Modifikasi Algoritma MD5 dengan Fungsi-F Algoritma 747

Husnulzaki Wibisono Haryadi 13515005[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13515005@std.stei.itb.ac.id

***Abstrak*—Fungsi *hash* merupakan salah satu konsep pemrograman yang seiring berjalannya waktu berkembang dengan pesatnya. Diantara berbagai macam kegunaan fungsi *hash*, kriptografi merupakan salah satu bidang yang mendapat dampak cukup besar darinya. Dalam laporan ini penulis akan membahas algoritma *hash* kriptografi MD5 yang penulis implementasi dan modifikasi sendiri dengan algoritma kriptografi 474 yang telah lebih dahulu penulis ciptakan.**

***Keywords*—*Hash*, MD5, kriptografi.**

I. Pendahuluan

Hash function atau fungsi *hash* merupakan konsep pemrograman yang pada prinsipnya cukup sederhana. Fungsi *hash* bertujuan untuk memetakan data dalam bentuk dan ukuran apapun menjadi sebuah nilai dengan panjang tetap. Nilai ini disebut sebagai *hash value* atau nilai *hash*. Dua buah data yang berbeda, tidak peduli seberapa kecil perbedaannya, harus mendapatkan nilai hash yang berbeda juga. Dengan demikian, nilai *hash* dapat diperlakukan sebagai identitas yang unik dari sebuah data, sedangkan fungsi hash berperan sebagai pembangkit identitas unik ini.

Sejak pertama kali dicetuskannya, pengaplikasian fungsi *hash* pada dunia komputasi kian bertambah seiring berjalannya waktu. Hingga saat ini, berbagai ragam algoritma *hash* telah dirilis dan memainkan peran penting dalam implementasi berbagai bidang teknologi informasi, mulai dari manajemen dan penyimpanan data, hingga penjaminan keamanan informasi. Peningkatan efisiensi dan kekuatan dari algoritma menjadi klaim utama dari tiap varian fungsi *hash* baru yang muncul. Salah satu dari berbagai alternatif algoritma *hash* ini adalah algoritma *Message Digest* 5 (MD5).

MD5 termasuk sebagai kategori fungsi *hash* kriptografi, yaitu fungsi *hash* yang cocok dan memenuhi kriteria tertentu untuk digunakan dalam bidang kriptografi. MD5 sendiri sesungguhnya bukanlah algoritma pembangkitan nilai *hash* yang paling mangkus yang ada saat ini. Algoritma ini diciptakan pada tahun 1991 dan telah diketahui mampu dipecahkan secara *brute force* oleh komputer personal modern semata. Di sisi lain, algoritma ini memiliki alur kerja yang mudah dipahami dan dengan demikian dapat diimplementasi dan dipelajari dengan cepat. Alasan inilah yang mendasari digunakannya algoritma MD5 sebagai dasar pengerjaan laporan ini.

Pada laporan ini, penulis melakukan implementasi algoritma MD5 pada bahasa Python dan lalu memodifikasi algoritma ini dengan fungsi kriptografi dari algoritma 474 penulis ciptakan sendiri. Modifikasi dan percobaan yang dilakukan dalam laporan ini bertujuan untuk mengeksplorasi cara kerja dan kinerja fungsi *hash,* khususnya terhadap algoritma MD5, sekaligus menguji teori-teori yang berkaitan dengan fungsi *hash* kriptografi.

II. Dasar Teori

A. *Fungsi* Hash *dan* Hash *Kriptografi*

Fungsi *hash* merupakan istilah bagi fungsi komputasi apapun yang dapat memetakan data dengan ukuran sembarang menjadi data dalam panjang tetap. Fungsi *hash* bekerja dengan menerima data dalam bentuk bit, mengolahnya dalam fungsi internal, dan menghasilkan rangkaian bit dalam panjang tetap sebagai keluaran. Bit hasil ini disebut sebagai nilai *hash* dari data masukan. Nilai *hash* dapat juga disebut sebagai kode *hash*, *digest*, ataupun *hash* saja.

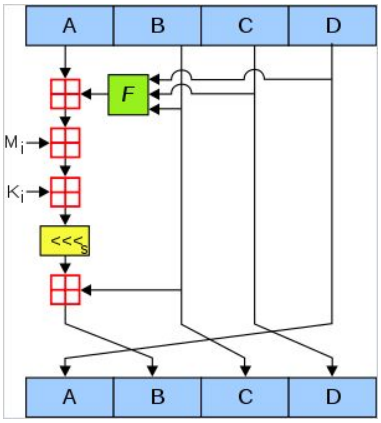
Diantara berbagai tujuan fungsi *hash*, terdapat jenis fungsi yang memenuhi kriteria untuk dipergunakan untuk tujuan kriptografi. Jenis fungsi ini disebut sebagai fungsi *hash* kriptografi (*cryptographic hash function*). Sifat-sifat yang harus dipenuhi oleh sebuah fungsi *hash* kriptografi yaitu:

1. *Deterministic*
   Prosedur *hash* harus menjamin hasil nilai *hash* yang selalu sama untuk input yang sama, tanpa memperdulikan faktor eksternal seperti waktu dan jumlah eksekusi.
2. *Collision resistant*
   Semua nilai dalam rentang keluaran fungsi *hash* harus memiliki probabilitas kemunculan yang sama. Dengan demikian, tidak boleh terdapat dua input berbeda yang memiliki *hash* yang sama persis.
3. *Non-invertible*

Prosedur pada fungsi *hash* bersifat satu arah, sehingga nilai *hash* dari sebuah data tidak dapat digunakan untuk melacak kondisi asli dari data tersebut.

B. Algoritma Message Digest *(MD5)*

Algoritma Message Digest MD5 adalah algoritma hash yang dikembangkan oleh Ronald Rivest pada tahun 1991 untuk menggantikan algoritma MD4 yang merupakan pendahulunya. Algoritma ini menciptakan *hash* dengan melakukan operasi *bitwise* internal secara berulang kali terhadap data masukan. Data yang masuk dipecah sebagai blok-blok yang berukuran 512 bit untuk selanjut dipecah kembali menjadi empat "bongkahan" atau *chunk* berukuran 128 bit (A, B, C, dan D) dalam *loop* utama. Operasi *bitwise* selanjutnya dilakukan terhadap tiap *chunk* sehingga menghasilkan keluaran berukuran 128 bit di akhir proses. Proses kerja algoritma MD5 dapat dilihat pada Gambar 1.

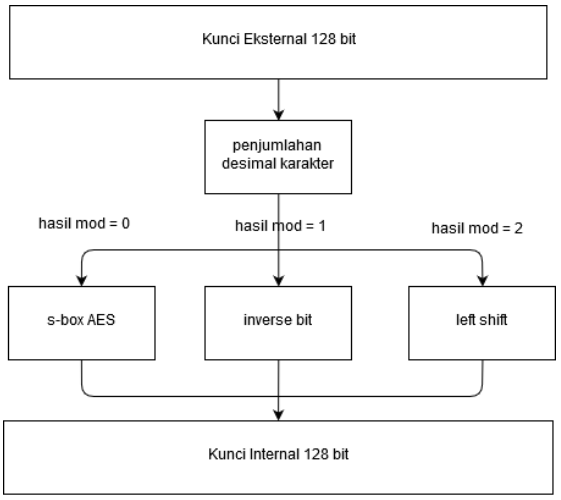


Gambar 1. Alur proses kerja algoritma MD5. Fungsi internal dilambangkan sebagai huruf F, penjumlahan dilambangkan sebagai kotak merah, dan operasi *circular left shift* sebagai kotak kuning. (*Wikipedia.org*)

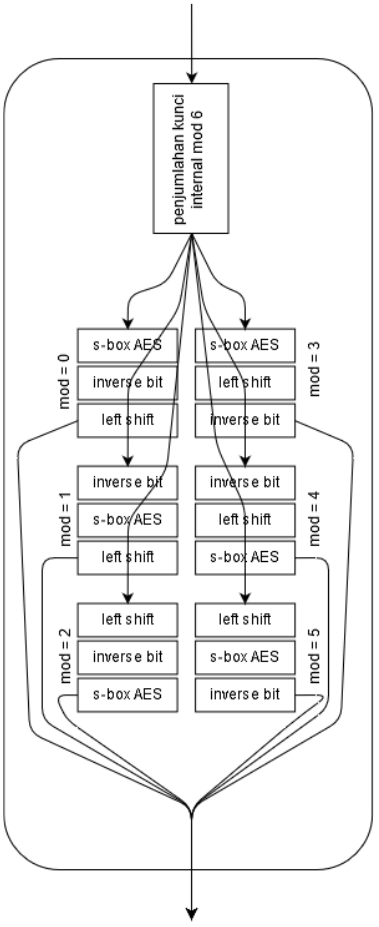
C. Algoritma 474

Algoritma 474 adalah algoritma kriptografi berjenis blok cipher yang diciptakan oleh Husnulzaki Wibisono Haryadi dan Aya Aurora Rimbamorani sebagai tugas pengganti UTS mata kuliah IF4020 Kriptografi tahun 2018/2019. Terinspirasi dari algoritma kriptografi lain yang bernama DES (*Data Encryption Standard*) dan AES (*Advanced Encryption Standard*), algoritma ini memanfaatkan 32 putaran jaringan *Feistel* dan kunci sepanjang 128 bit untuk melakukan operasi enkripsi-dekripsi datanya.

Karakteristik algoritma 474 terletak pada penggunaan tiga jenis operasi *bitwise* yang yang dieksekusi berurutan yang digunakan dalam pembangkitan kunci internal dan pada fungsi-f dalam jaringan Feistel. Ketiga operasi *bitwise* ini yaitu *circular left shift*, *inverse bit*, dan permutasi *S-box AES*. Urutan eksekusi dari ketiga operasi ini tidak tetap dan ditentukan oleh nilai kunci internal pada setiap putarannya.



Gambar 2. Penggunaan operasi *bitwise* pada pembangkitan kunci internal dalam Algoritma 474



Gambar 3. Penggunaan operasi *bitwise* pada fungsi-f Algoritma 474

III. Rancangan Algoritma

A. Implementasi Algoritma MD5

Algoritma MD5 diimplementasikan dengan mengacu ke *pseudo-code* yang dirilis oleh Ronald Rivest dan diadaptasi oleh Wikipedia. Algoritma ini menggunakan konstanta inisialisasi *buffer* dan tabel fungsi sinus (*T-table*) yang sama dengan implementasi asli, sehingga *hash* yang dihasilkan

diharapkan memiliki nilai yang sama seperti

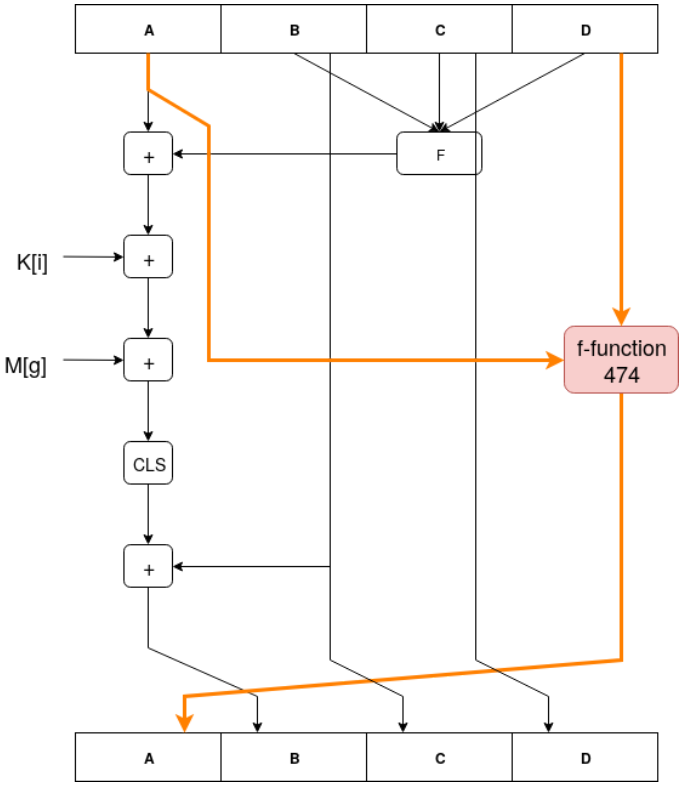
```
//Initialize variables:
var int a0 := 0x67452301   //A
var int b0 := 0xefcdab89   //B
var int c0 := 0x98badcfe   //C
var int d0 := 0x10325476   //D

//Pre-processing: adding a single 1 bit
append "1" bit to message
// Notice: the input bytes are considered as
bits strings,
//  where the first bit is the most
significant bit of the byte.[49]

//Pre-processing: padding with zeros
append "0" bit until message length in bits
≡ 448 (mod 512)
append original length in bits mod 2^64 to
message

//Process the message in successive 512-bit
chunks:
for each 512-bit chunk of padded message
    break chunk into sixteen 32-bit words
M[j], 0 ≤ j ≤ 15
    //Initialize hash value for this chunk:
    var int A := a0
    var int B := b0
    var int C := c0
    var int D := d0
    //Main loop:
    for i from 0 to 63
        var int F, g
        if 0 ≤ i ≤ 15 then
            F := (B and C) or ((not B) and
D)
            g := i
        else if 16 ≤ i ≤ 31 then
            F := (D and B) or ((not D) and
C)
            g := (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47 then
            F := B xor C xor D
            g := (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63 then
            F := C xor (B or (not D))
            g := (7×i) mod 16
        //Be wary of the below definitions
of a,b,c,d
        F := F + A + K[i] + M[g]
        A := D
        D := C
        C := B
        B := B + leftrotate(F, s[i])
    end for
    //Add this chunk's hash to result so
far:
    a0 := a0 + A
    b0 := b0 + B
    c0 := c0 + C
    d0 := d0 + D
end for

var char digest[16] := a0 append b0 append
c0   append d0 //(Output is in
little-endian)
```

Tabel 1. *Pseudocode* Algoritma MD5 (*Wikipedia*)

Untuk menyederhanakan pengujian, jenis masukan yang dapat diterima oleh implementasi MD5 pada laporan ini dibatasi pada tipe data *string* saja. Algoritma akan dibungkus dalam sebuah kelas yang dipanggil oleh sebuah program utama. Saat dieksekusi, program utama akan membaca sebuah *file* teks sebagai data yang akan di-*hash*, dan hasilnya akan ditampilkan pada layar.

B. Implementasi Algoritma 474

Perubahan yang dilakukan terhadap algoritma MD5 pada laporan ini dilakukan dengan menambahkan fungsi-f dari algoritma 474 ke dalam *loop* utama dari MD5. Fungsi-f akan disisipkan sebelum perubahan nilai *chunk* pada akhir *loop*, dengan nilai *chunk* D sebagai parameter input dan *chunk* A sebagai parameter *key*. Ilustrasi implementasi modifikasi ini dapat dilihat pada Gambar 4.



Gambar 4. Ilustrasi implementasi fungsi-f Algoritma 474 di dalam Algoritma MD5

IV. Pengujian dan Analisis

A. Uji Coba Eksekusi Algoritma

Uji coba eksekusi bertujuan untuk memeriksa apakah algoritma yang diimplementasikan mampu dieksekusi dengan benar dan memberikan hasil yang diharapkan. Pada uji coba ini, beberapa *file* teks dengan ukuran berbeda akan berperan sebagai masukan. Hasil uji coba eksekusi dapat dilihat pada

Tabel 2.a, 2.b, dan 2.c.

Plainteks	**`Hello World!`**
MD5 hasil implementasi	`89ee5cd7f7b14be6e10a3b97d665e872`
MD5 dengan modifikasi	`9cf7b87b20f954e4a71f540311ce5df`
hashlib.md5() [Python 3.7]	`ed076287532e86365e841e92bfc50d8c`

Tabel 2.a. Perbandingan hasil uji coba eksekusi untuk teks pendek

Plainteks	**`all work and no play makes jack a dull boy`**
MD5 hasil implementasi	`a4960742469b987d4a8291a32ab3c50`
MD5 dengan modifikasi	`2c76220a283344beb29366f252a710a7`
hashlib.md5() [Python 3.7]	`dd711b8ac1e7381d317dd6a9519120e4`

Tabel 2.b. Perbandingan hasil uji coba eksekusi untuk teks sedang

Plainteks	**`Praesent felis arcu, ullamcorper bibendum faucibus in, aliquam eu leo. Aenean finibus nibh felis, vel consequat arcu pellentesque non. Curabitur elit mi, convallis sit amet blandit vel, congue sed diam. Aenean a magna mauris. Sed sagittis neque vel arcu vulputate sollicitudin.`**
MD5 hasil implementasi	`6668c13ea37a9f8ba888e4ec31119ba3`
MD5 dengan modifikasi	`e13f4ad1817597c1507d8b96db3f330f`
hashlib.md5() [Python 3.7]	`f0df7277ef00fef273a0c588bbe5ad04`

Tabel 2.b. Perbandingan hasil uji coba eksekusi untuk teks panjang

Dari hasil uji coba dapat dilihat bahwa baik algoritma MD5 maupun MD5 termodifikasi yang diimplementasi sendiri mampu menghasilkan *hash* yang secara konsisten berukuran 32 heksadesimal (128 bit) tanpa mempedulikan ukuran teks masukan. Terlihat pula bahwa saat dibandingkan dengan fungsi MD5 pada modul *hashlib* (*library* algoritma *hash* standar Python), fungsi MD5 buatan sendiri selalu menghasilkan nilai *hash* yang berbeda. Hal ini diduga disebabkan karena kesalahan interpretasi *pseudocode* pada saat implementasi. Dengan demikian, implementasi algoritma MD5 pada laporan ini tidak dapat dianggap sebagai implementasi standar.

B. Uji Coba Sensitivitas Algoritma

Uji coba sensitivitas bertujuan untuk memeriksa seberapa peka algoritma MD5 termodifikasi terhadap perubahan nilai pada masukan. Uji coba dilakukan dengan membandingkan nilai *hash* dari dua teks masukan yang berbeda. Nilai yang akan menjadi masukan dapat dilihat pada Tabel 3.a, sedangkan hasil uji coba dapat dilihat pada Tabel 3.b.

Kode masukan	Teks
Teks 1	The MD5 message-digest algorithm is a widely used hash function producing a 128-bit hash value. Although MD5 was initially designed to be used as a cryptographic hash function, it has been found to suffer from extensive vulnerabilities.
Teks 2	The MD5 message-digest algorithm is a widely used hash function producing a 128-bit hash value. Although MD5 was initially designed to be used as a cryptographic hash function, it has been found to suffer from extensive vulnerabilities..
Teks 3	The MD5 message-digest algorithm is a widely used hash function producing a 129-bit hash value. Although MD5 was initially designed to be used as a cryptographic hash function, it has been found to suffer from extensive vulnerabilities.

Tabel 3.a. Teks masukan untuk pengujian sensitivitas

Kode masukan	Nilai *hash*
Teks 1	`7857cbd878e6b96025360a89d6275570`

Teks 2	`666dd093d7b0e1d636c5d3b1caa96250`
Teks 3	`8569f22b41d2b31a2f98ff8abe8e4aa3`

Tabel 3.b. Teks hasil pengujian sensitivitas

Dari hasil uji coba, dapat terlihat bahwa baik Teks 1, Teks 2, dan Teks 3 menghasilkan *hash* yang sangat berbeda. Algoritma mampu mendeteksi perbedaan besar seperti penambahan panjang data pada Teks 2, maupun perbedaan yang sangat kecil, seperti pada Teks 3 (perubahan karakter '8' menjadi karakter '9' hanya mengubah satu bit saja). Dengan demikian, pengujian ini membuktikan bahwa algoritma MD5 termodifikasi memiliki sensitivitas tinggi terhadap perubahan data.

C. Uji Coba Kecepatan Eksekusi

Uji coba kecepatan eksekusi dilakukan untuk menguji kinerja dan kemampuan algoritma dalam membangkitkan nilai *hash*. Dalam uji coba ini dipergunakan dua buah data, data 'pendek' yang berukuran 236 karakter ASCII, dan data 'panjang yang berukuran 10000 karakter ASCII. Pengukuran waktu dilakukan dengan memanggil fungsi *time.time()* dari *library* standar Python saat sebelum dan sesudah blok kode pembangkitan *hash*.

Algoritma	Waktu eksekusi
MD5 hasil implementasi	0.019831418991088867
MD5 dengan modifikasi	0.012295722961425781
hashlib.md5() [Python 3.7]	4.76837158203125e-06

Tabel 4.a. Waktu pembangkitan *hash* untuk data sepanjang 236 karakter ASCII

Algoritma	Waktu eksekusi
MD5 hasil implementasi	3.6149985790252686
MD5 dengan modifikasi	2.728424072265625
hashlib.md5() [Python 3.7]	6.198883056640625e-0

Tabel 4.b. Waktu pembangkitan *hash* untuk data sepanjang 10.000 karakter ASCII

Dari hasil uji coba, terlihat bahwa baik algoritma MD5 maupun MD5 termodifikasi memiliki waktu eksekusi yang jauh lebih lama daripada fungsi bawaan Python. Hal ini dapat terjadi karena fungsi milik Python telah terstandarisasi dan teroptimasi dengan baik apabila dibandingkan dengan algoritma implementasi sendiri. Terdapat hal menarik dari hasil pengujian, yaitu waktu eksekusi MD5 termodifikasi yang lebih cepat daripada algoritma yang tidak termodifikasi, kendatipun algoritma termodifikasi memiliki prosedur *hash* yang lebih panjang.

V. Kesimpulan dan Saran

Algoritma MD5 merupakan algoritma yang cukup mangkus dalam masalah pembangkitan *hash*. Hal ini berimbas pada modifikasi pada implementasi yang menghasilkan algoritma yang mangkus pula. Adapun kelemahan dari MD5 terdapat pada ukuran *hash*-nya yang tergolong kecil sehingga mudah untuk dipecahkan dengan komputer modern. Modifikasi pada algoritma pada laporan ini hanya akan menambah kompleksitas prosedur *hashing* tanpa menambah ukuran akhir *hash*, sehingga kelemahan tersebut belum dapat ditangani. Untuk mengatasi kelemahan ini, diperlukan uji kolisi dan kompleksitas, yang keduanya belum dapat dilakukan karena masalah komputasi.

Berkaca dari hasil pekerjaan yang telah dilakukan, penulis menyimpulkan bahwa Python bukanlah pilihan bahasa pemrograman yang paling baik dalam menangani data biner atau prosedur-prosedur *bitwise* karena karakteristik bahasa Python yang sangat *high-level*. Dengan demikian, penulis menyarankan untuk menggunakan bahasa pemrograman yang lebih *low-level* seperti C untuk menangani data biner.

VI. Ucapan Terima Kasih

Penulis mengucapkan terima kasih yang sebesar-besarnya kepada Allah SWT, yang dengan rahmat-Nya mengizinkan penulis untuk menyelesaikan laporan ini. Penulis juga ingin berterima kasih kepada Bapak Rinaldi Munir selaku dosen mata kuliah IF4020 Kriptografi yang telah dengan telaten membimbing dan mencurahkan ilmunya kepada penulis selama setahun penuh.

References

[1] Munir, R., *Slide Kuliah IF4020 Kriptografi,* Fungsi *Hash*, 2018.
[2] Munir, R., *Slide Kuliah IF4020 Kriptografi*, Algoritma MD5, 2018.
[3] Rivest, R., *The MD5 Message-Digest Algorithm*, RFC 1321, DOI 10.17487/RFC1321, April 1992.

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, 10 Mei 2019

Husnulzaki Wibisono Haryadi 13515005