Analisis Fungsi Hash MD6

Teuku Reza Auliandra Isma / 13507035
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if17035@students.if.itb.ac.id

***Abstract*—Saat ini National Institute of Standards and Technology(NIST) sedang menyelenggarakan kompetisi untuk menentukan fungsi SHA-3. Kompetisi sudah memasuki babak kedua dengan 14 fungsi yang masih berkompetisi. Pada makalah ini akan dibahas beberapa fungsi hash yang mengikuti kompetisi tersebut yaitu fungsi hash MD6.**
Fungsi MD6 dirancang oleh Ron Rivest, dkk. MD6 menggunakan struktur seperti Merkle-tree agar bisa melakukan komputasi hash secara paralel untuk input yang sangat panjang. Berdasarkan klaaim dari pembuat, struktur ini terbukti meningkatkan kecepatan dari fungsi ini. Meskipun begitu, pembuat dari fungsi MD6 mengakui bahwa fungsi ini belum layak untuk mengikuti kompetisi SHA-3 terkait beberapa isu mengenai kecepatan dan ketidakmampuan untuk menjamin keamanan pada versi yang lebih cepat. Fungsi MD6 gagal melaju ke babak 2 dari kompetisi tersebut.

***Index Terms*—NIST, SHA3, MD6, Analisis MD6.**

I. Pendahuluan

Sebuah fungsi hash h adalah fungsi yang memetakan sebuah masukan *M* bertipe bit string menjadi sebuah output string *h*(M) dengan panjang tetap *d*. Output biasanya berukuran lebih kecil dibandingkan dengan masukan yang diterima.

Fungsi hash memiliki banyak aplikasi, diantaranya digunakan pada tanda tangan digital (digital signatures), metode timestamp dan metode untuk pendeteksi modifikasi file.

Untuk bisa berguna pada berbagai aplikasi tersebut, fungsi hash *h* tidak boleh hanya menyediakan sebuah output dengan panjang tetap saja tetapi juga harus memenuhi sifat-sifat berikut::

- One-wayness: Fungsi hash hanya dapat berjalan satu arah
- Collision-resistance: Tidak boleh ada dua pesan berbeda memberikan nilai hash yang sama
- Pseudo-randomness: Fungsi h harus terlihat seperti fungsi acak dari masukannya.

II. Spesifikasi MD6

Bagian ini menjelaskan spesifikasi yang dimiliki oleh fungsi hash MD6.

2.1 Masukan MD6

Bagian ini akan menjelaskan masukan-masukan yang dimiliki oleh MD6. Dua dari masukan tersebut wajib dipenuhi sedangkan tiga sisanya bersifat opsional.

M – Pesan yang akan dihitung nilai hash-nya
d – Panjang message digest (dalam satuan bit) yang diinginkan
K – nilai kunci (opsional)
L – Mode control (opsional)
r – Jumlah putaran (opsional)

Masukan yang bersifat wajib hanyalah pesan *M* yang akan dihitung nilai hash-nya dan panjang message digest *d*. Masukan yang bersifat opsional akan menggunakan nilai yang sudah ditentukan jika tidak diberi masukan yang berbeda.

Kita mendefinisikan H sebagai fungsi hash MD6. Huruf kecil dibawahnya menyatakan masuk yang dimilikinya.

2.1.1 Pesan *M* yang akan dihitung nilai hash-nya

Masukan pertama yang wajib diberikan pada fungsi MD6 adalah pesan *M* yang akan dihitung nilai hash-nya. *M* adalah suatu deret bit dengan panjang terbatas *m* dimana

$$0 \leq m < 2^{64}$$

Panjang *m* tidak perlu diketahui sbelum melakukan hash dengan MD6. MD6 melakukan hash dengan menggunakan tree dan dapat dengan mudah dilakukan secara paralel. Jika seluruh pesan *M* sudah tersedia pada awalnya maka sejumlah processor yang berbeda dapat mulai melakukan operasi hash pada titik mulai yang berbeda pada pesan kemudian hasilnya digabungkan dengan menggunakan tree.

2.1.2 Panjang message digest *d*

Masukan kedua untuk diberikan kepada MD6 adalahh panjang message digest yang diinginkan *d* dari ouput fungsi hash dimana

$$0 < d \leq 512$$

Nilai dari *d* harus diketahui pada saat akan memulai komputasi hash karena nilai tersebut tidak hanya menetukan panjang dari output akhir fungsi MD6 tetapi mempengaruhi komputasi MD6 pada setiap operasi penghitungan hasil sementara.

Perubahan pada nilai *d* akan memberikan nilai hash yang sangat berbeda. Perbedaan tidak hanya terjadi pada

outputt yang memiliki panjang berbeda tetapi juga memberikan nilai hash yang berbeda dan tidak berhubungan dengan nilai hash yang dihitung dari pesan yang sama dengan nilai *d* yang berbeda.

2.1.3 Kunci *K*

Keamanan akan lebih baik jika menggunakan fungsi hash yang lebih lengkap $\{H_{d,K}\}$ yang melakukan indeks tidak hanya dengan ukuran digest d tetapi juga dengan kunci *K* yang terhingga. Kedua nilai ini harus terlihat tidak berhubungan dimana $H_{d,K}$ dan $H_{d,K'}$ tidak memiliki hubungan yang berarti untuk $K \neq K'$.

Pengguna MD6 dapat menyediakan sebagai *K* berukuran *keylen* bytes, untuk setiap panjang *keylen*, dimana

$$0 \leq keylen \leq 64.$$

Nilai awal dari kunci tidak diberikan adalah kunci nil dengan panjang 0. $H_d = H_{d,nil}$.

Kunci *K* juga bisa berfungsi sebagai "salt", seperti yang biasa digunakan pada hash dengan menggunakan password.

Kunci *K* bisa bersifat rahasia. Sehingga, MD6 dapat digunakan untuk menghitung langsung message authentication code atau MAC. MD6 berusaha menjamin bahwa tidak ada informasi berguna mengenai kunci yang dapat diketahui dari output MD6 sehingga akan terlindungi.

Kunci juga dapat berupa nilai yang dipilih secara acak sehingga dapat digunakan untuk aplikasi randomized hashing.

Pada MD6, kunci akan di-padding dengan bytes 0 hingga panjang tepat 64 bytes. Panjang asli *keylen* dari kunci akan disimpan dan digunakan sebagai masukan pelengkap untuk fungsi kompresi MD6.

Panjang maksimum kunci (64 bytes) cukup panjang sehingga memungkinkan sebuah kunci adalah gabungan dari nilai lain yang digunakan untuk tujuan berbeda jika diinginkan.

Jika kunci yang diinginkan lebih panjang dari 512 bits maka kunci tersebut dapat dihintung nilai hash-nya terlebih dahulu nilai *d* 512 bits kemudian hasil hash tersebut dapat digunakan sebagai kunci.

2.1.4 Mode control *L*

Mode operasi standar untuk MD6 adalah menggunakan tree dan bersifat hierarki.

Data dari pesan yang akan dihitung nilai hash-nya ditempatkan pada daun dari sebuah pohon 4-ary yang cukup besar. Komputasi dilakukan dari daun hingga akar. Setiap node bukan daun dari phon adalah sebuah eksekusi fungsi kompresi yang mengambil input sebanyak 64 words dan menghasilkan output sebesar 16 words dengan 1 word adalah 8 byte. Sebanyak *d* bits terakhir dari output yang dihasilkan pada akar diambil sebagai keluaran fungsi hash.

Berbagai pilihan bentuk tree dapat digunakan dengan menggunakan parameter mode operasi *L* yang bersifat opsional. Dengan menggunakan *L* yang bervariasi, MD6 akan memiliki bentuk yang bervariasi antara mode operasi sekuensial yang low-memory ($L = 0$) dan mode operasi menggunakan tree dengan memanfaatkan komputasi parale semaksimal mungkin ($L = 64$).

Mode operasi yang standar memiliki $L = 64$, untuk operasi hierarki secara penuh. Sebenarnya, setiap nilai $L \geq 27$ akan memberikan penghitungan hash secara hierarki. Nilai $L = 64$ digunakan sebagai standar dengan tujuan menggunakan nilai yang cukup besar dimana mode operasi sekuensial tidak akan pernah terjadi.

2.1.5 Jumlah putaran *r*

Fungsi kompresi MD6 *f* memiliki jumlah putaran r yang dapat dikendalikan. Secara kasarnya, setiap puataran mengacu pada satu clock cycle pada hardware implementation atau 16 langkah dalam software implementation.

Nilai awal dari r adalah

$$r = 40 + \lfloor d/4 \rfloor$$

sehingga $H_{d,K,L} = H_{d,K,L,40 + \lfloor d/4 \rfloor}$. Untuk $d = 160$, MD6 akan memiliki nilai awal $r = 80$ putaran; untuk $d = 512$, MD6 akan memiliki nilai awal $r = 168$ putaran. Putaran dapat ditambah untuk meningkatkan keamanan ataupun mengurangi putaran untuk meningkatkan kinerja dengan trade off antara keamanan dan kinerja.

Meskipun begitu, ketika menggunakan MD6 yang memiliki kunci kita membutuhkan nilai $r \geq 80$. Hal ini dilakukan untuk memberikan perlindungan pada kunci, bahkan ketika output yang dihasilkan pendek(seperti pada MAC). Sehingga, ketika MD6 memiliki kunci yang tidak kosong, nilai awal *r* adalah

$$r = \max(80, 40 + \lfloor d/4 \rfloor).$$

2.1.6 Parameter lain dari MD6

Beberapa parameter lain dari fungsi MD6 juga dapat divariasikan (e.g. w, Q, c, $t_0 \ldots t_5$, r_i, l_i, S_i untuk $0 \leq i < rc$). Tetapi berbagai varian dari MD6 yang menggunakan pengaturan berbeda untuk berbagai parameter tersebut masih dipelajari dan diuji keamanannya.

2.2 Keluaran MD6

Keluaran dari fungsi hash MD6 adalah bit string *D* dengan panjang *d* bits:

$$D = H_{d,K,L,r}(M);$$

D adalah nilai hash dari pesan masukan *M* atau dapat disebut juga sebagai message digest.

2.3 Mode Operasi MD6

Sebuah fungsi hash pada umumnya terbentuk dari sebuah fungsi kompresi, yang memetakan masukan dengan panjang tetap menjadi keluaran dengan panjang tetap yang lebih kecil. Sebuah mode operasi kemudian akan menentukan bagaimana sebuah fungsi kompresi dapat digunakan berulangkali untuk menghitung nilai hash dari sebuah masukan untuk menghasilkan keluaran dengan panjang tertentu.

Untuk menjelaskan sebuah fungsi hash, seseorang harus menjelaskan hal berikut:

- Mode operasi yang digunakan
- Fungsi kompresi yang digunakan
- Berbagai kosntanta yang digunakan pada komputasi

Fungsi kompresi MD6 f mengambil masukan dengan panjang tetap (n = 89 words), dan menghasilkan keluaran dengan panjang yang tetap tapi lebih kecil (c = 16 words):

$$f: W^{89} \rightarrow W^{16}.$$

Masukan 89-word yang diberikan pada fungsi f memilikikonstanta 15-word Q, kunci 8-word K, ID unik berukuran 1 word U, control 1-word V, dan blok data 64-word B.

Karena Q adalah sebuah konstanta, maka fungsi kompresi MD6 yang efektif f_Q memetakan masukan 74-word menjadi keluaran 16-word:

$$f: W^{74} \rightarrow W^{16}.$$

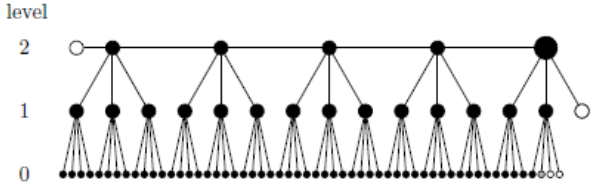
Sehingga, fungsi kompresi MD6 berhasil mereduksi ukuran blok data sebesar empat kali.

Bagian berikutnya akan membahas mode operasi MD6 dan fungsi kompresi MD6.

2.3.1 Mode operasi hierarki

Mode operasi standar untuk MD6 menggunakan tree. Implementasi dari mode hierarki ini membutuhkan kapasitas penyimpanan sesuai dengan tinggi dari pohon yang dibuat.

Mengingat beberapa perangkat yang sangat kecil tidak memiliki kapasitas penyimpanan yang memadai, MD6 menyediakan parameter permbatas tinggi L. Ketika tinggi pohon mencapai L + 1, MD6 akan bertukar dari operator kompresi paralel menjadi operator kompresi sekuensial.



Gambar 1. Pohon yang dibentuk dari L yang diubah

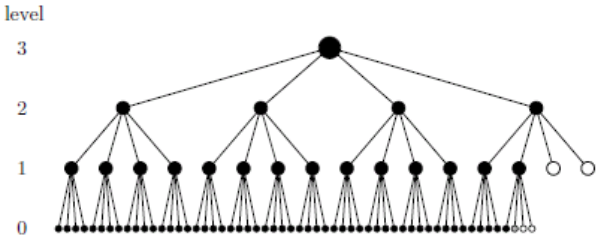
Mode operasi MD6 dapat diatur berdasarkan bilangan bulat L, $0 \leq L \leq 64$, yang memberikan perpindahan secara halus dari mode operasi standar dengan menggunakan pohon hierarki menjadi sebuah mode operasi iteratif.

Pada penjelasan mode ooperasi MD6, MD6 melewatkan data secara paralel sebanyak L, dimana pada setiap prosesnya mengurangi ukuran data sebanyak kelipatan 4 dan melewatkan data secara sekuensial sebanyak sekali untuk menyelesaikannya.

Karena ukuran masukan harus kurang dari 2^{64} bits dan fungsi kompresi yang terakhir akan menghasilkan keluaran berukuran 2^{10} = 1024 bits maka akan ada 27 proses paralel.

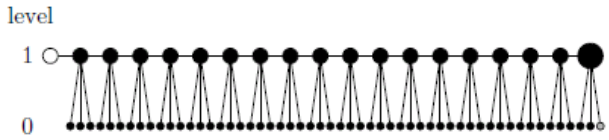
Secara umumnya, MD6 akan membuat barisan pohon 4-ary dengan tinggi maksimal i yang masing-masing memili 4^L daun berukuran 16 words dimana semuanya kemudian digabungkan menjadi sebuah nilai pada root.

Jika 4^L lebih besar dari jumlah blok pada pesan masukan maka hanya akan ada satu pohon yang dibuat dan MD6 akan menjadi metode yang menggunakan pohon secara murni.



Gambar 2. Fungsi kompresi dengan L = 64.

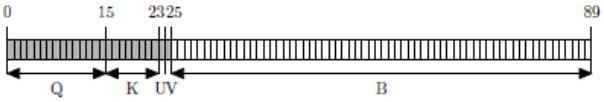
Sebaliknya, jika L =0 maka tidak akan ada pohon yang dibuat dan masukan akan dibagi menjadi blok berukuran 48 word yang digabungkan secara sekuensial.



Gambar 3. Fungsi kompresi dengan L = 0.

2.3.2 Fungsi Kompresi

Mode operasi dari MD6 memberikan format pada masukan untuk fungsi kompresi f dengan cara berikut.



Gambar 4. Format blok data

Empat parameter pertama Q, K, U, V, adalah masukan pelengkap sedangkan B adalah data.

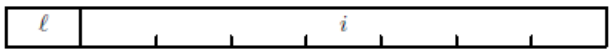
Q – sebuah vektor konstanta dengan panjang 15 word
K – sebuah kunci dengan panjang 8 word
U – sebuah ID unik dengan panjang 1 word
V – sebuah parameter control dengan panjang 1 word
B – sebuah blok data dengan panjang 64 word

2.3.2.1 ID Unik U

ID unik U adalah sebuah masukan pelengkap pada fungsi kompresi dengan panjang 1 word yang dibuat dari l dan i. ID ini dengan unik menjelaskan operasi fungsi kompresi yang sedang dilakukan dengan memberikan level dari operasi ini dan indeksnya pada level tersebut.

l – satu byte yang menyatakan level

i – tujuh byte yang menunjukkan posisi pada suatu level, dengan operasi fungsi kompresi paling kiri memiliki i = 0.



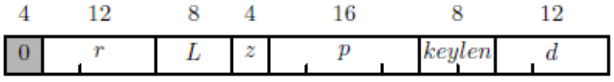
Gambar 5. Format U

Sebagai contoh, operasi kompresi yang pertama selalu memiliki U = (l, i) = (1,0). Sebuah node (l, i) pada level l,

$1 < l \leq L$ memiliki anak pada $(l-1, 4i)$, $(l-1, 4i+1)$, $(l-1, 4i+2)$ dan $(l-1, 4i+3)$.

2.3.2.2 Control Word V

Control word V adalah sebuah masukan pelengkap berukuran 1 word yang memberikan parameter relevan dengan komputasi.



Gambar 6. Control Word V

r – jumlah putaran pada fungsi kompresi (12 bits)

L – parameter mode (8 bits)

z – bernilai 1 jika ini adalah operasi kompresi yang terakhir dan bernilai 0 jika tidak (4 bits)

p – jumlah bits dari data yang dipadding pada blok *B* masukan (16 bits)

keylen – panjang dari kunci *K* (8 bits)

d – panjang message digest keluaran yang diinginkan (12 bits)

2.4 Fungsi Kompresi MD6

Fungsi kompresi *f* mengambil sebagai masukan sebuah array *N* dengan panjang $n = 89$ word. Kemudian menghasilkan keluaran sebuah array *C* dengan panjang $c = 16$ word.

Disini *f* dideskripsikan memiliki sebuah masukan 89-word *N*, meskipun hal itu juga dapat dilihat sebagai masukan pelengkap 25-word ($Q \| K \| U \| V$) diikuti oleh blok data 64-word *B*.

Fungsi kompresi *f* dapat dilihat pada gambar 7.

The MD6 compression function f

Input:

N: An input array $N[0..n-1]$ of $n = 89$ words. (This typically consists of a 25-word "auxiliary information" block, followed by a a $b = 64$-word data block B, but all inputs are treated uniformly here.)

r : A non-negative number of rounds.

Output:

C: An output array $C[0..c-1]$ of length $c = 16$ words.

Parameters: $c, r, t_0, t_1, t_2, t_3, t_4, r_i, \ell_i, S_i$, where $1 \leq t_i \leq n$ for all $0 \leq i \leq 4$, $0 < r_i, \ell_i \leq w/2$ for all i, and S_i is a w-bit word for each i, $0 \leq i < t$.

Procedure:

Let $t = rc$. (Each round has $c = 16$ steps.)

Let $A[0..t+n-1]$ be an array of $t+n$ words.

Initialization:

$A[0..n-1]$ is initialized with the input $N[0..n-1]$.

Main computation loop:

for $i = n$ **to** $t+n-1$: /* t steps */

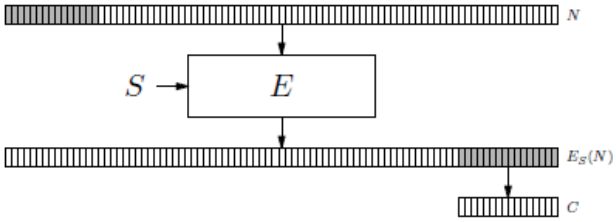
$$\begin{aligned} x &= S_{i-n} \oplus A_{i-n} \oplus A_{i-t_0} && \text{[line 1]} \\ x &= x \oplus (A_{i-t_1} \wedge A_{i-t_2}) \oplus (A_{i-t_3} \wedge A_{i-t_4}) && \text{[line 2]} \\ x &= x \oplus (x \gg r_{i-n}) && \text{[line 3]} \\ A_i &= x \oplus (x \ll \ell_{i-n}) && \text{[line 4]} \end{aligned}$$

Truncation and Output:

Output $A[t+n-c..t+n-1]$ as the output array $C[0..c-1]$ of length $c = 16$.

Gambar 7. Fungsi Kompresi *f*

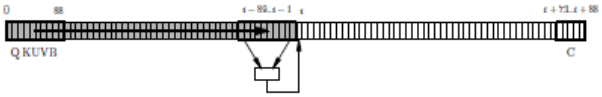
Fungsi kompresi bisa dilihat sebagai proses enkripsi dari *N* dengan menggunakan kunci *S* diikuti oleh operasi pemotongan yang hanya mengembalikan 16 word terakhir dari $E_S(N)$. Disini *S* menentukan konstanta putaran pada algoritma enkripsi.



Gambar 8. Proses enkripsi

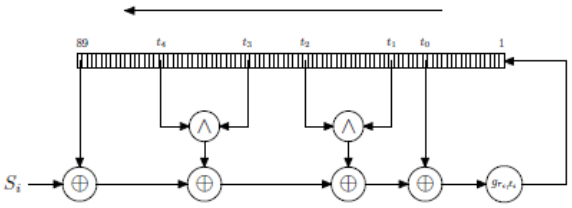
Fungsi kompresi memiliki loop utama sebanyak *r* putaran(masing-masing pada setiap anggota $c = 16$ langkah), diikuti operasi pemotongan yang memotong hasil akhir menjadi 16 word.

Loop utama kemudian berulang sebanyak $t = rc$ langkah, dimana pada setiap langkah menghitung satu word. Loop ini dapat diimplementasikan dengan menyimpan input ke *n* word pertama dari sebuah array *A* dengan panjang $n + t$, kemudian menghitung setiap *t* word yang tersisa sesuai giliran.



Gambar 9. Loop utama fungsi kompresi

Selain itu, loop ini juga dapat diimplementasikan sebagai nonlinear feedback shift register dengan sebanyak 89 word.



Gambar 10. Implementasi menggunakan nonlinear feedback shift register

Operasi pemotongan akan mengembalikan 16 word terakhir dari *A* sebagai keluaran fungsi kompresi. Fungsi kompresi selalu menghasilkan output $c = 16$ word (1024 bits). Ukuran ini minimal akan menghasilkan keluaran dua kali dari berbagai bekmungkinan keluaran MD6.

Fungsi kompresi mengambil posisi feedback tap t_0, t_1, t_2, t_3, ,t_4, masing-masing berada di antara 1 hingga $n - 1 = 88$, sebagai parameter.

MD6 constants

Tap positions:

t_0	t_1	t_2	t_3	t_4
17	18	21	31	67

Shift amounts:
The values r_{i-n}, ℓ_{i-n} are the initial right-shift amount and the subsequent left-shift amount within the step with index i; the index $i-n$ is taken modulo $c = 16$.

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

Round constants:
The values S_{i-n} for $i \geq n$ are determined as follows. Because S_{i-n} is constant within a round, but changes from round to round, S_{i-n} is defined in terms of an auxiliary sequence S'_j. The constants S'_0 and S^* determine the entire sequence.

$$S_{i-n} = S'_{\lfloor (i-n)/16 \rfloor}$$

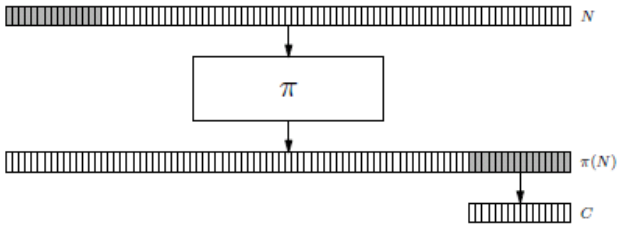
$$S'_0 = \texttt{0x0123456789abcdef}$$

$$S^* = \texttt{0x7311c2812425cfa0}$$

$$S'_{j+1} = (S'_j \lll 1) \oplus (S'_j \wedge S^*) \,.$$

Gambar 11. Konstanta MD6

Melakukan xor pada sebuah nilai yang sudah diberikan operasi shift bersifat invertible atau tidak dapat dikembalikan. Sehingga pemetaan di dalam *f* pada *n*-word pertama dari A hingga *n* word terakhir tidak dapat dikembalikan. Hal ini mengacu pada pemetaan π pada gambar berikut.



Gambar12. Sudut pandang alternatif untuk fungsi kompresi *f*

2.4.1 Langkah, putaran dan rotasi

Langkah $t = rc$ dapat diperkirakan sebagai barisan dari rotasi dimana setiap rotasi tersebut terdiri dari $n = 89$ langkah dan satu siklus penuh akan mengisi feedback shift register sebanyak *n*-word dengan menghitung *n* word selanjutnya pada state tersebut. Kita mengetahui bahwa satu rotasi terdiri dari $n/c = 89/16 = 5\ 9/16$ putaran dan dua rotasi mengacu pada sekitar 11 putaran.

Setiap rotasi menghasilkan vektor n-word yang bar dengan komputasi yang dapat dilihat sebagai rotasi *rc*/*n*, dimana masing-masing menghasilkan sebuah vektor *n*-word yang invertible dari vektor *n*-word sebelumnya. Masukannya adalah vekntor *n*-word pertama dan keluarannya adalah word terakhir dari *c* pada vektor *n*-word terakhir.

Fungsi kompresi juga dapat dilihat sebagai *r* putaran 16 langkah dimana masing-masing menghitung 16 word selanjutnya.

Konstanta feedback MD6 diatur sedemikian hingga menampakkan sudut pandang yang berorientasi putaran(round-oriented). Konstanta putaran S_i tetap sama untuk seluruh 16 langkah dari sebuah putaran dan berubah untuk putaran selanjutnya. Jumlah pergeseran *r*(.) dan *l*(.) berbeda dalam setiap putaran tetapi kemudian memiliki pola yang sama dengan variasi pada putaran selanjutnya.

Untuk implementasi MD6 pada perangkat lunak, akan lebih mudah melakukan 16 kali perputaran loop sehingga setiap putaran diimplementasikan sebagai blok kode standar tanpa pencabangan kondisional.

2.4.2 Intra-word Diffusion via xorshifts

Beberapa metoda dibutuhkan untuk mempengaruhi diffusion antara posisi bit yang bervariasi pada sebuah word. Intra-word diffusion diberikan dari $g_{ri,li}$ operator implisit di loop pada gambar 7:

$$g_{r_i,\ell_i}(x) = \{\ y = x \oplus (x \gg r_i);\ \textbf{return}\ y \oplus (y \ll \ell_i)\ \}$$

Parameter l_i dan r_i mengacu pada jumlah pergeseran yang dilakukan.

Fungsi *g* bersifat linear dan invertible atau tidak dapat dikembalikan.

Operator $x = x\ (+)\ (x >> r_i)$ dan $x = x\ (+)\ (x >> l_i)$ dikenal sebagai operator xorshift yang umumdigunakan pada fungsi hash.

2.4.3 Jumlah pergeseran

Nilai pergeseran adalah indeks mod $c = 16$. Nilai yang diberikan disini adalah hasil dari eksperimen menggunakan 1 juta tabel yang dibangkitkan secara random dari nilai pergeseran tersebut.

2.4.4 Konstanta putaran

Konstanta putaran S'_j memberikan beberapa perbedaan antara setiap putaran. Setiap putaran j memiliki konstanta S'_j yang berbeda. Setiap langkah pada satu putaran menggunakan konstanta putaran yang sama.

Persamaan berikut membangkitkan konstanta tersebut:

$$\begin{aligned} S'_0 &= \texttt{0x0123456789abcdef} \\ S^* &= \texttt{0x7311c2812425cfa0} \\ S'_{j+1} &= (S'_j \lll 1) \oplus (S'_j \wedge S^*)\,. \end{aligned}$$

Persamaan ini memiliki hubungan yang dapat dibalik. Dari S'_j seseorang dapat menentukan S'_{j+1} dan sebaliknya.

III. Analisis MD6

3.1 Security

Rancangan keseluruhan dari fungsi hash MD6 sudah baik dari segi keamanan. Rancangan penuh dari MD6 masih tahan terhadap collision dan memiliki sifat pseudo-randomeness. Meskipun begitu, masih ada beberapa isu keamanan untuk versi yang lebih ringan.

Pada MD6 yang memiliki putaran sebanyak 16, ditemukan collision. Kecepatan untuk menemukan collision pada MD6 dengan putaran yang dikurangi menjadi 16 putaran adalah 2^{17}.

Pada MD6 dengan putaran yang dikurang menjadi 33

putaran, terdapat permasalahan pada pseudo-randomness. Pada MD6 yang dikurangi tersebut muncul isu non-randomness.

3.2 Implementasi

Fungsi hash ini diimplementasikan dengan menggunakan bahasa C. Program bersifat text-based tanpa GUI. Berikut hasil hash untuk kata "abc" dengan parameter default:

Gambar 13.
Nilai hash untuk kata abc dengan parameter default
230637d4e6845cf0d092b558e87625f03881dd53a7439
da34cf3b94ed0d8b2c5

3.3 Kecepatan

Fungsi hash MD6 sebenarnya masih memiliki masalah kecepatan pada versi lengkapnya. Beberapa pengujian menunjukkan bahwa komputasi hash MD6 masih lebih lambat dibandingkan fungsi hash lain meskipun diuji pada processor dengan core lebih dari satu dimana arsitektur seperti itu akan mendukung komputasi paralel yang digunakan MD6.

32-bit Core 2 Duo 2.4GHz (T61)			
	XP + MSVS 2005		
	ticks/byte	ticks/comp. fn	speed (MB/sec)
MD6-160	54	20855	44.1
MD6-224	63	24363	37.7
MD6-256	68	26464	34.7
MD6-384	87	33607	27.4
MD6-512	106	40975	22.4

hash algorithm	speed (MB/sec)
MD4	509
MD5	370
SHA-1	209
SHA-224	108
SHA-256	106
SHA-384	37
SHA-512	38
RIPEMD-160	138
Tiger	108
HAVAL (5 passes)	171
Whirlpool	30

Gambar 14. Hasil uji kecepatan

IV. Conclusion

Fungsi hash MD6 memiliki rancangan yang sangat bagus dan cukup kompleks. Pemanfaatan komputasi paralel yang sedang berkembang saat ini sangat menarik untuk diperhatikan. Meskipun begitu, masih ada permasalahan keamanan pada versi yang lebih cepat. Selain itu, para perancang MD6 juga mengakui masih adanya kelemahan pada rancangan MD6 ini. Mungkin hal itulah yang menjadi penyebab tidak lolosnya fungsi hash ini pada kompetisi SHA-3.

References

[1] A. Regenscheid, R. Pelner, S. Chang, J. Kelsey, M. Nandi, S. Paul, "Status Report on the First Round of the SHA-3 Cryptographic Hash Algorithm Competition," NISTIR 7620, Sep 2009.

[2] Ronald L. Rivest et Al., "The MD6 Hash Function," Crypto 2008.

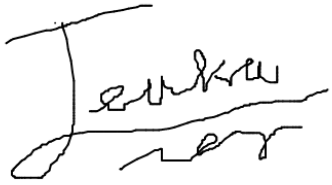
[3] T. Hodanek, "Analysis of reduced MD6," Western European Workshop on research in Cryptology 2009.

[4] http://group.csail.mit.edu/cis/md6/Rivest-TheMD6HashFunction.ppt. Waktu akses: 15 Mei 2010.

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



Teuku Reza Auliandra Isma / 13507035