STUDI, ANALISIS, DAN IMPLEMENTASI FUNGSI HASH RIPEMD-160

Ferry Mulia – NIM : 13506047

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if16047@students.if.itb.ac.id

Abstrak

Fungsi *hash* merupakan suatu *tool* yang penting dalam aplikasi kriptografi seperti sidik jari digital dari pesan, otentifikasi pesan, dan penurunan kunci. Selama lima tahun belakangan ini, beberapa perangkat lunak yang dapat melakukan fungsi hash dengan cepat telah ditemukan, dan sebagian besar didesain dengan menggunakan prinsip algoritma MD4 milik Ron Rivest. Salah satu diantaranya adalah RIPEMD, yang dibangun pada framework RIPE (*Race Integrity Primitives Evaluation*) dari proyek EU.

RIPEMD-160 adalah suatu fungsi hash kriptografi yang cepat yang dirancang untuk implementasi pada perangkat lunak dengan arsitektur 32 bit. Desain utama dari fungsi hash ini ada dua proses komputasional yang berbeda dan saling independen, dimana hasil dari kedua proses ini akan digabungkan pada akhir perhitungan dengan sebuah fungsi kompresi.

Sesuai dengan namanya RIPEMD-160 menghasilkan 160 bit. Hal ini dimaksudkan untuk menyediakan level sekuritas yang lebih tinggi untuk 10 tahun yang akan dating.

Selain itu, pada makalah ini juga akan dibahas mengenai implementasi dari fungsi hash ini sendiri, dan perbandingannya dengan beberapa algoritma yang juga berbasis algoritma MD4.

***Kata kunci* :**.*hash function, one-way hash function, message digest, shift-left* .

1. Pendahuluan

Fungsi *hash* adalah suatu fungsi yang memetakan suatu *string* dalam bentuk *bit* dengan panjang yang berbeda-beda ke dalam suatu *string* dengan panjang yang tetap.

Secara umum, persamaan fungsi *hash* adalah

$$h = H(M)$$

di mana M = pesan dengan panjang sembarang
h = nilai *hash* (*hash value*) atau pesan-ringkas (*message-digest)*

Fungsi *hash* akan mengembalikan suatu nilai *hash value* yang panjangnya jauh lebih pendek dibandingkan dengan panjang *string* masukan. Sebagai contoh, misalnya pesan yang ingin dicari nilai *hash*-nya memiliki ukuran sebesar 1 MB, maka nilai *hash value* yang dihasilkan hanya 128 bit. Perbedaan yang sangat signikan.

Selain itu, fungsi *hash* juga memiliki beberapa nama lain antara lain : fungsi kompresi (*compression function*), cetak-jari (*fingerprint*, tidak ada dua pesan yang berbeda yang menghasilkan nilai *hash* yang sama), *cryptographic checksum*, *message integrity check* (*MIC*), dan *manipulation detection code* (*MDC*).

Kebanyakan fungsi *hash* yang ada saat ini berupa fungsi *hash* satu-arah. Fungsi *hash* ini hanya bekerja satu arah, yang artinya pesan yang sudah diubah menjadi *message digest* tidak dapat dikembalikan lagi menjadi pesan semula (*irreversible*). Selain itu, fungsi *hash* satu-arah mempunyai sifat sebagai berikut:

1. Fungsi *H* dapat diterapkan pada blok data berukuran berapa saja.
2. *H* menghasilkan nilai (*h*) dengan panjang tetap (*fixed-length output*).
3. *H(x)* mudah dihitung untuk setiap nilai *x* yang diberikan.
4. Untuk setiap *h* yang dihasilkan, tidak mungkin dikembalikan nilai *x* sedemikian sehingga $H(x) = h$.
5. Untuk setiap *x* yang diberikan, tidak mungkin mencari $y \neq x$ sedemikian sehingga $H(y) = H(x)$.

6. Tidak mungkin mencari pasangan *x* dan *y* sedemikian sehingg *H(x) = H(y)*.

Fungsi *hash* yang paling popular, yang saat ini digunakan secara luas adalah fungsi *hash* yang dikembangkan dari fungsi *hash* MD4. MD4 diperkenalkan pada tahun 1990 oleh R, yang merupakan sebuah fungsi *hash* yang sangat cepat yang diintegrasikan ke dalam prosesor 32-bit. Fungsi *hash* ini ternyata memiliki kelemahan. Oleh karena itu, setahun kemudian dikembangkanlah fungsi MD5 untuk menutupi kelemahan pendahulunya. Dan disusul dengan banyaknya fungsi *hash* yang diturunkan dari MD4.

Konsorsium RIPE (*Race Integrity Primitives Evaluation*) kemudian mengajukan suatu portofolio mengenai fungsi *hash* yang telah diperkuat, yang disebut dengan nama RIPEMD. Inti dari fungsi ini adalah dua versi paralel dari MD4, dengan beberapa peningkatan pada pergeseran bit dan urutan karakter pada nilai *hash* yang dihasilkan.

Fungsi *hash* ini kemudian diperkuat lagi pada tahun April 1996 oleh Hans Dobbertin, Antoon Bosselaers, dan Bart Preneel. Fungsi *hash* ini termasuk ke dalam International Standard ISO/IEC 10118-3 pada akhir tahun 1997. Terdapat dua varian fungsi RIPEMD ini yaitu RIPEMD-128 dan RIPEMD-160. Akan tetapi, yang akan dibahas dalam makalah ini adalah RIPEMD-160.

2. RIPEMD-160

Sama halnya dengan varian MD4 lainnya, RIPEMD-160 beroperasi pada prosesor 32-bit. Operasi dasar pada fungsi ini adalah sebagai berikut:

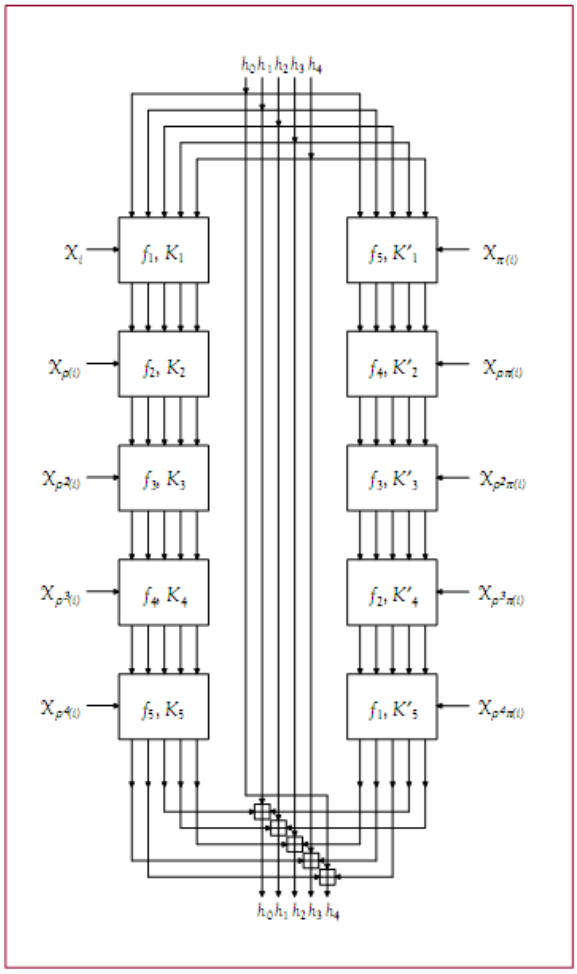
- rotasi kiri (*left-rotation* atau *left-spin*) dari pesan masukan;
- operasi *bitwise Boolean* (AND, NOT, OR, exclusive-OR);
- penambahan dua buah *string* sepanjang modulo 2^{32} pada nilai *hash*.

RIPEMD-160 melakukan kompresi terhadap pesan dengan membaginya ke dalam blok data yang masing-masing panjangnya 512 bit. Setiap blok kemudian akan dibagi lagi ke dalam 16 buah *string* dengan panjang 4-byte, dan setiap *string* ini akan dikonversikan menjadi sebuah kata sepanjang 32-bit dengan menggunakan konvensi *little-endian*.

Untuk menjamin panjang total dari pesan masukan merupakan kelipatan dari 512 bit, maka pesan masukan terlebih dahulu ditambahkan (*padding*) sejumlah bit dengan panjang tertentu. Serupa dengan varian MD4 lainnya, bit pengganjal (*padding bits*) ini berupa sebuah angka 1 yang diikuti oleh sejumlah angka 0 (dengan jumlah angka 0 lebih besar dari 0 dan lebih kecil dari 511). 64 bit terakhir dari perpanjangan masukan ini merupakan representasi bit dari ukuran panjang input, dengan *least significant byte* terletak di awal.

Hasil fungsi *hash* RIPEMD-160 terdapat dalam lima buah pesan yang masing-masing panjangnya 32-bit, yang juga membentuk *internal state* dari algoritma ini. Hasil akhir dari lima pesan ini diubah menjadi sebuah *string* sepanjang 160-bit, sekali lagi menggunakan konvensi *little-endian*.

Status dari proses dalam algoritma ini diinisialisasi dengan sebuha himpunan dari 5 buah pesan masing-masing sepanjang 32-bit, sebagai nilai inisialisasi. Bagian terpenting dari algoritma ini dikenal dengan fungsi kompresi (*compression function*); fungsi ini menghasilkan *state* baru dari *state* sebelumnya dan blok pesan masukan. Jumlah total langkah adalah $5 \times 6 \times 2 = 160$, bandingkan dengan $3 \times 16 = 48$ pada MD4 dan $4 \times 16 = 64$ pada MD5. Pertama, dua duplikat (*copy*) dibuat dari *state* yang lama (lima buah *registeri* kanan dan kiri masing-masing sepanjang 32-bit). Masing-masing bagian kanan kiri diproses secara independen. Tiap langkah akan mengubah satu dari *register-register* tersebut berdasarkan empat *register* lainnya dan sebuah blok pesan. Di akhir fungsi kompresi, algoritma ini mengkalkulasikan *state* baru dengan menambahkan masing-masing kata dari *state* sebelumnya sebuah *register* dari bagian kiri dan sebuah *register* dari bagian kanan.



Garis besar fungsi kompresi dari RIPEMD-160

2.1. Operasi pada setiap langkah. A := (A = *f* (B,C,D) + X + K)$^{<<s}$ + E dan C := C$^{<<10}$. Di sini $^{<<s}$ melambangkan operasi pergeseran bit (rotasi) sebanyak s bit.

2.2. Penyusunan urutan pesan. Fungsi *hash* ini menggunakan permutasi ρ:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\rho(i)$	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8

Langkah selanjutnya adalah mendefinisikan permutasi π dengan $\pi(i) = 9i + 5 \pmod{16}$. Penyusuan urutan dari pesan diberikan pada tabel berikut ini:

Line	Round 1	Round 2	Round 3	Round 4	Round 5
left	id	ρ	ρ^2	ρ^3	ρ^4
right	π	$\rho\pi$	$\rho^2\pi$	$\rho^3\pi$	$\rho^4\pi$

2.3. Fungsi *Boolean*. Fungsi *Boolean* yang digunakan pada algoritma ini adalah sebagai berikut:

$$f_1(x, y, z) = x \oplus y \oplus z,$$
$$f_2(x, y, z) = (x \wedge y) \vee (\neg x \wedge z),$$
$$f_3(x, y, z) = (x \vee \neg y) \oplus z,$$
$$f_4(x, y, z) = (x \wedge z) \vee (y \wedge \neg z),$$
$$f_5(x, y, z) = x \oplus (y \vee \neg z).$$

Fungsi-fungsi *Boolean* ini digunakan dalam urutan sebagai berikut:

Line	Round 1	Round 2	Round 3	Round 4	Round 5
left	f_1	f_2	f_3	f_4	f_5
right	f_5	f_4	f_3	f_2	f_1

2.4. Pergeseran bit. Untuk setiap putaran, algoritma ini menggunakan jumlah pergeseran seperti berikut (walaupun hal ini bisa dirubah sesuai kriteria yang akan disebutkan pada bagian berikutnya):

Round	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	X_{12}	X_{13}	X_{14}	X_{15}
1	11	14	15	12	5	8	7	9	11	13	14	15	6	7	9	8
2	12	13	11	15	6	9	9	7	12	15	11	13	7	8	7	7
3	13	15	14	11	7	7	6	8	13	14	13	12	5	5	6	9
4	14	11	12	14	8	6	5	5	15	12	15	14	9	9	8	6
5	15	12	13	13	9	5	8	6	14	11	12	11	8	6	5	5

2.5. Konstanta. Diambil dari nilai bilangan bulat (*integer*) dari operasi berikut:

Line	Round 1	Round 2	Round 3	Round 4	Round 5
left	0	$2^{30} \cdot \sqrt{2}$	$2^{30} \cdot \sqrt{3}$	$2^{30} \cdot \sqrt{5}$	$2^{30} \cdot \sqrt{7}$
right	$2^{30} \cdot \sqrt[3]{2}$	$2^{30} \cdot \sqrt[3]{3}$	$2^{30} \cdot \sqrt[3]{5}$	$2^{30} \cdot \sqrt[3]{7}$	0

3. Kriteria Proses dalam Algoritma RIPEMD-160

RIPEMD-160 memperoleh kekuatannya dari pemilihan yang bisa dikatakan bijaksana dan hati-hati dari parameter untuk masing-masing fungsi, dikombinasikan dengan kenyataan bahwa pemrosesan dua bagian pesan membedakanya dengan algoritma RIPEMD biasa: penyusunan blok pesan dalam dua iterasi sangat berbeda jauh dan urutan fungsi *Boolean* dibalik.

Operasi RIPEMD-160 pada *register* A berhubungan dengan operasi yang sama pada MD5 (tetapi di sini, lima kata digunakan); rotasi pada *register* C telah ditambahkan untuk mencegah serangan MD5 yang memfokuskan pada *most significant bit* [8]. SHA-1 juga mempunyai dua rotasi, tetapi pada posisi yang berbeda. Pergeseran sebanyak 10 bit pada *register* C

dipilih karena angka ini tidak dipakai pada rotasi pada algoritma lain.

Permutasi dari pesan pada RIPEMD didesain sedemikian rupa sehingga dua kata yang semula "berdekatan" (mirip) pada perputaran 1-2 terpisah jauh pada perputaran 2-3 (dan sebaliknya). Prinsip ini dipakai pada algoritma RIPEMD-160 ini, dengan sedikit modifikasi. Permutasi π dipilih sedemikian rupa sehingga dua pesan yang berdekatan pada bagian kiri akan selalu berada setidaknya berbeda tujuh posisi pada bagian kanan.

Untuk fungsi *Boolean*, diputuskan untuk menghilangkan fungsi mayoritas karena sifat simetris yang dimilikinya dan kerugian dalam hal performansi. Fungsi *Boolean* yang digunakan sekarang sama dengan fungsi yang digunakan pada algoritma MD5. Seperti yang disebutkan sebelumnya, fungsi *Boolean* pada bagian kanan dan kiri digunakan dengan urutan yang berbeda-beda.

Sedangkan syarat untuk pergeseran bit adalah sebagai berikut:

- pergeseran dipilih antara 5 dan 15 (pergeseran yang terlalu kecil atau terlalu besar dianggap tidak bagus, dan pilihan angka di atas 16 tidak membawa perubahan yang signifikan);
- setiap blok pesan harus dirotasikan dengan angka yang berbeda-beda, sehingga semuanya tidak memiliki keseimbangan (*parity*) yang sama;
- pergeseran yang digunakan pada setiap *register* tidak boleh mengandung suatu pola tertentu (sebagai contoh, total pergeseran tidak boleh habis dibagi 32);
- sebagian besar konstanta pergeseran tidak boleh habis dibagi 4.

Sebagai catatan, keputusan desain membutuhkan sebuah kompromi: adalah tidak mungkin untuk membuat sebuah pilihan yang bagus mengenai penyusunan urutan pesan dan konstanta pergeseran untuk lima putaran, yang juga "optimal" untuk tiga dari lima putaran.

4. Perbandingan dengan Fungsi *Hash* Lainnya

Pada bagian ini, penulis akan membandingkan performansi dari RIPEMD-160, RIPEMD-128, SHA-1, MD5, dan MD4. Implementasi ditulis dalam bahasa Assembly yang dioptimalkan untuk prosesor Pentium; optimisasi dilakukan dengan menggukan sifat paralel dari level instruksi dari prosesor ini. Di samping desain serial masing-masing algoritma, setiap algoritma dapat menggunakan fitur ini. Kecepatan yang ditujunkan kurang-lebih serupa dengan hasil perhitungan secara teoritis berdasarkan jumlah operasi yang dilakukan tiap algoritma. RIPEMD-160 lebih lambat 15% lebih lambat jika dibandingan dengan SHA-1 dan empat kali lebih lambat dibandingkan dengan MD4. Pada mesin RISC *big-endian*, perbedaan antara SHA-1 dan RIPEMD-160 akan sedikit lebih besar. Sedangkan pada implementasi pada bahasa C, perbedaannya sekitar 2,2 sampai 2,6 lebih lambat.

algorithm	performance (Mbit/s)	
	Assembly	*C*
MD4	190.6	81.4
MD5	136.2	59.7
SHA-1	54.9	21.2
RIPEMD-128	77.6	35.6
RIPEMD-160	45.3	19.3

Perbandingan performansi dari berbagai fungsi *hash*

5. Impelementasi

Fungsi *hash* ini diimplementasikan dalam bahasa C dan berjalan *under-DOS* sehingga tidak ada GUI.

Berikut ini adalah *pseudo-code* dari fungsi *hash* RIPEMD-160 yang diimpelementasikan:

Definisi fungsi *Boolean*:

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

Konstanta yang ditambahkan (dalam heksadesimal)

$K(j) = 00000000$ $\quad (0 \le j \le 15)$
$K(j) = 5A827999$ $\quad (16 \le j \le 31)$ $\quad \lfloor 2^{30}.\sqrt{2} \rfloor$
$K(j) = 6ED9EBA1$ $\quad (32 \le j \le 47)$ $\quad \lfloor 2^{30}.\sqrt{3} \rfloor$
$K(j) = 8F1BBCDC$ $\quad (48 \le j \le 63)$ $\quad \lfloor 2^{30}.\sqrt{5} \rfloor$
$K(j) = A953FD4E$ $\quad (64 \le j \le 79)$ $\quad \lfloor 2^{30}.\sqrt{7} \rfloor$
$K'(j) = 50A28BE6$ $\quad (0 \le j \le 15)$ $\quad \lfloor 2^{30}.\sqrt[3]{2} \rfloor$
$K'(j) = 5C4DD124$ $\quad (16 \le j \le 31)$ $\quad \lfloor 2^{30}.\sqrt[3]{3} \rfloor$
$K'(j) = 6D703EF3$ $\quad (32 \le j \le 47)$ $\quad \lfloor 2^{30}.\sqrt[3]{5} \rfloor$
$K'(j) = 7A6D76E9$ $\quad (48 \le j \le 63)$ $\quad \lfloor 2^{30}.\sqrt[3]{7} \rfloor$
$K'(j) = 00000000$ $\quad (64 \le j \le 79)$

Seleksi Karakter Pesan

r(j) = j (0 ≤ j ≤ 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

Jumlah pergeseran
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

Inisialisasi Nilai (dalam heksadesimal)
h0 = 67452301; h1 = EFCDAB89;
h2 = 98BADCFE; h3 = 10325476;
h4 = C3D2E1F0;

Padding yang dilakukan sama persis dengan yang dilakukan pada algoritma MD4 dan MD5 [20,21,22]. Pesan masukan setelah dilakukan *padding* terdiri dari blok-blok sebanyak *t* sepanjang 16-*word* yang dinotasikan dengan $\chi_i[j]$ dengan $0 \le i \le t\text{-}1$ dan $0 \le j \le 15$. Simbol ⊞ melambangkan penambahan modulo 2^{32} dan pergeseran ke kiri secara rotasi sebanyak *s* bit. Hasil akhir dari fungsi ini terdiri dari hasil penggabungan dari h0, h1, h2, h3, dan h4 setelah masing-masing diubah ke dalam *string* sepanjang 4-byte menggunakan konvensi *little-endian*.

Pseudo-code

```
for i := 0 to t–1 {
    A := h0; B := h1; C := h2; D := h3; E := h4;
    A′ := h0; B′ := h1; C′ := h2; D′ := h3; E′ := h4;
    for j := 0 to 79 {
        T := rol_s(j) (A ⊞ f(j,B,C,D) ⊞ X_i[r(j)] ⊞ K(j)) ⊞ E;
        A := E; E := D; D := rol_10(C); C := B; B := T;
        T := rol_s′(j) (A′ ⊞ f(79–j,B′,C′,D′) ⊞ X_i[r′(j)] ⊞ K′(j)) ⊞ E′;
        A′ := E′; E′ := D′; D′ := rol_10(C′); C′ := B′; B′ := T;
    }
    T := h1 ⊞ C ⊞ D′; h1 := h2 ⊞ D ⊞ E′; h2 := h3 ⊞ E ⊞ A′;
    h3 := h4 ⊞ A ⊞ B′; h4 := h0 ⊞ B ⊞ C′; h0 := T;
}
```

Nilai *hash* yang dihasilkan:

```
Hash of "" = 0x9c1185a5c5e9fc54612808977ee8f548b2258d31
Hash of "a" = 0x0bdc9d2d256b3ee9daae347be6f4dc835a467ffe
Hash of "abc" = 0x8eb208f7e05d987a9b044a8e98c6b087f15a0bfc
Hash of "message digest" = 0x5d0689ef49d2fae572b881b123a85ffa21595f36
Hash of "abcdefghijklmnopqrstuvwxyz" =
0xf71c27109c692c1b56bbdceb5b9d2865b3708dbc
Hash of "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq" =
0x12a053384a9c0c88e405a06c27dcf49ada62eb2b
Hash of "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
= 0xb0e20b6e3116640286ed3a87a5713079b21f5189
Hash of 8 times "1234567890" =
0x9b752e45573d4b39f4dbd3323cab82bf63326bfb
Hash of 1 million times "a" = 0x52783243c1697bdbe16d37f97f68f08325dc1528
```

6. Kesimpulan

Sesuai dengan namanya, algoritma RIPEMD-160 melakukan *hashing* terhadap pedan dan menghasilkan 160 bit *message digest*. Walaupun performansi algoritma ini kurang jika dibandingkan dengan algoritma lain, setidaknya fungsi ini masih tahan terhadap serangan-serangan yang biasanya dilakukan terhadap fungsi *hash* lainnya.

Daftar Pustaka

[1] M. Bellare, R. Canetti, H. Krawczyk, "Keying hash functions for message authentication," *Advances in Cryptology*, Proceedings Crypto'96, LNCS 1109, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 1-15. Full version: http://www.research.ibm.com/security/.

[2] T. Berson, "Differential cryptanalysis mod 232 with applications to MD5," *Advances in Cryptology*, *Proceedings Eurocrypt*'92, LNCS 658, R. A. Rueppel, Ed., Springer-Verlag, 1993, pp. 71-80.

[3] A. Bosselaers, R. Govaerts, J. Vandewalle, "Fast hashing on the Pentium," *Advances in Cryptology*, *Proceedings Crypto*'96, LNCS 1109, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 298-312.

[4] A. Bosselaers, H. Dobbertin, B. Preneel, "The RIPEMD-160 cryptographic hash function," *Dr. Dobb's Journal*, Vol. 22, No. 1, January 1997, pp. 24-28.

[5] A. Bosselaers, R. Govaerts, J. Vandewalle, "SHA: a design for parallel architectures?," *Advances in Cryptology*, *Proceedings Eurocrypt*'97, LNCS 1233, W. Fumy, Ed., Springer-Verlag, 1997, pp. 348-362.

[6] A. Bosselaers, "Even faster hashing on the Pentium," Presented at the rump session of Eurocrypt'97, Konstanz, Germany, May 12-15, 1997, and updated on November 13, 1997. Available as ftp://ftp.esat.kuleuven.ac.be/pub/COSIC/bosselae/pentiumplus.ps.gz.

[7] B. den Boer, A. Bosselaers, "An attack on the last two rounds of MD4," *Advances in Cryptology*, Proceedings Crypto'91, LNCS 576, J. Feigenbaum, Ed., Springer-Verlag, 1992, pp. 194-203.
[8] B. den Boer, A. Bosselaers, "Collisions for the compression function of MD5," *Advances in Cryptology, Proceedings Eurocrypt*'93, LNCS 765, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 293-304.
[9] H. Dobbertin, "Alf swindles Ann," *CryptoBytes*, Vol. 1,No 3, 1995, pp. 5.
[10] H. Dobbertin, "RIPEMD with two-round compress function is not collision free," *Journal of Cryptology*, Vol. 10, No. 1, 1997, pp. 51-69.
[11] H. Dobbertin, A. Bosselaers, B. Preneel, "RIPEMD-160: A Strengthened Version of RIPEMD," *Fast Software Encryption*, LNCS 1039, D. Gollman, Ed., Springer-Verlag, 1996, pp. 71-82. Final (corrected) version: http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.
[12] H. Dobbertin, "Cryptanalysis of MD4," *Fast Software Encryption*, LNCS 1039, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 53-69.
[13] H. Dobbertin, "Cryptanalysis of MD4," submitted to *Journal of Cryptology*.
[14] H. Dobbertin, "The status of MD5 after a recent attack," *CryptoBytes*, Vol. 2, No 2, 1996, pp. 1, 3-6.
[15] H. Dobbertin, "The first two rounds of MD4 are not one-way," *Fast Software Encryption*, LNCS, Springer-Verlag, 1998, to appear.
[16] FIPS 180, "Secure Hash Standard," NIST, US Department of Commerce, Washington D.C., May 1993.
[17] FIPS 180-1, "Secure Hash Standard," NIST, US Department of Commerce, Washington D.C., April 1995.
[18] ISO/IEC 10118, "Information technology — Security techniques — Hash-functions, Part 1: General (IS, 1994); Part 2: Hash-functions using an n-bit block cipher algorithm," (IS, 1994); Part 3: Dedicated hash-functions (IS, 1997); Part 4: Hash-functions using modular arithmetic, (FCD, 1997).
[19] B. Preneel, P.C. van Oorschot, "MDx-MAC and building fast MACs from hash functions," *Advances in Cryptology*, Proceedings Crypto'95, LNCS 963, D. Coppersmith, Ed., Springer-Verlag, 1995, pp. 1-14.
[20] RIPE, "Integrity Primitives for Secure Information Systems. Final Report of RACE Integrity Primitives Evaluation (RIPE-RACE 1040)," LNCS 1007, Springer-Verlag, 1995.
[21] R.L. Rivest, "The MD4 message digest algorithm," *Advances in Cryptology*, Proceedings Crypto'90, LNCS 537, S. Vanstone, Ed., Springer-Verlag, 1991, pp. 303-311.
[22] R.L. Rivest, "The MD4 message-digest algorithm," *Request for Comments (RFC)* 1320, Internet Activities Board, Internet Privacy Task Force, April 1992.
[23] R.L. Rivest, "The MD5 message-digest algorithm," *Request for Comments (RFC)* 1321, Internet Activities Board, Internet Privacy Task Force, April 1992.
[24] P.C. van Oorschot, M.J. Wiener, "Parallel collision search with application to hash functions and discrete logarithms," *Proceedings 2nd ACM Conference on Computer and Communications Security*, ACM, 1994, pp. 210-218.
[25] S. Vaudenay, "On the need for multipermutations: cryptanalysis of MD4 and SAFER," *Fast Software Encryption*, LNCS 1008, B. Preneel, Ed., Springer-Verlag, 1995, pp. 286-297.