Studi Perbandingan Fungsi *Hash* yang Sudah Ada

Rianto Fendy Kristanto - 13507036[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]if17036@students.if.itb.ac.id, f_nd_x@yahoo.com

***Abstract*—Fungsi *hash* adalah fungsi yang menerima masukkan *string* yang panjangnya sembarang dan mengkonversinya menjadi *string* keluaran yang panjangnya tetap atau *fixed*. Fungsi *hash* dapat diaplikasikan dalam kehidupan sehari-hari. Salah satu contoh aplikasi fungsi *hash* adalah verifikasi sebuah salinan dokumen dengan dokumen aslinya. Verifikasi salinan dokumen tersebut bertujuan untuk mengetahui apakah salinan dokumen masih sama atau sudah mengalami perubahan (secara sengaja atau tidak sengaja) jika dibandingkan dengan dokumen aslinya.**

Fungsi *hash* satu-arah (*one-way hash*) merupakan sebuah fungsi *hash* yang bekerja dalam satu arah. Maksudnya adalah pesan yang diubah dalam *message digest* tidak mungkin untuk dikembalikan dari *message digest* menjadi pesan aslinya. Hingga saat ini, fungsi *hash* satu-arah sudah banyak dibuat orang. Contohnya adalah *MD2, MD4, MD5, SHA, Snefru, N-hash, RIPE-MD*. Sayangnya fungsi *hash* satu-arah seperti *MD4* memiliki kelemahan yaitu kolisi dan dapat diserang oleh kriptanalis.

Saya akan menganalisis dan membandingkan fungsi *hash* satu-arah yang sudah ada untuk mengetahui kelemahan dan keunggulan dari fungsi *hash* satu-arah tersebut. Dalam makalah ini, saya melakukan analisis dan perbandingan algoritma *MD4, MD5,* dan *SHA*. Kemudian, saya mencoba merancang fungsi *hash* satu-arah yang mungkin dapat menjawab kelemahan-kelemahan dari algoritma *MD4, MD5,* dan *SHA*. Saya masih akan memperhatikan keunggulan *MD4, MD5,* dan *SHA*, untuk merancang fungsi *hash* satu-arah tersebut. Saya mengharapkan fungsi *hash* satu-arah yang saya rancang dapat diaplikasikan pada kehidupan sehari-hari dan bidang keilmuan lainnya.

***Index Terms*—analisis, fungsi, *hash*, *hash* satu-arah, rancangan.**

I. PENDAHULUAN

Fungsi *hash* adalah fungsi yang menerima masukkan *string* yang panjangnya sembarang dan mengkonversinya menjadi *string* keluaran yang panjangnya tetap atau *fixed* [2]. Jika *string* menyatakan pesan atau *message*, maka sembarang *message M* berukuran bebas dikompresi oleh fungsi *hash H* dan menghasilkan nilai *hash* (*hash-value*).

Jika fungsi *hash* dituliskan dalam notasi persamaan, maka penulisannya adalah sebagai berikut:

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

Pada (1), *h* merupakan keluaran *hash* dari fungsi *H* untuk masukkan *M*. Keluaran fungsi *hash* disebut juga nilai *hash* (*hash-value*) atau pesan-ringkas (*message digest*) [2]. Fungsi *hash* dapat mengkompresi sembarang pesan yang berukuran berapa saja menjadi *message digest* yang ukurannya selalu tetap (dan lebih pendek dari panjang pesan semula).

Fungsi *hash* dapat diaplikasikan untuk kehidupan sehari-hari, biasanya fungsi *hash* digunakan untuk mendeteksi atau memverifikasi keaslian suatu salinan dokumen. Pendeteksian keaslian dapat dilakukan dengan cara melakukan *hashing* terhadap dokumen asli dan salinan dokumennya. Jika *message digest* dokumen asli berbeda dengan *message digest* salinan dokumen, maka salinan dokumen tersebut disimpulkan sudah mengalami perubahan.

Fungsi *hash* satu-arah (*one-way hash*) adalah fungsi *hash* yang bekerja dalam satu arah, artinya pesan yang diubah dalam *message digest* tidak dapat dikembalikan lagi menjadi pesan semula [2]. Hingga saat ini, fungsi *hash* satu-arah sudah banyak dibuat orang. Contoh fungsi *hash* satu-arah yang sudah dibuat adalah *MD2*, *MD4*, *MD5*, *SHA*, *Snefru*, *N-hash*, *RIPE-MD*. Sayangnya fungsi *hash* satu-arah seperti *MD4* memiliki kelemahan yaitu kolisi dan dapat diserang oleh kriptanalis.

Saya akan melakukan analisis dan perbandingan fungsi *hash* satu-arah (*one-way hash*) yang sudah ada seperti *MD4*, *MD5*, dan *SHA*. Tujuannya adalah mengetahui kelemahan dan keunggulan dari fungsi *hash* satu-arah tersebut. Kemudian, saya akan merancang dan membuat sebuah fungsi *hash* satu-arah (*one-way hash*). Rancangan fungsi *hash* satu arah tersebut mungkin menjawab kelemahan yang saya temukan dari algoritma *MD4*, *MD5*, dan *SHA*. Tetapi, saya masih memperhatikan keunggulan yang dimiliki *MD4*, *MD5*, dan *SHA*, untuk merancang fungsi *hash* satu-arah tersebut.

Semoga fungsi *hash* satu-arah yang saya rancang dapat dipakai dalam kehidupan sehari-hari dan bidang keilmuan lainnya. Saya merasa ilmu keinformatikaan memiliki

manfaat yang besar jika diterapkan dalam kehidupan sehari-hari. Oleh karena itu, ilmu keinformatikaan akan memiliki manfaat yang jauh lebih besar jika diterapkan dengan bidang keilmuannya lainnya.

II. DASAR TEORI

Fungsi *hash* adalah fungsi yang menerima masukkan *string* yang panjangnya sembarang dan mengkonversinya menjadi *string* keluaran yang panjangnya tetap atau *fixed* [2]. Jika *string* menyatakan pesan atau *message*, maka sembarang *message M* berukuran bebas dikompresi oleh fungsi *hash H* dan menghasilkan nilai *hash* (*hash-value*).

Pada (1), *h* merupakan keluaran *hash* dari fungsi *H* untuk masukkan *M*. Keluaran fungsi *hash* disebut juga nilai *hash* (*hash-value*) atau pesan-ringkas (*message digest*) [2]. Fungsi *hash* dapat mengkompresi sembarang pesan yang berukuran berapa saja menjadi *message digest* yang ukurannya selalu tetap (dan lebih pendek dari panjang pesan semula).

A. Fungsi Hash Satu-Arah (One-way Hash)

Fungsi *hash* satu-arah adalah fungsi *hash* yang bekerja dalam satu arah, artinya pesan yang diubah dalam *message digest* tidak dapat dikembalikan lagi menjadi pesan semula [2].

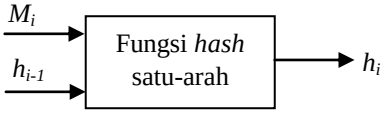
Sifat-sifat fungsi *hash* satu-arah adalah:

1. Fungsi *H* dapat diterapkan pada blok data berukuran berapa saja.
2. *H* menghasilkan nilai *hash h* dengan panjang tetap atau *fixed*
3. *H(x)* mudah dihitung untuk setiap nilai *x* yang diberikan
4. Untuk setiap *h* yang dihasilkan, tidak mungkin dikembalikan nilai *x* sedemikian sehingga *H(x) = h*. Itulah sebabnya fungsi *H* dikatakan sebagai fungsi *hash* satu-arah (*one-way hash*)
5. Untuk setiap *x* yang diberikan, tidak mungkin mencari $y \neq x$ sedemikan sehingga *H(y) = H(x)*
6. Tidak mungkin mencari pasangan *x* dan *y* sedemikian sehingga *H(y) = H(x)*

Masukkan fungsi *hash* adalah blok pesan (*M*) dan keluaran dari *hashing* blok pesan sebelumnya

$$h_i = H(M_i, h_{i-1}) \qquad (2)$$

Skema fungsi *hash* satu-arah adalah sebagai berikut:



Gambar 1. Fungsi *hash* satu-arah

Fungsi *hash* adalah publik (tidak dirahasiakan), dan keamanannya terletak pada sifat satu arah.

Ada beberapa fungsi *hash* satu-arah yang sudah pernah dibuat oleh orang [2], antara lain:

1. *MD2*, *MD4*, *MD5*,
2. *Secure Hash Function* (*SHA*),
3. *Snefru*,
4. *N-hash*,
5. *RIPE-MD*, dan lain-lain

III. ALGORITMA *MD4*

MD4 adalah fungsi *hash* satu-arah yang dibuat oleh Ronald Rivest [4]. *MD4* menerima masukkan berupa pesan atau *message* dengan ukuran sembarang dan menghasilkan *message digest* yang panjangnya 128 bit. MD4 digunakan juga dalam tandatangan digital (*Digital Signature Standard*) [3].

Langkah-langkah pembuatan *message digest* dengan algoritma *MD4* adalah sebagai berikut:

1. Penambahan bit-bit pengganjal (*padding bits*)

Pesan ditambah dengan sejumlah bit pengganjal (*padding bits*) sedemikian sehingga panjang pesan dalam satuan bit kongruen dengan 448 modulo 512. Pesan dengan panjang 448 bit harus tetap ditambah dengan bit-bit pengganjal. Jika panjang pesan 448 bit, maka pesan tersebut ditambah dengan 512 bit menjadi 960 bit. Jadi, panjang bit-bit pengganjal adalah antara 1 hingga 512. Bit-bit pengganjal (*padding bits*) terdiri dari sebuah bit 1 diikuti dengan sisanya bit 0.

2. Penambahan nilai panjang pesan semula

Pesan yang telah diberi bit-bit pengganjal atau *padding bits* selanjutnya ditambah lagi dengan 64 bit yang menyatakan panjang pesan semula. Jika panjang pesan lebih besar dari 2^{64}, maka panjang yang diambil adalah panjang pesan dalam modulo 2^{64}.

3. Inisialisasi penyangga (*buffer*) *MD*

MD4 membutuhkan 4 buah penyangga (*buffer*) yang masing-masing panjangnya 32 bit dan total panjang dari 4 buah penyangga adalah 128 bit. Keempat penyangga ini menampung hasil antara dan hasil akhir. Keempat penyangga ini diberi nama *A*, *B*, *C*, dan *D*. Setiap penyangga diinisialisasi dengan nilai-nilai dalam notasi HEX sebagai berikut:

A = 01234567
B = 89ABCDEF
C = FEDCAB98
D = 76543210

4. Pengolahan pesan dalam blok berukuran 512 bit

Pesan dibagi menjadi *L* buah blok yang masing-masing panjangnya 512 bit (Y_0 sampai Y_{L-1}) dan setiap blok

melewati proses H_{MD4} yaitu setiap blok 512-bit diproses bersama dengan penyangga *MD4* menjadi keluaran 128-bit.

Proses H_{MD4} terdiri dari 3 putaran dan masing-masing putaran melakukan operasi dasar *MD4* sebanyak 16 kali. Operasi dasar *MD4* berbeda-beda pada setiap putarannya. Perbedaanya adalah penambahan bilangan heksadesimal pada operasi dasar putaran kedua dan putaran ketiga.

Operasi dasar *MD4* dapat ditulis sebagai berikut (terurut mulai dari putaran 1, 2, dan 3):

$$a = a + CLS_s(a + g(b, c, d) + X[k]) \quad (3)$$

$$a = a + CLS_s(a + g(b, c, d) + X[k] + 5A827999) \quad (4)$$

$$a = a + CLS_s(a + g(b, c, d) + X[k] + 6ED9EBA1) \quad (5)$$

yang dalam hal ini,

a, *b*, *c*, *d* = empat buah peyangga 32-bit (berisi nilai peyangga *A*, *B*, *C*, dan *D*)

g = salah satu fungsi *F*, *G*, *H*

CLS_s = *circular left shift* sebanyak *s* bit

X[k] = kelompok 32-bit ke-*k* dari 512 bit *message* ke-*q*

Fungsi *F*, *G*, *H* merupakan fungsi yang dipakai pada setiap putaran. Fungsi *F*, *G*, dan *H* dituliskan sebagai berikut:

$$F(x, y, z) = (x \wedge y) \vee (\sim x \wedge z) \quad (6)$$

$$G(x, y, z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) \quad (7)$$

$$H(x, y, z) = x \oplus y \oplus z \quad (8)$$

Setelah putaran ketiga, *a*, *b*, *c*, dan *d* ditambahkan ke *A*, *B*, *C*, dan *D*, dan selanjutnya algoritma memproses untuk blok data selanjuntya (Y_{q+1}). Keluaran akhir dari algoritma *MD4* adalah hasil penyambungan dari bit-bit di *A*, *B*, *C*, dan *D*.

IV. ALGORITMA *MD5*

MD5 adalah fungsi *hash* satu-arah yang dibuat oleh Ronald Rivest [5]. Algoritma *MD5* merupakan perbaikan dari algoritma *MD4* setelah algoritma *MD4* diketahui memiliki kemungkinan terjadinya kolisi dan dapat diserang oleh kriptanalis. Algoritma *MD5* menerima masukkan berupa pesan atau *message* dengan ukuran sembarang dan menghasilkan *message digest* yang panjangnya 128 bit [2].

Langkah-langkah pembuatan *message digest* dengan algoritma *MD5* adalah sebagai berikut:

1. Penambahan bit-bit pengganjal (*padding bits*)

Pesan ditambah dengan sejumlah bit pengganjal (*padding bits*) sedemikian sehingga panjang pesan dalam satuan bit kongruen dengan 448 modulo 512. Pesan dengan panjang 448 bit harus tetap ditambah dengan bit-bit pengganjal. Jika panjang pesan 448 bit, maka pesan tersebut ditambah dengan 512 bit menjadi 960 bit. Jadi, panjang bit-bit pengganjal adalah antara 1 hingga 512. Bit-bit pengganjal (*padding bits*) terdiri dari sebuah bit 1 diikuti dengan sisanya bit 0.

2. Penambahan nilai panjang pesan semula

Pesan yang telah diberi bit-bit pengganjal atau *padding bits* selanjutnya ditambah lagi dengan 64 bit yang menyatakan panjang pesan semula. Jika panjang pesan lebih besar dari 2^{64}, maka panjang yang diambil adalah panjang pesan dalam modulo 2^{64}.

3. Inisialisasi penyangga (*buffer*) *MD*

MD5 membutuhkan 4 buah penyangga (*buffer*) yang masing-masing panjangnya 32 bit dan total panjang dari 4 buah penyangga adalah 128 bit. Keempat penyangga ini menampung hasil antara dan hasil akhir. Keempat penyangga ini diberi nama *A*, *B*, *C*, dan *D*. Setiap penyangga diinisialisasi dengan nilai-nilai dalam notasi HEX sebagai berikut:

A = 01234567
B = 89ABCDEF
C = FEDCAB98
D = 76543210

4. Pengolahan pesan dalam blok berukuran 512 bit

Pesan dibagi menjadi *L* buah blok yang masing-masing panjangnya 512 bit (Y_0 sampai Y_{L-1}) dan setiap blok melewati proses H_{MD5} yaitu setiap blok 512-bit diproses bersama dengan penyangga *MD5* menjadi keluaran 128-bit.

Proses H_{MD5} terdiri dari 4 putaran dan masing-masing putaran melakukan operasi dasar *MD5* sebanyak 16 kali dalam setiap operasi dasar memakai sebuah elemen *T*. Jadi setiap putaran memiliki 16 elemen Tabel *T*.

Operasi dasar *MD5* dapat ditulis sebagai berikut:

$$a = b + CLS_s(a + g(b, c, d) + X[k] + T[i]) \quad (9)$$

yang dalam hal ini,

a, *b*, *c*, *d* = empat buah peyangga 32-bit (berisi nilai peyangga *A*, *B*, *C*, dan *D*)

g = salah satu fungsi *F*, *G*, *H*, *I*

CLS_s = *circular left shift* sebanyak *s* bit

X[k] = kelompok 32-bit ke-*k* dari 512 bit *message* ke-*q*

T[i] = elemen Tabel *T* ke-*i* (32-bit)

Fungsi *F*, *G*, *H*, *I* merupakan fungsi untuk memanipulasi masukkan *a*, *b*, *c*, dan *d* dengan ukuran 32-bit. Fungsi *F*, *G*, *H*, dan *I* dituliskan sebagai berikut:

$$F(b, c, d) = (b \wedge c) \vee (\sim b \wedge d) \quad (10)$$

$$G(b, c, d) = (b \wedge d) \vee (c \wedge \sim d) \quad (11)$$

$$H(b, c, d) = b \oplus c \oplus d \quad (12)$$

$$I(b, c, d) = c \oplus (b \wedge \sim d) \quad (13)$$

Sedangkan nilai *T[i]* dibuat tetap dan ditulis dalam sebuah tabel. Tabel tersebut disusun dengan fungsi 2^{32} x *abs(sin(i))* (*i* dalam radian). Tetapi, saya tidak mencantumkan tabel nilai *T* pada makalah saya.

Setelah putaran keempat, *a*, *b*, *c*, dan *d* ditambahkan ke *A*, *B*, *C*, dan *D*, dan selanjutnya algoritma memproses untuk blok data selanjuntya (Y_{q+1}). Keluaran akhir dari algoritma *MD5* adalah hasil penyambungan dari bit-bit di *A*, *B*, *C*, dan *D*.

V. *SECURE HASH ALGORITHM* (*SHA*)

SHA adalah fungsi *hash* satu-arah yang dibuat oleh NIST dan digunakan bersama dengan DSS (*Digital Signature Standard*). *SHA* didasarkan pada *MD4* yang dibuat oleh Ronald L. Rivest [2]. *SHA* disebut aman atau *secure* karena *SHA* dirancang sedemikian rupa sehingga cara komputasi tidak mungkin menemukan pesan yang berkoresponden dengan *message digest* yang diberikan.

Algoritma *SHA* menerima masukan berupa pesan atau *message* dengan ukuran maksimum 2^{64} bit dan menghasilkan *message digest* yang panjangnya 160 bit. Hasil keluaran atau *message digest* dari SHA lebih panjang daripada *message digest* yang dihasilkan dengan *MD5*.

Langkah-langkah pembuatan *message digest* dengan algoritma *SHA* adalah sebagai berikut:

1. Penambahan bit-bit pengganjal (*padding bits*)

Pesan ditambah dengan sejumlah bit pengganjal (*padding bits*) sedemikian sehingga panjang pesan dalam satuan bit kongruen dengan 448 modulo 512. Pesan dengan panjang 448 bit harus tetap ditambah dengan bit-bit pengganjal. Jika panjang pesan 448 bit, maka pesan tersebut ditambah dengan 512 bit menjadi 960 bit. Jadi, panjang bit-bit pengganjal adalah antara 1 hingga 512. Bit-bit pengganjal (*padding bits*) terdiri dari sebuah bit 1 diikuti dengan sisanya bit 0.

2. Penambahan nilai panjang pesan semula

Pesan yang telah diberi bit-bit pengganjal atau *padding bits* selanjutnya ditambah lagi dengan 64 bit yang menyatakan panjang pesan semula. Ingat ukuran maksimum pesan adalah 2^{64} bit.

3. Inisialisasi penyangga (*buffer*) *MD*

SHA membutuhkan 5 buah penyangga (*buffer*) yang masing-masing panjangnya 32 bit dan total panjang dari 5 buah penyangga adalah 160 bit. Kelima penyangga ini menampung hasil antara dan hasil akhir. Kelima penyangga ini diberi nama *A*, *B*, *C*, *D*, dan *E*. Setiap penyangga diinisialisasi dengan nilai-nilai dalam notasi HEX sebagai berikut:

A = 67452301
B = EFCDAB89
C = 98BADCFE
D = 10325476
E = C3D2E1F0

4. Pengolahan pesan dalam blok berukuran 512 bit

Pesan dibagi menjadi *L* buah blok yang masing-masing panjangnya 512 bit (Y_0 sampai Y_{L-1}) dan setiap blok melewati proses H_{SHA} yaitu setiap blok 512-bit diproses bersama dengan penyangga *SHA* menjadi keluaran 160-bit.

Proses H_{SHA} terdiri dari 80 putaran dan masing-masing putaran menggunakan bilangan penambah K_n, yaitu:

Untuk $0 \le n \le 19$, K_n = 5A827999
Untuk $20 \le n \le 39$, K_n = 6ED9EBA1
Untuk $40 \le n \le 59$, K_n = 8F1BBCDC
Untuk $60 \le n \le 79$, K_n = CA62C1D6

Setiap putaran menggunakan operasi dasar yang sama (dinyatakan sebagai fungsi *f*).

Operasi dasar *SHA* dapat ditulis sebagai berikut:

$$\begin{aligned} a &= (CLS_5(a) + f_n(b, c, d) + e + W_n + K_n) \\ b &= a \\ c &= CLS_{30}(b) \\ d &= c \\ e &= d \end{aligned} \quad (14)$$

yang dalam hal ini,

a, *b*, *c*, *d*, *e* = lima buah peyangga 32-bit (berisi nilai peyangga *A*, *B*, *C*, *D*, dan *E*)
n = putaran, $0 \le n \le 79$
f_n = fungsi logika
CLS_s = *circular left shift* sebanyak *s* bit
W_n = *word* 32-bit ke-*k* dari 512 bit *message* ke-*q*
K_n = bilangan penambah

Fungsi *f* yang digunakan pada operasi dasar berbeda pada putaran tertentu. Fungsi *f* dapat dituliskan sebagai berikut:

$$\text{Untuk } 0 \le n \le 19,\ f_n = (b \wedge c) \vee (\sim b \wedge d) \quad (15)$$

$$\text{Untuk } 20 \le n \le 39,\ f_n = b \oplus c \oplus d \quad (16)$$

$$\text{Untuk } 40 \le n \le 59,\ f_n = (b \wedge c) \vee (b \wedge d) \vee (c \wedge d) \quad (17)$$

$$\text{Untuk } 60 \le n \le 79,\ f_n = b \oplus c \oplus d \quad (18)$$

Sedangkan nilai *W* yang terdiri dari W_1 sampai W_{16} diinisialisasi dari 16 *word* pada blok yang sedang

diproses, sedangkan nilai W_n pada putaran berikutnya diperoleh dengan persamaan:

$$W_n = W_{n-16} \oplus W_{n-14} \oplus W_{n-8} \oplus W_{n-3} \quad (19)$$

Setelah putaran ke-79, *a*, *b*, *c*, *d*, dan *e* ditambahkan ke *A*, *B*, *C*, *D*, dan *E* dan selanjutnya algoritma memproses untuk blok data berikutnya (Y_{q+1}). Keluaran akhir dari algoritma *SHA* adalah hasil penyambungan bit-bit di *A*, *B*, *C*, *D*, dan *E*.

VI. ANALISIS *MD4*, *MD5*, DAN *SHA*

Saya melakukan analisis terhadap fungsi *hash* satu-arah yang sudah saya pilih sebelumnya. Fungsi *hash* yang saya analisis adalah *MD4*, *MD5*, dan *SHA*. Saya melakukan analisis terhadap masing-masing fungsi *hash* dalam hal algoritma yang dipakai dan kemungkinan terjadinya kolisi pada fungsi *hash* tersebut. Analisis yang saya lakukan akan saya deskripsikan dalam bentuk keunggulan dan kelemahan fungsi *hash* satu-arah yang bersangkutan.

A. Analisis Algoritma MD4

Keunggulan algoritma *MD4* adalah:

1. Algoritma *MD4* didesain sedemikian rupa sehingga lebih cepat dikomputasi pada mesin 32-bit dan mudah untuk diimplementasikan [3]
2. *MD4* tidak memerlukan tabel substitusi.
   MD4 berbeda dengan *MD5* yang memerlukan tabel substitusi. Tabel substitusi pada MD5 diperlukan untuk menyimpan nilai *T* yang digunakan pada setiap putarannya.
3. Komputasi *MD4* sangat cepat karena *MD4* hanya terdiri dari 3 putaran dan setiap putaran dilakukan 16 kali.
   Untuk kasus *hashing* terhadap satu blok pesan berukuran 512 bit, total jumlah perulangan yang dilakukan adalah 48 kali.
4. Panjang pesan yang dapat di-*hash* lebih dari 2^{64} bit. Sedangkan *SHA* hanya dapat memproses pesan dengan panjang maksimal 2^{64} bit.

Kelemahan algoritma *MD4* adalah:

1. Algoritma *MD4* sudah dibuktikan memiliki kemungkinan terjadinya kolisi [1]. Kolisi yang dimaksudkan adalah kolisi *hash* (*hash colision*). Suatu fungsi hash H dikatakan memiliki hash colision apabila terdapat message x dan message y, $x \neq y$ dan $H(x) = H(y)$.
2. Algoritma *MD4* dapat diserang oleh kriptanalis.

B. Analisis Algoritma MD5

Keunggulan algoritma *MD5* adalah:

1. Panjang pesan yang dapat di-*hash* lebih dari 2^{64} bit. Sedangkan *SHA* hanya dapat memproses pesan dengan panjang maksimal 2^{64} bit.
2. Algoritma *MD5* lebih menjamin untuk tidak terjadinya kolisi *hash*. Karena *MD5* menggunakan tabel substitusi *T* yang digunakan setiap putarannya.
3. Tingkat keamanan algoritma *MD5* masih tinggi. Hal tersebut dapat didukung karena MD5 memiliki tabel substitusi *T* untuk digunakan dalam operasi dasar pada setiap putarannya.

Kelemahan algoritma *MD5* adalah:

1. Komputasi *MD5* memakan waktu yang lebih lama karena *MD5* memproses setiap blok pesan berukuran 512 bit dalam 4 putaran dan setiap putarannya diulangi selama 16 kali.

C. Analisis Algoritma SHA

Keunggulan algoritma *SHA* adalah:

1. Panjang nilai *hash* (*hash-value*) atau *message digest* yang dikeluarkan berukuran 160 bit. Menurut saya, semakin panjang *message digest* yang dikeluarkan semakin sulit terjadinya kolisi dari suatu fungsi *hash* tersebut.
2. Algoritma *SHA* diklaim aman atau *secure*. Keamanan algoritma *SHA* dapat disebabkan karena proses komputasi dalam algoritma *SHA* rumit dan berbeda-beda untuk setiap putarannya.

Kelemahan algoritma *SHA* adalah:

1. Panjang pesan yang dapat di-*hash* maksimal berukuran 2^{64} bit. Sedangkan *MD4* dan *MD5* dapat memproses pesan dengan panjang berapapun.
2. Kecepatan algoritma *SHA* lebih lambat daripada kecepatan komputasi algoritma *MD4*.

V. RANCANGAN FUNGSI *HASH*

Setelah saya melakukan analisis dan perbandingan antara fungsi *hash MD4*, *MD5*, dan *SHA*. Saya menarik beberapa kesimpulan sementara yaitu:

1. Kompleksitas atau tingkat kerumitan suatu algoritma pada fungsi *hash* sebanding atau berbanding lurus dengan tingkat keamanan fungsi *hash* tersebut.
2. Kompleksitas atau tingkat kerumitan suatu algoritma pada fungsi *hash* berbanding terbalik dengan kecepatan dari algoritma fungsi *hash* tersebut.
3. Untuk meningkatkan keamanan suatu fungsi *hash*, kita dapat membuat sebuah tabel substitusi *T* dengan nilai yang berbeda-beda mulai dari 1 hingga *t* (*t* adalah banyaknya operasi dasar pada fungsi *hash* tersebut). Setiap nilai dari tabel T dipakai dalam komputasi pada fungsi *hash* tersebut.
4. Cara lain untuk meningkatkan keamanan suatu fungsi *hash* adalah memperpanjang *message digest* atau *hash-value* fungsi *hash* tersebut. Semakin panjang ukuran *messsage digest*, semakin sulit terjadinya *hash colision* dan diserang oleh

kriptanalis.

Fungsi *hash* yang saya rancang akan memiliki kemiripan dengan algoritma *MD5*, karena saya akan menggunakan sebuah tabel substitusi *T* untuk menyimpan nilai-nilai yang saya gunakan pada setiap operasi dasar rancangan fungsi *hash*.

Fungsi *hash* yang saya rancang dapat menerima masukkan pesan atau *message* dengan panjang berapapun dan akan mengeluarkan nilai *hash* atau *message digest* dengan panjang 128 bit.

Langkah-langkah pembuatan *message digest* dari rancangan fungsi *hash* saya adalah sebagai berikut:

1. Penambahan bit-bit pengganjal (*padding bits*)

Pesan ditambah dengan sejumlah bit pengganjal (*padding bits*) sedemikian sehingga panjang pesan dalam satuan bit kongruen dengan 448 modulo 512. Pesan dengan panjang 448 bit harus tetap ditambah dengan bit-bit pengganjal. Jika panjang pesan 448 bit, maka pesan tersebut ditambah dengan 512 bit menjadi 960 bit. Jadi, panjang bit-bit pengganjal adalah antara 1 hingga 512. Bit-bit pengganjal (*padding bits*) terdiri dari sebuah bit 1 diikuti dengan sisanya bit 0.

Langkah penambahan bit-bit pengganjal mirip dengan langkah penambah bit-bit pengganjal pada algoritma *MD4*, *MD5*, dan *SHA*.

2. Penambahan nilai panjang pesan semula

Pesan yang telah diberi bit-bit pengganjal atau *padding bits* selanjutnya ditambah lagi dengan 64 bit yang menyatakan panjang pesan semula. Jika panjang pesan lebih besar dari 2^{64}, maka panjang yang diambil adalah panjang pesan dalam modulo 2^{64}.

Langkah penambahan nilai panjang pesan semula mirip dengan langkah penambahan nilai panjang pesan semula pada algoritma *MD4*, dan *MD5*.

3. Inisialisasi penyangga (*buffer*) *MD*

Rancangan fungsi *hash* saya membutuhkan 4 buah penyangga (*buffer*) yang masing-masing panjangnya 32 bit dan total panjang dari 4 buah penyangga adalah 128 bit. Keempat penyangga ini menampung hasil antara dan hasil akhir. Keempat penyangga ini diberi nama *A*, *B*, *C*, dan *D*. Setiap penyangga diinisialisasi dengan nilai-nilai dalam notasi HEX sebagai berikut:

A = AFD82557
B = 990146EA
C = 8276ADC3
D = A304BF1E

Nilai *A*, *B*, *C*, dan *D* dipilih secara acak dan tidak ada kaitannya dengan algoritma *MD4*, *MD5*, dan *SHA*. Jumlah penyangga pada rancangan fungsi *hash* saya sama dengan jumlah peyangga pada algoritma *MD4* dan *MD5*.

4. Pengolahan pesan dalam blok berukuran 512 bit

Pesan dibagi menjadi *L* buah blok yang masing-masing panjangnya 512 bit (Y_0 sampai Y_{L-1}) dan setiap blok 512-bit diproses bersama dengan penyangga *A*, *B*, *C*, dan *D* menjadi keluaran 128-bit.

Proses tersebut terdiri dari 10 putaran dan masing-masing putaran melakukan operasi dasar yang saya rancang sendiri. Setiap putaran akan diulang sebanyak 10 kali dan pada setiap putarannya digunakan sebuah tabel *P* yang berisi nilai-nilai yang digunakan untuk penambahan pada operasi dasar.

Operasi dasar yang saya rancang dapat ditulis sebagai berikut:

$$\begin{aligned} a &= a + b + c + d \\ b &= CLS_{10}(P[i] + b + f_n(a, b, c)) \\ c &= c + d + P[i] \\ d &= f_n(b, c, d) \end{aligned} \qquad (20)$$

yang dalam hal ini,

a, *b*, *c*, *d* = empat buah peyangga 32-bit (berisi nilai peyangga *A*, *B*, *C*, dan *D*)

n = putaran, $1 \le n \le 10$

f_n = fungsi logika

CLS_s = *circular left shift* sebanyak *s* bit

P[i] = elemen Tabel *P* ke-*i* (32-bit)

Fungsi *f* yang digunakan pada operasi dasar berbeda pada putaran tertentu. Fungsi *f* dapat dituliskan sebagai berikut:

$$\text{Untuk } 1 \le n \le 3,\ f_n = (x \wedge y)\ v\ (\sim x \wedge z) \qquad (21)$$

$$\text{Untuk } 4 \le n \le 6,\ f_n = x \oplus y \oplus z \qquad (22)$$

$$\text{Untuk } 7 \le n \le 9,\ f_n = (\sim x \wedge \sim y)\ v\ (x \wedge \sim z) \qquad (23)$$

$$\text{Untuk } n = 10,\ f_n = (x \wedge z)\ v\ (x \wedge y)\ v\ (z \wedge y) \qquad (24)$$

Fungsi *f* yang saya rancang memiliki kemiripan seperti fungsi *f* pada *SHA*. Kemiripannya adalah fungsi *f* yang saya rancang dan fungsi *y* pada *SHA* berubah atau berbeda setiap putaran.

Sedangkan nilai *P[i]* dibuat tetap dan ditulis dalam sebuah tabel. Tabel tersebut disusun dengan fungsi 2^{32} x *abs(sin(i))* (*i* dalam radian). Tetapi, saya tidak mencantumkan tabel nilai *P* pada makalah saya. Tabel *P* untuk rancangan fungsi *hash* saya sama dengan tabel *T* pada algoritma *MD5*. Saya menggunakan tabel *T* dari algoritma *MD5* sebagai tabel *P* rancangan fungsi *hash* saya karena saya merasa tabel subtitusi *T* pada algoritma *MD5* masih dapat menjaga keamanan dari fungsi *hash* *MD5*. Perbedaan tabel *T* dari algoritma *MD5* dan tabel *P* untuk rancangan fungsi *hash* saya adalah tabel *T* hanya memiliki nilai dari 1 hingga 64, sedangkan tabel *P* memiliki nilai dari 1 hingga 80.

Setelah putaran kesepuluh (n = 10), a, b, c, dan d ditambahkan ke A, B, C, dan D, dan selanjutnya algoritma memproses untuk blok data selanjuntya (Y_{q+1}). Keluaran akhir dari fungsi *hash* rancangan saya adalah hasil penyambungan dari bit-bit di A, B, C, dan D.

Rancangan fungsi *hash* saya buat masih belum saya buktikan apakah dapat mengalami *hash colision* atau dapat diserang oleh kriptanalis. Saya pun belum sempat mengimplementasikan rancangan fungsi *hash* saya dalam sebuah program.

Saya mengharapkan ada orang yang dapat mencoba mengimplementasikan fungsi *hash* yang saya buat dan menganalisis fungsi *hash* yang saya buat apakah dapat mengalami *hash colision* atau dapat diserang oleh kriptanalis. Semoga fungsi *hash* yang saya rancang dapat digunakan pada kehidupan sehari-hari dan bidang keilmuan lainnya.

VI. KESIMPULAN

Kesimpulan yang saya dapatkan dari makalah ini adalah:

1. Kompleksitas atau tingkat kerumitan suatu algoritma pada fungsi *hash* sebanding atau berbanding lurus dengan tingkat keamanan fungsi *hash* tersebut.
2. Kompleksitas atau tingkat kerumitan suatu algoritma pada fungsi *hash* berbanding terbalik dengan kecepatan dari algoritma fungsi *hash* tersebut.
3. Untuk meningkatkan keamanan suatu fungsi *hash*, kita dapat membuat sebuah tabel substitusi T dengan nilai yang berbeda-beda mulai dari 1 hingga t (t adalah banyaknya operasi dasar pada fungsi *hash* tersebut). Setiap nilai dari tabel T dipakai dalam komputasi pada fungsi *hash* tersebut.
4. Cara lain untuk meningkatkan keamanan suatu fungsi *hash* adalah memperpanjang *message digest* atau *hash-value* fungsi *hash* tersebut. Semakin panjang ukuran *messsage digest*, semakin sulit terjadinya *hash colision* dan diserang oleh kriptanalis.

REFERENCES

[1] Guo, Jian, San Ling, Christian Rechberger, Huaxiong Wang, "Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2", Dept. of Electrical Engineering ESAT/COSIC, 2010.

[2] Munir, Rinaldi, "Diktat Kuliah IF5054 Kriptografi", Program Studi Teknik Informatika, 2005.

[3] *RFC 1320 - The MD4 Message-Digest Algorithm*. http://tools.ietf.org/html/rfc1320. Diakses pada 16 Mei 2010.

[4] *RFC 1320 (rfc1320) - The MD4 Message-Digest Algorithm*. http://www.faqs.org/rfcs/rfc1320.html. Diakses pada 16 Mei 2010.

[5] *RFC 1321 - The MD5 Message-Digest Algorithm*. http://tools.ietf.org/html/rfc1321. Diakses pada 16 Mei 2010.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 17 Mei 2010

Rianto Fendy Kristanto
13507036