MyHash

William – 13508032
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
If18032@students.if.itb.ac.id

ABSTRAK

Dari banyak topik mengenai kriptografi, menurut penulis, fungsi hash adalah salah satu topik yang paling menarik. Fungsi hash ini adalah suatu prosedur deterministic yang gunanya adalah untuk mengambil suatu blok data apapun, dan akan mengembalikan sebuah string bit dengan ukuran yang tetap.

Salah satu keunikan yang membuat fungsi hash ini menarik adalah bahwa untuk ukuran blok data sebesar apapun, nilai *message digest*nya (hasil dari pengaplikasian fungsi hash terhadap blok data tersebut) akan selalu memiliki panjang yang tetap. Tidak hanya itu, sebuah fungsi hash yang ideal tidak hanya harus mudah untuk dilakukan komputasi untuk pesan sepanjang apapun, juga akan mengembalikan nilai *message digest* yang berbeda untuk setiap blok data; tidak ada 2 file yang berbeda yang akan menghasilkan nilai *message digest* yang sama.

Selain itu, suatu fungsi hash yang baik juga sebaiknya tahan terhadap serangan. Yang dimaksud dengan serangan disini adalah bahwa akan menjadi sangat sulit bagi penyerang untuk membentuk blok data / pesan yang memiliki nilai *message digest* yang sama dengan nilai yang sudah diketahui, dan bahwa akan sangat sulit untuk mengubah ini dari suatu blok data / pesan tanpa mengubah nilai *message digest*nya.

Pada makalah inim sesuai judulnya, akan dibahas mengenai myHash, yaitu sebuah fungsi Hash original untuk menghitung nilai *message digest* dari suatu file.

Kata kunci: fungi hash, kriptografik, *message digest*, pesan.

1. PENDAHULUAN

Suatu fungsi hash kriptografik banyak digunakan untuk apikasi keamanan informasi, terutama pada aplikasi tanda tangan digital, yang digunakan untuk kode autentifikasi pesan (*Message Authentication codes* – MAC), dan bentuk lainya dari autentifikasi. Namun aplikasinya tidak berheni disana. Suatu fungsi hash kriptografik dapat juga digunakan sebagai fungsi hasi biasa, yaitu untuk memberikan index pada data atau menidentifikasi file unik, dan juga dapat digunakan sebagai checksum untuk mendeteksi data yang berubah nilainya akibat kesengajaan ataupun tidak. Karena itu, untuk konteks keamanan informasi, nilai hash kriptografik sering juga disebut sebagai sidik jari (digital), checksum, atau hanya nilai hash.

Fungsi hash kriptografik yang baik, tidaklah mudah untuk dibuat. Salah satu syaratnya adalah bahwa fungsi ini haruslah dapat menahan segala jenis tipe serangan kriptanalisis. Dan paling tidak memiliki sifat sebagai berikut:

1. *Preimage resistance*
   Untuk suatu nilai hash *h*, haruslah sulit untuk mencari sebuah pesan *m* sehingga *h* = hash(*m*). Konsep ini berelasi dengan fungsi satu arah. Fungsi hash yang tidak memiliki sifat ini dapat diserang dengan serangan *preimage*.
2. *Second preimage resistance*
   Untuk setiap masukan *m1*, haruslah sulit untuk menemukan input *m2* lainnya dimana *m1* ≠ *m2* sehingga has(*m1*) = hash(*m2*). Sifat ini seringkali dikenal sebagai *weak collision resistance*, dan fungsi hash yang tidak memiliki sifat ini dapat diserang dengan serangan *second preimage*.
3. *Collision resistance*
   Haruslah sulit untuk menemukan 2 pesan yang berbeda *m1* dan *m2* sehingga hash(*m1*) = hash(*m2*). Kedua pasangan tersebut dikenal sebagai kolisi hash kriptografik. Sifat ini seringkali dikenal sebagai *strong collision resistance*. Sifat ini membutuhkan nilai hash yang seditaknya 2 kali lipat lebih panjang dari yang dibutuhkan untuk *preimage resistance*. Karena jika tidak, kolisi akan dapat ditemuka dengan serangan ulang tahun (*birthday attack*).

Sfat – sifat tersebut dibutuhkan agar penyerang tidak dapat mengubah atau mengganti data masukan tanpa mengubah nilai *message digest*-nya. Jadi, bila suatu masukan memiliki nilai *messag digest* yang sama, kita dapat yakin bahwa kedua masukan tersebut adalah identik. Namun demikian, fungsi yang memenuhi sifat tersebut, terkadang masih tetap memiliki sifat yang tidak diinginkan. Sekarang ini sedang populer suatu serangan yang disebut dengan *length-extension attack*. Serangan ini adalah bahwa untuk sebuah h(m) dan len(m) tapi

tidak m, dengan menemukan suatu nilai m', seorang penyerang dapat menghitung h(m||m') dimana || adalah konkatenasi. Sifat ini dapat digunakan untuk memecahkan skema autentifikasi naïf dari fungsi hash.

Idealnya, kita harus membuat suatu fungsi hash sehingga penyerang tidak dapat mendapatkan suatu nformasi yang penting dari suatu data masukan hanya dengan melihat nilai *message digest*-nya saja. Sehingga, suatu fungsi hash kriptografik haruslah bertindak semirip mungkin dengan fungsi acak (*random*) dan tetap bersifat deterministic dan dapat dikomputasi secara efisien.

2. TEORI DASAR

Seperti yang sudah disebutkan sebelumnya, untuk masukkan sepanjang apapun, setelah dikenakan fungsi hash, akan dihasilkan suatu *message digest* yang panjangnya selalu sama untuk algoritma tersebut. Hal tersebut dapat dicapai dengan memecah – mecah masukkan menjadi kumpulan blok dengan panjang yang sama, dan melakukan fungsi kompresi satu arah terhadap kumpulan blok tersebut. Fungsi kompresi yang digunakan dapat didesain khusus untuk melakukan fungsi hash, atau membentuknya dari *block cipher*.

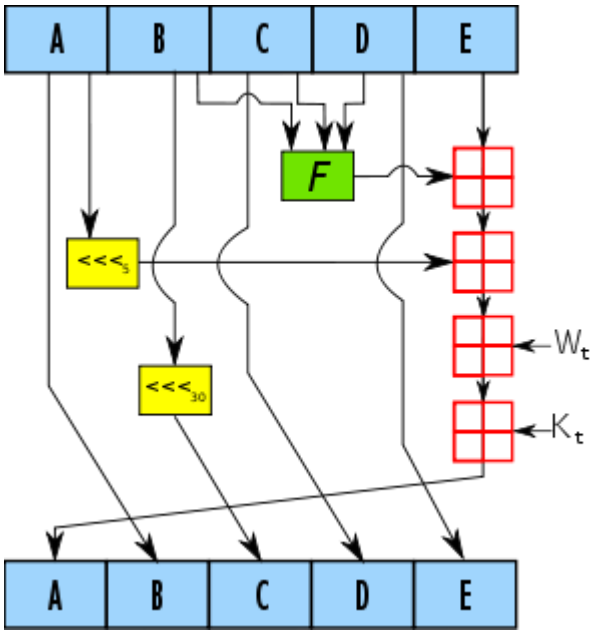
Karena algoritma yang akan dibangun ini memanfaatkan sifat – sifat yang dimiliki algoritma algoritma untuk menghitung fungsi hash yang sudah ada, akan dijelaskan mengenai proses – proses untuk menghitung nilai *message digest* padabeberapa algoritma yang sudah ada dan cukup sering dan aman untuk digunakan

2.1. SHA-1

SHA-1 menghasilkan *message digest* berukuran 160 bit berdasarkan prinsip yang mirip dengan yang digunakan oleh Ronald L. Rivest dari MIT pada desain dari algoritma MD4 dan MD5 namun memiliki desain yang lebih konservatif.

Spesifikasi original dari algoritma tersebut pertama kali dipublikasikan pada 1983 sebagai *Secure Hash Standard*, FIPS PUB 180 oleh NITS (*National Institute of Standards and Technology* – Institut Nasional Standard an Teknologi) dari Amerika Serikat. Versi ini sekarang sering juga disebut sebagai SHA-0. SHA-0 ditarik oleh NSA beberapa saat setelah dipublikasikan dan kemudian setelah dilakukan beberapa penyesuaian dan revisi untuk meningkatkan ketahanannya terhadap serangan, dipublikasikan kembali sebagai SHA-1. SHA-1 berbeda dengan SHA-0 hanyadengan satu rotasi bit pada pesan pada fungsi kompresinya.

Dibawah ini adalah gambar dari satu iterasi pada fungsi kompresi SHA-1, dengan:
A,B,C,D,dan E adalah pesan berukuran 32 bit
F adalah fungsi nonlinear yang digunakan
$\lll_n$ adalah rotasi bit ke kiri sejauh n dengan nilai n berbeda untuk setiap operasi
W_t adalah pesan pada perputaran ke t
K_t adalah konstanta perputaran pada perputaran ke
$\boxplus$ adalah penjumlahan dalam modulo 2^{32}.



Gambar 1. Satu ronde perputaran pada fungsi SHA-1
Gambar diambil dari Wikipedia

Langkah – langkah pengerjaan fungsi SHA-1 adalah:

1. Inisialisasi setiap variable yang akan digunakan
2. Tambahkan bit '1' pada pesan
3. Tambahkan 0 ≤ k ≤ 512 bits 0 sehingga panjang pesan kongruen dengan 448 ≡ -64 (mod 512)
4. Tambahkan panjang pesan sebelum emrosesan, dalam bits, sebagai integer 64-bit dalam *big endian*
5. Bentuk pesan menjadi blok – blok berukuran 512 bit
6. Untuk setiap blok pecahkan lagi menjadi 16 32-bit pesan w[i] dengan 0 ≤ i ≤ 15
7. Tambahkan w[i] dari I = 16 sampai I = 79 dengan aturan: w[i] = (w[i-3] xor w[i-8] xor w[i-14] xor w[i-16]). Geser hasil operasi ini ke kiri sejauh 1 bit
8. Terakhir, lakukan fungsi kompresi untuks setiap w setiap bloknya.

Nilai *message digest* dari fungsi ini adalah nilai konkatenasi dari h0, h1, h2, h3, dan h4 dengan nilai – nilai tersebut didapatkan dari fungsi kompresinya.

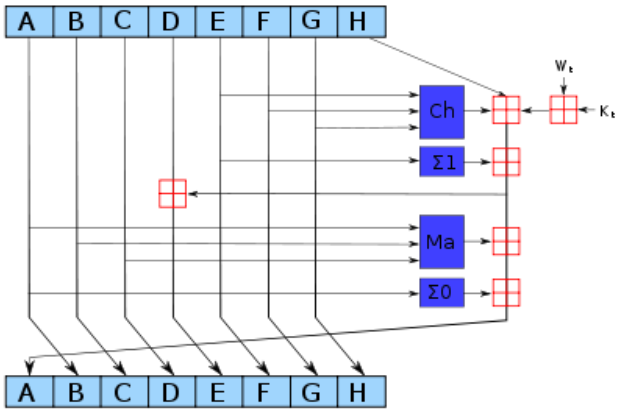
2.2. SHA-2

SHA-2 adalah hasil dari 4 buah publikasi NIST lainnya yaitu SHA-224, SHA-256, SHA-384, dan SHA-512 dengan angka – angka tersebut adalah panjang *message digest*-nya dalam bits.

Perbedaan dari setiap fungsi hash tersebut selain itu hanyalah ukuran panjang pesan yang digunakan, jumlah pergeseran bit, dan konstanta penambahan. Selain hal – hal tersebut, struktur yang mereka semua gunakan

adalah identik dengan perbdaan hanya pada jumlah perputaran.

Berikut ini adalah gambar yang memperlihatkan satu iterasi pada fungsi sha-2:



Gambar 2. Satu iterasi pada SHA-256.
Gambar diambil dari Wikipedia

Dengan:

$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$

$\mathrm{Ma}(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$

$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$

$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$

Angka – angka yang digunakan adalah untuk SHA-256. Angka untuk rotasi bit yang berbeda digunakan pada SHA-512. ⊞ adalah penjumlahan tanpa penyimpanan.

Langkah – langkah pengerjaan fungsi SHA-256 adalah:

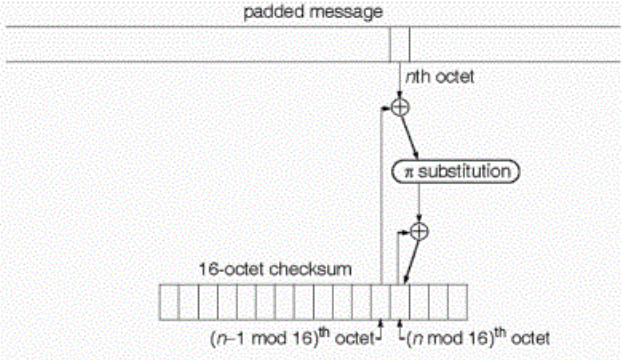
1. Inisialisasi variable
2. Tambahkan bit '1' pada pesan
3. Tambahkan k bit '0' dimana k adalah jumlah minimum >= 0 sehingga panjang pesan dalam bit kongruen dengan 448 (mod 512)
4. Tambahkan panjang pesan sebelum pemrosesan dalam bit sebagai integer *big endian* 64-bit
5. Bentuk pesan menjadi blok – blok berukuran 512 bit
6. Untuk setiap blok bentuk lagi menjadi 16 32-bit pesan dalam *big endian* sebagai w[0..15]
7. Tambahkan w[16..63] sesuai aturan yang ada
8. Lakukan fungsi kompresi untuk setiap w setiap bloknya.

Nilai *message digest*-nya adalah konkatenasi dari h0..h7 dengan nilai – nilai tersebut didapatkan setelah proses kompresi.

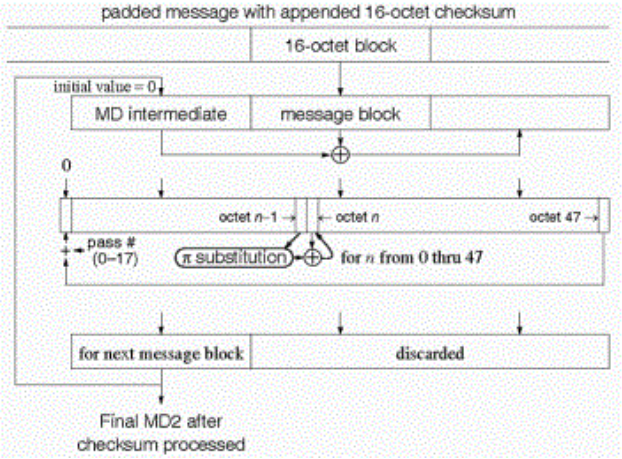
2.3. MD2

Algoritma MD2 adalah fungsi hash yang dikembangkan oleh Ronald Rivest pada 1989. Algoritma ini dioptimasikan untuk computer 8-bit. MD2 dispesifikasikan dalam RFC1319. Walau MD2 sudah tidak lagi aman, sampai sekarang, masih cukup banyak digunakan untuk infrastruktur kunci publik.

Nilai hash 128-bit dari pesan apapun dibentuk pertama – tama dengan melakukan padding dengan panjang blok sesuai kelipatan dari panjang blok computer (128 bit atau 16 byte) dan menambahkan 16-byte chechsum. Untuk kalkulasi, 48-byte blok tambahan dan 256-byte table S dibentuk secara tidak langsung dari digit dari nilai pecahan pi. Algoritma ini berjalan dalam loop dengan melakukan permutasi untuk setiap byte pada blok tambahan 18 kali untuk setiap 16 byte input yang diproses. Saat semua blok (yang sudah diperpanjang) diproses, bagian pertama dari blok tambahan menjadi nilai hash dari pesan.



Gambar 3. Penambahan checksum pada MD2



Gambar 4. Putaran terakhir pada MD2. Pemrosesan dilakukan setiap 16 byte, dan setiap 16 byte pesan akan diekspansi menjadi 48 byte.

2.4. MD-5

MD5 adalah salah satu fungsi hash kriptografik yang paling sering digunakan dengan nilah hash sepanjang 128-bit (16-byte). MD5 dispesifikasikan pada RFC1321. Panjang *message digest* biasanya adalah angka hexadecimal 32 digit.

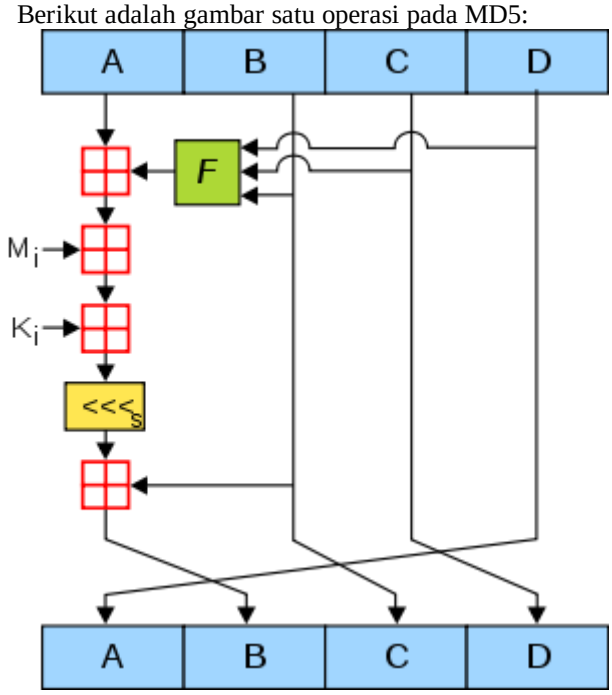
MD5 didesain oleh Ron Rivest pada 1991 untuk menggantikan fungsi hash sebelumnya, MD4. Walau kemudian, ditemukan banyak sekali celah yang ditemukan pada serangan – serangan yang dilakukan pada MD5 sehingga puncaknya setelah serangan yang dipublikasikan pada 2008, lembaga sertifikasi Amerika

Serikat kemudian menyatakan bahwa MD5 sudah dianggap rusak secara kriptografis dan tidak cocok lagi untuk digunakan. Namun demikian, hingga saat ini, kecuali aplikasi yang berhubungan dengan pemerintah Amerika Serikat, masih cukup banyak aplikasi yang mengandalkan MD5 untuk menghitung nilai *message digest* dari suatu masukan.

MD5 melakukan pemrosesan dengan cara:

1. Inisialisasi seluruh variabel
2. Bagi masukan menjadi blok berukuran 512-bit
3. Tambahkan bit '1' pada pesan
4. Tambahkan k bit '0' sehingga sehingga panjang pesan dalam bit kongruen dengan 448 (mod 512)
5. Tambahkan panjang pesan sebelum pemrosesan dalam bentuk integer 64 bit dalam big endian.
6. Lakukan fungsi kompresi MD5
7. Nilai *Message Digest*-nya adalah konkatenasi dari h0...h3 dimana nilai – nilai tersebut didapatkan setelah melalui proses kompresi.

Berikut adalah gambar satu operasi pada MD5:



Gambar 5. Satu operasi pada MD5.
Gambar diambil dari wikipedia

MD5 berisikan operasi berikut, dengan 4 kemungkinan fungsi f yang berbeda. Keempat fungsi f tersebut adalah:

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

3. Implementasi

Setelah memahami mengenai algoritma – algoritma dan sifat – sifat yang digunakan oleh fungsi – fungsi hash yang sudah ada, sekarang akan dibahas mengenai pembangunan dan implementasi dari myHash.

Sesuai aturan yang ada bahwa untuk mendapatkan nilai *message digest* yang panjangnya sama untuk file sepanjang apapun, pertama – tama perlu ditetapkan panjang blok yang untuk operasi untuk kemudian membagi pesan menjadi sepanjang blok – blok tersebut sebelum mulai dioperasikan.

Mengingat sekarang ini secara umum ukuran file sudah menjadi sangat besar, maka ukuran blok yang lebih besar pun sebenarnya tidak terlalu menjadi masalah. Karena itu, untuk algoritma ini, ukuran pemrosesan setiap blok utama berukuran 1024 bit. Pertimbangannya selain ukuran file yang secara umum sudah bertambah, kekuatan komputasi computer juga sudah bertambah menjadi sangat pesat sehingga ukuran blok utama berukuran 1024 tidaklah menjadi masalah.

Berikut ini adalah algoritma dari fungsi myHash:

Catatan 1: Seluruh variable adalah 32 bit unsigned integer dan saat melakukan kalkulasi digunakan modulo 2^{32}

Catatan 2: Seluruh konstanta dalam algoritma ini adalah dalam *big endian*

1. Inisialisasi variable yang digunakan. Variabel tersebut adalah:
   h0 = 0x67452301
   h1 = 0xEFCDAB89
   h2 = 0x98BADCFE
   h3 = 0x10325476

Pra pemrosesan

2. Tambahkan bit '1' pada pesan
3. Tambahkan $0 \leq k \leq 512$ bit '0' pada pesan, sehingga panjang pesan (dalam bit) adalah kongruen dengan $896 \equiv -128 \pmod{1024}$
4. Tambahkan panjang pesan (sebelum pemrosesan) dalam bit, sebagai integer 128 bit dalam big endian

Pemrosesan pesan kemudian dilakukan dalam blok – blok berukuran 1024 bit

5. Bentuk pesan menjadi blok – blok berukuran 1024 bit
6. Untuk setiap blok, bentuk menjadi 32 blok berukuran 32 bit, masing – masing dalam big endian. Namakan seluruh blok tersebut w[i] dengan i bernilai [0..31]

Tambahkan blok – blok tambahan sehingga 32 blok berukuran 32 bit menjadi 128 blok berukuran 32 bit.

7. Untuk i dari 32 sampai 127, nilai w[i] adalah:
   w[i] = (((w[i-32] xor w[i-1]) leftrotate 8) xor ((w[i-15] xor w[i-16]) rightrotate 8)) leftrotate 2
8. Inisialisasikan nilai hash untuk blok ini:
   a = h0
   b = h1
   c = h2
   d = h3

Loop utama:

9. Untuk n = [0..31], lakukan:
   Untuk w[4n+0], temp0 = nilai sha-1 dari w[4n+0] menjadikan temp0 berukuran sepanjang 40 bit

Untuk w[4n+1], temp1 = nilai sha-256 dari w[4n+1] menjadikan temp1 berukuran sepanjang 40 bit

Untuk w[4n+2], temp2 = nilai MD2 dari w[4n+2] menjadikan temp2 berukuran sepanjang 32 bit

Untuk w[4n+3], temp3 = nilai MD5 dari w[4n+3] menjadikan temp3 berukuran sepanjang 32 bit

a = bit [0..7] dari temp0 xor bit[0..7] dari temp1 xor bit[0..7] dari temp2 xor bit[0..7] dari temp3

b = bit [8..15] dari temp0 xor bit[8..15] dari temp1 xor bit[8..15] dari temp2 xor bit[8..15] dari temp3

c = bit [16..23] dari temp0 xor bit[16..23] dari temp1 xor bit[16..23] dari temp2 xor bit[16..23] dari temp3

d = bit [24..31] dari temp0 xor bit[24..31] dari temp1 xor bit[24..31] dari temp2 xor bit[24..31] dari temp3

Tambahkan nilai – nilai yang didapat sejauh ini, sehingga:

h0 = h0 + a

h1 = h1 + b

h2 = h2 + c

h3 = h3 + d

10. Konkatenasi nilai – nilai yang ada untuk mendapatkan nilai message digestnya, sehingga:
Message Digest = konkatenasi (h0,h1,h2,h3) sehingga didapatkan suatu nilai hash berukuran 32 bit untuk setiap pesan yang dikompresi.

4. ANALISIS

Ada beberapa kelebihan dan kekurangan dalam implementasi fungsi myHash ini untuk menghitung nilai *message digest* dari suatu masukan.

Kekurangan dari algoritma ini adalah:

1. Algoritma ini memiliki ukuran blok yang relative lebih besar daripada algoritma penghitungan nilai hash lainnya yang berarti algoritma ini akan memakan lebih banyak memori dalam pemrosesannya.

2. Algoritma ini mengimplementasikan algoritma – algoritma yang sudah ada sampai puluhan kali walau untuk setiap pengimplementasiannya hanya digunakan untuk menghitung masukan berukuran kecil. Sementara, setiap algoritma tersebut juga memiliki setiap fungsinya sendiri dengan jumlah perulangan yang tidak sedikit. Berarti, algoritma ini jauh lebih lambat daripada algoritma – algoritma tersebut.

Bandingkan dengan algoritma lainnya, untuk pesan berukuran 100 bit, dengan SHA-1 akan dipecah menjadi 80 blok masing – masing berukuran 32 bit dan melalui 1 loop utama saja. Sementara, dengan algoritma myHash, pesan berukuran 100 bit akan dipecah menjadi 128 blok berukuran 32 bit, dan akan dilakukan 32 kali pemanggilan fungsi SHA-1, SHA-2, MD2, dan MD5. Hal ini berarti ada 128 kali pemanggilan algoritma untuk menghitung fungsi hash yang berarti algoritma ini, untuk file berukuran kelipatan kurang dari 512 bit, akan menjadi 128 kali lebih lambat.

Sementara, untuk file berukuran misalnya 600 bit, dengan algoritma SHA-1, akan dipecah menjadi 160 blok berukuran 32 bit yang berarti akan terjadi 2 kali loop utama. Sementara dengan algoritma ini tetap. Sehingga, untuk file dengan ukuran kelipatan 512 -1024 bit, algoritma ini menjadi 64 kali lebih lambat. Ini berarti rata – ratanya, algoritma ini (128 + 64) : 2 = 96 kali lebih lambat dari algoritma lainnya dengan asumsi ukuran file percobaan merata.

Namun, walau algoritma ini membutuhkan lebih banyak memori dan lebih lambat, sebenarnya jika melihat kemampuan komputasi dan kekuatan computer dewasa ini, hal tersebut bukanlah suatu masalah. Sebagai contoh, lihat algoritma MD5 yang mulai didesain pada 1991. Kekuatan computer dewasa ini jika dibandingkan dengan tahun 1991, sudah lebih menjadi lebih baik lebih dari 100 kali lipat. Sehingga, algoritma ini memanfaatkan hal tersebut, yang mengakibatkan performansi yang lebih buruk sampai 100 kali lipat walau merupakan masalah yang cukup besar, masih dapat ditoleransi.

Terlebih lagi, melihat trend meningkatnya kekuatan computer, dalam beberapa tahun kedepan, setelah makalah ini dipublikasikan, kekuatan computer tentu sudah meningkat lagi sehingga bahkan 100 kali lipat lebih buruk daripada performansi standar pada tahun 1991 tidaklah menjadi masalah yang berarti.

Walau memiliki cukup banyak kekurangan terutama di segi performansi, algoritma ini jauh lebih aman daripada algoritma – algoritma hash lainnya. Menggunakan prinsip dasar dari rantai hash (yang dimanfaatkan pada TLS/SSL – *Transport layer Security/Secure Socket Layer*), yang adalah pemanfaatan banyak fungsi hash untuk mendapatkan suatu nilai hash baru dengan bentuk dasarnya:

Missal untuk 4 rantai, maka:

$$h(h(h(h(x))))$$

yang dapat juga dinotasikan dengan: $h^4(x)$

Sehingga, algoritma ini sangat suit diserang karena, seperti dibahas pada bagian kelemahan, terjadi sangat banyak fungsi penghitungan nilai hash dalam satu kali pembuatan nilai hash dari suatu masukan.

Selain itu, algoritma ini juga masih sangat baru yang berarti bahwa seandainya bisa diserang dan ditemukan celah dan kolisinya, masih belum ditemukan metoda yang tepat dan mangkus untuk melakukan penyerangan. Lihatlah algoritma MD5 yang sebenarnya banyak celah yang bisa dimanfaatkan, membutuhkan waktu sampai belasan tahun sejak pertama kali dipublikasikan sampai dianggap bahwa algoritma tersebut sudah tidak aman lagi dan tidak cocok digunakan untuk file – file yang bersifat sensitif dan membutuhkan kehati – hatian ekstra.

Tidak hanya itu, bahkan ada perkataan yang kurang lebih bunyinya seperti ini: “Mengkonkatenasi output dari beberapa fungsi hash mengakibatkan resistansi terhadap kolisi sebaik algoritma terbaik yang dimasukkan dalam hasil konkatenasi”. Hal ini menyebabkan, karena dalam algoritma ini memasukkan SHA-256 yang sampai saat

ini belum ditemukan kolisinya, berarti juga belum akan dapat ditemukan kolisinya dalam waktu dekat.

5. KESIMPULAN

Walau memiliki kekurangan yang cukup fatal, karena kekurangan tersebut terjadi dalam bentuk performansi sehingga berkaitan dengan kekuatan komputasi, karena dewasa ini kekuatan komputasi sudah menjadi sangat murah, maka akibatnya adalah kekurangan tersebut menjadi hamper dapat diabaikan. Karena, dengan mengorbankan performansi, akan didapatkan sebuah fungi Hash yang memenuhi syarat – syarat fungsi hash ideal, yang meliputi *Preimage resistance*, *second preimage resistance*, dan *collision resistance*.

Karena algoritma ini adalah algoritma baru, apabila bisa diserang, masih akan dibutuhkan waktu yang cukup lama untuk membangun cara dan metode penyerangan yang tepat sehingga dapat mematahkan keamanan algoritma ini. Namun, algoritma ini sudah terbukti sangat aman karena paling tidak sudah sesulit SHA-256 jika tidak lebih sulit dari itu untuk menemukan kolisi. Karena, algoritma ini selain mengimplementasikan operasi – operasi dasarnya sendiri, juga dalam satu proses mengimplementasikan sedikitnya 128 fungsi hash yang sudah ada. Akibatnya adalah untuk melakukan sekali penyerangan, terlebih dahulu si penyerang apabila tidak dapat menemukan pola yang tepat, harus menelusuri dan menembus pertahanan dari 128 fungsi algoritma penghitungan nilai hash ditambah fungsi tambahan yang ada.

6. REFERENSI

[1]http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[2]http://csrc.nist.gov/publications/nistpubs/800-107/NIST-SP-800-107.pdf

[3]http://en.wikipedia.org/wiki/Cryptographic_hash_function

[4]http://en.wikipedia.org/wiki/hash_chain

[5]http://www1.cse.wustl.edu/~jain/cse571-09/ftp/l_07hsh/sld016.htm

[6] http://www.faqs.org/rfcs/rfc1321.html

[7] http://www.rfc-archive.org/getrfc.php?rfc=1319

[8] http://searchsecurity.techtarget.com/definition/MD5
*Seluruh situs diatas diakses pada tanggal 8 Mei 2011

[9]Munir, Rinaldi. *Diktat Kuliah IF5054*, Departemen Teknik Informatika, Institut Teknologi Bandung, 2005.

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, 9 Mei 2011

William
13508032