Journal of Computer System and Informatics (JoSYC)
Volume 3, No. 1, November 2021, Page 16−25
ISSN 2714-8912 (media online)
ISSN 2714-7150 (media cetak)
DOI 10.47065/josyc.v3i1.822

Penerapan Fungsi Hash dengan Algoritma RIPEMD-128 Pada Aplikasi Duplicate PDF Scanner

Abdurrahman Al Habib

Fakultas Ilmu Komputer dan Teknologi Informasi, Prodi Teknik Informatika, Universitas Budi Darma, Medan, Indonesia
Email: abdurrahmanalhabib@gmail.com
Submitted: **27/07/2021**; Accepted: **13/11/2021**; Published: **30/11/2021**

Abstrak−Perkembangan era dunia digital terus tumbuh dan semakin meningkat. Banyak industri yang sudah beralih ke teknologi digital seperti foto, video, musik, dan sebagainya. Tentunya perubahan ini membawa banyak manfaat dalam hal kecepatan dan akses data. Kemudahan dalam mengakses internet juga berpengaruh besar terhadap kehidupan manusia. Kemudahan untuk mengakses dokumen secara online sangat membantu jika ingin mencari dan mendownload dokumen berbentuk pdf secara daring. Dengan kemudahan ini banyak file pdf yang sama lebih dari satu kali terdownload. Hal ini pasti akan membebani ruang penyimpanan. Untuk membedakan file pdf yang satu dengan yang lain dilakukan dengan cara melihat dan membaca isi dari file pdf tersebut. Hal ini sangat merepotkan dan memakan banyak waktu. Untuk mengatasi hal itu maka diperlukan suatu cara praktis yang dapat digunakan untuk mencari dokumen pdf yang sama sehingga bisa menghapus file pdf yang duplikat. Salah satu cara yang dapat digunakan untuk mengatasi masalah tersebut adalah dengan memanfaatkan fungsi hash. Fungsi hash di dalam kriptografi adalah merupakan alat yang sangat penting dalam berbagai aplikasi kriptografi, sebagai contoh dalam pembentukan tanda-tangan digital, otentikasi, dan sebagainya. Semenjak ditemukannya algoritma hash MD4 oleh Ron Rivest, telah banyak algoritma-algoritma hash yang lain yang telah dibentuk berdasarkan prinsip-prinsip dalam algoritma ini. Salah satunya adalah algoritma RIPEMD-128. RIPEMD-128 adalah fungsi hash berulang yang beroperasi pada kata-kata 32-bit. Dengan menggunakan fungsi hash maka dapat dibangkitkan identitas dari file pdf. Jika file pdf tersebut sama maka akan menghasilkan nilai hash yang sama pula. Hal ini akan memudahkan untuk mengelompokkan dokumen pdf yang sama sehingga akan mempercepat proses penghapusan file pdf yang sama. Dengan aplikasi pdf file scanner maka akan dicari file pdf ke semua folder dan akan mengelompokkan file pdf berdasarkan nilai hash yang sama. Sehingga akan memudahkan pengguna untuk menghapus file pdf yang sama.

Kata Kunci: Algoritma RIPEMD-128; Aplikasi PDF Scanner

Abstract−The development of the digital world era continues to grow and is increasing. Many industries have switched to digital technology such as photos, videos, music, and so on. Of course, this change brings many benefits in terms of speed and data access. Ease of accessing the internet also has a major impact on human life. The ease of accessing documents online is very helpful if you want to find and download pdf documents online. With this convenience, many of the same pdf files are downloaded more than once. This will definitely burden the storage space. To distinguish one pdf file from another, it is done by viewing and reading the contents of the pdf file. This is very inconvenient and time consuming. To overcome this, we need a practical way that can be used to search for the same pdf document so that it can delete duplicate pdf files. One way that can be used to overcome this problem is to use a hash function. The hash function in cryptography is a very important tool in various cryptographic applications, for example in digital signature formation, authentication, and so on. Since the discovery of the MD4 hash algorithm by Ron Rivest, many other hash algorithms have been developed based on the principles of this algorithm. One of them is the RIPEMD-128 algorithm. RIPEMD-128 is an iterative hash function that operates on 32-bit words. By using a hash function, the identity of the pdf file can be generated. If the pdf files are the same it will produce the same hash value as well. This will make it easier to group the same pdf documents so that it will speed up the process of deleting the same pdf files. With the pdf file scanner application, pdf files will be searched for all folders and will group pdf files based on the same hash value. So that it will be easier for users to delete the same pdf file.

Keywords: RIPEMD-128 Algorithm; PDF Scanner Application

1. PENDAHULUAN

Perkembangan jaringan internet yang semakin pesat membuat perubahan yang sangat signifikan di bidang jaringan. Begitu juga dengan referensi-referensi yang dalam bentuk buku kini sudah beralih menjadi bentuk digital. Banyak sekali buku-buku dalam bentuk digital yang biasa disebut dengan e-book disimpan menggunakan format pdf. Format pdf merupakan format standar untuk pengiriman dokumen serta di dukung oleh mayoritas sistem operasi sehingga file pdf ini bisa dibuka di berbagai sistem operasi dari berbagai platform.

Dalam ilmu kriptografi terdapat fungsi hash yang biasa digunakan untuk membangkitkan sidik jari atau fingerprint serta sering juga digunakan untuk membangkitkan tanda tangan digital (digital signature). Fungsi hash akan menghasilkan panjang nilai hash yang sama walaupun memiliki panjang input yang berbeda-beda. Hal inilah yang membuat fungsi hash di dalam kriptografi dapat digunakan untuk membangkitkan identitas dari sebuah file pdf. Dalam ilmu kriptografi banyak terdapat fungsi hash yang sering digunakan mulai dari fungsi hash yang memiliki tingkat kompleksitas yang rendah sampai fungsi hash yang memiliki tingkat kompleksitas yang tinggi.

Dalam fungsi hash terdapat dua varian dari algoritma RIPEMD yaitu dengan panjang nilai hash 128 bit dan 160 bit. Pemilihan algoritma RIPEMD-128 dikarenakan algoritma RIPEMD-128 memiliki performa yang lebih cepat jika dibandingkan dengan RIPEMD-160. Hal ini tentunya sangat berpengaruh mengingat jumlah file pdf yang akan di bangkitkan identitasnya cukup banyak, sehingga waktu proses sangat berperan penting dalam hal ini.

Journal of Computer System and Informatics (JoSYC)
Volume 3, No. 1, November 2021, Page 16−25
ISSN 2714-8912 (media online)
ISSN 2714-7150 (media cetak)
DOI 10.47065/josyc.v3i1.822

Dengan dukungan dari file pdf yang sudah menjadi standar pengiriman dokumen banyak sekali file-file yang berupa referensi-referensi diunggah ke internet dalam bentuk pdf, sehingga dapat dengan mudah file pdf tersebut diakses oleh orang yang membutuhkan.

Masalah tidak jarang juga file pdf yang sama di-download lebih dari satu kali dan tersimpan di sebuah media penyimpanan. Hal ini tentunya kurang efisien karena ruang penyimpanan terpakai oleh dua file pdf yang sama. Kesulitan yang dihadapi adalah ketika file pdf yang sama tersebut disimpan dengan nama yang berbeda, tentunya user harus membuka dan membaca isi dari file pdf tersebut dan membandingkan isinya dengan isi dari file pdf yang lain. Hal ini tentunya sangat menyulitkan user untuk mengetahui apakah file pdf tersebut duplikat atau tidak.

Untuk mengatasi masalah tersebut salah satu solusi yang dapat digunakan adalah dengan memberikan identitas bagi setiap file pdf, di mana jika identitas dari file pdf itu sama menandakan bahwa isi dari file pdf tersebut adalah sama juga, dalam kata lain file pdf tersebut duplikat. Untuk dapat menghasilkan identitas dari sebuah file pdf dibutuhkan cara yang dapat mengakomodasi bahwa tidak akan pernah terjadi dua buah file pdf yang berbeda tetapi memiliki identitas yang sama. Hal ini dikarenakan identitas harus bersifat unik dan tidak boleh sama dengan yang lainnya jika file pdf tersebut berbeda.

Pada penelitian sebelumnya dilakukan oleh Frank Landelle dan Thomas Peyrin dari Nanyang Technonogical University, Singapura yang di publikasikan dalam seminar EUROCRYPT tahun 2013 dengan judul Cryptanalysis of Full RIPEMD-128 menyimpulkan bahwa serangan Collision di dilakukan pada proses kompresi dari algoritma RIPEMD-128 memungkinkan algoritma RIPEMD-128 ini memiliki nilai hash yang sama dengan input yang berbeda, tetapi disisi lain algoritma RIPEMD-128 memiliki performa yang lebih cepat jika dibandingkan dengan RIPEMD-160 [1].

2. METODOLOGI PENELITIAN

2.1 Kriptografi

Perubahan besar yang mempengaruhi keamanan adalah pengenalan sistem terdistribusi dan penggunaan jaringan dan fasilitas komunikasi untuk membawa data antara pengguna terminal dan komputer dan antara komputer dan komputer. Tindakan keamanan jaringan diperlukan untuk melindungi data selama transmisi. Tindakan tersebut dinamakan kriptografi [1][2].

2.2. Fungsi Hash

Fungsi *Hash* merupakan sebuah algoritma yang mengubah *text* atau *message* menjadi sederetan karakter acak yang memiliki jumlah karakter yang sama. *Hash* juga termasuk salah satu bentuk teknik kriptografi dan dikategorikan sebagai kriptografi tanpa *key (unkeyed cryptosystem)*. Selain itu hash memiliki nama lain yang juga dikenal luas yaitu *"one-way function"* [3].

2.3. RIPEMD-128

Algoritma RIPEMD-128 *(RACE Integrity Primitives Evaluation Massage Di gest-128)*, dirancang oleh Bart Preneel, Antoon Bosselaers dan Hans Dobbertin pada tahun 1996, merupakan fungsi *hash* kriptografi yang cepat yang dibuat untuk diimplementasikan pada *software* yang dijalankan pada mesin berarsitektur 32-bit. RIPEMD-128 adalah salah satu contoh algoritma *hash* yang sering juga disebut dengan nama fungsi pembanding, fungsi penyusutan, intisari pesan, sidik jari, *message integrity check* (MIC) atau pemeriksa keutuhan pesan dan *manipulation detection code* (MDC) atau pendeteksi penyelenggaraan kode [4].

Fungsi *hash* atau arah dibuat berdasarkan ide tentang fungsi penempatan. Fungsi *hash* adalah sebuah fungsi atau persamaan matematika yang mengambil *input* dengan panjang variabel *(preimage)* dan mengubahnya menjadi panjang yang tetap (biasanya lebih pendek), keluarannya biasa disebut nilai *hash*. Fungsi *hash* satu arah adalah sebuah fungsi *hash* yang berjalan hanya satu arah. apabila ingin melindungi data dari modifikasi yang tidak terdeteksi, dapat di hitung hasil fungsi *hash* dari data tersebut, selanjutnya dapat menghitung hasil fungsi *hash* lagi dan membandingkan dengan hasil yang pertama apabila berbeda maka terjadi perubahan selama pengiriman.

RIPEMD-128 adalah suatu fungsi *hash* kriptografi yang dirancang untuk implementasi pada perangkat lunak dengan arsitektur 32 bit. Desain utama dari fungsi *hash* ini ada dua proses komputasional yang berbeda dan saling idenpenden, dimana hasil dari kedua proses ini akan digabungkan pada akhir perhitungan dengan sebuah fungsi komprosi. Sesuai dengan namamya RIPEMD-128 menghasilkan 128 bit. Hal ini dimaksudkan untuk menyediakan level sekuritas yang lebih tinggi untuk 10 tahun yang akan datang. Sama halnya dengan varian MD4 lainnya, RIPEMD-128 beroperasi pada fungsi ini adalah sebagai berikut [5]:

1. Rotasi kiri *(left-rotation atau left-spin)* dari pesan masukan;
2. Operasi *bitwise Boolean* (AND,NOT,OR,exclusive-OR);
3. Penambahan dua buah *string* sepanjang *module* 2 pada nilai *hash.*

Dibawah ini adalah deskripsi langkah-langkah proses perhitungan nilai *hash* pada algoritma RIPEMD-128:

1. Definisi fungsi, permutasi dan *inisialisasi buffer*

Journal of Computer System and Informatics (JoSYC)
Volume 3, No. 1, November 2021, Page 16−25
ISSN 2714-8912 (media online)
ISSN 2714-7150 (media cetak)
DOI 10.47065/josyc.v3i1.822

Sebelum proses perhitungan, terlebih dahulu didefinisikan fungsi-fungsi dan permutasi yang akan digunakan pada proses kompresi. Definisi kelima fungsi tersebut adalah:

a. F(x,y,z) = [(x) Xor (y) Xor (z) (1)
b. G(x,y,z)= [(x) And (y) or [(not(x)) And (z)] (2)
c. H(x,y,z)= [(x) or (not(y))] Xor (z))] (3)
d. I(x,y,z)= [[(x) And(z)] or [(y) And (not(z))]] (4)
e. J(x,y,z)= [(x) Xor [(y) or (not(z))]] (5)

Dimana urutan penggunaan fungsi-fungsi tersebut pada masing-masing rantai seperti terlihat pada

Tabel 1. Urutan Fungsi *Round* Pada Fungsi Kompresi Algoritma RIPEMD-128.

Line	***Round***				
	1	**2**	**3**	**4**	**5**
Left	F1	F2	F3	F4	F5
Right	F1	F2	F3	F4	F5

Sedangkan permutasi yang digunakan untuk menentukan urutan word seperti terlihat pada tabel 2.

Tabel 2. Urutan pengacakan *word*

Line	***Round***				
	1	**2**	**3**	**4**	**5**
Left	Id	P	p^2	p^3	p^4
Right	π	Pπ	$p^2\pi$	$p^3\pi$	$p^4\pi$

Dimana =[(9*Id) + 5] mod 16. Kemudian disiapkan juga suatu vektor awal, yaitu *buffer* (H_0,H_1,H_2,H_3,H_4), yang masing-masing nilainya dalam notasi heksa-desimal [6].

H_0 : 67452301_X;
H_1 : $EFCDAB89_X$;
H_2 : $98BADCFE_X$;
H_3 : 10325476_X ;
H_4 : $C3D2EIFO_X$;

3. HASIL DAN PEMBAHASAN

Permasalahan yang dibahas pada penelitian ini terkait dengan pencarian *file pdf* pada suatu media penyimpanan. *File pdf* merupakan *portable document format*, karena bersifat portabel maka *file* ini banyak di dukung oleh berbagai macam aplikasi untuk menampilkannya. Dengan kemudahan ini maka banyak sekali dokumen di buat dalam bentuk *pdf* sehingga mudah untuk di bagikan. Akan tetapi akan sangat merugikan jika terdapat beberapa *file pdf* yang sama atau duplikat dalam sebuah media penyimpanan.

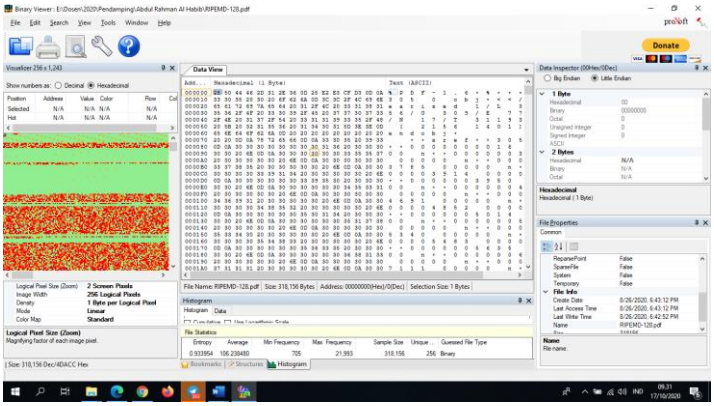
Untuk dapat membedakan *file pdf* yang satu dengan yang lainnya maka pengguna harus melihat isi dari *file pdf* tersebut, sehingga jika terdapat *file pdf* yang sama pengguna dapat menghapusnya dan menyisakan hanya satu *file* saja. Tetapi hal ini menjadi tidak mungkin untuk dilakukan jika *file pdf* tersebut terdiri dari banyak halaman dan di media penyimpanan tersebut terdiri dari banyak *file pdf.* Untuk itu dibutuhkan identitas dari setiap *file pdf*, di mana *file pdf* tersebut merupakan hasil representasi dari isi *file pdf.* Sehingga jika terdapat lebih dari satu *file pdf* yang memiliki nilai identitas yang sama, maka dapat dipastikan bahwa *file pdf* tersebut adalah ganda atau duplikat, dan pengguna dapat menghapus *file* tersebut sehingga hanya menyisakan satu *file pdf* saja.

Pada penelitian ini algoritma yang digunakan untuk membangkitkan identitas dari *file pdf* adalah algoritma RIPEMD-128 yang di terapkan pada aplikasi *duplicate pdf scanner*. Langkah pertama ketika akan melakukan pencarian *file pdf* yang ganda atau duplikat adalah dengan memilih lokasi penyimpanan yang akan di lakukan pencarian *file pdf*, setelah *file pdf* di temukan maka akan di bangkitkan identitas dari setiap *file pdt* tersebut berupa nilai *hash* yang digunakan untuk mengurutkan *file pdf* tersebut, sehingga *file pdf* yang memiliki identitas yang sama akan diletakkan berdekatan sehingga memudahkan pengguna untuk menghapus *file pdf* yang ganda atau duplikat.

3.1 Penerapan Algoritma RIPEMD-128

Untuk menjelaskan langkah penerapan algoritma RIPEMD-128 pada aplikasi *duplicate pdf scanner* dengan melakukan perhitungan secara manual. Sebagai contoh pada penelitian ini menggunakan *file* dengan nama RIPEMD-128 berekstensi *pdf.* Ukuran dari *file pdf* yang digunakan pada contoh kasus adalah 311 KB, namun untuk menyederhanakan perhitungan maka *file pdf* tersebut tidak di hitung secara keseluruhan, tetapi hanya di ambil sampel dari *file pdf* tersebut. Untuk melihat nilai *file pdf* tersebut dalam bentuk heksadesimal maka pada penelitian ini menggunakan alat bantu aplikasi binary viewer.

Journal of Computer System and Informatics (JoSYC)
Volume 3, No. 1, November 2021, Page 16−25
ISSN 2714-8912 (media online)
ISSN 2714-7150 (media cetak)
DOI 10.47065/josyc.v3i1.822



Gambar 1. Nilai Heksadesimal *File Pdf*

Pada gambar 1 di atas dapat dilihat nilai heksadesimal dari *file pdf* yang digunakan sebagai sampel. Sampel yang digunakan dalam contoh kasus ini diambil pada *address* 0000 sebanyak 12 bytes.

Tabel 3. Nilai Heksadesimal Sampel

M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11
25	50	44	46	2D	31	2E	36	0D	25	E2	E3

1. Penambahan *Padding Bit*
   Untuk melakukan penambahan *Padding Bit* maka pesan di ubah ke dalam bentuk biner. Panjang pesan yang di gunakan sebagai sampel adalah 12 bytes sehingga jika di ubah menjadi biner jumlah pesan ada 96 bit. Maka di perlukan penambahan *Padding Bit* dengan ketentuan sebagai berikut:
   L+1+M= 448 Mod 512
   L+1+96 = 448 Mod 512
   L = (448-97) Mod 512
   L = 351
   Maka diperlukan penambahan bit 1 kemudian di ikuti oleh bit 0 sebanyak 351 bit.

Tabel 4. Penambahan *Padding Bit*

00100101	01010000	01000100	01000110	00101101	00110001	00101110	00110110
00001101	00100101	11100010	11100011	**10000000**	**00000000**	**00000000**	**00000000**
00000000	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**
00000000	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**
00000000	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**
00000000	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**
00000000	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**

2. Penambahan Panjang Pesan
   Langkah kedua, adalah proses penambahan 64 bit nilai yang menyatakan panjang pesan semula. Jika panjang pesan lebih besar dari 2^{64} maka yang diambil adalah panjangnya dengan modulo 2^{64}. Dengan kata lain, jika panjang pesan semula adalah K bit, maka 64 bit yang ditambahkan menyatakan K modulo 2^{32}, proses ini dapat dinamakan *low-order word*. Setelah ditambah dengan 64 bit, panjang pesan menjadi 512 bit. Panjang data yang digunakan adalah 96 bit sehingga jika di ubah ke dalam bentuk biner maka hasilnya adalah 01100000. Untuk lebih jelasnya dapat di lihat pada tabel 5. berikut ini.

Tabel 5. Penambahan Panjang Pesan

00100101	01010000	01000100	01000110	00101101	00110001	00101110	00110110
00001101	00100101	11100010	11100011	10000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**	**00000000**	**01100000**

Journal of Computer System and Informatics (JoSYC)
Volume 3, No. 1, November 2021, Page 16−25
ISSN 2714-8912 (media online)
ISSN 2714-7150 (media cetak)
DOI 10.47065/josyc.v3i1.822

3. Inisialisasi
Variabel *input* sepanjang 128 bit di bagi menjadi 4 *word* masing-masing berukuran 32 bit yang digunakan untuk menginisialisasi cabang kiri dan cabang kanan.

Tabel 6. Inisialisasi Cabang Kiri dan Cabang Kanan

Kiri	X_{-3} = 67452301	X_{-2} = EFCDAB89	X_{-1} = 98BADCFE	X_0 = 10325476
Kanan	Y_{-3} = 67452301	Y_{-2} = EFCDAB89	Y_{-1} = 98BADCFE	Y_0 = 10325476

4. Ekspansi Pesan
Blok pesan masukkan 512 bit di bagi menjadi 16 *word* dan masing-masing berukuran 32 bit dapat dilihat pada tabel 4 Setiap *word* M_i akan digunakan sekali dalam setiap putaran dalam urutan yang ditentukan untuk setiap cabang. Pada tabel 5 dan 6 berikut ini adalah tabel permutasi *word* untuk ekspansi pesan.

Tabel 7. Pesan Dalam 16 *Word*

M0	=	00100101	01010000	01000100	01000110	=	25504446
M1	=	00101101	00110001	00101110	00110110	=	2D312E36
M2	=	00001101	00100101	11100010	11100011	=	0D25E2E3
M3	=	10000000	00000000	00000000	00000000	=	80000000
M4	=	00000000	00000000	00000000	00000000	=	00000000
M5	=	00000000	00000000	00000000	00000000	=	00000000
M6	=	00000000	00000000	00000000	00000000	=	00000000
M7	=	00000000	00000000	00000000	00000000	=	00000000
M8	=	00000000	00000000	00000000	00000000	=	00000000
M9	=	00000000	00000000	00000000	00000000	=	00000000
M10	=	00000000	00000000	00000000	00000000	=	00000000
M11	=	00000000	00000000	00000000	00000000	=	00000000
M12	=	00000000	00000000	00000000	00000000	=	00000000
M13	=	00000000	00000000	00000000	00000000	=	00000000
M14	=	00000000	00000000	00000000	00000000	=	00000000
M15	=	00000000	00000000	00000000	01100000	=	00000060

Tabel 8. Permutasi *Word* Ekspansi Pesan Cabang Kiri

Round J	$\pi_j^l(k)$															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8
2	3	10	14	4	9	15	8	1	2	7	0	6	13	11	5	12
3	1	9	11	10	0	8	12	4	13	3	7	15	14	5	6	2

Tabel 9. Permutasi *Word* Ekspansi Pesan Cabang Kanan

Round J	$\pi_j^r(k)$															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3	12
1	6	11	3	7	0	13	5	10	14	15	8	12	4	9	1	2
2	15	5	1	3	7	14	6	9	11	8	12	2	10	0	4	13
3	8	6	4	1	3	11	15	0	5	12	2	13	9	7	10	14

Untuk melakukan ekspansi pesan maka di kerjakan dengan menggunakan persamaan:
$W_j^l 16 + k = M\pi_j^l(k)$ dan $W_j^r 16 + k = M\pi_j^r(k)$
Di mana $0 \leq j \leq 3$ dan $0 \leq k \leq 15$
Ekspansi pesan dapat dilihat pada tabel berikut ini:

Tabel 10. Ekspansi Pesan

i	**j**	**k**	$\boldsymbol{W_j^l 16 + k}$	$\boldsymbol{M\pi_j^l(k)}$	$\boldsymbol{W_j^r 16 + k}$	$\boldsymbol{M\pi_j^r(k)}$
0	0	0	$W_0^l 16$	25504446	$W_0^r 16$	00000000
1	0	1	$W_0^l 17$	2D312E36	$W_0^r 17$	00000000
2	0	2	$W_0^l 18$	0D25E2E3	$W_0^r 18$	00000000
3	0	3	$W_0^l 19$	80000000	$W_0^l 19$	25504446

Journal of Computer System and Informatics (JoSYC)
Volume 3, No. 1, November 2021, Page 16−25
ISSN 2714-8912 (media online)
ISSN 2714-7150 (media cetak)
DOI 10.47065/josyc.v3i1.822

i	j	k	$W_j^l 16 + k$	$M\pi_j^l(k)$	$W_j^r 16 + k$	$M\pi_j^r(k)$
4	0	4	$W_0^l 20$	00000000	$W_0^l 20$	00000000
5	0	5	$W_0^l 21$	00000000	$W_0^l 21$	0D25E2E3
6	0	6	$W_0^l 22$	00000000	$W_0^l 22$	00000000
7	0	7	$W_0^l 23$	00000000	$W_0^l 23$	00000000
8	0	8	$W_0^l 24$	00000000	$W_0^l 24$	00000000
9	0	9	$W_0^l 25$	00000000	$W_0^l 25$	00000000
10	0	10	$W_0^l 26$	00000000	$W_0^l 26$	00000060
11	0	11	$W_0^l 27$	00000000	$W_0^l 27$	00000000
12	0	12	$W_0^l 28$	00000000	$W_0^l 28$	2D312E36
13	0	13	$W_0^l 29$	00000000	$W_0^l 29$	00000000
14	0	14	$W_0^l 30$	00000000	$W_0^l 30$	80000000
15	0	15	$W_0^l 31$	00000060	$W_0^l 31$	00000000
16	1	0	$W_1^l 16$	00000000	$W_1^l 16$	00000000
17	1	1	$W_1^l 17$	00000000	$W_1^l 17$	00000000
18	1	2	$W_1^l 18$	00000000	$W_1^l 18$	80000000
19	1	3	$W_1^l 19$	2D312E36	$W_1^l 19$	00000000
20	1	4	$W_1^l 20$	00000000	$W_1^l 20$	25504446
21	1	5	$W_1^l 21$	00000000	$W_1^l 21$	00000000
22	1	6	$W_1^l 22$	00000060	$W_1^l 22$	00000000
23	1	7	$W_1^l 23$	80000000	$W_1^l 23$	00000000
24	1	8	$W_1^l 24$	00000000	$W_1^l 24$	00000000
25	1	9	$W_1^l 25$	25504446	$W_1^l 25$	00000060
26	1	10	$W_1^l 26$	00000000	$W_1^l 26$	00000000
27	1	11	$W_1^l 27$	00000000	$W_1^l 27$	00000000
28	1	12	$W_1^l 28$	0D25E2E3	$W_1^l 28$	00000000
29	1	13	$W_1^l 29$	00000000	$W_1^l 29$	00000000
30	1	14	$W_1^l 30$	00000000	$W_1^l 30$	2D312E36
31	1	15	$W_1^l 31$	00000000	$W_1^l 31$	0D25E2E3
32	2	0	$W_2^l 16$	80000000	$W_2^l 16$	00000060
33	2	1	$W_2^l 17$	00000000	$W_2^l 17$	00000000
34	2	2	$W_2^l 18$	00000000	$W_2^l 18$	2D312E36
35	2	3	$W_2^l 19$	00000000	$W_2^l 19$	80000000
36	2	4	$W_2^l 20$	00000000	$W_2^l 20$	00000000
37	2	5	$W_2^l 21$	00000060	$W_2^l 21$	00000000
38	2	6	$W_2^l 22$	00000000	$W_2^l 22$	00000000
39	2	7	$W_2^l 23$	2D312E36	$W_2^l 23$	00000000
40	2	8	$W_2^l 24$	0D25E2E3	$W_2^l 24$	00000000
41	2	9	$W_2^l 25$	00000000	$W_2^l 25$	00000000
42	2	10	$W_2^l 26$	25504446	$W_2^l 26$	00000000
43	2	11	$W_2^l 27$	00000000	$W_2^l 27$	0D25E2E3
44	2	12	$W_2^l 28$	00000000	$W_2^l 28$	00000000
45	2	13	$W_2^l 29$	00000000	$W_2^l 29$	25504446
46	2	14	$W_2^l 30$	00000000	$W_2^l 30$	00000000
47	2	15	$W_2^l 31$	00000000	$W_2^l 31$	00000000
48	3	0	$W_3^l 16$	2D312E36	$W_3^l 16$	00000000
49	3	1	$W_3^l 17$	00000000	$W_3^l 17$	00000000
50	3	2	$W_3^l 18$	00000000	$W_3^l 18$	00000000
51	3	3	$W_3^l 19$	00000000	$W_3^l 19$	2D312E36
52	3	4	$W_3^l 20$	25504446	$W_3^l 20$	80000000
53	3	5	$W_3^l 21$	00000000	$W_3^l 21$	00000000
54	3	6	$W_3^l 22$	00000000	$W_3^l 22$	00000060
55	3	7	$W_3^l 23$	00000000	$W_3^l 23$	25504446
56	3	8	$W_3^l 24$	00000000	$W_3^l 24$	00000000

Journal of Computer System and Informatics (JoSYC)
Volume 3, No. 1, November 2021, Page 16−25
ISSN 2714-8912 (media online)
ISSN 2714-7150 (media cetak)
DOI 10.47065/josyc.v3i1.822

i	j	k	$W_j^l 16 + k$	$M\pi_j^l(k)$	$W_j^r 16 + k$	$M\pi_j^r(k)$
57	3	9	$W_3^l 25$	80000000	$W_3^l 25$	00000000
58	3	10	$W_3^l 26$	00000000	$W_3^l 26$	0D25E2E3
59	3	11	$W_3^l 27$	00000060	$W_3^l 27$	00000000
60	3	12	$W_3^l 28$	00000000	$W_3^l 28$	00000000
61	3	13	$W_3^l 29$	00000000	$W_3^l 29$	00000000
62	3	14	$W_3^l 30$	00000000	$W_3^l 20$	00000000
63	3	15	$W_3^l 31$	0D25E2E3	$W_3^l 31$	00000000

5. *Step Function*

Di setiap langkah ke i, register X_{i+1} dan Y_{i+1} di perbaharui menggunakan fungsi F_j^l dan F_j^r dan tergantung pada ronde ke j dan langkah ke i seperti di bawah ini:

$X_{i+1} = (X_{i-3} + \theta_j^l(X_i, X_{i-1}, X_{i-2}) + W_i^l + K_j^l) <<< s_i^l$

$Y_{i+1} = (Y_{i-3} + \theta_j^r(Y_i, Y_{i-1}, Y_{i-2}) + W_i^r + K_j^r) <<< s_i^r$

Dimana K_j^l dan K_j^rkonstanta 32 bit yang di definisikan setiap ronde j dan setiap cabang, s_i^l dan s_i^rkonstanta rotasi yang di definisikan setiap langkah ke i dan setiap cabang, θ_j^l dan θ_j^r adalah 32 bit fungsi boolean yang di definisikan setiap ronde j dan setiap cabang. Untuk lebih jelasnya dapat di lihat pada tabel di bawah ini.

Tabel 11. Konstanta RIPEMD-128

Round J	$\theta_j^l(X_i, X_{i-1}, X_{i-2})$	$\theta_j^r(Y_i, Y_{i-1}, Y_{i-2})$	K_j^l	K_j^r
0	$X_i \oplus X`_{i-1} \oplus X_{i-2}$	$Y_i \wedge Y_{i-1} \oplus \sim Y_i \wedge Y_{i-2}$	00000000	50A28BE6
1	$X_i \wedge X_{i-1} \oplus \sim X_i \wedge X_{i-2}$	$(Y_i \vee \sim Y_{i-1}) \oplus Y_{i-2}$	5A827999	5C4DD124
2	$(X_i \vee \sim X_{i-1}) \oplus X_{i-2}$	$Y_i \wedge Y_{i-1} \oplus \sim Y_i \wedge Y_{i-2}$	6ED9EBA1	6D703EF3
3	$X_i \wedge X_{i-1} \oplus \sim X_i \wedge X_{i-2}$	$Y_i \oplus Y_{i-1} \oplus Y_{i-2}$	8F1BBCDC	00000000

Tabel 12. Konstanta Rotasi Kiri Cabang Kiri RIPEMD-128

Round J	$s_j^l i$															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	11	14	15	12	5	8	7	9	11	13	14	15	6	7	8	9
1	7	6	8	13	11	9	7	15	7	12	15	9	11	7	13	12
2	11	13	6	7	14	9	13	15	14	8	13	6	5	12	7	5
3	11	12	14	15	14	15	9	8	9	14	5	6	8	6	5	12

Tabel 13. Konstanta Rotasi Kiri Cabang Kanan RIPEMD-128

Round J	$s_j^r i$															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	8	9	9	11	13	15	15	5	7	7	8	11	14	14	12	6
1	9	13	15	7	12	8	9	11	7	7	12	7	6	15	13	11
2	9	7	15	11	8	6	6	14	12	13	5	14	13	13	17	5
3	15	5	8	11	14	14	6	14	6	9	12	9	12	5	15	8

Step 0 : i=0, j=0

$X_{i+1} = (X_{i-3} + \theta_j^l(X_i, X_{i-1}, X_{i-2}) + W_i^l + K_j^l) <<< s_i^l$

$X_{0+1} = (X_{0-3} + \theta_0^l(X_0, X_{0-1}, X_{0-2}) + W_0^l + K_0^l) <<< s_0^l$

$X_1 = (X_{-3} + \theta_0^l(X_0, X_{-1}, X_{-2}) + W_0^l + K_0^l) <<< s_0^l$

$X_1 = (67452301 + \theta_0^l(X_0, X_{-1}, X_{-2}) + 25504446 + 00000000) <<< 11$

$\theta_0^l(X_0, X_{-1}, X_{-2}) = X_i \oplus X_{i-1} \oplus X_{i-2}$

$\theta_0^l(X_0, X_{-1}, X_{-2}) = X_0 \oplus X_{0-1} \oplus X_{0-2}$

$\theta_0^l(X_0, X_{-1}, X_{-2}) = 10325476 \oplus 98BADCFE \oplus EFCDAB89$

$\theta_0^l(X_0, X_{-1}, X_{-2}) = 67452301$

$X_1 = (67452301 + 67452301 + 25504446 + 00000000) <<< 11$

$X_1 = (F3DA8A48) <<< 11$

$X_1 = D452479E$

$Y_{i+1} = (Y_{i-3} + \theta_j^r(Y_i, Y_{i-1}, Y_{i-2}) + W_i^r + K_j^r) <<< s_i^r$

$Y_{0+1} = (Y_{0-3} + \theta_0^r(Y_0, Y_{0-1}, Y_{0-2}) + W_0^r + K_0^r) <<< s_0^r$

$Y_1 = (Y_{-3} + \theta_0^r(Y_0, Y_{-1}, Y_{-2}) + W_0^r + K_0^r) <<< s_0^r$

Journal of Computer System and Informatics (JoSYC)
Volume 3, No. 1, November 2021, Page 16−25
ISSN 2714-8912 (media online)
ISSN 2714-7150 (media cetak)
DOI 10.47065/josyc.v3i1.822

$Y_1 = (67452301 + \theta_0^r(Y_0, Y_{-1}, Y_{-2}) + 00000000 + 50A28BE6) <<< 8$
$\theta_j^r(Y_0, Y_{0-1}, Y_{0-2}) = Y_0 \wedge Y_{0-1} \oplus \sim Y_0 \wedge Y_{0-2}$
$\theta_j^r(Y_0, Y_{-1}, Y_{-2}) = Y_0 \wedge Y_{-1} \oplus \sim Y_0 \wedge Y_{-2}$
$\theta_j^r(Y_0, Y_{-1}, Y_{-2}) = 10325476 \wedge 98BADCFE \oplus \sim 10325476 \wedge EFCDAB89$
$\theta_j^r(Y_0, Y_{-1}, Y_{-2}) = 10325476 \oplus EFCDAB89$
$\theta_j^r(Y_0, Y_{-1}, Y_{-2}) = FFFFFFFF$
$Y_1 = (67452301 + FFFFFFFF + 00000000 + 50A28BE6) <<< 8$
$Y_1 = (B7E7AEE6) <<< 8$
$Y_1 = E7AEE6B7$

Step 1 : i=1, j=0
$X_{i+1} = (X_{i-3} + \theta_j^l(X_i, X_{i-1}, X_{i-2}) + W_i^l + K_j^l) <<< s_i^l$
$X_{1+1} = (X_{1-3} + \theta_0^l(X_1, X_{1-1}, X_{1-2}) + W_1^l + K_0^l) <<< s_1^l$
$X_2 = (X_{-2} + \theta_0^l(X_1, X_0, X_{-1}) + W_1^l + K_0^l) <<< s_1^l$
$X_2 = (EFCDAB89 + \theta_0^l(X_1, X_0, X_{-1}) + 2D312E36 + 00000000) <<< 14$
$\theta_0^l(X_1, X_0, X_{-1}) = X_i \oplus X`_{i-1} \oplus X_{i-2}$
$\theta_0^l(X_1, X_0, X_{-1}) = X_1 \oplus X_{1-1} \oplus X_{1-2}$
$\theta_0^l(X_1, X_0, X_{-1}) =$ D452479E $\oplus$ 10325476 $\oplus$ 98BADCFE
$\theta_0^l(X_1, X_0, X_{-1}) =$ 5CDACF16
$X_2 = (EFCDAB89 + 5CDACF16 + 2D312E36 + 00000000) <<<< 14$
$X_2 =$ CACECC4A $<<<< 14$
$X_2 = B312B2B3$

$Y_{i+1} = (Y_{i-3} + \theta_j^r(Y_i, Y_{i-1}, Y_{i-2}) + W_i^r + K_j^r) <<< s_i^r$
$Y_{1+1} = (Y_{1-3} + \theta_0^r(Y_1, Y_{1-1}, Y_{1-2}) + W_1^r + K_0^r) <<< s_1^r$
$Y_2 = (Y_{-2} + \theta_0^r(Y_1, Y_0, Y_{-1}) + W_1^r + K_0^r) <<< s_1^r$
$Y_2 = (EFCDAB89 + \theta_0^r(Y_1, Y_0, Y_{-1}) + 00000000 + 50A28BE6) <<< 9$
$\theta_0^r(Y_1, Y_0, Y_{-1}) = Y_1 \wedge Y_{0-0} \oplus \sim Y_1 \wedge Y_{0-1}$
$\theta_0^r(Y_1, Y_0, Y_{-1}) = Y_1 \wedge Y_0 \oplus \sim Y_1 \wedge Y_{-1}$
$\theta_0^r(Y_1, Y_0, Y_{-1}) = E7AEE6B7 \wedge 10325476 \oplus \sim E7AEE6B7 \wedge 98BADCFE$
$\theta_0^r(Y_1, Y_0, Y_{-1}) = 224436 \oplus 18101848$
$\theta_j^r(Y_0, Y_{-1}, Y_{-2}) = 18325C7E$
$Y_2 = (EFCDAB89 + 18325C7E + 00000000 + 50A28BE6) <<< 9$
$Y_2 = (58A293ED) <<< 9$
$Y_2 = 4527DAB1$

Step 2 : i=2, j=0
$X_{i+1} = (X_{i-3} + \theta_j^l(X_i, X_{i-1}, X_{i-2}) + W_i^l + K_j^l) <<< s_i^l$
$X_{2+1} = (X_{2-3} + \theta_0^l(X_2, X_{2-1}, X_{2-2}) + W_2^l + K_0^l) <<< s_2^l$
$X_3 = (X_{-1} + \theta_0^l(X_2, X_1, X_0) + W_2^l + K_0^l) <<< s_2^l$
$X_3 = (98BADCFE + \theta_0^l(X_2, X_1, X_0) + 0D25E2E3 + 00000000) <<< 15$
$\theta_0^l(X_2, X_1, X_0) = X_i \oplus X`_{i-1} \oplus X_{i-2}$
$\theta_0^l(X_2, X_1, X_0) = X_2 \oplus X_{2-1} \oplus X_{2-2}$
$\theta_0^l(X_2, X_1, X_0) =$ B312B2B3 $\oplus$ D452479E $\oplus$ 10325476
$\theta_0^l(X_2, X_1, X_0) =$ 7778AA51
$X_3 = (98BADCFE + 5CDACF16 + 0D25E2E3 + 00000000) <<<< 15$
$X_3 =$ 22C6DA4A $<<<< 15$
$X_3 = 6D251163$

$Y_{i+1} = (Y_{i-3} + \theta_j^r(Y_i, Y_{i-1}, Y_{i-2}) + W_i^r + K_j^r) <<< s_i^r$
$Y_{2+1} = (Y_{2-3} + \theta_0^r(Y_2, Y_{2-1}, Y_{2-2}) + W_2^r + K_0^r) <<< s_2^r$
$Y_3 = (Y_{-1} + \theta_0^r(Y_2, Y_1, Y_0) + W_2^r + K_0^r) <<< s_2^r$
$Y_3 = (98BADCFE + \theta_0^r(Y_2, Y_1, Y_0) + 0D25E2E3 + 50A28BE6) <<< 9$
$\theta_0^r(Y_2, Y_1, Y_0) = Y_2 \wedge Y_{1-0} \oplus \sim Y_2 \wedge Y_{0-0}$
$\theta_0^r(Y_2, Y_1, Y_0) = Y_2 \wedge Y_1 \oplus \sim Y_2 \wedge Y_0$
$\theta_0^r(Y_2, Y_1, Y_0) =$ 4527DAB1$\wedge$ E7AEE6B7 $\oplus \sim$ 4527DAB1 $\wedge$ 10325476
$\theta_0^r(Y_2, Y_1, Y_0) = 4526C2B1 \oplus 10100446$
$\theta_0^r(Y_2, Y_1, Y_0) =$ 5536C6F7
$Y_3 = (98BADCFE + 5536C6F7 + 0D25E2E3 + 50A28BE6) <<< 9$

Journal of Computer System and Informatics (JoSYC)
Volume 3, No. 1, November 2021, Page 16−25
ISSN 2714-8912 (media online)
ISSN 2714-7150 (media cetak)
DOI 10.47065/josyc.v3i1.822

Y_3 = (3E942FDB) <<< 9
Y_3 = 285FB67D

Step 3 : i=3, j=0
$X_{i+1} = (X_{i-3} + \theta_j^l(X_i, X_{i-1}, X_{i-2}) + W_i^l + K_j^l) <<< s_i^l$
$X_{3+1} = (X_{3-3} + \theta_0^l(X_3, X_{3-1}, X_{3-2}) + W_3^l + K_0^l) <<< s_3^l$
$X_4 = (X_0 + \theta_0^l(X_1, X_2, X_1) + W_3^l + K_0^l) <<< s_3^l$
X_4 = (10325476+ $\theta_0^l(X_1, X_2, X_1)$ + 80000000+00000000) <<< 12
$\theta_0^l(X_1, X_2, X_1) = X_i \oplus X`_{i-1} \oplus X_{i-2}$
$\theta_0^l(X_1, X_2, X_1) = X_3 \oplus X_{3-1} \oplus X_{3-2}$
$\theta_0^l(X_1, X_2, X_1)$= 6D251163⊕ B312B2B3⊕ D452479E
$\theta_0^l(X_1, X_2, X_1)$ = A65E44E
X_4 = (10325476+ A65E44E + 80000000+00000000) <<< 12
X_4= 9A9838C4<<<< 12
X_4 = 838C49A9

$Y_{i+1} = (Y_{i-3} + \theta_j^r(Y_i, Y_{i-1}, Y_{i-2}) + W_i^r + K_j^r) <<< s_i^r$
$Y_{3+1} = (Y_{3-3} + \theta_0^r(Y_3, Y_{3-1}, Y_{3-2}) + W_3^r + K_0^r) <<< s_3^r$
$Y_4 = (Y_0 + \theta_0^r(Y_3, Y_2, Y_1) + W_3^r + K_0^r) <<< s_3^r$
Y_4 = (10325476 + $\theta_0^r(Y_2, Y_1, Y_0)$ + 25504446+ 50A28BE6) <<<11
$\theta_0^r(Y_3, Y_2, Y_1) = Y_3 \wedge Y_{2-0} \oplus \sim Y_3 \wedge Y_{1-0}$
$\theta_0^r(Y_3, Y_2, Y_1) = Y_3 \wedge Y_2 \oplus \sim Y_3 \wedge Y_1$
$\theta_0^r(Y_3, Y_2, Y_1)$ = 285FB67D ^ 4527DAB1⊕ ~ 285FB67D ^ E7AEE6B7
$\theta_0^r(Y_3, Y_2, Y_1)$ = 79231⊕ C7A04082
$\theta_0^r(Y_3, Y_2, Y_1)$ = C7A7D2B3
Y_4 = (10325476 + C7A7D2B3 + 25504446+ 50A28BE6) <<<11
Y_4 = (4DCCF755) <<< 11
Y_4 = 67BAAA6

Step 4 : i=4, j=0
$X_{i+1} = (X_{i-3} + \theta_j^l(X_i, X_{i-1}, X_{i-2}) + W_i^l + K_j^l) <<< s_i^l$
$X_{4+1} = (X_{4-3} + \theta_0^l(X_4, X_{4-1}, X_{4-2}) + W_4^l + K_0^l) <<< s_4^l$
$X_5 = (X_1 + \theta_0^l(X_4, X_3, X_2) + W_4^l + K_0^l) <<< s_4^l$
X_5 = (10325476+ $\theta_0^l(X_4, X_3, X_2)$ + 00000000+00000000) <<< 5
$\theta_0^l(X_4, X_3, X_2) = X_i \oplus X`_{i-1} \oplus X_{i-2}$
$\theta_0^l(X_4, X_3, X_2) = X_4 \oplus X_{4-1} \oplus X_{4-2}$
$\theta_0^l(X_4, X_3, X_2)$ = 838C49A9⊕ 6D251163⊕ B312B2B3
$\theta_0^l(X_4, X_3, X_2)$ = 5DBBEA79
X_5 (10325476+ 5DBBEA79 + 00000000+00000000) <<< 5
X_5= 6DEE3EEF<<<< 5
X_5 = BDC7DDED

$Y_{i+1} = (Y_{i-3} + \theta_j^r(Y_i, Y_{i-1}, Y_{i-2}) + W_i^r + K_j^r) <<< s_i^r$
$Y_{4+1} = (Y_{4-3} + \theta_0^r(Y_4, Y_{4-1}, Y_{4-2}) + W_4^r + K_0^r) <<< s_4^r$
$Y_5 = (Y_1 + \theta_0^r(Y_4, Y_3, Y_2) + W_4^r + K_0^r) <<< s_4^r$
Y_5 = (E7AEE6B7 + $\theta_0^r(Y_4, Y_3, Y_2)$ + 00000000+ 50A28BE6) <<<13
$\theta_0^r(Y_4, Y_3, Y_2) = Y_4 \wedge Y_{3-0} \oplus \sim Y_4 \wedge Y_{2-0}$
$\theta_0^r(Y_4, Y_3, Y_2) = Y_4 \wedge Y_3 \oplus \sim Y_4 \wedge Y_2$
$\theta_0^r(Y_4, Y_3, Y_2)$ = 10325476^ 285FB67D⊕ ~ 10325476^ 4527DAB1
$\theta_0^r(Y_4, Y_3, Y_2)$ = 121474⊕ 45058A81
$\theta_0^r(Y_4, Y_3, Y_2)$ = 45179EF5
Y_5 = (E7AEE6B7 + 45179EF5 + 00000000+ 50A28BE6) <<<13
Y_5 = (7D691192) <<< 13
Y_5 = 22324FAD

Step 5 : i=5, j=0
$X_{i+1} = (X_{i-3} + \theta_j^l(X_i, X_{i-1}, X_{i-2}) + W_i^l + K_j^l) <<< s_i^l$
$X_{5+1} = (X_{5-3} + \theta_0^l(X_5, X_{5-1}, X_{5-2}) + W_5^l + K_0^l) <<< s_5^l$
$X_6 = (X_2 + \theta_0^l(X_5, X_4, X_3) + W_5^l + K_0^l) <<< s_5^l$

Journal of Computer System and Informatics (JoSYC)
Volume 3, No. 1, November 2021, Page 16−25
ISSN 2714-8912 (media online)
ISSN 2714-7150 (media cetak)
DOI 10.47065/josyc.v3i1.822

X_6 = (B312B2B3+ θ_0^l(X_5, X_4, X_3) + 00000000+00000000) <<< 8
θ_0^l(X_5, X_4, X_3) = $X_i \oplus X_{i-1} \oplus X_{i-2}$
θ_0^l(X_5, X_4, X_3) = $X_5 \oplus X_{5-1} \oplus X_{5-2}$
θ_0^l(X_5, X_4, X_3) = BDC7DDED ⊕ 83C49A9 ⊕ 6D251163
θ_0^l(X_5, X_4, X_3) = 536E8527
X_6 = (B312B2B3+ 536E8527 + 00000000+00000000) <<< 8
X_6 = (68137DA) <<< 8
X_6 = 8137DA06

$Y_{i+1} = (Y_{i-3} + \theta_j^r(Y_i, Y_{i-1}, Y_{i-2}) + W_i^r + K_j^r) <<< s_i^r$
$Y_{5+1} = (Y_{5-3} + \theta_0^r(Y_5, Y_{5-1}, Y_{5-2}) + W_5^r + K_0^r) <<< s_5^r$
$Y_6 = (Y_3 + \theta_0^r(Y_5, Y_4, Y_3) + W_5^r + K_0^r) <<< s_5^r$
Y_6 = (285FB67D+ θ_0^r(Y_5, Y_4, Y_3) + 00000000 + 50A28BE6) <<< 15
$\theta_j^r(Y_5, Y_{4-0}, Y_{3-0}) = Y_5 \wedge Y_{4-0} \oplus \sim Y_5 \wedge Y_{3-0}$
$\theta_0^r(Y_5, Y_4, Y_3) = Y_5 \wedge Y_4 \oplus \sim Y_5 \wedge Y_3$
$\theta_0^r(Y_5, Y_4, Y_3)$ = 22324FAD ^ 67BAAA6E⊕ ~22324FAD ^ 285FB67D
$\theta_0^r(Y_5, Y_4, Y_3)$ = 22320A2C⊕ 84DB050
$\theta_0^r(Y_5, Y_4, Y_3)$ = 22320A2C
Y_6 = 285FB67D+ 22320A2C + 00000000 + 50A28BE6) <<< 15
Y_6 = (9B344C8F) <<< 15
Y_6 = 2647CD9A

6. *Output*
   Keluaran nilai *hash* dari algoritma RIPEMD-128 dihasilkan dari 64 langkah di kedua cabangnya. 4 *word* berukuran 32 bit di dapatkan dari perhitungan berikut:
   $H_0 = X_{63} + Y_{62} + H_1$
   H_0 = 96E12657 + 475D09DD + 00000000
   H_0 = DE3E3034
   $H_1 = X_{62} + Y_{61} + H_2$
   H_1 = 7B9A6D51 + 22324FAD + 00000000
   H_1 = 9DCCBCFE
   $H_2 = X_{61} + Y_{64} + H_3$
   H_2 = BDC7DDED + 6E558E4C + 00000000
   H_2 = 2C1D6C39
   $H_3 = X_{64} + Y_{63} + H_0$
   H_3 = 96E12657 + E7AEE6B7 + 00000000
   H_3 = 7E900D0E

Sehingga keluaran dari nilai *hash* nya adalah DE3E30349DCCBCFE2C1D6C397E900D0E, nilai ini lah yang dijadikan acuan untuk menghapus *file pdf*, jika ada lebih dari satu *file pdf* yang memiliki nilai *hash* yang sama.

4. KESIMPULAN

Dari hasil penulisan dan analisa dari bab-bab sebelumnya, maka dapat diambil kesimpulan, dimana kesimpulan-kesimpulan tersebut kiranya dapat berguna bagi para pembaca, sehingga penulisan skripsi ini dapat lebih bermanfaat. Adapun kesimpulan-kesimpulan tersebut adalah sebagai Pencarian *file* pdf yang ganda atau duplikat di dalam sebuah media penyimpanan dapat dilakukan dengan memberikan identitas dari setiap *file* pdf menggunakan fungsi *hash* sehingga tidak perlu membuka dan membaca isi *file* pdf satu persatu Algoritma fungsi *hash* RIPEMD-128 dapat digunakan untuk merepresentasikan isi dari *file* pdf sehingga dapat digunakan untuk membandingkan *file* pdf yang ganda atau duplikat.

REFERENCES

[1] Shrivastava, “Keamanan,” 2016.
[2] J. W. &. Sons, “Kriptgrafi,” 1996.
[3] H. Dobbertin, A. Bosselaers dan B. Preneel, “A Strengthened Version of RIPEMD,” Springer, Belgium, 1996.
[4] T. Peyrin, “Cryptanalysis of Full RIPEMD-128,” Journal of Cryptologi, vol. III, no. 12, pp. 927-951, 2015.
[5] Y. Wicaksono, Seri Penuntun Praktis Membongkar File PDF, Jakarta: Elex Media Komputindo, 2013.
[6] S. Patil, N. Jagtap, S. Rajput dan R. Sangore, “A Duplicate File Finder System,” International Journal of Science Spirituality Business and Technology, pp. 10-14, 2017.