STUDI PENCARIAN KOLISI PADA *SHA-1* OLEH XIAOYUN WANG dkk.*

Yogie Adrisatria – NIM : 13503035

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13035@students.if.itb.ac.id

Abstraksi

Fungsi *hash* adalah sebuah fungsi yang menerima masukan sebuah string dan akan mengembalikan string keluaran yang panjangnya tetap. Pengaplikasian fungsi *hash* saat ini sudah sangat luas, terutama untuk melakukan pendeteksian keaslian suatu dokumen atau berkas. Algoritma-algoritma untuk melakukan fungsi *hash* ini cukup banyak jumlahnya. Akan tetapi, saat ini yan* g cukup populer dipakai adalah algoritma *hash SHA*-1. Pada awalnya, algoritma *SHA*-1 ini dianggap cukup aman karena satu-satunya cara untuk mencari kolisi (*collision*) adalah dengan menggunakan serangan *brute-force* yang memiliki kompleksitas operasi sebesar 2^{80}, yang dengan super komputer yang ada sekalipun hanya mampu dicari dalam hitungan jutaan tahun. Adapun percobaan serangan terhadap SHA-1 lainnya memang ada yang mampu mencari kolisi dengan kompleksitas kurang 2^{80} tetapi hal tersebut dilakukan terhadap versi yang dikurangi (*reduced-version*) dari SHA-1, yaitu putarannya hanya ada 53.

Pada bulan Februari 2005, tiga orang peneliti dari Cina, Xiaoyun Wang, Yiqun Lisa Yin, dan Hongbo Yu, mempublikasikan bahwa mereka dapat mencari kolisi dalam fungsi SHA-1 versi penuh (*full version*) dalam operasi dengan kompleksitas 2^{69}. Angka tersebut sebenarnya angka yang masih besar karena untuk melakukannya akan membutuhkan waktu 170.000 tahun dengan menggunakan komputer yang tercepat saat ini. Akan tetapi, hal ini cukup menjadi pembicaraan hangat dunia dalam dunia kriptografi. Metode yang dilakukan oleh para peneliti tersebut adalah dengan menggunakan metode yang berdasarkan metode yang sebelumnya telah digunakan untuk mencari kolisi terhadap fungsi-fungsi *hash* yang lain seperti MD5 dan SHA-0, tentunya dengan berbagai perbaikan.

Kata Kunci: *SHA-1*, *Hash, Collision*.

* Ada perubahan judul karena saya salah mengerti mengenai *collision search attack*. Awalnya saya mengira itu sebagai sebuah metode untuk mencari kolisi SHA-1. Ternyata itu adalah padanan kata untuk 'serangan pencarian kolisi'. Jika ada mahasiswa lain yang kebetulan topiknya sama, saya bersedia untuk dikenai pengurangan nilai.

1. Pendahuluan

SHA (*Secure Hash Algorithm*) adalah sebuah keluarga algoritma untuk melakukan fungsi *hash*. *SHA* ini diciptakan sebagai pengganti dari algoritma *hash MD5* yang lebih dulu digunakan secara luas. *SHA* sendiri dirancang oleh NIST (*National Institute of Standards and Technology*) Algoritma pertama yang muncul dalam keluarga ini adalah algoritma *SHA*-0 atau lebih sering disebut sebagai *SHA* yang dipublikasikan pada tahun 1993. Pada tahun 1995, algoritma *SHA*-1 dipublikasikan sebagai pengganti *SHA*-0. *SHA*-1 ini adalah yang paling populer dan telah diimplementasikan di berbagai aplikasi dan protokol keamanan seperti TLS, SSL, PGP, SSH, S/MIME, dan *Ipsec*. Varian lain dari keluarga SHA ini antara lain *SHA*-224, *SHA*-256, *SHA*-384, dan *SHA*-512. Keempat varian tersebut disebut sebagai *SHA*-2.

Salah satu syarat bahwa suatu fungsi *hash* aman digunakan adalah bahwa untuk dua nilai yang berbeda, misalkan x dan y, maka hasil hash dari kedua nilai tersebut tidak boleh sama, atau H(x) != H(y). Kejadian terjadinya nilai *hash* yang sama untuk dua nilai berbeda disebut sebagai kolisi (*collision*). Untuk itu, jika dalam suatu fungsi *hash* dapat ditemukan hal seperti itu, fungsi *hash* tersebut dapat diragukan keamanannya.

Seperti disebutkan sebelumnya, SHA-1 adalah algoritma SHA yang paling populer digunakan. Oleh karenanya, cukup banyak usaha yang dilakukan untuk menemukan kolisi tersebut. Dari percobaan-percobaan yang telah dilakukan, yang paling mencengangkan adalah percobaan yang dilakukan oleh tiga orang ilmuwan dari Cina. Ketiga orang tersebut adalah Xiaoyun Wang, Yiqun Lisa Yin, dan Hongbo Yu.

2. Fungsi *Hash*

Fungsi *hash* adalah fungsi yang menerima masukan *string* yang panjangnya sembarang dan mengonversinya menjadi *string* keluaran yang panjangnya tetap. Persamaan fungsi *hash* dapat dinyatakan sebagai berikut:

$$h = H(M)$$

dimana H adalah fungsi *hash* dengan masukkan pesan *string* M dan akan menghasilkan suatu nilai string *h*. Keluaran *h* disebut juga sebagai *hash-value* atau *message-digest*. Nama-nama lain dari fungsi *hash* antara lain: fungsi kompresi/kontraksi (*compression function*), *fingerprint*, *cryptographic checksum*, *message integrity check* (*MIC*), dan *manipulation detection code* (*MDC*).

Aplikasi dari fungsi *hash* dalam kehidupan sehari-hari cukup banyak. Penggunaan fungsi *hash* yang paling umum adalah untuk melakukan pemverifikasian keaslian suatu berkas. Caranya adalah pengirim pesan mengaplikasikan fungsi *hash* terhadap berkas tersebut. Kemudian, hasil *hash* (*hash-value*/*message-digest*) dari berkas tersebut disimpan. Teknis penyimpanan hasil tersebut dapat dilakukan dengan menyimpannya dalam berkas tersebut ataupun dengan mempublikasikannya secara umum. Pihak penerima kemudian akan melakukan hal serupa, yaitu mengaplikasikan fungi *hash* yang sama terhadap berkas tersebut. Setelah mendapatkan *message-digest*-nya, penerima dapat membandingkan nilai *hash* yang didapat dari pihak pengirim dan nilai *hash* yang ia miliki. Jika kedua nilai *hash* tersebut sama, berarti dapat disimpulkan bahwa pesan yang diterima adalah sama dengan pesan yang dikirimkan. Jika tidak, berarti telah terjadi pengubahan terhadap isi berkas tersebut. Pengubahan satu karakter saja dalam pesan dapat menghasilkan nilai *hash* yang sangat jauh berbeda.

Fungsi *hash* yang baik memiliki beberapa sifat dan karakteristik. Sifat-sifat tersebut adalah sebagai berikut:

1. Fungsi *H* dapat diterapkan pada blok data berukuran berapa saja.
2. *H* menghasilkan nilai *hash* dengan panjang tetap (*fixed-length output*).
3. $H(x)$ mudah dihitung untuk setiap nilai x yang diberikan.
4. Untuk setiap h yang diberikan, tidak mungkin menemukan x sedemikian sehingga $H(x) = h$. Itulah sebabnya fungsi *H* dikatakan fungsi *hash* satu-arah (*one-way hash function*).
5. Untuk setiap x yang diberikan, ti`dak mungkin mencari y != x sedemikian sehingga $H(y) = H(x)$.
6. Tidak mungkin secara komputasi mencari pasangan x dan y sedemikian sehingga $H(x) = H(y)$.

Dari keenam karakteristik di atas, karakteristik yang berkaitan dengan aspek keamanan adalah

karakteristik nomor empat, lima, dan enam. Hal-hal tersebut akan dibahas dalam sub-bab berikut.

2.1 Keamanan Fungsi *Hash*

Seperti disebutkan sebelumnya, aspek keamanan suatu fungsi kriptografi dapat dilihat dari karakteristik nomor empat, lima, dan enam. Khusus untuk nomor empat, yaitu melakukan *inverse* terhadap fungsi *hash*, pada umumnya semua fungsi *hash* yang ada telah menerapkan karakteristik ini dengan sempurna. Hal ini dimungkinkan dengan rancangan algoritma yang memang sudah didesain agar pesan dari suatu pesan-ringkas tidak dapat ditemukan. Meski demikian, karakteristik ini dapat diserang dengan menggunakan suatu metode yang disebut *dictionary attack*, yaitu dengan mendaftarkan kata-kata yang ada dengan pasangan nilai *hash*-nya. Hal ini akan cukup bermanfaat bila digunakan untuk menebak *password* seseorang yang biasanya di basis data disimpan sebagai sebuah nilai *hash*.

Karakteristik nomor lima dan enam merupakan aspek yang hangat dibicarakan dalam pengujian keamanan suatu fungsi *hash*. Serangan terhadap suatu fungsi *hash* pun dikonsentrasikan terhadap kedua jenis hal tersebut. Serangan-serangan tersebut disebut juga sebagai berikut:

1. *Pre-image attacks* : Suatu bentuk penyerangan terhadap karakteristik nomor 5, yaitu serangan dengan berusaha mencari suatu pesan yang memiliki nilai *hash* yang sama dengan suatu pesan yang telah terdefinisi sebelumnya.
2. *Collision attack*: Sutu bentuk serangan terhadap karakteristik nomor 6, yaitu serangan dengan berusaha mencari dua pesan yang memiliki nilai *hash* yang sama.

Sebenarnya secara teori kedua jenis serangan tersebut dapat dilakukan dengan cara *brute-force* [2]. Akan tetapi secara komputasi yang dibatasi oleh kemampuan komputer yang ada, serangan terhadap fungsi-fungsi *hash* yang ada sekarang ini dengan cara *brute-force* aan memakan waktu yang sangat-sangat lama.

Dari beberapa penelitian, ditemukanlah beberapa metode yang dapat digunakan untuk melakukan serangan terhadap suatu algoritma fungsi *hash* secara lebih efisien.

Dalam tabel berikut ini dapat dilihat beberapa fungsi *hash* yang cukup terkenal dan apakah kolisi dari fungsi tersebut telah dapat ditemukan atau belum.

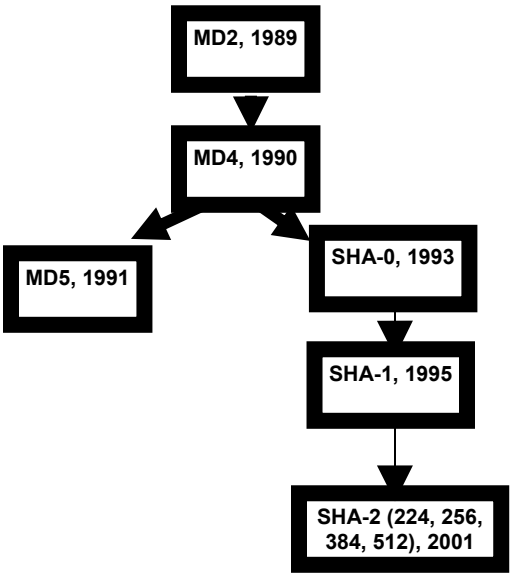
Algoritma	**MD**	**Blok Pesan**	**Kolisi**
MD2	128	128	Ya
MD4	128	512	Hampir
MD5	128	512	Ya
RIPEMD	128	512	Ya
RIPEMD-128/256	128/256	512	Tidak
RIPEMD-160/320	160/32/0	512	Tidak
SHA-0	160	512	Ya
SHA-1	160	512	Ada cacat
SHA-256/224	256/224	512	Tidak
SHA-512/384	512/384	1024	Tidak
WHIRLPOOL	512	512	Tidak

Pertanyaan yang mungkin muncul kemudian adalah apa bahayanya apabila dalam suatu fungsi *hash* ditemukan kolisinya. Salah satu contoh kasusnya misalnya diadakan sebuah perjanjian jual beli rumah dengan harga 100 juta. Dalam surat perjanjian misalnya tertulis “Saya membeli rumah dari tuan X seharga 100 juta”. Surat perjanjian tersebut kemudian dicari nilai *hash*-nya dan disimpan di suatu tempat dan otentikasi dokumen penjualan akan mengacu kepada nilai *hash* yang disimpan tersebut. Misalnya kemudian tuan X ingin berbuat curang dan ia memanfaatkan fungsi *hash* yang cukup lemah. Ia kemudian mencari nilai *hash* yang sama dengan dokumen tersebut. Caranya adalah dengan membuat dokumen palsu yang berisi “Saya membeli rumah dari tuan X seharga 200 juta”. Pertanyaan selanjutnya adalah bagaimana mungkin dua dokumen yang berbeda tersebut bisa langsung memiliki nilai *hash* yang sama. Tentu kemungkinannya sangat kecil. Akan tetapi, bila kolisi dari suatu fungsi *hash* telah dengan sangat mudah ditemukan, maka dokumen palsu tersebut dapat ditambahkan berbagai “kosmetik” seperti diberi nilai *null* di berbagai tempat dan dengan variasi nilai *null* tersebut akhirnya didapatkan dokumen palsu dengan nilai *hash* yang sama. Dengan cara ini, tuan X dapat menuntut orang yang membeli rumah tersebut ke pengadilan.

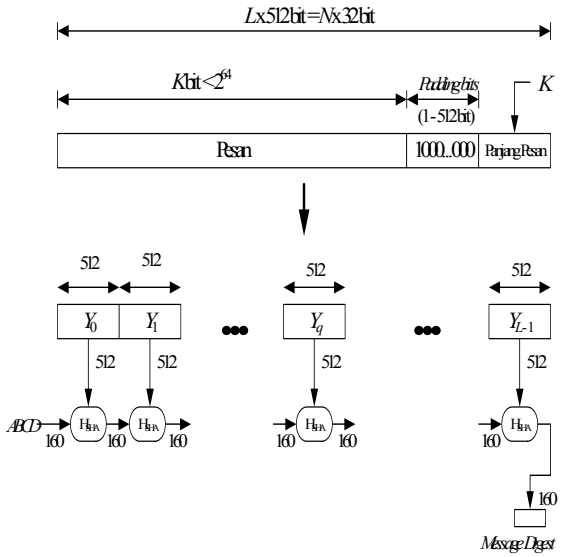
3. Algoritma SHA-1

Berikut akan dijelaskan algoritma dari SHA-1 berdasarkan [4]. *SHA*-1 menerima masukan berupa pesan dengan ukuran maksimum 2^{64} (2.147.483.648 *gigabyte*) dan menghasilkan *message digest* yang panjangnya 160 bit, lebih panjang dari *message digest* dengan algoritma MD5 yang hanya 128 bit.

Algoritma SHA-1 merupakan algoritma yang memiliki prinsip berdasarkan MD4, sebagaimana keluarga SHA lainnya. Dalam gambar di bawah ini, dapat dilihat bagaimana hubungan antara algoritma-algoritma tersebut.



SHA-1 sendiri memiliki proses yang sangat mirip dengan proses yang dilakukan oleh SHA-0, bedanya hanyalah di fungsi kompresinya (*message expansion*), yaitu adanya operasi *bit shift* sebesar 1. Gambaran umum pembuatan *message digest* dengan algoritma SHA-1 diperlihatkan pada gambar di bawah ini.



Secara umum, langkah-langkah pembuatan *message digest* dengan *SHA*-1 adalah sebagai berikut:

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

Pesan ditambah dengan sejumlah bit pengganjal sedemikian sehingga panjang pesan (dalam satuan bit) kongruen dengan 448 modulo 512. Ini berarti panjang pesan setelah ditambahi bit-bit pengganjal adalah 64 bit kurang dari kelipatan 512. Panjang bit-bit pengganjal haruslah berada antara 1 hingga 512 bit. Hal tersebut menyebabkan pesan dengan panjang 448 tetap harus ditambahkan bit penyangga sehingga panjangnya akan menjadi 960 bit. Bit-bit pengganjal sendiri terdiri dari sebuah bit 1 diikuti sisanya dengan bit 0.

2. Penambahan nilai panjang pesan semula.

Pesan yang telah diberi bit-bit pengganjal selanjutnya ditambah lagi dengan 64 bit yang menyatakan panjang pesan semula. Setelah ditambah dengan 64 bit, panjang pesan akan menjadi kelipatan 512 bit.

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

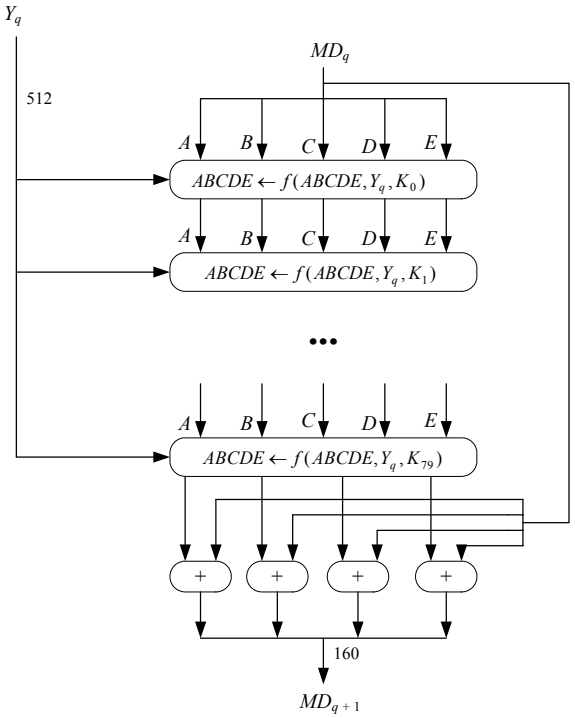
Algoritma SHA-1 dalam operasinya membutuhkan lima buah penyangga yang masing-masing besarnya 32 bit sehingga nantinya hasil akhirnya akan menjadi 160 bit. Kelima penyangga tersebut dalam operasi SHA-1 ini akan berperan untuk menyimpan hasil antara putaran sekaligus untuk menyimpan hasil akhir. Kelima penyangga tersebut memiliki nama

A, B, C, D ,dan E. Penyangga-penyangga tersebut harus diinisialisasi dengan nilai-nilai sebagai berikut (dalam notasi heksadesimal):
A= 67452301
B= EFCDAB89
C= 98BADCFE
D= 10325476
E= C3D2E1F0

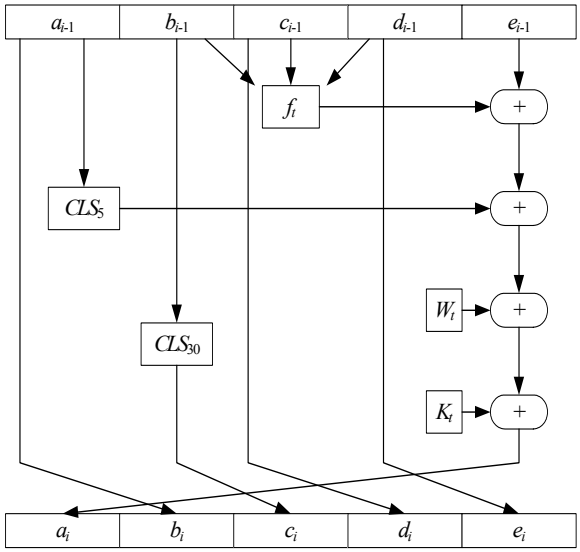
4. Pengolahan pesan dalam blok berukuran 512 bit.

Pesan dibagi menjadi L buah blok yang masing-masing panjangnya 512 bit. Setiap blok 512-bit diproses bersama dengan penyangga MD menjadi keluaran 128-bit. Proses tersebut disebut proses H_{SHA} .

Proses H_{SHA} terdiri dari 80 buah putaran dengan setiap putarannya dilakukan hal seperti gambar di bawah ini.



Dalam setiap putaran H_{SHA}, operasi yang dilakukan dapat diperlihatkan dalam gambar di bawah ini.



$a,b,c,d,e = (\mathrm{CLS}_5(a) + f_i(b,c,d) + e + W1 + K1)$, a, $\mathrm{CLS}_{30}(b)$, c, d

Penjelasan dari rumus di atas adalah sebagai berikut:

a,b,c,d,e = lima buah penyanga 32-bit yang berisi nilai-nilai A,B,C,D,E.
t = putaran, $0 \le t \le 79$
f_i = fungsi logika
CLS_s = circular left shift sebanyak s bit
W_t = word 32-bit yang diturunkan dari blok 512-bit yang sedang diproses
K_t= Konstanta penambah
+ = operasi penjumlahan dalam modulo 2^{32}

Fungsi dalam satu putaran H_{SHA} dapat dituliskan *pseudocode*-nya sebagai berikut:

```
For t := 0 to 79 do
     Tmp: = (a <<<5)   + ft(b,c,d) +
e + Wt + Kt)
      e :=d
      d:=c
      c:=b <<< 30
      b := a
      a:= tmp
endfor
```

Konstanta penambah dalam operasi H_{SHA} memiliki nilai berikut untuk masing-masing putaran, yaitu:

Putaran $0 \le t \le 19$	K_t=5A827999
Putaran $20 \le t \le 39$	K_t=6ED9EBA1
Putaran $40 \le t \le 59$	K_t=8F1BBCDC
Putaran $60 \le t \le 79$	K_t=CA62C1D6

Fungsi f_i adalah fungsi logika yang melakukan operasi *bitwise*. Operasi yang dilakukan dapat dilihat dalam tabel berikut ini:

Putaran	$f_t(b, c, d)$
0 – 19	$(b \wedge c) \vee (\sim b \wedge d)$
20 – 39	$b \oplus c \oplus d$
40 – 59	$(b \wedge c) \vee (b \wedge d) \vee (c \wedge d)$
60 – 79	$b \oplus c \oplus d$

Sementara itu, nilai W_0 sampai W_{15} berasal dari 16 *word* pada blok pesan yang sedang diproses sehingga masing-masing variabel *W* akan memiliki besar 32-bit, sedangkan nilai W_t berikutnya didapatkan dari persamaan

$$W_t = W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}$$

Fungsi di atas disebut juga sebagai fungsi kompresi dari SHA-1 karena fungsi tersebut berfungsi untuk melibatkan pesan masukan ke dalam proses putaran SHA-1. Selain itu fungsi di atas juga sering disebut sebagai fungsi *message expansion.* Di bagian-bagian selanjutnya dari makalah ini, nilai tersebut akan diacu sebagi nilai *message expansion*.

Setelah putaran ke-79, a, b, c, d, dan e dijumlahkan ke A, B ,C, D, dan E dan selanjutnya algoritma memproses blok data berikutnya. Keluaran akhir dari algoritma SHA-1 adalah hasil penyambungan bit-bit di dalam A, B, C, D, dan E.

4. Kriptanalisis SHA-1

Seperti disebutkan di bab 2, bahwa secara teori pencarian kolisi dari suatu fungsi *hash* mungkin dilakukan, yaitu dengan menggunakan metode serangan *brute-force*. Akan tetapi, apabila metode tersebut dilakukan akan terbentur dengan masalah kecepatan komputasi yang ada. SHA-1 sendiri apabila kolisinya hendak dicari dengan menggunakan metode serangan *brute-force* akan memiliki kompleksitas operasi sebesar 2^{80} atau bila dioperasikan dengan kluster super komputer sekalipun dapat memakan waktu jutaan tahun operasi.

Oleh karena itu, berbagai usaha untuk mencari kolisi dengan cara yang lebih efisien banyak dilakukan. Salah satunya dilakukan oleh Rijmen dan Oswald yang mampu mencari kolisi dari versi SHA-1 yang direduksi (dari 80 putaran dikurangi menjadi 58 putaran) dengan jumlah operasi kurang dari 2^{80}. Penemuan yang dilakukan oleh kedua orang ini tidak terlalu mengganggu dunia kriptografi karena menganggap tidak mengurangi nilai keamanan dari fungsi SHA-1.

Pada bulan Februari 2005, tiga orang peneliti dari Cina, Xiaoyun Wang, Yiqun Lisa Yin, dan Hongbo Yu, mempublikasikan bahwa mereka telah berhasil menemukan cara untuk melakukan pencarian kolisi fungsi SHA-1 dengan kompleksitas 2^{69}. Ketiga orang tersebut adalah sebuah tim riset yang sudah sangat bereputasi karena sebelumnya dapat mencari kolisi pada SHA-0 dengan kompleksitas 2^{39}. Walaupun kompleksitas operasi yang dibutuhkan masih sangat besar juga (dengan menggunakan komputer yang sama dengan contoh sebelumnya, akan dibutuhkan waktu sekitar 170.000 tahun, waktu yang tidak mungkin bagi manusia).

Meskipun demikian, hal ini cukup menjadi pembicaraan hangat dalam dunia kriptografi karena merupakan serangan pertama terhadap SHA-1 yang mampu menembus kompleksitas 2^{80}.

Metode yang dilakukan oleh ketiga orang tersebut adalah berdasarkan metode-metode untuk melakukan pencarian kolisi yang telah diaplikasikan kepada fungsi-fungsi *hash* lain seperti SHA-0 dan MD5. Akan tetapi, mereka memperbaharuinya sehingga mampu mencari kolisi SHA-1 dalam 2^{69}.

Berikut ini akan dijelaskan mengenai beberapa hal yang menjadi landasan teori dalam pencarian kolisi yang dilakukan oleh Xiaoyun Wang dkk. Hal-hal berikut ini adalah metode yang sebelumnya digunakan untuk mencari kolisi di fungsi SHA-1.

4.1 *Local Collision*

Local collision atau kolisi lokal adalah kolisi yang terjadi dalam beberapa putaran fungsi *hash*. SHA-0 memiliki kolisi lokal yang dapat terjadi dalam 6 putaran atau langkah dengan dimulai dari putaran manapun. Maksudnya adalah bila suatu nilai *i* diacu sebagai putaran awal untuk mencari kolisi dan nilai *i* adalah sembarang, maka dalam putaran i+6 akan dapat didapatkan kolisi lokal dari SHA-0. Dalam tabel di bawah

ini dapat dilihat bagaimana kolisi lokal yang terjadi dalam SHA-0.

	Δm	Δa	Δb	Δc	Δd	Δe	Δf()
i	0000 0001	0000 0001	0000 0000	0000 0000	0000 0000	0000 0000	
i+1	0000 0020	0000 0000	0000 0001	0000 0000	0000 0000	0000 0000	
i+2	0000 0001	0000 0000	0000 0000	4000 0000	0000 0000	0000 0000	0000 0001
i+3	4000 0000	0000 0000	0000 0000	0000 0000	4000 0000	0000 0000	4000 0000
i+4	4000 0000	0000 0000	0000 0000	0000 0000	0000 0000	4000 0000	4000 0000
i+5	4000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000

Keterangan:
Δm = Perbedaan nilai W_i antara dua pesan yang sedang dicoba dicari kolisi lokalnya, W sendiri adalah nilai 32 bit yang didapatkan dari fungsi kompresi/*message expansion* SHA-0.
Δa,b,c,d,e = Perbedaan antara nilai-nilai variabel MD dalam SHA-0 antara dua pesan yang sedang dicari kolisi lokalnya.
Δf() = Perbedaan nilai yang dihasilkan oleh fungsi boolean atau fungsi logika.

Dapat dilihat bahwa nilai variabel MD pada langkah i+5 memiliki perbedaan nilai 0, yang berarti nilai *message digest* untuk putaran tersebut untuk masing-masing pesan adalah sama (ingat bahwa nilai *hash* dari suatu pesan adalah hasil penyambungan string antara variabel-variable MD). Dari tabel di atas dapat terlihat juga bahwa kolisi lokal yang terjadi tidak bergantung kepada nilai *message expansion*. Perhatikan bahwa untuk nilai Δm yang terus-menerus meningkat justru dapat menghasilkan kolisi.

Kolisi lokal merupakan komponen dasar yang digunakan dalam pencarian kolisi ataupun *near collision* (kolisi yang terjadi hampir di seluruh bit *message digest*, misalnya terjadi 142 bit memiliki kesamaan nilai) fungsi SHA-0 dan SHA-1. Sama dengan SHA-0, dalam SHA-1 kolisi lokal yang terjadi juga tidak bergantung kepada perbedaan nilai dari *message expansion*.

4.2 *Differential Paths*

Differential paths didefinisikan sebagai kumpulan (*sequence*) dari beberapa kolisi lokal yang terjadi dalam fungsi *hash*. Untuk dapat mengonstruksi sebuah *differential path*, harus dilakukan pencarian nilai putaran pertama dari terjadinya suatu kolisi lokal. Untuk menspesifikasikan *starting point* tersebut, dapat digunakan sebuah vektor yang memiliki panjang 80 dan masing-masing elemennya berisikan nilai bit 0/1. Vektor tersebut dapat disebut sebagai *disturbance vector*. Vektor tersebut harus memenuhi persamaan fungsi *message expansion* dari fungsi *hash*, baik SHA-0 maupun SHA-1.

Dalam pencarian kolisi untuk SHA-0, ada tiga syarat yang harus dipenuhi oleh sebuah *disturbance vector* agar mampu menuju kepada terjadinya sebuah kolisi lokal, yaitu:

1. $x_i = 0, i \in \{75,76,77,78,79\}$
2. $x_i = 0, i \in \{-5,-4,-3,-2,-1\}$
3. Tidak nilai bit 1 untuk 17 nilai pertama x_i

Syarat-syarat tersebut dapat diimplementasikan kepada pencarian *disturbance vector* dalam pencarian kolisi di SHA-1. Pada pencarian *disturbance vector* SHA-0, isi masing-masing elemen adalah bit 0/1. Rumus penghitungan nilai *message expansion* dari SHA-0 mengisyaratkan bahwa nilai W_{16}-W_{79} akan bergantungan dari nilai W_0-W_{15}. Oleh karena itu, nilai elemen dari *disturbance vector* yang harus digunakan adalah sebanyak 16-bit yang berarti ruang pencarian untuk nilai-nilai tersebut adalah 2^{16}.

Akan tetapi, *disturbance vector* untuk SHA-1 berbeda. Alih-alih menggunakan nilai bit 0/1, *disturbance vector* di sini menggunakan nilai 32-bit. Dengan aturan *message expansion* yang mirip dengan SHA-1, dapat disimpulkan bahwa ruang pencarian untuk 16 nilai pertama *disturbance vector* adalah 2^{512}. Itu merupakan sebuah ruang pencarian yang sangat besar sehingga membuat pencarian kolisi memiliki kompleksitas yang besar. Xiaoyun Wang dkk memperkenalkan metode baru yang dapat membuat ruang pencarian menjadi kecil. Hal ini akan dijelaskan di sub-bab berikutnya.

Dalam memilih sebuah *disturbance vector* tentunya ada kriteria apakah sebuah *disturbance vector* yang dipilih sudah cukup bagus atau tidak, tentunya dengan pertama kali memenuhi syarat wajib dari suatu *disturbance vector*. Bagus tidaknya suatu *disturbance vector* dapat diukur dengan nilai *Hamming Weight*-nya. *Hamming Weight* secara singkat adalah banyaknya nilai bit 1 dalam suatu deretan nilai. Untuk *disturbance vector*, semakin sedikti nilai 1-nya berarti semakin baik pula *disturbance vector* tersebut untuk digunakan untuk mencari kolisi.

4.3 *Message Modification*

Teknik *message modification* adalah teknik yang digunakan untuk melakukan modifikasi pesan sehingga dapat mendekati terjadinya kolisi. Seperti disebutkan di bab mengenai SHA-1, SHA-1 adalah fungsi algoritma yang berdasarkan kepada MD-4. Salah satu inti dari fungsi *hash* yang berdasarkan MD-4 adalah bahwa nilai dari *chaining variables* (dalam SHA-1 adalah nilai variabel a, b, c, d, dan e) adalah berdasarkan nilai *chaining variables* di putaran sebelumnya dan berdasarkan nilai blok pesan masukan yang akan dicari nilai *hash*-nya. Hal tersebut dapat dirumuskan sebagai berikut:

$$a_i = F(\textit{input chaining vars}, m_{i-1})$$

Dari *differential path* yang telah didapatkan di langkah sebelumnya, penurunan kondisi dapat dilakukan. Penurunan kondisi di sini adalah membandingkan nilai antara *chaining variables*. Secara sederhana dapat digambarkan berikut:

$$a2 = 1 \text{ atau } a5 = \neg\, a3$$

Dari kondisi tersebut, hal yang kemudian dapat dilakukan adalah mengubah salah bit dalam blok m_{i-1} (m adalah *message expansion*) sehingga nantinya *chaining variables* nilainya dapat berubah yang tentunya akan mendekatkan kepada terjadinya suatu kolisi. Untuk pengubahan bit yang dilakukan pada i=0 hingga i=15 hal ini akan mudah dilakukan karena nilai-nilainya tidak bergantung kepada nilai lainnya dan hanya bergantung kepada nilai blok pesan. Sementara itu, untuk bit (*word* dalam SHA-1) i=16 sampai dengan i=79 hal tersebut tidak akan semudah itu dilakukan karena nilai-nilai tersebut didapatkan dari i=0 hingga i=15 sehingga apabila terjadi pengubahan di bit-bit tersebut akan banyak bit yang terpengaruh.

4.4 Metode Pencarian Kolisi oleh Xiaoyun Wang dkk.

Seperti yang disebutkan sebelumnya bahwa metode pencarian kolisi yang dilakukan terhadap SHA-1 adalah berdasarkan metode yang dilakukan terhadap SHA-0. Metode yang dulunya digunakan terhadap SHA-0 di-*improve* sehingga mampu mencari kolisi SHA-1 dalam kompleksitas 2^{69}.

Secara garis besar, teknik-teknik yang digunakan untuk mencari kolisi dalam SHA-1 yang diterapkan oleh Wang dkk. adalah sebagai berikut:

1. Mencari *disturbance vectors* dengan *Hamming Weight* yang rendah
2. Mengonstruksi *differential paths* yang akan digunakan untuk mencari kolisi.
3. Menurunkan kondisi dari *chaining variables* dan pesan sehingga dapat dijadikan dasar dalam melakukan modifikasi pesan.
4. Melakukan modifikasi pesan yang sesuai.
5. Mencari kolisi dengan menggunakan *near-collision*.

Berikut akan dijelaskan secara lebih rinci mengenai langkah-langkah tersebut.

4.4.1 Mencari *disturbance vectors* dengan *Hamming Weight* yang rendah

Disturbance vector adalah suatu vektor yang akan menspesifikasikan titik awal untuk terjadinya kolisi lokal (*local-collision*). Kolisi lokal sendiri adalah kolisi yang terjadi dalam beberapa putaran dalam fungsi *hash*. *Disturbance vector* dapat direpresentasikan sebagai sebuah kumpulan nilai x_0, x_1, x_2,…, x_{78}, x_{79}. Untuk SHA-0 masing-masing nilai x adalah nilai bit, yaitu 0/1. Sementara untuk SHA-1, nilai-nilai x adalah sebesar 32-bit. Pencarian *disturbance vector* dalam melakukan pencarian kolisi adalah langkah yang sangat penting.

Disturbance vector yang harus dicari adalah sebanyak 32 sehingga *disturbance vectors* dapat dipandang sebagai sebuah matriks berukuran 80x32. Pencarian *disturbance vector* dalam usaha pencarian kolisi dalam SHA-0 harus memenuhi tiga syarat seperti yang disebutkan di bab sebelumnya. Akan tetapi, bila hal tersebut diterapkan kepada SHA-1, akan sulit dipenuhi untuk mencari vektor yang memiliki nilai Hamming *Weight* yang rendah.

Oleh karena itu, Wang dkk. memiliki metode baru untuk mencari *disturbance vector* ini. Hal yang mereka lakukan adalah dengan menghilangkan ketiga syarat tersebut untuk mencari sebuah *disturbance vector*. Hal ini menyebabkan ruang pencarian *disturbance vector* menjadi sangat besar, yaitu 2^{512}. Untuk menghemat waktu pencarian, digunakanlah teknik heuristik. Heuristik yang mereka ambil adalah bahwa dari hasil pengamatan terhadap vektor-vektor yang memiliki nilai *hamming* yang rendah, ternyata nilai-nilai yang *non-zero*

berkonsentrasi di kolom-kolom yang berurutan. Dari hal tersebut, pemilihan nilai langsung dapat menggunakan dua kolom. Hal ini akan membuat ruang pencarian menjadi 2^{38}.

4.4.2 Mengonstruksi *differential paths* yang akan digunakan untuk mencari kolisi

Differential path merupakan hal yang sangat fundamental dalam mencari kolisi. Oleh karena itu, pengonstruksian *differential paths* yang baik sangat diperlukan. Xiaoyun Wang dkk. juga mengakui bahwa bagian ini merupakan bagian yang paling rumit dan sulit dibandingkan dengan bagian-bagian lainnya.

Seperti disebutkan di bagian sebelumnya, bahwa *differential paths* akan didapatkan dari *disturbance vector*. Oleh karena *disturbance vector* tidak lagi mengikut syarat-syarat seperti halnya yang dilakukan terhadap SHA-0, penentuan *differential paths* yang baik menjadi lebih sulit. Xiaoyun Wang dkk. kemudian menggunakan teknik-teknik baru untuk menentukan *differential paths*. Secara garis besar teknik-teknik tersebut adalah sebagai berikut:

1. Menggunakan operasi pengurangan daripada operasi *exclusive-or* sebagai pengukuran perbedaan dua buah nilai.
2. Menggunakan perbedaan pesan yang berbeda untuk enam langkah kolisi lokal.

4.4.3 Menurunkan kondisi dari *chaining variables* dan blok pesan

Setelah mendapat *differential path* yang valid dari langkah sebelumnya, langkah selanjutnya adalah menurunkan kondisi pesan seperti yang dijelaskan di bab sebelumnya.

Teknik yang digunakan untuk menurunkan kondisi pesan dan *chaining variables* tidak banyak berbeda dengan yang dilakukan terhadap SHA-0. Pada SHA-1, pada fungsi *message expansion*-nya terjadi *bit shifting* sebesar 1. Hal tersebut menyebabkan *disturbance* dapat terjadi di bit-bit lain selain bit 2, sementara untuk SHA-0, *disturbance* hanya terjadi di bit 2. Jika tidak ditangani dengan baik, hal ini akan menyebabkan terjadinya ledakan jumlah kondisi. Cara yang Xiaoyun Wang dkk. tempuh adalah dengan cara jika ada *disturbance* untuk bit 1 dan bit 2 dari x_i, perbedaan pesan Δm_i sebagai kebalikan dari bit-bit tersebut. Dengan menggunakan teknik tersebut, akan didapatkan *path* dengan 71 kondisi dan memproduksi *near-collision*.

4.4.4 Melakukan modifikasi pesan yang sesuai

Berdasarkan kondisi pesan dan *chaining variables* yang didapatkan di langkah sebelumnya, hal yang berikutnya harus dilakukan adalah melakukan modifikasi pesan. Dengan menggunakan teknik yang dijelaskan di bab sebelumnya, pesan masukan dapat dimodifikasi sehingga seluruh kondisi dari *chaining variables* bisa terpenuhi dalam 16 langkah. Dengan usaha tambahan, modifikasi pesan dapat dilakukan sehingga kondisi pada langkah 17 hingga 22 juga dapat terpenuhi.

4.4.6 Mencari kolisi dengan menggunakan *near-collision*

Setelah didapatkan *path* yang dapat menghasilkan *near-collision*, hal yang berikutnya adalah mengombinasikan dua blok tersebut menjadi sebuah *multi-block collision*. Misalnya

$\Delta h_1 = H(M'_0, IV) - H(M_0, IV)$

$\Delta h_2 = H(M'_1, h'_1) - H(M_1, h_1)$

Gunakan teknik konstruksi untuk menyerap Δh_1 selama 16 langkah dari *hash* kedua. Kemudian, set kondisi pada M_1 sehingga $\Delta h_2 = - \Delta h_1$. Dengan demikian, kompleksitas dari *near-collision* yang pertama sama dengan kompleksitas dari *near-collision* yang kedua.

5. Kesimpulan

Hasil dari penelitian dari Xiaoyun Wang dkk. tentang pencarian kolisi dalam algoritma *hash* SHA-1 memang cukup mengejutkan dunia kriptografi karena mampu secara signifikan mengurangi kompleksitas pencarian dari 2^{80} menjadi 2^{69}. Walaupun angka itu tetap tidak mungkin dicari dengan komputer yang ada sekarang ini, penemuan tersebut menyadarkan dunia kriptografi untuk menggunakan algoritma yang lebih aman. Sebagai informasi, ketika tulisan ini ditulis, ternyata Xiaoyun Wang telah mampu meningkatkan algoritmanya sehingga mampu melakukan operasi hanya dalam 2^{63}. Para praktisi dunia kriptografi pun saat ini sudah tidak menyarankan penggunaan SHA-1 dan menyarankan agar untuk saat mendatang lebih baik menggunakan fungsi-fungsi *hash* yang lebih baru, seperti yang ditawarkan di algoritma SHA-

2. Dari hal tersebut, bisa disimpulkan bahwa penggunaan SHA-1 tidak lagi sepenuhnya aman karena diperkirakan akan muncul penemuan lain yang lebih signifikan.

Referensi

[1] http://www.heise-security.co.uk/articles/75686. Terakhir diakses tanggal 30 Januari 2006.

[2] http://www.schneier.com/blog/archives/2005/02/sha1_broken.html. Terakhir diakses tanggal 30 Januari 2006.

[3] http://www.systemexperts.com/tutors/CryptographicHashUpdate.pdf

[4] Munir, Rinaldi. 2005. *Diktat Kuliah IF5054 Kriptografi*. Program Studi Teknik Informatika, Institut Teknologi Bandung.

[5] Wang, Xiaoyun. 2005. *Finding Collisions in the Full SHA-1*. Shandong University, Cina.