STUDI DAN ANALISIS ALGORITMA HASH MD6 SERTA PERBANDINGANNYA DENGAN MD5

Ferdian Thung (13507127)

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jalan Ganesha No. 10, Bandung 40132
e-mail: if17127@students.if.itb.ac.id

ABSTRAK

Fungsi hash merupakan fungsi satu arah yang banyak digunakan untuk pengecekan keaslian suatu pesan. Salah satu algoritma yang telah dikenal dan diimplementasikan secara luas adalah algoritma hash MD5. Algoritma ini dipublikasikan oleh Ron Rivest dan telah digunakan dalam jangka waktu yang cukup lama. Akan tetapi, algoritma ini telah terbukti tidak memenuhi salah satu properti *cryptographic hash function* yakni properti *collision resistant* sehingga dianggap tidak lagi aman digunakan. Oleh karena itu, Ron Rivest mengajukan penerus dari algoritma ini, yakni algoritma hash MD6. MD6 diharapkan dapat memperbaiki kekurangan MD5 dan menjadi algoritma hash baru yang mampu memenuhi properti-properti yang diinginkan dari suatu fungsi hash yang aman. Makalah ini akan membahas mengenai algorima hash MD6 serta perbandingannya dengan algoritma hash MD5.

Kata kunci: *fungsi hash, MD5, MD6, cryptographic hash function property*

1. PENDAHULUAN

1.1 Fungsi Hash

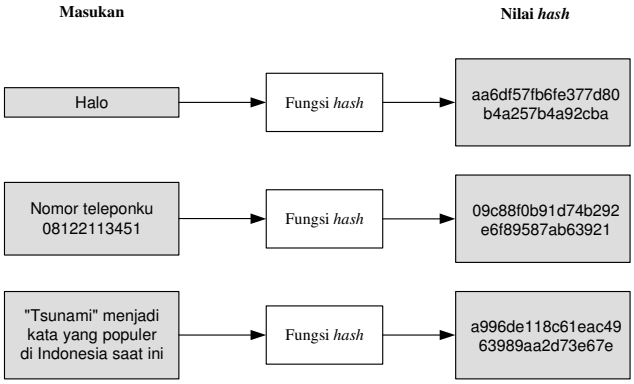
Fungsi hash adalah fungsi yang menerima masukan string yang panjangnya sembarang, lalu mentransformasikannya menjadi string keluaran yang panjangnya tetap (*fixed*) (umumnya berukuran jauh lebih kecil daripada ukuran string semula). Persamaan fungsi hash dapat dituliskan dengan persamaan berikut.

$$h = H(M) \quad (1)$$

dengan

- M = pesan berukuran sembarang.
- h = nilai hash (*hash value*) atau pesan ringkas (*message-digest*) dengan ukuran tetap.

Berikut gambaran penggunaan fungsi hash yang mengubah suatu string menjadi string lain dengan panjang tertentu.



Gambar 1 Contoh Penggunaan Fungsi Hash

Dalam kriptografi, keberadaan fungsi hash yang aman secara kriptografik sangatlah penting. Fungsi hash semacam ini memiliki properti-properti sebagai berikut.

- *Collision resistance* : seseorang seharusnya tidak dapat menemukan dua pesan berbeda, sebut saja M dan M′ sedemikian sehingga H(M)= H(M′)(terjadi kolisi).
- *First preimage resistance* : seseorang yang diberikan hasil hash h seharusnya tidak dapat menemukan M di mana H(M) = h. Salah satu contoh mengapa hal ini penting yakni pada umumnya sandi lewat pengguna disimpan dalam bentuk hash. Jika seseorang memiliki akses pada data sandi lewat yang telah di*hash* maka seharusnya ia tidak bisa memperoleh sandi lewat yang asli dari data tersebut. Akan tetapi, hal ini mungkin terjadi jika properti tidak dipenuhi.
- *Second preimage resistance* : seseorang yang memiliki pesan M seharusnya tidak dapat memperoleh pesan M′ di mana M tidak sama dengan M′ tetapi H(M) = H(M′). Properti ini merupakan implikasi dari *collision resistance*.

Selain properti yang telah disebutkan di atas, ada dua properti lain lagi yang ditujukan untuk fungsi hash yang termasuk ke dalam *keyed hash function*, yakni.

- *Pseudorandomness* : seseorang seharusnya tidak bisa membedakan keluaran fungsi hash dari fungsi random murni. Properti ini secara langsung mengimplikasikan *unpredictability*, tetapi tidak sebaliknya.
- *Unpredictability* : seseorang yang diberikan akses terhadap suatu fungsi hash seharusnya tidak dapat menebak keluarannya tanpa sebelumnya mengetahui pesan yang akan dihash. Dengan kata lain, seseorang dengan hak akses tersebut tidak mungkin mendapatkan pasangan nilai (M, h) di mana H(M) = h tanpa mengetahui nilai M.

Fungsi hash telah digunakan secara luas dalam berbagai aplikasi misalnya integritas pesan, otentikasi, tanda tangan digital, *secure timestamping*, dan banyak aplikasi lainnya. Sekuriti aplikasi-aplikasi tersebut seringkali bergantung secara langsung pada properti sekuriti dari fungsi hash yang digunakan. Jika fungsi hash tidak seaman yang dipercaya sebelumnya (properti sekuriti tidak lagi dipenuhi), maka aplikasi juga tidak lagi aman. Oleh karena itu, seringkali ada ketertarikan yang besar dalam membuktikan, dengan seteliti mungkin, bahwa suatu algoritma fungsi hash benar-benar memiliki properti sekuriti dan tahan pada berbagai macam serangan.

1.2 Keluarga Fungsi Hash MD

Keluarga fungsi hash MD termasuk keluarga fungsi hash yang telah memiliki banyak versi. Semua versi fungsi hash tersebut dirancang oleh Professor Ronald Rivest dari MIT. Hingga makalah ini ditulis, keluarga fungsi hash ini telah mencapai versi 6. Keluarga fungsi ini merupakan keluarga fungsi hash yang banyak digunakan dalam berbagai aplikasi. Bahkan, tercatat bahwa MD2 masih digunakan dalam beberapa aplikasi hingga tahun 2009 di mana pada tahun tersebut ia tidak didukung lagi oleh aplikasi-aplikasi tersebut. Walau begitu, ia bertahan cukup lama sejak diketahui adanya serangan yang berhasil terhadapnya. Sementara itu, fungsi hash MD4 dan MD5 masih banyak digunakan dalam berbagai aplikasi. MD5 sendiri sebenarnya merupakan perbaikan dari MD4.

Walau masih banyak digunakan, MD5 sebenarnya telah terbukti tidak lagi aman secara kriptografik. Pada tanggal 1 Maret 2005, Arjen Lenstra, Xiaoyun Wang, dan Benne de Weger berhasil mendemostrasikan pembentukan dua buah sertifikat X.509 dengan kunci publik yang berbeda tetapi mempunyai nilai hash yang sama. Bahkan, beberapa hari kemudian, Vlastimil Klima memperbaiki algoritma tersebut di mana ia mampu menghasilkan kolisi MD5 hanya dalam waktu beberapa jam dengan menggunakan komputer PC. Dari sini jelas bahwa fungsi hash ini sebenarnya tidak lagi dijamin aman digunakan. Akibatnya, berbagai aplikasi yang menggunakannya juga tidak lagi dapat menjamin ketahanannya terhadap serangan.

Walau sebenarnya sudah rentan, namun memang pada prakteknya belum banyak terdengar masalah dalam penggunaan MD5. Akan tetapi, fakta bahwa MD5 telah berhasil diserang merupakan tanda perlunya pembuatan versi baru dari keluarga algoritma hash ini. Menjawab hal tersebut, Ron Rivest kini telah membuat rancangan algoritma baru penerus dari MD5 yakni algoritma hash MD6. Algoritma ini, berikut dengan pendahulunya, akan dibahas pada bagian selanjutnya.

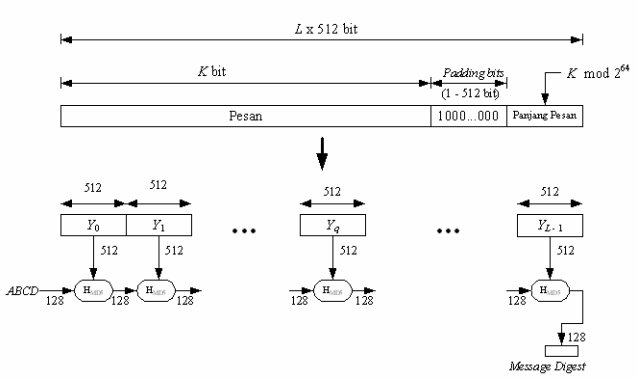
2. MD5

2.1 Mode Operasi MD5

MD5 didasarkan pada mode operasi iteratif berantai yang umumnya disebut dengan konstruksi Merkle-Damgard. Konstruksi Merkle-Damgard pada dasarnya menggunakan fungsi kompresi $f:\{0,1\}^{n+\ell}\rightarrow\{0,1\}^{n}$, atau sebuah *block cipher* E yang dibuat untuk berperilaku sebagai fungsi kompresi melalui transformasi Davies-Meyer: $f(x,y)= Ex(y) \oplus y$. Jika f merupakan sebuah fungsi kompresi, maka konstruksi Merkle-Damgard yang menggunakan fungsi kompresi ini, misalkan saja namanya MD_f, dimulai dengan menambahkan pesan masukan m agar memiliki panjang yang merupakan kelipatan ℓ, kemudian mengambil vektor inisialisasi IV berukuran n-bit. Ia kemudian berjalan secara sekuensiaol melalui ℓ-bit *message chunks*, mulai dari *chunk* pertama, dan berakhir setelah pemrosesan *chunk* terakhir. Berikut *pseudocode* algoritma tersebut.

Input : $m = m_1 \| m_2 \| \dots \| m_t$, di mana $\|m_i\| = \ell$ untuk semua i. Output: *message digest*, h.
$y_0 \leftarrow IV$ for $i \leftarrow 1$ to t $y_i \leftarrow f(y_{i-1},m_i)$ $h \leftarrow y_t$

Ilustrasi operasi di atas dapat dilihat pada gambar (1). Gambar (1) tersebut merupakan contoh konstruksi Merkle-Damgard pada MD5.



Gambar 2 Konstruksi Merkle-Damgard pada MD5

2.2 Algoritma MD5

Algoritma MD5 dapat dibagi menjadi empat tahap, yakni penambahan bit-bit pengganjal (*padding bits*), penambahan nilai panjang pesan semula, inisialisasi penyangga (*buffer*) *MD*, dan pengolahan pesan dalam blok berukuran 512 bit. Berikut penjelasan masing-masing tahap tersebut.

2.2.1 *Penambahan Bit-bit Pengganjal*

Pesan ditambah dengan sejumlah bit pengganjal sedemikian sehingga panjang pesan dalam satuan bit kongruen dengan 448 modulo 512. Jika panjang pesan ternyata 448 bit, maka pesan tersebut ditambah 512 bit sehingga menjadi 960 bit. Jadi, panjang bit-bit pengganjal adalah antara 1 sampai 512. Bit-bit pengganjal terdiri dari sebuah bit 1 diikuti dengan bit 0 untuk sisanya.

2.2.2 *Penambahan Nilai Panjang Pesan*

Pesan yang telah diberi bit-bit pengganjal selanjutnya ditambah lagi dengan 64 bit yang menyatakan panjang pesan semula. Jika panjang pesan > 2^{64} maka yang diambil adalah panjangnya dalam modulo 2^{64}. Dengan kata lain, jika panjang pesan semula adalah *K* bit, maka 64 bit yang ditambahkan menyatakan *K* modulo 2^{64}. Setelah ditambah dengan 64 bit, panjang pesan sekarang telah menjadi kelipatan 512 bit.

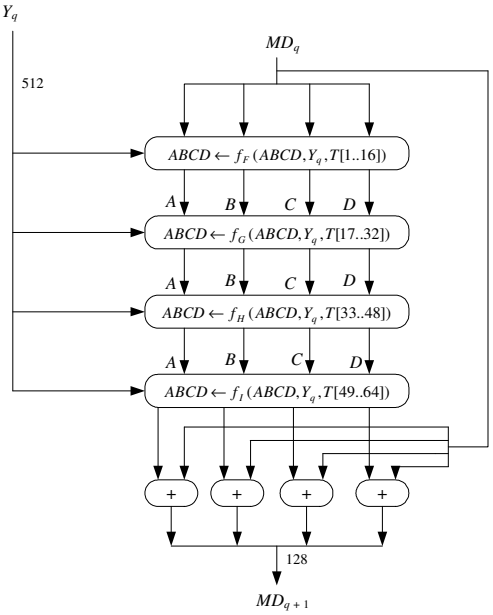
2.2.3 *Inisialisasi Penyangga MD*

MD5 membutuhkan 4 buah penyangga (*buffer*) yang masing-masing panjangnya 32 bit. Total panjang penyangga adalah 4 × 32 = 128 bit. Keempat penyangga ini menampung hasil antara dan hasil akhir. Keempat penyangga ini diberi nama *A*, *B*, *C*, dan D. Setiap penyangga diinisialisasi dengan nilai-nilai heksadesimal berikut:

A = 01234567
B = 89ABCDEF
C = FEDCBA98
D = 76543210

2.2.3 *Pengolahan Pesan dalam Blok Berukuran 512 bit*

Pesan dibagi menjadi *L* buah blok yang masing-masing panjangnya 512 bit (Y_0 sampai Y_{L-1}). Setiap blok 512-bit diproses bersama dengan penyangga *MD* menjadi keluaran 128-bit, dan ini disebut proses H_{MD5}. Gambaran proses H_{MD5} diperlihatkan pada Gambar (3).



Gambar 3 Proses H_{MD5}

Pada Gambar (3), Y_q menyatakan blok 512-bit ke-*q* dari pesan yang telah ditambah bit-bit pengganjal dan tambahan 64 bit nilai panjang pesan semula. MD_q adalah nilai *message digest* 128-bit dari proses H_{MD5} ke-*q*. Pada awal proses, MD_q berisi nilai inisialisasi penyangga *MD*.Proses H_{MD5} terdiri dari 4 buah putaran. Fungsi-fungsi f_F, f_G, f_H, dan f_I masing-masing berisi 16 kali operasi dasar terhadap masukan, setiap operasi dasar menggunakan nilai Ti sesuai tabel (1). Perbedaan masing-masing fungsi f_F, f_G, f_H, dan f_I sendiri dapat dilihat pada tabel (2).

Tabel 1 Nilai Ti

T[1] = D76AA478	T[33] = FFFA3942
T[2] = E8C7B756	T[34] = 8771F681
T[3] = 242070DB	T[35] = 69D96122
T[4] = C1BDCEEE	T[36] = FDE5380C
T[5] = F57C0FAF	T[37] = A4BEEA44
T[6] = 4787C62A	T[38] = 4BDECFA9
T[7] = A8304613	T[39] = F6BB4B60
T[8] = FD469501	T[40] = BEBFBC70
T[9] = 698098D8	T[41] = 289B7EC6

T[10] = 8B44F7AF	T[42] = EAA127FA
T[11] = FFFF5BB1	T[43] = D4EF3085
T[12] = 895CD7BE	T[44] = 04881D05
T[13] = 6B901122	T[45] = D9D4D039
T[14] = FD987193	T[46] = E6DB99E5
T[15] = A679438E	T[47] = 1FA27CF8
T[16] = 49B40821	T[48] = C4AC5665
T[17] = F61E2562	T[49] = F4292244
T[18] = C040B340	T[50] = 432AFF97
T[19] = 265E5A51	T[51] = AB9423A7
T[20] = E9B6C7AA	T[52] = FC93A039
T[21] = D62F105D	T[53] = 655B59C3
T[22] = 02441453	T[54] = 8F0CCC92
T[23] = D8A1E681	T[55] = FFEFF47D
T[24] = E7D3FBCB	T[56] = 85845DD1
T[25] = 21E1CDE6	T[57] = 6FA87E4F
T[26] = C33707D6	T[58] = FE2CE6E0
T[27] = F4D50D87	T[59] = A3014314
T[28] = 455A14ED	T[60] = 4E0811A1
T[29] = A9E3E905	T[61] = F7537E82
T[30] = FCEFA3F8	T[62] = BD3AF235
T[31] = 676F02D9	T[63] = 2AD7D2BB
T[32] = 8D2A4C8A	T[64] = EB86D391

Tabel 2 Fungsi Dasar MD5

Nama	Notasi	$g(b, c, d)$
f_F	$F(b, c, d)$	$(b \wedge c) \vee (\sim b \wedge d)$
f_G	$G(b, c, d)$	$(b \wedge d) \vee (c \wedge \sim d)$
f_H	$H(b, c, d)$	$b \oplus c \oplus d$
f_I	$I(b, c, d)$	$c \oplus (b \wedge \sim d)$

Operasi dasar *MD5* diilustrasikan pada gambar (4) dan dapat ditulis dengan sebuah persamaan sebagai berikut:

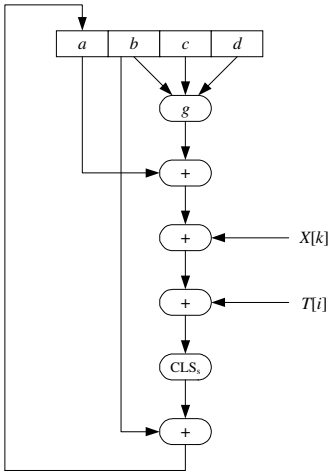
$$a \leftarrow b + \mathrm{CLS}_s(a + g(b, c, d) + X[k] + T[i]) \qquad (2)$$

dengan

- a, b, c, d = empat buah peubah penyangga 32-bit
- g = salah satu fungsi F, G, H, I
- CLS_s = *circular left shift* sebanyak s bit
- $X[k]$ = kelompok 32-bit ke-k dari blok 512 bit *message* ke-q. k bernilai antara 0 sampai 15.
- $T[i]$ = elemen Tabel T ke-i (32-bit)
- $+$ = operasi penjumlahan modulo 2^{32}

Untuk nilai shift yang digunakan pada setiap putaran akan mengikuti aturan berikut.

- Untuk i antara 0 sampai 15, nilai shift berulang mengikuti pola 7, 12, 17, 22.
- Untuk i antara 16 sampai 31, nilai shift berulang mengikuti pola 5, 9, 14 , 20.
- Untuk i antara 32 sampai 47, nilai shift berulang mengikuti pola 4, 11, 16, 23.
- Untuk i antara 48 sampai 63, nilai shift berulang mengikuti pola 6, 10, 15, 21.



Gambar 4 Operasi Dasar MD5

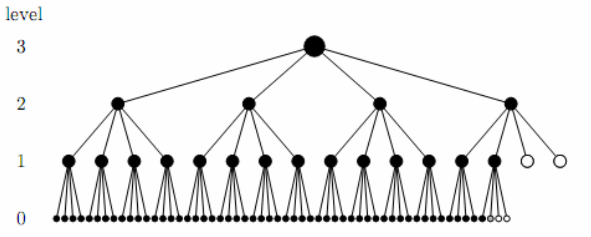
Dalam 16 kali operasi dasar, setiap kali satu operasi dasar diselesaikan, penyangga akan digeser ke kanan secara sirkuler dengan pertukaran sebagai berikut:

temp ← d
d ← c
c ← b
b ← a
a ← temp

3. MD6

3.1 Mode Operasi MD6

MD6 pada standarnya menggunakan mode operasi berbasis pohon yang mendukung paralelisme lebih besar. Sementara konstruksi Merkle-Damgard, jika dilihat sebagai graf, pada dasarnya adalah rantai yang panjang, mode pohon MD6 tampak seperti gambar (5), dengan sebuah fungsi kompresi *4-to-1* yang mereduksi panjang pesan keseluruhan pada setiap level.



Gambar 5 Mode Operasi Berbasis Pohon dari MD6

Komputasi pada mode ini dimulai dari daun paling bawah dan berjalan hingga mencapai akar. *Node* akar merepresentasikan fungsi kompresi final yang akan

mengeluarkan *message digest*. Setiap *node* bukan daun pada pohon diberi label dengan informasi tambahan yang juga menjadi masukan fungsi kompresi. Dengan kata lain, setiap *node* diberikan identifier unik dan *node* akar diberi tanda dengan sebuah bit z yang mengidentifikasikan bahwa ia merupakan fungsi kompresi final yang digunakan. Informasi tambahan yang dikodekan dalam masukan untuk setiap fungsi kompresi mencegah serangan fungsi hash di mana seseorang bisa memproduksi sebuah *query* pesan yang berkorespondensi dengan substruktur *query* lain.

Selain mode operasi berbasis pohon seperti dijelaskan di atas, MD6 juga mendukung mode operasi iteratif yang mirip dengan konstruksi Merkle-Damgard. Ini perlu karena terkadang memori yang diperlukan untuk pohon tersebut terlalu banyak untuk suatu sistem. Oleh karena itu, pada MD6 terdapat masukan kontrol mode L. Jika L = 0, maka mode operasinya iteratif mirip dengan Merkle-Damgard dan jika L = 64, maka mode operasinya berbasis pohon dengan paralelisme tinggi.

3.2 Algoritma MD6

Algoritma MD6 menerima masukan berupa sebuah pesan M dengan panjang positif m dan panjang keluaran d yang berkisar antara 1 sampai dengan 512 bit. Ada pula masukan opsional yakni key K, parameter mode L, dan jumlah ronde operasi r. Secara garis besar, algoritmanya mengikuti langkah berikut.

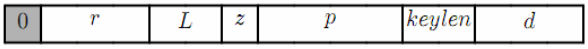
1. Inisialisasi nilai $\ell = 0$, $M_0 = M$, dan $m_0 = m$. ℓ adalah *current level*.
2. Kemudian, untuk setiap level, lakukan proses berikut.
   - Jika $\ell = L + 1$, panggil fungsi SEQ($M_{\ell-1}$,d,K,L,r) sebagai keluaran fungsi hash.
   - Panggil fungsi PAR($M_{\ell-1}$,d,K,L,r,ℓ) dan masukkan hasilnya ke dalam M_ℓ.
   - Kemudian, jika panjang M_ℓ sama dengan c word, kembalikan d bit terakhir dari M_ℓ sebagai keluaran fungsi hash. Secara default, nilai c adalah 16 word.

Perlu diperhatikan bahwa jika fungsi SEQ dipanggil, maka fungsi PAR tidak akan dipanggil.

3.2.1 *Fungsi SEQ*

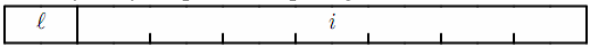
Fungsi SEQ($M_{\ell-1}$,d,K,L,r) merupakan fungsi hash sekuensial yang mirip dengan konstruksi Merkle-Damgard. Fungsi ini tidak dipanggil pada pengaturan *default* yakni nilai L = 64. Akan tetapi, misalnya jika nilai L = 0, fungsi ini dipanggil. Fungsi ini memiliki proses sebagai berikut.

1. Q merupakan array dengan panjang q = 15 word yang berisi angka di belakang koma dari $\sqrt{6}$.
2. f_r menyatakan fungsi kompresi MD6 yang memetakan blok data input B berukuran 64 word menjadi keluaran *chunk* C berukuran 16 word setelah melalui komputasi sebanyak r ronde (f_r juga menerima masukan informasi tambahan sebesar 25 word).
3. Misalkan C_1 merupakan vektor nol dengan panjang c =16 word yang bertindak sebagai IV.
4. Tambahkan bit-bit pengganjal bernilai 0 pada $M_{\ell-1}$ hingga panjangnya merupakan kelipatan (b-c) = 48 word. Setelah ini, $M_{\ell-1}$ dapat dilihat sebagai blok-blok B_0, B_1, …, B_{j-1} masing-masing berukuran (b-c) word di mana $j = \max(1, m_{\ell-1}/(b-c)w)$.
5. Untuk setiap blok B_i dengan i mulai dari 0 hingga j-1, hitung C_i secara paralel melalui proses berikut.
   - Misalkan p merupakan panjang bit-bit pengganjal pada B_i dengan rentang nilai antara 0-3072.
   - Masukkan nilai z = 1 ketika j = 1 dan z = 0 untuk kondisi lainnya (z = 1 hanya untuk blok terakhir yang akan dikompresi).
   - Misalkan V adalah nilai sepanjang satu word dari r||L||z||p||keylen||d seperti gambar (6).



Gambar 6 Layout Nilai V

   - Misalkan $U = \ell \cdot 2^{56} + I$ adalah ID unik dari *node*, yakni nilai sebuah nilai sebesar satu word yang unik untuk setiap fungsi kompresi. Layoutnya dapat dilihat pada gambar (7).



Gambar 7 Layout Nilai U

   - Panggil fungsi kompresi $f_r(Q||K||U||V||C_{i-1}||B_i)$ dan masukkan hasilnya ke dalam C_i (panjang C_i adalah c = 16 word).
6. Kembalikan d bit terakhir dari C_{j-1} sebagai keluaran fungsi hash.

3.2.2 *Fungsi PAR*

Fungsi PAR($M_{\ell-1}$,d,K,L,r,ℓ) merupakan fungsi yang melakukan operasi kompresi paralel yang membentuk level ℓ pohon dari level ℓ-1. Fungsi ini memiliki algoritma sebagai berikut.

1. Q merupakan array dengan panjang q = 15 word yang berisi angka di belakang koma dari $\sqrt{6}$.
2. f_r menyatakan fungsi kompresi MD6 yang memetakan blok data input B berukuran 64 word menjadi keluaran *chunk* C berukuran 16 word setelah melalui komputasi sebanyak r ronde (f_r juga menerima masukan informasi tambahan sebesar 25 word).
3. Jika diperlukan, tambahkan bit-bit pengganjal untuk $M_{\ell-1}$ yakni dengan menambahkan bit 0 hingga

panjangnya menjadi kelipatan dari b = 64 word. Setelah ini, $M_{\ell-1}$ dapat dilihat sebagai blok-blok B_0, B_1, …, B_{j-1} masing-masing berukuran b word di mana $j = \max(1, m_{\ell-1}/bw)$.

4. Untuk setiap blok B_i dengan i mulai dari 0 hingga j-1, hitung C_i secara paralel melalui proses berikut.
   - Misalkan p merupakan panjang bit-bit pengganjal pada B_i dengan rentang nilai antara 0-4096.
   - Masukkan nilai z = 1 ketika j = 1 dan z = 0 untuk kondisi lainnya (z = 1 hanya untuk blok terakhir yang akan dikompresi).
   - Misalkan V adalah nilai sepanjang satu word dari r||L||z||p||keylen||d seperti gambar (6).
   - Misalkan $U = \ell \cdot 2^{56} + I$ adalah ID unik dari *node*, yakni nilai sebuah nilai sebesar satu word yang unik untuk setiap fungsi kompresi. Layoutnya dapat dilihat pada gambar (7).
   - Panggil fungsi kompresi $f_r(Q||K||U||V||B_i)$ dan masukkan hasilnya ke dalam C_i (panjang C_i adalah c = 16 word).
5. Pada akhir pemrosesan, $M\ell = C_0||C_1|| \ ... \ ||C_{j-1}$ merupakan keluaran fungsi.

3.2.3 *Fungsi Kompresi f*

Pada fungsi SEQ maupun PAR, disebutkan fungsi kompresi f yang memetakan blok 64 word menjadi 16 word. Fungsi ini menerima masukan jumlah ronde r dan array N[0..n-1] dengan n = 89 word yang terdiri dari 64 word data dan 25 word informasi tambahan. Secara default, nilai r dihitung dengan persamaan (3).

$$r = 40 + d/4 \qquad (3)$$

dengan d adalah panjang bit keluaran hash yang rentang nilainya 0-512.

Fungsi kompresi f ini memiliki langkah sebagai berikut.

1. Pertama, hitung t = rc (setiap ronde memiliki c = 16 putaran).
2. Array A[0..t+n-1] merupakan array dari (t+n) word.
3. Inisialisasi nilai A[0..n-1] dengan masukan N[0..n-1].
4. Kemudian, dilakukan loop untuk perhitungan elemen A ke-i. i memiliki rentang nilai n-(t+n-1). Proses perhitungannya yakni dengan persamaan (4).

$$\begin{aligned} x &= S_{i-n} \oplus A_{i-n} \oplus A_{i-t0} \\ x &= x \oplus (A_{i-t1} \wedge A_{i-t2}) \oplus (A_{i-t3} \wedge A_{i-t4}) \\ x &= x \oplus (x \gg r_{i-n}) \\ A_i &= x \oplus (x \ll \ell_{i-n}) \end{aligned} \qquad (4)$$

dengan konstanta berikut.

- S_{i-n} = konstanta ronde
- t_0,t_1,t_2,t_3,t_4 = posisi *tap*.
- r_{i-n} = banyaknya shift kanan.
- ℓ_{i-n} = banyaknya shift kiri.

5. Nilai A[t+n-c..t+n-1] merupakan keluaran dari fungsi kompresi yang memiliki panjang c = 16 word.

3.2.4 *Konstanta MD6*

Pada fungsi kompresi, digunakan beberapa konstanta dalam perhitungan. Konstanta tersebut yakni.

1. Posisi *Tap*
   Nilai konstanta ini dapat dilihat pada tabel (3) berikut.

Tabel 3 Nilai Posisi *Tap*

t_0	t_1	t_2	t_3	t_4
17	18	21	31	67

2. Jumlah Shift
   Nilai r_{i-n} dan ℓ_{i-n} adalah nilai awal jumlah shift kanan dan shift kiri pada langkah dengan index i. index (i-n) diambil modulonya terhadap c = 16. Nilai shift ini dapat dilihat pada tabel (4).

Tabel 4 Konstanta Jumlah Shift

(i-n) mod 16	$\mathbf{r_{i-n}}$	$\boldsymbol{\ell_{i-n}}$
0	10	11
1	5	24
2	13	9
3	10	16
4	11	15
5	12	9
6	2	27
7	7	15
8	14	6
9	15	2
10	7	29
11	13	8
12	11	15
13	7	5
14	6	31
15	12	9

3. Konstanta Ronde
   Nilai S_{i-n} ditentukan dengan persamaan (5) berikut.

$$\begin{aligned} S_{i-n} &= S'_{(i-n)/16} \\ S'_0 &= 0x0123456789abcdef \\ S^* &= 0x7311c2812425cfa0 \\ S'_{j+1} &= (S'_j \lll 1) \oplus (S'_j \wedge S^*) \end{aligned} \qquad (5)$$

Berdasarkan persamaan tersebut, nilai S_{i-n} sebenarnya diambil dari deret S'_j di mana nilainya ditentukan oleh basis S'_0 dan S^*.

4. PEMBAHASAN DAN ANALISIS

Sebagai penerus MD5, MD6 diharapkan dapat memberikan kinerja lebih dibanding pendahulunya. Hal ini dijawab salah satunya dengan kemampuan MD6 dalam memanfaatkan teknologi chip dan memori yang telah jauh berkembang sejak MD5 dirancang. Dengan menggunakan mode operasi berbasis pohon, MD6 dapat memanfaatkan banyak core dalam pemrosesan di mana perhitungan dapat dilakukan secara paralel. Dengan kemampuan paralelisme semacam ini, MD6 dapat bekerja secara lebih efisien dibanding MD5 pada prosesor yang telah mendukung *multicore*. Walau memang dibutuhkan memori yang besar untuk menyimpan pohon yang digunakan, hal ini tidak menjadi masalah karena ukuran memori yang sekarang sudah besar.

Walau secara *default* MD6 menggunakan mode operasi berbasis pohon, tetapi MD6 juga mendukung mode operasi sekuensial yang mirip dengan konstruksi Merkle-Damgard. Fleksibilitas mode operasi yang dimiliki MD6 ini membuatnya mudah diimplementasikan dalam setiap jenis software dan hardware yang mungkin. MD6 dikatakan dapat berjalan dengan baik dengan setidaknya 1KB RAM di mana mode opeasinya adalah sekuensial. Dengan begitu, implementasinya dapat disesuaikan dengan kapasitas memori yang dimiliki sistem. Sistem yang memiliki memori lebih besar dapat memanfaatkannya untuk meningkatkan paralelisme dan efisiensi komputasi MD6.

Memori yang besar juga memberikan keleluasaan dalam menentukan ukuran blok data masukan yang lebih besar. Dalam hal ini, MD6 memilih ukuran blok sebesar 512 byte. Ukuran sebesar ini memberikan manfaat tersendiri bagi MD6. Pertama, ia memudahkan penambahan masukan tambahan di mana dengan ukuran masukan yang lebih besar, masukan tambahan akan mengambil bagian yang lebih kecil dari masukan. Kedua, ia memungkinkan pemanfaatan paralelisme dengan lebih maksimal. Dalam MD6, 16 proses dalam satu ronde dapat dieksekusi secara bersamaan. Ketiga, ia dapat mengakomodasi pesan kecil cukup dengan satu blok. Misalnya, ukuran 512 byte merupakan ukuran standar blok pada *harddisk* sehingga hanya diperlukan satu kali pemanggilan fungsi kompresi untuk meng-hash blok tersebut. Keempat, tentunya ia akan mempersulit kriptanalisis karena prosesnya sendiri bertambah banyak.

Hal menarik lainnya yang terdapat pada MD6 yakni bahwa ia memiliki masukan opsional key K. Hal ini membuat MD6 mudah untuk ditambahkan kunci, *salt*, atau nilai indeks untuk mendapat versi lain fungsi hashnya. Hal ini memberikan kemudahan dalam melakukan konversi fungsi hash MD6 menjadi MAC di mana dilakukan pertukaran kunci rahasia antara pengirim dan penerima. Pada MD5, masukan ini tidak tersedia sehingga penambahan tersebut harus dilakukan sendiri. Hal ini tidak baik karena dapat menghasilkan risiko yang tidak diharapkan.

Pada MD6, tampak bahwa operasi komputasi yang digunakan sangatlah sederhana. MD6 hanya menggunakan tiga operasi logika, yakni operasi OR, AND, dan FIXED SHIFT. Operasi lain seperti operasi NOT, OR, dan FIXED ROTATE dapat dilakukan dengan mudah menggunakan operasi logika tersebut. Operasi logika ini memberikan keuntungan karena dapat diimplementasikan dalam waktu tetap dan tidak bergantung pada ukuran word.

Sementara itu, konstanta-konstanta yang digunakan dalam MD6 dipilih menggunakan berbagai macam pertimbangan dengan fokus pada efisiensi dan sekuriti terhadap berbagai macam serangan. Misalnya saja, posisi tap dipilih berdasarkan ketentuan berikut.

- $c < t_0 < t_1 < t_2 < t_3 < t_4 < t_5 = n$.
- Modulo c dari posisi *tap* tidak boleh nol.
- Modulo c dari setiap posisi *tap* harus berbeda satu sama lainnya.
- Posisi *tap* kecuali t_5 harus relatif prima terhadap n.
- Selisih (t_4-t_3) tidak boleh sama dengan selisih (t_2-t_1).

Ketentuan pemilihan *tap* tersebut dibuat dengan tujuan agar nilai Ai pada persamaan (4) bergantung pada nilai masukan A[0] … A[n-1]. Ini perlu agar perubahan pada masukan dijamin benar-benar mengubah keluaran. Hal ini merupakan bagian penting yang harus diperhatikan dalam fungsi hash.

Contoh lainnya yakni pada pemilihan jumlah shift, baik shift kiri maupun shift kanan. Dapat diperhatikan bahwa tidak ada jumlah shift yang bernilai nol. Hal ini dilakukan untuk memastikan bahwa perubahan masukan akan mengakibatkan perubahan keluaran. Menurut data, perubahan satu bit pada masukan fungsi akan mengakibatkan perubahan dua hingga empat bit dari keluaran fungsi. Adapun jumlah shift dipilih sehingga untuk mendapatkan perubahan satu bit output diperlukan perubahan setidaknya lima bit masukan. Sifat ini dikatakan mendukung ketahanan MD6 terhadap *differential attack*.

Dua contoh proses pemilihan konstanta tersebut menunjukkan bahwa konstanta pada MD6 memang dipilih dengan pertimbangan tertentu. Konstanta selain yang telah disebutkan juga memiliki ketentuan pemilihan tersendiri. Pemilihan konstanta semacam ini umumnya ditujukan untuk meningkatkan efisiensi dan sekuriti dari fungsi hash MD6.

5. KESIMPULAN

Berdasarkan pembahasan dan analisis yang dilakukan, dapat ditarik kesimpulan berikut:

4.1 MD6 pada standarnya menggunakan mode operasi berbasis pohon yang memungkinkan paralelisme tinggi dan tetap mendukung mode operasi sekuensial mirip konstruksi Merkle-Damgard sehingga memiliki fleksibilitas tinggi.

4.2 MD6 memiliki ukuran blok input besar yang mampu meningkatkan kemudahan penambahan input tambahan, efisiensi, paralelisme dan sekuriti.

4.3 Berbagai versi MD6 mudah dibuat dengan mengubah-ubah masukan opsional key K, misalnya untuk aplikasi MAC.

4.4 Operasi logika yang digunakan dalam algoritma hash MD6 hanya berupa operasi OR, AND, dan FIXED SHIFT.

4.5 Konstanta pada MD6 bukan merupakan angka yang acak diambil, melainkan dipilih dengan pertimbangan sekuriti dan efisiensi.

REFERENSI

[1] Munir, Rinaldi. 2005. *Diktat Kuliah IF 5054 Kriptografi.* Departemen Teknik Informatika Institut Teknologi Bandung : Bandung.

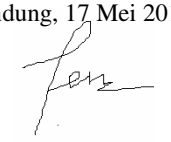
[2] Ronald L. Rivest et Al. 2009. *The MD6 hash function A proposal to NIST for SHA-3*. Computer Science and Artificial Intelligence Laboratory Massachusetts Institute of Technology : Cambridge.

[3] Crutchfield, Christopher Y. 2008. *Security Proofs for the MD6 Hash Function Mode of Operation*. Department of Electrical Engineering and Computer Science Massachusetts Institute of Technology : Cambridge.

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, 17 Mei 2010



Ferdian Thung / 13507127