash-function-pada-aplikasi-kriptografi/
Diakses tanggal 8 Mei 2011

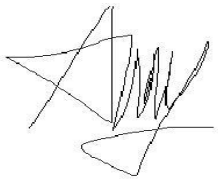
[5] Slide bahan kuliah IF3058 Kriptografi "Fungsi Hash". Rinaldi Munir, Teknik Informatika, STEI-ITB.

[6] Slide bahan kuliah IF3058 Kriptografi "Algoritma MD5". Rinaldi Munir, Teknik Informatika, STEI-ITB.

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, 23 Maret 2011



Kevin Chandra Irwanto
13508063