Studi dan Implementasi Kolisi pada Fungsi Hash MD5

Ade Gunawan – 13504049

Teknik Informatika
Sekolah Teknik Elekltro dan Informatika
Institut Teknologi Bandung
2008

Abstrak – *Makalah ini membahas masalah pencarian kolisi pada fungsi hash MD5. Fungsi hash adalah fungsi yang menerima masukan pesan dengan panjang sembarang dan mengkonversinya menjadi string keluaran dengan panjang (fixed) dan umumnya berukuran jauh lebih kecil daripada ukuran string semula. Keluaran dari fungsi hash ini disebut juga sebagai message digest. Kolisi pada fungsi hash terjadi jika ada dua pesan yang berbeda yang memiliki message digest yang sama.*

MD5, salah satu fungsi hash yang paling sering digunakan saat ini, sudah tidak lagi aman karena beberapa kelemahannya sudah ditemukan. Karena fungsi hash sering digunakan untuk menjaga keamanan (security) pengiriman data seperti otentikasi dan integritas pesan, penemuan kolisi akan memberikan efek yang sangat besar terhadap keamanan pengiriman data. Berdasarkan hal tersebut, pada makalah ini, dilakukan studi dan implementasi kolisi pada fungsi hash MD5.

Perangkat lunak yang dikembangkan bernama MD5 Clone yang memiliki fungsi utama untuk membangkitkan pasangan pesan dengan panjang 1024 bit yang menghasilkan message digest yang sama. Algoritma yang digunakan pada pencarian kolisi ini pertama kali diperkenalkan oleh Wang Xiaoyun dan selanjutnya dimodifikasi oleh Vlastimil Klima menjadi lebih efisien. Perangkat lunak ini dikembangkan pada sistem operasi Windows XP Home Edition dengan bahasa pemrograman C dan kompilator MinGW 5.1.4. Perangkat lunak yang telah dibangun telah dapat membangkitkan pasangan-pasangan pesan yang menghasilkan message digest yang sama dalam waktu rata-rata 30.94 menit. Hal ini membuktikan bahwa pencarian kolisi pada fungsi hash MD5 dapat dilakukan secara efisien dalam waktu yang cukup cepat.

Kata kunci: *MD5, fungsi hash, kolisi, message digest, serangan diferensial*.

1. PENDAHULUAN

Pada era teknologi informasi ini, pengiriman data dan informasi menjadi hal yang sangat penting. Lebih-lebih dengan adanya internet yang mampu memfasilitasi pengambilan data dan informasi dengan mudah dan cepat dan mampu menjangkau seluruh dunia. Dengan adanya arus informasi yang begitu pesat ini, keamanan (security) menjadi hal yang sangat penting dalam melakukan pengiriman data.

Dalam pengambilan data, sering kali pengguna membutuhkan sesuatu yang dapat meyakinkan mereka bahwa data yang diambilnya tersebut adalah data yang benar. Salah satu cara yang saat ini sering dipakai adalah dengan menggunakan tanda tangan digital (digital signature). Dengan adanya tanda tangan digital, pengguna dapat yakin bahwa data yang diambilnya benar dengan mencocokkan message digest hasil fungsi hash dari data yang diterimanya tersebut dengan hasil dekripsi dari tanda tangan digital yang diberikan oleh sumber.

Fungsi *hash* itu sendiri sering dipakai pada aplikasi lain yang bertujuan untuk menjaga keamanan seperti otentikasi dan integritas pesan. Fungsi *hash* adalah fungsi yang menerima masukan *string* dengan panjang sembarang dan mengkonversinya menjadi *string* keluaran yang panjangnya tetap (*fixed*) dan umumnya berukuran jauh lebih kecil daripada ukuran *string* semula [MUN06]. Keluaran dari fungsi *hash* ini disebut juga sebagai *message digest*.

Karena umumnya tanda tangan digital menggunakan fungsi *hash* untuk menentukan keamanan sebuah data, tentunya kemampuan dari tanda tangan digital ini sangat bergantung pada kekuatan dari fungsi *hash* yang dipakainya. Salah satu syarat penting sebuah fungsi *hash* sehingga fungsi tersebut dapat dikatakan sebagai fungsi *hash* yang baik adalah bahwa tidak mungkin (secara komputasi) dapat ditemukan pasangan data yang berbeda yang menghasilkan sebuah *message digest* yang sama (*strong collision resistance*). Perlu disadari bahwa hubungan antara data dan *message digest*-nya pasti bukan korespondensi satu ke satu sehingga dapat disimpulkan bahwa kolisi pasti ada. Namun, ada permasalahan karena untuk menemukan sebuah kolisi tersebut dengan cara *brute force*, diperlukan waktu tahunan bahkan dengan komputer tercanggih saat ini. Berdasarkan fakta tersebut, cara efisien yang dapat menemukan kolisi pada sebuah fungsi *hash* akan menjadi hal yang menarik untuk dibahas.

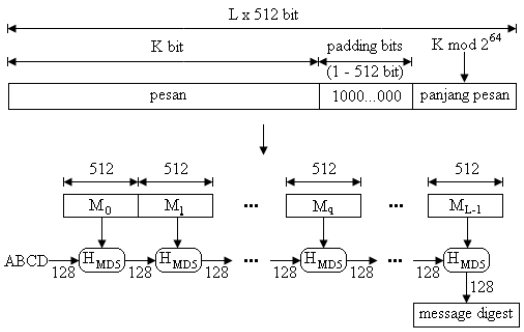
MD5, salah satu fungsi hash yang paling sering digunakan saat ini, sudah tidak lagi aman karena beberapa kelemahannya sudah ditemukan. Pada tahun 1994, Bert den Boer dan Antoon Bosselaers menemukan semacam kolisi semu (pseudo-collision) pada MD5 [BOE94]. Pada tahun 1996, Hans Dobbertin berhasil menemukan kelemahan pada MD5 [DOB96-a] dan serangan tersebut dia jelaskan pada [DOB96-b]. Setelah mengetahui serangan yang dilakukan Hans Dobbertin, Wang Xiaoyun kemudian dapat menemukan kolisi MD5 yang sebenarnya dalam hitungan jam dalam [WAN04] dan [WAN05]. Pada akhirnya pekerjaannya menginspirasi berbagai orang untuk melakukan peningkatan performa seperti yang telah dilakukan oleh [KLI05] dan [SAS05]. Bedasarkan hal tersebut, dibuatlah tugas akhir yang berjudul "Studi dan Implementasi Kolisi pada fungsi hash MD5" dengan harapan untuk mengetahui bagaimana cara menemukan kolisi pada fungsi hash MD5 secara efisien. Karena, penemuan kolisi pada fungsi hash MD5 akan menimbulkan berbagai kemungkinan penyerangan keamanan pengiriman data yang menggunakan fungsi hash MD5.

2. LANDASAN TEORI

2.1. Fungsi Hash MD5

MD5 adalah salah satu fungsi *hash* yang paling sering digunakan saat ini. Fungsi *hash* ini dirancang oleh Ronald Rivest pada tahun 1992. MD5 itu sendiri sebenarnya dibuat sebagai perbaikan dari MD4 yang kolisinya telah berhasil ditemukan [RIV92].

MD5 menerima masukan pesan dengan panjang sembarang dan menghasilkan keluaran berupa *message digest* dengan panjang 128 bit dengan menggunakan konstruksi Merkle Damgård. Gambaran umum pembuatan *message digest* dengan algoritma MD5 diperlihatkan pada Gambar 1 [MUN06]:



Gambar 1 Pembuatan *message digest* dengan algoritma MD5

Secara garis besar, langkah-langkah pembuatan *messages digest* dengan fungsi *hash* MD5 adalah sebagai berikut:

1. Penambahan bit-bit pengganjal.

   Pesan ditambah dengan sejumlah bit pengganjal sedemikian sehingga panjang pesan (dalam satuan bit) kongruen dengan 448 modulo 512 yang berarti 64 bit kurang dari kelipatan 512. Bit-bit pengganjal itu sendiri terdiri dari sebuah bit 1 yang sisanya diikuti dengan bit 0.

2. Penambahan nilai panjang pesan semula.

   Setelah ditambahkan bit-bit pengganjal, pesan akan ditambah lagi dengan 64 bit yang menyatakan panjang pesan semula. Jika panjang pesan melebihi 2^{64}, nilai yang dimasukkan adalah panjangnya yang dimodulo dengan 2^{64}.

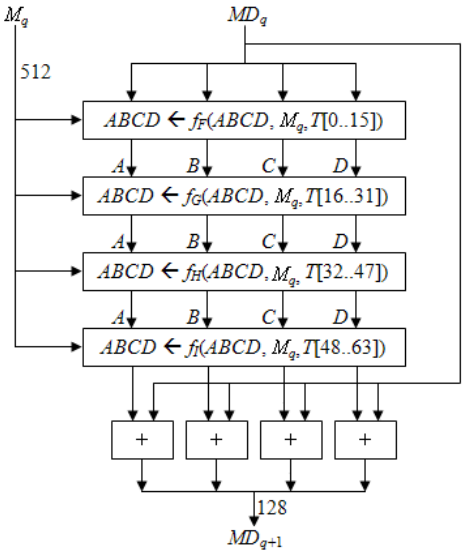
3. Inisialisasi penyangga (*buffer*) MD.

   MD5 membutuhkan 4 buah penyangga (*buffer*) yang masing-masing panjangnya 32 bit. Keempat penyangga ini menampung hasil antara dan hasi akhir. Setiap penyangga diinisialisasi dengan nilai-nilai (dalam notasi HEX) sebagai berikut [RIV92]: *A*=67452301, *B*=EFCDAB89, *C*=98BADCFE, *D*=10325467.

Pengolahan pesan dalam blok berukuran 512 bit.

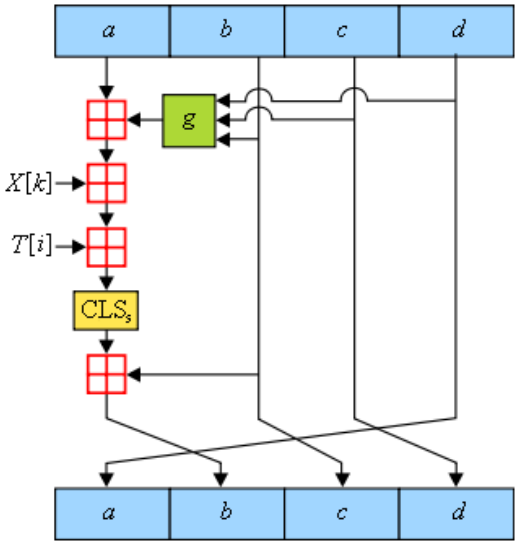
2.2. Pengolahan Pesan

Pesan dibagi menjadi L buah blok yang masing-masing panjangnya 512 bit (Y_0 sampai Y_{L-1}). Setiap blok diproses bersama penyangga MD menjadi keluaran 128-bit, dan ini disebut proses H_{MD5}. Proses H_{MD5} ini terdiri dari 4 putaran (*round*), dan masing-masing putaran melakukan operasi dasar MD5 sebanyak 16 kali dan setiap operasi dasar memakai sebuah elemen *T*. Gambaran proses ini diperlihatkan pada Gambar 2 [MUN06]. Pada Gambar 3 ini, M_q menyatakan blok ke-*q* dari pesan. MD_q adalah nilai *message digest* 128-bit dari proses H_{MD5} ke-*q*. Pada awal proses, MD_q berisi nilai inisialisasi penyangga MD.



Gambar 2 Pengolahan blok 512 bit (Proses H_{MD5})

Fungsi-fungsi f_F, f_G, f_H, dan f_I masing-masing berisi 16 kali operasi dasar terhadap masukan. Setiap operasi dasar menggunakan elemen tabel T. Operasi dasar MD5 diperlihatkan pada Gambar 3 (diambil dari www.wikipedia.org):



Gambar 3 Operasi dasar MD5 dengan pergeseran penyangga ke kanan secara sirkuler

Operasi dasar MD5 yang diperlihatkan pada gambar 3 dapat ditulis dengan sebuah persamaan:

$$b \leftarrow b + \mathrm{CLS}_s(a + g(b, c, d) + X[k] + T[i]) \quad \ldots\ldots\ldots (1)$$

yang dalam hal ini:

a, b, c, d = empat buah penyangga 32-bit (berisi nilai penyangga A, B, C, D)
g = salah satu fungsi F, G, H, I
CLS_s = *circular left shift* sebanyak s bit (notasi: $<<< s$)
$X[k]$ = kelompok 32-bit ke-k dari blok 512 bit *message* ke-q ($0< k<15$)
$T[i]$ = elemen Tabel T ke-i (32 bit) + = operasi penjumlahan modulo 32

Selanjutnya, setiap kali selesai satu operasi dasar, penyangga-penyangga tersebut digeser ke kanan secara sirkuler.

Fungsi f_F, f_G, f_H, dan f_I adalah fungsi untuk memanipulasi masukan a, b, c, dan d dengan ukuran 32 bit. Masing-masing fungsi dapat dilihat pada Tabel 1.

Tabel 1 Fungsi-fungsi dasar MD5

Nama	Notasi	$g(b, c, d)$	Putaran
f_F	$F(b, c, d)$	$(b \wedge c) \vee (\sim b \wedge d)$	1
f_G	$G(b, c, d)$	$(b \wedge d) \vee (c \wedge \sim d)$	2
f_H	$H(b, c, d)$	$b \oplus c \oplus d$	3
f_I	$I(b, c, d)$	$c \oplus (b \oplus \sim d)$	4

Nilai-nilai $T[i]$ diperoleh dengan fungsi $T[i] = 2^{32}$ x abs(sin(i+1)), dengan i adalah sudut dalam radian. Nilai k pada setiap operasi dasar dapat disajikan secara matematis dengan persamaan:

$k = i$	untuk $i < 16$
$k = (5i + 1) \bmod 16$	untuk $16 \le i < 32$
$k = (3i + 5) \bmod 16$	untuk $32 \le i < 48$
$k = 7i \bmod 16$	untuk $48 \le i < 64$

Sedangkan untuk nilai s pada CLS_s dapat ditemukan dengan menggunakan Tabel 2.

Tabel 2 Nilai s pada operasi dasar ke-i

s		i div 16			
		0	1	2	3
i mod 4	0	7	5	4	6
	1	12	9	11	10
	2	17	14	16	15
	3	22	20	23	21

2.3. Serangan Diferensial

Serangan diferensial adalah serangan yang memakai perbedaan (*difference*) data yang

dihasilkan oleh sebuah proses sebagai indikator keberhasilan serangan. Sebuah *difference* untuk dua parameter X and X' didefinisikan sebagai $\Delta X = X' - X$. Untuk dua pesan sembarang M and M' yang dibagi menjadi k blok $M = (M_0, M_1, \ldots, M_{k-1})$, $M' = (M_0', M_1', \ldots, M_{k-1}')$, sebuah *full differential* untuk sebuah fungsi *hash* didefinisikan pada persamaan (2):

$$\Delta H_0 \xrightarrow{(M_0, M'_0)} \Delta H_1 \xrightarrow{(M_1, M'_1)} \ldots \Delta H_{k-1} \xrightarrow{(M_{k-1}, M'_{k-1})} \Delta H \ldots (2)$$

Dimana ΔH_0 adalah nilai *difference* awal yang pasti sama dengan 0, ΔH adalah hasil *difference* akhir untuk kedua pesan tersebut, sedangkan ΔH_i adalah hasil *difference* untuk iterasi ke-i yang juga merupakan nilai *difference* awal untuk iterasi berikutnya.

2.4. Notasi

Sebelum masuk ke serangan terhadap MD5, akan dikenalkan notasi-notasi yang akan digunakan:

1. $M = (m_0, m_1, \ldots, m_{15})$ dan $M' = (m'_0, m'_1, \ldots, m'_{15})$ adalah dua pesan dengan panjang 512 bit (panjang blok yang dipakai dalam MD5) yang dibagi dalam 16 bagian dengan panjang masing-masing 32 bit (*word*).

2. $\Delta M = (\Delta m_0, \Delta m_1, \ldots, \Delta m_{15})$ adalah *difference* yang diperoleh dari hasil pengurangan M' dengan M, sedangkan $\Delta m_i = m'_i - m_i$ itu sendiri adalah *difference* dari *word* ke-i.

3. $Q_{t+1} = Q_t + (f_t(Q_t, Q_{t-1}, Q_{t-2}) + W_t + k_t + Q_{t-3})<<<s_t$ adalah hasil dari operasi dasar MD5 ke-t dengan W_t adalah *word* ke-t dari pesan, k_t adalah nilai ke-t dari tabel T, dan f_t adalah fungsi dasar MD5 sesuai dengan putarannya.

4. $R_t = (f_t(Q_t, Q_{t-1}, Q_{t-2}) + W_t + k_t + Q_{t-3})<<<s_t$ adalah hasil dari operasi dasar MD5 ke-t sebelum ditambah dengan Q_t yang berarti bahwa $Q_{t+1} = Q_t + R_t$.

5. $T_t = f_t(Q_t, Q_{t-1}, Q_{t-2}) + W_t + k_t + Q_{t-3}$ adalah R_t sebelum dikenakan *circular left shift* sebanyak s_t yang berarti bahwa $R_t = T_t<<<s_t$.

6. $\nabla X = X'$ XOR $X = (\pm Y)$, dimana $Y = 0, 1, 2, \ldots, 15$ adalah *modular XOR differential* dengan Y berarti bahwa bit ke-Y dari X adalah 1 sedangkan bit ke-Y dari X' adalah 0. Sebaliknya dengan $-Y$ yang berarti bahwa bit ke-Y dari X adalah 0 sedangkan bit ke-Y dari X' adalah 1.

2.5. Serangan Diferensial Terhadap MD5

Wang Xiayun menemukan *differential* dengan probabilitas kolisi yang tinggi pada dua pesan dengan panjang 2 kali panjang blok yang digunakan MD5 (1024 bit) [WAN05]. Karena panjang pesan adalah 2 blok, akan ada dua iterasi yang dilakukan:

$$\Delta H_0 \xrightarrow{(M_0, M'_0)} \Delta H_1 \xrightarrow{(M_0, M'_0)} \Delta H = 0 \ldots\ldots\ldots\ldots\ldots\ldots (3)$$

dimana:

$\Delta M_0 = (0, 0, 0, 0, 2^{31}, 0, 0, 0, 0, 0, 0, 2^{15}, 0, 0, 2^{31}, 0)$

$\Delta M_1 = (0, 0, 0, 0, 2^{31}, 0, 0, 0, 0, 0, 0, -2^{15}, 0, 0, 2^{31}, 0)$

2.6. Syarat-syarat yang Diperlukan

Untuk dapat menghasilkan kolisi, pada setiap operasi dasar MD5, akan ada syarat-syarat yang harus dipenuhi. Syarat-syarat untuk iterasi pertama dan kedua diberikan pada lampiran A dan C. Dari syarat-syarat tersebut, akan dapat dihasilkan kondisi pada bit-bit Q_t yang diberikan pada lampiran B dan D.

2.7. *Single-message Modification*

Pada 16 operasi dasar pertama (putaran 1), setelah kondisi pada Q_t telah dipenuhi, akan dilakukan *single-message modification* untuk merubah pesan secara langsung. Nilai *word* (32 bit) ke-t dapat dihitung dengan persamaan (4):

$$m \leftarrow (Qt_{+1} - Q_t)>>>s_t - F(Q_t, Q_{t-1}, Q_{t-2}) - k_t - Q_{t-3} \ldots (4)$$

Secara umum proses yang dilakukan adalah:

1. Pilih m_t secara acak (*random*)
2. Hitung Qt_{+1} dengan persamaan (5):
   $Q_{t+1} = Q_t + (f_t(Q_t, Q_{t-1}, Q_{t-2}) + W_t + k_t + Q_{t-3})<<<s_t \ldots\ldots (5)$
3. Ubah beberapa bit Qt_{+1} sehingga memenuhi kondisi yang benar

Hitung kembali m_t dan lakukan *single-message modification* dengan persamaan (4).

2.8. *Multi-message Modification*

Setelah putaran pertama, 16 *word* pada pesan telah didefinisikan. Hal ini menyebabkan *single-message modification* tidak mungkin dilakukan lagi pada putaran kedua karena akan menyebabkan ketidakkonsistenan pesan. Untuk mengatasi masalah ini, harus dicek apakah bit pada pesan yang dikenai perubahan pada ronde kedua akan menyebabkan kondisi pada putaran pertama tidak lagi dipenuhi. Jika hal itu terjadi, akan dilakukan perubahan-perubahan kembali yang berarti aka nada beberapa *word* pesan yang harus dimodifikasi pesan atau disebut juga *multi-message modification*.

2.9. Algoritma

Algoritma menemukan kolisi pada fungsi *hash* MD5 secara umum:

1. Iterasi pertama
    1. Pilih pesan acak (*random*) M_0
    2. Pada setiap operasi dasar, penuhi setiap kondisi yang berlaku dengan melakukan *message modification*. Jika ada kondisi yang tidak dapat dipenuhi, kembali ke langkah 1a.
    3. Cek apakah semua *difference* dipenuhi.
    4. Isi nilai penyangga dengan hasil iterasi pertama.
2. Iterasi kedua
    a. Pilih pesan acak (*random*) M_1
    b. Pada setiap operasi dasar, penuhi setiap kondisi yang berlaku dengan melakukan *message modification*. Jika ada kondisi yang tidak dapat dipenuhi, kembali ke langkah 2a.
    c. Cek apakah semua *difference* dipenuhi.
    d. Isi nilai M'_0 dan M'_1 sesuai dengan *difference* yang telah ditentukan dan kolisi telah ditemukan.

2.10. Modifikasi Klima

Vlastimil Klima [KLI05] menyajikan sedikit modifikasi pada urutan pencarian Q_t (bukan modifikasi pesan) yang dapat mempercepat pencarian pada iterasi pertama. Melihat fakta bahwa pada iterasi pertama Q_1 dan Q_2 tidak memiliki kondisi apapun, Klima menyatakan bahwa m_0 dan m_1 sebaiknya ditentukan dengan kondisi Q_{17} dan Q_{20}.

Langkah-langkah pencarian dengan modifikasi Klima:

1. Pilih Q_3, Q_4, ..., Q_{16} secara acak (*random*), penuhi kondisi yang diperlukan.
2. Hitung nilai m_6, m_7, ..., m_{15}.
3. Pilih Q_{17} dan hitung Q_{18} dan Q_{19} sampai kondisi ketiganya dipenuhi.
4. Hitung m_1 dari nilai Q_{17}, Q_{16}, Q_{15}, Q_{14} dan Q_{13}.
5. Pilih Q_{20}, hitung m_0, Q_1, Q_2, m_2, m_3, m_4, m_5.

Periksa apakah semua kondisi dipenuhi, jika tidak, kembali ke langkah 5.

3. ANALISIS

3.1. Inisialisasi Penyangga MD

Walaupun Wang Xiayun [WAN05] memakai nilai *default* dari inisialisasi penyangga MD (a = 0x67452301, b = 0xefcdab89, c = 0x98badcfe, d = 0x10325476), sebenarnya serangan yang dia kemukakan tetap dapat bekerja pada sembarang nilai *initialization vector* (*IV*). Hal ini dikarenakan hanya ada dua hal yang dilihat pada serangannya yaitu *add-difference* dan *modular XOR difference*. Sesuai dengan kata *difference* yang berarti perbedaan, tidak akan ada masalah pada penggantian *initialization vector* karena *initialization vector* yang digunakan pada kedua pesan adalah sama.

Sekilas tampak tidak ada yang dapat dimanfaatkan dengan mengubah *initialization vector* yang digunakan, tetapi sebenarnya ada manfaat yang sangat besar. MD5 menggunakan konstruksi Merkle Damgård yang berarti output dari hasil iterasi sebelumnya akan dipakai sebagai input pada iterasi setelahnya.

Dengan dapat digunakannya sembarang *initialization vector*, akan dapat diciptakan kolisi MD5 pada dua sembarang pesan dengan panjang sembarang yang hanya terdapat perbedaan pada 1024 bit di sembarang tempat di dalam pesan. Misal kedua pesan itu adalah:

Pesan 1: $M_0, M_1, ..., M_{i-1}, \boldsymbol{M_i}, \boldsymbol{M_{i+1}}, M_{i+2}, ..., M_n,$

Pesan 2: $M_0, M_1, ..., M_{i-1}, \boldsymbol{M'_i}, \boldsymbol{M'_{i+1}}, M_{i+2}, ..., M_n.$

Isi pesan awal ($M_0, M_1, ..., M_{i-1}$) dan akhir ($M_{i+2}, ..., M_n$) dapat ditentukan sesuka hati. Untuk menghasilkan kolisi, dipakailah hasil iterasi pada M_{i-1} sebagai *initialization vector* pada serangan yang dikemukakan Wang. Karena hasil iterasi pada $\boldsymbol{M_{i+1}}$ dan $\boldsymbol{M'_{i+1}}$ sama, hasil *message digest* akhir dari kedua pesan tentu akan sama pula (yang berarti kolisi).

3.2. Aplikasi dari Serangan Ini

Baik *file postscript* maupun *source code* program memungkinkan struktur *if* yang dapat menghasilkan keluaran yang berbeda sesuai dengan kondisi yang ditentukan. Hal ini memberikan ruang untuk melakukan kejahatan dengan serangan yang dikemukakan oleh Wang.

Karena serangan Wang terlihat hanya dapat mencari kolisi pada dua pesan dengan panjang 1024 bit, awalnya serangan ini terlihat tidak dapat diaplikasikan pada dunia nyata. Tetapi melihat fakta yang disajikan pada subbab 3.1, aplikasi sebenarnya dari serangan tersebut akan dapat dilakukan.

Sebagai gambaran mari kita lihat dua struktur *source code* program berikut ini:

source code-1 = if (data1==data1) then {aksi-1} else {aksi-2}

source code-2 = if (data2==data1) then {aksi-1} else {aksi-2}

Dapat kita simpulkan bahwa program pertama akan melakukan aksi-1 sedangkan program kedua akan melakukan aksi-2.

Untuk menghasilkan kolisi pada kedua *file*, yang harus dilakukan adalah mengisi data pada data1 dengan panjang 1024 (misalkan $M_0 + M_1$) bit kemudian mencari kolisinya (misalkan $M'_0 + M'_1$) dan mengisikan nilai tersebut pada data2. Hal terakhir yang harus dilakukan adalah membuat isi file sebelum data1 / data2 menjadi kelipatan 512 bit. Hal ini dapat dilakukan dengan member komentar dengan panjang tertentu pada awal *file*.

Kolisi fungsi *hash* MD5 tidak hanya pada *source code* program saja, tetapi juga pada *file executable*-nya. Hal ini dapat dilakukan dengan membuat source code seperti pada *source code*-1 yang dikemukakan sebelumnya, tetapi dengan mengisi data1 dengan data sembarang dengan panjang 1536 bit dan kemudian meng-*compile*-nya menjadi *file executable*. Setelah itu data sembarang yang telah dimasukkan tadi dicari letaknya pada *file executable* tersebut.

Alasan panjang 1536 bit dari data tersebut adalah untuk menjamin bahwa data tersebut dapat diubah langsung pada *file executable* tersebut tanpa mempengaruhi bagian-bagian penting lainnya. Melihat fakta bahwa kita harus menghitung terlebih dahulu nilai hash sementara dari isi awal *file executable* dengan kelipatan 512 bit, besar kemungkinan data sembarang tadi juga menjadi bagian kelipatan 512 bit. Setelah itu, dicarilah kolisi dari data 1024 bit.

4. IMPLEMENTASI DAN PENGUJIAN

4.1. Lingkungan Pengembangan

Perangkat keras yang digunakan dalam pengembangan perangkat lunak MD5 Clone adalah seperangkat komputer dengan spesifikasi sebagai berikut:

1. Prosesor Intel Centrino Duo T2300 @ 1.66 GHz
2. RAM 512 MB
3. *Harddisk* 80 GB
4. Monitor 15 inci

Sistem operasi yang menjadi lingkungan pengembangan perangkat lunak ini adalah Microsoft Windows XP Home Edition. Bahasa yang digunakan dalam implementasi perangkat lunak adalah bahasa C sedangkan kompilator yang digunakan adalah MinGW 5.1.4.

4.2. Pengujian

Telah dilakukan sekitar 10 kali pengujian dan semua pasangan pesan yang berhasil dihasilkan memiliki *message digest* yang sama (terjadi kolisi). Dari pengujian-pengujian yang berhasil tersebut, didapatkan waktu rata-rata proses sebesar 30.94 menit. Sebagai informasi, program untuk menghitung *message digest* dari pesan tersebut diperoleh dari http://www.fourmilab.ch.

Contoh beberapa kolisi yang didapatkan diperlihatkan pada pasangan pesan berikut ini (kolisi ditemukan dalam waktu 26.4 menit).

Pesan 1:

```
cbbfc80d10feb2189a41f11be97e2dc4e0
d45224c7adafc1ce66415e9651f92dfe33
6cc5deb345fe8394ab6b543a177adf8ac6
f84f1c0de6306ad9956520a7b5607bd6af
f7e75c01651b1e73f5182e2235b5f5643d
8c760cd7b3cd04afcd4e5102bae79909de
f426e3f9e54ebeb4d855bff628ab3f141a
6f02b1341e828488c4
```

Pesan 2:

```
cbbfc80d10feb2189a41f11be97e2dc460
d45224c7adafc1ce66415e9651f92dfe33
6cc5deb345fe8394ab6b543a977adf8ac6
f84f1c0de6b06ad9956520a7b5607bd6af
f7e75c01651b1e73f5182e22b5b5f5643d
8c760cd7b3cd04afcd4e5102bae79909de
f426e3f9e54ebeb45855bff628ab3f141a
6f82b1341e828488c4
```

Dengan nilai *message digest* keduanya:

```
fe8f08b1f3c01de0efe8aa8300a4f872
```

5. KESIMPULAN DAN SARAN

4.3. Kesimpulan

Berdasarkan hal-hal yang telah dipelajari pada pelaksanaan tugas akhir ini, dapat diambil kesimpulan sebagai berikut:

1. Pencarian kolisi pada fungsi *hash* MD5 dapat dilakukan secara efisien dengan algoritma yang diperkenalkan oleh Wang Xiaoyun dan dimodifikasi oleh Vlastimil Klima.

2. MD5 Clone telah berhasil membangkitkan pasangan-pasangan pesan yang memiliki *message digest* yang sama dengan waktu rata-rata 30.94 menit.

3. Karena pencarian kolisi yang diperkenalkan oleh Wang menggunakan pembangkitkan data *random*, perbedaan waktu yang diperlukan untuk mencari kolisi-kolisi sangat bervariasi. Hal ini dibuktikan dengan proses pencarian yang dapat berlangsung sangat cepat (4.2 menit) ataupun sangat lama (77.19 menit).

4.4. Saran

Adapun saran terkait dengan pelaksanaan tugas akhir ini antara lain:

1. Dibutuhkan perangkat keras dengan kemampuan prosesor sangat kuat untuk membantu proses pengembangan perangkat lunak, mengingat waktu yang diperlukan untuk mencari sebuah pasangan pesan cukup lama. Dengan adanya bantuan perangkat keras tersebut, pengembangan aplikasi dari serangan ini, yaitu pembuatan dua *file executable* yang saling berkolisi akan lebih mudah dilakukan.

2. Sebaiknya MD5 tidak lagi digunakan untuk menjamin keamanan (*security*) dari pengiriman pesan karena MD5 sudah tidak lagi aman untuk digunakan.

DAFTAR PUSTAKA

[BOE93] Boer, Bert den dan Antoon Bosselaers. (1993). *Collisions for the Compression Function of MD5*. K.U. Leuven. www.esat.kuleuven.ac.be/~cosicart/pdf/AB-9300.pdf. Tanggal akses: 24 Februari 2008.

[DOB96-a] Dobbertin, Hans. (1996). *Cryptanalysis of MD5 Compress*. UCSD Department of Computer Science and Engineering. http://www-cse.ucsd.edu/~bsy. Tanggal akses: 14 April 2008.

[DOB96-b] Dobbertin, Hans. (1996). *The Status of MD5 after a Recent Attack*. ECE 575 Data Security & Criptography. http://islab.oregonstate.edu/koc/ece575 . Tanggal akses: 24 Februari 2008.

[HAW05] Hawkes, Philip et. al. (2005). *Musing at the Wang et al. MD5 Collision*. Cpriptology ePrint Archive: Report 2004/264. http://eprint.iacr.org/2004/264. Tanggal Akses: 16 Mei 2008.

[KAM04] Kaminsky, Dan. (2004). *MD5 to Be Considered Harmful Someday*. Doxpara Research. http://www.doxpara.com. Tanggal akses: 19 Februari 2008.

[KLI05] Klima, Vlastimil. (2005). *Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications*. Cryptology ePrint Archive: Report 2005/102. http://eprint.iacr.org/2005/102. Tanggal akses: 6 Maret 2008.

[MAD01] Madhav, Buddhi. (2001). *New Way of Cryptanalizing MD5*. Department of Computer Science and Engineering, Indian Institute of Technology. http://www.cse.iitk.ac.in/research/mtech1999/9911109.html. Tanggal akses: 18 Februari 2008.

[MIK04] Mikle, Ondrej. (2004). *Practical Attack on Digital Signatures Using MD5 Message Digest*. Personal Page: Vlastimil Klima, Dr. http://cryptography.hyperlink.cz. Tanggal akses: 20 Februari 2008.

[MUN06] Munir, Rinaldi. (2006). *Diktat Kuliah IF5054 Kriptografi*. Insitut Teknologi Bandung.

[RIV92] Rivest, Ron L. (1992). *The MD5 Message-Digest Algorithm*. MIT Laboratory for Coumputer Science and RSA Data Security, Inc. http://www.ietf.org/rfc/rfc1321.txt. Tanggal akses: 20 Februari 2008.

[SAS05] Sasaki, Yu et. al. (2005). *Improved Collision Attack on MD5*. Cryptology ePrint Archive: Report 2005/400. http://eprint.iacr.org/2005/400. Tanggal akses: 24 Februari 2008.

[THO05] Thomsen, Steffen. (2005). *Criptographic Hash Functions*. Technical University of Denmark. http://www2.mat.dtu.dk/people/S.Thomsen. Tanggal akses: 27 Mei 2008.

[WAN04] Wang, Xiaoyun. (2004). *Collision for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*. Cryptology ePrint Archive: Report 2004/199. http://eprint.iacr.org/2004/199. Tanggal akses: 19 Februari 2008.

[WAN05] Wang, Xiaoyun dan Hongbo Yu. (2005). *How to Break MD5 and Other Hash Function*. SpringerLink – Book Chapter. http://www.springerlink.com/content/m4ye44cbfjfemuk0. Tanggal akses: 20 Februari 2008.

LAMPIRAN

A. Syarat-syarat pada Iterasi Pertama

t	W_t	S_t	ΔW_t	ΔQ_t	∇Q_t
-3	-	-	-	-	-
-2	-	-	-	-	-
-1	-	-	-	-	-
0	m_0	7	0	0	-
1	m_1	12	0	0	
2	m_2	17	0	0	-
3	m_3	22	0	0	-
4	m_4	7	-2^{31}	0	-
5	m_5	12	0	-2^6	(6,. . . ,21,−22)
6	m_6	17	0	$2^{31}+2^{23}-2^6$	(−6,23,31)
7	m_7	22	0	$-2^{27}+2^{23}-2^6+2^0$	(0,1,2,3,4,−5,6,7,8,9,10,−11,−23, −24,−25,26,27,28,29,30,31)
8	m_8	7	0	$-2^{23}-2^{17}-2^{15}+2^0$	(1,16,−17,18,19,20,−21,−24)
9	m_9	12	0	$-2^{31}-2^6+2^1-2^0$	(-0,1,6,7,-8,-31)
10	m_{10}	17	0	$2^{31}+2^{13}-2^{12}$	(-12,13,31)
11	m_{11}	22	2^{15}	$2^{31}+2^{30}$	(30,31)
12	m_{12}	7	0	$2^{31}-2^{13}-2^7$	(7,−8, 13, . . . , 18,−19, 31)
13	m_{13}	12	0	$2^{31}+2^{24}$	(−24, 25, 31)
14	m_{14}	17	-2^{31}	2^{31}	(31)
15	m_{15}	22	0	$2^{31}-2^{15}+2^3$	(3,−15, 31)
16	m_1	5	0	$2^{31}-2^{29}$	(-29,31)
17	m_6	9	0	2^{31}	(31)
18	m_{11}	14	2^{15}	2^{31}	(31)
19	m_0	20	0	$2^{31+}2^{17}$	(17,31)
20	m_5	5	0	2^{31}	(31)
21	m_{10}	9	0	2^{31}	(31)
22	m_{15}	14	0	2^{31}	(31)
23	m_4	20	-2^{31}	0	-
24	m_9	5	0	0	-
25	m_{14}	9	-2^{31}	0	-
26	m_3	14	0	0	-
...	...	...	…	…	…
34	m_{11}	16	2^{15}	0	-
35	m_{14}	23	-2^{31}	2^{31}	(31)
36	m_1	4	0	2^{31}	(31)
37	m_4	11	-2^{31}	2^{31}	(31)
38	m_7	16	0	-2^{31}	(-31)
39	m_{10}	23	0	-2^{31}	(-31)
40	m_{13}	4	0	-2^{31}	(-31)
41	m_0	11	0	2^{31}	(31)
42	m_3	16	0	-2^{31}	(-31)
43	m_6	23	0	-2^{31}	(-31)
44	m_9	4	0	-2^{31}	(-31)
45	m_{12}	11	0	2^{31}	(31)
46	m_{15}	16	0	2^{31}	(31)
47	m_2	23	0	-2^{31}	(-31)
48	m_0	6	0	2^{31}	(31)
49	m_7	10	0	-2^{31}	(-31)
50	m_{14}	15	-2^{31}	-2^{31}	(-31)
51	m_5	21	0	-2^{31}	(-31)
...	...	...	…	…	…
59	m_{13}	21	0	-2^{31}	(-31)
60	m_4	6	-2^{31}	-2^{31}	(31)

t	W_t	S_t	ΔW_t	ΔQ_t	∇Q_t
61	m_{11}	10	2^{15}	-2^{31}	(-31)
62	m_2	15	0	$2^{31}+2^{25}$	(25,31)
63	m_9	21	0	$-2^{31}+2^{25}$	(25,-31)

B. Kondisi Bit Q_t pada Iterasi Pertama

t	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1																																
2																																
3													0								0					0						
4	C								0	▲	▲	▲	1	▲	▲	▲	▲	▲	▲	▲	1	▲	▲	▲	▲	0						
5	C				1		0		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1			1		1
6	B	▲	▲	▲	0	▲	1	▲	0	1	1	1	1	1	1	1	1	0	1	1	1	1	0	0	0	1	0	▲	▲	0	▲	1
7	A	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0
8	0	0	0	0	0	0	0	1	1			1	0	0	0	1	0		0		0	1	0	1	0	1	0	0	0	0	0	0
9	E	1	1	1	1	0	1	1				1	0	0	0	0	0		1	▲	1	1	1	1	0	0	1	1	1	1	0	1
10	A	1							0			1	1	1	1	1	1		0	1				0	0	1					0	0
11	A	0											0	0	0	1	1	▲	0	0				0	1	1					1	0
12	A	0					▲	▲					1	0	0	0	0	0	0	1				1	0							
13	A	1					0	1					1	1	1	1	1	1	1					0	0				1			
14	A		0				0	0					1	0	1	1	1	1	1					1	1				1			
15	H		1				0	1									1												0			
16	H		1																													
17	H														0		▲												▲			
18	H		▲												1																	
19	H														0																	
20	H																															
21	H														▲																	
22	H																															
23	0																															
24	1																															
25-45																																
46	I																															
47	J																															
48	I																															
49	J																															
50	K																															
51	J																															
52	K																															
53	J																															
54	K																															
55	J																															
56	K																															
57	J																															
58	K																															
59	J																															
60	I						0																									
61	J						1																									
62	I						0																									
63	J						0																									
64																																
−3																																
−2	L						0																									
−1	L					0	1																									
0	L					0	0																				0					

1. if (A = B) then C =1 else C = 0
2. E = ~A dan I = ~K
3. ▲ = bit tersebut harus memiliki nilai yang sama dengan bit di atasnya
4. Semua bit dengan huruf yang sama harus memiliki nilai yang sama
5. Tidak ada simbol berarti tidak ada kondisi (sembarang nilai)

[THO05]

C. Syarat-syarat pada Iterasi Kedua

t	W_t	S_t	ΔW_t	ΔQ_t	∇Q_t
-3	_	-	-	2^{31}	(31)
-2	-	-	-	$2^{31}+2^{25}$	(25,31)
-1	-	-	-	$2^{31}+2^{25}$	(-25,26,31)
0	m_0	7	0	$2^{31}+2^{25}$	(25,31)
1	m_1	12	0	$-2^{31}+2^{25}$	(25,-31)
2	m_2	17	0	$-2^{31}+2^{25}+2^{5}$	(5,25,-31)
3	m_3	22	0	$-2^{31}+2^{25}+2^{16}+2^{11}+2^{5}$	(-5,-6,7,-11,12,-16,-17,-18,-19,-20, 21,-25,-26,-27,-28,-29,30,-31)
4	m_4	7	-2^{31}	$-2^{31}+2^{25}+2^{5}-2^{1}$	(1,2,3,-4,5,-25,26,-31)
5	m_5	12	0	$2^{31}+2^{9}+2^{6}+2^{0}$	(0,-6,7,8,-9,-10,-11,12,31)
6	m_6	17	0	$2^{31}-2^{20}-2^{16}$	(16,-17,20,-21,31)
7	m_7	22	0	$-2^{31}-2^{27}-2^{6}$	(7,8,9,-10,27,-28,-31)
8	m_8	7	0	$-2^{31}-2^{23}-2^{17}+2^{15}$	(-15,16,-17,23,24,25,-26,-31)
9	m_9	12	0	$-2^{31}+2^{6}+2^{0}$	(-0,1,-6,-7,-8,9,-31)
10	m_{10}	17	0	$-2^{31}+2^{12}$	(12,-31)
11	m_{11}	22	-2^{15}	-2^{31}	(-31)
12	m_{12}	7	0	$-2^{31}-2^{13}-2^{7}$	(-7,13,14,15,16,17,18-19,-31)
13	m_{13}	12	0	$2^{31}+2^{24}$	(-24,-25,-26,-27,-28,-29,30,31)
14	m_{14}	17	-2^{31}	2^{31}	(31)
15	m_{15}	22	0	$2^{31}+2^{15}+2^{3}$	(3,15,31)
16	m_1	5	0	$2^{31}-2^{29}$	(-29,31)
17	m_6	9	0	2^{31}	(31)
18	m_{11}	14	-2^{15}	2^{31}	(31)
19	m_0	20	0	$2^{31}+2^{17}$	(17,31)
20	m_5	5	0	2^{31}	(31)
21	m_{10}	9	0	2^{31}	(31)
22	m_{15}	14	0	2^{31}	(31)
23	m_4	20	-2^{31}	0	-
...	...	...	…	…	…
34	m_{11}	16	-2^{15}	0	-
35	m_{14}	23	-2^{31}	-2^{31}	(-31)
36	m_1	4	0	2^{31}	(31)
37	m_4	11	-2^{31}	2^{31}	(31)
38	m_7	16	0	-2^{31}	(-31)
39	m_{10}	23	0	-2^{31}	(-31)
40	m_{13}	4	0	-2^{31}	(-31)
41	m_0	11	0	2^{31}	(31)
42	m_3	16	0	-2^{31}	(-31)
...	...	...	…	…	…
46	m_{15}	16	0	-2^{31}	(-31)
47	m_2	23	0	2^{31}	(31)
48	m_0	6	0	-2^{31}	(-31)
49	m_7	10	0	2^{31}	(31)
...	...	...	…	…	…
59	m_{13}	21	0	2^{31}	(31)
60	m_4	6	-2^{31}	-2^{31}	(-31)
61	m_{11}	10	-2^{15}	2^{31}	(31)
62	m_2	15	0	$-2^{31}-2^{25}$	(-25,-31)
63	m_9	21	0	$2^{31}-2^{25}$	(25,-26,31)

D. Kondisi Bit Q_t pada Iterasi Kedua

t	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
−3																																
−2	A						0																									
−1	A					0	1																									
0	A					0	0																				0					
1	B				0	1	0				1										0						0					
2	B	▲	▲	▲	1	1	0				0	▲	▲	▲	▲	▲				▲	1				▲	▲	0			0	0	
3	B	0	1	1	1	1	1				0	1	1	1	1	1				0	1			1	0	1	1	▲	▲	1	1	
4	B	0	1	1	1	0	1				0	0	0	1	0	0				0	0	▲	▲	0	0	0	0	1	0	0	0	▲
5	A	1	0	0	1	0					1	0	1	1	1	1				0	1	1	1	0	0	1	0	1	0	0	0	0
6	A			0	0	1	0				1	0			1	0				0	1	1	0	0	0	1	0	1	0	1	1	0
7	B			1	0	1	1	▲	▲		0	0			0	1	▲			1	1	1	1	0	0	0						1
8	B			0	0	1	0	0	0		1	1			1	0	1						1	1	1	1				▲	0	
9	B			1	1	1	0	0	0						0	1	0			▲			0	1	1	1				0	1	
10	B					1	1	1	1					0	1	1	1			0			1	1	1	1				0	0	
11	B												▲	1	0	1	1	▲	▲	0			1	1	1	1				1	1	
12	B	▲	▲	▲	▲	▲	▲	▲					1	0	0	0	0	0	0	1					1							
13	A	0	1	1	1	1	1	1					1	1	1	1	1	1	1						0				1			
14	A	1	0	0	0	0	0	0					1	0	1	1	1	1	1						1				1			
15	C	1	1	1	1	1	0	1									0												0			
16	C		1																													
17	C														0		▲												▲			
18	C		▲												1																	
19	C														0																	
20	C																															
21	C														▲																	
22	C																															
23	0																															
24	1																															
25-45																																
46	D																															
47	E																															
48	D																															
49	E																															
50	F																															
51	E																															
52	F																															
53	E																															
54	F																															
55	E																															
56	F																															
57	E																															
58	F																															
59	E																															
60	D						0																									
61	E						1																									
62	D						1																									
63	E						1																									
64																																

1. B = ~A dan F = ~D
2. ▲ = bit tersebut harus memiliki nilai yang sama dengan bit di atasnya
3. Semua bit dengan huruf yang sama harus memiliki nilai yang sama
4. Tidak ada simbol berarti tidak ada kondisi (sembarang nilai)

[THO05]