ALGORITMA RIPEMD

Roland L. Bu'ulölö – 135 04 072

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-Mail: if14072@students.if.itb.ac.id

Abstrak

Fungsi hash adalah suatu fungsi dalam bidang kriptografi yang merupakan alat yang sangat penting dalam berbagai aplikasi kriptografi, sebagai contoh dalam pembentukan tanda-tangan digital, otentikasi, dan sebagainya. Semenjak ditemukannya algoritma hash MD4 oleh Ron Rivest, telah banyak algoritma-algoritma hash yang lain yang telah dibentuk berdasarkan prinsip-prinsip dalam algoritma ini. Salah satunya adalah algoritma RIPEMD.

Algoritma RIPEMD terdiri dari beberapa varian, di antaranya adalah RIPEMD-128 dan RIPEMD-160. Algoritma RIPEMD-160 adalah algoritma hash yang merupakan peningkatan dari RIPEMD-128. Peningkatan dan pengembangan ini dilakukan karena telah banyak usaha-usaha kriptanalisis yang telah dilakukan terhadap RIPEMD-128, dan berdasarkan informasi yang didapat maka diciptakanlah RIPEMD-160.

Dalam makalah ini, akan dibahas secara mendalam mengenai algoritma RIPEMD secara umum dan berbagai variannya, serta usaha-usaha kriptanalisis terhadap RIPEMD dan pengaruhnya terhadap berbagai variannya.

1. Pendahuluan

Di dalam kriptografi, terdapat fungsi yang berguna untuk aplikasi keamanan untuk menjaga integritas pesan dan otentikasinya. Fungsi tersebut adalah fungsi hash. Fungsi ini menerima masukan string yang panjangnya sembarang dan mengkonversinya menjadi string keluaran yang panjangnya tetap dan umumnya berukuran jauh lebih kecil daripada ukuran string semula.

Bentuk persamaan umum fungsi hash adalah:

$$h = f(X)$$

Dengan f adalah fungsi hash, X adalah string pesan masukan, dan h adalah nilai hasil fungsi hash. Nilai hasil fungsi hash ini disebut juga sebagai hash value atau message digest.

Fungsi hash yang sering dan aman digunakan adalah fungsi yang termasuk ke dalam jenis fungsi satu arah (one-way function). Fungsi hash yang satu arah berarti pesan yang diubah menjadi message digest oleh fungsi tersebut tidak akan dapat dikembalikan lagi.

Prinsip-prinsi fungsi hash yang baik adalah sebagai berikut:

1. Fungsi f dapat diterapkan pada blok data berukuran berapa saja.
2. f menghasilkan nilai (h) dengan panjang tetap (fixed-length output).
3. $f(X)$ mudah dihitung untuk setiap nilai X yang diberikan.
4. Untuk setiap h yang diberikan, tidak mungkin menemukan X sedemikian

sehingga $f(X) = h$. Sifat ini menyatakan bahwa fungsi f harus merupakan fungsi satu arah. Sifat ini disebut juga dengan *preimage resistance*.

5. Secara komputasi tidak mungkin mencari pasangan nilai X dan Y sehingga $f(X) = f(Y)$. Sifat ini disebut juga dengan *second preimage resistance*.
6. Untuk setiap X yang diberikan, tidak mungkin mencari $Y \neq X$ sedemikian sehingga $f(Y) = f(X)$. Sifat ini disebut juga dengan *collision resistance*.

Untuk suatu fungsi hash tahan kolisi yang ideal dengan hasil m-bit, cara tercepat untuk mencari kolisi dari fungsi tersebut adalah dengan serangan *square-root* yang kira-kira membutuhkan $2^{m/2}$ operasi.

Kebanyakan fungsi hash merupakan proses yang iteratif yang menerima input dengan panjang sembarang dan memproses blok-blok input dengan panjang tetap.

Sebuah input X dibagi ke dalam t blok X_1 sampai X_t. Dengan definisi input yang demikian, fungsi hash h dapat dijelaskan sebagai berikut:

$$H_0 = IV$$

$$H_i = f(H_{i-1}, X_i),\ 1 \leq i \leq t$$

$$h(X) = H_t$$

Dalam skema di atas, f adalah fungsi kompresi dari h, H_i adalah variabel penyambung (*chaining variable*) antara langkah ke *i-1* dengan langkah ke *i*, dan *IV* menandakan nilai awal (*initial vector*).

2. Latar Belakang Algoritma RIPEMD

Fungsi hash yang paling populer digunakan dalam aplikasi yang menggunakan teknologi kriptografi adalah fungsi-fungsi yang merupakan turunan dari keluarga fungsi hash MD4. MD4 ditemukan oleh Ron Rivest pada tahun 1990. Algoritma ini adalah algoritma yang cepat dan sangat baik implementasinya dalam prosesor 32-bit. Namun, oleh karena ditemukannya kolisi dalam fungsi hash ini, maka kemudian pada tahun 1991 Ron Rivest mengajukan fungsi hash baru yang merupakan peningkatan dari fungsi hash MD4 ini, yaitu MD5. Sampai pada saat ini, MD5 merupakan algoritma yang cukup luas digunakan. Namun, dalam fungsi hash MD5 ini juga masih ada ditemukan kolisi. Untuk aplikasi-aplikasi yang umum, kolisi pada MD5 tidak menunjukkan masalah yang berarti, tetapi tetap saja dengan ditemukannya kolisi masih ada prinsip perancangan fungsi hash yang baik yang dilanggar.

Fungsi hash RIPEMD diajukan oleh konsorsium RIPE (RACE Integrity Primitives Evaluation) sebagai hasil realisasi dari hasil analisis terhadap MD4 dan MD5. Fungsi hash RIPEMD (RIPE Message Digest) adalah algoritma kriptografi hash yang ditujukan untuk implementasi software pada mesin berarsitektur 32-bit. Algoritma ini dikembangkan dari algoritma hash MD4 varian 256-bit yang pertama sekali diperlihatkan pada tahun 1990 oleh Ron Rivest. Fitur utama dari algoritma RIPEMD ini adalah adanya dua rantai komputasi yang berbeda, independen, dan paralel, di mana hasil kedua komputasi ini digabungkan pada akhir prosesnya. Varian RIPEMD, RIPEMD-160 menghasilkan nilai hash 160-bit. Algoritma ini ditujukan untuk memberikan tingkat keamanan yang tinggi selama 10 tahun atau lebih. Algoritma varian RIPEMD yang lain, RIPEMD-128 adalah varian yang memiliki performa yang lebih cepat daripada RIPEMD-160, yang menghasilkan nilai hash 128-bit. Algoritma RIPEMD-160 sebenarnya adalah peningkatan dari algoritma RIPEMD-128 yang memberikan tingkat keamanan yang lebih baik dari RIPEMD-128 dari segi penghindaran kolisi.

3. Deskripsi RIPEMD-160

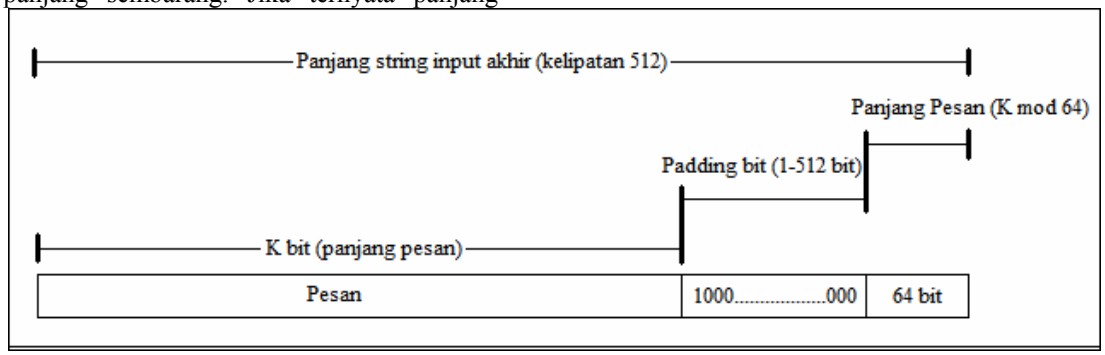
Seperti semua fungsi hash varian MD4, RIPEMD-160 bekerja pada ukuran word 32-bit. Operasi primitifnya adalah:

- Rotasi bit ke kiri (left-spin)
- Operasi boolean bitwise (AND, OR, NOT, XOR)
- Operasi penjumlahan modulo 2^{32} terhadap word dalam mode *two's complement*.

RIPEMD-160 mengkompresi string input dengan panjang sembarang dengan membagi

string tersebut menjadi blok-blok yang masing-masing sebesar 512 bit. Setiap blok dibagi menjadi 16 string dengan panjang masing-masing subblok adalah 4 byte. Kemudian masing-masing string 4 byte tersebut diubah ke word 32-bit dengan metode *little-endian*. Metode little-endian ini digunakan karena metode ini terimplementasikan secara hardware ke dalam prosesor Intel 80x86 yang umum digunakan.

String yang menjadi input fungsi ini memiliki panjang sembarang. Jika ternyata panjang string ini bukan merupakan kelipatan 512 bit, maka sebelum dibagi, string input ini akan di-*padding* dengan cara yang sama seperti yang umum dalam keluarga MD4, yaitu dengan menambahkan satu bit 1 dan diikuti dengan kumpulan bit 0. 64 bit-bit yang terakhir dari *padding bits* (yang merupakan tambahan pada input sebenarnya) menyatakan panjang string input sebenarnya yang dinyatakan dalam bit. Diagramnya sebagai berikut:



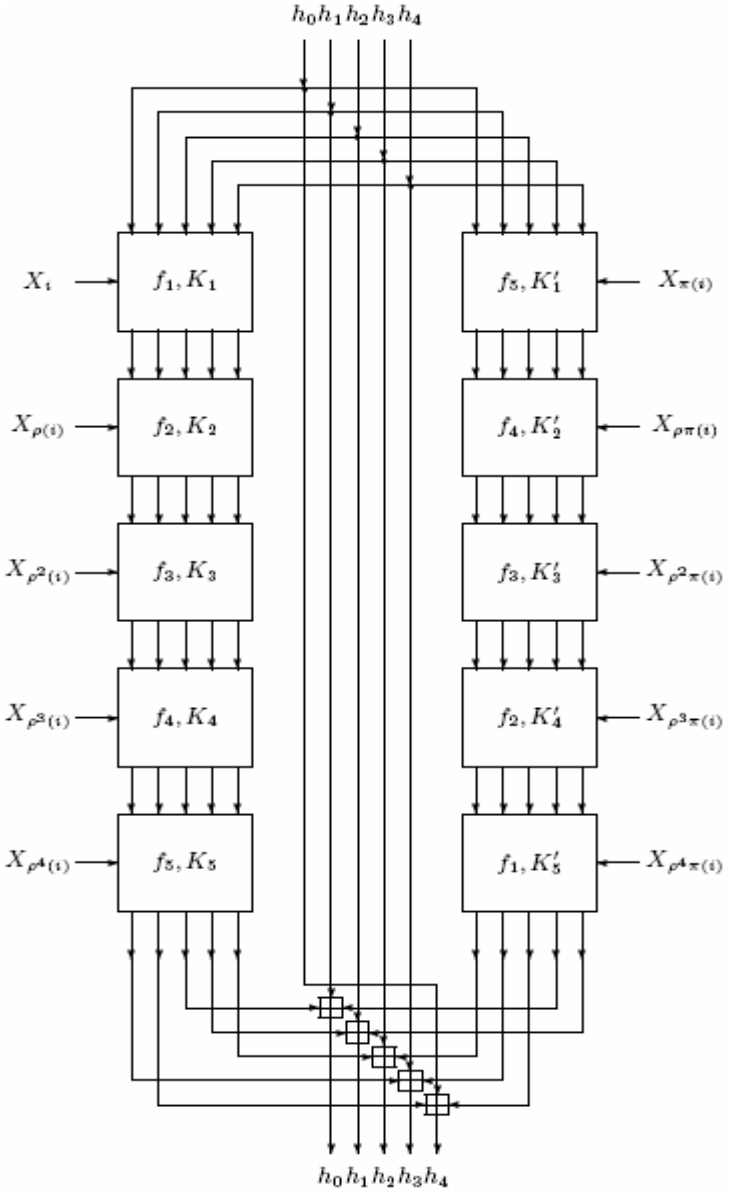
Gambar 1 – Diagram pemodifikasian string input dengan padding.

Nilai hash hasil proses RIPEMD-160 disimpan dalam 5 word 32 bit. Pembentukan kelimanya membentuk struktur umum dari algoritma ini. Isi akhir dari kelima word ini akan digabung menjadi string 160 bit, dengan metode little-endian.

Pada awalnya kelima word 32-bit ini diinisialisasi dengan suatu nilai tertentu (*initial vector*). Nilai-nilai tersebut adalah sebagai berikut (dalam heksadesimal):

h_0 = 67452301
h_1 = EFCDAB89
h_2 = 98BADCFE
h_3 = 10325476
h_4 = C3D2E1F0

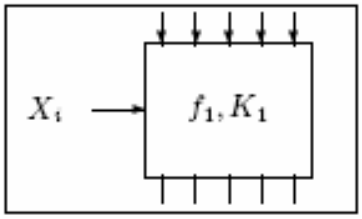
Variabel h_0 sampai h_4 adalah variabel penyambung (*chaining variable*). Kondisi di mana variabel penyambung masih merupakan IV, disebut dengan status awal. Bagian utama dari algoritma dalam RIPEMD-160 adalah fungsi kompresinya. Fungsi kompresi menerima input berupa blok string input beserta variabel penyambung dan menghasilkan nilai berikutnya untuk variabel penyambung, dengan kata lain fungsi ini menghitung status baru dari status sebelumnya. Diagram fungsi kompresi adalah sebagai berikut:



Gambar 2 – Diagram fungsi kompresi *f*

Dari diagram di atas terlihat bahwa fungsi kompresi ini terdiri dari 5 ronde. Dalam 1 ronde ada 2 pemrosesan yang terjadi dan keduanya bekerja secara paralel dan bebas satu sama lain. Dalam setiap pemrosesan tersebut ada 16 langkah. Ada 16 langkah karena fungsi kompresi menerima input 1 blok 512 bit yang menjadi 16 subblok yang masing-masing sebesar 16 bit. Dalam satu langkah, hanya 1 subblok yang diproses (subblok 0 sampai subblok 15). Oleh karena itu, dalam sekali pemanggilan fungsi kompresi akan terjadi langkah sebanyak 5 × 2 × 16 = 160 langkah. Bandingkan dengan MD4 yang hanya 3 × 16 = 48 langkah, dan MD5 yang hanya 4 × 16 = 64 langkah.

Dari gambar 2 di atas, terlihat banyak pola yang memiliki bentuk umum yang seperti ini:



Gambar 3 – Satu tahapan fungsi kompresi.

Dari gambar di atas ini, terlihat ada beberapa notasi. Notasi-notasi tersebut adalah X_i, f_1, dan K_1. Pengertian dari masing-masing notasi tersebut adalah sebagai berikut.

Notasi yang pertama adalah X_i. X_i adalah subblok ke i dari string input. Lajur kiri dan lajur kanan dari gambar 2 memiliki pemilihan subblok yang berbeda. Ronde-ronde yang berbeda juga memiliki pemilihan subblok yang berbeda. Untuk semua nilai i yang mungkin (0,1,...,15) terdapat fungsi permutasi ρ dan π yang memetakan i ke suatu nilai yang akan digunakan untuk pemilihan subblok. Tabel permutasi untuk fungsi permutasi ρ adalah:

i	0	1	2	3	4	5	6	7
ρ(i)	7	4	13	1	10	6	15	3

i	8	9	10	11	12	13	14	15
ρ(i)	12	0	9	5	2	14	11	8

Sementara itu, fungsi permutasi π adalah:

$$\pi(i) = 9i + 5 \pmod{16}$$

Dengan definisi ρ dan π yang demikian, maka pemilihan subblok dalam setiap ronde didefinisikan sebagai berikut:

Lajur	R1	R2	R3	R4	R5
Kiri	i	ρ(i)	ρ^2(i)	ρ^3(i)	ρ^3(i)
Kanan	π(i)	ρπ (i)	$\rho^2\pi$ (i)	$\rho^3\pi$ (i)	$\rho^3\pi$ (i)

Dalam tabel di atas $\rho^2(i) = \rho(\rho(i))$, $\rho^3(i) = \rho(\rho(\rho(i)))$, dan seterusnya.

Notasi berikutnya adalah *f*. Fungsi *f* disini menyatakan fungsi boolean yang menerima 3 input word 32 bit. Dalam implementasinya, fungsi ini akan menerima input 3 variabel penyambung. Definisi fungsi *f* ini adalah:

$f_1(x,y,z) = x \oplus y \oplus z$

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

Fungsi-fungsi boolean di atas diaplikasikan berbeda-beda dalam setiap ronde. Aplikasi fungsi tersebut dalam setiap ronde adalah:

Lajur	R1	R2	R3	R4	R5
Kiri	f_1	f_2	f_3	f_4	f_5
Kanan	f_5	f_4	f_3	f_2	f_1

Notasi berikutnya adalah *K*. *K* adalah nilai konstanta yang akan ditambahkan dalam setiap langkah. Nilai *K* yang digunakan dalam masing-masing ronde adalah:

Lajur	Kiri	Kanan
Ronde 1	0	$2^{30} \times \sqrt{2}$
Ronde 2	$2^{30} \times \sqrt{2}$	$2^{30} \times \sqrt{3}$
Ronde 3	$2^{30} \times \sqrt{3}$	$2^{30} \times \sqrt{5}$
Ronde 4	$2^{30} \times \sqrt{5}$	$2^{30} \times \sqrt{7}$
Ronde 5	$2^{30} \times \sqrt{7}$	0

Berdasarkan definisi notasi-notasi di atas, definisi 1 langkah operasi dalam suatu tahapan adalah:

A ← ((A + f(B,C,D) + X + K) << s) + E
C ← C << 10

Dalam formula di atas, lambang "<< s" menandakan operasi rotasi ke kiri sebanyak s bit, dan variabel A, B, C, D, dan E menyatakan kumpulan variabel penyambung.

Operasi formula di atas dilakukan terhadap keseluruhan subblok string input (16 subblok berukuran 32 bit). Dalam implementasi formula di atas, fungsi f yang digunakan, nilai X, dan nilai K ditentukan oleh ronde dan subblok ke berapa yang sedang diproses, sesuai dengan yang telah didefinisikan.

Jumlah rotasi yang dilakukan dalam operasi di atas juga ditentukan oleh subblok dan ronde. Jumlah rotasi s yang dilakukan untuk ronde dan subblok tertentu adalah:

	R1	R2	R3	R4	R5
X_0	11	12	13	14	15
X_1	14	13	15	11	12

	R1	R2	R3	R4	R5
X_2	15	11	14	12	13
X_3	12	15	11	14	13
X_4	5	6	7	8	9
X_5	8	9	7	6	5
X_6	7	9	6	5	8
X_7	9	7	8	5	6
X_8	11	12	13	15	14
X_9	13	15	14	12	11
X_{10}	14	11	13	15	12
X_{11}	15	13	12	14	11
X_{12}	6	7	5	9	8
X_{13}	7	8	5	9	6
X_{14}	9	7	6	8	5
X_{15}	8	7	9	6	5

4. Algoritma Hash RIPEMD-160

Dalam membangun algoritma RIPEMD-160, definisi dari variabel-variabel dan rumus-rumus di atas masih kurang baik untuk diimplementasikan, karena algoritma yang dihasilkan tidak akan cukup efisien. Untuk itu, diperlukan pendefinisian khusus yang merupakan penurunan dari definisi yang telah disebutkan di atas.

4.1 Definisi Ronde

Dalam 1 ronde, terdapat 16 langkah yang dilakukan. Berdasarkan ini, didefinisikan suatu nilai *j*, dimana rentang nilai *j* adalah $0 \leq j \leq 79$, sehingga setiap ronde didefinisikan ke dalam *j* dengan ketentuan:

Ronde 1: $0 \leq j \leq 15$
Ronde 2: $16 \leq j \leq 31$
Ronde 3: $32 \leq j \leq 47$
Ronde 4: $48 \leq j \leq 63$
Ronde 5: $64 \leq j \leq 79$

Dapat dilihat di sini, bahwa rentang nilai *j* dalam masing-masing ronde adalah tetap dan sama, yaitu 16. Maka, untuk suatu langkah yang ada pada ronde ke *N*, dengan indeks subblok yang diproses adalah *I*, nilai *j* yang bersesuaian dapat dihitung dengan rumus:

$$j = (N \times 16) + I - 16$$

4.2 Definisi Fungsi Boolean *f*

$f(j,x,y,z) = x \oplus y \oplus z$	Ronde 1
$f(j,x,y,z) = (x \wedge y) \vee (\neg x \wedge z)$	Ronde 2
$f(j,x,y,z) = (x \vee \neg y) \oplus z$	Ronde 3
$f(j,x,y,z) = (x \wedge z) \vee (y \wedge \neg z)$	Ronde 4
$f(j,x,y,z) = x \oplus (y \vee \neg z)$	Ronde 5

4.3 Definisi Konstanta *K*

Nilai konstanta *K* yang ditulis di sini adalah dalam bentuk heksa desimal.

$$
\begin{array}{lll}
K(j) = \texttt{00000000}_{\texttt{X}} & (0 \leq j \leq 15) & \\
K(j) = \texttt{5A827999}_{\texttt{X}} & (16 \leq j \leq 31) & \lfloor 2^{30} \cdot \sqrt{2} \rfloor \\
K(j) = \texttt{6ED9EBA1}_{\texttt{X}} & (32 \leq j \leq 47) & \lfloor 2^{30} \cdot \sqrt{3} \rfloor \\
K(j) = \texttt{8F1BBCDC}_{\texttt{X}} & (48 \leq j \leq 63) & \lfloor 2^{30} \cdot \sqrt{5} \rfloor \\
K(j) = \texttt{A953FD4E}_{\texttt{X}} & (64 \leq j \leq 79) & \lfloor 2^{30} \cdot \sqrt{7} \rfloor \\
K'(j) = \texttt{50A28BE6}_{\texttt{X}} & (0 \leq j \leq 15) & \lfloor 2^{30} \cdot \sqrt[3]{2} \rfloor \\
K'(j) = \texttt{5C4DD124}_{\texttt{X}} & (16 \leq j \leq 31) & \lfloor 2^{30} \cdot \sqrt[3]{3} \rfloor \\
K'(j) = \texttt{6D703EF3}_{\texttt{X}} & (32 \leq j \leq 47) & \lfloor 2^{30} \cdot \sqrt[3]{5} \rfloor \\
K'(j) = \texttt{7A6D76E9}_{\texttt{X}} & (48 \leq j \leq 63) & \lfloor 2^{30} \cdot \sqrt[3]{7} \rfloor \\
K'(j) = \texttt{00000000}_{\texttt{X}} & (64 \leq j \leq 79) &
\end{array}
$$

Gambar 4 – Konstanta *K*

K(j) adalah konstanta yang digunakan pada lajur kiri, dan *K'(j)* adalah konstanta yang digunakan pada lajur kanan.

4.4 Definisi Pemilihan Subblok

Pada definisi sebelumnya, cara pemilihan subblok yang akan diproses setiap ronde adalah dengan memetakan indeks dari subblok tersebut dengan fungsi permutasi ρ dan π. Untuk implementasi ke dalam pseudo-code, maka didefinisikan fungsi permutasi *r*, yang merupakan fungsi permutasi hasil penurunan kombinasi fungsi permutasi ρ dan fungsi permutasi π, dan menerima input *j* di mana *j* adalah seperti yang telah didefinisikan sebelumnya ($0 \leq j \leq 79$).

$$
\begin{array}{ll}
r(j) & = j \qquad (0 \leq j \leq 15) \\
r(16..31) & = 7, 4, 13, 1, 10, 6, 15, 3, 12, 0, 9, 5, 2, 14, 11, 8 \\
r(32..47) & = 3, 10, 14, 4, 9, 15, 8, 1, 2, 7, 0, 6, 13, 11, 5, 12 \\
r(48..63) & = 1, 9, 11, 10, 0, 8, 12, 4, 13, 3, 7, 15, 14, 5, 6, 2 \\
r(64..79) & = 4, 0, 5, 9, 7, 12, 2, 10, 14, 1, 3, 8, 11, 6, 15, 13 \\
r'(0..15) & = 5, 14, 7, 0, 9, 2, 11, 4, 13, 6, 15, 8, 1, 10, 3, 12 \\
r'(16..31) & = 6, 11, 3, 7, 0, 13, 5, 10, 14, 15, 8, 12, 4, 9, 1, 2 \\
r'(32..47) & = 15, 5, 1, 3, 7, 14, 6, 9, 11, 8, 12, 2, 10, 0, 4, 13 \\
r'(48..63) & = 8, 6, 4, 1, 3, 11, 15, 0, 5, 12, 2, 13, 9, 7, 10, 14 \\
r'(64..79) & = 12, 15, 10, 4, 1, 5, 8, 7, 6, 2, 13, 14, 0, 3, 9, 11
\end{array}
$$

Gambar 5 – Fungsi permutasi *r*

Jika dilihat dari pemetaan fungsi permutasi *r* ini, hasil yang ditunjukkan sama dengan hasil pemetaan yang dilakukan dengan menggunakan fungsi permutasi ρ dan fungsi permutasi π.

4.5 Definisi Jumlah Rotasi *s*

Untuk dapat diimplementasikan dengan baik dalam algoritma, jumlah rotasi pada subblok dan ronde tertentu didefinisikan dengan fungsi *s*, yang menerima input *j*. Definisi fungsi *s* ini adalah:

$$
\begin{array}{ll}
s(0..15) & = 11, 14, 15, 12, 5, 8, 7, 9, 11, 13, 14, 15, 6, 7, 9, 8 \\
s(16..31) & = 7, 6, 8, 13, 11, 9, 7, 15, 7, 12, 15, 9, 11, 7, 13, 12 \\
s(32..47) & = 11, 13, 6, 7, 14, 9, 13, 15, 14, 8, 13, 6, 5, 12, 7, 5 \\
s(48..63) & = 11, 12, 14, 15, 14, 15, 9, 8, 9, 14, 5, 6, 8, 6, 5, 12 \\
s(64..79) & = 9, 15, 5, 11, 6, 8, 13, 12, 5, 12, 13, 14, 11, 8, 5, 6 \\
s'(0..15) & = 8, 9, 9, 11, 13, 15, 15, 5, 7, 7, 8, 11, 14, 14, 12, 6 \\
s'(16..31) & = 9, 13, 15, 7, 12, 8, 9, 11, 7, 7, 12, 7, 6, 15, 13, 11 \\
s'(32..47) & = 9, 7, 15, 11, 8, 6, 6, 14, 12, 13, 5, 14, 13, 13, 7, 5 \\
s'(48..63) & = 15, 5, 8, 11, 14, 14, 6, 14, 6, 9, 12, 9, 12, 5, 15, 8 \\
s'(64..79) & = 8, 5, 12, 9, 12, 5, 14, 6, 8, 13, 6, 5, 15, 13, 11, 11
\end{array}
$$

Gambar 6 – Fungsi penghasil jumlah rotasi bit

4.6 Pseudo-code Algoritma RIPEMD-160

```
for i ← 0 to t – 1 {
    // Inisialisasi variabel penyambung dengan IV

    // Lajur kiri
    A ← h0 ; B ← h1 ; C ← h2 ; D ← h3 ; E ← h4
    // Lajur kanan
    A' ← h0 ; B' ← h1 ; C' ← h2 ; D' ← h3 ; E' ← h4

    // Loop
    for j ← 0 to 79 {
        T ← ((A + f(j,B,C,D) + Xi[r(j)] + K(j)) << s(j)) + E ;
        A ← E ; E ← D ; D ← C << 10 ; C ← B ; B ← T ;

        T' ← ((A' + f(j,B',C',D') + Xi[r'(j)] + K'(j)) << s'(j)) + E' ;
        A' ← E' ; E' ← D' ; D' ← C' << 10 ; 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 ;
}
```

Gambar 7 – Algoritma hash RIPEMD-160

Variabel *t* menyatakan jumlah blok dari string input, dan X_0, X_1, ..., X_t adalah blok-blok dari string input tersebut. Subblok dari blok input dinyatakan dengan *Xi*[*k*], dan dalam algoritma di atas terlihat bahwa yang diproses adalah subblok yang ditunjuk oleh fungsi permutasi *r*. Nilai konstanta dan jumlah rotasi yang digunakan untuk memproses subblok ditunjukkan oleh *K(j)* dan *s(j)*.

Variabel *A*, *B*, *C*, *D*, dan *E* adalah variabel penyambung yang pada akhir algoritma akan digabungkan sehingga menjadi 160-bit string yang menyatakan nilai hash yang dihasilkan.

5. Deskripsi RIPEMD-128

RIPEMD-160 adalah peningkatan dari RIPEMD-128. Dalam RIPEMD-128, nilai hash yang dihasilkan hanya memiliki panjang 128 bit. Hanya terdapat sedikit perbedaan antara RIPEMD-160 dengan RIPEMD-128, di antaranya adalah:

1. Satu langkah operasi:

$$A = (A + f(B,C,D) + X + K) << s$$

2. Fungsi boolean:

Lajur	R1	R2	R3	R4
Kiri	f_1	f_2	f_3	f_4
Kanan	f_5	f_4	f_3	f_2

3. Konstanta:

Lajur	Kiri	Kanan
Ronde 1	0	$2^{30} \times \sqrt{2}$
Ronde 2	$2^{30} \times \sqrt{2}$	$2^{30} \times \sqrt{3}$
Ronde 3	$2^{30} \times \sqrt{3}$	$2^{30} \times \sqrt{5}$
Ronde 4	$2^{30} \times \sqrt{5}$	0

Terlihat di sini, jumlah ronde yang dilakukan lebih sedikit daripada RIPEMD-160. Selain itu, RIPEMD-128 juga hanya menggunakan 4 variabel penyambung. Karakteristik ini membuat RIPEMD-128 memiliki performa yang lebih cepat daripada RIPEMD-160, tetapi dengan sedikit pengurangan keamanan jika dibandingkan dengan RIPEMD-128.

6. Contoh Hasil Hashing Dengan Algoritma RIPEMD

Berikut adalah contoh hasil hashing beberapa arsip tes dengan menggunakan algoritma RIPEMD:

RIPEMD-160

String input	Nilai Hash
"" (string kosong)	9c1185a5c5e9fc54612808977ee8f548b2258d31
"a"	0bdc9d2d256b3ee9daae347be6f4dc835a467ffe
"abc"	8eb208f7e05d987a9b044a8e98c6b087f15a0bfc
"message digest"	5d0689ef49d2fae572b881b123a85ffa21595f36
"abcdefghijklmnopqrstuvwxyz"	f71c27109c692c1b56bbdceb5b9d2865b3708dbc
"abcdbcdecdefdefgefghfghighijhijkij kljklmklmnlmnomnopnopq"	12a053384a9c0c88e405a06c27dcf49ada62eb2b
"ABCDEFGHIJKLMNOPQRSTUV WXYZabcdefghijklmnopqrstuvwxyz 0123456789"	b0e20b6e3116640286ed3a87a5713079b21f5189
"12345678901234567890123456789 01234567890123456789012345678901 23456789012345678 90"	9b752e45573d4b39f4dbd3323cab82bf63326bfb
(1 juta karakter "a")	52783243c1697bdbe16d37f97f68f08325dc1528

RIPEMD-128

String input	Nilai Hash
"" (string kosong)	cdf26213a150dc3ecb610f18f6b38b46
"a"	86be7afa339d0fc7cfc785e72f578d33
"abc"	c14a12199c66e4ba84636b0f69144c77
"message digest"	9e327b3d6e523062afc1132d7df9d1b8
"abcdefghijklmnopqrstuvwxyz"	fd2aa607f71dc8f510714922b371834e
"abcdbcdecdefdefgefghfghighijhijkij kljklmklmnlmnomnopnopq"	a1aa0689d0fafa2ddc22e88b49133a06
"ABCDEFGHIJKLMNOPQRSTUV WXYZabcdefghijklmnopqrstuvwxyz 0123456789"	d1e959eb179c911faea4624c60c5c702
"12345678901234567890123456789 01234567890123456789012345678901 23456789012345678 90"	3f45ef194732c2dbb2c4a2c769795fa3
(1 juta karakter "a")	4a7f5723f954eba1216c9d8f6320431f

7. Evaluasi Performa

Dalam bagian ini, ditampilkan hasil perbandingan performa antara algoritma RIPEMD-160, RIPEMD-128, SHA-1, MD5, dan MD4. Evaluasi dilakukan dilakukan dalam lingkungan yang menggunakan prosesor Intel Pentium 90 MHz. Implementasi algoritma ditulis dalam bahasa assembly dan bahasa C. Berikut hasilnya:

Algoritma	Performance (Mbit/s)	
	Assembly	C
MD4	165.7	81.4
MD5	113.5	59.7
SHA-1	46.5	21.2
RIPEMD-128	63.8	35.6
RIPEMD-160	39.8	19.3

Dari tabel di atas, dapat terlihat bahwa algoritma MD4 dan MD5 memiliki performa yang sangat cepat jika dibandingkan dengan RIPEMD-160, maupun RIPEMD-128. Namun, algoritma SHA-1 memiliki performa yang hampir sama dengan RIPEMD-160, dan RIPEMD-128 bahkan lebih cepat daripada SHA-1. Algoritma MD4 dan MD5 tidak

memiliki langkah yang sebanyak RIPEMD-160 maupun RIPEMD-128. Hal ini menyebabkan perbedaan performa yang cukup besar di antara keduanya.

8. Keamanan RIPEMD-160

Keamanan algoritma hash pada umumnya ditentukan oleh sulit mudahnya menemukan kolisi pada algoritma tersebut. Kolisi dalam algoritma hash adalah kejadian di mana adanya pasangan input yang dapat menghasilkan nilai hash yang sama oleh algoritma hash tersebut. Semua algoritma hash dirancang agar tidak memiliki kolisi, atau setidaknya sangat sulit ditemukan.

Untuk mencari kolisi dalam suatu algoritma hash, serangan yang umum dilakukan adalah *birthday attack* atau juga dikenal dengan nama *square-root attack*. Untuk algoritma hash yang menghasilkan output sepanjang n-bit, seseorang akan memerlukan nilai-nilai hash sebanyak $2^{(n/2)}$ buah. Jika ternyata kolisi dapat ditemukan dengan kompleksitas komputasi yang lebih kecil dari $2^{(n/2)}$, maka algoritma hash tersebut dapat dianggap tidak aman secara akademis.

Algoritma RIPEMD-160, seperti yang telah disebutkan dalam latar belakang, merupakan peningkatan dari algoritma MD4. MD4 menjadi dasar pengembangan karena MD4 memiliki performa yang tinggi dalam arsitektur 32-bit. Namun, MD4 itu sendiri pada saat sudah tidak dianggap aman lagi, karena telah ditemukan kolisi dalam MD4 dan kompleksitas komputasi untuk menemukannya cukup kecil (sebuah PC hanya butuh waktu sebentar untuk menyerang MD4). Oleh karena itu, algoritma RIPEMD dikembangkan dengan memodifikasi dan menambahkan algoritma perbaikan dari yang semula MD4.

Pada awal 1995, Hans Dobbertin, salah seorang anggota tim pencipta RIPEMD, menemukan kolisi dalam algoritma RIPEMD itu sendiri. Algoritma RIPEMD-160, dan RIPEMD-128 adalah hasil perbaikan lanjut dari RIPEMD melalui informasi hasil serangan terhadapnya. Berikut ini adalah 2 pasang kolisi yang ada pada RIPEMD:

M_1	579faf8e	9ecf579	574a6aba	78413511	a2b410a4	ad2f6c9f	b56202c	4d757911
	bdeaae7	78bc91f2	47bc6d7d	9abdd1b1	a45d2015	817104ff	264758a8	61064ea5
M_1'	579faf8e	9ecf579	574a6aba	78513511	a2b410a4	ad2f6c9f	b56202c	4d757911
	bdeaae7	78bc91f2	c7c06d7d	9abdd1b1	a45d2015	817104ff	264758a8	e1064ea5
H	1fab152	1654a31b	7a33776a	9e968ba7				
M_2	579faf8e	9ecf579	574a6aba	78413511	a2b410a4	ad2f6c9f	b56202c	4d757911
	bdeaae7	78bc91f2	47bc6d7d	9abdd1b1	a45d2015	a0a504ff	b18d58a8	e70c66b6
M_2'	579faf8e	9ecf579	574a6aba	78513511	a2b410a4	ad2f6c9f	b56202c	4d757911
	bdeaae7	78bc91f2	c7c06d7d	9abdd1b1	a45d2015	a0a504ff	b18d58a8	670c66b6
H	1f2c159f	569b31a6	dfcaa51a	25665d24				

Sampai pada saat ini, masih belum ditemukan kolisi yang muncul dalam algoritma RIPEMD-128 maupun RIPEMD-160. Namun, dengan telah ditemukannya kolisi pada RIPEMD, kemungkinan besar pada algoritma RIPEMD-128 dan RIPEMD-160 bisa ditemukan kolisi juga dengan menyerangnya dengan cara yang sama dengan menyerang RIPEMD. Ditemukannya kolisi dalam suatu algoritma hash memang menunjukkan adanya kelemahan dalam algoritma hash tersebut. Namun, dalam implementasi di dunia nyata

kolisi akan sangat jarang sekali muncul dalam pesan-pesan yang berarti (contoh: pesan teks). Terlebih lagi, jika suatu pesan yang berarti diubah dengan kolisi, maka kemungkinan besar pengubahan itu dapat dideteksi dengan akuran dengan melakukan pengecekan terhadap arti pesan yang telah berubah tersebut. Hal ini karena perubahan melalui kolisi bekerja dalam mode bit-bit, sehingga sangat sulit untuk menemukan pasangan input pesan yang **memiliki arti** yang mudah dilihat dan sekaligus juga menghasilkan kolisi.

Namun demikian, algoritma hash yang memiliki kolisi dan di-cap sebagai "tidak aman secara akademis" tidak selalu juga aman dari pelanggaran keamanan. Jika data yang ditambahkan ke dalam pesan, seperti kode kontrol, bit-bit padding, atau barisan bilangan acak, dapat diubah tanpa merusak data, maka kolisi dapat saja muncul antara pasangan pesan yang sama sekali berbeda di mana perubahan terhadap pesan tersebut sama sekali tidak mengganggu makna dari pesan. Hal ini tentunya sangat sulit dideteksi dari segi pengecekan makna pesan, dan lubang keamanan seperti itu akan dapat dimanfaatkan oleh oknum-oknum yang memiliki niat tertentu.

9. Kesimpulan

- Algoritma hash yang umum digunakan adalah algoritma hash yang merupakan turunan dari algoritma hash MD4. MD4 baik karena algoritma ini sangat baik performanya dalam mesin 32-bit.

- Performa RIPEMD-128 dan RIPEMD-160 masih lebih lambat jika dibandingkan dengan algoritma hash turunan MD4 yang lain. Hal ini dikarenakan RIPEMD memiliki jumlah langkah dan operasi yang lebih banyak daripada algoritma hash turunan MD4 yang lain.

- Algoritma RIPEMD-160 memiliki 5 ronde yang masing-masingnya terdiri dari 2 tahapan yang dikerjakan secara paralel. Masing-masing tahapan terdiri dari 16 langkah, sehingga jumlah langkah total dalam proses algoritma ini secara keseluruhan adalah 160 langkah. Jauh lebih banyak jika dibandingkan dengan total langkah MD4 (48 langkah) dan MD5 (64 langkah).

- Keamanan suatu algoritma hash ditentukan dari kompleksitas komputasi yang dibutuhkan untuk menemukan kolisi dalam algoritma hash tersebut.

- Jumlah nilai hash minimal yang diperlukan atau kompleksitas komputasi minimal untuk menemukan kolisi dalam algoritma hash dengan hasil n-bit adalah $2^{(n/2)}$.

- Suatu pesan yang diubah melalui celah algoritma hash akibat kolisi masih dapat dideteksi jika ternyata pesan tersebut memiliki makna tertulis, contohnya seperti pada pesan teks.

- Kolisi dalam algoritma RIPEMD-160 dan RIPEMD-128 sampai saat ini masih belum ditemukan. Namun, dalam algoritma sebelumnya, RIPEMD, kolisi telah ditemukan.

10. Daftar Pustaka

[1] Munir, Rinaldi. (2006). Diktat Kuliah IF5054 Kriptografi. Institut Teknologi Bandung.

[2] Dobbertin, Hans c.s. (1996). *RIPEMD-160 A Strengthened Version of RIPEMD*. German Information Security Agency.

[3] Wang, Xiaoyun c.s. (2004). *Collisions for Hash Functions MD4, MD5, HAVAL-128, and RIPEMD*. Shandong University, Chinese Academy of Sciences, Shanghai Jiatong University.

[4] Masashi, Une c.s. (2006). *IMES Discussion Paper Series – Year 2010 issues on cryptographic algorithms*. Bank of Japan.

[5] RSA Laboratories. (1997). *CryptoBytes – volume 3, number 2*. RSA Laboratories.

[6] Preneel, Bart c.s. (1997). *The Cryptographic Hash Function RIPEMD-160*. RSA Laboratories.