Algoritma Fungsi Hash Baru dengan Menggabungkan MD5, SHA-1 dan Penyertaan Panjang Pesan Asli

Candra Alim Sutanto - 13508069
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
candra_alim@itb.ac.id

***Abstract*—Keamanan informasi sudah menjadi kebutuhan yang mendesak di zaman sekarang. Salah satu cara untuk mengetahui pengubahan informasi dengan memakai fungsi Hash. Fungsi hash merupakan fungsi yang menerima masukan apapun dan mentransformasikannya menjadi string keluaran yang panjangnya tetap. MD5 dan SHA-1 adalah suatu algoritma fungsi hash yang digunakan untuk menyandikan suatu pesan. Sejauh ini algoritma MD5 dan SHA-1 sudah terdapat kelemahannya.**

Dengan melihat kelebihan dan kekurangan dari kedua algoritma baik MD5 maupun SHA-1, maka dirancanglah algoritma fungsi hash yang baru dengan menggabungkan kedua algoritma MD5 dan SHA-1. Selain itu penyertaan panjang pesan juga akan dipakai dalam pembuatan fungsi hash yang baru. Setelah ditambahkan panjang pesan, kemudian dilakukan lagi Fungsi Hash terhadap *message digest*. Dengan menggabungkan kedua algoritma dan penyertaan panjang pesan diharapkan dihasilkan Fungsi Hash yang kuat dan kolisi yang muncul bisa dikurangi.

***Index Terms*—Fungsi Hash, MD-5, SHA-1, Panjang Pesan.**

I. PENDAHULUAN

Informasi menjadi salah satu kunci perkembangan hidup manusia. Kini, pertukaran informasi dapat terjadi di mana saja, kapan saja, dan melalui berbagai media. Media-media tersebut harus menjamin keamanan informasi yang dipertukarkan. Salah satu faktor keamanan informasi adalah keaslian pesan. Fungsi hash merupakan salah satu cara untuk menjamin keaslian dari suatu pesan.

Fungsi hash merupakan fungsi yang selalu mengahasilkan string dengan panjang yang tetap meskipun panjang pesan asli yang merupakan masukan berapa pun. Fungsi akan menghasilkan keluaran yang sama untuk tiap masukkan yang sama. Dengan begitu bisa diketuhui apakah informasi masih asli dan tidak mengalami perubahan.

Selain itu, fungsi hash merupakan sebuah fungsi yang dapat emnghasikan nilai yang sangat berbeda meskipun masukannya hanya berbeda satu bit. Dengan demikian, fungsi hash merupakan salah satu solusi terbaik untuk melakukan pengecekan terhadap keaslian dari suatu pesan.

MD5 dan SHA-1 merupakan contoh algoritma dalam fungsi hash satu arah. Kedua algoritma ini sudah dapat terpecahkan oleh para kriptanalis. Ada kasus algoritma MD5, dengan menggunakan cara *brute force* maka akan ditemukan dua atau lebih pesan yang mempunyai *message digest* yang sama. Sedangkan pada kasus algoritma SHA-1, terdapat *hash collision*. Pada SHA-1 kelemahannya terletak pada kurangnya kompleksitas algoritma.

Dengan melihat kelemahan dari kedua algoritma fungsi hash, maka dalam makalah ini dirancanglah algoritma yang menggabungkan algoritma MD5 dan SHA-1. Selain itu untuk menambah komplekasitas ada penyertaan panjang pesan asli yang sudah diubah menjadi string dalam hexadecimal. Selanjutnya dilakukan fungsi hash kembali untuk menghasilkan string keluaran yang memiliki panjang pesan yang sama.

II. DASAR TEORI

A. Fungsi Hash

Fungsi hash merupakan fungsi yang menerima masukan bermacam-macam dan menghasilkan string keluaran yang panjangnya tetap. Panjang string keluaran biasanya berukuran jauh lebih kecil daripada ukuran masukan.

Fungsi hash satu arah merupakan fungsi hash yang bekerja satu arah, sekali pesan diubah menjadi *message digest*, maka tidak dapat lagi dikembalikan menjadi pesan semula atau *irreversible*.

Fungsi hash satu arah harus memenuhi sejumlah kriteria berikut

1. *Preimage resistant*, yaitu tidak mungkin menemukan pesan masukan berdasarkan sebuah message digest.
2. *Second preimage resistant*, yaitu tidak mungkin menemukan dua masukan berbeda yang dapat menghasilkan message digest yang sama.
3. *Collision resistant*, yaitu tidak mungkin menemukan dua pesan masukan dengan nilai hash yang sama.
4. Fungsi hash mudah dihitung.
5. Panjang message digest tetap.
6. Fungsi hash dapat diterapkan paada pesan masukan dengan panjang sebarang.

Meskipun fungsi hash satu arah harus memenuhi

keenam kriteria tersebut, masih dapat ditemukan sejumlah kolisi pada fungsi hash. Tabel berikut menggambarkan kolisi yang mungkin terjadi secara lebih detil.

Algoritma	Ukuran *message digest* (bit)	Ukuran 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	160/320	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

Penggunaan fungsi hash sangat mudah dijumpai, sebagai contoh adalah menjaga integritas data. Fungsi *hash* sangat peka terhadap perubahan 1 bit pada pesan. Jika nilai hash sama, maka pesan masih asli, jika berbeda maka pesan sudah diubah. Selain itu penggunaan nilai hash untuk menormalkan panjang data yang beraneka ragam. Misalnya penyimpanan sandi atau *password* di dalam basisdata. Dengan menyimpan nilai hash dari *password* maka panjang field di dalam basisdata bisa serupa.

B. MD5

MD5 di desain oleh Ronald Rivest pada tahun 1991 untuk menggantikan hash function sebelumnya, MD4. Pada tahun 1996, sebuah kecacatan ditemukan dalam desainnya, walau bukan kelemahan fatal, pengguna kriptografi mulai menganjurkan menggunakan algoritma lain, seperti SHA-1. Pada tahun 2004, kecacatan-kecacatan yang lebih serius ditemukan.

Langkah-langkah proses pembuatan message digest pada MD5 adalah sebagai berikut :

1. Penambahan bit pengganjal (padding bits)
2. Penambahan nilai panjang pesan semula
3. Inisialisasi penyangga (buffer) *Message Digest*
4. Pengolahan pesan dalam blok berukuran

Pada tahun 1996, Dobertian mengumumkan adanya kolisi dalam MD-5. Lalu mulai Maret 2004, terdapat proyek yang bernama MD5CRK yang memperlihatkan kelemahan MD5 dengan birthday attack. Dan pada tahun 2005, Arjen Lenstra, Xiaoyun Wang dan Benne de Weger mendemonstrasikan 2 buah sertifikat X.509 dengan kunci publik berbeda namun memberikan nilai hash yang sama. Lalu beberapa hari setelahnya Vlastimil Klima mampu menemukan kolisi MD5 dalam beberapa jam dengan menggunakan komputer tunggal.

Bahkan pada 2006, Klima mempublikasikan algoritma yang mampu menemukan kolisi dari MD5 dalam hitungan satu menit dengan menggunakan komputer tunggal

Berikut ini *pseudocode* dari Algoritma MD5

```
  //Mendefinisikan r sebagai berikut
  var int[64] r, k
  r[ 0..15] := {7, 12, 17, 22,  7, 12, 17, 22,  7,
12, 17, 22,  7, 12, 17, 22}
  r[16..31] := {5,  9, 14, 20,  5,  9, 14, 20,  5,
9, 14, 20,  5,  9, 14, 20}
  r[32..47] := {4, 11, 16, 23,  4, 11, 16, 23,  4,
11, 16, 23,  4, 11, 16, 23}
  r[48..63] := {6, 10, 15, 21,  6, 10, 15, 21,  6,
10, 15, 21,  6, 10, 15, 21}

  //Menggunakan bagian fraksional biner dari
integral sinus sebagai konstanta:
  for i from 0 to 63
      k[i] := floor(abs(sin(i + 1)) × 2^32)

  //Inisialisasi variabel:
  var int h0 := 0x67452301
  var int h1 := 0xEFCDAB89
  var int h2 := 0x98BADCFE
  var int h3 := 0x10325476

  //Pemrosesan awal:
  Menambahkan bit "1" ke dalam pesan
  Menambahkan sejumlah bit "0" sehingga panjang
pesan dalam bit ≡ 448 (mod 512)
  Menambahkan panjang bit dari pesan dalam 64-bit
little-endian integer ke dalam pesan

  //Pengolahan pesan paada kondisi gumpalan 512-bit:
  Untuk setiap 512-bit bagian dari pesan
      Bagi tiap bagian menjadi enam belas 32-bit
little-endian words w(i), 0 ≤ i ≤ 15

      //Inisialisasi nilai hash pada bagian ini:
      var int a := h0
      var int b := h1
      var int c := h2
      var int d := h3

      //Kalang utama:
      for i from 0 to 63
          if 0 ≤ i ≤ 15 then
              f := (b and c) or ((not b) and d)
              g := i
          else if 16 ≤ i ≤ 31
              f := (d and b) or ((not d) and c)
              g := (5×i + 1) mod 16
          else if 32 ≤ i ≤ 47
              f := b xor c xor d
              g := (3×i + 5) mod 16
          else if 48 ≤ i ≤ 63
              f := c xor (b or (not d))
              g := (7×i) mod 16

          temp := d
          d := c
          c := b
          b := ((a + f + k(i) + w(g)) leftrotate
r(i)) + b
          a := temp

      //Tambahkan hash ke dalam hasil:
      h0 := h0 + a
```

```
      h1 := h1 + b
      h2 := h2 + c
      h3 := h3 + d

var char digest[16] := h0 append h1 append h2 append
h3 //(diwujudkan dalam little-endian)
```

C. SHA-1

SHA-1 atau *Secure Hash Algorithm* didesain oleh *National Security Agent* diperkenalkan oleh NIST sebagai U.S. Federal Information Processing Standard.

SHA-1 message digest sepanjang 160 bit. Algoritma utama SHA-1 terdiri dari enam proses utama berikut.

1. Inisialisasi variabel kunci
2. Penambahan bit 1
3. Pemecahan pesan ke dalam kelompok berukuran 512-bit
4. Ekstensi pesan
5. Iterasi utama
6. Pembangkitan hash value.

Berikut ini *pseudocode* dari Algoritma SHA-1

```
  Initialize variables:
  h0 = 0x67452301
  h1 = 0xEFCDAB89
  h2 = 0x98BADCFE
  h3 = 0x10325476
  h4 = 0xC3D2E1F0

  Pre-processing:
  append the bit '1' to the message
  append 0 ≤ k < 512 bits '0', so that the resulting
message length (in bits) is congruent to 448 ≡ −64
(mod 512)
  append length of message (before pre-processing),
in bits, as 64-bit big-endian integer

  Process the message in successive 512-bit chunks:
  break message into 512-bit chunks
  for each chunk
      break chunk into sixteen 32-bit big-endian
words w[i], 0 ≤ i ≤ 15

      Extend the sixteen 32-bit words into eighty
32-bit words:
      for i from 16 to 79
          w[i] = (w[i-3] xor w[i-8] xor w[i-14] xor
w[i-16]) leftrotate 1

      Initialize hash value for this chunk:
      a = h0
      b = h1
      c = h2
      d = h3
      e = h4

      Main loop:
      [28]
      for i from 0 to 79
          if 0 ≤ i ≤ 19 then
              f = (b and c) or ((not b) and d)
              k = 0x5A827999
          else if 20 ≤ i ≤ 39
              f = b xor c xor d
              k = 0x6ED9EBA1
          else if 40 ≤ i ≤ 59
              f = (b and c) or (b and d)or (c and d)
              k = 0x8F1BBCDC
          else if 60 ≤ i ≤ 79
              f = b xor c xor d
              k = 0xCA62C1D6

          temp = (a leftrotate 5) + f + e + k + w[i]
          e = d
          d = c
          c = b leftrotate 30
          b = a
          a = temp

      Add this chunk's hash to result so far:
      h0 = h0 + a
      h1 = h1 + b
      h2 = h2 + c
      h3 = h3 + d
      h4 = h4 + e

  Produce the final hash value (big-endian):
digest = hash = h0 append h1 append h2 append h3
append h4
```

Pada awal 2005, Rijmen dan Oswald menerbitkan sebuah serangan pada versi penurunan dari SHA-1, berupa 53 dari 80 putaran, menemukan kolisi dengan upaya komputasi kurang dari 2^{80} operasi.

Pada bulan Februari 2005, sebuah serangan oleh Xiaoyun Wang, Yiqun Lisa Yin, Bayarjargal, dan Hongbo Yu dipublikasikan. Serangan dapat menemukan kolisi dari versi penuh SHA-1, danmembutuhkan kurang dari 2^{69} operasi. Sebuah pencarian brute-force akan membutuhkan 2^{80} operasi.

III. STRUKTUR FUNGSI HASH BARU

A. Rancangan Algoritma

Dari penjelasan kedua algoritma fungsi hash di atas, bisa terlihat pada dasarnya langkah-langkah proses pembuatan message digest kedua algoritma tersebut adalah sama. Perbedaannya terletak pada inisialisasi variabel, nilai penyangga, jumlah looping dan bit jumlah hasil nilai hash.

Dengan melihat semua perbedaan kedua algoritma MD5 dan SHA-1 kemudian dirancanglah algoritma fungsi hash baru yang memanfaatkan kedua algoritma. Dengan menggabungkan kedua algoritma, maka akan diperoleh algoritma fungsi hash yang baru dan bisa dikurangi kolisi yang muncul.

Algoritmafungsi hash yang baru akan menghasilkan *message digest* atau nilai hash sebanyak 160 bit. Hal ini berarti algoritma fungsi hash yang baru lebih kuat dari algoritma MD5 yang hanya menghasilkan panjang message digest 128 bit.

Inisialisai yang dipakai dalam algoritma fungsi hash baru merupakan gabungan dari kedua algoritma. Kemudian dilakukan *padding* atau penambahan bit yang mana kedua algoritma melakukan proses *padding* yang sama. Selanjutnya dilakukan untuk tiap 512 bit pesan dilakukan looping sebanyak 64 kali sesuai dengan algoritma MD5. Looping dilakukan hanya sebanyak 64

kali, bukan 80 kali seperti SHA-1 dikarenakan inisialisasi variabel array pada awal program hanya sampai elemen ke 64. Sehingga iterasi hanya dilakukan sebanyak 64 kali.

B. Penyertaan Panjang Pesan

Penyertaan panjang pesan asli pada *Message Digest* atau nilai hash akan menghasilkan nilai hash yang unik. Hal ini dikarenakan tidak ada dua pesan berbeda dengan panjang yang sama dapat menghasilkan nilai hash yang sama. Dengan kata lain, nilai hash(pesan1) ≠ nilai hash(pesan2) akan selalu berlaku jika dengan panjang(pesan1) = panjang(pesan2) dan pesan1 ≠ pesan2.

Rancangan proses penyertaan pesan antara lain adalah dengan mengubah panjang pesan beruapa sebuah bilangan menjadi sebuah hexadecimal. Proses pengubahan panjang pesan menjadi hexadecimal dengan cara panjang pesan di-mod 16, kemudian dihasilkan huruf hexadecimal. Kemudian panjang pesan tadi di bagi dengan 10 dan dilakukan fungsi mod lagi. Proses ini dilakukan sampai panjang pesan menjadi 0.

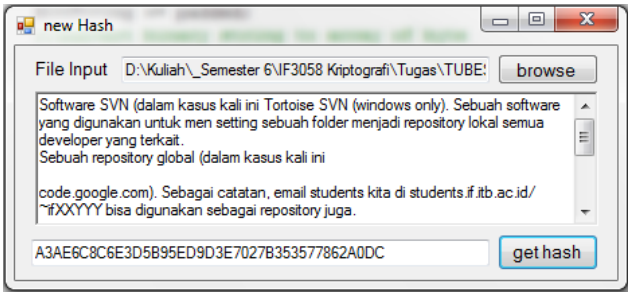
```
int copy <- Message.length
int temp <- 0
string result
While copy > 0
  temp = copy % 16
  result = toHex(temp)
  copy = copy / 10
result
```

Fungsi hash haruslah menghasilkan string keluaran dengan panjang yang tetap, ketika dilakukan penyertaan panjang pesan asli, maka panjang string keluaran dari fungsi hash pasti akan berbeda-beda, tergantung dari panjang pesan asli sebagai masukan dari fungsi hash. Message digest dengan panjang yang berbeda dengan panjang pada umumnya akan memicu kecurigaan. Oleh karena itu, message digest yang ditambahkan panjang dokumen menjadi input fungsi hash lagi. Message digest hasil fungsi tersebut yang menjadi message digest sebenarnya. Penerapan fungsi hash sebanyak dua kali tentu menambah kompleksitas.

IV. IMPLEMENTASI DAN PENGUJIAN

A. Implementasi

Implementasi modifikasi yang dipaparkan pada bagian sebelumnya dilakukan pada IDE Microsoft Visual Studio 2008 dan bahasa pemrograman C#. Antarmuka aplikasi yang dikembangkan dapat dilihat pada gambar di bawah ini.



Berikut ini pseudocode dari algoritma dari fungsi hash yang baru

```
  //perubahan
  Initialize variables:
  var int[64] r, k
  r[ 0..15] := {7, 12, 17, 22,  7, 12, 17, 22,  7,
12, 17, 22,  7, 12, 17, 22}
  r[16..31] := {5,  9, 14, 20,  5,  9, 14, 20,  5,
9, 14, 20,  5,  9, 14, 20}
  r[32..47] := {4, 11, 16, 23,  4, 11, 16, 23,  4,
11, 16, 23,  4, 11, 16, 23}
  r[48..63] := {6, 10, 15, 21,  6, 10, 15, 21,  6,
10, 15, 21,  6, 10, 15, 21}

  //perubahan
  //Use fraksional biner from sinus as constanta:
  for i from 0 to 63
      k[i] := floor(abs(sin(i + 1)) × 2^32)

  Initialize variables:
  h0 = 0x67452301
  h1 = 0xEFCDAB89
  h2 = 0x98BADCFE
  h3 = 0x10325476
  h4 = 0xC3D2E1F0

  Pre-processing:
  append the bit '1' to the message
  append 0 ≤ k < 512 bits '0', so that the resulting
message length (in bits) is congruent to 448 ≡ −64
(mod 512)
  append length of message (before pre-processing),
in bits, as 64-bit big-endian integer

  Process the message in successive 512-bit chunks:
  break message into 512-bit chunks
  for each chunk
      break chunk into sixteen 32-bit big-endian
words w[i], 0 ≤ i ≤ 15

      Extend the sixteen 32-bit words into eighty
32-bit words:
      for i from 16 to 79
          w[i] = (w[i-3] xor w[i-8] xor w[i-14] xor
w[i-16]) leftrotate 1

      Initialize hash value for this chunk:
      a = h0
      b = h1
      c = h2
      d = h3
      e = h4

      Main loop:
      [28]
      for i from 0 to 64
          if 0 ≤ i ≤ 19 then
```

```
                f = (b and c) or ((not b) and d)
                g = i    //perubahan
            else if 20 ≤ i ≤ 39
                f = b xor c xor d
                g = (5 * i + 1) % 16    //perubahan
            else if 40 ≤ i ≤ 59
                f = (b and c) or (b and d)or (c and d)
                g = (3 * i + 5) % 16    //perubahan
            else if 60 ≤ i ≤ 79
                f = b xor c xor d
                g = (7 * i) % 16    //perubahan

            temp = leftrotate((a + f + k[i] + w[g]),
r[i]) + b    //perubahan
            e = d leftrotate 5    //perubahan
            d = c
            c = b leftrotate 30
            b = a
            a = temp

        Add this chunk's hash to result so far:
        h0 = h0 + a
        h1 = h1 + b
        h2 = h2 + c
        h3 = h3 + d
        h4 = h4 + e

    Produce the final hash value (big-endian):
digest = hash = h0 append h1 append h2 append h3
append h4
```

Dari pseudocode di atas dapat terlihat adanya perubahan pada algoritma SHA-1. Pada pseudocode di atas sudah ditunjukkan bagian mana saja yang diubah atau diberi penambahan yang merupakan bagian dari algoritma MD5. Dari penambahan tersebut juga dilakukan beberapa penyesuaian agar tepat bila dipasang pada algoritma SHA-1.

Berikut adalah implementasi fungsi pengubahan panjang pesan asli menjadi heksa. Fungsi ini digunakan untuk penyertaan panjang pesan asli pada ujung message digest.

```
public string lengthToHex()
   //convert length message hex string 10 byte
   {
       int copy = input.Length, temp2 = 0;
       string result = "";
       do
       {
           Math.DivRem(copy, 16, out temp2);
           result += toHex(temp2);
           copy = copy / 10;
       } while (copy > 0);
       return result.ToUpper();
   }
```

Output fungsi tersebut ditambahkan ke akhir *message digest* fungsi hash pertama. Selanjutnya, *message digest* ini bertambah sejumlah karakter heksa tergantung dari panjang pesan asli sebagai masukan dari fungsi hash. Message digest ini kemudian dilakukan fungsi hash kembali agar mengahasilkan nilai hash yang sama. Hasil fungsi hash kedua ini merupakan nilai hash dari pesan.

B. Pengujian

Pengujian dilakukan dengan menggunakan masukan yang sama dengan masukan pada tabel. Berikut merupakan hasil pengujian yang dilakukan pada aplikasi yang telah menggunakan algoritma baru hasil gebungan MD5 dan SHA-1 dan melakukan penyertaan panjang pesan seperti yang sudah dijelaskan pada bagian III.

Input pesan	SHA-1	MD5	Hash Baru
Algoritma hash baru dengan memanfaatkan MD5, SHA-1 dan panjang pesan asli	52871043F D9EBC3FD 46D16DEE 124F87215 D50A1A	B1ACB823 69F38D68E AE5CADE 9D3A5F9F	5B38B1BB C53DCF9A EE4DC2A3 2B75EB9A 317F7D8F
Algoritma hash baru dengan memanfaatkan MD5, SHA-1 dan panjang pesan asli!	D4B845D5 4BB5F6A1 29DB16F0 B0AC813C 202C875E	11CF5D2C E807EFCD 99295EC76 AE6639F	2452894D5 C28FE204F 7C46AE11 A515F4C06 82B1B

Dari hasil diatas dapat terlihat bahwa terdapat perbedaan yang signifikan antara algoritma SHA-1, MD5 dan algoritma baru. Algoritma MD5 menghasilkan 128 bit nilai hash, sedangkan algoritma hash yang baru sama seperti SHA-1 menghasilkan 160 bit nilai hash.

Untuk waktu eksekusi dari masing-masing algoritma bisa dilihat di tabel berikut ini.

Input pesan	SHA-1	MD5	Hash Baru
tes.jpg (16KB)	05.2352994 detik	05.2072978 detik	05.4983145 detik

Dari tabel diatas, bisa dilihat bahwa waktu eksekusi dari fungsi hash baru lebih lama dari algoritma SHA-1 dan MD5 meskipun tidak terlalu signifikan. Untuk ukuran data yang lebih besar mungkin selisih waktu akan lebih signifikan. Hal ini menunjukkan bahwa algoritma fungsi hash yang baru lebih kompleks secara logika dan teori dibandingkan dengan algoritma SHA-1 dan MD5.

Dengan adanyan kompleksitas pada operasi proses pembuatan message digest, maka dapat dipastikan kompleksitas yang diperlukan untuk mencari kolisi pada algoritma ini akan lebih tinggi daripada algoritma SHA-1 ataupun MD5. Selain itu dengan penyertaan panjang pesan asli maka akan lebih aman meskipun tetap saja tidak bisa menjamin tidak ditemukan kolisi. Kemungkinan adanya dua pesan menghasilkan nilai hash yang sama masih mungkin terjadi.

Analisis ini hanya sebatas perhitungan logika saja. Dibutuhkan waktu yang sangat lama bila dilakukan analisis berdasarkan komputasi.

V. KESIMPULAN

Berdasarkan pemaparan yang ada pada bagian-bagian sebelumnya, dapat diperoleh sejumlah kesimpulan berikut.

1. Pada dasarnya algoritma ini seperti memodifikasi algoritma SHA-1. Namun algoritma di atas hanya merupakan salah satu contoh saja. Bisa juga menggunakan algoritma MD5 sebagai algoritma inti lalu dilakukan modifikasi dengan menambahkan beberapa pengoperasian dari algoritma SHA-1. Namun dengan menggunakan algoritma SHA-1 sebagai dasar dapat memberikan tingkat keamanan yang lebih.
2. Penganalisaan merupakan hanya sebatas perhitungan logika, tidak secara komputasi. Sehingga dapat ditemukan cara yang lebih kompleks dalam pembentukan algoritma di atas dimana bila dengan menggunakan komputasi sebagai dasar analisis akan menjadi lebih baik.
3. Modifikasi penyertaan panjang pesan asli dapat meningkatkan jaminan keaslian data, namun tidak menutup kemungkinan adanya kolisi.

Hasil riset yang telah dilakukan belum melalui sejumlah tahap pengujian, khususnya pengujian serangan. Modifikasi ini sebaiknya dikembangkan lebih lanjut, khususnya untuk mencari tahu apakah modifikasi ini mempermudah kolisi atau tidak.

DAFTAR PUSTAKA

[1] Munir, Rinaldi. Slide Perkuliahan IF3058 Kriptografi.
[2] http://en.wikipedia.org/wiki/SHA-1
[3] http://en.wikipedia.org/wiki/MD5
[4] http://en.wikipedia.org/wiki/Cryptographic_hash_function

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, 09 Mei 2011

Candra Alim Sutanto - 13508069