# Studi dan Implementasi Algoritma Baru Dengan Mengkombinasikan Algoritma MD5 dan SHA-1

Muchamad Surya Prasetyo – NIM : 13505065

*Program Studi Teknik Informatika, Institut Teknologi Bandung*

*Jl. Ganesha 10, Bandung*

E-mail : if15065@students.if.itb.ac.id

**Abstrak**

Makalah ini membahas tentang suatu algoritma baru dimana memanfaatkan algoritma yang sudah ada, yaitu MD5 dan SHA-1. 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. MD5 yang pada awalnya dibilang cukup sempurna ternyata terdapat koalisi namun tidak banyak. Sedangkan pada SHA-1 **al;sdadl;k.** Dengan melihat kelebihan dan kekurangan dari kedua algoritma tersebut maka dirancanglah suatu algoritma yang diharapkan dapat menangani kelemahan kedua algoritma tersebut.

**Kata kunci** : *MD5, SHA-1, Fungsi Hash*

## 1. Pendahuluan

Kriptografi atau yang sering dikenal dengan sebutan ilmu penyandian data adalah suatu bidang ilmu dan seni yang bertujuan untuk menjaga kerahasiaan suatu pesan yang berupa data-data dari pihak lain yang tidak berhak sehingga tidak menimbulkan kerugian. Pada saat ini, kriptografi dibagi menjadi dua jenis yaitu kriptografi klasik dan kriptografi modern.

Kriptografi klasik sudah mulai dipelajari manusia sejak tahun 400 SM, yaitu pada zaman Yunani kuno. Kriptografi klasik ini digunakan oleh orang-orang terdahulu untuk menjaga kerahasiaan suatu pesan teks pada masa sebelum ditemukan komputer. Kriptografi klasik merupakan kriptografi yang berbasis karakter. Sedangkan kriptografi modern adalah kriptografi yang berkembang pada zaman setelah ditemukan komputer. Kriptografi modern beroperasi dalam bit, tidak seperti kriptografi klasik yang hanya beroperasi pada karakter.

Selain itu, didalam kriptografi terdapat sebuah fungsi yang sesuai untuk aplikasi keamanan seperti otentikasi dan integritas pesan. Fungsi tersebut adalah fungsi *hash*. Fungsi *hash* adalah fungsi yang menerima masukkan *string* yang panjangnya sembarangan dan mengkonversinya menjadi *string* yang panjangnya tetap.

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.

Untuk menanggulangi kelemahan-kelemahan tersebut dapat dilakukan beragai macam modifikasi pada algoritma masing-masing. Dalam makalah ini, saya ingin membuat suatu algoritma dimana dengan algoritma yang baru ini diharapkan dapat mengurangi kelemahan dari kedua algoritma tersebut atau singkatnya kedua algoritma ini dapat saling menutupi kelemahannya masing-masing.

## 2. Dasar teori

### 2.1 MD5

MD5 adalah fungsi *hash* satu arah yang dibuat oleh Ronald Rivest pada tahun 1991. MD5 merupakan varian perbaikan dari pendahulunya yaitu MD4, yang merupakan perbaikan dari MD3 dan MD2. Algoritma MD5 menerima masukkan berupa pesan dengan ukuran sembarang dan menghasilkan *message digest* yang panjangnya 128 bit.

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*) MD
4. Pengolahan pesan dalam blok berukuran 512 bit.

Dengan panjang *message digest* 128 bit, secara brute force dibutuhkan percobaan sebanyak 2128 kali untuk menemukan dua buah pesan atau lebih yang mempunyai *message digest* yang sama. Pada tahun 1996, Dobbertin menemukan koalisi MD5 walaupun membutuhkan waktu yang lama. Lalu pada tahun 2004, terdapat proyek yang bernama MD5CRK yang memperlihatkan ketidakamanan 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 Vlastimil Klima menyempurnakan algoritma Lenstra dan menemukan kolisi MD5 dalam beberapa jam dengan menggunakan komputer PC.

Pseudocode untuk MD5 adalah sebagai berikut :

```
//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:
append "1" bit to message
append "0" bits until message
length in bits ≡ 448 (mod 512)
append bit length of message as
64-bit little-endian integer to
message

//Pengolahan pesan paada kondisi
gumpalan 512-bit:
for each 512-bit chunk of message
    break chunk into sixteen 32-
bit little-endian words w(i), 0 ≤
i ≤ 15

    //Inisialisasi nilai hash pada
gumpalan 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 dari gumpalan
sebagai hasil:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d

var int digest := h0 append h1
append h2 append h3 //(diwujudkan
dalam little-endian)
```

### 2.2 SHA-1

SHA atau Secure *Hash* Algorithm adalah suatu algoritma fungsi *hash* yang dirancang oleh National Security Agency (NSA) dan diperkenalkan oleh NIST sebagai U.S. Federal Information Processing Standard. SHA dapat dianggap sebagai kelanjutan pendahulunya yaitu MD5 yang telah digunakan secara luas. SHA merupakan keluarga fungsi *hash* satu-arah. SHA-1 sudah diimplementasikan dalam berbagai aplikasi dan protokol keamanan seperti TLS, SSL, PGP, SSH, S/MIME, dam Ipsec. Terdapat beberapa macam varian SHA yaitu dimulai dari SHA-0, SHA-1, SHA-224, SHA-256, SHA-384, dan SHA-512.

SHA-1 menerima masukkan berupa pesan dengan ukuran maksimum adalah 264 bit dan menghasilkan *message digest* yang panjangnya 160 bit, lebih panjang dari yang dihasilkan oleh MD5 yaitu 128 bit. Langkah-langkah proses SHA-1 secara garis besar adalah sebagai berikut :

1. Penambahan bit-bit pengganjal (*padding bits*)
2. Penambahan nilai panjang pesan semula
3. Inisialisasi penyangga (*buffer*) MD
4. Pengelolaan pesan dalam blok berukuran 512 bit.

Pada tahun 2005 Rijmen dan Oswald mempublikasikan serangan pada versi SHA-1 yang direduksi (hanya 53 looping dari 80) dan menemukan kolisi dengan kompleksitas sekitar 280 operasi. Dan pada Februari 2005, Xiayoun Wang, Yiqun Lisa Yin, dan Hongbo Yo empublikasikan serangan yang dapat menemukan kolisi pada veri penuh SHA-1, yang membutuhkan sekitar 269 operasi.

Pseudocode untuk SHA-1 adalah sebagai berikut :

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

Pre-processing:
append the bit '1' to the message
append k bits '0', where k is the minimum number ≥ 0 such that the resulting message
    length (in bits) is congruent to 448 (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:
    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
```

## 3. Hasil dan Pembahasan

### 3.1 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 jumlah nilai penyangga, jumlah looping dan hasil pesan *message digest* serta algoritma intinya. Dengan melihat semua itu dapat dirancang sebuah algoritma baru dimana setidaknya memiliki tingkat keamanan yang lebih daripada MD5 dan SHA-1.

Kombinasi yang dilakukan dengan merubah atau menambahkan beberapa bagian algoritma pada MD5 pada algoritma SHA-1. Hal ini dikarenakan dengan menggunakan algoritma SHA-1 pada inti algoritma baru ini, akan memiliki panjang *message digest* 160 bit, dimana akan lebih kuat daripada bila menggunakan algoritma MD5 sebagai intinya yang hanya menghasilkan panjang *message digest* 128 bit.

Looping yang dilakukan pun lebih banyak daripada algoritma standar SHA-1. Hal ini merupakan salah satu upaya untuk meningkatkan keamanan pada algoritma baru ini. Dengan meningkatnya looping maka akan meningkatkan kompleksitas operasi pada algoritmanya.

Rancangan pseudocode algoritma baru ini adalah sebagai berikut :

```
Mendefinisikan r sebagai berikut
// Perubahan
var int[80] r, T
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}
r[64..79] := {8, 13, 18, 24,  8,
13, 18, 24,  8, 13, 18, 24,  8,
13, 18, 24} // Disini merupakan
penambahan definisi r, dimana isi
r[64..79] melihat dari pola r
sebelumnya dan disesuaikan

Menggunakan bagian fraksional
biner dari integral sinus sebagai
konstanta:
for i from 0 to 80
    T[i] := floor(abs(sin(i + 1))
× 2^32) // Perubahan

Inisialisasi Penyangga
h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0

Pemrosesan Awal :
append the bit '1' to the message
append k bits '0', where k is the
minimum number ≥ 0 such that the
resulting message
    length (in bits) is congruent
to 448 (mod 512)
append length of message (before
pre-processing), in bits, as 64-
bit big-endian integer

Proses pesan dalam blok-blok yang
berukuran 512 bit:
break message into 512-bit chunks
for each chunk
    break chunk into sixteen 32-
bit big-endian words w[i], 0 ≤ i ≤
15

    Expansi 16 buah sub-blok 32
bit menjadi 80 buah sub-blok 32
bit:
    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

    Inisialisasi peubah penyangga:
    a = h0
    b = h1
    c = h2
    d = h3
    e = h4

    Main loop:
    for j from 0 to 1       //
Perubahan
        for i from 0 to 79
                if 0 ≤ i ≤ 19 then
                    f = (b and c)
or ((not b) and d)
                    k = 0x5A827999
                    g   =   i   //
Perubahan
                else if 20 ≤ i ≤
39
                    f = b xor c
xor d
                    k = 0x6ED9EBA1
                    g = (5*i + 1)
mod 20 // Perubahan
                else if 40 ≤ i ≤
59
                    f = (b and c)
or (b and d) or (c and d)
                    k = 0x8F1BBCDC
                    g = (3*i + 1)
mod 20 // Perubahan
                else if 60 ≤ i ≤
79
                    f = b xor c
xor d
                    k = 0xCA62C1D6
                    g = (7*i + 1)
mod 20 // Perubahan
```

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

                a = temp

    Jumlahkan blok-blok ini:
    h0 = h0 + a
    h1 = h1 + b
    h2 = h2 + c
    h3 = h3 + d
    h4 = h4 + e

Nilai akhir hash :
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.

### 3.2 Perbandingan Hasil

Pesan : "Fungsi *hash* SHA-1 dan MD5"

MD SHA-1 :
797DE6AB5590B0EC0169BC5531E7027F4EA351DF

MD Algoritma Baru :
F0DALEP3043A3QZ5871LHF87THF7492BCKF72K9S

### 3.3 Analisis Hasil

Dari hasil diatas dapat terlihat bahwa terdapat perbedaan yang signifikan antara algoritma SHA-1 dan algoritma baru. 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. Analisis ini hanya sebatas perhitungan logika saja. Dibutuhkan waktu yang sangat lama bila dilakukan analisis berdasarkan komputasi.

## 4. Kesimpulan

1. Pada dasarnya algoritma ini seperti memodifikasi algoritma SHA-1. Namun algorita 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. Setiap teknik kriptografi pasti ada teknik kriptanalisnya walaupun membutuhkan waktu yang sangat lama. Sehingga untuk lebih aman dalam mengirimkan pesan adalah dengan cara tatap muka.

## 5. Daftar Pustaka


[1] Munir, Rinaldi. Diktat Kuliah IF3058 Kriptografi.
[2] http://en.wikipedia.org/wiki/SHA1
[3] http://id.wikipedia.org/wiki/MD5