Perbandingan HMAC SHA-512 dan HMAC RIPEMD-160 dengan Penggunaan Kunci Bilangan Acak

Reza Brianca Widodo / 13507013[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]if17013@students.if.itb.ac.id

***Abstrak*—Dalam perkembangan kemananan informasi, fungsi *hash* banyak berperan dalam menjamin integritas data. Hal ini sesuai dengan salah satu prinsip dari keamanan informasi itu sendiri. Salah satu cara untuk menjamin integritas data tersebut adalah dengan menggunakan fungsi *hash*. Fungsi ini mempunyai kelebihan karena memiliki sensitivitas tinggi terhadap perubahan data sekecil apapun. Akan tetapi fungsi ini belum memiliki kemampuan verifikasi yang juga diperlukan dalam keamanan informasi. Kemampuan ini dicoba ditangani oleh mekanisme HMAC. Mekanisme ini memerlukan kunci masukan sebelum fungsi *hash* dilaksanakan. Makalah ini membahas dua buah fungsi HMAC dengan fungsi hash yang terlibat yaitu SHA-512 dan RIPEMD-160. Modifikasi dilakukan dengan cara kunci masukan tidak diambil dari masukan pengguna tetapi diambil dari sebuah bilangan acak yang bersifat prima sepanjang 512 bit. Dari hasil pengujian didapatkan bahwa mekanisme HMAC dengan fungsi *hash* SHA-512 mempunyai waktu lebih lambat daripada mekanisme HMAC dengan fungsi *hash* RIPEMD-160. Hal ini dikarenakan SHA-512 menghasilkan keluaran yang lebih panjang dari RIPEMD-160 yaitu sepanjang 64 byte dimana fungsi RIPEMD-160 memiliki keluaran sepanjang 20 byte saja. Fungsi RIPEMD-160 juga belum ditemukan kolisinya sampai saat ini sehingga dapat dikatakan bahwa fungsi RIPEMD-160 lebih baik dari SHA-512. Di sisi lain, tingkat keamanan dari SHA-512 juga dapat lebih ditingkatkan karena fungsi ini mampu menerima masukan kunci sampai sepanjang 1024 bit.**

***Kata kunci*— bilangan acak, HMAC, RIPEMD-160, SHA-512, waktu**

I. PENDAHULUAN

Pada perkembangan teknologi saat ini, keamanan informasi telah menjadi sebuah kebutuhan wajib bagi siapapun pihak yang terlibat di dalamnya. Dari perkembangan tersebut, salah satu prinsip dalam keamanan informasi adalah integritas data. Artinya, sebuah data yang dikirimkan harus mampu dipastikan kembali keaslian dari data tersebut. Jika terdapat perubahan sedikit saja dalam data, penerima dapat langsung mengetahuinya. Dalam perkembangannya, fungsi tersebut dikenal sebagai fungsi *hash*. Fungsi ini adalah fungsi satu arah yang mampu mengubah pesan dengan ukuran besar menjadi hanya berukuran puluhan byte saja.

Pada saat ini, fungsi *hash* banyak berperan dalam menjamin integritas dari pesan yang dikirimkan. Hal ini disebabkan oleh sensitivitas fungsi tersebut terhadap perubahan data sekecil apapun. Akan tetapi, penggunaan fungsi *hash* juga memiliki kekurangan, salah satu diantaranya adalah tidak mampu menjamin pihak manakah yang telah mengirim pesan tersebut. Fungsi *hash* tidak menyimpan identitas dari pengirim pesan untuk keperluan verifikasi apabila diperlukan.

Maka dari itulah Hash-Message Authentication Code (HMAC) berkembang. Berbeda dengan fungsi *hash* sebelumnya, fungsi HMAC menghasilkan sebuah message *digest* yang nilainya bergantung dari kata kunci yang dipilih oleh pengguna. Tingkat keamanan ditentukan dari panjang kunci yang digunakan, semakin panjang kunci semakin baik. Fungsi HMAC terdiri dari dua jenis, yaitu HMAC berbasis CBC dan HMAC berbasis fungsi *hash*.

Dari beberapa pilihan fungsi *hash*, pada makalah ini pembahasan akan dibatasi pada dua jenis fungsi *hash* yaitu RIPEMD-160 dan SHA-512. Untuk meningkatkan keamanan maka kata kunci tidak diambil dari masukan pengguna namun memanfaatkan pembangkit bilangan acak sepanjang 512 bit. Makalah ini akan melakukan implementasi dan pengujian dari kedua metode tersebut. Kemudian, hasil dari pengujian akan dianalisis manakah metode yang mampu menghitung nilai *hash* lebih cepat. Tujuan utama dari pengujian adalah untuk membandingkan kecepatan penghitungan berbagai ukuran data dari kedua fungsi HMAC tersebut.

II. DASAR TEORI

A. HMAC

Keyed-hash based Message Authentication Code (HMAC) adalah sebuah kode otentikasi pesan yang menggunakan kunci kriptografi bersamaan dengan fungsi *hash*[3]. Proses yang terjadi dalam fungsi ini secara sederhana dapat dijelaskan sebagai berikut :

```
function hmac (key, message)
//jika ukuran kunci lebih besar dari
ukuran blok
if (length(key) > blocksize) then
key = hash(key)
end if
```

```
  //padding dengan 0 jika ukuran kunci lebih pendek dari ukuran blok, || adalah simbol penyambungan
  if (length(key) < blocksize) then
  key = key || [0x00 * (blocksize - length(key))]
  end if
  o_key_pad = [0x5c * blocksize] XOR key
  i_key_pad = [0x36 * blocksize] XOR key
  //hasil akhir
  return hash(o_key_pad || hash(i_key_pad || message))
end function
```

Fungsi HMAC akan mengambil nilai *hash* dari kunci apabila panjang kunci lebih besar dari ukuran blok dari fungsi hash yang digunakan. Jika panjang kunci lebih pendek dari panjang blok, HMAC akan melakukan *padding* kunci dengan bit `0` sampai panjang kunci sama dengan ukuran blok. Lalu terdapat variabel `o_key_pad` yang berisi hasil `XOR` antara kunci dengan konstanta `0x5c5c5c5c5c`… seukuran panjang blok. Variabel terakhir yaitu `i_key_pad` berisi hasil `XOR` antara kunci dengan konstanta `0x36363636`… seukuran panjang blok.

Nilai dari `i_key_pad` akan disambung dengan pesan untuk kemudian dihitung nilai *hash*-nya. Hasil *hash* ini akan disambung dengan `o_key_pad`. Hasil akhir fungsi HMAC adalah nilai *hash* dari penyambungan tersebut.

B. SHA-512

Fungsi *hash* SHA-512 adalah fungsi untuk menghasilkan *message digest* dengan ukuran 512 bit dan panjang blok 1024 bit. Berbeda dengan SHA-1, fungsi ini menggunakan 64-bit konstanta yang berbeda dalam setiap putaran. Terdapat 80 putaran dalam fungsi ini. Untuk melakukan padding bit masih dilakukan dengan cara yang sama dengan SHA-1, hanya saja ukuran blok menjadi 1024 bit, bukan 512 bit. Daftar konstanta setiap putaran dapat dilihat pada referensi poin [2]. Fungsi yang terlibat dalam setiap putarannya adalah sebagai berikut :

1. Penjadwalan pesan

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{512\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{512\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 79 \end{cases}$$

2. Inisialisasi

$$a = H_0^{(i-1)}$$
$$b = H_1^{(i-1)}$$
$$c = H_2^{(i-1)}$$
$$d = H_3^{(i-1)}$$
$$e = H_4^{(i-1)}$$
$$f = H_5^{(i-1)}$$
$$g = H_6^{(i-1)}$$
$$h = H_7^{(i-1)}$$

3. Fungsi setiap putaran

$$T_1 = h + \sum_1^{\{512\}}(e) + Ch(e,f,g) + K_t^{\{512\}} + W_t$$
$$T_2 = \sum_0^{\{512\}}(a) + Maj(a,b,c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

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

$$\sum_0^{\{512\}}(x) = ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x)$$
$$\sum_1^{\{512\}}(x) = ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x)$$
$$\sigma_0^{\{512\}}(x) = ROTR^{1}(x) \oplus ROTR^{8}(x) \oplus SHR^{7}(x)$$
$$\sigma_1^{\{512\}}(x) = ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^{6}(x)$$

4. Hitung nilai *hash* sementara

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$
$$H_2^{(i)} = c + H_2^{(i-1)}$$
$$H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)}$$
$$H_5^{(i)} = f + H_5^{(i-1)}$$
$$H_6^{(i)} = g + H_6^{(i-1)}$$
$$H_7^{(i)} = h + H_7^{(i-1)}$$

5. Nilai hash akhir

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}$$

Dengan panjang blok yang lebih besar maka ukuran *file* yang dapat diproses juga menjadi lebih besar. Selain itu, metode ini juga dianggap masih aman dari peluang ditemukannya kolisi, tidak seperti SHA-1.

C. RIPEMD-160

Metode ini merupakan pengembangan lebih lanjut dari

fungsi *hash* RIPEMD-128. Berbeda dengan keluarga fungsi SHA, metode ini melibatkan seleksi dari penjadwalan pesan dan rotasi yang dilakukan juga berbeda tergantung dari putaran saat itu.

1. Fungsi f setiap putaran

$$f(j,x,y,z) = x \oplus y \oplus z \qquad (0 \le j \le 15)$$
$$f(j,x,y,z) = (x \wedge y) \vee (\neg x \wedge z) \qquad (16 \le j \le 31)$$
$$f(j,x,y,z) = (x \vee \neg y) \oplus z \qquad (32 \le j \le 47)$$
$$f(j,x,y,z) = (x \wedge z) \vee (y \wedge \neg z) \qquad (48 \le j \le 63)$$
$$f(j,x,y,z) = x \oplus (y \vee \neg z) \qquad (64 \le j \le 79)$$

2. Konstanta tambahan

$$K(j) = \texttt{00000000}_{\texttt{x}} \qquad (0 \le j \le 15)$$
$$K(j) = \texttt{5A827999}_{\texttt{x}} \qquad (16 \le j \le 31)$$
$$K(j) = \texttt{6ED9EBA1}_{\texttt{x}} \qquad (32 \le j \le 47)$$
$$K(j) = \texttt{8F1BBCDC}_{\texttt{x}} \qquad (48 \le j \le 63)$$
$$K(j) = \texttt{A953FD4E}_{\texttt{x}} \qquad (64 \le j \le 79)$$
$$K'(j) = \texttt{50A28BE6}_{\texttt{x}} \qquad (0 \le j \le 15)$$
$$K'(j) = \texttt{5C4DD124}_{\texttt{x}} \qquad (16 \le j \le 31)$$
$$K'(j) = \texttt{6D703EF3}_{\texttt{x}} \qquad (32 \le j \le 47)$$
$$K'(j) = \texttt{7A6D76E9}_{\texttt{x}} \qquad (48 \le j \le 63)$$
$$K'(j) = \texttt{00000000}_{\texttt{x}} \qquad (64 \le j \le 79)$$

3. Seleksi penjadwalan pesan

$$r(j) = j \qquad (0 \le j \le 15)$$
$$r(16..31) = 7,4,13,1,10,6,15,3,12,0,9,5,2,14,11,8$$
$$r(32..47) = 3,10,14,4,9,15,8,1,2,7,0,6,13,11,5,12$$
$$r(48..63) = 1,9,11,10,0,8,12,4,13,3,7,15,14,5,6,2$$
$$r(64..79) = 4,0,5,9,7,12,2,10,14,1,3,8,11,6,15,13$$
$$r'(0..15) = 5,14,7,0,9,2,11,4,13,6,15,8,1,10,3,12$$
$$r'(16..31) = 6,11,3,7,0,13,5,10,14,15,8,12,4,9,1,2$$
$$r'(32..47) = 15,5,1,3,7,14,6,9,11,8,12,2,10,0,4,13$$
$$r'(48..63) = 8,6,4,1,3,11,15,0,5,12,2,13,9,7,10,14$$
$$r'(64..79) = 12,15,10,4,1,5,8,7,6,2,13,14,0,3,9,11$$

4. Jumlah pergeseran tiap putaran

$$s(0..15) = 11,14,15,12,5,8,7,9,11,13,14,15,6,7,9,8$$
$$s(16..31) = 7,6,8,13,11,9,7,15,7,12,15,9,11,7,13,12$$
$$s(32..47) = 11,13,6,7,14,9,13,15,14,8,13,6,5,12,7,5$$
$$s(48..63) = 11,12,14,15,14,15,9,8,9,14,5,6,8,6,5,12$$
$$s(64..79) = 9,15,5,11,6,8,13,12,5,12,13,14,11,8,5,6$$
$$s'(0..15) = 8,9,9,11,13,15,15,5,7,7,8,11,14,14,12,6$$
$$s'(16..31) = 9,13,15,7,12,8,9,11,7,7,12,7,6,15,13,11$$
$$s'(32..47) = 9,7,15,11,8,6,6,14,12,13,5,14,13,13,7,5$$
$$s'(48..63) = 15,5,8,11,14,14,6,14,6,9,12,9,12,5,15,8$$
$$s'(64..79) = 8,5,12,9,12,5,14,6,8,13,6,5,15,13,11,11$$

5. Fungsi utama

for $i := 0$ *to* $t-1$ {
  $A := h_0$; $B := h_1$; $C := h_2$; $D = h_3$; $E = h_4$;
  $A' := h_0$; $B' := h_1$; $C' := h_2$; $D' = h_3$; $E' = h_4$;
  for $j := 0$ *to* 79 {
    $T := \mathrm{rol}_{s(j)}(A \boxplus f(j,B,C,D) \boxplus X_i[r(j)] \boxplus K(j)) \boxplus E$;
    $A := E$; $E := D$; $D := \mathrm{rol}_{10}(C)$; $C := B$; $B := T$;
    $T := \mathrm{rol}_{s'(j)}(A' \boxplus f(79-j,B',C',D') \boxplus X_i[r'(j)] \boxplus K'(j)) \boxplus E'$;
    $A' := E'$; $E' := D'$; $D' := \mathrm{rol}_{10}(C')$; $C' := B'$; $B' := T$;
  }
  $T := h_1 \boxplus C \boxplus D'$; $h_1 := h_2 \boxplus D \boxplus E'$; $h_2 := h_3 \boxplus E \boxplus A'$;
  $h_3 := h_4 \boxplus A \boxplus B'$; $h_4 := h_0 \boxplus B \boxplus C'$; $h_0 := T$;
}

Hasil akhir dari fungsi hash ini adalah penggabungan dari h0 hingga h4.

III. IMPLEMENTASI

Untuk melakukan pengujian terhadap kinerja dari kedua metode, terlebih dahulu dilakukan implementasi. Proses implementasi dilakukan dengan spesifikasi sebagai berikut :

1. Kakas pengembangan menggunakan Microsoft Visual Studio 2008
2. Kelas BigInteger mengacu pada [3].
3. Prosessor Intel Centrino Duo 1.6 GHz
4. RAM DDR2 1280 MB
5. Sistem Operasi Microsoft Windows XP SP3

Implementasi dibatasi hanya untuk melakukan proses penghitungan nilai *hash* dari sebuah *file* dan menghitung waktu total yang diperlukan. Pertama, pengguna memilih *file* yang akan dihitung nilai HMAC-nya. Kedua, pengguna membangkitkan bilangan acak sepanjang 512 bit sebagai kunci rahasia HMAC. Terakhir, pengguna dapat melakukan penghitungan nilai HMAC dari masukan yang telah diberikan sebelumnya. Penghitungan selesai saat *progress bar* sampai ke tahap maksimal dan nilai dari waktu yang diperoleh muncul di program.

Proses yang terjadi di dalam program adalah sebagai berikut :

1. Membaca setiap byte dari *file*
2. Untuk metode SHA-512, terdapat penambahan padding bit untuk kunci agar panjang kunci menjadi 1024 bit
3. Untuk metode RIPEMD-160 tidak perlu dilakukan penambahan padding karena panjang kunci telah sama dengan ukuran blok
4. Menghitung nilai variabel `o_key_pad` dan `i_key_pad`
5. Melakukan penyambungan antara `i_key_pad` dengan pesan
6. Menghitung nilai *hash* dari langkah 5
7. Melakukan penyambungan `o_key_pad` dengan hasil dari langkah 6
8. Menghitung nilai *hash* dari langkah 7

Penghitungan waktu total adalah penjumlahan antara waktu yang diperlukan pada langkah 5 dan waktu pada langkah 8. Hal ini dilakukan agar performa yang dilakukan benar-benar terukur dari kecepatan dua buah fungsi tersebut dalam melakukan penghitungan nilai *hash* dari file dalam mekanisme HMAC, tidak sebatas pada penghitungan nilai *hash* untuk hasil akhir *digest*-nya saja.

Nilai HMAC dari pesan masukan adalah hasil dari

langkah 8. Panjang nilai HMAC tergantung dari fungsi *hash* yang digunakan. Pada SHA-512 hasilnya sepanjang 512 bit, sedangkan pada RIPEMD-160 hasilnya sepanjang 160 bit.

Pada program ini pengguna tidak dapat menyimpan hasil penghitungan nilai *hash* maupun kunci. Hal ini disebabkan tujuan pembuatan program adalah untuk menghitung waktu dari proses yang dilakukan oleh kedua metode tersebut dalam menghasilkan nilai *hash* dari sebuah *file*.

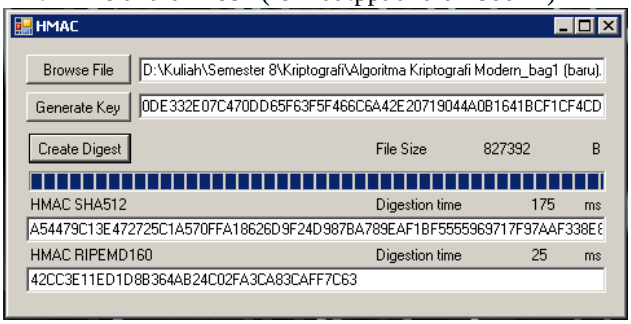
IV. PENGUJIAN

Pengujian dilakukan dengan menggunakan tiga buah *file* dengan ukuran kecil (800kB), ukuran sedang (9MB), dan ukuran besar (81 MB). Ukuran *file* dibatasi karena dalam kenyataannya, *file* yang dikirim sebagai *attachment e-mail* atau *e-mail* yang hanya berupa teks saja besarnya tidak dapat lebih dari 25 MB. Pengujian dilakukan dengan kunci yang sama, yaitu bilangan acak yang bersifat prima sepanjang 512 bit. Kunci yang digunakan dalam pengujian ini adalah :

D76C75D591801763E4E82A8BCD8B5C41C626A
F8F99D2738BDE7A2D019B1DC07F492F06760DE
332E07C470DD65F63F5F466C6A42E20719044A
0B1641BCF1CF4CD

Hasil pengujian program ini adalah sebagai berikut :

1. File ukuran kecil (format .ppt ukuran 800kB)



Gambar 1. Pengujian dengan file ukuran 800kB

Pada gambar 1 terlihat bahwa HMAC dengan metode SHA-512 memerlukan waktu 175 ms untuk melakukan penghitungan nilai hash, sedangkan HMAC dengan metode RIPEMD-160 hanya memerlukan waktu 25 ms (7 kali lebih cepat).

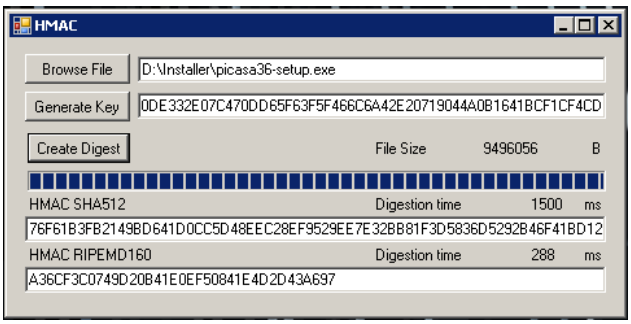
Nilai HMAC dengan metode SHA-512 adalah sebagai berikut :

A54479C13E472725C1A570FFA18626D9F24D9
87BA789EAF1BF5555969717F97AAF338E83DE9
F9BFE1CF2E79D2CF7CA16DB603CB1A8CA28ED8
A43B952E953

Nilai HMAC dengan metode RIPEMD-160 adalah sebagai berikut :

42CC3E11ED1D8B364AB24C02FA3CA83CAFF7C
63

2. File ukuran sedang (format .exe ukuran 9 MB)



Gambar 2. Pengujian dengan file ukuran 9 MB

Pada gambar 2 terlihat bahwa HMAC dengan metode SHA-512 memerlukan waktu 1500 ms untuk melakukan penghitungan nilai hash, sedangkan HMAC dengan metode RIPEMD-160 hanya memerlukan waktu 288 ms (sekitar 5 kali lebih cepat).

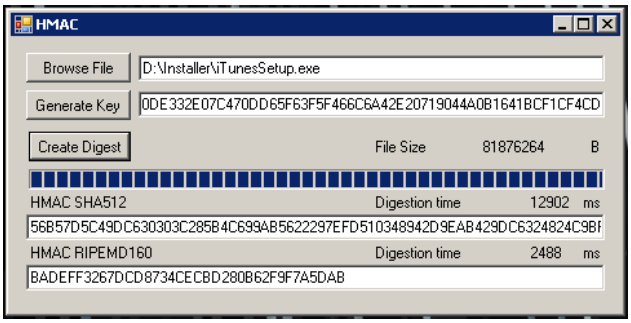
Nilai HMAC dengan metode SHA-512 adalah sebagai berikut :

76F61B3FB2149BD641D0CC5D48EEC28EF9529
EE7E32BB81F3D5836D5292B46F41BD128BA1AD
0B2BD85135282A2EE94FB2DC11DA5AB668FA4A
FE86DDDF8C8B4DC

Nilai HMAC dengan metode RIPEMD-160 adalah sebagai berikut :

A36CF3C0749D20B41E0EF50841E4D2D43A697

3. File ukuran besar (format .exe ukuran 81 MB)



Gambar 3. Pengujian dengan file ukuran 81 MB

Pada gambar 3 terlihat bahwa HMAC dengan metode SHA-512 memerlukan waktu 12902 ms untuk melakukan penghitungan nilai hash, sedangkan HMAC dengan metode RIPEMD-160 hanya memerlukan waktu 2488 ms (sekitar 5 kali lebih cepat).

Nilai HMAC dengan metode SHA-512 adalah sebagai berikut :

56B57D5C49DC630303C285B4C699AB5622297
EFD510348942D9EAB429DC6324824C9BF23042
1509CE2FD888A512EB64A2FAD7DA45D441816C
1FCD4EB387

Nilai HMAC dengan metode RIPEMD-160 adalah sebagai berikut :

BADEFF3267DCD8734CECBD280B62F9F7A5DAB

V. Analisis Hasil Pengujian

Dari penjelasan kedua metode diatas, terlihat bahwa SHA-512 memiliki tingkat keamanan yang lebih tinggi jika dilihat berdasarkan panjang keluaran yang dihasilkan. SHA-512 memiliki ukuran 512 bit (64 byte) sedangkan RIPEMD-160 memiliki ukuran 160 bit (20 byte). Hal ini disebabkan karena semakin panjang ukuran *digest* maka semakin sulit ditemukan kolisi, yaitu kondisi dimana dua buah pesan yang berbeda memiliki nilai hash yang sama.

Pada metode HMAC, nilai hash dari sebuah pesan bergantung pada kunci masukan pengguna. Karena kunci yang digunakan adalah kunci tunggal, tidak diperlukan adanya manajemen kunci seperti pada infrastruktur kunci publik. Prinsip kriptografi yang digunakan adalah algoritma kunci simetri.

Tingkat keamanan pada HMAC bertumpu pada kekuatan kunci. Maka dari itulah, pengguna perlu memasukkan kunci dengan ukuran panjang pesan tidak mudah ditembus. Di sisi lain, kunci dengan ukuran besar akan sulit diingat oleh pengguna, maka pemanfaatan bilangan acak sebagai kunci dapat dimanfaatkan sebagai faktor yang meningkatkan keamanan dari pesan itu sendiri. Pada pengujian ini digunakan panjang kunci sepanjang 512 bit (64 byte) dan kunci tersebut adalah bilangan prima.

Dari pengujian yang telah dilakukan, terlihat bahwa mekanisme HMAC dengan penggunaan fungsi hash RIPEMD-160 mampu mengeluarkan hasil digest dengan lebih cepat. Selisih kecepatan juga berbeda jauh terutama pada pengujian dengan file berukuran kecil. Hal ini dapat dilihat pada hasil pengujian pertama yang menunjukkan RIPEMD-160 mampu melakukan penghitungan nilai hash sampai 7 kali lebih cepat.. Pada ukuran file yang lebih besar, selisih kecepatan juga besar walaupun tidak sebesar hasil pengujian dengan file ukuran kecil. Hal ini terlihat dari hasil pengujian kedua dan ketiga yang masing-masing menghasilkan perbedaan waktu penghitungan nilai hash sampai sekitar 5 kali lebih cepat.

Akan tetapi, tingkat keamanan dari HMAC SHA-512 mampu lebih unggul daripada HMAC RIPEMD-160. Hal ini disebabkan ukuran keluaran dari fungsi SHA-512 adalah sepanjang 64 byte. Walaupun sampai saat ini belum ada laporan yang menunjukkan adanya kolisi dari fungsi hash RIPEMD-160 dengan panjang keluaran 20 byte, fungsi SHA-512 tidak akan ditemukan kolisinya lebih cepat dari fungsi RIPEMD-160.

Berdasarkan teori dari setiap metode, terlihat bahwa SHA-512 juga mampu meningkatkan keamanannya lebih baik lagi. Hal ini disebabkan panjang kunci untuk fungsi ini dapat mencapai 1024 bit sesuai dengan teori HMAC yang menggunakan kunci sepanjang ukuran blok masukan fungsi hash yang digunakan. Jadi walaupun fungsi SHA-512 berjalan lebih lambat dari RIPEMD-160, tingkat keamanan dari mekanisme HMAC dengan fungsi ini lebih tinggi karena alasan tersebut.

VI. SIMPULAN

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan, terdapat beberapa hal yang dapat disimpulkan dari perbandingan mekanisme HMAC dengan dua buah fungsi hash SHA-512 dan RIPEMD-160. Kesimpulan dari hasil pengujian adalah sebagai berikut :

1. Mekanisme HMAC mampu melakukan verifikasi dari pesan yang dikirim tanpa perlu adanya mekanisme manajemen kunci.
   Hal ini disebabkan kunci yang terlibat dalam verifikasi sama dengan kunci yang digunakan untuk menghitung nilai hash dari file tersebut. Pada kernyataannya, mekanisme ini lebih mudah karena hanya menerapkan algoritma fungsi simetri dan tidak melibatkan proses komputasi yang rumit seperti pada algoritma kunci publik.
2. Fungsi RIPEMD-160 mampu menghitung nilai hash yang lebih cepat daripada fungsi SHA-512.
   Hal ini disebabkan fungsi RIPEMD-160 memiliki keluaran sepanjang 20 byte sehingga proses yang dilakukan di dalamnya lebih pendek dari fungsi SHA-512.
3. Fungsi SHA-512 memiliki tingkat keamanan yang lebih tinggi daripada fungsi RIPEMD-160.
   Hal ini disebabkan fungsi SHA-512 mempunyai keluaran sepanjang 64 byte sehingga sampai sekarang kolisi dari fungsi ini masih belum ditemukan. Walaupun RIPEMD-160 juga belum ada kolisinya, panjang keluaran 20 byte dari fungsi akan membuat kolisi dari RIPEMD-160 akan lebih cepat diketahui daripada SHA-512.
4. Tingkat keamanan SHA-512 dapat lebih ditingkatkan.
   Hal ini disebabkan dalam mekanisme HMAC, fungsi SHA-512 mampu menerima masukan kunci sampai sepanjang 1024 bit sesuai dengan panjang blok masukannya. RIPEMD-160 tidak mampu menerima masukan kunci lebih besar dari 512 bit karena ukuran blok masukan dari fungsi ini hanya sepanjang 512 bit.
5. Penggunaan bilangan acak sebagai kunci dapat menjadi alternatif dalam mengamankan pesan.
   Bilangan acak berukuran besar (512 bit) dan bersifat prima sangat sulit untuk ditemukan dengan mudah jika hanya dengan komputasi biasa. Di sisi lain, penggunaan kunci dengan ukuran besar juga sulit diingat oleh pengguna sehingga perlu disapkan mekanisme penyimpanan kunci agar kunci juga dapat diketahui oleh penerima pesan yang diinginkan.

Daftar Referensi

[1] Dobbertin, Hans, Bosselaers, Anton, Preneel, Bart. 1996. *RIPEMD-160 : A Strengthened Version of RIPEMD*. German Information Security Agency and Katholieke Universiteit Leuven, ESAT-COSIC.

[2] Federal Information Processing Standard Publication. 2002. *Secure Hash Standard*. National Institute Standards and Technology.

[3] Federal Information Processing Standard Publication. 2002. *The Keyed-Hash Message Authentication Code (HMAC)*. National Institute Standards and Technology.
[4] http://www.codeproject.com/KB/cs/biginteger.aspx

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, 29 Mei 2011

ttd

Reza Brianca Widodo / 13507013