Pembahasan Serangan Kolisi (*Collision Attack*) Dan Variasinya Pada Algoritma Hash MD5

Teguh Pamuji – NIM : 13503054

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13054@students.if.itb.ac.id

Abstraksi`

Message Digest Algorithm 5 (MD5) merupakan salah satu algoritma hash kriptografis populer. Algoritma ini diciptakan dengan nilai hash 128 bit oleh Ronald Rivest pada 1991 untuk menggantikan MD4. Pada tahun 1996 ditemukan lubang keamanan pada desainnya, dan pada 2004 beberapa masalah keamanan serius ditemukan menjadikan tingkat keamanan algoritma ini sangat dipertanyakan.

Makalah ini menjelaskan tentang salah satu masalah keamanan yang ditemukan pada MD5, yaitu serangan kolisi (*collision attack*) yang ditemukan oleh Xiaoyun Wang, didahului dengan konsep algoritma MD5 tersebut. Selain itu beberapa variasi serangan tersebut, diantaranya Fast Collision Attack dan Improved Collision Attack, juga akan dibahas pada makalah ini. Selanjutnya makalah ini akan diakhiri dengan kesimpulan tingkat keamanan MD5.

Kata kunci: MD5, hash, collision attack

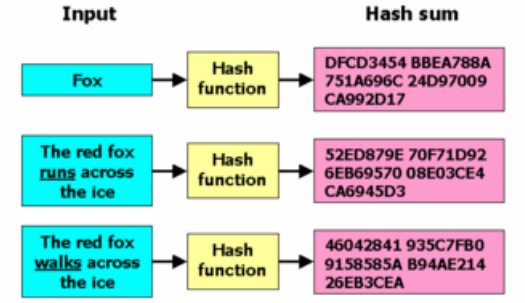
1. Pendahuluan

Sudah lama terjadi perkembangan pesat pada bidang teknologi informasi. Salah satu hasil penting dari perkembangan tersebut adalah pertukaran informasi melalui bentuk digital. Bahkan, sebagian informasi yang bersifat rahasia juga seringkali dikirimkan dalam format digital. Karena sifat rahasia itu, keamanan dalam pertukaran data menjadi salah satu hal yang sangat dipentingkan dan harus dikembangkan juga menyesuaikan dengan perkembangan teknologi informasi.

Satu teknik yang dapat digunakan untuk mengamankan pengiriman informasi digital adalah Kriptografi. Definisi kriptografi adalah ilmu sekaligus seni untuk menjaga kerahasiaan dan keamanan suatu pesan. Beberapa teknik dalam kriptografi yang umum dipakai pada aplikasi digital yaitu steganografi, *watermarking*, cipher blok (*block cipher*), kriptografi kunci publik (*public-key cryptographic* atau *asymmetric cryptosystem*), dan fungsi hash.

2. Fungsi Hash

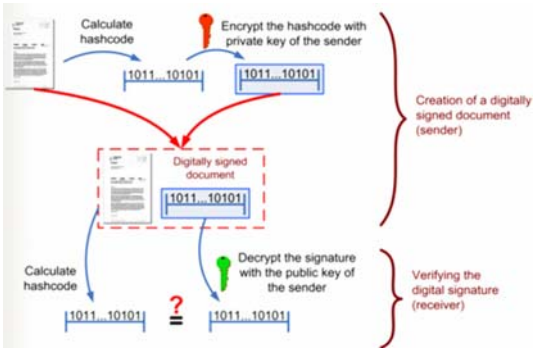
Salah satu teknik dalam kriptografi seperti yang sudah disebutkan di atas adalah fungsi hash. Fungsi hash biasanya dipakai sebagai sarana pengujian otentikasi dan integritas pesan. Secara umum fungsi hash adalah sebuah fungsi kriptografi yang menerima masukan *string* dengan panjang dan ukuran sembarang dan mengubahnya menjadi *string* keluaran yang panjangnya selalu tetap (*fixed length*). Keluaran dari fungsi hash disebut nilai hash (*hash value*) atau *message digest*. Keluaran fungsi hash pasti tidak sama untuk setiap pesan yang berbeda. Berikut adalah skema umum fungsi hash:



Fungsi hash yang dimaksud disini adalah fungsi hash satu arah (*one way hash*), yang berarti pesan yang sudah berbentuk *message digest* tidak dapat dikembalikan menjadi pesan semula. Selain itu pesan hash bersifat publik atau tidak dirahasiakan, sehingga atribut keamanan utamanya terletak pada sifat satu arahnya. Properti-properti yang harus dimiliki fungsi hash satu arah adalah:

a. *Preimage Resistant*, yaitu untuk sebuah nilai hash *h*, tidak mungkin menemukan pesan masukan *m* yang memenuhi *h* = hash (*m*).
b. *Second Preimage Resistant*, yaitu untuk sebuah nilai pesan masukan *m1*, tidak mungkin menemukan pesan masukan *m2* yang memenuhi hash (*m1*) = hash (*m2*) dimana *m1* tidak sama dengan *m2*.
c. *Collision Resistant*, yaitu tidak mungkin menemukan pesan masukan *m1* dan *m2* yang memiliki nilai hash yang sama.

Contoh penggunaan fungsi hash adalah pengiriman dan pencocokan sandi lewat (*password*) antara *client* dan *server*. Masukan sandi lewat diberikan fungsi hash sehingga menghasilkan nilai hash yang khas untuk kemudian dikirimkan kepada server dan dibandingkan dengan nilai hash yang tersimpan pada basisdata server. Sandi lewat dinyatakan benar jika nilai hashnya sesuai dengan nilai hash yang tersimpan pada server. Contoh penggunaan lainnya adalah pada aplikasi tanda tangan digital (*digital signature*). Pada aplikasi tersebut fungsi hash diterapkan untuk mencari nilai hash pesan dan selanjutnya dienkripsi menjadi tanda tangan. Pada penerima tanda tangan tersebut akan didekripsi dan dibandingkan dengan nilai hash pesan. Berikut adalah gambaran umum aplikasi tanda tangan digital dengan menggunakan fungsi hash:



Terdapat berbagai macam fungsi hash yang pernah dikembangkan dan sebagian masih dipakai hingga sekarang. Algoritma hash yang paling sering dipakai saat ini adalah SHA-1 dan MD5 meskipun sudah ditemukan beberapa lubang keamanan pada keduanya. Berikut adalah tabel perbandingan untuk beberapa algoritma hash:

Algoritma	Ukuran *message digest* (bit)	Ukuran blok pesan	Kolisi
MD2	128	128	Ya
MD4	128	512	Hampir
MD5	128	512	Ya
RIPEMD	128	512	Ya
RIPEMD-128/256	128/256	512	Tidak
RIPEMD-160/320	160/320	512	Tidak
SHA-0	160	512	Ya
SHA-1	160	512	Ada cacat
SHA-256/224	256/224	512	Tidak
SHA-512/384	512/384	1024	Tidak
WHIRLPOOL	512	512	Tidak

3. Message Digest Algorithm 5

Message Digest Algorithm 5, atau biasa disebut MD5, merupakan salah satu fungsi hash yang paling umum dan luas dipakai. Pesaing utamanya adalah algoritma SHA-1 yang dianggap memiliki kemampuan yang mirip dengan MD5. Keduanya dianggap memiliki tingkat keamanan yang tinggi dan umumnya telah digunakan pada aplikasi otentikasi dan pengujian integritas suatu arsip atau data.

Algoritma MD5 didesain oleh Ronald Rivest, yang juga ikut mendesain algoritma kriptografi RSA, pada tahun 1991. Algoritma ini diperuntukkan sebagai algoritma hash perbaikan MD4, juga dibuat olehnya. Saat itu MD4 dianggap sudah tidak layak dan tidak aman lagi dipakai karena memiliki beberapa kelemahan yang jelas ditemukan oleh Hans Dobbertin dan sudah berhasil diserang oleh kriptanalis.

Berikut adalah sebuah contoh sederhana penggunaan algoritma MD5:

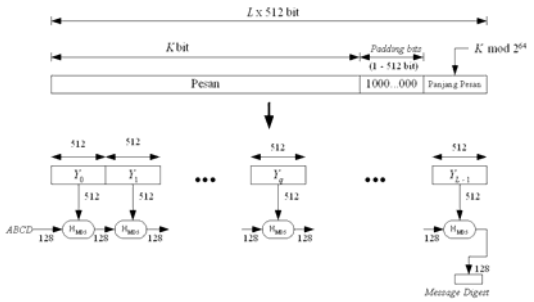
```
MD5("The quick brown fox jumps
over the lazy dog")
  = 9e107d9d372bb6826bd81d3542a419d6

MD5("The quick brown fox jumps
over the lazy cog")
  = 1055d3e698d289f2af8663725127bd4b
```

Terlihat bahkan pesan kedua yang hanya berbeda satu karakter memiliki nilai hash keluaran yang sangat berbeda. Berikut adalah contoh penerapan MD5 pada pesan kosong:

```
MD5("")
  = d41d8cd98f00b204e9800998ecf8427e
```

MD5 menerima masukan pesan dengan ukuran sembarang dan mengonversi pesan tersebut dengan algoritma hashnya menjadi message digest berukuran 128 bit, yang biasanya merupakan rangkaian 32 digit karakter heksadesimal. Lebih spesifik lagi, MD5 bekerja pada satuan blok-blok masukan berukuran 512 bit yang diproses secara berulang. Di bawah ini adalah gambaran umum pemrosesan pesan menjadi *message digest* menggunakan algoritma MD5:



Langkah-langkah pemrosesannya secara umum ada empat. Yang pertama adalah menambahkan bit-bit pengganjal pada pesan masukan agar memiliki panjang kelipatan 512 bit dikurang 64 bit. Langkah berikutnya yaitu mengisi sisa ruang 64 bit tersebut dengan informasi panjang pesan semula. Langkah ketiga adalah menginisialisasi penyangga untuk memroses pesan. Yang terakhir dilakukan yaitu memecah pesan dalam satuan blok-blok berukuran 512 bit dan mengolahnya masing-masing.

Lebih spesifik lagi, berikut adalah langkah langkah pemrosesan pesan masukan menjadi keluaran nilai hash:

a. Pesan ditambah dengan sejumlah bit pengganjal sedemikian sehingga panjang pesan kongruen dengan 448 bit modulo 512, artinya panjang pesan setelah ditambahi bit-bit pengganjal adalah 64 bit kurang dari kelipatan 512, seperti yang sudah disebutkan sebelumnya. Hal ini berlaku juga untuk pesan dengan panjang 448 bit, dimana pesan tersebut harus ditambahi bit-bit pengganjal sebanyak 512 bit menjadi 960 bit. Dari peristiwa khusus tersebut, dapat dilihat bahwa panjang bit-bit pengganjal yang diperbolehkan adalah 1 sampai 512 bit. Penambahan bit-bit pengganjal tersebut dilakukan dengan prosedur tersendiri, yaitu diawali dengan satu buah bit 1 pada akhir pesan dan selanjutnya menambahkan bit-bit 0 sampai memenuhi syarat jumlah 64 bit kurang dari kelipatan 512.
b. Pesan yang sudah diberi bit-bit pengganjal, seharusnya berukuran 64 bit kurang dari kelipatan 512, selanjutnya ditambah dengan 64 bit yang menyatakan informasi panjang pesan semula sehingga panjang pesan menjadi tepat kelipatan 512 bit. Jika panjang pesan lebih besar dari 2^{64} maka dilakukan proses modulo 2^{64} terhadap panjang tersebut, baru kemudian ditambahkan pada pesan. Dengan kata lain, jika panjang pesan semula adalah *n* bit maka 64 bit yang ditambahkan menyatakan *n* modulo 2^{64}.
c. Untuk melakukan proses *hashing*, MD5 membutuhkan empat buah penyangga (*buffer*) yang masing-masing panjangnya 32 bit sehingga totalnya 128 bit. Penyangga-penyangga tersebut berfungsi sebagai penampung hasil antara dan hasil akhir selama menjalankan algoritma hash. Keempat penyangga ini diberi nama *A*, *B*, *C*, dan *D*. Setiap penyangga diinisialisasi dengan pengisian nilai-nilai dalam notasi heksadesimal sebagai berikut:

   ***A* = 01234567**

   ***B* = 89ABCDEF**

   ***C* = FEDCBA98**

   ***D* = 76543210**

   Beberapa versi MD5 memiliki nilai inisialisasi yang berbeda, yaitu:

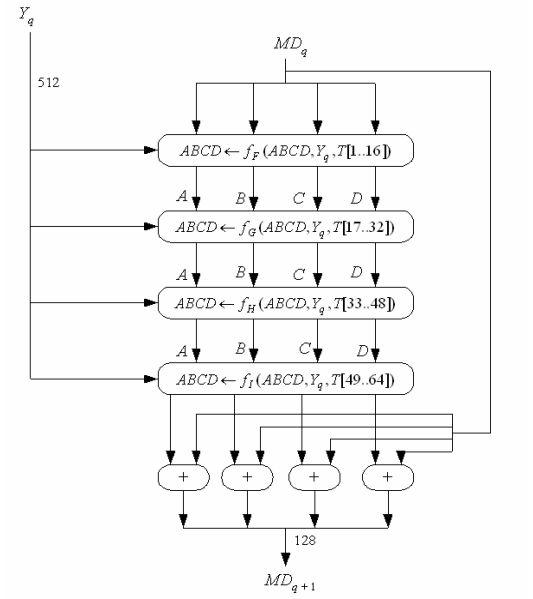
   ***A* = 67452301**

   ***B* = EFCDAB89**

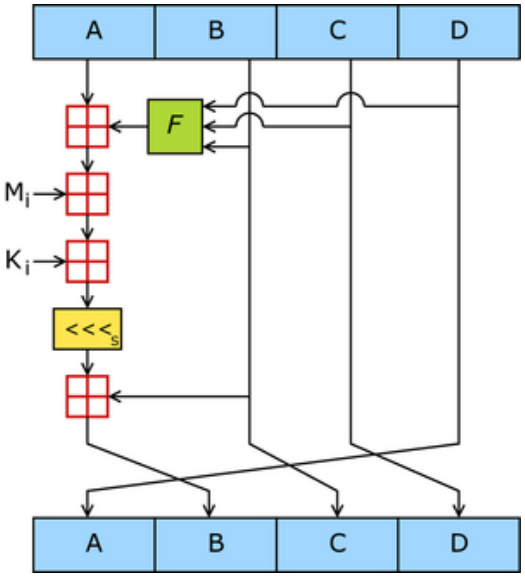
   ***C* = 98BADCFE**

   ***D* = 10325476**

d. Langkah terakhir yaitu mengolah pesan dalam blok-blok berukuran 512 bit, dimana setiap blok diproses bersama dengan penyangga *MD* menjadi keluaran 128 bit. Proses ini terdiri dari empat buah putaran dan masing-masing putaran melakukan operasi dasar MD5 sebanyak enam belas kali. Setiap operasi dasar memakai sebuah elemen *T* secara spesifik, yang akan ditampilkan nanti. Jadi setiap putaran memakai enam belas elemen tabel *T*, dan jumlah total putaran proses yang dilakukan adalah enam puluh empat buah. Proses tersebut diperlihatkan dalam gambar seperti ditampilkan di bawah ini:



Pada gambar tersebut Y_q menyatakan blok 512 bit urutan ke-q dari pesan yang sudah dipecah, sedangkan MD_q adalah nilai *message digest* 128 bit dari proses. Pada awal proses MD_q berisi nilai inisialisasi penyangga MD. Fungsi-fungsi f_F, f_G, f_H, dan f_I masing masing berisi enam belas operasi dasar terhadap masukan dengan tambahan operasi pergeseran penyangga secara sirkuler pada masing masing operasi dasar. Berikut adalah gambaran satu operasi dasar dengan melakukan pergeseran sirkuler pada penyangga:



Lambang ⊞ menyatakan penjumlahan dengan modulo 2^{32}, $\lll_s$ menyatakan *Circular Left Shift* (CLS) sebanyak s bit, dimana s bervariasi untuk setiap operasi, M_i menyatakan kelompok 32 bit ke-i dari blok 512 bit pesan ke-q dengan nilai i adalah 0 sampai 15, dan K_i adalah elemen tabel ke-i berukuran 32 bit. Sementara itu F adalah salah satu fungsi F, G, H, dan I, berfungsi untuk memanipulasi masukan A, B, C, dan D, yaitu:

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	$I(b, c, d)$	$c \oplus (b \wedge \sim d)$

Catatan: operator logika AND, OR, NOT, XOR masing-masing dilambangkan dengan $\wedge$, $\vee$, $\sim$, $\oplus$

Tabel berikut adalah nilai *T[i]* yang disusun oleh fungsi $t(i) = 2^{32}$ abs (sin i), dimana i adalah sudut dalam radian:

```
T[1]  = D76AA478     T[17] = F61E2562
T[2]  = E8C7B756     T[18] = C040B340
T[3]  = 242070DB     T[19] = 265E5A51
T[4]  = C1BDCEEE     T[20] = E9B6C7AA
T[5]  = F57C0FAF     T[21] = D62F105D
T[6]  = 4787C62A     T[22] = 02441453
T[7]  = A8304613     T[23] = D8A1E681
T[8]  = FD469501     T[24] = E7D3FBCB
T[9]  = 698098D8     T[25] = 21E1CDE6
T[10] = 8B44F7AF     T[26] = C33707D6
T[11] = FFFF5BB1     T[27] = F4D50D87
T[12] = 895CD7BE     T[28] = 455A14ED
T[13] = 6B901122     T[29] = A9E3E905
T[14] = FD987193     T[30] = FCEFA3F8
T[15] = A679438E     T[31] = 676F02D9
T[16] = 49B40821     T[32] = 8D2A4C8A
```

```
T[33] = FFFA3942     T[49] = F4292244
T[34] = 8771F681     T[50] = 432AFF97
T[35] = 69D96122     T[51] = AB9423A7
T[36] = FDE5380C     T[52] = FC93A039
T[37] = A4BEEA44     T[53] = 655B59C3
T[38] = 4BDECFA9     T[54] = 8F0CCC92
T[39] = F6BB4B60     T[55] = FFEFF47D
T[40] = BEBFBC70     T[56] = 85845DD1
T[41] = 289B7EC6     T[57] = 6FA87E4F
T[42] = EAA127FA     T[58] = FE2CE6E0
T[43] = D4EF3085     T[59] = A3014314
T[44] = 04881D05     T[60] = 4E0811A1
T[45] = D9D4D039     T[61] = F7537E82
T[46] = E6DB99E5     T[62] = BD3AF235
T[47] = 1FA27CF8     T[63] = 2AD7D2BB
T[48] = C4AC5665     T[64] = EB86D391
```

Pada bagian akhir proses *hashing* algoritma MD5 dilakukan proses penyambungan bit-bit di A, B, C, dan D untuk mendapatkan *message digest* terakhir. *Message digest* itulah keluaran akhir dari seluruh operasi yang sudah dilakukan dan itulah nilai hash MD5.

Dari semua uraian sebelumnya, secara umum algoritma hash MD5 dapat ditulis dalam persamaan matematis sebagai terlihat di bawah ini:

$$MD_0 = IV$$
$$MD_{q+1} = MD_q + f_I\,(Y_q + f_H\,(Y_q + f_G\,(Y_q + f_F\,(Y_q + MD_q))))$$
$$MD = MD_{L-1}$$

Dalam notasi matematis di atas, *IV* menyatakan initial vector dari penyangga A, B, C, dan D yang dilakukan pada proses inisialisasi penyangga, Y_q adalah blok pesan berukuran 512 bit urutan ke-*q*, *L* menyatakan jumlah blok pesan, dan *MD* menyatakan nilai akhir *message digest*.

Dengan noatsi matematis yang lebih lengkap, berikut adalah skema proses pembuatan *message digest* menggunakan MD5:

Step1: $\sum_1 = a_0 + F(b_0, c_0, d_0) + m_0 + 0xd76aa478$ $a_1 = b_0 + \sum_1 \lll 7$;
Step2: $\sum_2 = d_0 + F(a_1, b_0, c_0) + m_1 + 0xe8c7b756$ $d_1 = a_1 + \sum_2 \lll 12$;
Step3: $\sum_3 = c_0 + F(d_1, a_1, b_0) + m_2 + 0x242070db$ $c_1 = d_1 + \sum_3 \lll 17$;
Step4: $\sum_4 = b_0 + F(c_1, d_1, a_1) + m_3 + 0xc1bdceee$ $b_1 = c_1 + \sum_4 \lll 22$;
Step5: $\sum_5 = a_1 + F(b_1, c_1, d_1) + m_4 + 0xf57c0faf$ $a_2 = b_1 + \sum_5 \lll 7$;
Step6: $\sum_6 = d_1 + F(a_2, b_1, c_1) + m_5 + 0x4787c62a$ $d_2 = a_2 + \sum_6 \lll 12$;
Step7: $\sum_7 = c_1 + F(d_2, a_2, b_1) + m_6 + 0xa8304613$ $c_2 = d_2 + \sum_7 \lll 17$;
Step8: $\sum_8 = b_1 + F(c_2, d_2, a_2) + m_7 + 0xfd469501$ $b_2 = c_2 + \sum_8 \lll 22$;
Step9: $\sum_9 = a_2 + F(b_2, c_2, d_2) + m_8 + 0x698098d8$ $a_3 = b_2 + \sum_9 \lll 7$;
Step10: $\sum_{10} = d_2 + F(a_3, b_2, c_2) + m_9 + 0x8b44f7af$ $d_3 = a_3 + \sum_{10} \lll 12$;
Step11: $\sum_{11} = c_2 + F(d_3, a_3, b_2) + m_{10} + 0xffff5bb1$ $c_3 = d_3 + \sum_{11} \lll 17$;
Step12: $\sum_{12} = b_2 + F(c_3, d_3, a_3) + m_{11} + 0x895cd7be$ $b_3 = c_3 + \sum_{12} \lll 22$;
Step13: $\sum_{13} = a_3 + F(b_3, c_3, d_3) + m_{12} + 0x6b901122$ $a_4 = b_3 + \sum_{13} \lll 7$;
Step14: $\sum_{14} = d_3 + F(a_4, b_3, c_3) + m_{13} + 0xfd987193$ $d_4 = a_4 + \sum_{14} \lll 12$;
Step15: $\sum_{15} = c_3 + F(d_4, a_4, b_3) + m_{14} + 0xa679438e$ $c_4 = d_4 + \sum_{15} \lll 17$;
Step16: $\sum_{16} = b_3 + F(c_4, d_4, a_4) + m_{15} + 0x49b40821$ $b_4 = c_4 + \sum_{16} \lll 22$;
Step17: $\sum_{17} = a_4 + G(b_4, c_4, d_4) + m_1 + 0xf61e2562$ $a_5 = b_4 + \sum_{17} \lll 5$;
Step18: $\sum_{18} = d_4 + G(a_5, b_4, c_4) + m_6 + 0xc040b340$ $d_5 = a_5 + \sum_{18} \lll 9$;
Step19: $\sum_{19} = c_4 + G(d_4, a_4, b_4) + m_{11} + 0x265e5a51$ $c_5 = d_4 + \sum_{19} \lll 14$;
Step20: $\sum_{20} = b_4 + G(c_5, d_5, a_5) + m_0 + 0xe9b6c7aa$ $b_5 = c_5 + \sum_{20} \lll 20$;
…… …… …… …… …… …… ……
Step61: $\sum_{61} = a_{15} + I(b_{15}, c_{15}, d_{15}) + m_4 + 0xf7537e82$,
$a_{16} = b_{15} + \sum_{61} \lll 6$, $aa_0 = a_{16} + a_0$;
Step62: $\sum_{62} = d_{15} + I(a_{16}, b_{15}, c_{15}) + m_{11} + 0xbd3af235$,
$d_{16} = a_{16} + \sum_{62} \lll 10$, $dd_0 = d_{16} + d_0$;
Step63: $\sum_{63} = c_{15} + I(d_{16}, a_{16}, b_{15}) + m_2 + 0x2ad7d2bb$,
$c_{16} = d_{16} + \sum_{63} \lll 15$, $cc_0 = c_{16} + c_0$;
Step64: $\sum_{64} = b_{15} + I(c_{16}, d_{16}, a_{16}) + m_9 + 0xeb86d391$,
$b_{16} = c_{16} + \sum_{64} \lll 21$, $bb_0 = b_{16} + b_0$;

4. Serangan-serangan Pada MD5

Belum ada satupun algoritma kriptografis yang benar-benar sempurna. Selalu ada kemungkinan serangan berhasil yang ditujukan pada algoritma manapun. Sama seperti pada algoritma lain, hal ini juga berlaku pada algoritma MD5. Pada tahun 1996, secara teoritis dinyatakan tidak sulit menemukan dua buah set data berbeda yang memiliki nilai hash yang sama pada algoritma MD5.

Berdasarkan properti-properti fungsi hash satu arah yang sudah dijelaskan di atas, secara umum ada tiga macam serangan yang mungkin dilakukan, dengan properti *L* adalah panjang nilai hash, yaitu:

a. Serangan terhadap properti *Collision Resistant*, yaitu *Collision Attack*, yang artinya usaha menemukan dua pesan *M1* dan *M2* yang memiliki nilai hash yang sama dengan percobaan sebanyak kurang dari $2^{L/2}$.
b. Serangan terhadap properti satu arah, yaitu *First Preimage Attack*, yang berarti usaha untuk menemukan pesan masukan jika diketahui nilai hashnya dalam percobaan sejumlah kurang dari 2^L.
c. Serangan kedua terhadap properti satu arah, yaitu *Second Preimage Attack*, yang berarti usaha untuk menemukan pesan masukan M2 jika diketahui pesan M1 yang memiliki nilai hash yang sama dalam percobaan sejumlah kurang dari 2^L.

Secara praktikal, serangan pertama mungkin dilakukan dengan usaha-usaha dan manipulasi-manipulasi tertentu. Sementara itu serangan kedua dan ketiga masih dinyatakan hampir tidak mungkin untuk dilakukan dengan kondisi komputasi saat ini.

Dengan panjang nilai hash keluaran 128 bit, secara *brute force* dibutuhkan percobaan sebanyak 2^{128} kali untuk menemukan dua buah pesan atau lebih yang mempunyai nilai hash yang sama. Usaha tersebut dianggap sangat sulit dan hampir tidak mungkin dilakukan karena membutuhkan waktu yang sangat lama untuk diselesaikan.

Selain hal tersebut, muncul juga teori yang menunjukkan kelemahan MD5, yaitu jika ada dua arsip memiliki nilai hash yang sama, maka saat pada kedua arsip tersebut ditempelkan data yang serupa, nilai hash keluaran kedua arsip tersebut akan tetap sama, dengan asumsi panjang arsip dimodulo 64 adalah nol. Dalam notasi matematis adalah jika

MD5 (x) = MD5 (y) maka
MD5 (x+q) = MD5 (y+q)

Pada tahun 1993, Bert den Boer dan Antoon Bosselaers mengumumkan adanya kolisi tidak

sempurna pada MD5, disebut juga *pseudo-collision*. Kolisi tersebut menyatakan bahwa mungkin terjadi `MD5(I,X) = MD5(J,X)` dengan penggunaan dua buah vektor inisialisasi (*Initialization Vector* atau *IV*) *I* dan *J* yang memiliki perbedaan empat bit.

Selanjutnya pada tahun 1996 Hans Dobbertin memublikasikan kolisi pada MD5, tepatnya dalam fungsi kompresinya, disebut juga sebagai *free-start collision*. Usaha ini berhasil menyatakan secara teoretis bahwa MD5 memiliki lubang keamanan yang fatal dan tidak sulit untuk diserang. Serangan kolisi ini menggunakan *IV* yang berbeda dengan yang dimiliki oleh MD5 standard. Meskipun bukan merupakan sebuah serangan menyeluruh terhadap MD5, bahkan ada yang menyebutnya ”bukan benar-benar serangan”, usaha ini semakin mendekati keberhasilan untuk benar-benar menyatakan bahwa MD5 sudah tidak aman digunakan. Percobaan yang dilakukan menggunakan dua buah pesan berbeda yang berukuran 512 bit dengan menggunakan *IV* spesifik khusus sebagai berikut:

***A* = 0x12AC2375**

***B* = 0x3B341042**

***C* = 0x5F62B97C**

***D* = 0x4BA763ED**

Pada tahun 2004, tepatnya pada bulan Maret dimulailah sebuah proyek terdistribusi yang diberi kode nama MD5CRK. Proyek tersebut bertujuan untuk mendemonstrasikan bahwa MD5 secara praktikal tidak aman dengan menemukan kolisi menggunakan *birthday attack*. Namun proyek ini cepat sekali dihentikan, tepatnya pada tanggal 17 Agustus 2004, ketika kolisi sempurna pada MD5 diumumkan oleh Xiaoyun Wang, Dengguo Feng, Xuejia Lai, dan Hongbo Yu dari China. Serangan analitis mereka dinyatakan dapat dilakukan dengan membutuhkan waktu satu jam menggunakan IBM p690. Pengumuman kolisi ini dilakukan pada salah satu sesi dalam acara kriptografi Crypt 2004.

Setelah momen itu, mulailah bermunculan serangan-serangan baru berdasarkan serangan tersebut. Variasi-variasi serangan itu sukses menghasilkan serangan kolisi terhadap MD5 dengan waktu yang semakin cepat menggunakan metode-metode yang berbeda. Setelah Crypt 2004, Hawkes, Paddon, dan Rose memublikasikan paper yang menjelaskan derivasi kondisi berdasarkan paper yang dibuat Wang dan timnya.

Lebih lanjut lagi, pada tahun 2005 Wang dan timnya kembali memublikasikan penjelasan mengenai serangan kolisi temuan mereka, namun dengan lebih detil, tepatnya pada konferensi Eurocrypt 2005. Pada tanggal 1 Maret 2005 Arjen Lestra, Xiaoyun Wang, dan Benne de Weger mendemonstrasikan konstruksi dua buah sertifikat X.509 dengan kunci publik yang berbeda tetapi memiliki nilai hash MD5 yang sama. Beberapa hari kemudian Vlastimil Klima memaparkan algoritma yang telah dikembangkan dari algoritma Lenstra, Wang, dan de Weger sebelumnya, berhasil mengonstruksi kolisi MD5 hanya dalam beberapa jam saja dengan satu komputer notebook. Berikutnya giliran Stach-Liu merilis implementasi serangan yang mampu menghasilkan kolisi dalam empat puluh lima menit. Yang terbaru, pada 18 Maret 2006 Klima mengumumkan algoritma yang merupakan hasil pengembangannya, dimana algoritma tersebut mampu menemukan sebuah kolisi pada MD5 dalam satu menit saja pada satu buah komputer notebook menggunakan metode baru temuannya yang diberi nama olehnya sebagai metode *tunneling*.

Semua serangan tersebut mengarah kepada satu diskusi besar bahwa secara fungsional MD5 memiliki tingkat keamanan yang lebih rendah dibandingkan SHA-1. Detilnya, MD5 tidak bisa lagi memastikan perilaku data *executable*. Hal tersebut dapat dinotasikan sebagai berikut:

MD5 (exe1) = MD5 (exe2)
Behavior (exe1) ?= Behavior (exe2)

Notasi di atas dapat menyatakan bahwa dua data *executable* dengan perilaku yang berbeda bisa saja memiliki nilai hash MD5 yang sama. Sifat ini dapat dibuktikan dengan menggunakan sebuah kakas, Stripwire, yang akan dijelaskan lebih lanjut pemakaiannya nanti.

Selain itu MD5 tidak bisa memastikan kaserupaan informasi pada dataset. Hal tersebut dinotasikan di bawah ini:

MD5 (data1) = MD5 (data2)
Information (data1) ?= Information (data2)

Dari notasi di atas dapat dilihat bahwa dua data yang berisi informasi yang berlainan dapat memiliki nilai hash MD5 yang sama. Perilaku di

atas dapat dimodelkan menggunakan serangan P2P. Serangan ini termasuk berbahaya karena dapat mengambil informasi informasi penting semisal data jaringan (alamat *Internet Protocol*, *MAC address*), *cookies*, *caches*, sandi lewat, bahkan konfigurasi sistem dan materi kiriman seperti misalnya kode aktifasi Microsoft Windows.

5. Serangan Kolisi (Collision Attack)

Serangan kolisi adalah sebuah usaha untuk menemukan dua buah pesan masukan pesan *M1* dan *M2* yang memiliki nilai hash yang sama. Secara teoretis usaha tersebut memiliki kemungkinan berhasil yang sangat kecil tanpa menggunakan teknik khusus, karena perubahan sekecil apapun akan berpengaruh pada nilai hash keluarannya. Untuk menemukan satu pasang pesan *M1* dan *M2* yang memiliki nilai hash yang sama dibutuhkan $2^{L/2}$ percobaan, dimana *L* menyatakan panjang nilai hash. Penting juga diingat bahwa untuk melakukan usaha tersebut, setidaknya salah satu pesan sudah diketahui untuk memudahkan usaha. Akan jauh lebih sulit jika harus menemukan langsung dua buah pesan yang memiliki nilai hash yang sama.

Sudah banyak serangan kolisi yang dipublikasikan, baik yang sifatnya teoretis maupun praktikal. Dimulai dengan Serangan Kolisi Wang et al pada tahun 2004, selanjutnya banyak ahli kriptografi yang mengembangkan serangan tersebut memakai metodenya masing-masing dan menghasilkan serangan-serangan yang lebih efisien dan dalam waktu yang semakin singkat. Beberapa variasi serangan tersebut yaitu Extended Sufficient Condition Attack oleh Jun Yajima dan Takeshi Shimoyama, Improved Collision Attack oleh Yu Sasaki dan timnya, Improved Collision Attack oleh Jie Liang dan Xuejia Lai, dan Fast Collision Attack oleh Marc Stevens.

5.1. Serangan Kolisi Wang

Serangan kolisi yang dilakukan Xiaoyun Wang beserta timnya dapat menemukan lebih dari satu kolisi sesungguhnya yang berasal dari dua buah pesan berukuran 1024 bit dengan menggunakan *IV* standard milik MD5, tidak seperti usaha Hans Dobbertin yang memakai *IV* lain. Serangan kolisi milik Wang ini seringkali disebut juga *Two-block Collision Differential Path*. Usaha Wang ini diyakini jauh lebih efisien dan jelas membutuhkan waktu yang lebih singkat. Kondisi tersebut dapat digambarkan sebagai berikut di bawah:

$$IV: A_0 = 0x67452301, B_0 = 0xefcdab89, C_0 = 0x98badcfe, D_0 = 0x10325476$$

$$M' = M + \Delta C_1, \Delta C_1 = (0,0,0,0,2^{31},...,2^{15},...,2^{31},0)$$

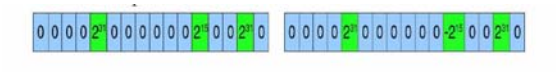
$$N_i' = N_i + \Delta C_2, \Delta C_2 = (0,0,0,0,2^{31},...,-2^{15},...,2^{31},0)$$

(non-zeros at position 4,11 and 14)

Dimana

$$MD5(M, N_i) = MD5(M', N_i').$$

Pesan 1024 bit tersebut hanya berselisih pada bit-bit tertentu saja, dan posisi bit-bit yang berbeda tersebut akan dapat digambarkan dengan lebih jelas lagi seperti terlihat pada gambar di bawah ini:



Dalam penjelasannya, Wang dikatakan dapat melakukan usaha serangan kolisi tersebut dalam waktu kira-kira satu jam dengan menggunakan IBM P690 untuk menemukan *M* dan *M'*, dimana kasus tercepat yang pernah terjadi hanya memakan waktu lima belas menit, dan setelah itu tinggal membutuhkan lima belas detik sampai lima menit untuk menemukan N_i dan N_i' sehingga pasangan (*M*, N_i) dan (*M'*, N_i') menghasilkan nilai hash keluaran yang sama persis. Lebih lanjut lagi, serangan ini dipastikan akan bekerja dengan menggunakan *IV* bagaimanapun.

Berikut ini adalah contoh dua pasang pesan berukuran 1024 bit yang berhasil mengeluarkan kolisi, dimana dua contoh tersebut memiliki sebagian awal 512 bit yang sama, ditampilkan sebagai berikut:

X_1	M	2dd31d1 c4eee6c5 69a3d69 5cf9af98 87b5ca2f ab7e4612 3e580440 897ffbb8 634ad55 2b3f409 8388e483 5a417125 e8255108 9fc9cdf7 f2bd1dd9 5b3c3780
	N_1	d11d0b96 9c7b41dc f497d8e4 d555655a c79a7335 cfdebbf0 66f12930 8fb109d1 797f2775 eb5cd530 baade822 5c15cc79 ddcb74ed 6dd3c55f d80a9bb1 e3a7cc35
X_1	M'	2dd31d1 c4eee6c5 69a3d69 5cf9af98 7b5ca2f ab7e4612 3e580440 897ffbb8 634ad55 2b3f409 8388e483 5a41f125 e8255108 9fc9cdf7 72bd1dd9 5b3c3780
	N_1'	d11d0b96 9c7b41dc f497d8e4 d555655a 479a7335 cfdebbf0 66f12930 8fb109d1 797f2775 eb5cd530 baade822 5c154c79 ddcb74ed 6dd3c55f 580a9bb1 e3a7cc35
H		9603161f f41fc7ef 9f65ffbc a30f9dbf
X_2	M	2dd31d1 c4eee6c5 69a3d69 5cf9af98 87b5ca2f ab7e4612 3e580440 897ffbb8 634ad55 2b3f409 8388e483 5a417125 e8255108 9fc9cdf7 f2bd1dd9 5b3c3780
	N_2	313e82d8 5b8f3456 d4ac6dae c619c936 b4e253dd fd03da87 6633902 a0cd48d2 42339fe9 e87e570f 70b654ce 1e0da880 bc2198c6 9383a8b6 2b65f996 702af76f
X_2	M'	2dd31d1 c4eee6c5 69a3d69 5cf9af98 7b5ca2f ab7e4612 3e580440 897ffbb8 634ad55 2b3f409 8388e483 5a41f125 e8255108 9fc9cdf7 72bd1dd9 5b3c3780
	N_2'	313e82d8 5b8f3456 d4ac6dae c619c936 34e253dd fd03da87 6633902 a0cd48d2 42339fe9 e87e570f 70b654ce 1e0d2880 bc2198c6 9383a8b6 ab65f996 702af76f
H		8d5e7019 6324c015 715d6b58 61804e08

Usaha serangan ini termotivasi dari serangan milik Hans Dobbertin, dimana Wang berusaha mencari kemungkinan untuk menemukan satu pasang pesan, masing-masing terdiri dari dua blok, yang menghasilkan kolisi setelah blok kedua. Lebih spesifik lagi, tim Wang ingin menemukan pasangan (M_0, M_1) dan (M_0', M_1') sehingga:

$$\begin{aligned}(a,b,c,d) &= \mathrm{MD5}(a_0,b_0,c_0,d_0,M_0),\\ (a',b',c',d') &= \mathrm{MD5}(a_0,b_0,c_0,d_0,M_0'),\\ \mathrm{MD5}(a,b,c,d,M_1) &= \mathrm{MD5}(a',b',c',d',M_1'),\end{aligned}$$

Dimana a_0, b_0, c_0, dan d_0 menyatakan IV untuk MD5.

Usaha serangan ini menunjukkan bahwa kolisi pada MD5 dapat ditemukan dengan efisien, dimana menemukan blok pertama (M_0, M_0') membutuhkan sekitar 2^{39} operasi MD5 dan menemukan blok kedua (M_1, M_1') memerlukan 2^{32} operasi MD5. Pernyataan kolisi ini sudah dipublikasikan pada Crypto 2004 *rump session*.

Ditambahkan juga, serangan ini dapat juga diaplikasikan pada beberapa fungsi hash lain, seperti MD4, HAVAL-128, dan RIPEMD. Bahkan khusus pada MD4, serangan ini bisa menghasilkan kolisi kurang dari satu detik dan juga bisa menemukan *second preimage* untuk banyak pesan.

Sebelum menjelaskan lebih detil mengenai serangan ini, akan diberikan dahulu penjelasan mengenai notasi-notasi yang akan digunakan. M = (m_0, m_1, ..., m_{15}) dan M' = (m_0', m_1', ..., m_{15}') merepresentasikan dua buah pesan berukuran 512 bit. ΔM = (Δm_0, Δm_1, ...,Δm_{15}) menyatakan perbedaan pada blok-blok pesan. Selanjutnya Δm_i menyatakan perbedaan urutan ke-i.

Sementara itu a_i, d_i, c_i, b_i menyatakan keluaran dari langkah kompresi M pada urutan ke-(4i - 3), ke-(4i - 2), ke-(4i - 1), dan ke-4i, dimana $1 \le i \le 16$. Selain itu, $a_{i,j}$, $d_{i,j}$, $c_{i,j}$, $b_{i,j}$ menyatakan bit ke-j dari a_i, d_i, c_i, b_i. $\varphi i,j$ adalah bit ke-j dari keluaran fungsi nonlinear pada langkah ke-i. Lebih jauh lagi, $\Delta x_{i,j} = x_{i,j}' - x_{i,j} = \pm 1$ adalah perbedaan bit yang diperoleh dari pengubahan bit ke-j dari x_i. $x_i[j]$, $x_i[-j]$, dimana x bisa merupakan a, b, c, d, atau φ, adalah nilai keluaran bit ke-j dari x_i. $x_i[j]$ didapatkan dari pengubahan bit ke-j dari x_i dari 0 ke 1 dan $x_i[-j]$ didapatkan dari pengubahan bit ke-j dari x_i dari 1 ke 0.

Pertama adalah menentukan IV standar untuk digunakan dalam proses. Karena IV tersebut sudah dituliskan sebelumnya, maka selanjutnya memilih *collision differential* dengan dua iterasi sebagai berikut:

$$\Delta H_0 \xrightarrow{(M_0,M_0')} \Delta H_1 \xrightarrow{(M_1,M_1')} \Delta H = 0$$

Dimana:

$$\begin{aligned}\Delta M_0 &= M_0' - M_0 = (0,0,0,0,2^{31},0,0,0,0,0,0,2^{15},0,0,2^{31},0)\\ \Delta M_1 &= M_1' - M_1 = (0,0,0,0,2^{31},0,0,0,0,0,0,-2^{15},0,0,2^{31},0)\\ \Delta H_1 &= (2^{31},2^{31}+2^{25},2^{31}+2^{25},2^{31}+2^{25}).\end{aligned}$$

Masukan bukan nol ΔM_0 dan ΔM_1 berada di posisi kelima, kedua belas, dan kelima belas. ΔM_0 dipilih untuk memastikan bahwa putaran diferensial 3-4 berlangsung dengan probabilitas tinggi. Sementara itu ΔM_1 tidak hanya berfungsi untuk memastikan bahwa putaran diferensial 3-4 berlangsung dengan probabilitas tinggi saja, namun juga untuk menghasilkan perbedaan keluaran yang bisa dibatalkan dengan perbedaan keluaran ΔH_1. *Collision differential* beserta karakteristiknya akan dilampirkan di bagian akhir dokumen.

Berikutnya adalah penjelasan mengenai *Sufficient Condition* pada serangan ini. *Sufficient Condition* berfungsi untuk memastikan terjadinya karakteristik diferensial pada Langkah 8 MD5 (dapat dilihat pada lampiran). Karakteristik diferensial pada Langkah 8 MD5 tersebut adalah:

$$(\Delta c_2, \Delta d_2, \Delta a_2, \Delta b_1) \longrightarrow \Delta b_2.$$

dan setiap variabel di dalamnya memenuhi satu di antara persamaan di bawah ini:

$$\begin{aligned}b_1' &= b_1\\ a_2' &= a_2[7,...,22,-23]\\ d_2' &= d_2[-7,24,32]\\ c_2' &= c_2[7,8,9,10,11,-12,-24,-25,-26,27,28,29,30,31,32,1,2,3,4,5,-6]\\ b_2' &= b_2[1,16,-17,18,19,20,-21,-24]\end{aligned}$$

Berdasarkan operasi pada langkah kedelapan, didapatkan:

$$\begin{aligned}b_2 &= c_2 + ((b_1 + F(c_2,d_2,a_2) + m_7 + t_7) \lll 22\\ b_2' &= c_2' + ((b_1 + F(c_2',d_2',a_2') + m_7' + t_7) \lll 22\\ \phi_7 &= F(c_2,d_2,a_2) = (c_2 \wedge d_2) \vee (\neg c_2 \wedge a_2)\end{aligned}$$

Derivasi yang dipakai berdasarkan fakta-fakta berikut ini:

a. Didapat $\Delta b_1 = 0$ dan $\Delta m_7 = 0$, maka diketahui bahwa:

$$\Delta b_2 = \Delta c_2^{NF} + (\Delta \phi_7 \lll 22).$$

b. Tetapkan salah satu dari variabel pada *F* sehingga *F* hanya akan menyisakan satu buah variabel.

Untuk mengetahui lebih detil mengenai *Sufficient Condition* ini, paper yang dapat dibaca adalah "How to Break MD5 and Other Hash Function", bisa diambil dan diunduh bebas dari http://www.infosec.sdu.edu.cn/paper/md5-attack.pdf.

Selanjutnya adalah mengenai modifikasi pesan (*message modification*). Ada dua jenis modifikasi pesan, yaitu *Single-message Modification* dan *Multi-message Modification*.

Untuk membuat sebuah serangan menjadi efisien, adalah sebuah hal yang penting untuk mengembangkan metode probabilistik yang sudah dijelaskan sebelumnya di atas dengan cara memperbaiki isi dari pesan masukan untuk memenuhi beberapa syarat kondisi tertentu. Dari pengamatan yang sudah dilakukan, ternyata sangat mudah untuk membangun pesan-pesan yang memenuhi semua syarat kondisi pada enam belas langkah awal MD5. Kejadian inilah yang disebut juga sebagai Modifikasi Pesan Tunggal atau *Single-message Modification*.

Lebih jauh lagi, ternyata ada kemungkinan untuk memenuhi sebagian syarat kondisi dari tiga puluh dua langkah pertama MD5 menggunakan *Multi-message Modification*. Metode ini pertama kali diperkenalkan oleh Vlastimil Klima untuk menemukan kolisi pada MD5 menggunakan PC notebook standard dengan waktu kasar sekitar delapan jam dengan kompleksitas kerja rata-rata 2^{36} operasi MD5. Dengan melakukan modifikasi beberapa pesan sekaligus, dimana modifikasi tersebut akan menghasilkan kolisi parsial pada sebagian langkah awal, maka syarat kondisi yang diperlukan akan tetap terpenuhi. *Multi-message Modification* secara umum dapat meningkatkan efisiensi kerja serangan dan mempersingkat waktu pencarian kolisi, namun bersifat lebih rumit dibandingkan menggunakan *Single-message Modification*. Kondisi lainnya bisa diperbaiki menggunakan teknik modifikasi yang sama atau teknik lain yang lebih presisi. Sama seperti sebelumnya, untuk mengetahui teknis detil mengenai modifikasi pesan, paper yang dapat dibaca adalah "How to Break MD5 and Other Hash Function", dengan file md5-attack.pdf bisa diambil dan diunduh bebas dari situs http://www.infosec.sdu.edu.cn/paper/md5-attack.pdf.

Langkah terakhir adalah melakukan serangan diferensial pada MD5. Dimulai dengan notasi persamaan sebagai berikut:

$$H_0 \overset{(M_0, M_0'), 2^{-37}}{\longrightarrow} \Delta H_1 \overset{(M_1, M_1'), 2^{-30}}{\longrightarrow} \Delta H = 0.$$

Selanjutnya mengulangi langkah-langkah ini sampai blok pertama ditemukan:

a. Memilih pesan acak M_0.
b. Memodifikasi M_0 dengan salah satu teknik modifikasi.
c. Berikutnya M_0 dan $M_0' = M_0 + \Delta M_0$ menghasilkan diferensial iterasi pertama (*first iteration differential*) berikut ini:

   $$\Delta M_0 \longrightarrow (\Delta H_1, \Delta M_1)$$

   dengan probabilitas 2^{-37}.
d. Menguji apakah semua karakteristik benar-benar terpenuhi dengan mengaplikasikan fungsi kompresi pada M_0 dan M_0'.

Hal terakhir yang harus dilakukan adalah mengikuti langkah-langkah berikut sampai kolisi ditemukan:

a. Memilih pesan acak M_1.
b. Memodifikasi M_1 dengan salah satu teknik modifikasi.
c. Berikutnya M_1 dan $M_1 + \Delta M_1$ menghasilkan diferensial iterasi kedua (*second iteration differential*) seperti berikut ini:

   $$(\Delta H_1, \Delta M_1) \longrightarrow \Delta H = 0$$

   dengan probabilitas 2^{-30}.
d. Menguji apakah pasangan pesan ini menghasilkan kolisi.

Seluruh operasi ini beserta kompleksitasnya tidak melebihi waktu kasar 2^{39} operasi MD5. dengan kata lain operasi ini merupakan operasi yang efisien dan paling mungkin dilakukan untuk menemukan kolisi. Berikut ini adalah temuan dua pasang kolisi pada MD5 yang dihasilkan menggunakan metode Wang seperti di atas:

M_0	2dd31d1 c4eee6c5 69a3d69 5cf9af98 87b5ca2f ab7e4612 3e580440 897ffbb8
	634ad55 2b3f409 8388e483 5a417125 e8255108 9fc9cdf7 f2bd1dd9 5b3c3780
M_1	d11d0b96 9c7b41dc f497d8e4 d555655a c79a7335 cfdebf0 66f12930 8fb109d1
	797f2775 eb5cd530 baade822 5c15cc79 ddcb74ed 6dd3c55f d80a9bb1 e3a7cc35
M_0'	2dd31d1 c4eee6c5 69a3d69 5cf9af98 7b5ca2f ab7e4612 3e580440 897ffbb8
	634ad55 2b3f409 8388e483 5a41f125 e8255108 9fc9cdf7 72bd1dd9 5b3c3780
M_1'	d11d0b96 9c7b41dc f497d8e4 d555655a 479a7335 cfdebf0 66f12930 8fb109d1
	797f2775 eb5cd530 baade822 5c154c79 ddcb74ed 6dd3c55f 580a9bb1 e3a7cc35
H	9603161f a30f9dbf 9f65ffbc f41fc7ef
H^*	a4c0d35c 95a63a80 5915367d cfe6b751
M_0	2dd31d1 c4eee6c5 69a3d69 5cf9af98 87b5ca2f ab7e4612 3e580440 897ffbb8
	634ad55 2b3f409 8388e483 5a417125 e8255108 9fc9cdf7 f2bd1dd9 5b3c3780
M_1	313e82d8 5b8f3456 d4ac6dae c619c936 b4e253dd fd03da87 6633902 a0cd48d2
	42339fe9 e87e570f 70b654ce 1e0da880 bc2198c6 9383a8b6 2b65f996 702af76f
M_0'	2dd31d1 c4eee6c5 69a3d69 5cf9af98 7b5ca2f ab7e4612 3e580440 897ffbb8
	634ad55 2b3f409 8388e483 5a41f125 e8255108 9fc9cdf7 72bd1dd9 5b3c3780
M_1'	313e82d8 5b8f3456 d4ac6dae c619c936 34e253dd fd03da87 6633902 a0cd48d2
	42339fe9 e87e570f 70b654ce 1e0d2880 bc2198c6 9383a8b6 ab65f996 702af76f
H	8d5e7019 61804e08 715d6b58 6324c015
H^*	79054025 255fb1a2 6e4bc422 aef54eb4

Ternyata sebuah hal yang dinyatakan sangat sulit dan hampir tidak mungkin dilakukan pada awal kemunculan MD5, yaitu menemukan dua buah pesan yang memiliki nilai hash MD5 yang sama, terbukti dapat dilakukan secara mudah dalam paper hasil kerja Xiaoyun Wang beserta timnya dari China. Apalagi secara umum metode serangan kolisi Wang ini dapat digunakan untuk menyerang fungsi hash lainnya, yaitu HAVAL-128, MD4, RIPEMD, dan SHA-0.

5.2. Jie Liang and Xuejia Lai's Improved Collision Attack

Secara umum, serangan kolisi ini adalah serangan kolisi yang dikembangkan berdasarkan serangan kolisi milik Wang. Sebagai informasi tambahan, Xuejia Lai adalah salah satu kolega Xiaoyun Wang dalam mendesain serangan kolisi pada Crypt 2004. Lai menganggap *Sufficient Conditions* yang dikemukakan Wang belum mencukupi untuk dapat menemukan kolisi secara efisien, maka dari itu ia mendesain metode pengembangannya ini.

Secara garis besar, metode ini mirip dengan Extended Sufficient Condition Attack oleh Jun Yajima dan Takeshi Shimoyama. Namun lebih dari itu , setelah mempelajari risetnya, Lai menganggap pengembangan *Sufficient Condition* milik Jun Yajima dan Takeshi Shimoyama tersebut juga masih belum mencukupi *Sufficient Condition* yang layak.

Selain hal di atas, melalui eksperimennya Jie Liang dan Xuejia Lai menemukan bahwa teknik modifikasi pesan dasar milik Wang tidak selalu berhasil dilakukan. Berdasarkan penemuan di atas, kedua ahli ini mengembangkan teknik modifikasi pesan tersendiri yang berbeda dengan teknik milik Wang, dimana teknik baru ini membuat modifikasi yang dilakukan akan selalu berhasil.

Serangan ini mirip dengan metode serangan kolisi milik Wang, namuan memiliki perbedaan bahwa metode baru ini menggunakan teknik pencarian lingkup kecil (*small range searching*) dan mengabaikan langkah komputasi pengecekan karakteristik dalam algoritma. Jika dibandingkan dengan teknik milik Wang, teknik pencarian lingkup kecil ini dapat memperbaiki empat kondisi tambahan pada diferensial iterasi pertama dan tiga kondisi tambahan pada diferensial iterasi kedua, disebut juga dengan relaksasi kondisi atau *condition relaxation*, yang pada akhirnya meningkatkan probabilitas dan memperbaiki kompleksitas penemuan kolisi. Seluruh proses serangan baru ini dinyatakan dapat diselesaikan selama lima jam menggunakan komputer personal dengan spesifikasi CPU Pentium4 1,7 GHz.

Dalam riset ini dinyatakan bahwa kondisi-kondisi dalam tabel set *Sufficient Condition* untuk *First Iteration Differential* dan dalam tabel set *Sufficient Condition* untuk *Second Iteration Differential* tidak berkecukupan untuk memastikan kemunculan jalur kolisi (*collision path*). Tabel-tabel tersebut dapat dilihat di lampiran pada akhir dokumen ini. Sebagai tambahan, beberapa kondisi dalam tabel-tabel tersebut direlaksasi untuk memperbesar set kolisinya. Pada akhirnya diberikan penggunaan teknik pencarian lingkup kecil untuk memperbaiki kondisi-kondisi pada putaran kedua tetapi tetap mempertahankan seua kondisi yang berasal dari putaran pertama. Dengan penggunaan teknik-teknik di atas, kompleksitas pencarian dapat disederhanakan sebanyak kira-kira 2^{34} operasi MD5 dalam blok pertama dan sekitar 2^{28} operasi MD5 untuk blok kedua.

Relaksasi kondisi tidak memiliki pengaruh apapun terhadap karakteristik diferensial. Setelah kondisi-kondisi ini direlaksasi, set kolisi keluaran seharusnya memiliki ukuran delapan kali lebih besar dibandingkan semula.

Selanjutnya mengenai teknik pencarian lingkup kecil, secara umum ide yang diusulkan adalah untuk `N = U + [L + F (X, Y, Z) + M + konstanta] <<< k`, nilai bit *n* pada *N* dapat diubah dengan mencari bit *n* pada *U*dan bit n-k dalam *L*, *X*, *Y*, *Z*, *M*. Selain itu dapat dicari juga bit-bit yang lebih rendah dari *n* dalam *U* dan bit-bit yang lebih rendah dari n-k dalam *L*, *X*, *Y*,

Z, *M* untuk mengubah nilai bit *n* dalam *N* secara bawaan.

Algoritma ini memiliki dua bagian proses, yaitu Fast Attack Algorithm pada blok pertama dan blok kedua. Masing-masing bagian tersebut memiliki pemilihan acak terhadap variabel-variabel isinya.

Untuk mengetahui detil teknis pemrosesannya, paper yang dapat dibaca berjudul "Improved Collision Attack on Hash Function MD5", hasil karya Jie Liang dan Xuejia Lai dari Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. Paper ini dapat diunduh secara bebas dari situs internet http://eprint.iacr.org/2004/425.

Secara perkiraan, dengan menggunakan teknik modifikasi pesan miliknya, Jie Liang dan Xuejia Lai menyatakan bahwa algoritma ini bisa menyelesaikan sebuah serangan enam belas kali lebih cepat dibandingkan serangan kolisi milik Wang et al. Dari proses Fast Attack Algorithm pada masing-masing blok, bisa diasumsikan bahwa setiap melakukan pemilihan acak untuk pencarian memerlukan rata-rata sekitar tiga puluh dua langkah MD5. Maka dapat diambil perhitungan bahwa diferensial iterasi pertama menjalankan sebanyak 2^{34} operasi MD5 dan diferensial iterasi kedua menjalankan sampai 2^{28} operasi MD5 dengan mengabaikan kondisi-kondisi tertentu yang memiliki probabilitas keberhasilan tinggi. Hasil akhirnya adalah dengan menggunakan metode ini, kompleksitas dalam menemukan kolisi dari pesan berukuran 1024 bit tidak akan melebihi sebanyak 2^{35} operasi MD5.

Dibandingkan dengan metode *Multi-message Modification*, teknik *small range searching* ini seharusnya lebih efisien jika hanya mencari sebagian bit untuk memperbaiki kondisi yang sama dalam putaran kedua. Sepertinya teknik *small range searching* ini menggabungkan kedua teknik sebelumnya untuk mempercepat serangan kolisi terhadap MD5. Sebagai tambahan, algoritma serangan ini didasarkan pada *sufficient conditions* yang sebenar-benarnya, sehingga tidak perlu lagi ada proses pengujian apakah karakteristiknya memenuhi setiap langkah seperti metode Wang dan Patrick Stach. Jadi dapat disimpulkan bahwa algoritma serangan kolisi ini memiliki kecepatan dua kali lipat dibandingkan algoritma serangan milik Vlastimil Klima dan Patrick Stach.

Dalam eksperimennya, dengan menggunakan komputer personal berspesifikasi CPU Pentium4 1,7 GHz, waktu proses yang dibutuhkan untuk mengolah blok pertama adalah sekitar empat jam dan waktu proses yang dibutuhkan untuk mengolah blok kedua adalah sekitar dua puluh menit. Pada akhirnya, sebuah contoh kolisi MD5 ditemukan selama lima jam. Contoh tersebut dibangun dengan $c_{4,32} = b_{4,32} = a_{5,32} = d_{5,32} = c_{5,32} = b_{5,32} = a_{6,32} = d_{6,32} = 1$ pada iterasi pertama, menghasilkan nilai hash keluaran sebagai berikut ini:

IV	67452301	efcdab89	98badcfe	10325476				
M_0	055a604a	a3461df0	12221694	6c449744	25c44d2c	a1b99a33	92681957	3c554e32
	0632ed41	03f3f7fc	805eb737	1300af02	befc06c7	0099e023	ff80803f	0000bd93
M_1	c0e83a00	37f3af4e	95243bff	f2e16edf	b4cc3b03	fcbaa5a3	852088e8	c00d7bd1
	fe32fffd	a7e84fe0	30803ffe	dc833c85	5f1330ed	088bde83	6f89b53d	819a57f0
M'_0	055a604a	a3461df0	12221694	6c449744	a5c44d2c	a1b99a33	92681957	3c554e32
	0632ed41	03f3f7fc	805eb737	13012f02	befc06c7	0099e023	7f80803f	0000bd93
M'_1	c0e83a00	37f3af4e	95243bff	f2e16edf	34cc3b03	fcbaa5a3	852088e8	c00d7bd1
	fe32fffd	a7e84fe0	30803ffe	dc82bc85	5f1330ed	088bde83	ef89b53d	819a57f0
H	9cd5a4f9	3b375002	8ca3c972	901209ef				

Penggunaan teknik *small range searching* ini juga bisa diterapkan kepada serangan kolisi untuk algoritma hash lainnya seperti MD4 dan RIPEMD.

5.3. Fast Collision Attack

Sama seperti serangan-serangan sebelumnya, serangan kolisi ini bertujuan untuk menemukan kolisi dua-blok (*two-block collision*) pada Fungsi Hash MD5. Serangan ini masih menggunakan jalur diferensial dan kumpulan *sufficient conditions* yang sama seperti yang dipakai oleh Wang et al. Yang baru pada serangan ini adalah teknik pengembangan yang secara deterministik memenuhi aturan untuk merotasi diferensial pada putaran pertama. Algoritma baru juga diperkenalkan untuk menemukan blok pertama, sedangkan untuk menemukan blok kedua dipergunakan algoritma milik Klima. Selain itu, untuk mengoptimasi kalang dalam (*inner loop*) maka diperlukan optimasi juga pada kumpulan *sufficient conditions*.

Mengenai waktu proses, untuk menemukan sebuah kolisi, serangan ini hanya memerlukan rata-rata satu menit dengan kompleksitas rata-rata $2^{32,3}$ pada komputer personal berspesifikasi CPU Pentium4 3,0 GHz dengan penggunaan *IV* terekomendasi acak. Jika menggunakan *IV* yang benar-benar acak, maka dengan mesin yang sama serangan dapat diselesaikan dengan waktu rata-rata lima menit dengan kompleksitas rata-rata $2^{34,1}$.

Seperti disebutkan di atas, untuk memproses blok pertama, algoritma yang digunakan adalah sebuah algoritma baru yang dikembangkan berdasarkan algoritma milik Vlastimil Klima. Sementara itu untuk mengolah blok kedua, algoritma yang digunakan adalah algoritma milik Klima sepenuhnya. Pada kenyataannya, kedua algoritma secara deterministik dapat memilih blok-blok pesan yang memenuhi kumpulan *sufficient conditions* pada putaran pertama.

Langkah-langkah algoritma pengolahan blok pertama adalah sebagai berikut:

a. Memilih Q_1, Q_2, ..., Q_{16} yang memenuhi *fulfilling condition*.
b. Mengalkulasi m_0, m_6, ..., m_{15}.
c. Lakukan sampai Q_{17}, ..., Q_{21} memenuhi *fulfilling condition*:
   i. Memilih *fulfilling condition* untuk Q_{17}.
   ii. Mengalkulasi m_1 pada $t = 16$.
   iii. Mengalkulasi Q_2 dan m_2, m_3, m_4, m_5.
   iv. Mengalkulasi Q_{18}, ..., Q_{21}.
d. Lakukan berulang pada semua kemungkinan memenuhi kondisi Q_1, Q_{10}, dimana m_{11} tidak berubah:
   i. Mengalkulasi m_8, m_9, m_{10}, m_{12}, m_{13}.
   ii. Mengalkulasi Q_{22}, ..., Q_{64}.
   iii. Memverifikasi kondisi Q_{22}, ..., Q_{64}, T_{22}, T_{34} dan kondisi IV blok berikutnya. Menghentikan proses jika semua kondisi telah terpenuhi situasi mendekati kolisi terverifikasi.
e. Mengulangi lagi langkah satu.

Untuk mengolah blok kedua langkah-langkah yang digunakan adalah langkah-langkah pemrosesan Klima, yang secara umum sebagai berikut:

a. Memilih Q_2, ..., Q_{16} yang memenuhi *fulfilling condition*.
b. Mengalkulasi m_5, ..., m_{15}.
c. Lakukan sampai Q_{17}, ..., Q_{21} memenuhi *fulfilling condition*:
   i. Memilih *fulfilling condition* untuk Q_1.
   ii. Mengalkulasi m_0, ..., m_4.
   iii. Mengalkulasi Q_2 dan m_2, m_3, m_4, m_5.
   iv. Mengalkulasi Q_{17}, ..., Q_{21}.
d. Lakukan berulang pada semua kemungkinan memenuhi kondisi Q_1, Q_{10}, dimana m_{11} tidak berubah:
   v. Mengalkulasi m_8, m_9, m_{10}, m_{12}, m_{13}.
   vi. Mengalkulasi Q_{22}, ..., Q_{64}.
   vii. Memverifikasi kondisi Q_{22}, ..., Q_{64}, T_{22}, T_{34} dan kondisi IV blok berikutnya. Menghentikan proses jika semua kondisi telah terpenuhi situasi mendekati kolisi terverifikasi.
e. Mengulangi lagi langkah satu.

Untuk mengetahui lebih detil, paper yang dapat dibaca adalah ”Fast Collision Attack on MD5” hasil riset Marc Stevens, dapat diunduh bebas di *www.win.tue.nl/hashclash/fastcoll.pdf*

Di bawah ini adlah tabel hasil pewaktuan dari percobaan-percobaan yang sudah dilakukan menggunakan serangan ini:

	Avg. time	Max. time	Avg. complexity	Nr. of tests
MD5 IV	64	428	$2^{32.25}$	1048
recommended IV's	67	660	$2^{32.33}$	1138
random IV's	291	15787	$2^{34.08}$	879

Note: Avg. complexity is the average number of MD5 block compressions.

Dengan penggunaan metode ini, terlihat bahwa IV serangan memiliki pengaruh yang signifikan terhadap kompleksitas rata-rata dalam menemukan kolisi pada MD5. Menggunakan dua buah kondisi IV mampu menghasilkan waktu proses rata-rata 67 detik lebih singkat.

6. Diskusi

Serangan-serangan kolisi di atas hanyalah sebagian dari serangan-serangan kolisi yang dipublikasikan. Semua serangan tersebut didasarkan pada serangan kolisi milik Wang et al yang dipublikasikan pada Crypt 2004. Ternyata serangan kolisi yang ditemukan Wang beserta timnya tersebut mampu memberikan pengaruh yang signifikan terhadap perkembangan kriptoanalisis terhadap MD5. Setelah event Crypt 2004 itu, mulailah bermunculan metode-metode pengembangan untuk menemukan serangan kolisi pada MD5 oleh kriptanalis-kriptanalis dari seluruh dunia.

Langkah-langkah serangan biasanya merupakan hasil modifikasi algoritma atau hasil kombinasi algoritma-algoritma sebelumnya. Kesemuanya itu mampu menghasilkan serangan yang semakin efisien, waktu yang semakin singkat, dan tingkat kompleksitas yang semakin mengecil. Memang wajar jika harus ada biaya yang dikeluarkan, yaitu kerumitan menyusun algoritma yang tepat untuk dipakai mengolah pesan.

Ada beberapa properti penting yang tampak penting dibahas dalam algoritma-algoritma serangan di atas. Properti pertama adalah *Sufficient Conditions* yang pertama kali diperkenalkan oleh Wang et al. *Sufficient Conditions* memiliki definisi syarat-syarat

kondisi yang harus dipenuhi dalam mengolah pesan. Tabel *Sufficient Conditions* milik Wang ini akan dilampirkan di akhir dokumen ini. Setelah diajukan oleh Wang et al ternyata masih ada ketidakpuasan mengenai properti ini, yang dinyatakan dalam beberapa paper tidak cukup *sufficient*.

Properti berikutnya yaitu waktu dan kompleksitas. Semua kriptanalis berlomba-lomba untuk mendapatkan waktu proses dan tingkat kompleksitas terbaik di antara yang lain. Untuk mencapai hal tersebut, usaha-usaha yang dilakukan yaitu menciptakan algoritma-algoritma baru yang semakin efisien. Selain itu usaha lain yang dilakukan adalah menciptakan syarat-syarat dan aturan-aturan khusus untuk memenuhi kondisi tertentu.

Properti terakhir adalah munculnya metode-metode dan algoritma-algoritma baru dalam mengolah kedua blok pesan masukan. Contoh metode baru tersebut adalah *small range searching* yang diciptakan Jie Liang dan Xuejia Lai. Hasil penciptaan metode-metode baru ini adalah peningkatan efisiensi proses yang akhirnya berujung pada waktu proses yang semakin singkat dan kompleksitas yang semakin baik.

7. Kesimpulan

Dari bahasan di atas dapat ditarik kesimpulan singkat bahwa MD5 sudah tidak cukup aman jika dibandingkan dengan pesaingnya, SHA-1. Sudah banyak metode-metode serangan yang muncul dan dipublikasikan, dan serangan-serangan baru akan terus bermunculan dengan sifat yang semakin efisien, wktu proses yang semakin singkat, dan kompleksitas yang semakin baik. Sudah saatnya sebuah algoritma baru diajukan sebagai pengganti MD5.

DAFTAR PUSTAKA

Buku Acuan

Munir, Rinaldi. 2004. *Bahan Kuliah IF5054 Kriptografi*. Departemen Teknik Informatika, Institut Teknologi Bandung.

Paper Acuan

Dobbertin, Hans. 1996. *The Status of MD5 After A Recent Attack*. http://www.rsasecurity.com/rsalabs/node.asp?id=2149

Highland, Trevor. A *Study of the MD5 Attacks: Insights and Improvements*. http://www.cs.colorado.edu/~jrblack/papers.html

Kaminsky, Dan. *MD5 to be Considered Harmful Someday*. http://www.ccc.de/congress/2004/fahrplan/files/298-md5-considered-harmful-slides.ppt

Kaminsky, Dan. 2004. *MD5 to be Considered Harmful Someday*. http://www.doxpara.com/md5_someday.pdf

Klima, Vlastimil. 2005. *Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications*. http://eprint.iacr.org/2004/102/

Klima, Vlastimil. 2005. *Finding MD5 Collisions – a Toy for a Notebook*. http://eprint.iacr.org/2004/075/

Klima, Vlastimil. 2004. *Hash functions, MD5 and Chinese attack*. http://cryptography.hyperlink.cz/2004/Hasovaci_funkce_a_cinsky_utok_MFFUK_2004.pdf

Liang, Jie, dan Xuejia Lai. *Improved Collision Attack on Hash Function MD5*. http://eprint.iacr.org/2004/425/

Mikle, Ondrej. 2004. *Practical Attacks on Digital Signatures Using MD5 Message Digest*. http://eprint.iacr.org/2004/365/

Rivest, Ronald. 1992. *The MD5 Message Digest Algorithm*. http://theory.lcs.mit.edu/~rivest/Rivest-MD5.txt

Sasaki, Yu, et al. *Improved Collision Attack on MD5*. http://eprint.iacr.org/2004/400/

Stevens, Marc. *Fast Collision Attack on MD5*. www.win.tue.nl/hashclash/fastcoll.pdf

Wang, Xiaoyun, et al. 2004. *Collisions for Hash Function*. http://eprint.iacr.org/2004/199/

Wang, Xiaoyun, dan Hongbo Yu. 2005. *How to Break MD5 and Other Hash Function*. http://infosec.sdu.edu.cn/paper/md5-attack.pdf

Yajima, Jun, dan Takeshi Shimoyama. *Wang's Sufficient Condition of MD5 are not Sufficient*. *http://eprint.iacr.org/2004/263/*

Situs Acuan

community.roxen.com/developers/idocs/rfc/rfc4270.html
Tanggal akses: 21 Desember 2006 pukul 13.00.

http://www.securiteam.com/securityreviews/6N00C0KC0Q.html
Tanggal akses: 21 Desember 2006 pukul 13.00.

http://en.wikipedia.org/wiki/Cryptographic_hash_function
Tanggal akses: 26 Desember 2006 pukul 15.00.

http:// en.wikipedia.org/wiki/MD5
Tanggal akses: 21 Desember 2006 pukul 13.00.

Lampiran A: Karakteristik Diferensial pada First Iteration Differential

Step	The output in i-th step for M_0	w_i	s_i	Δw_i	The output difference in i-th step	The output in i-th step for M_0'
4	b_1	m_3	22			
5	a_2	m_4	7	2^{31}	-2^6	$a_2[7,\ldots,22,-23]$
6	d_2	m_5	12		$-2^6+2^{23}+2^{31}$	$d_2[-7,24,32]$
7	c_2	m_6	17		$-1-2^6+2^{23}-2^{27}$	$c_2[7,8,9,10,11,-12,-24,-25,-26,27,28,29,30,31,32,1,2,3,4,5,-6]$
8	b_2	m_7	22		$1-2^{15}-2^{17}-2^{23}$	$b_2[1,16,-17,18,19,20,-21,-24]$,
9	a_3	m_8	7		$1-2^6+2^{31}$	$a_3[-1,2,7,8,-9,-32]$
10	d_3	m_9	12		$2^{12}+2^{31}$	$d_3[-13,14,32]$
11	c_3	m_{10}	17		$2^{30}+2^{31}$	$c_3[31,32]$
12	b_3	m_{11}	22	2^{15}	$-2^7-2^{13}+2^{31}$	$b_3[8,-9,14,\ldots,19,-20,32]$
13	a_4	m_{12}	7		$2^{24}+2^{31}$	$a_4[-25,26,32]$
14	d_4	m_{13}	12		2^{31}	$d_4[32]$
15	c_4	m_{14}	17	2^{31}	$2^3-2^{15}+2^{31}$	$c_4[4,-16,32]$
16	b_4	m_{15}	22		$-2^{29}+2^{31}$	$b_4[-30,32]$
17	a_5	m_1	5		2^{31}	$a_5[32]$
18	d_5	m_6	9		2^{31}	$d_5[32]$
19	c_5	m_{11}	14	2^{15}	$2^{17}+2^{31}$	$c_5[18,32]$
20	b_5	m_0	20		2^{31}	$b_5[32]$
21	a_6	m_5	5		2^{31}	$a_6[32]$
22	d_6	m_{10}	9		2^{31}	$d_6[32]$
23	c_6	m_{15}	14			c_6
24	b_6	m_4	20	2^{31}		b_6
25	a_7	m_9	5			a_7
26	d_7	m_{14}	9	2^{31}		d_7
27	c_7	m_3	14			c_7
...	...	...	...	...	...	...
34	d_9	m_8	11			d_9
35	c_9	m_{11}	16	2^{15}	2^{31}	$c_9[+32]$
36	b_9	m_{14}	23	2^{31}	2^{31}	$b_9[+32]$
37	a_{10}	m_1	4		2^{31}	$a_{10}[+32]$
38	d_{10}	m_4	11	2^{31}	2^{31}	$d_{10}[+32]$
39	c_{10}	m_7	16		2^{31}	$c_{10}[+32]$
...	...	...	...	...	...	...
45	a_{12}	m_9	4		2^{31}	$a_{12}[+32]$
46	d_{12}	m_{12}	11		2^{31}	$d_{12}[32]$
47	c_{12}	m_{15}	16		2^{31}	$c_{12}[32]$
48	b_{12}	m_2	23		2^{31}	$b_{12}[32]$
49	a_{13}	m_0	6		2^{31}	$a_{13}[32]$
50	d_{13}	m_7	10		2^{31}	$d_{13}[-32]$
51	c_{13}	m_{14}	15	2^{31}	2^{31}	$c_{13}[32]$
52	b_{13}	m_5	21		2^{31}	$b_{13}[-32]$
...	...	...	...	...	...	...
58	d_{15}	m_{15}	10		2^{31}	$d_{15}[-32]$
59	c_{15}	m_6	15		2^{31}	$c_{15}[32]$
60	b_{15}	m_{13}	21		2^{31}	$b_{15}[32]$
61	$aa_0=a_{16}+a_0$	m_4	6	2^{31}	2^{31}	$aa_0'=aa_0[32]$
62	$dd_0=d_{16}+d_0$	m_{11}	10	2^{15}	2^{31}	$dd_0'=dd_0[26,32]$
63	$cc_0=c_{16}+c_0$	m_2	15		2^{31}	$cc_0'=cc_0[-26,27,32]$
64	$bb_0=b_{16}+b_0$	m_9	21		2^{31}	$bb_0'=bb_0[26,-32]$

Lampiran B: Sufficient Conditions untuk First Iteration Differential

c_1	$c_{1,7}=0,\ c_{1,12}=0,\ c_{1,20}=0$
b_1	$b_{1,7}=0,\ b_{1,8}=c_{1,8},\ b_{1,9}=c_{1,9},\ b_{1,10}=c_{1,10},\ b_{1,11}=c_{1,11},\ b_{1,12}=1,\ b_{1,13}=c_{1,13},$ $b_{1,14}=c_{1,14},\ b_{1,15}=c_{1,15},\ b_{1,16}=c_{1,16},\ b_{1,17}=c_{1,17},\ b_{1,18}=c_{1,18},b_{1,19}=c_{1,19},$ $b_{1,20}=1,\ b_{1,21}=c_{1,21},\ b_{1,22}=c_{1,22},\ b_{1,23}=c_{1,23},\ b_{1,24}=0,\ b_{1,32}=1$
a_2	$a_{2,1}=1,\ a_{2,3}=1,\ a_{2,6}=1,\ a_{2,7}=0,\ a_{2,8}=0,\ a_{2,9}=0,\ a_{2,10}=0,\ a_{2,11}=0,$ $a_{2,12}=0,\ a_{2,13}=0,\ a_{2,14}=0,\ a_{2,15}=0,\ a_{2,16}=0,\ a_{2,17}=0,\ a_{2,18}=0,\ a_{2,19}=0,$ $a_{2,20}=0,\ a_{2,21}=0,\ a_{2,22}=0,\ a_{2,23}=1,\ a_{2,24}=0,\ a_{2,26}=0,\ a_{2,28}=1,\ a_{2,32}=1$
d_2	$d_{2,1}=1,\ d_{2,2}=a_{2,2},\ d_{2,3}=0,\ d_{2,4}=a_{2,4},\ d_{2,5}=a_{2,5},\ d_{2,6}=0,\ d_{2,7}=1,\ d_{2,8}=0,$ $d_{2,9}=0,\ d_{2,10}=0,\ d_{2,11}=1,\ d_{2,12}=1,\ d_{2,13}=1,\ d_{2,14}=1,\ d_{2,15}=0,\ d_{2,16}=1,$ $d_{2,17}=1,\ d_{2,18}=1,\ d_{2,19}=1,\ d_{2,20}=1,\ d_{2,21}=1,\ d_{2,22}=1,\ d_{2,23}=1,\ d_{2,24}=0,$ $d_{2,25}=a_{2,25},\ d_{2,26}=1,\ d_{2,27}=a_{2,27},\ d_{2,28}=0,\ d_{2,29}=a_{2,29},\ d_{2,30}=a_{2,30},$ $d_{2,31}=a_{2,31},\ d_{2,32}=0$
c_2	$c_{2,1}=0,\ c_{2,2}=0,\ c_{2,3}=0,\ c_{2,4}=0,\ c_{2,5}=0,\ c_{2,6}=1,\ c_{2,7}=0,\ c_{2,8}=0,\ c_{2,9}=0,$ $c_{2,10}=0,\ c_{2,11}=0,\ c_{2,12}=1,\ c_{2,13}=1,\ c_{2,14}=1,\ c_{2,15}=1,\ c_{2,16}=1,\ c_{2,17}=0,$ $c_{2,18}=1,\ c_{2,19}=1,\ c_{2,20}=1,\ c_{2,21}=1,\ c_{2,22}=1,\ c_{2,23}=1,\ c_{2,24}=1,\ c_{2,25}=1,$ $c_{2,26}=1,\ c_{2,27}=0,\ c_{2,28}=0,\ c_{2,29}=0,\ c_{2,30}=0,\ c_{2,31}=0,\ c_{2,32}=0$
b_2	$b_{2,1}=0,\ b_{2,2}=0,\ b_{2,3}=0,\ b_{2,4}=0,\ b_{2,5}=0,\ b_{2,6}=0,\ b_{2,7}=1,\ b_{2,8}=0,\ b_{2,9}=1,$ $b_{2,10}=0,\ b_{2,11}=1,\ b_{2,12}=0,\ b_{2,14}=0,\ b_{2,16}=0,\ b_{2,17}=1,\ b_{2,18}=0,\ b_{2,19}=0,$ $b_{2,20}=0,\ b_{2,21}=1,\ b_{2,24}=1,\ b_{2,25}=1,\ b_{2,26}=0,\ b_{2,27}=0,\ b_{2,28}=0,\ b_{2,29}=0,$ $b_{2,30}=0,\ b_{2,31}=0,\ b_{2,32}=0$
a_3	$a_{3,1}=1,\ a_{3,2}=0,\ a_{3,3}=1,\ a_{3,4}=1,\ a_{3,5}=1,\ a_{3,6}=1,\ a_{3,7}=0,\ a_{3,8}=0,\ a_{3,9}=1,$ $a_{3,10}=1,\ a_{3,11}=1,\ a_{3,12}=1,\ a_{3,13}=b_{2,13},\ a_{3,14}=1,\ a_{3,16}=0,\ a_{3,17}=0,\ a_{3,18}=0,$ $a_{3,19}=0,\ a_{3,20}=0,\ a_{3,21}=1,\ a_{3,25}=1,\ a_{3,26}=1,\ a_{3,27}=0,\ a_{3,28}=1,\ a_{3,29}=1,$ $a_{3,30}=1,\ a_{3,31}=1,\ a_{3,32}=1$
d_3	$d_{3,1}=0,\ d_{3,2}=0,\ d_{3,7}=1,\ d_{3,8}=0,\ d_{3,9}=0,\ d_{3,13}=1,\ d_{3,14}=0,\ d_{3,16}=1,$ $d_{3,17}=1,\ d_{3,18}=1,\ d_{3,19}=1,\ d_{3,20}=1,\ d_{3,21}=1,\ d_{3,24}=0,\ d_{3,31}=1,\ d_{3,32}=0$
c_3	$c_{3,1}=0,\ c_{3,2}=1,\ c_{3,7}=1,\ c_{3,8}=1,\ c_{3,9}=0,\ c_{3,13}=0,\ c_{3,14}=0,\ c_{3,15}=d_{3,15},$ $c_{3,17}=1,\ c_{3,18}=0,\ c_{3,19}=0,\ c_{3,20}=0,\ c_{3,16}=1,\ c_{3,31}=0,\ c_{3,32}=0$
b_3	$b_{3,8}=0,\ b_{3,9}=1,\ b_{3,13}=1,\ b_{3,14}=0,\ b_{3,15}=0,\ b_{3,16}=0,\ b_{3,17}=0,\ b_{3,18}=0,$ $b_{3,20}=1,\ b_{3,25}=c_{3,25},\ b_{3,26}=c_{3,26},\ b_{3,19}=0,\ b_{3,31}=0,\ b_{3,32}=0$
a_4	$a_{4,4}=1,\ a_{4,8}=0,\ a_{4,9}=0,\ a_{4,14}=1,\ a_{4,15}=1,\ a_{4,16}=1,\ a_{4,17}=1,\ a_{4,18}=1,$ $a_{4,20}=1,\ a_{4,25}=1,\ a_{4,26}=0,\ a_{4,31}=1,\ a_{4,19}=1,\ a_{4,32}=0$
d_4	$d_{4,4}=1,\ d_{4,8}=1,\ d_{4,9}=1,\ d_{4,14}=1,\ d_{4,15}=1,\ d_{4,16}=1,\ d_{4,17}=1,\ d_{4,18}=1,$ $d_{4,19}=0,\ d_{4,20}=1,\ d_{4,25}=0,\ d_{4,26}=0,\ d_{4,30}=0,\ d_{4,32}=0$
c_4	$c_{4,4}=0,\ c_{4,16}=1,\ c_{4,25}=1,\ c_{4,26}=0,\ c_{4,30}=1,\ c_{4,32}=0$
b_4	$b_{4,30}=1,\ b_{4,32}=0$
a_5	$a_{5,4}=b_{4,4},\ a_{5,16}=b_{4,16},\ a_{5,18}=0,\ a_{5,32}=0$
d_5	$d_{5,18}=1,\ d_{5,30}=a_{5,30},\ d_{5,32}=0$
c_5	$c_{5,18}=0,\ c_{5,32}=0$
b_5	$b_{5,32}=0$
a_6-b_6	$a_{6,18}=b_{5,18},\ a_{6,32}=0,\ d_{6,32}=0,\ c_{6,32}=0,\ b_{6,32}=c_{6,32}+1$
$c_9,\ b_{12}$	$c_{24,32}=0,\ b_{12,32}=d_{12,32}$
$a_{13}-b_{13}$	$a_{13,32}=c_{12,32},\ d_{13,32}=b_{12,32}+1,\ c_{13,32}=a_{13,32},\ b_{13,32}=d_{13,32}$
$a_{14}-b_{14}$	$a_{14,32}=c_{13,32},\ d_{14,32}=b_{13,32},\ c_{14,32}=a_{14,32},\ b_{14,32}=d_{14,32}$
a_{15}	$a_{15,32}=c_{14,32}$
d_{15}	$d_{15,32}=b_{14,32}$
c_{15}	$c_{15,32}=a_{15,32}$
b_{15}	$b_{15,26}=0,\ b_{15,32}=d_{15,32}+1$
$aa_0=a_{16}+a_0$	$a_{16,26}=1,\ a_{16,27}=0,\ a_{16,32}=c_{15,32}$
$dd_0=d_{16}+d_0$	$dd_{0,26}=0,\ d_{16,32}=b_{15,32}$
$cc_0=c_{16}+c_0$	$cc_{0,26}=1,\ cc_{0,27}=0,\ cc_{0,32}=dd_{0,32},\ c_{16,32}=d_{16,32}$
$bb_0=b_{16}+b_0$	$bb_{0,26}=0, bb_{0,27}=0, bb_{0,6}=0, bb_{0,32}=cc_{0,32}$

Lampiran C: Karakteristik Diferensial pada Second Iteration Differential

Step	The output in i-th step for M_1	w_i	s_i	Δw_i	The output Difference in i-th step	The output in i-th step for M_1'
IV	aa_0, dd_0 cc_0, bb_0					$aa_0[32]$, $dd_0[26,32]$ $cc_0[-26,27,32]$, $bb_0[26,-32]$
1	a_1	m_0	7		$2^{25}+2^{31}$	$a_1[26,-32]$
2	d_1	m_1	12		$2^5+2^{25}+2^{31}$	$d_1[6,26,-32]$
3	c_1	m_2	17		$2^5+2^{11}+2^{16}$ $+2^{25}+2^{31}$	$c_1[-6,-7,8,-12,13,$ $-17,\ldots,-21,22,-26,\ldots,-30,31,-32]$
4	b_1	m_3	22		$-2+2^5+2^{25}+2^{31}$	$b_1[2,3,4,-5,6,-26,27,-32]$
5	a_2	m_4	7	2^{31}	$1+2^6+2^8+2^9+2^{31}$	$a_2[1,-7,8,9,-10,-11,-12,13,32]$
6	d_2	m_5	12		$-2^{16}-2^{20}+2^{31}$	$d_2[17,-18,21,-22,32]$
7	c_2	m_6	17		$-2^6-2^{27}+2^{31}$	$c_2[7,8,9,-10,28,-29,-32]$
8	b_2	m_7	22		$2^{15}-2^{17}-2^{23}+2^{31}$	$b_2[-16,17,-18,24,25,26,-27,-32]$
9	a_3	m_8	7		$1+2^6+2^{31}$	$a_3[-1,2,-7,-8,-9,10,-32]$
10	d_3	m_9	12		$2^{12}+2^{31}$	$d_3[13,-32]$
11	c_3	m_{10}	17		2^{31}	$c_3[-32]$
12	b_3	m_{11}	22	-2^{15}	$-2^7-2^{13}+2^{31}$	$b_3[-8,14,15,16,17,18,19,-20,-32]$
13	a_4	m_{12}	7		$2^{24}+2^{31}$	$a_4[-25,\ldots,-30,31,32]$
14	d_4	m_{13}	12		2^{31}	$d_4[32]$
15	c_4	m_{14}	17	2^{31}	$2^3+2^{15}+2^{31}$	$c_4[4,16,32]$
16	b_4	m_{15}	22		$-2^{29}+2^{31}$	$b_4[-30,32]$
17	a_5	m_1	5		2^{31}	$a_5[32]$
18	d_5	m_6	9		2^{31}	$d_5[32]$
19	c_5	m_{11}	14	-2^{15}	$2^{17}+2^{31}$	$c_5[18,32]$
20	b_5	m_0	20		2^{31}	$b_5[32]$
21	a_6	m_5	5		2^{31}	$a_6[32]$
22	d_6	m_{10}	9		2^{31}	$d_6[32]$
23	c_6	m_{15}	14			$c_6[32]$
24	b_6	m_4	20	2^{31}		$b_6[32]$
25	a_7	m_9	5			a_7
26	d_7	m_{14}	9	2^{31}		d_7
27	c_7	m_3	14			c_7
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
34	d_9	m_8	11			d_9
35	c_9	m_{11}	16	-2^{15}	2^{31}	$c_9[*32]$
36	b_9	m_{14}	23	2^{31}	2^{31}	$d_9[*32]$
37	a_{10}	m_1	4		2^{31}	$a_{10}[*32]$
38	d_{10}	m_4	11	2^{31}	2^{31}	$d_{10}[*32]$
39	c_{10}	m_7	16		2^{31}	$c_{10}[*32]$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
49	a_{13}	m_0	6		2^{31}	$a_{13}[32]$
50	d_{13}	m_7	10		2^{31}	$d_{13}[-32]$
51	c_{13}	m_{14}	15	2^{31}	2^{31}	$c_{13}[32]$
52	b_{13}	m_5	21		2^{31}	$b_{13}[-32]$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
59	c_{15}	m_6	15		2^{31}	$c_{15}[32]$
60	b_{15}	m_{13}	21		2^{31}	$b_{15}[32]$
61	$a_{16}+aa_0$	m_4	6	2^{31}		$a_{16}+aa_0=a'_{16}+aa'_0$
62	$d_{16}+dd_0$	m_{11}	10	-2^{15}		$d_{16}+dd_0=d'_{16}+dd'_0$
63	$c_{16}+cc_0$	m_2	15			$c_{16}+cc_0=c'_{16}+cc'_0$
64	$b_{16}+bb_0$	m_9	21			$b_{16}+bb_0=b'_{16}+bb'_0$

Lampiran D: Sufficient Conditions untuk Second Iteration Differential

a_1	$a_{1,6}=0,\ a_{1,12}=0,\ a_{1,22}=1,\ a_{1,26}=0,\ a_{1,27}=1,\ a_{1,28}=0,\ a_{1,32}=1$
d_1	$d_{1,2}=0,\ d_{1,3}=0,\ d_{1,6}=0,\ d_{1,7}=a_{1,7},\ d_{1,8}=a_{1,8},\ d_{1,12}=1,\ d_{1,13}=a_{1,13},\ d_{1,16}=0,$ $d_{1,17}=a_{1,17},\ d_{1,18}=a_{1,18},\ d_{1,19}=a_{1,19},\ d_{1,20}=a_{1,20},\ d_{1,21}=a_{1,21},\ d_{1,22}=0,$ $d_{1,26}=0,\ d_{1,27}=1,\ d_{1,28}=1,\ d_{1,29}=a_{1,29},\ d_{1,30}=a_{1,30},\ d_{1,31}=a_{1,31},\ d_{1,32}=1$
c_1	$c_{1,2}=1,\ c_{1,3}=1,\ c_{1,4}=d_{1,4},\ c_{1,5}=d_{1,5},\ c_{1,6}=1,\ c_{1,7}=1,\ c_{1,8}=0,\ c_{1,9}=1,\ c_{1,12}=1,$ $c_{1,13}=0,\ c_{1,17}=1,\ c_{1,18}=1,\ c_{1,19}=1,\ c_{1,20}=1,\ c_{1,21}=1,\ c_{1,22}=0,\ c_{1,26}=1,\ c_{1,27}=1,$ $c_{1,28}=1,\ c_{1,29}=1,\ c_{1,30}=1,\ c_{1,31}=0,\ c_{1,32}=1$
b_1	$b_{1,1}=c_{1,1},\ b_{1,2}=0,\ b_{1,3}=0,\ b_{1,4}=0,\ b_{1,5}=1,\ b_{1,6}=0,\ b_{1,7}=0,\ b_{1,8}=0,\ b_{1,9}=0,$ $b_{1,10}=c_{1,10},\ b_{1,11}=c_{1,11},\ b_{1,12}=0,\ b_{1,13}=0,\ b_{1,17}=0,\ b_{1,18}=0,\ b_{1,19}=1,\ b_{1,20}=0,$ $b_{1,21}=0,\ b_{1,22}=0,\ b_{1,26}=1,\ b_{1,27}=0,\ b_{1,28}=1,\ b_{1,29}=1,\ b_{1,30}=1,\ b_{1,31}=0,\ b_{1,32}=1$
a_2	$a_{2,1}=0,\ a_{2,2}=0,\ a_{2,3}=0,\ a_{2,4}=0,\ a_{2,5}=1,\ a_{2,6}=0,\ a_{2,7}=1,\ a_{2,8}=0,\ a_{2,9}=0,$ $a_{2,10}=1,\ a_{2,11}=1,\ a_{2,12}=1,\ a_{2,13}=0,\ a_{2,17}=1,\ a_{2,18}=1,\ a_{2,19}=1,\ a_{2,20}=1,$ $a_{2,27}=0,\ a_{2,28}=1,\ a_{2,29}=0,\ a_{2,30}=0,\ a_{2,21}=0,\ a_{2,22}=1,\ a_{2,31}=1,\ a_{2,32}=0$
d_2	$d_{2,1}=0,\ d_{2,2}=1,\ d_{2,3}=1,\ d_{2,4}=0,\ d_{2,5}=1,\ d_{2,6}=0,\ d_{2,7}=1,\ d_{2,8}=0,\ d_{2,9}=0,$ $d_{2,10}=0,\ d_{2,11}=1,\ d_{2,12}=1,\ d_{2,13}=0,\ d_{2,17}=0,\ d_{2,18}=1,\ d_{2,21}=0,\ d_{2,22}=1,$ $d_{2,26}=0,\ d_{2,27}=1,\ d_{2,28}=0,\ d_{2,29}=0,\ d_{2,32}=0$
c_2	$c_{2,1}=1,\ c_{2,7}=0,\ c_{2,8}=0,\ c_{2,9}=0,\ c_{2,10}=1,\ c_{2,11}=1,\ c_{2,12}=1,\ c_{2,13}=1,$ $c_{2,16}=d_{2,16},\ c_{2,17}=1,\ c_{2,18}=0,\ c_{2,21}=0,\ c_{2,22}=0,\ c_{2,24}=d_{2,24},\ c_{2,25}=d_{2,25},$ $c_{2,26}=1,\ c_{2,27}=1,\ c_{2,28}=0,\ c_{2,29}=1,\ c_{2,32}=1$
b_2	$b_{2,1}=0,\ b_{2,2}=c_{2,2},\ b_{2,7}=1,\ b_{2,8}=1,\ b_{2,9}=1,\ b_{2,10}=1,\ b_{2,16}=1,\ b_{2,17}=0,\ b_{2,18}=1,$ $b_{2,21}=1,\ b_{2,22}=1,\ b_{2,24}=0,\ b_{2,25}=0,\ b_{2,26}=0,\ b_{2,27}=1,\ b_{2,28}=0,\ b_{2,29}=0,\ b_{2,32}=1$
a_3	$a_{3,1}=1,\ a_{3,2}=0,\ a_{3,7}=1,\ a_{3,8}=1,\ a_{3,9}=1,\ a_{3,10}=0,\ a_{3,13}=b_{2,13},\ a_{3,16}=0,$ $a_{3,17}=1,\ a_{3,18}=0,\ a_{3,24}=0,\ a_{3,25}=0,\ a_{3,26}=0,\ a_{3,27}=1,\ a_{3,28}=1,\ a_{3,29}=1,$ $a_{3,32}=1$
d_3	$d_{3,1}=0,\ d_{3,2}=0,\ d_{3,7}=1,\ d_{3,8}=1,\ d_{3,9}=1,\ d_{3,10}=1,\ d_{3,13}=0,\ d_{3,16}=1,\ d_{3,17}=1,$ $d_{3,18}=1,\ d_{3,19}=0,\ d_{3,24}=1,\ d_{3,25}=1,\ d_{3,26}=1,\ d_{3,27}=1,\ d_{3,32}=1$
c_3	$c_{3,1}=1,\ c_{3,2}=1,\ c_{3,7}=1,\ c_{3,8}=1,\ c_{3,9}=1,\ c_{3,10}=1,\ c_{3,13}=0,\ c_{3,14}=d_{3,14},$ $c_{3,15}=d_{3,15},\ c_{3,16}=1,\ c_{3,17}=1,\ c_{3,18}=0,\ c_{3,19}=1,\ c_{3,20}=d_{3,20},\ c_{3,32}=1$
b_3	$b_{3,8}=1,\ b_{3,13}=1,\ b_{3,14}=0,\ b_{3,15}=0,\ b_{3,16}=0,\ b_{3,17}=0,\ b_{3,18}=0,\ b_{3,19}=0,$ $b_{3,20}=1,\ b_{3,25}=c_{3,25},\ b_{3,26}=c_{3,26},\ b_{3,27}=c_{3,27},\ b_{3,28}=c_{3,28},\ b_{3,29}=c_{3,29},$ $b_{3,30}=c_{3,30},\ b_{3,31}=c_{3,31},\ b_{3,32}=1$
a_4	$a_{4,4}=1,\ a_{4,8}=0,\ a_{4,14}=1,\ a_{4,15}=1,\ a_{4,16}=1,\ a_{4,17}=1,\ a_{4,18}=1,\ a_{4,19}=1,\ a_{4,20}=1$ $a_{4,25}=1,\ a_{4,26}=1,\ a_{4,27}=1,\ a_{4,28}=1,\ a_{4,29}=1,\ a_{4,30}=1,\ a_{4,31}=0,\ a_{4,32}=0$
d_4	$d_{4,4}=1,\ d_{4,8}=1,\ d_{4,14}=1,\ d_{4,15}=1,\ d_{4,16}=1,\ d_{4,17}=1,\ d_{4,18}=1,\ d_{4,19}=0,\ d_{4,20}=1$ $d_{4,25}=0,\ d_{4,26}=0,\ d_{4,27}=0,\ d_{4,28}=0,\ d_{4,29}=0,\ d_{4,30}=0,\ d_{4,31}=1,\ d_{4,32}=0$
c_4	$c_{4,4}=0,\ c_{4,16}=0,\ c_{4,25}=1,\ c_{4,26}=0,\ c_{4,27}=1,\ c_{4,28}=1,\ c_{4,29}=1,\ c_{4,30}=1$ $c_{4,31}=1,\ c_{4,32}=0$
b_4	$b_{4,30}=1,\ b_{4,32}=0$
a_5	$a_{5,4}=b_{4,4},\ a_{5,16}=b_{4,16},\ a_{5,18}=0,\ a_{5,32}=0$
d_5	$d_{5,18}=1,\ d_{5,30}=a_{5,30},\ d_{5,32}=0$
c_5	$c_{5,18}=0,\ c_{5,32}=0$
b_5	$b_{5,32}=0,$
a_6-b_6	$a_{6,18}=b_{5,18}, a_{6,32}=0,\ d_{6,32}=0,\ c_{6,32}=0,\ b_{6,32}=c_{6,32}+1$
$c_9,\ b_{12}$	$\phi_{34,32}=1,\ b_{12,32}=d_{12,32},$
$a_{13}-b_{13}$	$a_{13,32}=c_{12,32},\ d_{13,32}=b_{12,32}+1,\ c_{13,32}=a_{13,32},\ b_{13,32}=d_{13,32}$
$a_{14}-b_{14}$	$a_{14,32}=c_{13,32},\ d_{14,32}=b_{13,32},\ c_{14,32}=a_{14,32},\ b_{14,32}=d_{14,32}$
$a_{15}-b_{15}$	$a_{15,32}=c_{14,32},\ d_{15,32}=b_{14,32},\ c_{15,32}=a_{15,32},\ b_{15,32}=d_{15,32}+1$
a_{16}	$a_{16,26}=1,\ a_{16,32}=c_{15,32}$
d_{16}	$d_{16,26}=1, d_{16,32}=b_{15,32}$
c_{16}	$c_{16,26}=1, c_{16,32}=a_{16,32}$
b_{16}	$b_{16,26}=1$