Jurnal Sains Komputer & Informatika (J-SAKTI)
Volume 6 Nomor 1, Maret 2022, pp. 172-187
ISSN: 2548-9771/EISSN: 2549-7200
https://tunasbangsa.ac.id/ejurnal/index.php/jsakti

Perbandingan Performa Algoritma Md5 Dan Sha-256 Dalam Membangkitkan Identitas File

Imam Saputra[1], Surya Darma Nasution[2]
[1,2]Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Budi Darma
Jl. Sisingamangaraja No.338, Siti Rejo I, Kec. Medan Kota, Kota Medan, Sumatera Utara 20219, (061) 7875998
email : saputraimam69@gmail.com, surya.darma.nasution1@gmail.com

Abstract

The ease of accessing the internet makes it very easy for humans to share various kinds of files. These files are very easy to download, so it is not uncommon for these files to become duplicates or the same files are stored on personal storage media. This of course will take up personal storage space owned. To avoid this, the steps that can be taken are to provide the identity of each file that represents the contents of the file. So that it can be known whether the contents of the file are the same or not. To generate the identity of a file, a special algorithm is needed that can represent the contents of the file. Algorithms that can be used to generate file identities are algorithms that fall into the hash function category. But in its development there are many algorithms in hash functions that can be used to generate file identities. The algorithms that are often used are the MD5 algorithm and the SHA-256 algorithm. The MD5 and SHA-256 algorithms have different algorithm structures so they have different performance when generating the identity of a file. By comparing the performance between the MD5 and SHA-256 algorithms, we will get an algorithm that has better performance when used to generate file identities.

***Keywords**: Hash Function, MD5, SHA-256*

Abstrak

Kemudahan dalam mengakses internet membuat manusia dengan sangat mudah berbagi berbagai macam file. File-file tersebut dengan sangat mudah di unduh, dengan begitu tidak jarang file – file tersebut menjadi duplikat atau file yang sama tersimpan pada media penyimpanan pribadi. Hal ini tentunya akan memakan ruang penyimpanan pribadi yang dimiliki. Untuk menghindari hal tersebut maka langkah yang dapat ditempuh adalah dengan memberikan identitas dari setiap file yang merepresentasikan isi dari file tersebut. Sehingga dapat diketahui apakah isi atau konten dari file tersebut sama atau tidak. Untuk membangkitkan identitas dari sebuah file diperlukan algoritma khusus yang dapat merepresentasikan isi dari file tersebut. Algoritma yang dapat digunakan untuk membangkitkan identitas file adalah algoritma yang masuk ke dalam kategori fungsi hash. Namun dalam perkembangannya banyak sekali terdapat algoritma dalam fungsi hash yang dapat digunakan untuk membangkitkan identitas file. Algoritma yang sering digunakan adalah algoritma MD5 dan algoritma SHA-256. Algoritma MD5 dan SHA-256 memiliki struktur algoritma yang berbeda sehingga memiliki performa yang berbeda pula ketika membangkitkan identitas dari sebuah file. Dengan membandingkan performa antara algoritma MD5 dan SHA-256 maka akan di dapatkan algoritma yang memiliki performa yang lebih baik ketika digunakan untuk membangkitkan identitas file.

***Kata kunci**: Fungsi Hash, MD5, SHA-256*

1. PENDAHULUAN

Kebiasaan manusia berubah seiring waktu dan menyesuaikan dengan perkembangan dunia teknologi yang terjadi. Sekarang ini masanya Revolusi Industri 4.0 di mana kemajuan di dunia teknologi berkembang sangat pesat

salah satunya pada bagian *Cloud Computing*. Sebelumnya kebiasaan manusia dalam menyimpan file-file digital menggunakan media penyimpanan pribadi, namun sekarang kebiasaan itu telah berubah dan media penyimpanan yang di gunakan untuk menyimpan *file-file* yang dimiliki telah berbasis *Cloud*. Hal ini karena penyimpanan berbasis *Cloud* tergolong lebih praktis, dapat di akses di mana pun selagi memiliki koneksi internet, dapat diakses dari berbagai perangkat. Hal inilah yang mengubah kebiasaan manusia dalam menyimpan *file-file* yang di milikinya.

Ada banyak *platform* yang menyediakan penyimpanan *Cloud* yang gratis dan memiliki banyak fitur, seperti dapat melakukan *share* atau berbagi pakai *file* secara umum. Karena bersifat gratis dan dapat di akses oleh siapa saja maka siapa pun dapat mengunggah *file* ke penyimpanan *cloud* tersebut. Hal ini tentunya memaksa pengelola penyimpanan *cloud* untuk menyediakan ruang penyimpanan yang begitu besar, terlebih jika ada banyak *user* atau pengguna yang mengunggah *file* yang sama, tentunya akan memakan ruang penyimpanan yang cukup besar hanya untuk *file* yang sama. Cara yang dapat di lakukan oleh pengelola penyimpanan *cloud* tersebut adalah memberikan identitas dari setiap *file* yang diunggah oleh pengguna. Sehingga jika ada *file* yang sama sistem akan mempertahankan satu *file* untuk di simpan.

Dalam ilmu kriptografi terdapat fungsi *hash* yang dapat di gunakan dalam membangkitkan identitas dari sebuah *file*. Dengan memanfaatkan fungsi *hash* ini maka akan di dapatkan sebuah nilai *hash* yang merupakan representasi isi dari sebuah *file*. Jika isi dari sebuah *file* sama persis atau *file* tersebut duplikat maka akan menghasilkan nilai *hash* yang sama. Jika terjadi perubahan sedikit saja pada isi dari *file* tersebut maka akan menghasilkan nilai *hash* yang berbeda pula. Dengan memanfaatkan kemampuan fungsi *hash* tersebut maka dapat digunakan untuk membangkitkan identitas dari sebuah *file.*

Dalam menghitung nilai *hash* dari sebuah *file* terdapat beberapa tahapan yang harus dilalui sehingga menghasilkan nilai *hash* yang unik di mana jika ada *file* yang berbeda tidak akan menghasilkan nilai *hash* yang sama. Dalam prosesnya untuk menghasilkan nilai *hash* setiap algoritma fungsi *hash* akan melakukan operasi terhadap nilai keseluruhan dari sebuah *file*. Hal ini tentu akan membuat proses yang dilakukan cukup panjang, membutuhkan penggunaan prosesor dan RAM yang besar, terlebih jika ukuran *file* yang dihitung nilai hash nya berukuran besar.

Algoritma fungsi *hash* yang sering digunakan saat ini adalah algoritma MD5 dan algoritma SHA-256. Ada banyak sekali penelitian terkait penggunaan dari kedua algoritma ini. Algoritma MD5 dapat digunakan untuk memverifikasi keutuhan dari suatu *file* dengan cara menghitung nilai MD5 dari *file* tersebut dan membandingkannya dengan nilai MD5 yang diperoleh dari si pengirim [1]. Pada penelitian tersebut algoritma MD5 digunakan untuk memverifikasi apakah terjadi kerusakan pada saat terjadi pengiriman sebuah *file*. Algoritma MD5 juga dapat digunakan untuk pengamanan *password* pada untuk *login* pada sebuah *website* [2]. Pada penelitian tersebut

algoritma MD5 digunakan untuk membangkitkan *password* sekali pakai sehingga dapat meningkatkan keamanan *login* dari sebuah *website*. Metode SHA-256 terbukti dapat meningkatkan kompleksitas kunci pada AES 128 bits terlihat dari hasil uji *Randomness Test* bahwa kunci pada AES 128 bits yang menggunakan SHA-256 telah lolos semua tes uji sedangkan yang tidak menggunakan SHA-256 pada tahap Uji Poker gagal dan Uji Run gagal [3]. Pada penelitian tersebut algoritma SHA-256 digunakan untuk meningkatkan kompleksitas kunci algoritma AES 128. Algoritma SHA-256 bisa digunakan untuk mendeteksi orisinalitas citra digital hasil pemindaian ijazah ataupun transkrip nilai. Algoritma SHA-256 mampu mendeteksi perubahan pada citra hasil pemindaian ijazah atau transkrip nilai meskipun perubahan yang terjadi hanya satu piksel saja. Algoritma SHA-256 menghasilkan perbedaan yang sangat signifikan pada nilai hash nya walaupun perubahan yang terjadi pada citra hasil pemindaian ijazah atau transkrip nilai hanya satu piksel saja [4].

Dari beberapa penelitian yang telah dilakukan di atas menunjukkan bahwa algoritma MD5 dan SHA-256 banyak digunakan di berbagai bidang keamanan data, untuk itu perlu di lakukan perbandingan performa antara kedua algoritma, agar dapat diketahui algoritma yang memiliki performa yang lebih baik yang di tinjau dari berbagai paramater seperti penggunaan prosesor, penggunaan RAM, waktu yang dibutuhkan untuk menghasilkan nilai *hash* dan lain sebagainya.

2. METODOLOGI PENELITIAN

2.1. Teknik Pengumpulan Data

Teknik pengumpulan data pada penelitian ini dilakukan dengan cara mencari referensi yang relevan dengan permasalahan yang berkaitan dengan fungsi hash, baik itu algoritma MD5 ataupun SHA-256. Referensi yang digunakan berupa jurnal-jurnal ataupun buku-buku yang berkaitan dengan topik yang diteliti.

2.2. Algoritma MD5

Algoritma MD5 merupakan algoritma yang menggunakan nilai masukkan dengan panjang sembarang dan menghasilkan nilai keluaran sepanjang 128 bit [7]. Secara komputasi tidak dimungkinkan untuk mendapatkan nilai keluaran yang sama dari nilai masukkan yang berbeda. Algoritma ditujukan untuk tanda tangan digital, dan algoritma ini dirancang cukup cepat bekerja pada mesin dengan arsitektur 32 bit. Algoritma MD5 tidak memerlukan tabel subsitusi yang berukuran besar dan merupakan pengembangan dari algoritma MD4. Algoritma MD5 terdiri dari beberapa langkah yaitu [7]:

a) *Append Padding Bit*

Penambahan panjang pesan dilakukan jika panjang setiap blok pesan tidak kongruen dengan 448 modulus 512. Penambahan panjang pesan dilakukan dengan menambahkan bit 1 kemudian diikuti bit 0 sehingga

panjang pesan kongruen dengan 448 modulus 512. Secara keseluruhan setidaknya ada penambahan 1 bit dan paling banyak penambahan 512 bit.

b) *Append Lenght*

Penambahan panjang pesan sebelum dilakukan penambahan *padding bit* direpresentasikan dalam bilangan biner 64 bit. Jika panjang pesan lebih dari 264 maka hanya 64 bit orde terendah yang digunakan.

c) *Initialize MD Buffer*

Empat *buffer* (A, B, C, D) berukuran 32 bit digunakan untuk menghitung nilai hash. Empat buffer itu diinisialisasikan dalam bentuk heksadesimal dengan nilai sebagai berikut :

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

d) *Process Message in 16-Word Blocks*

Mendefinisikan empat fungsi tambahan dengan masukkan 32 bit dan keluaran 32 bit. Fungsi tambahan tersebut dinotasikan sebagai berikut :

F(X,Y,Z) = (X ^ Y) v (~X ^ Z), G(X,Y,Z) = (X ^ Z) v (Y ^ ~Z)

H(X,Y,Z) = (X ⊕ Y ⊕ Z), I(X,Y,Z) = Y ⊕ (X v ~Z)

fungsi G, H dan I mirip dengan fungsi F dan bertindak sebagai *bitwise parallel* untuk menghasilkan keluaran dari X, Y dan Z sedemikian rupa sehingga jika bit X, Y dan Z adalah independen dan tidak bias maka setiap bit keluaran dari fungsi G, H dan I juga independen dan tidak bias. Langkah ini menggunakan 64 elemen tabel T yang di bangun dari fungsi sinus. Berikut ini adalah tabel elemen T :

Tabel 1. Elemen T

D76AA478	E8C7B756	242070DB	C1BDCEEE	F57C0FAF	X4787C62A	A8304613	FD469501
698098D8	8B44F7AF	FFFF5BB1	895CD7BE	6B901122	FD987193	A679438E	49B40821
F61E2562	C040B340	265E5A51	E9B6C7AA	D62F105D	02441453	D8A1E681	E7D3FBC8
21E1CDE6	C33707D6	F4D50D87	455A14ED	A9E3E905	FCEFA3F8	676F02D9	8D2A4C8A
FFFA3942	8771F681	6D9D6122	FDE5380C	A4BEEA44	4BDECFA9	F6BB4B60	BEBFBC70
289B7EC6	EAA127FA	D4EF3085	04881D05	D9D4D039	E6DB99E5	1FA27CF8	C4AC5665
F4292244	432AFF97	AB9423A7	FC93A039	655B59C3	8F0CCC92	FFEFF47D	85845DD1
6FA87E4F	FE2CE6E0	A3014314	4E0811A1	F7537E82	BD3AF235	2AD7D2BB	EB86D391

Ronde 1: Misalkan [abcd k s i] dinotasikan sebagai berikut:

a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s)

Ronde 2: Misalkan [abcd k s i] dinotasikan sebagai berikut:

a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s)

Ronde 3: Misalkan [abcd k s i] dinotasikan sebagai berikut:

a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s)

Ronde 4: Misalkan [abcd k s i] dinotasikan sebagai berikut:

a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s).

Langkah selanjutnya adalah menambahkan empat register dengan nilai sebelum operasi pada blok ini dimulai

A = A+AA, B = B+BB, C = C+CC, D = D+DD

e) *Output*

Message Digest dihasilkan sebagai output berasal dari nilai A, B, C, D. Yang di mulai dari byte orde terendah yaitu A sampai byte orde tertinggi yaitu D.

Jurnal Sains Komputer & Informatika (J-SAKTI)
Volume 6 Nomor 1, Maret 2022, pp. 172-187
ISSN: 2548-9771/EISSN: 2549-7200
https://tunasbangsa.ac.id/ejurnal/index.php/jsakti

2.3. Algoritma SHA-256

Secure Hash Algorithm (SHA) 256 merupakan fungsi *hash* yang umum digunakan, sampai saat ini belum ada yang dapat memecahkan algoritma fungsi hash SHA-256. Algoritma SHA-256 memiliki 8 langkah pengerjaan yaitu sebagai berikut [8]:

a) Tambahkan *Padding Bit*
Pesan diisi sehingga panjangnya kongruen dengan 448, modulus 512. Padding 1 bit ditambahkan di akhir pesan, diikuti oleh banyaknya nol yang diperlukan sehingga panjang bit sama dengan 448 modulus 512.

b) Panjang *Append*
Representasi panjang pesan 64 bit ditambahkan pada hasil akhirnya, langkah ini untuk membuat panjang pesan kelipatan 512 bit.

c) *Parsing* Pesan
Pesan padding diuraikan menjadi N blok pesan 512 bit, M(1), M(2),...M(N), dengan menambahkan blok 64 bit.

d) Inisialisasi Nilai *Hash*
Nilai hash awal, H(0) diatur, terdiri dari delapan kata 32 bit, dalam bentuk heksadesimal.

Tabel 2. *Initial Hash Value*

$H_0^{(0)}$	$H_1^{(0)}$	$H_2^{(0)}$	$H_3^{(0)}$	$H_4^{(0)}$	$H_5^{(0)}$	$H_6^{(0)}$	$H_7^{(0)}$
6A09E667	BB67EA85	3C6EF372	A54FF53A	510E527F	9B05688C	1F83D9AB	5BE0CD19

e) Mempersiapkan jadwal pesan
SHA-256 menggunakan jadwal pesan enam puluh empat kata 32 bit, kata-kata dari jadwal pesan diberi label W0, W1, ..., W63.

$$Wt = \begin{cases} M_t^{(t)} & 0 \le t \le 15 \\ \sigma_1^{(256)}(W_{i-2}) + W_{i-7} + \sigma_0^{(256)}(W_{i-15}) + W_{i-16}, & 16 \le t \le 63 \end{cases}$$

Di mana :

$$\sigma_1^{(256)}(W_{i-2}) = \big((W_{i-2})ROTR\ 17\big) \oplus \big((W_{i-2})ROTR\ 19\big) \oplus \big((W_{i-2})SHR10\big)$$

$$\sigma_0^{(256)}(W_{i-15}) = \big((W_{i-15})ROTR\ 7\big) \oplus \big((W_{i-15})ROTR\ 18\big) \oplus ((W_{i-15})SHR3)$$

Wt = Blok pesan yang baru
Mt = Blok pesan yang lama
W_{i-2} = Blok pesan dari W ke i-2
W_{i-15} = Blok pesan dari W ke i-15
ROTR = *Rotate Right*
SHR = *Shift Right*
$\oplus$ = Operator *XOR*

f) Inisialisasi delapan variabel kerja a, b, c, d, e, f, g, dan h dengan nilai *hash* (i-1)
For t= 0 *to* 63
{

$$T_1 = h + \sum_1^{(256)}(e) + Ch(e,f,g) + K_1^{(256)} + W_t \quad (1)$$

$$T_2 = \sum_0^{(256)}(a) + Maj(a,b,c) \quad (2)$$

$$h = g, \quad g = f, \quad f = e, \quad e = d + T_1, \quad d = c, \quad c = b, \quad b = a, \quad a = T_1 + T_2$$

Jurnal Sains Komputer & Informatika (J-SAKTI)
Volume 6 Nomor 1, Maret 2022, pp. 172-187
ISSN: 2548-9771/EISSN: 2549-7200
https://tunasbangsa.ac.id/ejurnal/index.php/jsakti

}

Di mana :

$\sum_{1}^{(256)}(e) = (e\ ROTR\ 6) \oplus (e\ ROTR\ 11) \oplus (e\ ROTR\ 25)$

$\sum_{0}^{(256)}(a) = (a\ ROTR\ 2) \oplus (a\ ROTR\ 13) \oplus (a\ ROTR\ 22)$

$Ch(e, f, g) = (e\ ^\wedge\ f) \oplus (\sim e\ ^\wedge\ g)$

$Maj(a, b, c) = (a\ ^\wedge\ b) \oplus (a\ ^\wedge\ c) \oplus (b\ ^\wedge\ c)$

a, b, c, d, e, f, g, h = Variabel yang berisi pesan heksadesimal

$K_1^{(256)}$ = Konstansta SHA-256

ROTR = *Rotate Right*

$\oplus$ = Operator *XOR*

^ = Operator *AND*

Tabel 3. Konstanta SHA-256

428A2F98	71374491	B5C0FBCF	E9B5DBA5	3956C25B	59F111F1	923F82A4	AB1C5ED5
D807AA98	12835B01	243185BE	550C7DC3	72BE5D74	80DEB1FE	9BDC06A7	C19BF174
E49B69C1	EFBE4786	0FC19DC6	240CA1CC	2DE92C6F	4A7484AA	5CB0A9DC	76F988DA
983E5152	A831C66D	B00327C8	BF597FC7	C6E00BF3	D5A79147	06CA6351	14292967
27B70A85	2E1B2138	4D2C6DFC	53380D13	650A7354	766A0ABB	81C2C92E	92722C85
A2BFE8A1	A81A664B	C24B8B70	C76C51A3	D192E819	D6990624	F40E3585	106AA070
19A4C116	1E376C08	2748774C	34B0BCB5	391C0CB3	4ED8AA4A	5B9CCA4F	682E6FF3
748F82EE	78A5636F	84C87814	8CC70208	90BEFFFA	A4506CEB	BEF9A3F7	C67178F2

g) Menjumlahkan hasil akhir a, b, c, d, e, f, g, h dengan *initial hash value* H(i)
H0(i)=a + H0(i), H1(i)=b + H1(i), H2(i)=c + H2(i), H3(i)=d + H3(i), H4(i)=e + H4(i), H5(i)=f + H5(i), H6(i)=g + H6(i), H7(i) =h + H7(i)

h) Output

$$H_0^{(N)} \,||\, H_1^{(N)} \,||\, H_2^{(N)} \,||\, H_3^{(N)} \,||\, H_4^{(N)} \,||\, H_5^{(N)} \,||\, H_6^{(N)} \,||\, H_7^{(N)}$$

3. HASIL DAN PEMBAHASAN

3.1. Pembahasan

Dalam penelitian ini setiap jenis file dapat dibangkitkan nilai hash-nya, tetapi dalam pembahasan ini akan diberikan contoh sebuah file teks MD5 DAN SHA-256 DALAM MEMBANGKITKAN IDENTITAS FILE dan karakter tersebut diubah menjadi bilangan heksadesimal menggunakan tabel ASCII.

Tabel 4. Nilai Input

4D	44	35	20	44	41	4E	20
53	48	41	2D	32	35	36	20
44	41	4C	41	4D	20	4D	45
4D	42	41	4E	47	4B	49	54
4B	41	4E	20	49	44	45	4E
54	49	54	41	53	20	46	49
4C	45						

3.2. Implementasi Algoritma MD5

Sebelum membangkitkan identitas sebuah *file* menggunakan algoritma MD5 maka nilai *input* diubah ke dalam bentuk biner.

Jurnal Sains Komputer & Informatika (J-SAKTI)
Volume 6 Nomor 1, Maret 2022, pp. 172-187
ISSN: 2548-9771/EISSN: 2549-7200
https://tunasbangsa.ac.id/ejurnal/index.php/jsakti

Tabel 5. Nilai *Input* biner

01001101	01000100	00110101	00100000	01000100	01000001	01001110	00100000
01010011	01001000	01000001	00101101	00110010	00110101	00110110	00100000
01000100	01000001	01001100	01000001	01001101	00100000	01001101	01000101
01001101	01000010	01000001	01001110	01001101	00100000	01001001	01010100
01001011	01000001	01001110	00100000	01001001	01000100	01000101	01001110
01010100	01001001	01010100	01000001	01010011	00100000	01000110	01001001
01001100	01000101						

a) Penambahan *Padding Bit*

Dari tabel 6. di atas diketahui bahwa panjang M=288 bit. Proses berikutnya adalah dengan menambahkan *padding bit* 1 dan sisanya 0 sejumlah k, dengan persamaan sebagai berikut :

$k = l + 1 \equiv 448 \bmod 512$, $k = 400 + 1 \equiv 448 \bmod 512$, $k = 401 \equiv 448 \bmod 512$

$k = 448 - 401$, $k = 48$

Maka banyaknya *padding* bit 0 yang ditambahkan adalah 48 bit.

Tabel 6. Penambahan *Padding* Bit

01001101	01000100	00110101	00100000	01000100	01000001	01001110	00100000
01010011	01001000	01000001	00101101	00110010	00110101	00110110	00100000
01000100	01000001	01001100	01000001	01001101	00100000	01001101	01000101
01001101	01000010	01000001	01001110	01001101	00100000	01001001	01010100
01001011	01000001	01001110	00100000	01001001	01000100	01000101	01001110
01010100	01001001	01010100	01000001	01010011	00100000	01000110	01001001
01001100	01000101	**10000000**	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**

b) Penambahan Panjang *Append*

Penambahan panjang *append* dilakukan dengan penambahan panjang pesan sebanyak 64 bit di akhir. Panjang pesan adalah 400bit sehingga ditambahkan panjang *append* sebagai berikut:

Tabel 7. Penambahan Panjang *Append*

01001101	01000100	00110101	00100000	01000100	01000001	01001110	00100000
01010011	01001000	01000001	00101101	00110010	00110101	00110110	00100000
01000100	01000001	01001100	01000001	01001101	00100000	01001101	01000101
01001101	01000010	01000001	01001110	01001101	00100000	01001001	01010100
01001011	01000001	01001110	00100000	01001001	01000100	01000101	01001110
01010100	01001001	01010100	01000001	01010011	00100000	01000110	01001001
01001100	01000101	10000000	00000000	00000000	00000000	00000000	00000000
00000000	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**	**00000001**	**10010000**

c) Inisialisasi *MD Buffer*

Setelah proses penambahan panjang *append* maka langkah selanjutnya adalah inisialisasi *MD Buffer* di mana nilai ini merupakan sebuah ketentuan yaitu:

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

d) *Process Message in 16-Word Blocks*

Langkah ke 4 adalah melakukan proses pesan menggunakan Fungsi F, G, H dan I untuk setiap 16 *blok word* yaitu pesan 512 bit yang di bagi menjadi 16 bagian dan setiap bagian terdiri dari 32 bit. Proses kompresi terdiri dari 4 ronde untuk setiap 16 *blok word* pesan. Untuk ronde pertama memproses pesan menggunakan fungsi F, ronde kedua memproses pesan dengan menggunakan fungsi G, dan pada ronde ketiga memproses pesan menggunakan fungsi H dan ronde keempat menggunakan fungsi I.

Jurnal Sains Komputer & Informatika (J-SAKTI)
Volume 6 Nomor 1, Maret 2022, pp. 172-187
ISSN: 2548-9771/EISSN: 2549-7200
https://tunasbangsa.ac.id/ejurnal/index.php/jsakti

Sebelum masuk ronde pertama maka pesan di bagi menjadi 16 *blok word* yaitu X0-X15 yang dapat di lihat pada tabel 8. berikut ini:

Tabel 8. *Parsing* Pesan

X0	:	01001101	01000100	00110101	00100000	=	**4D443520**
X1	:	01000100	01000001	01001110	00100000	=	**44414E20**
X2	:	01010011	01001000	01000001	00101101	=	**5348412D**
X3	:	00110010	00110101	00110110	00100000	=	**32353620**
X4	:	01000100	01000001	01001100	01000001	=	**44414C41**
X5	:	01001101	00100000	01001101	01000101	=	**4D204D45**
X6	:	01001101	01000010	01000001	01001110	=	**4D42414E**
X7	:	01001101	00100000	01001001	01010100	=	**4D204954**
X8	:	01001011	01000001	01001110	00100000	=	**4B414E20**
X9	:	01001001	01000100	01000101	01001110	=	**4944454E**
X10	:	01010100	01001001	01010100	01000001	=	**54495441**
X11	:	01010011	00100000	01000110	01001001	=	**53204649**
X12	:	01001100	01000101	10000000	00000000	=	**4C458000**
X13	:	00000000	00000000	00000000	00000000	=	**00000000**
X14	:	00000000	00000000	00000000	00000000	=	**00000000**
X15	:	00000000	00000000	00000001	10010000	=	**00000190**

Setelah membagi pesan menjadi 16 bagian maka selanjutnya adalah masuk ke dalam ronde pertama dengan menggunakan fungsi F, ronde kedua dengan menggunakan fungsi G, dan ronde ketiga menggunakan fungsi H serta ronde keempat dengan fungsi I.

Ronde 1, Putaran 0, K = 0, S = 7, I = 1

i	A	B	C	D
initial	01234567	89ABCDEF	FEDCBA98	76543210

A = B + (A + F (B,C,D) + X [K] + T[I] <<< S)
A = B + (A + F (B,C,D) + X [0] + T[1] <<< 7)
A = 89ABCDEF + (01234567 + F (B,C,D) + 4D443520 + D76AA478 <<< 7)
F (B,C,D) = B and C or not (B) and D
F (B,C,D) = 89ABCDEF and FEDCBA98 or not (89ABCDEF) and 76543210
F (B,C,D) = 88888888 or 76543210
F (B,C,D) = FEDCBA98
A = 89ABCDEF + (01234567 + FEDCBA98 + 4D443520 + D76AA478 <<< 7)
A = 89ABCDEF + (24AED997<<< 7)
A = 89ABCDEF + 576CCB80
A = 040EC2BF

Operasi di atas dilakukan seterusnya hingga putaran ke 64 menggunakan empat fungsi tambahan dan hasilnya berikut ini:

Result	8D2EE287	5C6268EB	FBE3EFF4	B3ABD213

e) *Output*

Message digest yang dihasilkan sebagai *output* merupakan penjumlahan dari *intermediate hash value* dengan *output* dari proses 64 ronde.

A = A + AA = 01234567 + 8D2EE287 = 8E5127EF
B = B + BB = 89ABCDEF + 5C6268EB = E50D36DB
C = C + CC = FEDCBA98 + FBE3EFF4 = F9C0AA8D
D = D + DD = 76543210 + B3ABD213 = 29000525

i	A	B	C	D
output	8E5127EF	E50D36DB	F9C0AA8D	29000525

Jurnal Sains Komputer & Informatika (J-SAKTI)
Volume 6 Nomor 1, Maret 2022, pp. 172-187
ISSN: 2548-9771/EISSN: 2549-7200
https://tunasbangsa.ac.id/ejurnal/index.php/jsakti

Sehingga nilai *hash*nya adalah 8E5127EFE50D36DBF9C0AA8D29000525, nilai inilah yang dijadikan identitas suatu *file* untuk mengetahui *file* yang ganda atau duplikat.

3.3. Implementasi Algoritma SHA-256

Sebelum membangkitkan identitas sebuah *file* menggunakan algoritma SHA-256 maka nilai *input* diubah ke dalam bentuk biner.

Tabel 9. Nilai *Input* biner

01001101	01000100	00110101	00100000	01000100	01000001	01001110	00100000
01010011	01001000	01000001	00101101	00110010	00110101	00110110	00100000
01000100	01000001	01001100	01000001	01001101	00100000	01001101	01000101
01001101	01000010	01000001	01001110	01001101	00100000	01001001	01010100
01001011	01000001	01001110	00100000	01001001	01000100	01000101	01001110
01010100	01001001	01010100	01000001	01010011	00100000	01000110	01001001
01001100	01000101						

a) Penambahan *Padding Bit*

Dari tabel 3.2. di atas diketahui bahwa panjang M=288 bit. Proses berikutnya adalah dengan menambahkan *padding bit* 1 dan sisanya 0 sejumlah k, dengan persamaan sebagai berikut :

$k = l + 1 \equiv 448 \bmod 512 = 400 + 1 \equiv 448 \bmod 512 = 401 \equiv 448 \bmod 512 = 448 - 401 = 48$

Maka banyaknya padding bit 0 yang ditambahkan adalah 48 bit.

Tabel 10. Penambahan *Padding* Bit

01001101	01000100	00110101	00100000	01000100	01000001	01001110	00100000
01010011	01001000	01000001	00101101	00110010	00110101	00110110	00100000
01000100	01000001	01001100	01000001	01001101	00100000	01001101	01000101
01001101	01000010	01000001	01001110	01001101	00100000	01001001	01010100
01001011	01000001	01001110	00100000	01001001	01000100	01000101	01001110
01010100	01001001	01010100	01000001	01010011	00100000	01000110	01001001
01001100	01000101	**10000000**	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**

b) Penambahan Panjang *Append*

Penambahan panjang *append* dilakukan dengan penambahan panjang pesan sebanyak 64 bit di akhir. Panjang pesan adalah 400 bit sehingga ditambahkan panjang *append* sebagai berikut:

Tabel 11. Penambahan Panjang *Append*

01001101	01000100	00110101	00100000	01000100	01000001	01001110	00100000
01010011	01001000	01000001	00101101	00110010	00110101	00110110	00100000
01000100	01000001	01001100	01000001	01001101	00100000	01001101	01000101
01001101	01000010	01000001	01001110	01001101	00100000	01001001	01010100
01001011	01000001	01001110	00100000	01001001	01000100	01000101	01001110
01010100	01001001	01010100	01000001	01010011	00100000	01000110	01001001
01001100	01000101	10000000	00000000	00000000	00000000	00000000	00000000
00000000	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**	**00000001**	**10010000**

c) Parsing Pesan

Pada kasus ini panjang pesan tidak lebih dari 512 sehingga hanya menghasilkan 1 blok 512 bit yaitu $M^{(0)}$. Tahap selanjutnya adalah melakukan *parsing* pesan dengan membagi setiap blok 512 bit menjadi 16 blok berukuran 32 bit.

Jurnal Sains Komputer & Informatika (J-SAKTI)
Volume 6 Nomor 1, Maret 2022, pp. 172-187
ISSN: 2548-9771/EISSN: 2549-7200
https://tunasbangsa.ac.id/ejurnal/index.php/jsakti

Tabel 12. *Parsing* Pesan

$M_0^{(0)}$	:	01001101	01000100	00110101	00100000
$M_1^{(0)}$	:	01000100	01000001	01001110	00100000
$M_2^{(0)}$	:	01010011	01001000	01000001	00101101
$M_3^{(0)}$	:	00110010	00110101	00110110	00100000
$M_4^{(0)}$	:	01000100	01000001	01001100	01000001
$M_5^{(0)}$	:	01001101	00100000	01001101	01000101
$M_6^{(0)}$	:	01001101	01000010	01000001	01001110
$M_7^{(0)}$	:	01001101	00100000	01001001	01010100
$M_8^{(0)}$	:	01001011	01000001	01001110	00100000
$M_9^{(0)}$	:	01001001	01000100	01000101	01001110
$M_{10}^{(0)}$	:	01010100	01001001	01010100	01000001
$M_{11}^{(0)}$	:	01010011	00100000	01000110	01001001
$M_{12}^{(0)}$	:	01001100	01000101	10000000	00000000
$M_{13}^{(0)}$	:	00000000	00000000	00000000	00000000
$M_{14}^{(0)}$	:	00000000	00000000	00000000	00000000
$M_{15}^{(0)}$	:	00000000	00000000	00000001	10010000

d) Inisialisasi Nilai *Hash*

Setelah proses *parsing* pesan maka langkah selanjutnya adalah inisialisasi nilai *hash* di mana nilai ini merupakan sebuah ketentuan yaitu:

$H_0^{(0)}$	$H_1^{(0)}$	$H_2^{(0)}$	$H_3^{(0)}$	$H_4^{(0)}$	$H_5^{(0)}$	$H_6^{(0)}$	$H_7^{(0)}$
6A09E667	BB67EA85	3C6EF372	A54FF53A	510E527F	9B05688C	1F83D9AB	5BE0CD19

e) Penjadwalan Pesan

Kemudian dilakukan proses penjadwalan pesan, langkah ini diawali dengan mengubah setiap blok pesan menjadi bilangan heksadesimal dengan ketentuan sebagai berikut:

$$Wt = \begin{cases} M_t^{(t)} & 0 \le t \le 15 \\ \sigma_1^{(256)}(W_{i-2}) + W_{i-7} + \sigma_0^{(256)}(W_{i-15}) + W_{i-16}, & 16 \le t \le 63 \end{cases}$$

Tabel 13. Penjadwalan Pesan

4D443520	44414E20	5348412D	32353620	444414C41	4D204D45	4D42414E	474B4954
4B414E20	4944454E	54495441	53204649	4C458000	00000000	00000000	00000190
87960CB8	**DA096ED6**	**6EF452A2**	**F212E43E**	**77CF2E6B**	**8125D132**	**325A581D**	**66729154**
1135F958	**B003A875**	**132F9CCE**	**D5180BB0**	**0B655147**	**36F58E4A**	**89604F0F**	**BE23842A**
7D99D43F	**FDA48524**	**EF3D5B27**	**E30E642E**	**B6FBD3B1**	**B35941E5**	**8458DE31**	**3941EFBE**
9B57816A	**E2D7B7FA**	**A0E1D044**	**947A1A90**	**81E366AC**	**278ADDD6**	**45F87EE6**	**931B1160**
0837075C	**8E2F1CAF**	**BFAB6721**	**5432685E**	**4B56AAE9**	**B693783C**	**98647974**	**3AB45A59**
E19B455E	**323907C5**	**7454954A**	**137C00F3**	**30540850**	**1BDFFA7A**	**1CF85529**	**EFEC9D10**

Untuk menjadwalkan pesan ke 16 sampai 63 dilakukan perhitungan sebagai berikut :

$\sigma_1^{(256)}(W_{i-2}) = ((W_{i-2})ROTR\ 17) \oplus ((W_{i-2})ROTR\ 19) \oplus ((W_{i-2})SHR 10)$

$((W_{16-2})ROTR\ 17)$ = $((W_{14})ROTR\ 17)$ = ((00000000) $ROTR$ 17)
= ((00000000 00000000 **00000000 00000000**)$ROTR$ 17)
= (**00000000 00000000 0**0000000 00000000)

$((W_{16-2})ROTR\ 19)$ = $((W_{14})ROTR\ 19)$ = ((00000000) $ROTR$ 19)
= ((00000000 00000**000 00000000 00000000**)$ROTR$ 19)
= (**00000000 00000000 000**00000 00000000)

$((W_{16-2})SHR\ 10)$ = $((W_{14})SHR\ 10)$= ((00000000) SHR 10)
= ((**00000000 00**000000 00000000 00000000)SHR 10)
= (00000000 00000000 00000000 00000000)

$\sigma_1^{(256)}(W_{i-2})$ = (00000000 00000000 00000000 00000000)
(00000000 00000000 00000000 00000000)
(00000000 00000000 00000000 00000000) $\oplus$
= (00000000 00000000 00000000 00000000)
= 00000000

Jurnal Sains Komputer & Informatika (J-SAKTI)
Volume 6 Nomor 1, Maret 2022, pp. 172-187
ISSN: 2548-9771/EISSN: 2549-7200
https://tunasbangsa.ac.id/ejurnal/index.php/jsakti

W_{16-7} = W_9 = 4B414E20

$\sigma_0^{(256)}(W_{i-15}) = ((W_{i-15})ROTR\ 7) \oplus ((W_{i-15})ROTR\ 18) \oplus ((W_{i-15})SHR 3)$

$((W_{16-15})ROTR\ 7)$ = $((W_1)ROTR\ 7)$ = ((44414E20) $ROTR$ 7)
= ((01000100 01000001 01001110 0**0100000**) $ROTR$ 7)
= (**01000000** 10001000 10000010 10011100)

$((W_{16-15})ROTR\ 18)$ = $((W_1)ROTR\ 18)$ = ((44414E20) $ROTR$ 18)
= ((01000100 01000001 **01001110 00100000**) $ROTR$ 18)
= (**10100111 00010000 0**0100010 00100000)

$((W_{16-15})SHR\ 3)$ = $((W_1)SHR\ 3)$ = ((44414E20) SHR 3)
= ((01000100 01000001 01001110 00100**000**) SHR 3)
= (00001000 10001000 00101001 11000100)

$\sigma_0^{(256)}(W_{i-15})$ = (01000000 10001000 10000010 10011100)
(10100111 00010000 00100010 00100000)
(00001000 10001000 00101001 11000100) ⊕
= (11101111 00010000 10001001 01111000)
= EF108978

W_{16-16} = W_0
= 4D443520

$Wt = \sigma_1^{(256)}(W_{i-2}) + W_{i-7} + \sigma_0^{(256)}(W_{i-15}) + W_{i-16}$
Wt = 00000000 + 4B414E20 + EF108978 + 4D443520
Wt = 1 **87960CB8**

Demikian seterusnya hingga $W_{63}^{(i-1)}$, di mana i adalah jumlah blok 512 bit.

f) Inisialisasi Variabel Kerja

Selanjutnya melakukan inisialisasi variabel kerja *a, b, c, d, e, f, g* dan *h* di mana setiap variabel diambil dari *initial hash value* $a=H_0^{(0)}$, $b=H_1^{(0)}$, $c=H_2^{(0)}$, $d=H_3^{(0)}$, $e=H_4^{(0)}$, $f=H_5^{(0)}$, $g=H_6^{(0)}$, $h=H_7^{(0)}$. Selanjutnya dilakukan proses komputasi fungsi *hash* SHA-256 dari t=0 sampai t=63.

Tabel 14. Proses Komputasi Fungsi *Hash* SHA-256

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
Init	6A09E667	BB67AE85	3C6EF372	A54FF53A	510E527F	9B05688C	1F83D9AB	5BE0CD19
t=0	494CBD6D	6A09E667	BB67AE85	3C6EF372	E828520B	510E527F	9B05688C	1F83D9AB
t=1	1AF375D8	494CBD6D	6A09E667	BB67AE85	048E9261	E828520B	510E527F	9B05688C
t=2	18488AD8	1AF375D8	494CBD6D	6A09E667	5A2B7383	048E9261	E828520B	510E527F
t=3	34E89AA8	18488AD8	1AF375D8	494CBD6D	6A09E667	5A2B7383	048E9261	E828520B
t=4	7783AFE8	34E89AA8	18488AD8	1AF375D8	494CBD6D	6A09E667	5A2B7383	048E9261
t=5	6FE15350	7783AFE8	34E89AA8	18488AD8	1AF375D8	494CBD6D	6A09E667	5A2B7383
t=6	A4292982	6FE15350	7783AFE8	34E89AA8	18488AD8	1AF375D8	494CBD6D	6A09E667
t=7	C93FF6B4	A4292982	6FE15350	7783AFE8	34E89AA8	18488AD8	1AF375D8	494CBD6D
t=8	361E1F68	C93FF6B4	A4292982	6FE15350	7783AFE8	34E89AA8	18488AD8	1AF375D8
t=9	3B2108A6	361E1F68	C93FF6B4	A4292982	6FE15350	7783AFE8	34E89AA8	18488AD8
t=10	AF685F21	3B2108A6	361E1F68	C93FF6B4	A4292982	6FE15350	7783AFE8	34E89AA8
t=11	EEA22FBE	AF685F21	3B2108A6	361E1F68	C93FF6B4	A4292982	6FE15350	7783AFE8
t=12	C6C37F33	EEA22FBE	AF685F21	3B2108A6	361E1F68	C93FF6B4	A4292982	6FE15350
t=13	B332D23C	C6C37F33	EEA22FBE	AF685F21	3B2108A6	361E1F68	C93FF6B4	A4292982
t=14	99DB146E	B332D23C	C6C37F33	EEA22FBE	AF685F21	3B2108A6	361E1F68	C93FF6B4
t=15	6257FBA7	99DB146E	B332D23C	C6C37F33	EEA22FBE	AF685F21	3B2108A6	361E1F68
t=16	1FF577D8	6257FBA7	99DB146E	B332D23C	C6C37F33	EEA22FBE	AF685F21	3B2108A6
t=17	F8231502	1FF577D8	6257FBA7	99DB146E	B332D23C	C6C37F33	EEA22FBE	AF685F21
t=18	2190D99D	F8231502	1FF577D8	6257FBA7	99DB146E	B332D23C	C6C37F33	EEA22FBE
t=19	AF3A1A1E	2190D99D	F8231502	1FF577D8	6257FBA7	99DB146E	B332D23C	C6C37F33
t=20	BC556237	AF3A1A1E	2190D99D	F8231502	1FF577D8	6257FBA7	99DB146E	B332D23C
t=21	0E85A6DB	BC556237	AF3A1A1E	2190D99D	F8231502	1FF577D8	6257FBA7	99DB146E
t=22	9123986B	0E85A6DB	BC556237	AF3A1A1E	2190D99D	F8231502	1FF577D8	6257FBA7
t=23	AA3CADE9	9123986B	0E85A6DB	BC556237	AF3A1A1E	2190D99D	F8231502	1FF577D8
t=24	D451A3B3	AA3CADE9	9123986B	0E85A6DB	BC556237	AF3A1A1E	2190D99D	F8231502
t=25	A526A9C7	D451A3B3	AA3CADE9	9123986B	0E85A6DB	BC556237	AF3A1A1E	2190D99D
t=26	63EF0741	A526A9C7	D451A3B3	AA3CADE9	9123986B	0E85A6DB	BC556237	AF3A1A1E
t=27	280B4CC8	63EF0741	A526A9C7	D451A3B3	AA3CADE9	9123986B	0E85A6DB	BC556237
t=28	52E30372	280B4CC8	63EF0741	A526A9C7	D451A3B3	AA3CADE9	9123986B	0E85A6DB
t=29	8AB8EA34	52E30372	280B4CC8	63EF0741	A526A9C7	D451A3B3	AA3CADE9	9123986B
t=30	94032439	8AB8EA34	52E30372	280B4CC8	63EF0741	A526A9C7	D451A3B3	AA3CADE9
t=31	11FC922F	94032439	8AB8EA34	52E30372	280B4CC8	63EF0741	A526A9C7	D451A3B3
t=32	B413630D	11FC922F	94032439	8AB8EA34	52E30372	280B4CC8	63EF0741	A526A9C7
t=33	DFC21EE3	B413630D	11FC922F	94032439	8AB8EA34	52E30372	280B4CC8	63EF0741

Jurnal Sains Komputer & Informatika (J-SAKTI)
Volume 6 Nomor 1, Maret 2022, pp. 172-187
ISSN: 2548-9771/EISSN: 2549-7200
https://tunasbangsa.ac.id/ejurnal/index.php/jsakti

t=34	4D6A2ED1	DFC21EE3	B413630D	11FC922F	94032439	8AB8EA34	52E30372	280B4CC8
t=35	6108980C	4D6A2ED1	DFC21EE3	B413630D	11FC922F	94032439	8AB8EA34	52E30372
t=36	FED07D9B	6108980C	4D6A2ED1	DFC21EE3	B413630D	11FC922F	94032439	8AB8EA34
t=37	08F20B82	FED07D9B	6108980C	4D6A2ED1	DFC21EE3	B413630D	11FC922F	94032439
t=38	1AA301AA	08F20B82	FED07D9B	6108980C	4D6A2ED1	DFC21EE3	B413630D	11FC922F
t=39	F6A41D58	1AA301AA	08F20B82	FED07D9B	6108980C	4D6A2ED1	DFC21EE3	B413630D
t=40	C10F4AEF	F6A41D58	1AA301AA	08F20B82	FED07D9B	6108980C	4D6A2ED1	DFC21EE3
t=41	C0BFCC13	C10F4AEF	F6A41D58	1AA301AA	08F20B82	FED07D9B	6108980C	4D6A2ED1
t=42	18505526	C0BFCC13	C10F4AEF	F6A41D58	1AA301AA	08F20B82	FED07D9B	6108980C
t=43	B69753B4	18505526	C0BFCC13	C10F4AEF	F6A41D58	1AA301AA	08F20B82	FED07D9B
t=44	51D4936A	B69753B4	18505526	C0BFCC13	C10F4AEF	F6A41D58	1AA301AA	08F20B82
t=45	CE68186A	51D4936A	B69753B4	18505526	C0BFCC13	C10F4AEF	F6A41D58	1AA301AA
t=46	B3777712	CE68186A	51D4936A	B69753B4	18505526	C0BFCC13	C10F4AEF	F6A41D58
t=47	33537D6A	B3777712	CE68186A	51D4936A	B69753B4	18505526	C0BFCC13	C10F4AEF
t=48	513CB746	33537D6A	B3777712	CE68186A	51D4936A	B69753B4	18505526	C0BFCC13
t=49	840F87DD	513CB746	33537D6A	B3777712	CE68186A	51D4936A	B69753B4	18505526
t=50	D30C0CB5	840F87DD	513CB746	33537D6A	B3777712	CE68186A	51D4936A	B69753B4
t=51	E89757D7	D30C0CB5	840F87DD	513CB746	33537D6A	B3777712	CE68186A	51D4936A
t=52	A5284AE7	E89757D7	D30C0CB5	840F87DD	513CB746	33537D6A	B3777712	CE68186A
t=53	DCABFFA6	A5284AE7	E89757D7	D30C0CB5	840F87DD	513CB746	33537D6A	B3777712
t=54	22A40C65	DCABFFA6	A5284AE7	E89757D7	D30C0CB5	840F87DD	513CB746	33537D6A
t=55	29C11368	22A40C65	DCABFFA6	A5284AE7	E89757D7	D30C0CB5	840F87DD	513CB746
t=56	EF37E83A	29C11368	22A40C65	DCABFFA6	A5284AE7	E89757D7	D30C0CB5	840F87DD
t=57	3969F370	EF37E83A	29C11368	22A40C65	DCABFFA6	A5284AE7	E89757D7	D30C0CB5
t=58	F56CE4F7	3969F370	EF37E83A	29C11368	22A40C65	DCABFFA6	A5284AE7	E89757D7
t=59	BB607C18	F56CE4F7	3969F370	EF37E83A	29C11368	22A40C65	DCABFFA6	A5284AE7
t=60	296E198E	BB607C18	F56CE4F7	3969F370	EF37E83A	29C11368	22A40C65	DCABFFA6
t=61	A4CD294A	296E198E	BB607C18	F56CE4F7	3969F370	EF37E83A	29C11368	22A40C65
t=62	628F6CC6	A4CD294A	296E198E	BB607C18	F56CE4F7	3969F370	EF37E83A	29C11368
t=63	B89622BB	628F6CC6	A4CD294A	296E198E	BB607C18	F56CE4F7	3969F370	EF37E83A

Untuk *t=0* lakukan perhitungan sebagai berikut :

A	*B*	*c*	*d*	*e*	*f*	*g*	*h*
6A09E667	BB67AE85	3C6EF372	A54FF53A	510E527F	9B05688C	1F83D9AB	5BE0CD19

$T_1 = h + \sum_1^{(256)}(e) + Ch(e,f,g) + K_t^{(256)} + W_t$

h = 5BE0CD19

$\sum_1^{(256)}(e) = (e\ ROTR\ 6) \oplus (e\ ROTR\ 11) \oplus (e\ ROTR\ 25)$

$(e\ ROTR\ 6)$ = $(e\ ROTR\ 6)$ = ((510E527F) $ROTR$ 6)
= ((01010001 00001110 01010010 01**111111**) $ROTR$ 6)
= (**11111**101 01000100 00111001 01001001)

$(e\ ROTR\ 11)$ = $(e\ ROTR\ 11)$ = ((510E527F) $ROTR$ 11)
= ((01010001 00001110 01010**010 01111111**) $ROTR$ 11)
= (**01001111 111**01010 00100001 11001010)

$(e\ ROTR\ 25)$ = $(e\ ROTR\ 25)$ = ((510E527F) $ROTR$ 25)
= ((01010001 **00001110 01010010 01111111**) $ROTR$ 25)
= (**10000111 00101001 00111111 1**0101000)

$\sum_1^{(256)}(e)$ = (11111101 01000100 00111001 01001001)
(01001111 11101010 00100001 11001010)
(10000111 00101001 00111111 10101000) ⊕
= (00110101 10000111 00100111 00101011)
= 3587272B

$Ch(e,f,g)$ = $(e\ \wedge\ f) \oplus (\sim e\ \wedge\ g)$
= (510E527F ^ 9B05688C)⊕ (~510E527F ^1F83D9AB)

$(e\ \wedge\ f)$ = (01010001 00001110 01010010 01111111)
(10011011 00000101 01101000 10001100) ^
= (00010001 00000100 01000000 00001100)

$\sim e$ = (~01010001 00001110 01010010 01111111)
= (10101110 11110001 10101101 10000000)

$(\sim e\ \wedge\ g)$ = (10101110 11110001 10101101 10000000)
(00011111 10000011 11011001 10101011) ^
= (00001110 10000001 10001001 10000000)

Jurnal Sains Komputer & Informatika (J-SAKTI)
Volume 6 Nomor 1, Maret 2022, pp. 172-187
ISSN: 2548-9771/EISSN: 2549-7200
https://tunasbangsa.ac.id/ejurnal/index.php/jsakti

$Ch(e,f,g)$ = (00010001 00000100 01000000 00001100)
(00001110 10000001 10001001 10000000) ⊕
= (00011111 10000101 11001001 10001100)
= 1F85C98C
$K_0^{(256)}$ = 428A2F98
W_t = 4F606F69
T_1 = 5BE0CD19 + 3587272B + 1F85C98C + 428A2F98 + 4F606F69
= 1 **42D85CD1**

$T_2 = \sum_0^{(256)}(a) + Maj(a,b,c)$

$\sum_0^{(256)}(a) = (a\ ROTR\ 2) \oplus (a\ ROTR\ 13) \oplus (a\ ROTR\ 22)$

$(a\ ROTR\ 2)$ = $(a\ ROTR\ 2)$ = ((6A09E667) $ROTR$ 2)
= ((01101010 00001001 11100110 01100**11**) $ROTR$ 2)
= (**11**011010 10000010 01111001 10011001)

$(a\ ROTR\ 13)$ = $(a\ ROTR\ 13)$ = ((6A09E667) $ROTR$ 13)
= ((01101010 00001001 111**00110 01100111**) $ROTR$ 13)
= (**00110011 00111**011 01010000 01001111)

$(a\ ROTR\ 22)$ = $(a\ ROTR\ 22)$ = ((6A09E667) $ROTR$ 22)
= ((01101010 00**001001 11100110 01100111**) $ROTR$ 22)
= (01101010 00**001001 11100110 01100111**)
= (**00100111 10011001 10011**101 10101000)

$\sum_0^{(256)}(a)$ = (11011010 10000010 01111001 10011001)
(00110011 00111011 01010000 01001111)
(00100111 10011001 10011101 10101000) ⊕
= (11001110 00100000 10110100 01111110)
= CE20B47E

$Maj(a,b,c)$ = $(a\ \wedge\ b) \oplus (a\ \wedge\ c) \oplus (b\ \wedge\ c)$
$Maj(a,b,c)$ = (6A09E667 ^ BB67AE85)⊕(6A09E667 ^ 3C6EF372)⊕(BB67AE85 ^ 3C6EF372)
$(a\ \wedge\ b)$ = (01101010 00001001 11100110 01100111)
(10111011 01100111 10101110 10000101) ^
= (00101010 00000001 10100110 00000101)
$(a\ \wedge\ c)$ = (01101010 00001001 11100110 01100111)
(00111100 01101110 11110011 01110010) ^
= (00101000 00001000 11100010 01100010)
$(b\ \wedge\ c)$ = (10111011 01100111 10101110 10000101)
(00111100 01101110 11110011 01110010) ^
= (00111000 01100110 10100010 00000000)
$Maj(a,b,c)$ = (00101010 00000001 10100110 00000101)
(00101000 00001000 11100010 01100010)
(00111000 01100110 10100010 00000000) ⊕
= (00111010 01101111 11100110 01100111)
= 3A6FE667
T_2 = CE20B47E + 3A6FE667 = 1 **08909AE5**
h = g = 1F83D9AB
g = f = 9B05688C
f = e = 510E527F
e = d + T_1 = A54FF53A + 42D85CD1 = E828520B
d = c = 3C6EF372
c = b = BB67AE85
b = a = 6A09E667
a = T_1 + T_2 = 42D85CD1 + 08909AE5 = 4B68F7B6

Jurnal Sains Komputer & Informatika (J-SAKTI)
Volume 6 Nomor 1, Maret 2022, pp. 172-187
ISSN: 2548-9771/EISSN: 2549-7200
https://tunasbangsa.ac.id/ejurnal/index.php/jsakti

g) Menjumlahkan variabel kerja dengan inisial nilai *hash*
Langkah selanjutnya adalah menjumlahkan variabel kerja *a, b, c, d, e, f, g, h* dengan *initial hash value.*

Tabel 15. Penjumlahan dengan *initial hash value*

Variabel	*Initial Hash Value*		*Variabel Kerja*	*Hasil*
$H_0^{(0)}$	6A09E667	+	B89622BB	**22A00922**
$H_1^{(0)}$	BB67EA85	+	628F6CC8	**1DF71B4D**
$H_2^{(0)}$	3C6EF372	+	A4CD294A	**E13C1CBC**
$H_3^{(0)}$	A54FF53A	+	296E198E	**CEBE0EC8**
$H_4^{(0)}$	510E527F	+	0FD38646	**60E1D8C5**
$H_5^{(0)}$	9B05688C	+	D6CCD20D	**71D23A99**
$H_6^{(0)}$	1F83D9AB	+	267DAA66	**46018411**
$H_7^{(0)}$	5BE0CD19	+	0F4F6670	**6B303389**

h) Output
Output dari SHA-256 merupakan penggabungan dari H0(0) sampai H7(0) sebagai berikut :
22A00922 || 1DF71B4D || E13C1CBC || CEBE0EC8 || 60E1D8C5 || 71D23A99 || 46018411 || 6B303389
Sehingga didapat nilai hash dari pesan M adalah sebagai berikut :
22A009221DF71B4DE13C1CBCCEBE0EC860E1D8C571D23A994601841 16B303389

3.7. Hasil

Pada penelitian ini dilakukan beberapa pengujian untuk mengukur kinerja dari algoritma SHA-256 dan MD5 dalam membangkitkan identitas *file*. Di mana sampel *file* yang digunakan terdiri dari lima jenis *file*, sedangkan parameter yang diukur adalah penggunaan prosesor, penggunaan memori dan juga waktu prosesnya. Peneliti menggunakan prosesor Intel Core i5 generasi kedua dengan RAM sebesar 6GB. Hasil dari pengujiannya dapat dilihat pada tabel 16. berikut ini.

Tabel 16. Hasil Pengujian SHA-256

Ekstensi	Ukuran	SHA-256			
		Nilai Hash	Prosesor	Memori	Waktu
MP4	680164 KB	65FB849512720864 78EC854D04DD25A2 0803098776E9C011 D614EAEDC79AF934	31,6%	17,0 MB	6 Detik
PDF	7348 KB	BC851CD019A9D817 F475A19C446ABE8B 0F6414B2BCC4B8EF F03C74EEDCA26116	0,8%	16,2 MB	0,1 Detik
DOCX	110722 KB	0890B930FB7CB4B9 D1CF10A99C514617 316637C0B0525A8A E00F0A8BEAD0B516	15,4%	16,4 MB	1 Detik
MP3	54180 KB	A10EA48CAD6F11D3 678DF4860D37C581 B8C821330029C564 EBD41E4F67635700	6,5%	17,3 MB	0,8 Detik
TXT	18KB	A75019AC38E0D357	0,2%	16,0 MB	0,05

Ekstensi	Ukuran	SHA-256			
		Nilai Hash	Prosesor	Memori	Waktu
		0633FA282F3D95D2 0763657F4A2FE851 FAE52A3185D1EDED			Detik

Tabel 17. Hasil Pengujian MD5

Ekstensi	Ukuran	MD5			
		Nilai Hash	Prosesor	Memori	Waktu
MP4	680164 KB	65FB849512720864 78EC854D04DD25A2 0803098776E9C011 D614EAEDC79AF934	23,2%	17,7%	2 Detik
PDF	7348 KB	BC851CD019A9D817 F475A19C446ABE8B 0F6414B2BCC4B8EF F03C74EEDCA26116	0,8%	17,8 MB	0,05 Detik
DOCX	110722 KB	0890B930FB7CB4B9 D1CF10A99C514617 316637C0B0525A8A E00F0A8BEAD0B516	4,3%	16,2 MB	0,5 Detik
MP3	54180 KB	A10EA48CAD6F11D3 678DF4860D37C581 B8C821330029C564 EBD41E4F67635700	4,2%	16,1 MB	0,3 Detik
TXT	18KB	A75019AC38E0D357 0633FA282F3D95D2 0763657F4A2FE851 FAE52A3185D1EDED	0,6%	15,4 MB	0,02 Detik

Berdasarkan hasil pengujian yang dilakukan pada tabel 4.1. dan 4.2 di atas membuktikan bahwa algoritma MD5 lebih unggul dibandingkan dengan algoritma SHA-256 dalam semua parameter yang digunakan untuk menguji performa dari kedua algoritma tersebut dalam membangkitkan identitas *file.* Hal ini dipengaruhi oleh panjang dari nilai *hash* yang di hasilkan, di mana MD5 menghasilkan nilai *hash* dengan panjang 128bit sedangkan SHA-256 memiliki nilai *hash* dengan panjang 256 bit.

4. SIMPULAN

Berdasarkan penelitian yang telah dilakukan maka kesimpulan yang dapat diambil adalah algoritma MD5 memiliki performa yang lebih unggul dibandingkan dengan algoritma SHA-256 dalam hal penggunaan prosesor, penggunaan memori dan waktu yang dibutuhkan untuk membangkitkan identitas file. Perbedaan ekstensi dari file tidak mempengaruhi performa dari algoritma MD5 maupun SHA-256, tetapi ukuran dari file yang mempengaruhi performa dari kedua algoritma tersebut. Kompleksitas algoritma dan panjang nilai hash mempengaruhi performa dari algoritma MD5 maupun algoritma SHA-256.

Untuk meningkatkan penelitian agar lebih berguna ke depannya maka saran yang dapat peneliti berikan adalah membandingkan performa algoritma dalam fungsi hash yang memiliki panjang nilai hash yang sama.

Membangun aplikasi yang dapat digunakan untuk menganalisa performa dari berbagai algoritma yang ada dalam fungsi hash.

DAFTAR PUSTAKA

[1] N. Hayati, M. A. Budiman dan A. Sharif, "Implementasi Algoritma RC4A dan MD5 untuk Menjamin Confidentiality dan Integrity pada File Teks," *Sinkron,* vol. I, no. 2, pp. 51-57, 2017.

[2] G. H. Editya dan S. Mulyati, "Aplikasi Mobile One Time Password Menggunakan Algoritma MD5 dan SHA-1 Untuk Meningkatkan Keamanan Website," *SNIKA,* vol. I, no. 2, pp. 618-623, 2018.

[3] A. Y. Mulyadi, E. P. Nugroho dan R. R. JP, "Implementasi Algoritma AES 128 dan SHA –256 Dalam Pengkodean pada Sebagian Frame Video CCTV MPEG-2," *JATIKOM,* vol. II, no. 1, pp. 33-39, 2018.

[4] I. Saputra dan S. D. Nasution, "Analisa Algoritma SHA-256 Untuk Mendeteksi Orisinalitas Citra Digital," dalam *Prosiding Seminar Nasional Riset Information Science (SENARIS)*, Pematang Siantar, 2019.

[5] R. Munir, Kriptografi, Bandung: Informatika Bandung, 2006.

[6] Y. Kurniawan, Kriptografi Keamanan Internet dan Jaringan Komunikasi, Bandung: Informatika, 2017.

[7] R. L. Rivest, "The MD5 Message-Digest Algorithm," Massachusetts Institute of Technology Laboratory for Computer Science, Cambridge, 1990.

[8] D. Rachamawati, J. T. Tarigan dan A. B. C. Ginting, "A Comparative Study of Message Digest 5 (MD5) and SHA-256 Algorithm," dalam *2nd International Conference on Computing and Applied Informatics 2017*, Medan, 2018.

[9] F. I. P. Standards Publications, "Secure Hash Standard," National Institute of Standards and Technology, Amerika Serikat, 2002.

[10] A. dan D. S. M. Ghosh, "Review Paper on Secure Hash Algorithm With Its Variants," *International Journal of Innovation in Modern Engineering & Science (IJTIMES),* vol. III, no. 05, pp. 1-7, 2017.

[11] T. Kirubakaran, K. Srinivasan, P. J. Vignesh dan V. V. A, "Implementation of Secure Hash Algorithm-256," *International Journal of Engineering Science and Computing (IJESC),* pp. 5785-5788, 2017.