PENGUJIAN IMPLEMENTASI DAN PERBANDINGAN FUNGSI HASH SATU ARAH SHA-1 dan SHA-2

Ivan Nugraha – NIM : 13506073

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if16073@students.if.itb.ac.id

Abstrak

Komunikasi merupakan faktor penting dalam kehidupan manusia. Kini manusia semakin dipermudah oleh teknologi untuk menyampaikan informasi. Media–media komunikasi yang diciptakan manusia tersebut memang memudahkan penyampaian informasi, tapi di sisi lain penyampaian pesan melalui media tertentu tidak menjamin keamanan terhadap kerahasiaan pesan. Oleh karena itu dikembangkan ilmu kriptografi untuk menjaga keamanan informasi saat penyampaian melalui media informasi.

Dua tujuan terpenting dari sistem sekuriti adalah kerahasiaan dan integritas pesan. Namun, ada kalanya dimana kerahasiaan tidak menjadi fokus utama, tapi kita tetap menginginkan agar pesan yang dikirim melalui media internet tidak diubah oleh pihak lain. Dalam hal ini, fungsi hash dapat membantu dalam menjaga integritas pesan. Fungsi hash dapat menghitung nilai hash atau message digest dari pesan apapun yang diberikan sebagai input. Nilai message digest yang dihasilkan tidak dapat dikonstruksi kembali menjadi pesan semula, sehingga fungsi hash disebut juga fungsi searah. Keamanan dan integritas pesan dapat diverifikasi dengan fungsi hash ini. Ada berbagai jenis algoritma hash yang telah dibuat, diantaranya memiliki kelebihan dan kekurangan, baik dari sisi implementasi, kinerja, serta ukuran message digest yang dihasilkan. Makalah ini akan mencoba membandingkan tiga jenis algoritma hash yang telah distandardisasi oleh National Institute of Standards and Technology. Algoritma yang telah distandardisasi diantaranya SHA-1 dan SHA-2 (SHA-256 dan SHA-512).

Algoritma SHA-1 dapat diserang dan ditemukan collision dengan kurang dari 269 operasi. Collision adalah dua data pesan yang berbeda yang menghasilkan message digest yang sama. Oleh karena itu SHA-2 (SHA-256 dan SHA-512) dirancang untuk menutupi kekurangan yang dimiliki oleh SHA-1, dengan menambah jumlah putaran (loop) dalam algoritmanya dan meningkatkan panjang message digest yang dihasilkannya.

***Kata kunci**: Fungsi Hash, SHA-1, SHA-2, Message Digest*

1. Pendahuluan

Komunikasi merupakan faktor penting dalam kehidupan manusia. Kini manusia semakin dipermudah oleh teknologi untuk menyampaikan informasi. Media–media komunikasi yang diciptakan manusia tersebut memang memudahkan penyampaian informasi, tapi di sisi lain penyampaian pesan melalui media tertentu tidak menjamin keamanan terhadap kerahasiaan pesan. Oleh karena itu dikembangkan ilmu kriptografi untuk menjaga keamanan informasi saat penyampaian melalui media informasi.

SHA adalah fungsi hash satu-arah yang dibuat oleh NIST dan digunakan bersama DSS (Digital Signature Standard). Oleh NSA, SHA dinyatakan sebagai standard funsi hash satu arah. SHA dapat dianggap sebagai kelanjutan pendahulunya, MD5, yang telah digunakan secara luas. SHA disebut aman (secure) karena ia dirancang sedemikian rupa sehingga secara komputasi tidak mungkin menemukan pesan yang berkores[onden dengan message digest yang diberikan. [1]

SHA merupakan keluarga fungsi HASH satu arah. Fungsi hash SHA yang paling umum digunakan adalah SHA-1 yang telah diimplementasikan di dalam berbagai aplikasi dan protokol keamanan seperti TSS, SSL, PGP, SSH, S/MIME, dan Ipsec. Anggota pertama keluarga SHA adalah SHA-0 yang dipublikasikan pada tahun 1993.. SHA-0 sering diacu sebvagai SHA saja. Berikutnya pada tahun 1995 SHA-1 dipublkikasikan. Empat varianj lain juga telah dipublikasikan yaitu SHA-224, SHA-256, SHA-384, dan SHA-512. Keempat varian ini disebut sebagai SHA-2.

2. SHA-1

2.1 Algoritma SHA-1

Fungsi Hash SHA adalah lima fungsi hash kriptografis yang dirancang oleh National Security Agency (NSA) dan yang diterbitkan

oleh NIST sebagai U.S. Federal Information Processing Standard. SHA adalah kepanjangan dari Secure Hash Algorithm. Algoritma-algoritma hash menghitung suatu representasi digital dengan panjang yang pasti atau telah ditentukan (yang yang dikenal sebagai suatu message digest) dari suatu urutan data masukan (pesan) yang panjangnya bermacam-macam. Mereka disebut aman ketika, dapat dikomputasi untuk:

1. Menemukan suatu pesan yang berpasangan dengan suatu message digest yang diberi, atau

2. Menemukan dua pesan yang berbeda yang menghasilkan message digest yang sama.

Setiap perubahan akan, dengan kemungkinan yang sangat tinggi, menghasilkan suatu message digest yang berbeda."

SHA-1 menghasilkan suatu message digest dengan panjang 160 bit; nomor di dalam nama-nama empat algoritma yang lainnya menandakan panjangnya bit dari digest yang mereka hasilkan.

SHA-1 digunakan secara luas di dalam beberapa aplikasi-aplikasi keamanan dan protokol-protokol, termasuk TLS dan SSL, PGP, SSH, S/MIME, dan IPsec. SHA-1 dianggap sebagai pengganti lanjutan MD5, algoritma yang telah populer sebelumnya.

Keamanan SHA-1 telah dibicarakan oleh para ahli kriptografi. Meski tidak ada serangan-serangan telah diberitakan varian-varian SHA-2, mereka secara algoritma serupa dengan SHA-1, dengan demikian masih dilakukan usaha-usaha untuk meningkatkan algoritma fungsi hash alternatif.

Pseudocode untuk SHA-1 adalah sebagai berikut: [3]

```
Note 1: All variables are unsigned 32 bits and wrap modulo 2^32 when calculating
Note 2: All constants in this pseudo code are in big endian. Within each word, the most significant bit is stored in the leftmost bit position

Initialize variables:
h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0

Pre-processing:
append the bit '1' to the message
append k bits '0', where k is the minimum number ≥ 0 such that the resulting message
    length (in bits) is congruent to 448 (mod 512)
append length of message (before pre-processing), in bits, as 64-bit big-endian integer

Process the message in successive 512-bit chunks:
break message into 512-bit chunks
for each chunk
    break chunk into sixteen 32-bit big-endian words w[i], 0 ≤ i ≤ 15

    Extend the sixteen 32-bit words into eighty 32-bit words:
    for i from 16 to 79
        w[i] = (w[i-3] xor w[i-8] xor w[i-14] xor w[i-16]) leftrotate 1

    Initialize hash value for this chunk:
    a = h0
    b = h1
    c = h2
    d = h3
    e = h4

    Main loop:
    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

    Add this chunk's hash to
result so far:
    h0 = h0 + a
    h1 = h1 + b
    h2 = h2 + c
    h3 = h3 + d
    h4 = h4 + e

Produce the final hash value
(big-endian):
digest = hash = h0 append h1
append h2 append h3 append h4
```

2.2 Kriptanalisis SHA-1

Pada tahun 2005, Rijmen dan Oswald mempublikasiknan serangan pada versi SHA-1 yang direduksi (hanya menggunakan 53 putaran dari 80 putaran) dan menemukan kolisi dnegan kompleksitas sekitar 2^{80} operasi. [1]

Pada bulan Februari 2005, Xiayoun Wang, Yiqun Lisa Yin, dan Hongbo Yo mempublikasikan serangan yang dapat menemukan *collision* pada versi penuh SHA-1, yang membutuhkan sekitar 2^{69} operasi. [1]

Di dalam ilmu pengetahuan komputer, *hash collision* adalah suatu situasi yang terjadi ketika dua masukan yang berbeda pada suatu fungsi hash menghasilkan keluaran yang identik.

Semua fungsi hash berpotensial untuk menghasilkan hash collision, meskipun demikian fungsi hash yang dirancang dengan baik dapat mengurangi intensitas terjadinya hash collision (bandingkan dengan suatu fungsi dengan kurang baik merancang) atau lebih sulit untuk ditemukan. Didalam aplikasi-aplikasi yang menggunakan kemungkinan data input yang relatif adalah yang dikenal sebelum waktu yang ditetapkan dimungkinkan untuk membangun suatu fungsi hash yang sempurna dengan memetakan semua masukan pada keluaran-keluaran yang berbeda. Bagaimanapun, banyak sekali fungsi hash, termasuk fungsi hash yang sering digunakan dalam kriptografi, menghasilkan suatu keluaran ukuran yang ditetapkan dari satu pesan yang panjang. Dalam desain yang demikian, akan selalu ada *hash collision*, karena setiap fungsi hash yang diberi harus berpasangan dengan satu bilangan tak hingga dari masukan-masukan yang mungkin.

Contoh terjadinya *hash collision* adalah sebagai berikut:

$$f(x) = f(y) \; ; \; x \; != \; y$$

(untuk nilai masukan yang berbeda x dan y, fungsi hash f menghasilkankan nilai keluaran yang sama)

3. SHA-2

3.1 Algoritma SHA-256

Untuk menghindari kriptanalisis yang disebabkan oleh Collision pada SHA-1, maka dikembangkan algoritma-algoritma selanjutnya. Algoritma SHA generasi selanjutnya disebut juga algoritma SHA-2. Pada makalah ini akan dibahas salah satu dari algoritma SHA-2 yaitu SHA-256.

Algoritma SHA-256 dapat digunakan untuk menghitung nilai message digest dari sebuah pesan, dimana pesan tersebut memiliki panjang maksimum 2^{64} bit. Algoritma ini menggunakan sebuah message schedule yang terdiri dari 64 elemen 32-bit word, delapan buah variabel 32-bit, dan variabel penyimpanan nilai hash 8 buah word 32-bit. Hasil akhir dari algoritma SHA-256 adalah sebuah message digest sepanjang 256-bit.

Preprocessing dilakukan dengan menambahkan bit pengganjal, membagi-bagi pesan dalam block berukuran 512-bit, dan terakhir menginisiasi nilai hash awal. Proses penambahan bit pengganjal adalah sama dengan aturan penambahan bit pengganjal pada SHA-1.

Dalam proses komputasinya, SHA-256 menggunakan enam fungsi lojik, dimana setiap fungsi beroperasi mengguankan tiba buah word 32-bit (x,y, dan z) dan keluarannya berupa sebuah word 32-bit. Berikut ini adalah fungsi-fungsi dalah SHA-256:

$$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$
$$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\sum_0^{256}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$$
$$\sum_0^{256}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$$
$$\sigma_0^{256}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$$
$$\sigma_1^{256}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$$

Nilai hash awal pada algoritma SHA-256 adalah sebagai berikut:

```
h0 := 0xc1059ed8
h1 := 0x367cd507
h2 := 0x3070dd17
h3 := 0xf70e5939
h4 := 0xffc00b31
h5 := 0x68581511
h6 := 0x64f98fa7
```

```
h7 := 0xbefa4fa4
```

Dan konstanta dalam SHA-256 adalah sebagai berikut (64 buah):

```
428a2f98 71374491 b5c0fbcf e9b5dba5
3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3
72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc
2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7
c6e00bf3 d5a79147 06ca6351 14292967

27b70a85 2e1b2138 4d2c6dfc 53380d13
650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3
d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5
391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208
90befffa a4506ceb bef9a3f7 c67178f2
```

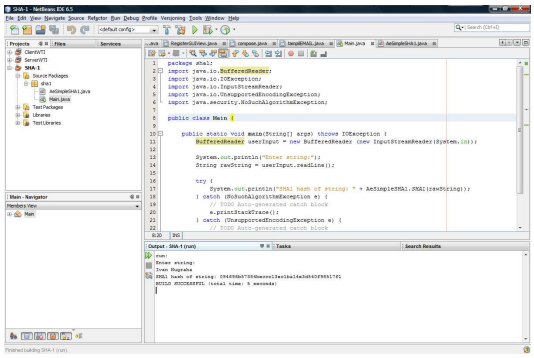
Berikut ini adalah pseudo-code dari algoritma SHA-256.

```
Note 1: All variables are
unsigned 32 bits and wrap
modulo 2^32 when calculating
Note 2: All constants in this
pseudo code are in big endian

Initialize variables
(first 32 bits of the
fractional parts of the square
roots of the first 8 primes
2..19):
h0 := 0x6a09e667
h1 := 0xbb67ae85
h2 := 0x3c6ef372
h3 := 0xa54ff53a
h4 := 0x510e527f
h5 := 0x9b05688c
h6 := 0x1f83d9ab
h7 := 0x5be0cd19

Initialize table of round
constants
(first 32 bits of the
fractional parts of the cube
roots of the first 64 primes
2..311):
k[0..63] :=
   0x428a2f98, 0x71374491,
0xb5c0fbcf, 0xe9b5dba5,
0x3956c25b, 0x59f111f1,
0x923f82a4, 0xab1c5ed5,
   0xd807aa98, 0x12835b01,
0x243185be, 0x550c7dc3,
0x72be5d74, 0x80deb1fe,
0x9bdc06a7, 0xc19bf174,
   0xe49b69c1, 0xefbe4786,
0x0fc19dc6, 0x240ca1cc,
0x2de92c6f, 0x4a7484aa,
0x5cb0a9dc, 0x76f988da,
   0x983e5152, 0xa831c66d,
0xb00327c8, 0xbf597fc7,
0xc6e00bf3, 0xd5a79147,
0x06ca6351, 0x14292967,
   0x27b70a85, 0x2e1b2138,
0x4d2c6dfc, 0x53380d13,
0x650a7354, 0x766a0abb,
0x81c2c92e, 0x92722c85,
   0xa2bfe8a1, 0xa81a664b,
0xc24b8b70, 0xc76c51a3,
0xd192e819, 0xd6990624,
0xf40e3585, 0x106aa070,
   0x19a4c116, 0x1e376c08,
0x2748774c, 0x34b0bcb5,
0x391c0cb3, 0x4ed8aa4a,
0x5b9cca4f, 0x682e6ff3,
   0x748f82ee, 0x78a5636f,
0x84c87814, 0x8cc70208,
0x90befffa, 0xa4506ceb,
0xbef9a3f7, 0xc67178f2

Pre-processing:
append the bit '1' to the
message
append k bits '0', where k is
the minimum number >= 0 such
that the resulting message
    length (in bits) is
congruent to 448 (mod 512)
append length of message
(before pre-processing), in
bits, as 64-bit big-endian
integer

Process the message in
successive 512-bit chunks:
break message into 512-bit
chunks
for each chunk
    break chunk into sixteen
32-bit big-endian words
w[0..15]

    Extend the sixteen 32-bit
words into sixty-four 32-bit
words:
    for i from 16 to 63
        s0 := (w[i-15]
rightrotate 7) xor (w[i-15]
rightrotate 18) xor (w[i-15]
rightshift 3)
        s1 := (w[i-2]
rightrotate 17) xor (w[i-2]
rightrotate 19) xor (w[i-2]
rightshift 10)
        w[i] := w[i-16] + s0 +
w[i-7] + s1
```

```
    Initialize hash value for
this chunk:
    a := h0
    b := h1
    c := h2
    d := h3
    e := h4
    f := h5
    g := h6
    h := h7

    Main loop:
    for i from 0 to 63
        s0 := (a rightrotate 2)
xor (a rightrotate 13) xor (a
rightrotate 22)
        maj := (a and b) xor (a
and c) xor (b and c)
        t2 := s0 + maj
        s1 := (e rightrotate 6)
xor (e rightrotate 11) xor (e
rightrotate 25)
        ch := (e and f) xor
((not e) and g)
        t1 := h + s1 + ch +
k[i] + w[i]

        h := g
        g := f
        f := e
        e := d + t1
        d := c
        c := b
        b := a
        a := t1 + t2

    Add this chunk's hash to
result so far:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d
    h4 := h4 + e
    h5 := h5 + f
    h6 := h6 + g
    h7 := h7 + h

Produce the final hash value
(big-endian):
digest = hash = h0 append h1
append h2 append h3 append h4
append h5 append h6 append h7
```

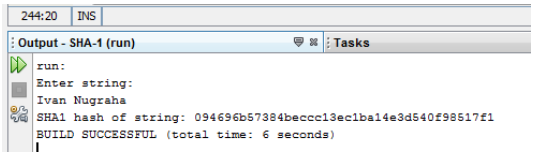
4. Pengujian

Pengujian dilakukan dengan memakai bahasa Java dan IDE Net Beans 6.5.

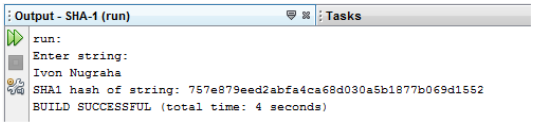


Pengujian pertama dengan SHA-1 menghasilkan hasil:

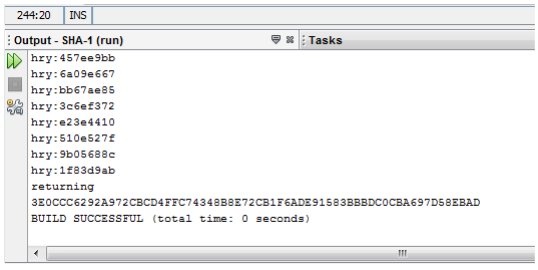
String : Ivan Nugraha
SHA1 hash of string:
094696b57384beccc13ec1ba14e3d540f98517f1



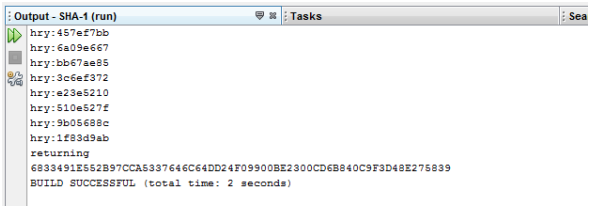
String: Ivon Nugraha
SHA1 hash of string:
757e879eed2abfa4ca68d030a5b1877b069d1552



Penguian kedua dengan SHA-256 menghasilkan hasil:
String : Ivan Nugraha
Sha256 hash of String:
3E0CCC6292A972CBCD4FFC74348B8E72CB1F6ADE91583BBBDC0CBA697D58EBAD



String: Ivon Nugraha:
Sha256 hash of String:
6833491E552B97CCA5337646C64DD24F09900BE2300CD6B840C9F3D48E275839



4.1 Analisis Pengujian

Kedua algoritma SHA tersebut sebenarnya sudah tergolong aman. Algoritma SHA dikatakan aman karena tidak mungkin secara komputasional untuk menemukan pesan yang berpasangan dengan sebuah message digest tertentu. Selain itu, tidak mungkin juga secara komputasional untuk menghasilkan dua pesan berbeda yang menghasilkan message digest yang sama. Sedikit perubahan pada pesan kemungkinan besar akan menghasilkan message digest yang sangat berbeda.

Keamanan algoritma SHA dibuktikan dengan *avalanche effect*, yaitu jika terjadi perubahan sedikit pada pesan, maka message digest yang dihasilkan dapat berbeda jauh. [2]

Perubahan 1 huruf pada kata Ivan menjadi Ivon mengubah hasil sangat jauh dari hasil awal.

5. Kesimpulan

Dari pembahasan di atas dapat ditarik kesimpulan:

1. Fungsi hash satu arah mengubah pesan yang panjang menjadi nilai hash, dan hampir tidak mungkin untuk mengkonstruksi pesan kembali dari nilai hash tersebut.
2. Fungsi SHA-1 dan SHA-256 memiliki karakteristik dan implementasi yang hampir mirip. Akan tetapi, ketiganya menggunakan konstanta yang berbeda, panjang variabel berbeda, fungsi yang berbeda, jumlah loop berbeda, serta menghasilkan nilai message digest yang panjangnya berbeda.
3. *Avalanche effect* dapat terjadi hanya dengan mengubah sedikit bagian dari pesan. Akan tetapi, algoritma SHA-1 masih dapat ditemui *collision*, sehingga dianggap masih kurang aman. SHA-256 dianggap lebih aman dibanding SHA-1 karena tingkat kompleksitas yang lebih tinggi dan *message* digest yang dihasilkan lebih panjang.

Daftar Referensi

[1] Munir, Rinaldi. (2004). Bahan Kuliah IF5054 Kriptografi. Departemen Teknik Informatika, Institut Teknologi Bandung.

[2] Packtizer, Secure Hashing Algorithm. http://www.packetizer.com/security/sha1/ , tanggal akses: 7 Mei 2009

[3] Eastlake, D., Jones, P. (2001). Request For Comment (RFC)-3174 : US Secure Hash Algorithm 1 (SHA1). The Internet Society. http://tools.ietf.org/html/rfc3174 , tanggal akses : 7 Mei 2009

[4] AnyExample.com http://www.anyexample.com/programming/java/java_simple_class_to_compute_sha_1_hash.xml , tanggal akses: 20 Mei 2009