Analisis *Birthday Attack* untuk Menemukan *collision* pada Algoritma *Hash* MD5

Febrian Aris Rosadi – NIM 13503041
Program Studi Teknik Informatika Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
Email : if13041@students.if.itb.ac.id

Abstraksi:

Fungsi *hash* adalah salah satu bagian penting dari protokol keamanan dalam kriptografi. Fungsi *hash* menjadi suatu sarana dalam melakukan uji integritas dan verifikasi data.

Beberapa jenis algoritma diajukan untuk memenuhi kebutuhan akan fungsi *hash* antara lain MD2, MD4, MD5, SHA-0, SHA-0, RIPEMD dan beberapa fungsi lain namun untuk saat ini yang sering digunakan adalah algoritma MD5 dan SHA-1. Namun pengunaan ini harus dievaluasi lagi karena melalui beberapa proyek penelitian kriptanalisis, para kriptanalis berhasil menemukan serangan *collision* terutama terhadap MD5. Kerentanan ini patut dijadikan bahan acuan dalam menjadikan algoritma MD5 bagian dari suatu protokol keamanan. Serangan yang dikembangkan salah satunya adalah *birthday attack* berdasarkan permasalahan dalam ilmu statistika yang lebih dikenal sebagai *birthday paradox.*

Kata Kunci: *Birthday Paradox, Birthday Attack*, Fungsi *Hash*, Kriptanalisis MD5

1 Pendahuluan

Terdapat beberapa permasalahan dalam kriptografi berkaitan dengan masalah keamanan yang tidak dapat diselesaikan dengan fungsi enkripsi dan dekripsi biasa. Masalah keamanan yang harus dihadapi itu antara lain masalah verifikasi dan integritas data. Fungsi *hash* adalah salah satu fungsi dalam kriptografi yang mampu memenuhi kebutuhan akan verifikasi dan integritas data. Misalnya, fungsi *hash* bersama algoritma enkripsi kunci publik telah digunakan sebagai fitur dalam *digital signature*, selain itu fungsi hash juga digunakan dalam penyimpanan *password*.

Dalam peranan penting yang diemban fungsi *hash* untuk aspek keamanan, fungsi *hash* harus memiliki ketahanan terhadap serangan-serangan. Namun kenyataannya, untuk beberapa jenis algoritma yang digunakan sebagai fungsi *hash*, telah diketahui rentan terhadap serangan tertentu yang diteliti dan dikembangkan oleh para kriptanalis. Algoritma yang telah diketahui rentan terhadap serangan itu salah satunya adalah MD5.

MD5 telah diketahui dapat diserang dengan suatu serangan yang dikembangkan berdasarkan *birthday attack.* Dalam makalah ini akan dibahas bagaimana *birthday attack* tersebut dikembangkan terutama dalam beberapa pendekatan matematis yang digunakan. Selain itu juga terdapat pembahasan tambahan mengenai aspek-aspek yang perlu diperhatikan dalam keputusan untuk berpindah dari algoritma MD5 dan algoritma fungsi *hash* lainnya secara umum, ketika algoritma tersebut telah lemah terhadap suatu serangan.

2 Fungsi *Hash*

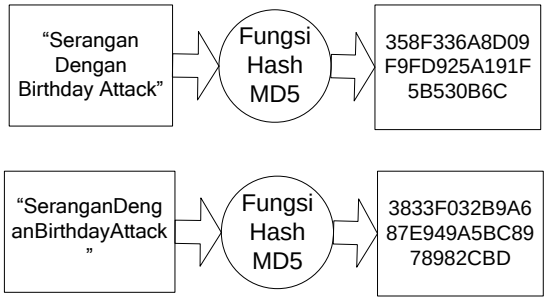
Fungsi *hash* adalah fungsi yang menerima masukan string yang panjangnya sembarang dan mengkonversinya menjadi string keluaran dengan panjang tetap yang umumnya lebih pendek dari panjang string semula [1]. Nama lain dari fungsi *hash* adalah : fungsi kompresi/kontraksi (*compression function*), cetak-jari (*fingerprint*), *cryptographic checksum*, *message integrity check (MIC)*, *manipulation detection code (MDC).*

Fungsi *hash* akan menerima masukan pesan (*message*) dan menghasilkan pesan ringkas (*message digest*) atau disebut juga nilai *hash* (*hash value*). Pesan yang menjadi masukan fungsi *hash* dapat berupa string dengan panjang apa saja. Namun hasil

keluaran pesan ringkas dari fungsi *hash* adalah pesan ringkas yang panjangnya selalu sama untuk tiap fungsi, umumnya selalu lebih pendek dari panjang pesan semula. Dapat dikatakan sembarang pesan *m* berukuran bebas dikompresi oleh fungsi *hash f* menjadi pesan ringkas *h* melalui persamaan

$$h = f(m)$$

Syarat lain dari fungsi *hash* adalah fungsi *hash* harus dapat memberikan nilai *hash* yang unik untuk tiap pesan yang berbeda-beda. Dua pesan yang berbeda tidak dapat menghasilkan pesan ringkas yang sama kecuali dua pesan tersebut merupakan dua pesan yang benar-benar sama.



Gambar di atas akan memberikan gambaran bagaimana fungsi *hash* akan bekerja. Dua buah pesan yang berbeda-beda akan menghasilkan pesan ringkas yang panjangnya tetap.

Dalam contoh ini fungsi *hash* yang digunakan adalah MD5, salah satu jenis fungsi *hash* yang terkenal, pesan ringkas yang dihasilkan memiliki panjang tetap 128 bit (16 byte karakter) dinyatakan dalam karakter heksadesimal yaitu tiap karakter heksadesimal akan menyimpan 4 bit dari pesan ringkas, sehingga seluruh karakter heksadesimal yang dihasilkan berjumlah 32 karakter. String masukan pertama hanya berbeda dengan string masukan kedua dengan tidak adanya spasi tiap kata. Tentu saja string kedua akan memiliki panjang kurang dari string pertama. Namun perbedaan yang kecil itu mengubah secara drastis pesan ringkas hasil keluaran fungsi *hash* sehingga tidak ada kesamaan yang mencolok antara kedua keluaran. Hal ini lah yang menjadi syarat fungsi *hash* bahwa untuk pesan yang berbeda maka keluarannya akan berbeda jauh sehingga secara komputasi tidak dapat dilacak perbedaan pesan dari kedua keluaran tersebut.

Dilihat dari mekanisme fungsi *hash*, Terdapat fungsi *hash* satu-arah yaitu yang bekerja dengan mengubah pesan semula menjadi pesan ringkas (*message digest)* yang tidak dapat dikembalikan menjadi pesan semula. Fungsi *hash* yang dibahas di makalah ini adalah fungsi *hash* satu-arah sehingga untuk berikutnya sebutan fungsi *hash* akan mengacu pada fungsi *hash* satu-arah.

3 Aplikasi Fungsi *Hash* dalam Verifikasi dan Integritas Data

Aplikasi fungsi *hash* misalnya untuk meverifikasi kesamaan salinan suatu arsip dengan arsip aslinya yang tersimpan di dalam sebuah basisdata terpusat. Ketimbang mengirim salinan tersebut secara keseluruhan, akan lebih efisien untuk mengirimkan pesan ringkas dari pesan tersebut. Jika pesan ringkas yang dikirim tersebut sama dengan pesan ringkas dari dokumen yang akan diverifikasi, berarti salinan tersebut sam dengan arsip yang ada di dalam basis data. Fungsi *hash* bersama algoritma enkripsi kunci publik juga digunakan sebagai fitur dalam *digital signature*.

Fungsi *hash* menjadi bagian penting dalam pemberian tanda-tangan digital, termasuk dalam proses verifikasi tanda-tangan digital tersebut. Proses pemberian tanda-tangan digital dapat dijelaskan seperti di bawah ini.

Pengirim pesan mula-mula menghitung pesan ringkas *m'* dari pesan *m* (dokumen atau data yang akan ditandatangani), dengan menggunakan salah satu fungsi *hash f*.

$$m' = f(m)$$

Kemudian pesan ringkas *m'* tersebut dienkripsi dengan algoritma kriptografi kunci publik *e* dengan menggunakan kunci privat *sk* pengirim. Hasil enkripsi inilah yang menjadi tanda-tangan digital *s*.

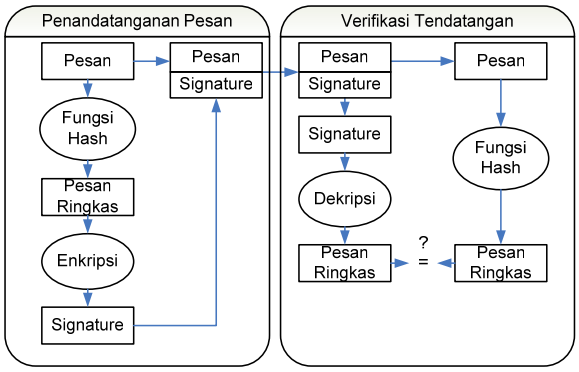
$$e' = e_{sk}(m')$$

Tanda tangan *s* ini akan dilekatkan ke pesan m dengan cara menyambung (*append*) ke pesan *m*. Hasil penyambungan ini yang akan dikirim dan dikatakan telah ditanda-tangani oleh pengirim dengan tanda-tangan digital *s*.

Sedangkan dalam proses verifikasinya, otentikasi data diuji dengan cara mendeskripsikannya dengan kunci publik *pk* pengirim pesan. Bila tanda tangan tersebut otentik, maka proses deskripsi ini akan menghasilkan pesan ringkas semula, *m'*.

$$m' = d_{pk}(s)$$

Langkah kedua dalam proses verifikasi adalah membangkitkan nilai *hash* dari pesan m yang dikirim bersama tanda-tangan digital *s* menjadi suatu pesan ringkas *m''*, dengan fungsi *hash f* yang sama yang digunakan dalam proses pemberian tanda-tangan digital. Jika pesan yang dikirim bersama tanda-tangan masih sama dengan pesan asli ketika pesan tersebut ditandatangani , maka *m''* akan sama dengan *m'* hasil deskripsi sebelumnya. Inilah yang akan diuji di langkah terakhir dalam proses verifikasi tanda-tangan digital yaitu jika *m'' = m'* maka pesan yang dikirim beserta tanda-tangannya adalah otentik.



Selain dalam *digital signature*, fungsi *hash* ini juga digunakan dalam MAC. Sebagai deskripsi singkatnya, pengirim pesan akan bertukar kunci rahasia *k* dengan penerima pesan. Pengirim kemudian menyambung pesan *m* dengan kunci *k*, lalu mengitung pesan ringkas dari hasil penyambungan tersebut, yaitu *f(m,k).* Inilah yang akan menjadi MAC dari pesan. Ketika penerima memperoleh pesan dan MAC tersebut, penerima dapat melakukan otentikasi terhadap pesan karena ia dapat menghitung pesan ringkas yang sama dari kunci *k* yang ia peroleh.

Aplikasi fungsi *hash* yang tidak kalah pentingnya adalah dalam penyimpanan *password*. Suatu data *password* pengguna dalam suatu sistem adalah data yang sangat rahasia dan tidak boleh diketahui pihak yang tidak berkepentingan untuk menjaga keamanan sistem tersebut. Namun tanpa perlindungan yang cukup, seorang penyusup dapat masuk membobol basis data dan memperoleh semua data. Walaupun begitu penyusup tersebut tidak boleh memperoleh data *password* , disinilah fungsi *hash* menjadi benteng terakhir untuk perlindungan data *password* tersebut. Fungsi *hash* akan mengubah suatu nilai *password* menjadi suatu pesan ringkas yang bersesuaian. Pesan ringkas inilah yang akan disimpan dalam database sebagai data *password*. Sehingga, suatu pihak yang mengetahui data *password* ini tidak akan mendapatkan pengetahuan apapun ketika memperoleh data *password* tersebut. Nilai pesan ringkas tersebut tidak dapat dibalik untuk memperoleh nilai *password* sebenarnya. Sebaliknya dalam melakukan verifikasi pengguna, proses yang terjadi adalah pencocokkan pesan ringkas yang dihasilkan oleh fungsi *hash* terhadap *password* masukan pengguna, dengan pesan ringkas yang disimpan dalam basis data. Jika cocok, maka verifikasi pengguna berhasil karena tidak ada nilai lain yang dapat menghasilkan pesan ringkas tersebut selain *password* asli.

4 Resiko Keamanan terhadap Fungsi *Hash*

Ada 3 properti fundamental yang harus dipenuhi oleh fungsi *hash* :

1) *Pre-image Resistance* (dinamakan juga *non-invertibility*).
   Diketahui suatu nilai pesan ringkas *h*, secara komputasi tidak boleh ditemukan cara untuk mendapatkan pesan *x* yang bersesuaian dengan pesan ringkasnya jika $f(x) = h$.
2) *Second Pre-image Resistance.*
   Diberikan suatu nilai pesan *x*, secara komputasi, tidak boleh ditemukan pesan *x'* sehingga $f(x) = f(x')$.
3) *Collision resistance.*
   Secara komputasi, tidak boleh ditemukan cara untuk menemukan lebih dari satu pesan *x,y* dimana $x \neq y$ dan menghasilkan pesan ringkas yang sama yaitu $f(x) = f(y)$.

Ilustrasi resiko keamanan yang dapat terjadi sebagai berikut:

Dalam beberapa, kondisi isi pesan asli dapat diketahui oleh orang banyak, karena dalam beberapa protokol yang menggunakan fungsi *hash* seperti *digital signature*, tidak dipentingkan aspek kerahasiaan pesan namun yang penting adalah aspek otentikasi pesan tersebut. Bila isi pesan asli diketahui, maka dari pesan tersebut tidak boleh secara komputasi mampu dihasilkan pesan lain yang dapat menghasilkan pesan ringkas yang sama (*Second Pre-image Resistance*).

Aspek *Collision Resistance* jika tidak dipenuhi akan mengurangi secara drastis kekuatan fungsi *hash* dalam melakukan otentikasi terhadap *digital signature*. Dalam prosesnya ketika pihak penguji

melakukan pengujian otentikasi dengan membangkitkan pesan ringkas dari pesan yang dikirim,dan membandingkannya dengan pesan ringkas yang dikirim bersama dengan pesan, otentikasi pesan tidak dapat dijamin tanpa jaminan bahwa tidak ada pesan lain yang berbeda yang akan menghasilkan pesan ringkas yang sama. Setidaknya kemungkinan itu haruslah sangat kecil hingga mendekati tidak mungkin terjadi.

Penyimpanan *password* memanfaatkan aspek *non-invertibility* atau *Pre-image Resistence* yang dimiliki oleh sebuah fungsi *hash*,, sehingga data *password* yang disimpan tidak mungkin dapat dibalik untuk memperoleh data *password* sebenarnya. Sementara verifikasi *password* memanfaatkan aspek lain dari fungsi *hash* yaitu *collision resistance* dimana penyusup tidak dapat memperoleh akses masuk dengan menggunakan nilai *password* lain karena tidak ada nilai *password* lain selain *password* yang asli yang dapat menghasilkan pesan ringkas yang disimpan dalam data *password*.

Serangan generasi sekarang diarahkan pada menyerang prinsip *collision resistance*. Algoritma MD5 secara efektif telah tidak bisa digunakan dilihat dari perspektif ini.

Algoritma MD5 yang dibuat oleh Ronald Rivest terkenal dan banyak digunakan karena kehandalannya yang dikatakan telah terbukti. Tetapi, pada tahun 2004 sebuah proyek yang bernama MD5CRK memperlihatkan bahwa MD5 tidak aman karena ditemukan kolisi dengan *birthday attack*. Sebuah *birthday attack* adalah sebuah tipe serangan kriptografik, yang menerapkan pengetahuan matematik di belakang *birthday paradox*.

5 Algoritma MD5

Algoritma MD5 dibuat oleh Ronald Rivest pada tahun 1991, sebagai perbaikan dari algoritma MD4 yang telah diketahui rentan diserang oleh kriptanalisis. Algoritma ini menerima pesan dengan ukuran sembarang dan menghasilkan pesan ringkas dengan panjang 128 bit.

Secara umum algoritma MD5 membangkitkan pesan ringkas bekerja dengan cara :

1. Menambahkan bit-bit pengganjal;
2. Menambahkan nilai panjang pesan semula;
3. Inisialisasi penyangga (*buffer*);
4. Pengolahan pesan dalam blok berukuran 512 bit.

Pertama kali akan dilakukan penambahan bit pengganjal. Bit-bit pengganjal ditambahkan pada pesan sehingga panjangnya dapat dibagi dengan 512. Angka 512 ini muncul karena MD5 mengolah pesan dalam blok-blok yang berkuran 512 bit. Sebuah bit 1 akan disambungkan di akhir pesan yang akan ditambahkan dengan 0 sebanyak yang dibutuhkan agar panjang pesan adalah kurang 64 dari suatu kelipatan 512. Sisa dari 64 tersebut akan diisi dengan sebuah nilai integer yang merepresentasikan panjang asli pesan tersebut.

Algoritma utama MD5 akan beroperasi dengan menggunakan state 128 bit yang dibagi ke dalam empat 32-bit penyangga *(buffer).* Dinotasikan sebagai penyangga A, B, C dan D, yang akan diinisialisasi dengan konstanta tertentu. Untuk versi MD5 yang umum buffer tersebut diinisialisasi sebagai berikut

```
A = 01234567
B = 89ABCDEF
C = FEDCBA98
D = 76543210
```

Pesan kemudian akan dibagi menjadi sejumlah blok 512 bit. Algoritma kemudian akan beroperasi di tiap 512 bit blok dari pesan, tiap blok akan memodifikasi penyangga. Pemrosesan untuk tiap blok pesan terdiri dari 4 tingkat yang mirip yang dinamakan *round* . Tiap *round* terdiri dari 16 operasi yang mirip berdasarkan sebuah fungsi non linear F, *modular addition*, dan *left rotation*. Fungsi dibawah mengilustrasikan ssatu operasi didalam sebuah *round.* Terdapat 4 kemungkinan fungsi F, yang masing-masing akan digunakan di tiap *round.*

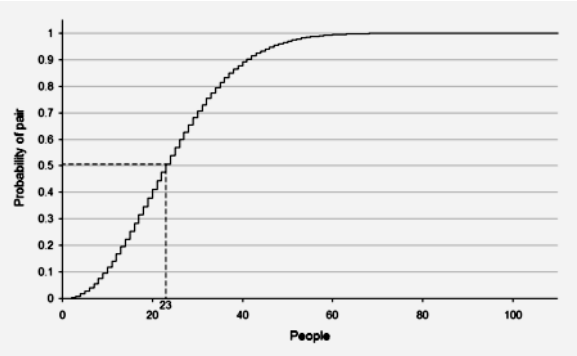
$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \wedge \neg Z)$$

$\wedge, \vee, \oplus, \neg$ menyatakan masing-masing operator AND, OR, XOR, dan NOT.

Yang patut menjadi perhatian utama di algoritma MD5 dalam masalah *birthday attack* adalah pesan dengan panjang sembarang akan selalu menghasilkan pesan ringkas dengan panjang 128, sehingga jumlah kemungkinan keluaran yang harus dihasilkan adalah 2^{128} kemungkinan pesan ringkas unik.

6 *Birthday Paradox*

Dalam teori probabilitas, *birthday paradox* menyatakan bahwa disediakan sekelompok orang yang terdiri dari 23 atau lebih yang dipilih secara acak, kemungkinannya lebih dari 50% bahwa dua orang dari mereka akan memiliki hari yang tahun yang sama.



Secara intuitif, untuk menerima *birthday paradox* adalah dengan menyadari bahwa mengetahui cara menghitung kemungkinan sepasang manusia dapat memiliki ulang tahun yang sama. Secara spesifik misalnya, di antara 23 orang, akan terdapat C(32,2) = 23 x 22 / 2 = 253 pasang, dimana setiap pasang merupakan kandidat untuk ditemukannya pasangan yang cocok tanggal ulang tahunnya.

Kunci untuk memahami permasalahan ini adalah dengan berpikir bahwa peluang tidak ada dua orang yang memiliki ulang tahun yang sama adalah peluang bahwa orang ke-1 akan memiliki ulang tahun yang berbeda dengan orang ke-2 dan akan berbeda dengan orang ke-3 dan seterusnya. Semakin banyak orang yang dilibatkan, peluang tidak adanya dua orang dengan tanggal ulang tahun yang sama akan semakin kecil.

Untuk menghitung kemungkinan terdekat bahwa didalam sebuah ruang dengan jumlah peserta *n* orang akan ada dua orang yang berbagi tanggal ulang tahun, kita akan mengabaikan variasi tahun kabisat, terdapat kembar, dan asumsi bahwa 365 tanggal akan memiliki kemungkinan yang sama. Di dalam kehidupan nyata, tanggal-tanggal dalam 365 hari tdak akan memiliki kemungkinan yang sama.

Pertama akan kita hitung peluang $\bar{p}(n)$ yaitu semua *n* orang di dalam ruang tidak akan memiliki tanggal ulang tahun yang sama. Peluang dapat dihitung sebagai berikut :

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

$$\bar{p}(n) = \frac{365.364.363...(365-n+1)}{365}$$

$$\bar{p}(n) = \frac{365!}{365^n\,(365-n)!}$$

Orang ke-2 harus memiliki tanggal ulang tahun yang berbeda dengan orang ke-1 dengan peluang kejadian 364/365 , orang ke-3 harus memiliki tanggal ulang tahun yang berbeda dengan dua orang pertama dengan peluang kejadian 363/365, dan seterusnya.

Peluang setidaknya dua orang dari kumpulan tersebu berbagi tanggal ulang tahun yang sama adalah komplemen dari peluang tanggal ulang tahun semua orang akan berbeda, yaitu :

$$p(n) = 1-\bar{p}(n)$$

$$p(n) = 1-\frac{365!}{365^n\,(365-n)!}$$

Dari perhitungan di atas ternyata nilai *p(n)* akan melebihi 50% ketika *n*=23. Untuk nilai jelasnya akan ditampilkan di tabel berikut :

n	p(n)
10	11.69 %
20	41.14 %
23	50.73 %
30	70.63%
50	97.04 %
57	99.01%
100	99.99996927510721 %
200	99.99999999999999999999999999839 %
366	100 %

Berdasarkan *pigeonhole principle*, jika *n*=366 maka peluang *p(n)* adalah 100%. Dilihat dari tabel di atas, bahkan dengan *n*=57, maka peluang akan terdapat setidaknya dua orang akan memiliki ulang tahun yang sama dapat mencapai 99% kemungkinan terjadi.

7 Pendekatan Lain dalam *Birthday Paradox*

Pendekatan lain dapat digunakan untuk menghitung kemungkinan dalam *birthday paradox* ini, seperti dalam pendekatan permutasi string (*string permutation*) seperti ditunjukkan di bawah ini.

Setiap hari dapat direpresentasikan sebagai sebuah karakter unik di 365-karakter alfabet. Sebuah string dengan panjang *n* (diambil dari kumpulan alphabet tersebut) dapat digunakan untuk merepresentasikan hari ulang tahun dari semua *n* orang. Kemungkinan *p(n)* dapat dihitung dengan memperhatikan semua string yang dapat dibentuk sesuai penjelasan di atas. Jumlah totalnya adalah

$$n_{total} = 365^n$$

Jika mencari jumlah total dari string yang tidak mengandung pengulangan karakter (untuk $n \le 365$), maka kita dapat menghitung dengan menggunakan *n*-permutasi dari 365 karakter tersebut

$$n_{unik} = \frac{365!}{(365-n)!}$$

Jika string diambil secara acak dari seluruh kemungkinan string yang dapat dihasilkan, kemungkinan bahwa sebuah string akan memiliki karakter yang berulang adalah

$$p(n) = 1 - \frac{n_{total}}{n_{unik}} = \frac{\frac{365!}{(365-n)!}}{365^n}$$

Rumus di atas akan kembali ke pendekatan pertama yang digunakan menjadi rumus

$$p(n) = 1 - \frac{365!}{365^n (365-n)!}$$

Selain dengan permutasi string, terdapat pendekatan dengan menggunakan *poison approximation* di bawah ini.

Pada dasarnya, kemungkinan untuk dua orang sembarang tidak memiliki hari ulang tahun yang sama adalah 364/365. Di dalam sebuah ruangan dengan sejumlah *n* orang, terdapat C(*n*,2) sehingga terdapat even sejumlah C(n,2) kemungkinan pasangan yang dapat dibentuk. Kita dapat memperkirakan kemungkinan dua orang akan memiliki ulang tahun yang sama dengan mengasumsikan semua kejadian adalah independen, kemudian mengalikan semua kemungkinan tiap kejadian. Jadi kita dapat membentuk perhitungan dengan mengalikan 364/365 dengan banyaknya kejadian, C(*n*,2), membentuk

$$\bar{p}(n) = \left(\frac{364}{365}\right)^{C(n,2)}$$

Jadi peluang *p(n)* terdapat dua orang memiliki ulang tahun yang sama adalah

$$p(n) \approx 1 - \left(\frac{364}{365}\right)^{C(n,2)}$$

Jika menggunakan *n*=23 dan dengan menggunakan Poisson Aproximation untuk binomial, maka

$$Poi\left(\frac{C(23,2)}{365}\right) \approx Poi\left(\frac{253}{365}\right) \approx Poi(0.6932)$$

$$p_r(x > 0) = 1 - p_r(x = 0)$$

$$p_r(x > 0) = 1 - e^{-0.6932}$$

$$p_r(x > 0) = 1 - 0.49998 = 0.50002$$

Hasil dari perhitungan tersebut tidak jauh dari perhitungan sebelumnya, yaitu dengan *n*=23, didaptkan peluang lebih dari 50 persen.

8 *Empiric Test* untuk *Birthday Paradox*

Sebuah *empiric test* telah dirancang untuk memberikan pengujian secara empiris dari *birthday paradox*. Di sini *empiric test* dijalankan dengan alat bantu Visual Studio 2005. Pseudo algoritma yang akan dijalankan dalam tes ini adalah sebagai berikut

```
days := 365;
prob := 0.0;
targetProb := 0.5
for (numPeople := 1; prob <
targetProb; numPeople++)
{
      comProb := (1 - prob) * (1 -
(numPeople - 1) / days)
      prob := 1 - comProb;
      print "Number of people: " +
numPeople;
      print "Prob. of same birthday:
" + prob;
}
```

Pengujian yang dilakukan membawa hasil yang sama dengan perhitungan yang telah dilakukan sebelumnya.

….

Number of people : 20
Prob. of same *birthday* : 0,411438
Number of people : 21
Prob. of same *birthday* : 0,443688
Number of people : 22
Prob. of same *birthday* : 0,475695
Number of people : 23
Prob. of same *birthday* : 0,507297

9 *Birthday Attack* dalam MD5

Untuk sebuah algoritma *hash* yang bagus, pada dasarnya untuk menemukan sebuah pesan dengan pesan ringkas tertentu dibutuhkan pencarian di pesan-pesan yang dapat dibangkitkan sejumlah tergantung dari ukuran dari pesan ringkas. JIka pesan ringkas tersebut berukuran 64-bit maka ruang pencarian adalah sejumlah 2^{64}, dengan kasus terburuk tentunya. Jadi dengan ukuran pesan ringkas *n*-bit, ruang pencarian akan berjumlah 2^n.

Ruang pencarian seesar itu adalah kasus terburuk karena posisi pesan yang dicari belum tentu berada pada posisi 2^n. Namun dengan menggunakan *birthday paradox*, ruang pencarian akan berubah menjadi jauh lebih kecil dari ruang pencarian semula.

Jika *m* adalah ukuran dari ruang pencarian pesan ringkas contohnya 2^{64}, sehingga dari *birthday paradox*, peluang terjadinya *collision p(n)* adalah satu dikurangi peluang setiap pesan memiliki pesan ringkas yang unik $\overline{p}(n)$.

$$\overline{p}(n) = 1\left(1-\frac{1}{m}\right)\left(1-\frac{2}{m}\right)...\left(1-\frac{n-1}{m}\right)$$

$$\overline{p}(n) = \frac{365!}{365^n (365-n)!}$$

$$p(n) = 1-\overline{p}(n)$$

Kita akan lebih tertarik menemukan nilai *n* sehingga perhitungan di atas harus dikembangkan. Dengan menggunakan pendekatan *taylor series*, kita dapat menyatakan

$$e^x = 1+x+\frac{x^2}{2}+\frac{x^3}{3}+...$$

$$e^{-x/m} = 1-x+\frac{x^2}{2m^2}-\frac{x^3}{3m^3}+...$$

$\left(\frac{x^2}{2m^2}-\frac{x^3}{3m^3}+...\right)$ dapat kita anggap sebagai error yang sangat kecil sehingga dapat kita nyatakan $e^{-x/m} \approx 1-\frac{x}{m}$

Akhirnya, peluang $\overline{p}(n)$ dan *p(n)* bisa didekati dengan

$$\overline{p}(n) = 1\left(1-\frac{1}{m}\right)\left(1-\frac{2}{m}\right)...\left(1-\frac{n-1}{m}\right)$$

$$\overline{p}(n) \approx 1.e^{-1/m}.e^{-2/m}...e^{-(n-1)/m}$$

$$\overline{p}(n) = e^{-(1+2+3+4+5+...+(n-1))/m}$$

$$\overline{p}(n) = e^{-n(n-1)/2m}$$

n(n-1) untuk nilai *n* yang sangat besar, akan sangat dekat dengan n^2, sehingga

$$\overline{p}(n) \approx e^{-n^2/2m}$$

Perhitungan diatas dapat kita balik untuk memenuhi persoalan yang dikemukakan diatas.

- Cari nilai *n* terbesar sehingga nilai *p(n)* adalah lebih kecil dari nilai *p* yang diberikan, atau
- Cari nilai *n* terkecil sehingga nilai *p(n)* adalah lebih besar dari nilai *p* yang diberikan

Dengan *n(p)* menyatakan nilai *n* yang harus dipenuhi untuk memperoleh peluang *p(n)* dalam permasalahan *birthday paradox*, dan *m* menyatakan panjang pesan.

$$p(n) = 1-e^{-n^2/2m}$$

$$1-p = e^{-n^2/2m}$$

$$-\ln\left(\frac{1}{1-p}\right) = -\frac{n^2}{2m}$$

$$n^2 = \ln\left(\frac{1}{1-p}\right)2m$$

$$n(p) = \sqrt{\ln\left(\frac{1}{1-p}\right)2}\sqrt{m}$$

Misalnya untuk mencapai peluang terdapat nilai *hash* yang sama untuk dua pesan yang berbeda p=0.5, maka

$$n(0.5) = \sqrt{2.\ln 2}\sqrt{m}$$

Dengan nilai $\sqrt{2.\ln 2} \approx 1.17741$, maka secara sederhana dapat kita nyatakan untuk mencari *collision* pada pesan ringkas dengan panjang *n*-bit, dengan peluang 50 persen, kita harus mencari sekitar sebanyak $\sqrt{m}$ yaitu $2^{n/2}$. Dengan panjang pesan ringkas 64 bit (m=2^{64} atau sekitar 1.8 x $10^{19)}$, kita hanya perlu mencari di ruang pencarian sebanyak 2^{32} atau sekitar 5.1x$10^{12.}$ Berikut tabel perbandingan jumlah ruang pencarian asal yaitu jumlah kemungkinan output yang dapat dihasilkan dengan panjang pesan ringkas tertentu dibandingkan dengan jumlah ruang pencarian memanfaatkan *birthday paradox* dengan p=50% untuk beberapa panjang pesan (Bits):

Bits	*Ruang Pencarian*	*Ruang Pencarian Akhir*
64	1.8x10^{19}	5.1 × 10^{9}
128	3.4x10^{38}	2.2 × 10^{19}
256	1.1x10^{77}	4.0 × 10^{38}
384	3.9 × 10^{115}	7.4 × 10^{57}
512	1.3 × 10^{154}	1.4 × 10^{77}

Seperti kita lihat, ruang pencarian akhir akan berkurang cukup jauh dari ruang pencarian semula. Hal ini akan membuat pencarian *collision* semakin mungkin secara komputasi dengan menggunakan *brute force attack.*

Dari penjelasan diatas, *birthday attack* dapat dijabarkan sebagai berikut. Jika sebuah fungsi *hash* yang akan mengembalikan 2^n keluaran yang memiliki kemungkinan muncul sama dengan peluang *p*, diberikan sejumlah masukan random, maka jika secara berulang dilakukan evaluasi fungsi terhadap input yang berbeda, kita dapat mengharapkan memperoleh output yang sama dalam $\sqrt{2\ln\left(\frac{1}{1-p}\right)}.2^{n/2}$ percobaan. *birthday attack* adalah sebuah proses yang menggunakan prinsip *birthday paradox* untuk memilih masukan terhadap fungsi *hash f* yang akan diserang sampai menemukan sepasang masukan tersebut yang dapat menghasilkan sebuah kolisi.

Untuk secara khusus melihat kekuatan algoritma MD5 dalam serangan *yang* merupakan sebuah kelas dari *brute force attack* ini, kita dapat membandingkan panjang pesan pesan MD5 dengan algoritma-algoritma lain :

Algoritma	*Panjang Pesan Ringkas*
MD2	128 bit
MD4	128 bit
MD5	128 bit
RIPEMD	128 bit
RIPEMD-128/256	128/256 bit
RIPEMD-160/320	160/320 bit
SHA-0	160 bit
SHA-1	160 bit
SHA-256/224	256/224 bit
SHA-512/284	512/284 bit
WHIRLPOOL	512 bit

Melihat tabel di atas, maka dapat kita lihat bahwa alogoritma MD5 adalah termasuk algoritma yang paling lemah karena memiliki panjang pesan yang paling pendek yaitu 128 bit, sedangkan yang paling kuat adalah algoritma whirlpool maupun sha-512 yang memiliki panjang pesan 512 bit.

Selain itu, untuk melihat tiap sasaran kemungkinan dalam MD5 (128 bit), kita dapat melihat tabel berikut :

p(n)	*n* yang dibutuhkan
10^{-12}	2.6 x 10^{13}
10^{-9}	8.2 x 10^{14}
10^{-6}	2.6 x 10^{16}
0.1%	8.3 x 10^{17}
1%	2.6 x 10^{18}
25%	1.4 x 10^{19}
50%	2.2 x 10^{19}
75%	3.1 x 10^{19}
99%	5.6 x 10^{19}

Dari tabel di atas dapat kita lihat, bahkan untuk mencapai kemungkinan 99% untuk ditemukannya *collision* dalam algoritma MD5, kita hanya perlu mencari di ruang pencarian sebanyak 5.6 x 10^{19} dan pada dasarnya, ruang pencarian yang ditunjukkan di atas adalah berdasarkan kemunginan terburuk (*worst case*), dengan menggunakan distribusi merata. Pada kenyataannya distribusi peluang dalam ruang pencarian adalah tidak merata dengan kemungkinan sejumlah pesan akan menghasilkan pesan ringkas

tertentu akan lebih besar dibandingkan dengan pesan-pesan lainnya. Sebenarnya hal ini berlaku juga dalam serangan *brute force attack* sederhana atau *naive* dengan mencari kemungkinan pada setiap output yang mungin dihasilkan, namun dalam *birthday attack* ini, ruang pencarian direduksi jauh lebih kecil sehingga pencarian *collision* akan jauh lebih cepat dibandingkan *brute force attack* sederhana

Di samping itu kenyataan bahwa distribusi peluang dalan ruang pencarian adalah tidak merata membawa kita pada konsep *balance* yang diajukan oleh Mihir Bellare dan Tadayoshi Kohno [2].

MD5CRK adalah sebuah usaha terdistribusi yang diluncurkan oleh Jean-Luc Cooke dan perusahaannya, *CertainKey Cryptosystems*, untuk mendemonstrasikan bahwa pesan ringkas yang dihasilkan oleh MD5 adalah tidak aman, dengan menemukan *collision* untuk algoritma ini. Proyek ini dimulai dari tanggal 1 Maret 2004 dan berakhir Agustus 2004 setelah sebuah *collision* untuk MD5 telah ditemukan dengan metode analitis.

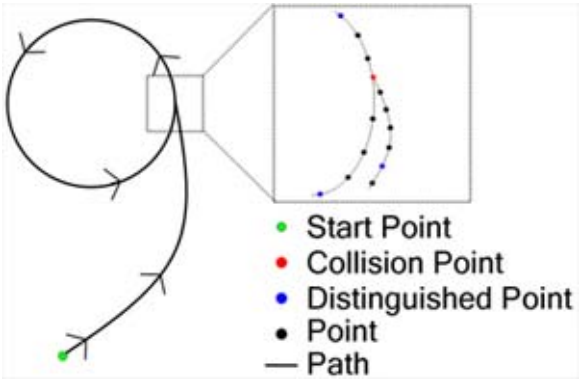
Dengan menggunakan menggunakan penjelasan dalam *birthday attack* di atas, kompleksitas perhitungan untuk melakukan komputasi yang menghasilkan *collision* di MD5 (128 bit) adalah dapat direduksi dari 2^{128} menjadi 1.17741 x $2^{n/2}$ atau 1.17741 x 2^{64} yaitu sekitar 2.2×10^{19}, sesuai dengan tabel yang telah disebutkan sebelumnya. Sebagai ilustralisi, dengan menggunakan system hardware Virginia Tech's yang memiliki performansi maksimum 12.25 Teraflops atau 12 x 10^{12}, maka waktu yang akan dibutuhkan adalah $\frac{2.2 \times 10^{19}}{12.25 \times 10^{12}} \approx 1.770.000$ detik atau sekitar 3 minggu. Kompleksitas waktu perhitungannya secara *naïve* adalah $O(2^{n/2})$, namun sayangnya karena tiap kemungkinan output akan disimpan dalam memori maka kompleksitas memori juga akan sebesar $O(2^{n/2})$.

Teknik yang dinamakan algoritma *Pollard's rho* (sebuah algoritma pendeteksian sirkular) telah digunakan untuk menemukan sebuah *collision* untuk MD5. Teknik ini memunyai kelebihan dalam efisiensi penggunaan memori yang akan membuat komputasi secara praktek menjadi lebih mungkin. Algoritma *Pollard's rho* dapat dideskripsi dengan analogi terhadap *random walk*.

Dengan menggunakan prinsip bahwa sembarang fungsi dengan jumlah kemungkinan keluaran tidak tak terhingga yang ditempatkan dalam sebuah *feedback loop* dapat membentuk kasus sirkular karena, dalam fungsi *hash*, fungsi ini akan berbagi wilayah *domain* dan *range* (sebuah pesan ringkas dapat menjadi pesan pula). Sequence yang tercipta dalam membentuk setiap kemungkinan nilai *hash* untuk pencarian *collision* akan berupa suatu kepala yang diikuti oleh siklus yang berkepanjangan dan tanpa akhir.

Untuk itu, hanya diperlukan sejumlah kecil memory untuk menyimpan keluaran dengan struktur tertentu dan menggunakannya sebagai penanda (*marker*) untuk mendeteksi ketika sebuah marker telah digunakan atau telah masuk dalam proses tersebut sebelumnya.

Penanda atau *marker* ini dinamakan *distinguished point*, dan titik dimana dua masukan akan menghasilkan output yang sama dinamakan *collision point*. MD5CRK menggunakan keluaran-keluaran yang memiliki 32 bit pertama nol (0) sebagai *distinguished point.*



Selain dengan menggunakan algoritma *Pollard's rho, birthday attack* telah dikembangkan dalam serangan-serangan lain yaitu *Teske's modification* (yang masih dikembangkan berdasarkan *Pollard's rho*) dan *parallel collision search*.

Parallel collision search [5] berdasarkan pada ide bahwa teknik *Pollard's rho* akan melibatkan pada pseudo-random walks melalui ruang pencaraian dimana kita harus menunggu selesainya hasil suatu langkah sebelum maju ke langkah berikutnya. Jadi pencarian ini pada dasarnya adalah serial sehingga tidak memungkinkan diadakannya paralelisasi perhitungan. Dengan menggunakan *parallel collision search* yang memanfaatkan kekuatan komputasi paralel, maka akan semakin besar keyakinan bahwa serangan akan semakin mudah dipraktekankan.

Hasil dari penelitian secara intensif dalam kriptanalisis MD5, bahwa blok dalam karakter heksadecimal di bawah ini

```
d131dd02c5e6eec4693d9a0698aff95c
2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325 71415a
085125e8f7cdc99fd91dbd f280373c5b
d8823e3156348f5bae6dacd436c919c6
dd53e2 487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080 80d1e
c69821bcb6a8839396f965 b6ff72a70
```

dan blok berikut

```
d131dd02c5e6eec4693d9a0698aff95c
2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a
085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6
dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b6708028 0d1e
c69821bcb6a8839396f965 b6ff72a70
```

dengan menggunakan algoritma MD5, akan menghasilkan nilai *hash* yang sama yaitu `79054025255fb1a26e4bc422aef54eb4`.

10 *Birthday Attack* dalam *Digital signature*

Digital signature rentan terhadap serangan dengan menggunakan *birthday attack*. Sebuah pesan *m* akan ditandatangani dengan menggunakan pesan ringkas yang akan dibangktikan dengan fungsi *f(m)*, dan kemudian megenkripsinya dengan sebuah kunci privat menjadi suatu tanda tangan atau *digital signature*.

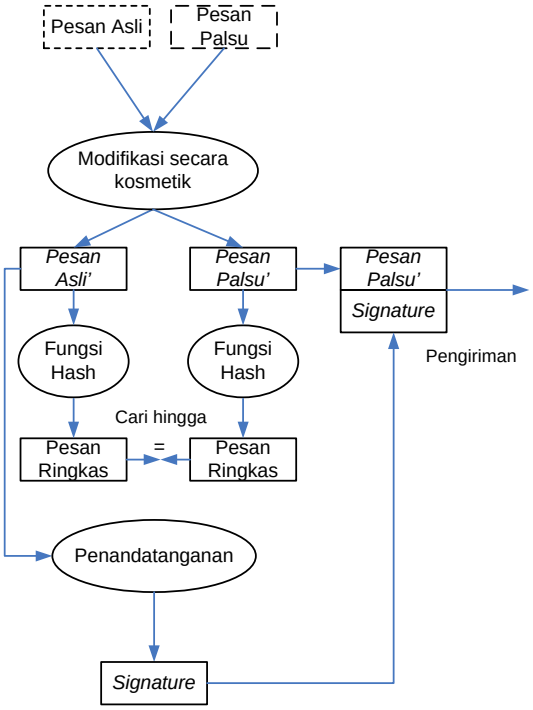
Seandainya Alice menginginkan Bob untuk menandatangani kontrak yang telah dipalsukan. Alice akan menyiapkan sebuah kontrak asli *m* dan kontrak palsu *m'*. Dia kemudian mencari sejumlah cara bagaimana m dapat diubah tanpa mengubah makna dari kontrak seperti menambahkan koma, pergantian baris, spasi setelah sebuah kata atau kalimat, mengganti dengan sinonom dan cara-cara lainnya. Dengan kombinasi-kombisi pengubahan tersebut, Alice dapat menciptakan sejumlah besar variasi dari *m* yang merupakan kontrak asli. Dengan cara yang sama, Alice juga membangkitkan sejumlah besar variasi dari kontrak yang telah dipalsukan *m'*.

Alice kemudian akan menerapkan fungsi *hash* yang sama terhadap semua variasi yang ada sampai dia menemukan sepasang versi variasi dari kontrak palsu dan kontrak asli yang akan berbagi nilai *hash* atau pesan ringkas yang sama, *f(m) = f(m').* Alice akan menyerahkan versi kontrak asli tersebut kepada Bob untuk ditandatangani. Namun kemudian, *digital signature* yang diberikan dari versi kontrak asli ini akan dipindahkan ke pasangan kontrak palsunya. *Digital signature* yang dipindahkan inilah yang akan digunakan oleh Alice untuk "membuktikan" bahwa Bob telah menandatangani kontrak palsu tersebut.

Penerapan ini sebenarnya akan sedikit berbeda dengan permasalahan *birthday attack* yang telah dibahas, dimana yang dicari seharusnya dua kontrak palsu atau kalau tidak dua kontrak asli yang berbagi nilai *hash* yang sama. Namun dalam penerapan ini, dasar dalam *birthday attack* akan digunakan dalam memperkecil ruang pencarian seluruh variasi yang mungkin baik dari kontrak asli maupun kontrak palsu.

Secara spesifik, serangan yang diajukan dapat dijabarkan sebagai berikut [8] :

1. Penyerang memilih dua pesan, pesan *m* yang akan diserang dan pesan palsu *m'* yang ingin ditandatangani.
2. Penyerang membangkitkan $2^{n/2}$ (*n* adalah panjang pesan rigkas) variasi dari pesan asli *m* dengan membuat perubahan minor seperti penambahan *white space* dan sebagainya tanpa mengubah makna, kemudian dengan cara yang sama, bangkitkan variasi dari pesan palsu *m*' dengan cara yang sama.
3. Bangkitkan pesan ringkas dengan sebuah fungsi f dari kombinasi pasangan pesan palsu m dengan pesan asli m' sampai ditemukan keduanya berbagi nilai *hash* yang sama yaitu jika *f(m) = f(m').*
4. Berdasarkan *birthday paradox*, kemungkinan sebuah pesan dari variasi pesan asli akan bersesuaian pesan ringkasnya dengan pesan ringkas dari sebuah variasi dari pesan palsu adalah sekitar 0.5 (karena menggunakan jumlah variasi $2^{n/2}$.
5. Dapatkan tanda tangan dari pesan asli dari pasangan pesan yang ditemukan, kemudian ambil pesan palsu dari pasangan tadi untuk menggantikan pesan asli. Karena pesan palsu tadi akan memiliki pesan ringkas yang sama dengan pesan asli, maka nilai tanda tangan pesan palsu akan sama dengan nilai tanda tangan dari pesan asli, sehingga pergantian ini tidak dapat dideteksi.



11 Serangan terhadap Fungsi *Hash* dalam Sejarah

Algoritma MD5 telah menemui kesulitan di tahun 1993 ketika Bert den Boer dan Antoon Bosselaers menemukan masalah di dalam fungsi kompresinya, lebih jauh lagi masalah ditemukan tiga tahun kemudia oleh Hans Dobbertin. Situasi yang lebih buruk terjadi ketika di tahun 2004, di dalam pertemuan kriptografi di Santa Barbara California Xiaoyun Wang, Denggou Feng, Xuejia Lai, dan Hongbo Yu menerima *standing ovation* untuk kerja mereka dalam menemukan serangan *collision* terhadap MD5, dan juga serangan terhadap algoritma lain HAVAL, MD4, dan RIPEMD. Di pertemuan yang sama Eli Biham dan Rafi Chen menunjukkan cara untuk hampir menemukan *collision* di SHA-0 dan Antoine Joux mendemonstrasikan sebuah serangan actual untuk SHA-0. SHA-1 masih terbilang aman.

Namun di tahun 2005, situasi berubah kembali ketika Wang dengan kolaborasi bersama Yiqun Lisa Yin dan Hongbo Yu menunjukkan serangan *collision* terhadap SHA-1 dengan 2^{69} yang jika menggunakan prinsip *birthday paradox* akan membutuhkan 2^{80} langkah. Kemudian Wang berkolaborasi dengan Andrew Yao dan Frances Yao mendemonstasikan serangan *collision* terhadap SHA-1 dengan hanya membutuhkan 2^{63} langkah [3].

12 Peluang Menghindari Serangan

Untuk mengindari serangan *birthday attack* ini, panjang output dari MD5 ataupun fungsi *hash* lain harus dipilih yang cukup besar untuk membuat serangan menjadi tidak bisa dilakukan secara komputasi. Namun seiring berkembangnya teknologi, cara ini hanya akan membuat para kriptografer berpacu dengan kecepatan kemajuan pengembangan perangkat keras dan perangkat lunak yang mampu melakukan komputasi jauh lebih cepat. Semakin besar output yang dihasilkan semakin sulit serangan dilakukan, namun tentunya kondisi ini akan berbeda bila serangan dilakukan 10 tahun kemudian. Intinya, para kriptografer tetap harus mengembangkan algoritma yang kuat dan mampu bertahan, untuk menggantikan algoritma yang lain yang telah diketahui lemah terhadap serangan-serangan tertentu.

Hal lain yang perlu dipertimbangkan, memperpanjang pesan ringkas untuk memperkuat fungsi *hash* menjadi *trade off* tersendiri terhadap kekuatan fungsi *hash* sebagai pemadat atau kompreser isi pesan asli.

Dalam serangan terhadap *digital signature*, dalam contoh ilustrasi yang telah dijelaskan di atas, Bob direkomendasikan melakukan perubahan secara kosmetik terhadap kontrak apapun sebelum ditandatangani secara digital. Setidaknya hal ini akan mempersulit Alice untuk melakukan pemalsuan namun tidak secara penuh mengatasi masalah karena Alice masih tetap dapat melakukan serangan setelahnya.

13 Berpindah dari Algoritma MD5

Sejauh yang telah dibahas dan telah dilakukan dalam prakteknya, algoritma MD5 sangat lemah dalam memenuhi prinsip *collision resistance*. Namun untuk berpindah atau melakukan transisi menggunakan algoritma *hash* lainnya terdapat beberapa masalah yang akan coba dijelaskan disini.

Masalah dalam transisi *hash* adalah kasus special dari masalah umum dalam transisi protokol hash [1]. Ketika terdapat versi baru dari sebuah protokol, pihak implementor harus mencari cara untuk mendapatkan transisi yang lancar (*smooth transition*) dari versi lama ke versi baru dengan tingkat gangguan yang

rendah. Terdapat tiga agen yang ada pada lingkungan transisi protokal secara umum :

1) Old Agent
Yaitu agen yang hanya bisa menggunakan versi lama dari protokol.

2) New Agent
Yaitu agen yang hanya bisa menggunakan versi baru dari protokol.

3) Switch-Hitting Agent
Yaitu agen yang bisa menggunakan baik versi baru maupun versi lama dari protokol.

Di awal transisi hanya akan ada *old agent* sementara di akhir masa transisi, setidaknya secara teori, hanya akan ada *new agent*, yang pada kondisi nyata seringkali sistem lama harus dipertahankan.

Permasalahan yang harus diperhatikan dalam proses transisi antara lain :

1) Backward compatibility
Old agent dan Switch-hitting agent harus dapat berkomunikasi denggan sistem versi lama.

2) Newest common version
Ketika dua switch-hitting agent berkomunikasi mereka bisa saja menggunakan versi lama atau versi baru dari protokol. Karena tujuan dari transisi adalah menuju versi yang lebih baru, maka diharapkan bisa menggunakan versi yang terbaru.

3) Downgrade protection
Requirement tambahan untuk protokol keamanan adalah mampu memberikan dukungan untuk downgrade versi. Seperti ketika terjadi situasi dimana dua *peers* mendukung dua versi algoritma yang mana yang satu lemah dan yang satu lebih kuat, maka ketika terjadi serangan, kemampuan untuk berpindah ke versi yang lebih kuat harus dapat diandalkan.

4) Credentials versus implementations.
Di dalam sistem yang menggunakan kunci publik, sebuah kunci publik akan diotentikasi menggunakan sertifikat. Sertifikat adalah surat kepercayaan (*credential*) umum dan tidak terkait dengan perubahan spesifik dari protokol keamanan. Pihak peers harus mampu untuk berkomunikasi dengan agent yang memiliki variasi dan kombinasi dari surat kepercayaan lama dan baru, termasuk kemampuan protokol yang dibawanya.

Dari beberapa algoritma lain, algoritma SHA-1 dan variasi lanjutnya menjadi kandidat kuat dalam menggantikan algoritma MD5 walaupun dari sejarah yang telah disebutkan di atas menunjukkan bahwa SHA-1 juga telah berhasil diserang walaupun mungkin masih jauh dari status dapat dipraktekkan.

14 Kesimpulan

Berikut kesimpulan yang dapat kita ambil dari makalah ini :

1. Fungsi *hash* adalah bagian sangat penting dalam fungsi verifikasi dan integritas data dalam protokol keamanan dalam kriptografi. Aplikasinya antara lain dalam MAC, *digital signature*, penyimpanan *password*, dan sebagainya.
2. Keamanan sebuah fungsi *hash* ditentukan oleh kemampuan fungsi tersebut dalam memenuhi prinsip fundamental *Pre-image Resistance, Second Pre-image Resistance*, dan *collision* Resistance.
3. Fungsi *hash* terdiri dari beberapa jenis algoritma seperti MD2, MD4, MD5, SHA-0, SHA-1, RAPEMD, dan beberapa jenis lainnya. Namun untuk beberapa jenis algoritma tersebut telah diketahui rentan terhadap serangan.
4. *Birthday attack* adalah sebuah proses yang menggunakan prinsip *birthday paradox* untuk memilih masukan terhadap fungsi *hash f* yang akan diserang sampai menemukan sepasang masukan tersebut yang dapat menghasilkan sebuah kolisi.
5. Teknik-teknik yang dikembangkan dalam *birthday attack* ini antara lain *Pollard's rho, ,Teske's modification , parallel collision search*.
6. Untuk menghindari *birthday attack* maka kemungkinan keluaran pesan ringkas harus diperbesar dengan memperpanjang panjang pesan ringkas untuk mempersulit dilakukannya serangan secara komputasi.
7. Dari sejarah , telah berhasil diciptakan serangan-serangan yang membuktikan semakin lemahnya MD5 terhadap serangan terutama *collision attack.* Namun melakukan transisi antar protokol dengan mengganti algoritma MD5 dengan algoritma *hash* lain tidaklah mudah dan harus mempertimbangkan aspek-aspek yang menjadi bagian dari kebutuhan akan protokol tersebut.

DAFTAR PUSTAKA

[1] Bellovin, Steven M and Eric K. Rescorla. *Deploying New Hash Algorithm*. Technical Report CUCS-036-05, Dept. of Computer Science, Columbia University. October 2005.

[2] Bellare, Mihir and Tadayoshi Kohno. *Hash Function Balance and its Impact on birthday attacks*.University of California at San Diego. November 2002.

[3] Landau, Susan. *Find Me a Hash*. Notices to AMS Volume 53 Number 2. March 2006.

[4] Munir, Rinaldi. *Bahan Kuliah IF5054 Kriptografi*. Departemen Teknik Informatika, Institut Teknologi Bandung. 2004.

[5] Van Oorschot, Paul C. and Michael J. Wiener. *Parallel collision Search with Application to Hash Function and Discret Logarithm*. Journal of Cryptology vol. 12 no. 1. 1999.

[6] Wiener, Michael J. *Bounds on Birthday Attack Times*. Canada, September 2005.

[7] Wang, Xiaouyun and Hongbo Yu. *How to Break MD5 and Other Hash Functions.* Cina. 2005.

[8] Yuval, Gideon. *How to swindle Rabin*. Cryptologia 3. 1979.

[9] *www.wikipedia.org*. Diakses pada Desember 2006.

[10] *www.mathworld.com*. Diakses pada November 2006.

[11] *www.rsasecurity.com*. Diakses pada Desember 2006.

[12] *www.webreference.com*. Diakses pada Desember 2006.

[13] *blogs.msdn.com/drnick.* Diakses pada Desember 2006.