

Technical Report

FUNGSI HASH : TIGER

Sebuah Kajian Spesifikasi dan Keamanan

TUGAS EC7010
Keamanan Sistem Lanjut

Oleh :

ROSWAN LATUCONSINA
NIM : 23205340



INSTITUT TEKNOLOGI BANDUNG
2006

ABSTRAK

Fungsi *Hash* adalah suatu cara menciptakan "*fingerprint*" dari berbagai data masukan. Fungsi *Hash* akan mengganti atau mentranspose-kan data tersebut untuk menciptakan fingerprint, yang biasa disebut *hash value*. *Hash value* biasanya digambarkan sebagai suatu string pendek yang terdiri atas huruf dan angka yang terlihat random (data biner yang ditulis dalam notasi heksadesimal). Algoritma fungsi *hash* yang baik adalah yang menghasilkan sedikit *hash collision*. Sudah banyak algoritma fungsi *hash* yang diciptakan, namun fungsi *hash* yang umum digunakan saat ini adalah MD5 dan SHA (Secure Hash Algorithm). Kedua algoritma tersebut didesain untuk prosesor 32 bit, dan tidak dapat diimplementasikan untuk prosesor 64 bit.

Melalui serangkaian penelitian telah ditemukan adanya *collision* pada algoritma MD5 dan SHA (SHA-0 dan SHA-1). Untuk itu diciptakanlah algoritma Tiger yang dapat diimplementasikan untuk prosesor 32 bit dan 64 bit, dan diklaim oleh penciptanya (Eli Biham dan Ross Anderson) sebagai "*fast and secure hash function*".

Dalam *technical report* ini penulis akan mencoba mengkaji spesifikasi komputasi serta keamanan (*security*) algoritma Tiger.

DAFTAR ISI

	Halaman
ABSTRAK	i
DAFTAR ISI	ii
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Fungsi Hash	1
1.3 Tanda Tangan Digital	3
BAB II SERANGAN TERHADAP FUNGSI HASH	6
2.1 Jenis-Jenis Serangan	6
2.2 Fungsi hash dan Digital Signature	7
2.3 Kelemahan-kelemahan Fungsi Hash	7
2.4 Serangan Analitik	8
2.5 Eksploit	9
BAB III FUNGSI HASH TIGER	11
1.1 Spesifikasi Fungsi Hash Tiger	11
1.2 Keamanan (Security)	14
BAB IV PENUTUP	16
DAFTAR PUSTAKA	17

BAB I

PENDAHULUAN

1.1 Latar Belakang

Saat ini banyak data-data bisnis yang penting dipertukarkan menggunakan media internet, yang mana media ini bukanlah merupakan media yang aman. Pesan-pesan e-mail dan data dapat saja di baca, diubah, dirusak jika tidak diamankan dengan tepat. e-mail menyediakan metode komunikasi yang mudah, tidak mahal, serta paling reliable di seluruh dunia. Pesan-pesan yang dikirimkan menggunakan e-mail dapat saja berisikan data plaintext (mudah dibaca), sisipan gambar-gambar, file-file suara, dan elemen-elemen lainnya. Pesan-pesan e-mail ini dapat saja dengan mudah di baca, di rubah oleh user yang tidak berhak jika tidak terdapat mekanisme keamanan di dalamnya. Keamanan terhadap e-mail sangat penting, sebab pesan-pesan e-mail harus melewati beberapa server dahulu sebelum akhirnya di terima oleh penerima e-mail tersebut. Dalam keamanan e-mail yang harus di pertimbangkan adalah:

- Kerahasiaan, dengan keamanan e-mail harus dapat menjamin kerahasiaan dari pesan-pesan e-mail tersebut serta kerahasiaan datanya.
- Integritas, keamanan e-mail juga harus memastikan integritas pesan-pesan e-mail harus terjaga dengan menggunakan algoritma hashing yang menghasilkan pesan-pesan yang di enkripsi.
- Autentikasi, dalam lingkungan yang aman pesan-pesan e-mail di enkripsi menggunakan kunci rahasia yang hanya di ketahui oleh pengirim dan penerima e-mail.
- Nonrepudiation, merupakan proses mem-verifikasi keabsahan dari pengirim dan penerima pesan. Nonrepudiation dapat di implementasikan dengan menyertakan tanda tangan digital yang unik pada pesan e-mail yang di kirimkan. Tanda tangan digital memastikan bahwa yang mengirimkan pesan adalah pengirim yang sah.

1.2 Fungsi Hash

Hash function atau fungsi hash adalah suatu cara menciptakan “*fingerprint*” dari berbagai data masukan. *Hash function* akan mengganti atau

mentranspose-kan data tersebut untuk menciptakan *fingerprint*, yang biasa disebut *hash value*. *Hash value* biasanya digambarkan sebagai suatu string pendek yang terdiri atas huruf dan angka yang terlihat random (data biner yang ditulis dalam notasi heksadesimal).

Suatu *hash function* adalah sebuah fungsi matematika, yang mengambil sebuah panjang variabel string input, yang disebut *pre-image* dan mengkonversikannya ke sebuah string output dengan panjang yang tetap dan biasanya lebih kecil, yang disebut *message digests*.

Hash function digunakan untuk melakukan *fingerprint* pada *pre-image*, yaitu menghasilkan sebuah nilai yang dapat menandai (mewakili) *pre-image* sesungguhnya.

Fungsi *hash* satu arah (*one-way hash function*) adalah *hash function* yang bekerja satu arah, yaitu suatu *hash function* yang dengan mudah dapat menghitung *hash value* dari *pre-image*, tetapi sangat sukar untuk menghitung *pre-image* dari *hash value*.

Sebuah fungsi *hash* satu arah, $H(M)$, beroperasi pada suatu *pre-image* pesan M dengan panjang sembarang, dan mengembalikan nilai *hash* h yang memiliki panjang tetap. Dalam notasi matematika fungsi *hash* satu arah dapat ditulis sebagai:

$$h = H(M), \text{ dengan } h \text{ memiliki panjang } b$$

Ada banyak fungsi yang mampu menerima input dengan panjang sembarang dan menghasilkan output dengan panjang tetap, tetapi fungsi *hash* satu arah memiliki karakteristik tambahan yang membuatnya satu arah :

Diberikan M , mudah menghitung h .

Diberikan h , sulit menghitung M agar $H(M) = h$.

Diberikan M , sulit menemukan pesan lain, M' , agar $H(M) = H(M')$.

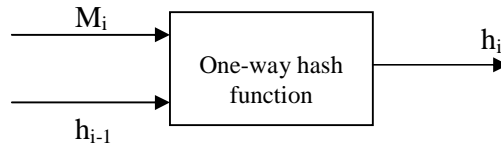
Dalam dunia nyata, fungsi *hash* satu arah dikembangkan berdasarkan ide sebuah fungsi kompresi. Fungsi satu arah ini menghasilkan nilai *hash* berukuran n bila diberikan input berukuran b . Input untuk fungsi kompresi adalah suatu blok pesan dan hasil blok teks sebelumnya. Sehingga *hash* suatu blok M , adalah

$$h_i = f(M_i, h_{i-1})$$

dengan

$$h_i = \text{hash value saat ini.}$$

M_i = blok pesan saat ini.
 h_{i-1} = *hash value* blok teks sebelumnya.



Gbr.1.1 Fungsi hash satu arah

Fungsi hash sangat berguna untuk menjaga integritas sebuah data. Sudah banyak algoritma *hash function* yang diciptakan, namun *hash function* yang umum digunakan saat ini adalah MD5 dan SHA (Secure Hash Algorithm). Algoritma *hash function* yang baik adalah yang menghasilkan sedikit *hash collision*.

1.3 Tanda Tangan Digital (Digital Signature)

Tanda tangan digital adalah sebuah tanda tangan elektronik yang dapat digunakan untuk menyediakan mekanisme autentikasi dan memastikan integritas. Tidak seperti tanda tangan konvensional (tulisan tangan), jenis tanda tangan digital tergantung pada data yang akan ditandai (tanda tangani). Tanda tangan digital dapat digunakan oleh penerima dokumen untuk verifikasi integritas data dan identitas pengirim dokumen.

Biasanya tanda tangan digital dibuat menggunakan sepasang kunci dari teknik enkripsi Asimetris dan algoritma hash, contohnya MD5 (Message Digest 5). Nilai hash-nya kemudian dihitung berdasarkan pada dokumen atau pesan yang akan ditandai (tanda tangani). Penggunaan tanda tangan digital biasanya melibatkan 2 proses, yaitu: proses pembuatan tanda tangan digital dan proses verifikasi tanda tangan digital.

€ Proses pembuatan tanda tangan digital

Untuk menandatangani sebuah dokumen atau pesan, penanda tangan (the signer) pertama kali harus menentukan secara pasti apa yang akan ditandai tangani. Fungsi hash diimplementasikan pada pesan untuk menghasilkan

sebuah keluaran yang unik yang disebut nilai hash (hash value). Hal ini dapat dilakukan dengan menggunakan algoritma hash seperti SHA (Secure Hash Algorithm) atau MD5 (Message Digest 5). Kunci privat penanda tangan (the signer), berasal dari sepasang kunci publik dan kunci privat, kemudian digunakan untuk mentransformasikan nilai hash menjadi tanda tangan digital yang unik untuk pesan tersebut dan kunci privat si penanda tangan (the signer). Setelah itu tanda tangan digital disertakan dikirim bersamaan dalam pesan yang akan dikirim atau dapat juga dikirim secara terpisah dari pesan yang akan dikirim sebagai elemen terpisah.

€ **Proses verifikasi tanda tangan digital**

Ketika pesan yang telah ditandai dengan tanda tangan digital telah diterima oleh si Penerima, maka pesan tersebut harus diverifikasi untuk memastikan apakah pesan tersebut berasal dari pengirim yang sudah dikenal dan apakah pesan tersebut belum diubah isinya tanpa seizin pemilik dokumen. Fungsi hash yang sama yang diaplikasikan oleh si penanda tangan (the signer) pada pesan yang dikirim juga diaplikasikan oleh si Penerima pesan tersebut. Nilai hash yang baru kemudian dibandingkan dengan nilai hash yang digenerate oleh si penanda tangan. Jika nilai hash tersebut sama nilainya, maka dapat dipastikan bahwa pesan tersebut belum pernah dilakukan perubahan selama proses pengiriman pesan dan integritas datanya dapat dipastikan. Nilai hash kemudian digunakan untuk menghubungkan dengan kunci publik milik penanda tangan (the signer) untuk mem-verifikasi apakah kunci privat si Penanda tangan yang digunakan untuk menghasilkan tanda tangan digital tersebut.

Tanda tangan digital dapat digunakan untuk menjamin integritas data pada sebuah pesan atau dokumen namun tidak dapat menjamin kerahasiaannya. Data yang dienkripsi oleh tanda tangan digital hanya nilai hash yang terdapat pada pesan tersebut, bukan enkripsi pada isi pesannya. Sangat dimungkinkan untuk mengkombinasikan tanda tangan digital sebuah pesan yang ter-enkripsi untuk menjamin agar integritas dan kerahasiaan data terjamin.

Autentikasi adalah sebuah proses verifikasi bahwa pesan tersebut benar-benar dikirimkan oleh user yang dikenal oleh si penerima pesan. Dalam sistem kunci rahasia (secret key system), dapat dikatakan bahwa pesan yang dikirimkan dikirim oleh seseorang yang mengetahui kunci rahasianya. Pada sistem kunci publik dan tanda tangan digital, dapat diketahui bahwa pemilik dari kunci privat yang digunakan untuk membuat tanda tangan digital adalah benar-benar si pengirim pesan tersebut.

BAB II

SERANGAN TERHADAP FUNGSI HASH

2.1 Jenis-Jenis Serangan

Kriptografi fungsi hash untuk autentikasi dan integritas data, satu persatu telah berhasil dipecahkan. Terdapat 3 jenis serangan terhadap fungsi hash, yaitu serangan yang memanfaatkan :

- kelemahan struktural
- kelemahan analitik
- eksploitasi keduanya

Kriptografi fungsi hash $H: \{0,1\}^* \rightarrow \{0,1\}^n$ memetakan set input infinit ke sekumpulan terbatas n-bit *hash value*. Jelasnya, sebuah fungsi hash H sebaiknya berkelakuan ideal (seperti sebuah *oracle random*). Hal ini tidak akan berguna untuk sebuah definisi formal. Sebagai gantinya adalah tujuan keamanan yang lebih sederhana untuk $H: \{0,1\}^* \rightarrow \{0,1\}^n$.

Sementara *collision* (input $X \neq Y$ dengan $H(X) = H(Y)$) perlu ada, sebagaimana terdapat lebih banyak input daripada output, sebuah fungsi hash sebaiknya memiliki ketahanan terhadap *collision*; diberikan H , sebaiknya infeasible terhadap musuh untuk menemukan setiap *collision*.

Sebagai tambahan, suatu fungsi hash harus tahan terhadap serangan preimage dan 2nd preimage; Diberikan Z yang sebaiknya infeasible terhadap musuh untuk menemukan setiap X dengan $H(X) = Z$, dan diberikan Y yang sebaiknya infeasible untuk menemukan Y dengan $H(X) = H(Y)$.

Secara formal, serangan (*attack*) dapat dibedakan sebagai berikut :

1. *Collision attack* : temukan 2 buah message $M \neq M'$ dengan $H(M) = H(M')$.
2. *Preimage attack* : diberikan sebuah nilai acak $Y \in \{0,1\}^n$, temukan sebuah message M dengan $H(M) = Y$.
3. *2nd preimage attack* : diberikan sebuah message M , temukan sebuah message $M' \neq M$ dengan $H(M) = H(M')$.
4. *K-collision attack* untuk $K \geq 2$: temukan K message M^i berbeda, dengan $H(M^1) = \dots = H(M^K)$.

5. *K-way (2nd) preimage attack* untuk $K \geq 1$: diberikan Y (atau M dengan $H(M) = Y$), temukan K message M^i berbeda, dengan $H(M^i) = Y$ (dan $M^i \neq M$).

2.2 Fungsi Hash dan Digital Signature

Mengapa sangat penting bagi suatu fungsi hash untuk resisten terhadap *2nd preimage* ataupun *collision*? Mengapa sangat berbahaya jika sebuah fungsi hash tidak dapat bertahan terhadap serangan *2nd preimage* atau *collision* ?

Sebagai contoh pada skema *digital signature*. Sebuah message M harus diberi tanda. Skema *signature* dipecahkan, jika musuh dapat memecahkan sebuah *signature* untuk message M' lainnya. Pada prakteknya, kriptografi skema *signature* mengikuti paradigma *hash and sign*. Sebagai ganti penandaan message M (berpotensi sangat panjang), menghitung suatu “fingerprint” $H(M)$ yang pendek dan menandai X .

Diketahui sebuah *signature* untuk beberapa message M , dan musuh dapat menghitung *2nd preimage* M' dengan $H(M) = H(M')$ dapat forge sebuah *signature* untuk M' , sebagai *signature* M adalah valid untuk M' .

Begitu banyak tentang pentingnya ketahanan *2nd preimage*. Tetapi mengapa ketahanan *collision* merupakan hal yang besar?

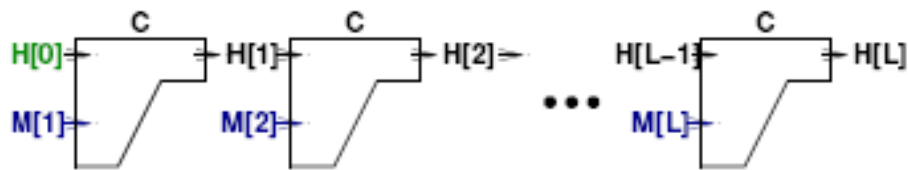
Ketahanan *2nd preimage* adalah baik jika kita senantiasa percaya bagian yang memilih message M untuk ditandai. Tetapi jika kita tidak percaya, kita membutuhkan ketahanan *collision*. Sebaliknya, musuh dapat menghitung *collision* $M \neq M'$, menandai M atau M' yang baru untuk ditandai, dan kemudian mengklaim *signature* M' .

2.3 Kelemahan-Kelemahan Fungsi Hash

Desain kriptografi fungsi hash yang ada saat ini, termasuk MD4, MD5, SHA-0, SHA-1, keluarga SHA-2, RIPEMD, Whirlpool, Tiger, mengikuti struktur MD (Merkle and Damgård).

Karena fungsi hash $H : \{0,1\}^* \rightarrow \{0,1\}^n$ harus menangani input-input yang panjang, maka dirancanglah sebuah fungsi kompresi internal $C : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$ dengan ukuran input yang tetap, untuk memecah message M kedalam m bit blok $M[1], \dots, M[L]$, dan untuk menghitung hash secara

straightforward menggunakan iterasi kompresi C seperti yang ditunjukkan pada gambar berikut :



Gbr. 2.1 Iterasi struktur Merkle-Damgård untuk fungsi hash

Fungsi hash h akan resisten terhadap *collision* jika fungsi kompresinya resisten terhadap *collision*. Ini menjadi keunggulan dari struktur MD. Sayangnya terdapat pula kelemahan struktur MD, yaitu *failure-unfriendly*, yang menyebabkan fungsi hash gagal dan tidak seperti yang diharapkan.

2.4 Serangan Analitik

Rancangan kriptografi operasi-operasi primitif seperti fungsi hash, bergantung pada seni dan sains. Studi struktur internal fungsi hash adalah sains. Menemukan desain yang baik untuk blok bangunan utama (fungsi kompresi) adalah lebih mengarah ke seni daripada ke sains.

Jika kompresi fungsi hash sekuat sebagaimana seharusnya, serangan seperti Joux akan membutuhkan lebih dari $2^{n/2}$ unit waktu (untuk saat ini $n > 160$; mungkin 25 tahun mendatang $n > 200$).

Walau bagaimanapun, perancang-perancang modern fungsi hash belumlah seperti yang diharapkan. Berbagai macam serangan terhadap fungsi hash (misal: serangan terhadap fungsi kompresi) telah ditemukan baru-baru ini. Menemukan *collision* adalah tugas yang paling mudah bagi para penyerang, sebagian besar serangan adalah berupa *collision attack* dan bahkan *2nd preimage attack*.

SHA-1 mungkin adalah kriptografi fungsi hash yang paling banyak digunakan, paling tidak untuk konteks digital signature. Hingga awal 2005, tidak ada serangan terhadap SHA-1. Pada February 2005, berdasarkan analisis fungsi hash telah ditemukan *collision attack* pada SHA-1.

2.5 Eksploit

Dengan sedikit pengecualian, serangan terhadap fungsi hash adalah terbatas dalam menemukan *collision*, dan *collision* ini kurang lebih acak. Bagi kriptografer, hasil tersebut adalah menarik, tetapi bagi para praktisi hal tersebut tidak relevan. Serangan atau *attack* mengurangi 2 kualitas penting.

- Mereka hanya mendapatkan *collision*, misalnya sebuah algoritma menghasilkan dua message M dan M' dengan $H(M) = H(M')$. Skenario serangan praktis akan membutuhkan *2nd preimages*, contohnya algoritma yang menerima M sebagai inputnya dan menghasilkan M' dengan $H(M) = H(M')$.
- Musuh tidak memiliki kontrol terhadap M dan M' , dengan probabilitas M dan M' yang terlihat seperti sampah acak. Sebagaimana terlihat, skenario serangan mengasumsikan musuh untuk menemukan *message* M' yang berarti dari *message* M yang diberikan.

Pada prakteknya, semua *collision* menemukan algoritma

- dapat mengambil sebuah nilai awal $H(0)$ sebagai inputnya dan kemudian menghitung dua message yang bentrok yaitu M dan M' dengan dua properti sebagai berikut.
- M dan M' keduanya memiliki panjang yang sama, dan
- M dan M' pendek (misalnya 1024 bit).

Sekarang konstruksi Merkle/Damgard memungkinkan untuk membangun message $(S||M||T)$ dan $(S||M'||T)$ yang lebih panjang dan juga bentrok. Asumsikan S terdiri atas i blok, $S[1], \dots, S[i]$. Hitung hash *intermediate* $H[i]$. Gunakan algoritma penemuan *collision* dengan menggunakan $H[i]$ sebagai input, sebagai ganti nilai awal semula yaitu $H[0]$. Hasilnya adalah dua message M dan M' . Melalui sifat-sifat konstruksi Merkle/Damgard akan diperoleh :

$$H(S||M) = H(S||M')$$

Untuk setiap postfix, akan diperoleh

$$H(S||M||T) = H(S||M'||T)$$

Sehingga, berikut ini adalah urutan pengerjaannya :

2. Pilih prefix S , dan hitung input $H[i]$ untuk algoritma penemuan *collision*.

3. Diperoleh M dan M' dari algoritma penemuan collision.
4. Pilih postfix T

Pertimbangkan penulisan beberapa tabel eksekusi. Prefix S berakhir dengan beberapa *forward-jump* ke dalam T . Postfix T mereferensikan beberapa konstanta yang dilokasikan pada posisi alamat dimana M atau M' tersimpan. Maka, eksekusi $(S//M//T)$ akan berbeda dari *collision hash*-nya $(S//M'//T)$, dan ketika penulisan T , pembuatnya telah mengetahui M dan M' dan dapat menentukan eksekusi apa yang sementara berjalan. Misalnya, menjalankan $(S//M//T)$ akan melayani beberapa tujuan, sementara menjalankan $(S//M'//T)$ mungkin akan membuka *back door* terhadap sistem.

Diawal-awal setelah penemuan pertama collision pada MD5, Kaminski dan Mikle mengutak-atik tabel eksekusi. Tabel eksekusi Mike yang merupakan arsip yang terekstrak dengan sendirinya, mengekstraksi bahan yang berbeda. Tabel eksekusi Kaminski adalah aman, sementara yang lainnya berbahaya dalam pembukaaan sebuah *backdoor*.

Lenstra, Wang dan de Weger adalah yang pertama mendemonstrasikan trick yang serupa terhadap non executable. Mereka membangun *collision* sertifikat X.509. Kemudian, Daum dan Lucks mendemonstrasikan *collision* MD5 pada dokumen PostScript menggunakan konstruksi *if-then-else*. Serangan yang sama dapat terjadi pula bagi dokumen-dokumen lain seperti PDF atau TIFF. Ini telah ditunjukkan oleh Gebhardt, Illies, dan Schindler yang memainkan tipuan tabel warna dan informasi dokumen yang serupa.

Kemampuan menghasilkan sejumlah besar pesan yang *collision* dapat digunakan sebuah tool yang hebat untuk menaikkan *Fuzzing Attacks*. Ide utama dari Fuzzing adalah untuk menguji aplikasi-aplikasi dengan mengirimkan data random. Ini merupakan langkah yang panjang dalam pengujian software tetapi juga digunakan dalam tool-tool penyerangan otomatis.

Seorang penyerang dapat memperoleh sebuah tandatangan untuk paket yang bebas dari bahaya, memiliki sejumlah besar paket acak yang tertanda dan telah diuji Fuzzing.

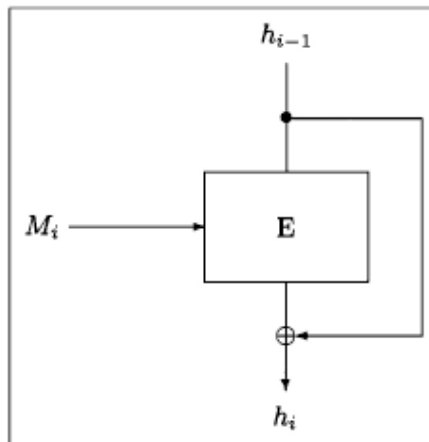
BAB III

FUNGSI HASH : TIGER

Diantara kriptografi fungsi hash yang tidak berbasis block chiper, MD4 dan Snefru terlihat sangat menarik untuk aplikasi-aplikasi yang membutuhkan software hashing yang cepat. Namun, *collision* pada Snefru ditemukan pada tahun 1990, dan tidak lama kemudian *collision* pada MD4 juga ditemukan. Demikian pula varian-varianannya seperti RIPE-MD, MD5, SHA, SHA-1 dan Snefru-8 yang telah berhasil ditemukan *collision* di dalamnya. Lebih jauh lagi, semua fungsi hash tersebut dirancang untuk prosesor 32 bit, dan tidak dapat diimplementasikan pada generasi prosesor 64 bit seperti DEC Alpha. Pada tahun 1996, dirancanglah fungsi hash Tiger oleh Ross Anderson dan Eli Biham yang didesain dapat berjalan secara cepat pada prosesor 64 bit.

3.1 Spesifikasi Fungsi Hash Tiger

Fungsi kompresi Tiger berdasarkan pada penerapan fungsi internal “*block chipper*”, yang memerlukan 192 bit plaintext dan 512 bit *key* untuk menghitung 192 bit “*ciphertext*”. Fungsi “*block chipper*” diterapkan menurut konstruksi Davies-Meyer : 512 bit blok *message* digunakan sebagai kunci (*key*) untuk meng-encrypt 192 bit *hash value*, dan kemudian input *hash value* di-*feedforward* untuk menjadikan seluruh fungsi *non-invertible*.



Gbr.3.1 Mode *Feedforward*

Tiger dirancang dengan arsitektur 64 bit. Untuk itu, akan dinotasikan sebuah *unsigned integer* 64 bit sebagai sebuah “*word*”. *Word* akan direpresentasikan dalam bentuk heksadesimal. Tiger menggunakan operasi-operasi aritmetika, bit-wise XOR, NOT, operasi logika shift, dan aplikasi S-Box. Operasi aritmetika terhadap *word* adalah modulus 2^{64} . *Hash value* direpresentasikan secara internal sebagai 3 buah register 64 bit *a*, *b*, dan *c*. Register-register ini diinisialisasi dengan h_0 dimana :

```
a = 0x0123456789ABCDEF
b = 0xFEDCBA9876543210
c = 0xF096A5B4C3B2E187
```

Masing-masing suksesor 512 bit blok message dibagi kedalam delapan *word* 64 bit x_0, x_1, \dots, x_7 , dan komputasi berikutnya dilakukan untuk meng-update h_i hingga h_{i+1} .

Komputasi ini terdiri dari 3 laluan (*passing*), dan di antara setiap laluan tersebut terdapat sebuah *key schedule*, yaitu suatu transformasi invertible dari data input yang mencegah penyerang memaksakan input di ketiga round tersebut. Pada akhirnya terdapat suatu tahap *feedforward* yang mana nilai baru *a*, *b* dan *c* dikombinasikan dengan nilai awalnya untuk menghasilkan h_{i+1} .

```
save_abc
pass(a,b,c,5)
key_schedule
pass(c,a,b,7)
key_schedule
pass(b,c,a,9)
feedforward
```

dengan

1. `save_abc` menyimpan nilai h_i

```
aa = a ;
bb = b ;
cc = c ;
```

2. `pass(a,b,c,mul)` adalah

```
round(a,b,c,x0,mul);
round(b,c,a,x1,mul);
round(c,a,b,x2,mul);
round(a,b,c,x3,mul);
round(b,c,a,x4,mul);
```

```

round(c,a,b,x5,mul);
round(a,b,c,x6,mul);
round(b,c,a,x7,mul);

```

dengan `round(a,b,c,x,mul)` adalah

```

c ^= x ;
a -= t1[c_0] ^ t2[c_2] ^ t3[c_4] ^ t4[c_6] ;
b += t4[c_1] ^ t3[c_3] ^ t2[c_5] ^ t1[c_7] ;
b *= mul;

```

dan `ci` adalah byte ke-*i* dari `c` ($0 \leq i \leq 7$). Catatan, kita menggunakan notasi pemrograman dengan bahasa C, dengan `^` melambangkan operator XOR, dan notasi `X op=Y` berarti `X = X op Y` untuk setiap operator `op`.

3. `key_schedule` adalah

```

x0 -= x7 ^ 0xA5A5A5A5A5A5A5A5;
x1 ^= x0;
x2 += x1;
x3 -= x2 ^ ((~x1)<<19);
x4 ^= x3;
x5 += x4;
x6 -= x5 ^ ((~x4)>>23);
x7 ^= x6;
x0 += x7;
x1 -= x0 ^ ((~x7)<<19);
x2 ^= x1;
x3 += x2;
x4 -= x3 ^ ((~x2)>>23);
x5 ^= x4;
x6 += x5;
x7 -= x6 ^ 0x0123456789ABCDEF;

```

dengan `<<` dan `>>` adalah operator logika *shift left* dan *shift right*.

4. `feedforward` adalah

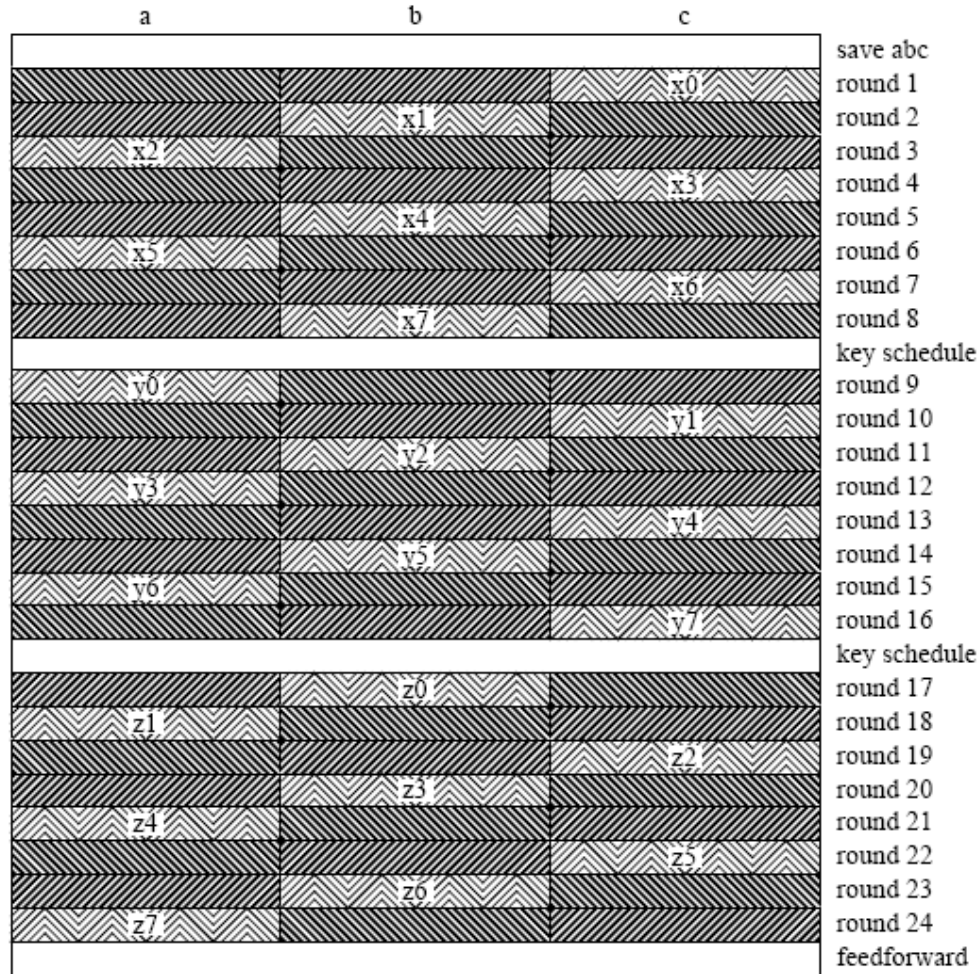
```

a ^= aa ;
b -= bb ;
c += cc ;

```

Resultan register-register `a`, `b`, `c` adalah 192 bit untuk hash value `hi+1`.

Gambar berikut memperlihatkan fungsi kompresi. Pada gambar, area berwarna hitam menggambarkan register-register yang terlibat, dimana arsiran miring (*slanted line*) menunjuk pada byte-byte di area berwarna putih. Variabel-variabel y_0, y_1, \dots, y_7 dan z_0, z_1, \dots, z_7 melambangkan nilai-nilai x_1, x_2, \dots, x_7 pada passing (laluhan) kedua dan ketiga. Akhirnya, nilai akhir h_n diambil sebagai output Tiger/192.



Gbr.3.2. Fungsi Kompresi Tiger

3.2 Keamanan (*Security*)

Beberapa aspek keamanan yang dapat dikemukakan sehubungan dengan fungsi hash Tiger adalah :

1. Ketidak-linear-an sebagian besar berasal dari S-Box 8 bit hingga 64 bit. Ini lebih baik dibanding hanya mengkombinasikan penjumlahan dan XOR,

dan ini cenderung pada seluruh bit-bit output, bukan hanya pada bit-bit tetangga.

2. Terdapat *avalanche* yang kuat, pada setiap bit *message* di ketiga register setelah putaran (*round*) ketiga. Ini lebih cepat dibanding fungsi hash yang lain. *Avalanche* pada *word* 64 bit (dan 64 bit S-Box) lebih cepat dibanding ketika menggunakan *word-word* yang lebih pendek.
3. Seperti yang telah dijelaskan sebelumnya, semua serangan pada MD / Snefru menargetkan pada satu blok *intermediate*. Penambahan nilai *intermediate* ke 192 bit membantu menghalangi serangan tersebut.
4. *Key schedule* menjamin bahwa perubahan sejumlah kecil bit *message* mempengaruhi banyak bit selama proses *passing* (lalu) berlangsung. Bersama dengan *avalanche* yang kuat, ia dapat membantu Tiger untuk dapat bertahan terhadap serangan *Dobbertin's differential attack* yang terjadi pada MD4 (dimana perubahan bit-bit *message* tertentu mempengaruhi paling banyak 2 bit di banyak putaran (*round*), dan kemudian perbedaan kecil ini dapat dikeluarkan di lalu (*passing*) terakhir).
5. Perkalian register b di tiap putaran (*round*) juga memberikan ketahanan terhadap serangan, dikarenakan jaminan bahwa bit-bit yang digunakan sebagai input di S-Box pada putaran sebelumnya dicampur ke dalam S-Box lainnya juga, dan dalam S-Box yang sama dengan sebuah perbedaan input. Perkalian ini juga mencegah serangan *related-key* pada fungsi hash dikarenakan perbedaan konstanta di setiap putaran.
6. *Feedforward* melindungi terhadap *birthday attack* yang terjadi di pertengahan yang menemukan *preimage* fungsi hash (meskipun kompleksitasnya mencapai 2^{96}).

BAB IV P E N U T U P

Fungsi hash Tiger dirancang untuk menjadi yang tercepat dan teraman (*secure*) pada saat itu. Intinya adalah 3 putaran (*round*), masing-masing putaran menggunakan 8 *lookup* kedalam 8 hingga 64 bit S-Box yang memberikan kekuatan *avalanche non-linier* ditambah sejumlah operasi register untuk meningkatkan difusi dan membuat sulitnya serangan.

Tiger dapat diimplementasikan secara efisien pada mesin 32 bit dan 64 bit. Pada bentukannya, ia secepat SHA-1, tetapi tidak seperti SHA-1, ia dapat digunakan di mesin 64 bit dimana kira-kira 2,5 kali lebih cepat dari SHA-1. Tiger dapat pula diimplementasikan pada mesin 16 bit, dimana dia lebih cepat dari SHA-1.

Tiger menghasilkan output 192 bit *hash value*. Untuk kompatibilitas dengan fungsi hash yang ada, disarankan outputnya di-*truncated* ke 160 bit atau 128 bit jika diperlukan dalam rangka kompatibilitas dengan aplikasi-aplikasi yang ada.

Kecepatan dan keamanan fungsi Hash Tiger tidak menjamin bahwa ia akan bebas dari *collision*. John Kelsey dan Stefan Lucks telah menemukan *near-collision* pada *Reduced-Round Tiger*. Teknik ini melawan terhadap 20 *rounds* Tiger dan membutuhkan pekerjaan komputasi fungsi kompresi sebesar 2^{48} .

DAFTAR PUSTAKA

1. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996, "*Handbook of Applied Cryptography*".
2. Ross Anderson, Wiley, 2001, "*Security Engineering: A Guide to Building Dependable Distributed Systems*".
3. Bruce Schneier, John Wiley & Sons, 1996, "*Applied Cryptography*".
4. Ross Anderson, "*The Classification of Hash Functions*", in 'Codes and Ciphers', proceedings of Fourth IMA Conference on Cryptography and Coding.
5. Ross Anderson and Eli Biham., *Tiger: A Fast New Hash Function*, Fast Software Encryption, FSE'96, LNCS 1039, 1996.