KRIPTOGRAFI MESSAGE DIGEST SEBAGAI SALAH SATU ENKRIPSI POPULER

Akhmad Ratriono Anggoro
Program Studi Teknik Informatika
Sekolah Teknil Elektro dan Informatika
Institut Teknologi Bandung
If15003@students.if.itb.ac.id

Abstrak

Makalah ini membahas studi secara pragmatis tentang algoritma enkripsi Message Digest yang sering disebut dengan MD. MD adalah salah satu dari sekian banyak algoritma enkripsi yang menggunakan fungsi hash sebagai dasarnya. Sampai saat ini, rangkaian algoritma MD mencapai serinya yang ke 5 (MD5). Namun, beberapa enkripsi lain seperti SHA1, SHA-0, dan lain-lain, dianggap sebagai penerus dari serial MD karena masih menggunakan basis yang sama yaitu fungsi hash.

Fungsi hash sendiri adalah sebuah metode yang menghasilkan alamat memori dari kunci perintah yang diberikan. Fungsi ini membuat sebuah "sidik jari digital" dari data jenis apapun dengan cara mentranspose dan mensubtitusi data tersebut. "sidik jari digital" ini dikenal luas dengan sebutan hash value. Fungsi hash ini bersifat non-injektif. Ini berarti 2 nilai yang sama dalam suatu range belum tentu (namun sangat disarankan) memiliki domain yang sama. Selain itu, fungsi hash memiliki domain yang tidak terbatas dan range yang dapat ditentukan pengguna sesuka hati(terbatas).

Dalam perkembangannya, fungsi hash yang dipakai di kriptografi secara umum memiliki sifat yang sama dalam memroses domain. Fungsi hash dalam kriptografi membaca string, rentetan karakter yang sangat panjang, kemudian memrosesnya dan menghasilkan sebuah kode dengan panjang digit tertentu. Fungsi hash yang digunakan dalam kriptografi bersifat sangat ideal, karena perubahan 1 karakter dalam string yang diproses akan mengubah kode hasil enkripsi secara total, sehingga hampir mustahil untuk dipecahkan.

Kata kunci : MD2, MD4, MD5, SHA-1, birthday paradox, *collision*, Xiaoyun Wang

1. Pendahuluan

Sejak teknologi internet mulai diperkenalkan sekitar 30 tahun yang lalu, perkembangannya dapat dikatakan hampir tak terkontrol. Setiap hari ada sekian banyak orang yang mampu membuat sebuah algoritma baru untuk mengoptimalisasi keadaan di dunia maya tersebut, sementara yang lainnya berusaha menghancurkan hirearki yang sudah ada sejak World Wide Web Consortium didirikan.

Lama kelamaan internet menjadi sebuah ladang bisnis besar yang mampu menghasilkan keuntungan maha besar, atau sebagai sarana bertukar informasi penting tanpa hambatan ruang dan waktu. Dalam 2 hal di atas, kerahasiaan adalah suatu hal yang mutlak. Karena itu, dibutuhkan sebuah sistem yang dapat menjaga kerahasiaan dari pesan-pesan yang dikirim via internet agar pihak yang tidak berkepentingan tidak dapat membacanya. Sistem ini dikenal di masa sekarang dengan sebutan kriptografi.

Kriptografi, secara umum terdiri dari 2 proses utama, yaitu enkripsi dan dekripsi. Enkripsi, adalah kegiatan yang dilakukan oleh pengirim pesan. Enkripsi merupakan rangkaian kegiatan untuk mengubah pesan menjadi sebuah kode rahasia dengan kunci-kunci unik. Pesan ini kemudian akan di dekripsi, diubah kembali ke bentuk asalnya oleh mereka yang berhak membaca pesan tersebut.

Sejarah Kriptografi telah ada sejak jaman sebelum internet. Yang paling menonjol adalah Kode Telegram Zimmermann. Kode ini berisi rentetan angka yang dikirim pemerintah Jerman ke pemerintah Meksiko. Isi dari telegram tersebut adalah ajakan kepada Meksiko untuk ikut serta 'meramaikan' perang dengan menyerang Amerika. Kode ini akhirnya dipecahkan Intelijen Inggris, dan membuat Amerika yang semula netral ikut serta dalam Perang Dunia Pertama.

DI dunia digital, pertama kalinya pada awal tahun 70-an NIST merancang sebuah enkripsi bernama DES, Data Encryption System. DES bekerja dalam sistem 56 bit yang merupakan sebuah kelebihan

tersendiri dalam mengkodekan sebuah pesan di masa itu. DES tidak memiliki sedikitpun flaw di masa awal-awal perilisannya. Namun, seiring dengan kemajuan teknologi perangkat keras dan sistem operasi baru, DES mulai termakan usia. DES dapat dengan mudah di dekripsi dalam waktu singkat berkat kencangnya perangkat keras generasi baru dalam memroses data.

Mengetahui hal ini, publik tidak dapat lagi mempercayai DES. Dalam perkembangannya, akhirnya banyak masyarakat sipil yang berlomba-lomba membuat sebuah algoritma kriptografi yang benar-benar efisien dan aman. Salah satu di antara mereka yang berhasil adalah Ronald Rivest, seorang professor dari MIT. Algoritma yang ia rancang diambil atas dasar *hash function* diberi nama MD atau Message Digest Algorithm.

2. MD 2

Message Digest 2 pertama kali dirancang pada tahun 1989 dan dirancang untuk komputer berbasiskan 8-bit. Detail tentang MD2 dapat dilihat di RFC 1319. Walaupun memiliki banyak flaw, sampai sekarang MD2 masih dipakai sebagai infrastruktur untuk sertifikat RSA.

MD 2 mengubah pesan string ke dalam kode heksadesimal 32 bit. Secara fisik, MD2 bekerja dengan mengkompresi 128 bit *hash value* dari pesan sembarang ke dalam blok-blok yang masing-masing berukuran 128 bit (16 byte) kemudian menambahkan sebuah checksum. Untuk kalkulasi yang sebenarnya, digunakan blok sebesar 48 byte dan tabel 256 byte yang dihasilkan secara tidak langsung dari pi. Apabila semua block dari pesan yang dipanjangkan telah diproses, fragmen pertama dari blok 48 byte menjadi *hash value* dari pesan.

Contoh dari enkripsi MD 2 pada 43 byte ASCII codes,

```
MD2("The quick brown fox jumps
over the lazy dog")
menjadi
03d85a0d629d2c442e987525319fc471
```

Apabila satu karakter diubah, kode MD 2 nya akan berubah dengan drastis.

```
MD2("The quick brown fox jumps
over the lazy cog")
menjadi
6b890c9292668cdbbfda00a4ebf31f05
```

2.1 Kelemahan pada MD2

Fungsi hash, memiliki kelemahan utama yang biasa disebut dengan *collision*. Kelemahan ini didapat berdasarkan sifat injektifnya, di mana range yang sama belum tentu memiliki domain yang sama. Secara matematis *collision* dalam hash function dibagi menjadi dua. Apabila $H(y)=H(x)$ namun sulit untuk menemukan $y \neq x$, *collision* ini disebut *collision* lemah. Tetapi jika $H(y)=H(x)$ namun pasangan (x,y) tidak dapat ditemukan, ini disebut *collision* kuat.

Dalam MD2 *flaw* tersebut ditemukan pertama kali oleh Rogier dan Chauvaud (1997) untuk fungsi kompresi pada MD2, namun cacat dalam MD2 ini tidak fatal dan mereka tidak mampu menemukan kelemahan lebih lanjut dari MD2. Akhirnya pada tahun 2004, fungsi kompresi MD2 dibuktikan sangat rentan pada cracking code dengan kompleksitas 2^{104} oleh Muller.

3. MD4

Ditulis tahun 1990 oleh rivest, MD4 adalah versi revisi dari MD2, MD4 digunakan terutama untuk memeriksa integritas dari sebuah pesan. Enkripsi ini masih menggunakan panjang 128 bit dan menggunakan fungsi hash. MD4 memiliki pengaruh besar dalam seri MD yang terakhir (MD5) dan algoritma-algoritma lain sesudahnya (SHA, RIPEMD). Sampai sekarang enkripsi MD4 digunakan untuk identifikasi unik file transfer dalam Peer2Peer populer seperti eDonkey2000.

Kelemahan dari MD4 dipublikasikan pertama kali oleh Den Boers dan Bosselaer, namun sebenarnya ini adalah *flaw* general yang masih bekerja di MD5, bahasan mengenai kelemahan ini bisa dilihat di subbab kelemahan MD5.

4. MD5

MD5, melanjutkan seri sebelumnya, masih memiliki panjang 128 bit. MD5, yang ditulis tahun 1991, menjadi standar internet sampai 2004 ketika kelemahan fatal dalam sekuritasnya ditemukan.

4.1 Aplikasi

MD5 digunakan secara luas di dunia perangkat lunak sebagai alat jaminan kebenaran file yang telah diunduh. Kebanyakan server penyedia file selalu memberikan sebuah fungsi *checksum* yang telah dihitung dengan format MD5 untuk memeriksa kesesuaian file. Ketika pengunduh selesai mengunduh file yang diinginkannya, Dia hanya perlu memeriksa dengan *checksum* yang telah disediakan.

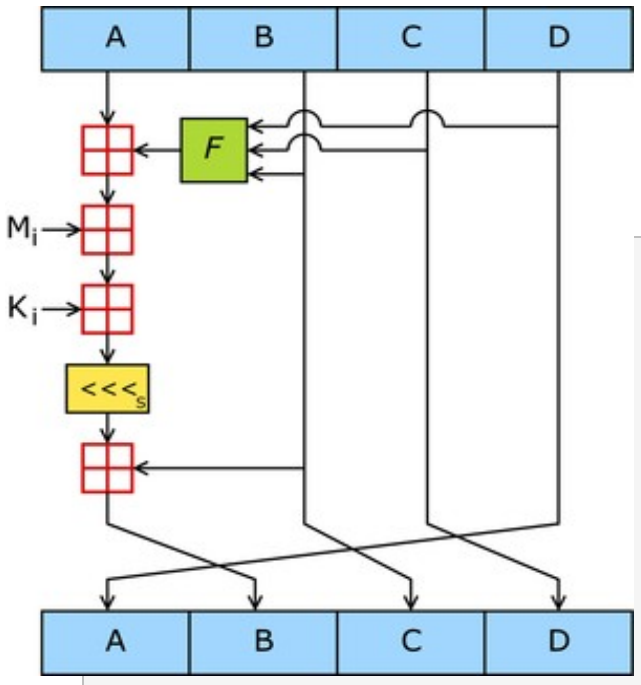
Namun, karena begitu mudahnya untuk menciptakan *collision* pada MD5, bisa jadi *checksum* akan memvalidasi dan membenarkan file yang diunduh, padahal yang dimaksud adalah file yang benar-benar berbeda.

MD5 juga seringkali dipakai untuk enkripsi password, namun, karena sudah banyak beredar tabel-tabel heksadesimal untuk membalik kode dari MD5, keamanan penggunaan MD5 sebagai metode enkripsi password sudah sangat tidak aman.

Kebanyakan dari praktisi kriptografer menggunakan sedikit 'garam' dalam password MD5 mereka dan menggunakan hashing lebih dari sekali. 'Garam' di sini berarti menggunakan bit-bit random sebagai *carrier* input ke dalam enkripsi. Salt terbukti efektif untuk melawan *dictionary attack* pada password yang disimpan.

CRAM MD5, challenge response authentication mechanism, adalah sebuah mekanisme pengesahan client dengan MD5. Dalam prosesnya, CRAM MD5 mengirimkan string ke client, kemudian client membalas dengan mengirimkan username diikuti spasi dan rentetan kode 16 byte dengan notasi hexadesimal. Kode ini adalah password yang nantinya akan di autentikasi oleh server.

4.2 Algoritma



Keterangan ; Mi = 32 bit pesan masuk, Ki = 32 bit konstanta, F = fungsi non linear, <<< rotasi bit terkiri dari pesan, ⊞ penambahan modulo 2^{32}. Satu operasi MD5 terdiri dari 64 buah proses di atas. Proses-proses tersebut dikelompokkan dalam 4 grup yang masing-masing berisi 16 proses.

Pada dasarnya, MD5 menghasilkan kode dengan panjang tetap dari sebuah pesan sembarang dengan panjang sembarang. Pesan yang masuk dipecah menjadi fragmen-fragmen dengan panjang masing-masing 512 bit. Pesan tersebut haruslah dimanipulasi sehingga dapat dibagi 512. Proses manipulasi ini menambahkan bit 1 ke akhir pesan, lalu menambahkan 0 sampai panjang pesan sama dengan kelipatan 512 dikurangi 64. Enam puluh empat bit terakhir ini digunakan untuk menyimpan integer dari panjang pesan yang sebenarnya.

Algoritma utama MD5 bekerja dalam keadaan 128 bit, yang terbagi menjadi empat 32 bit *word*, A, B, C, dan D. Kemudian algoritma akan bekerja di setiap fragmen 512 bit. Proses dalam setiap fragmen ini terdiri dari empat putaran, satu putaran terdiri dari 16 operasi berdasarkan fungsi F, penambahan modular, dan rotasi bit kiri.

Dalam prosesnya terdapat 4 kemungkinan yang berbeda untuk fungsi F, masing-masing digunakan dalam putaran yang berbeda. Fungsi-fungsi tersebut adalah

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$

Dalam bahasa C, prosedur MD5 direalisasikan sebagai berikut,

```
/*dikutip dengan perubahan dari
ReHash (www.reichlsoft.de.vu)*/
#include <stdio.h>
#include <stdlib.h>

#include "md5.h"

/* Typedef a 32 bit type */
#ifndef UINT4
typedef unsigned long int UINT4;
#endif

/* Data structure for MD5 (Message Digest) computation */
```

```
typedef struct {
UINT4           i[2];           /* Number of _bits_ handled mod 2^64
*/
UINT4           buf[4];         /* Scratch buffer
*/
unsigned char   in[64];         /* Input buffer
*/
unsigned char   digest[16];     /* Actual digest after MD5Final call
*/
} MD5_CTX;

/* Padding */
static unsigned char MD5_PADDING[64] = {
         0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
 };

 /* MD5_F, MD5_G and MD5_H are basic MD5 functions: selection, majority,
parity */
#define MD5_F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define MD5_G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define MD5_H(x, y, z) ((x) ^ (y) ^ (z))
#define MD5_I(x, y, z) ((y) ^ ((x) | (~z)))

 /* ROTATE_LEFT rotates x left n bits */
#ifndef ROTATE_LEFT
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))
#endif

/* MD5_FF, MD5_GG, MD5_HH, and MD5_II transformations for rounds 1, 2,
3, and 4 */
/* Rotation is separate from addition to prevent recomputation */
#define MD5_FF(a, b, c, d, x, s, ac) {(a) += MD5_F ((b), (c), (d)) + (x)
+ (UINT4)(ac); (a) = ROTATE_LEFT ((a), (s)); (a) += (b); }
#define MD5_GG(a, b, c, d, x, s, ac) {(a) += MD5_G ((b), (c), (d)) + (x)
+ (UINT4)(ac); (a) = ROTATE_LEFT ((a), (s)); (a) += (b); }
#define MD5_HH(a, b, c, d, x, s, ac) {(a) += MD5_H ((b), (c), (d)) + (x)
+ (UINT4)(ac); (a) = ROTATE_LEFT ((a), (s)); (a) += (b); }
#define MD5_II(a, b, c, d, x, s, ac) {(a) += MD5_I ((b), (c), (d)) + (x)
+ (UINT4)(ac); (a) = ROTATE_LEFT ((a), (s)); (a) += (b); }

/* Constants for transformation */
#define MD5_S11 7  /* Round 1 */
#define MD5_S12 12
```

```
#define MD5_S13 17
#define MD5_S14 22
#define MD5_S21 5  /* Round 2 */
#define MD5_S22 9
#define MD5_S23 14
#define MD5_S24 20
#define MD5_S31 4  /* Round 3 */
#define MD5_S32 11
#define MD5_S33 16
#define MD5_S34 23
#define MD5_S41 6  /* Round 4 */
#define MD5_S42 10
#define MD5_S43 15
#define MD5_S44 21

/* Basic MD5 step. MD5_Transform buf based on in */
static void MD5_Transform (UINT4 *buf, UINT4 *in)
{
         UINT4 a = buf[0], b = buf[1], c = buf[2], d = buf[3];

         /* Round 1 */
MD5_FF ( a, b, c, d, in[ 0], MD5_S11, (UINT4) 3614090360u); /* 1 */
MD5_FF ( d, a, b, c, in[ 1], MD5_S12, (UINT4) 3905402710u); /* 2 */
MD5_FF ( c, d, a, b, in[ 2], MD5_S13, (UINT4)  606105819u); /* 3 */
MD5_FF ( b, c, d, a, in[ 3], MD5_S14, (UINT4) 3250441966u); /* 4 */
MD5_FF ( a, b, c, d, in[ 4], MD5_S11, (UINT4) 4118548399u); /* 5 */
MD5_FF ( d, a, b, c, in[ 5], MD5_S12, (UINT4) 1200080426u); /* 6 */
MD5_FF ( c, d, a, b, in[ 6], MD5_S13, (UINT4) 2821735955u); /* 7 */
MD5_FF ( b, c, d, a, in[ 7], MD5_S14, (UINT4) 4249261313u); /* 8 */
MD5_FF ( a, b, c, d, in[ 8], MD5_S11, (UINT4) 1770035416u); /* 9 */
MD5_FF ( d, a, b, c, in[ 9], MD5_S12, (UINT4) 2336552879u); /* 10 */
MD5_FF ( c, d, a, b, in[10], MD5_S13, (UINT4) 4294925233u); /* 11 */
MD5_FF ( b, c, d, a, in[11], MD5_S14, (UINT4) 2304563134u); /* 12 */
MD5_FF ( a, b, c, d, in[12], MD5_S11, (UINT4) 1804603682u); /* 13 */
MD5_FF ( d, a, b, c, in[13], MD5_S12, (UINT4) 4254626195u); /* 14 */
MD5_FF ( c, d, a, b, in[14], MD5_S13, (UINT4) 2792965006u); /* 15 */
MD5_FF ( b, c, d, a, in[15], MD5_S14, (UINT4) 1236535329u); /* 16 */

/* Round 2 */
MD5_GG ( a, b, c, d, in[ 1], MD5_S21, (UINT4) 4129170786u); /* 17 */
MD5_GG ( d, a, b, c, in[ 6], MD5_S22, (UINT4) 3225465664u); /* 18 */
MD5_GG ( c, d, a, b, in[11], MD5_S23, (UINT4)  643717713u); /* 19 */
MD5_GG ( b, c, d, a, in[ 0], MD5_S24, (UINT4) 3921069994u); /* 20 */
MD5_GG ( a, b, c, d, in[ 5], MD5_S21, (UINT4) 3593408605u); /* 21 */
MD5_GG ( d, a, b, c, in[10], MD5_S22, (UINT4)   38016083u); /* 22 */
MD5_GG ( c, d, a, b, in[15], MD5_S23, (UINT4) 3634488961u); /* 23 */
MD5_GG ( b, c, d, a, in[ 4], MD5_S24, (UINT4) 3889429448u); /* 24 */
MD5_GG ( a, b, c, d, in[ 9], MD5_S21, (UINT4)  568446438u); /* 25 */
```

```
MD5_GG ( d, a, b, c, in[14], MD5_S22, (UINT4) 3275163606u); /* 26 */
MD5_GG ( c, d, a, b, in[ 3], MD5_S23, (UINT4) 4107603335u); /* 27 */
MD5_GG ( b, c, d, a, in[ 8], MD5_S24, (UINT4) 1163531501u); /* 28 */
MD5_GG ( a, b, c, d, in[13], MD5_S21, (UINT4) 2850285829u); /* 29 */
MD5_GG ( d, a, b, c, in[ 2], MD5_S22, (UINT4) 4243563512u); /* 30 */
MD5_GG ( c, d, a, b, in[ 7], MD5_S23, (UINT4) 1735328473u); /* 31 */
MD5_GG ( b, c, d, a, in[12], MD5_S24, (UINT4) 2368359562u); /* 32 */
/* Round 3 */
     MD5_HH ( a, b, c, d, in[ 5], MD5_S31, (UINT4) 4294588738u); /* 33
*/
     MD5_HH ( d, a, b, c, in[ 8], MD5_S32, (UINT4) 2272392833u); /* 34
*/
     MD5_HH ( c, d, a, b, in[11], MD5_S33, (UINT4) 1839030562u); /* 35
*/
     MD5_HH ( b, c, d, a, in[14], MD5_S34, (UINT4) 4259657740u); /* 36
*/
     MD5_HH ( a, b, c, d, in[ 1], MD5_S31, (UINT4) 2763975236u); /* 37
*/
     MD5_HH ( d, a, b, c, in[ 4], MD5_S32, (UINT4) 1272893353u); /* 38
*/
     MD5_HH ( c, d, a, b, in[ 7], MD5_S33, (UINT4) 4139469664u); /* 39
*/
     MD5_HH ( b, c, d, a, in[10], MD5_S34, (UINT4) 3200236656u); /* 40
*/
     MD5_HH ( a, b, c, d, in[13], MD5_S31, (UINT4)  681279174u); /* 41
*/
     MD5_HH ( d, a, b, c, in[ 0], MD5_S32, (UINT4) 3936430074u); /* 42
*/
     MD5_HH ( c, d, a, b, in[ 3], MD5_S33, (UINT4) 3572445317u); /* 43
*/
     MD5_HH ( b, c, d, a, in[ 6], MD5_S34, (UINT4)   76029189u); /* 44
*/
     MD5_HH ( a, b, c, d, in[ 9], MD5_S31, (UINT4) 3654602809u); /* 45
*/
     MD5_HH ( d, a, b, c, in[12], MD5_S32, (UINT4) 3873151461u); /* 46
*/
     MD5_HH ( c, d, a, b, in[15], MD5_S33, (UINT4)  530742520u); /* 47
*/
     MD5_HH ( b, c, d, a, in[ 2], MD5_S34, (UINT4) 3299628645u); /* 48
*/

         /* Round 4 */
     MD5_II ( a, b, c, d, in[ 0], MD5_S41, (UINT4) 4096336452u); /* 49
*/
     MD5_II ( d, a, b, c, in[ 7], MD5_S42, (UINT4) 1126891415u); /* 50
*/
     MD5_II ( c, d, a, b, in[14], MD5_S43, (UINT4) 2878612391u); /* 51
*/
     MD5_II ( b, c, d, a, in[ 5], MD5_S44, (UINT4) 4237533241u); /* 52
*/
     MD5_II ( a, b, c, d, in[12], MD5_S41, (UINT4) 1700485571u); /* 53
*/
```

```
     MD5_II ( d, a, b, c, in[ 3], MD5_S42, (UINT4) 2399980690u); /* 54
*/
     MD5_II ( c, d, a, b, in[10], MD5_S43, (UINT4) 4293915773u); /* 55
*/
     MD5_II ( b, c, d, a, in[ 1], MD5_S44, (UINT4) 2240044497u); /* 56
*/
     MD5_II ( a, b, c, d, in[ 8], MD5_S41, (UINT4) 1873313359u); /* 57
*/
     MD5_II ( d, a, b, c, in[15], MD5_S42, (UINT4) 4264355552u); /* 58
*/
     MD5_II ( c, d, a, b, in[ 6], MD5_S43, (UINT4) 2734768916u); /* 59
*/
     MD5_II ( b, c, d, a, in[13], MD5_S44, (UINT4) 1309151649u); /* 60
*/
     MD5_II ( a, b, c, d, in[ 4], MD5_S41, (UINT4) 4149444226u); /* 61
*/
     MD5_II ( d, a, b, c, in[11], MD5_S42, (UINT4) 3174756917u); /* 62
*/
     MD5_II ( c, d, a, b, in[ 2], MD5_S43, (UINT4)  718787259u); /* 63
*/
     MD5_II ( b, c, d, a, in[ 9], MD5_S44, (UINT4) 3951481745u); /* 64
*/
          buf[0] += a;
          buf[1] += b;
          buf[2] += c;
          buf[3] += d;
}

// Set pseudoRandomNumber to zero for RFC MD5 implementation
void MD5Init (MD5_CTX *mdContext, unsigned long pseudoRandomNumber)
{
          mdContext->i[0] = mdContext->i[1] = (UINT4)0;

          /* Load magic initialization constants */
          mdContext->buf[0] = (UINT4)0x67452301 + (pseudoRandomNumber *
11);
          mdContext->buf[1] = (UINT4)0xefcdab89 + (pseudoRandomNumber *
71);
          mdContext->buf[2] = (UINT4)0x98badcfe + (pseudoRandomNumber *
37);
          mdContext->buf[3] = (UINT4)0x10325476 + (pseudoRandomNumber *
97);
}

void MD5Update (MD5_CTX *mdContext, unsigned char *inBuf, unsigned int
inLen)
{
          UINT4           in[16];
          int             mdi = 0;
          unsigned int    i = 0, ii = 0;
```

```
        /* Compute number of bytes mod 64 */
        mdi = (int)((mdContext->i[0] >> 3) & 0x3F);

        /* Update number of bits */
        if ((mdContext->i[0] + ((UINT4)inLen << 3)) < mdContext->i[0])
mdContext->i[1]++;
        mdContext->i[0] += ((UINT4)inLen << 3);
        mdContext->i[1] += ((UINT4)inLen >> 29);

        while (inLen--) {
                /* Add new character to buffer, increment mdi */
                mdContext->in[mdi++] = *inBuf++;

                /* Transform if necessary */
                if (mdi == 0x40) {
                        for (i = 0, ii = 0; i < 16; i++, ii += 4)
                           in[i] = (((UINT4)mdContext->in[ii+3]) << 24)
|
                                   (((UINT4)mdContext->in[ii+2]) << 16)
|
                                   (((UINT4)mdContext->in[ii+1]) << 8)
|
                                   ((UINT4)mdContext->in[ii]);
                        MD5_Transform (mdContext->buf, in);
                        mdi = 0;
                }
        }
}

void MD5Final (MD5_CTX *mdContext)
{
        UINT4           in[16];
        int             mdi = 0;
        unsigned int    i = 0, ii = 0, padLen = 0;

        /* Save number of bits */
        in[14] = mdContext->i[0];
        in[15] = mdContext->i[1];

        /* Compute number of bytes mod 64 */
        mdi = (int)((mdContext->i[0] >> 3) & 0x3F);

        /* Pad out to 56 mod 64 */
        padLen = (mdi < 56) ? (56 - mdi) : (120 - mdi);
        MD5Update (mdContext, MD5_PADDING, padLen);

        /* Append length in bits and transform */
        for (i = 0, ii = 0; i < 14; i++, ii += 4)
```

```
                in[i] = (((UINT4)mdContext->in[ii+3]) << 24) |
                        (((UINT4)mdContext->in[ii+2]) << 16) |
                        (((UINT4)mdContext->in[ii+1]) <<  8) |
                        ((UINT4)mdContext->in[ii]);
        MD5_Transform (mdContext->buf, in);

        /* Store buffer in digest */
        for (i = 0, ii = 0; i < 4; i++, ii += 4) {
                mdContext->digest[ii]   = (unsigned char)( mdContext-
>buf[i]        & 0xFF);
                mdContext->digest[ii+1] = (unsigned char)((mdContext-
>buf[i] >>  8) & 0xFF);
                mdContext->digest[ii+2] = (unsigned char)((mdContext-
>buf[i] >> 16) & 0xFF);
                mdContext->digest[ii+3] = (unsigned char)((mdContext-
>buf[i] >> 24) & 0xFF);
        }
}

00291 void CalculateMD5(unsigned char *buffer, int length, unsigned char
*checksum)
{
        int             i;
        MD5_CTX         m_md5;

        MD5Init(&m_md5, 0);
        MD5Update(&m_md5, buffer, length);
        MD5Final(&m_md5);

        for (i = 0; i < 16; i++) sprintf(checksum+i*2,"%02X",
m_md5.digest[i]);
}
```

4. 3 Kriptoanalis dan Kelemahan

MD5 dibuat sebagai pengganti MD4 karena alasan keamanan. MD4 memiliki *flaw* fatal dalam proses eksekusinya sehingga kode 32 bit heksadesimal yang dihasilkannya dapat ditembus walaupun waktu yang diperlukan untuk membaca kode relatif lama.

Sama pada MD4 Den Boer dan Bosselaer, *flaw* yang sama masih bisa ditemukan pada MD5. *Flaw* ini ditemukan pada fungsi kompresi vektor dimana, jika MD5compress(*I*,*X*) = MD5compress(*J*,*X*) namun terdapat perbedaan 4 bith pada I dan J, padahal seharusnya I = J. *Collision* ini masih dianggap *pseudo,* karena efek nya tidak terlalu parah, dan hanya bekerja pada vektor.

Terdapat sebuah fenomena unik dalam hash yang disebut dengan birthday paradox. Birthday paradox sebuah teori keunikan tentang kemungkinan 2 orang memiliki hari ulang tahun sama pada sekelompok orang. Sebenarnya ini bukan paradox, karena logika matematik membuktikan benar namun intuisi dan akal sehat berkata sebaliknya sehingga dianggap sebagai sebuah paradox.

Adapun rumus dasar dari paradox tersebut adalah: Pandang n sebagai jumlah orang dan p(n) kemungkinan 2 orang memiliki ulang tahun sama, dengan fungsi pidgeonhole diperoleh

$$\bar{p}(n) = 1 \cdot \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdots \left(1 - \frac{n-1}{365}\right)$$

$$= \frac{365 \cdot 364 \cdots (365 - n + 1)}{365^n} = \frac{365!}{365^n (365 - n)}$$

Fungsi diatas berlaku untuk selang satu tahun (365) hari, sedangkan untuk mencari kesamaan kode pada MD5 yang memiliki panjang 128bit lebih mudah sampai hampir sepertiganya, dari sini diketahui bahwa MD5 rentan terhadap serangan Bithday Paradox.

Hal ini dibuktikan dalam serangan terhadap MD5 source pada maret 2004, namun ini tidaklah berlangsung lama karena 5 bulan kemudian, seorang profesor dari Shandong university yang bernama XiouYun Wang menemukan rentetan *Collision* dalam algoritma hash di MD5 berdasarkan pseudo-*collison* Den Boer, adapun analisis Wang adalah sebagai berikut:
Ambil Vo dengan nilai standar untuk 2 buah pesan 1024 bit
Vo =

- Ao = 0x67452301
- Bo=0xEFCDAB89
- Co=0x98BADCFE
- Do=0x10325476

Dan untuk M dan N sebagai pesan 1024 bit maka

$M' = M+\Delta C_1,C_1$
$N' = N+\Delta C_1,C_1$

Dari fungsi di atas didapat MD5(M,N) = MD5(M',N'). *Collision* di atas bekerja di puluhan pasangan lainnya. Hebatnya, serangan dengan algoritma di atas hanya bekerja 1 jam pada mesin IBM p690. Tidak berhenti sampai di sana, 2 hari kemudian, Vlastimil Klima, berhasil menulis algoritma untuk mencari *collision* yang lebih cepat di sebuah laptop hanya dalam waktu beberapa jam.

Setelah ditemukan begitu banyak *collision* dalam enkripsi MD5, para kriptografer akhirnya menyarankan SHA-1, sebagai bentuk yang lebih aman dari MD5.

5. Sekilas tentang SHA, Penerus MD5

SHA-1 dikembangkan oleh National Security Agency (NSA) Amerika. SHA dalam perkembangannya memiliki berbagai macam varian, SHA-1, SHA-224, SHA-256, SHA-512. Dalam versi-versi setelah SHA-224, angka dibelakang *dash* menunjukkan bit dimana enkripsi bekerja. Bit menjadi semakin besar sehingga Birthday attack menjadi semakin sulit untuk dieksekusi .

SHA sendiri masih menggunakan *hash***,** namun bukan berarti masih membawa kelemahan dari MDA5. SHA masih memiliki *collision* sebagai sebuah ciri khas *hash***,** tetapi tidak sefatal yang terjadi pada Message Digest. *Collision attack* kini memiliki waktu eksekusi yang jauh lebih lama daripada MD. Namun bukan berarti SHA tanpa cacat. Mengenai SHA lebih lanjut tidak akan dijelaskan di makalah ini.

6. Konklusi

MD bisa dikatakan hal yang paling banyak dibicarakan oleh para kriptografer sebelum era AES karena aplikasinya yang mudah, umum, namun relatif sulit untuk dibajak. Manusia selalu tertarik pada kecepatan dan *hash* menyediakannya.Rivest menulis sebuah *hash* untuk kita., dan beberapa dari kita berusaha memajukan apa yang ditulis Rivest untuk kode yang lebih baik. Sangat sederhana dan *reliable.*

MD5 mudah sekali untuk diimplementasikan, semua bahasa pemrograman dapat dengan mudah merealisasikan algoritma MD5, bahkan assembly sekalipun. Karena itu, sampai saat ini, MD(5)masih umum digunakan walaupun *collision* masih menghantui.

Secara umum, Implementasi kriptografi pada masa modern bisa dikatakan primitif. Bukan pada kompleksitasnya, tetapi lebih pada manajemen kompleksitas, *hash* ataupun *cipher*. Aplikasi semua variannya mendasar dan sama efektif. Tidak memberikan dampak khusus bagi pengguna. Pembeda hanyalah kecepatan akses dan kecepatan hardware. Jadi pada dasarnya, kemampuan untuk membuat sebuah kode yang optimal bukannya tidak ada. Masalahnya hanya pada sebuah persaingan, bukan hanya pada pemecah kode dan pengkode, tapi antara pengkode itu sendiri. Namun, dunia tanpa persaingan adalah dunia yang hambar.

Daftar Pustaka

[1] S. Kent and R. Atkinson.
Security Architecture for the Internet Protocol. In *RFC 2401*, 1998.
Tanggal akses 28 Desember 2007

[2] en.wikipedia.org
Tanggal akses 1, 2, 3 Januari 2007

[3] www.reichlsoft.de.vu
Tanggal akses 28 Desember 2007

[4] Yunfei Wu and Stephan Wong.

Design Challenges in Security Processing
Delft University of technology

[5] Rivest, MD5 the Message Digest Algorithm
Protocol In *RFC 1321*, 1991
Tanggal akses 28 Desember 2007

[6] Stevens, Mark.
Fast Collision Attack on MD5
Eindhoven Institute of Technology

[7] Xiaoyun Wang, Dengguo Feng, Xuejia Lai.
Collision on Hash Function
Shandong Republic University.

[8] Andy Green Peter Barth, Jeff Mears,
"Project b (hacking) overview," 2004,
http www.xbox-linux . org /docs/
projectboverview . html.
Tanggal akses 28 Desember 2007