Journal of Informatics, Electrical and Electronics Engineering
ISSN 2807-9507(Media Online)
Vol 2, No 2, Desember 2022
Hal 39-50
https://djournals.com/jieee

Perancangan Aplikasi Duplicate PDF Scanner Menerapkan Algoritma MD2

Felisiana Apustriani Gulo

Fakultas Ilmu Komputer dan Teknologi Informasi, Program Studi Teknik Informatika, Universitas Budi Darma, Medan, Indonesia
Email: felisianarni04@gmail.com
Email Penulis Korespondensi: felisianarni04@gmail.com

Abstrak−Saat ini kita sudah berada dalam era dunia digital. Banyak industri yang sudah beralih ke teknologi digital seperti foto, video, musik, dan sebagainya. Tentunya perubahan ini membawa banyak manfaat untuk kita 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 , salah satu fungsi hash dalam kriptografi adalah algoritma MD 2 atau Message Digest 2. 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 tersebut.

Kata Kunci: Duplikat; Message Digest; Verifikasi; MD2

Abstract−We are now in the era of the digital world. Many industries have switched to digital technology such as photos, videos, music, and so on. Of course, this change brings many benefits for us in terms of speed and data access. The 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, one of the hash functions in cryptography is the MD 2 algorithm or Message Digest 2. By using the 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: Duplicate; Message Digest; Verification; MD2

1. PENDAHULUAN

Kriptografi berperan penting dalam layanan keamanan seperti kerahasiaan, integritas data, otentikasi dan pencegahan penyangkalan. Kriptografi modern menggunakan kunci yang harus dirahasiakan untuk mengatasi masalah keamanan kriptografi. Permasalahan dalam penggunaan satu kunci yang sama oleh dua entitas yang saling berkomunikasi untuk bertukar pesan adalah cara mendistribusikan kunci. Permasalahan ini dapat diatasi dengan penggunaan kriptografi kunci-publik, yang memungkinkan pengguna berkomunikasi secara aman tanpa perlu berbagi kunci rahasia.

Banyak industri yang sudah beralih ke teknologi digital seperti foto, video, musik, dan sebagainya.Tentunya perubahan ini membawa banyak manfaat untuk kita 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*, salah satu fungsi *hash* dalam kriptografi adalah algoritma MD 2 atau *Message Digest 2*. 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.

2. METODOLOGI PENELITIAN

2.1 Kriptografi

Awal mula kriptografi dipahami sebagai ilmu tentang menyembunyikan pesan, tetapi seiring perkembangan zaman hingga saat ini pengertian kriptografi berkembang menjadi ilmu tentang teknik matematis yang digunakan untuk menyelesaikan persoalan keamanan berupa privasi dan otentikasi [2]. Kriptografi adalah suatu ilmu yang mempelajari penulisan secara rahasia. Kriptografi merupakan bagian dari suatu cabang ilmu matematika yang disebut Cryptology. Kriptografi bertujuan menjaga kerahasiaan informasi yang terkandung dalam data sehingga informasi tersebut tidak dapat diketahui oleh pihak yang tidak sah. Dalam menjaga kerahasiaan data, kriptografi mentransformasikan data jelas *(plaintext)* ke dalam bentuk data sandi *(ciphertext)* yang tidak dapat dikenali. *Ciphertext* inilah yang kemudian dikirimkan oleh pengirim *(sender)* kepada penerima *(receiver)*. Setelah sampai di penerima, *ciphertext* tersebut ditranformasikan kembali ke dalam bentuk *plaintext* agar dapat dikenali.

2.2 Algoritma MD2

Message Digest 2 pertama kali dirancang pada tahun 1989 dan dirancang untuk komputer berbasiskan 8-bit. Detail tentang MD2 dapat dilihat di RFC 1319. Walaupun memiliki banyak *flaw*, sampai sekarang MD2 masih dipakai sebagai infrastruktur untuk sertifikat RSA. MD2 mengubah pesan *string* ke dalam kode heksadesimal 32 bit. Secara fisik, MD2 bekerja dengan mengkompresi 128 bit *hash value* dari pesan sembarang ke dalam blok-blok yang masing-masing berukuran 128 bit (16 *byte*) kemudian menambahkan sebuah *checksum*. Untuk kalkulasi yang sebenarnya, digunakan blok sebesar 48 *byte* dan tabel 256 *byte* yang dihasilkan secara tidak langsung. Apabila semua block dari pesan yang dipanjangkan telah diproses, fragmen pertama dari blok 48 *byte* menjadi *hash value* dari pesan[3].

Algoritma MD2 dikembangkan oleh Ron Rivest pada tahun 1989. Algoritma ini dioptimasikan dengan menggunakan komputer 8-bit. MD2 sebenarnya dispesifikasikan dalam RFC 1319. Algoritma MD2 menghasilkan nilai *hash* yang berukuran 128-bit dan menerima input pesan dengan panjang yang tidak ditentukan. Pesan yang akan dijadikan input untuk fungsi *hash* ini akan terlebih dahulu akan di*padding*. Untuk kalkulasi yang sebenarnya. Misalkan kita memiliki sejumlah b-*byte* pesan sebagai input, dan kita mengharapkan untuk dapat mendapatkan *message digest* dari pesan tersebut. Dalam hal ini, b merupakan suatu bilangan bulat sembarang yang bernilai positif, bisa juga nol, dan besarnya sembarang. Kita nyatakan bahwa *byte* dari pesan ditulis dalam bentuk :
Di bawah ini akan dijelaskan 4 tahap proses untuk menghasilkan *message digest* untuk algoritma MD2.

1. Memasukkan *Padding byte*
   Pesan akan ditambahkan melalui proses *padding* sehingga panjang pesan tersebut kongruen dengan 0 modulo 16. Maka, pesan akan diperluas sehingga panjangnya merupakan kelipatan dari 16 *byte*. Proses *padding* ini selalu dilakukan meskipun panjang pesan awal sebelum dilakukan *padding* sudah kongruen dengan 0 modulo 16. *Padding* dilakukan dengan mengikuti : "i" *byte* dari nilai "i" akan ditambahkan pada pesan sehingga panjang pesan kongruen dengan 0 modulo 16. Maka ukuran *padding byte* paling sedikit 1 *byte* sampai 16 *byte*. Pada tahap ini pesan hasil *padding* memiliki panjang pesan kelipatan 16 *byte*. Kita bagi pesan menjadi M[0 . . . N-1] dimana N merupakan kelipatan 16.
2. Memasukkan *Checksum*
   Sebanyak 16 *byte checksum* dari pesan akan ditambahkan pada hasil dari tahap sebelumnya. Pada langkah ini digunakan sebuah 256-*byte* yang dibangkitkan secara acak yang dibuat dengan nilai digit dari pi. Jika S[i] menotasikan untuk elemen ke-i pada tabel.
3. Inisialisasi Penyangga MD
   Sebuah 48-*byte* penyangga X digunakan untuk menghasilkan *message digest*, nilai penyangga diinisialisasi dengan nol.
4. Proses pesan dalam blok 16-*byte*
   Langkah ini menggunakan angka hasil pemnbangkitan sebanya 256-*byte* yang sama yang dihasilkan pada proses 2.

3. HASIL DAN PEMBAHASAN

3.1 Analisa Masalah

Permasalahan yang di hadapi ketika ingin menghapus *file pdf* yang duplikat atau ganda dalam sebuah media penyimpanan adalah pengguna harus membuka dan melihat isi dari *file pdf* tersebut satu persatu dan membandingkan dengan isi dari *file pdf* yang lainnya. Hal ini tentunya sangat merepotkan pengguna, apalagi jika di dalam media penyimpanan tersebut terdapat banyak *file pdf*.

Untuk mengatasi hal tersebut maka di perlukan sebuah aplikasi yang dapat mencari dan mengelompokkan *file pdf* yang ganda atau duplikat sehingga pengguna dapat dengan mudah menghapus *file pdf* yang ganda dan hanya menyisakan satu *file pdf* saja. Aplikasi ini bisa disebut sebagai aplikasi *duplicate pdf scanner*. Dengan memanfaatkan aplikasi ini maka pengguna akan dengan mudah menghapus *file pdf* yang ganda atau duplikat.

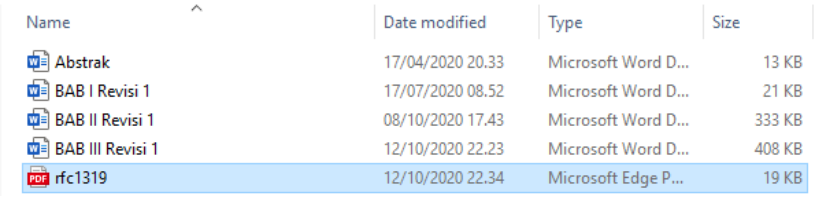
Masalah yang perlu diselesaikan berikutnya adalah bagaimana mengidentifikasi *file pdf* yang duplikat. Dengan memanfaatkan fungsi *hash* yang ada dalam cabang ilmu kriptografi maka identitas dari setiap *file pdf* dapat di bangkitkan. Dengan memanfaatkan identitas ini maka jika terdapat *file pdf* yang memiliki nilai *hash* yang sama bisa di pastikan jika *file pdf* tersebut merupakan *file pdf* yang ganda atau duplikat.

Dalam aplikasi *Duplicate pdf Scanner* yang akan di bangun ini menerapkan algoritma MD2 yang digunakan untuk membangkitkan identitas dari setiap *file pdf* yang terdapat dalam sebuah media penyimpanan. Langkah

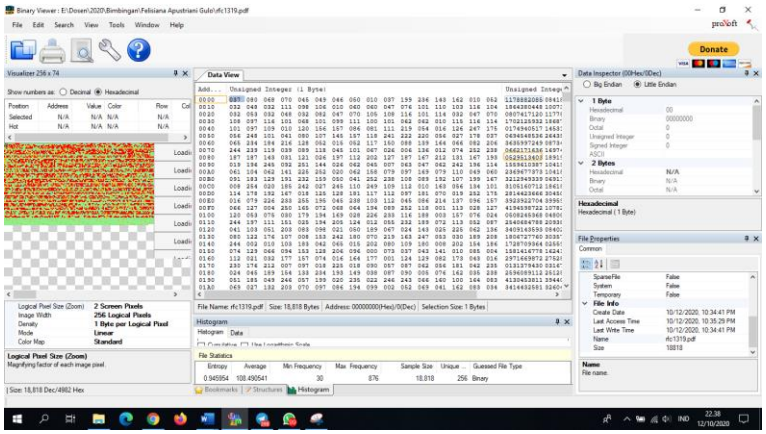
pertama yang di tempuh ketika akan melakukan pencarian *file pdf* yang duplikat atau ganda adalah memilih media penyimpanan yang akan di lakukan pencarian *file pdf*. Setelah menentukan media penyimpanannya maka langkah selanjutnya adalah mencari *file* dengan ekstensi *pdf* dan mengabaikan *file* dengan ekstensi lainnya. Setelah menemukan semua *file pdf* pada media penyimpanan tersebut maka langkah selanjutnya adalah membangkitkan identitas dari setiap *file pdf* yang telah di temukan. Selanjutnya adalah melakukan pengurutan *file pdf* berdasarkan nilai dari identitasnya. Jika terdapat *file pdf* dengan identitas yang sama maka pengguna dapat menghapus *file* tersebut dan hanya menyisakan satu *file pdf* saja.

3.1.1 Penerapan Algoritma MD2

Dalam hal ini pengguna akan menggunakan *file* berekstensi pdf dengan ukuran file 19 KB dan menggunakan aplikasi bantuan *binary viewer* untuk mendapatkan bilangan *hexadecimal* dari file tersebut.



Gambar 1. *File Pdf*



Gambar 2. Aplikasi Binary Viewer

Dengan menggunakan bantuan aplikasi binary viewer tersebut makan dapat dilihat nilai integer dari *file pdf* tersebut. Untuk mempermudah proses perhitungan secara manual maka diambil sampel dari mulai *address* 0000 sebanyak 12 byte. Maka data yang di dapat disajikan dalam tabel 1. sebagai berikut:

Tabel 1. Data Sample

M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11
37	80	68	70	45	49	46	50	10	37	199	236

1. Menambahkan *Padding Bytes*
   Penambahan *padding bytes* dilakukan jika panjang pesan tidak sampai 16 bytes. Pesan akan di tambahkan *padding bytes* hingga panjang pesan kongruen dengan 0 modulus 16. Artinya pesan tersebut di perpanjang hingga panjang pesan merupakan kelipatan dari 16. Penambahan pesan dilakukan dengan memberikan nilai ke i pada posisi ke i. Maka hasilnya dapat di lihat pada tabel 2. di bawah ini:

Tabel 2. Penambahan *Padding Bytes*

M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15
37	80	68	70	45	49	46	50	10	37	199	236	**12**	**13**	**14**	**15**

2. *Append Checksum*
   Langkah selanjutnya adalah penambahan *Checksum*, langkah ini menggunakan permutasi acak 256 byte yang di bangun dari digit pi.

Tabel 3. Permutasi Acak 256 Byte

Indeks	0	1	2	3	4	5	6	7
Nilai pi	41	46	67	201	162	216	124	1

Indeks	8	9	10	11	12	13	14	15
Nilai pi	61	54	84	161	236	240	6	19
Indeks	16	17	18	19	20	21	22	23
Nilai pi	98	167	5	243	192	199	115	140
Indeks	24	25	26	27	28	29	30	31
Nilai pi	152	147	43	217	188	76	130	202
Indeks	32	33	34	35	36	37	38	39
Nilai pi	30	155	87	60	253	212	224	22
Indeks	40	41	42	43	44	45	46	47
Nilai pi	103	66	111	24	138	23	229	18
Indeks	48	49	50	51	52	53	54	55
Nilai pi	190	78	196	214	218	158	222	73
Indeks	56	57	58	59	60	61	62	63
Nilai pi	160	251	245	142	187	47	238	122
Indeks	64	65	66	67	68	69	70	71
Nilai pi	169	104	121	145	21	178	7	63
Indeks	72	73	74	75	76	77	78	79
Nilai pi	148	194	16	137	11	34	95	33
Indeks	80	81	82	83	84	85	86	87
Nilai pi	128	127	93	154	90	144	50	39
Indeks	88	89	90	91	92	93	94	95
Nilai pi	53	62	204	231	191	247	151	3
Indeks	96	97	98	99	100	101	102	103
Nilai pi	255	25	48	179	72	165	181	209
Indeks	104	105	106	107	108	109	110	111
Nilai pi	215	94	146	42	172	86	170	198
Indeks	112	113	114	115	116	117	118	119
Nilai pi	79	184	56	210	150	164	125	182
Indeks	120	121	122	123	124	125	126	127
Nilai pi	118	252	107	226	156	116	4	241
Indeks	128	129	130	131	132	133	134	135
Nilai pi	69	157	112	89	100	113	135	32
Indeks	136	137	138	139	140	141	142	143
Nilai pi	134	91	207	101	230	45	168	2
Indeks	144	145	146	147	148	149	150	151
Nilai pi	27	96	37	173	174	176	185	246
Indeks	152	153	154	155	156	157	158	159
Nilai pi	28	70	97	105	52	64	126	15
Indeks	160	161	162	163	164	165	166	167
Nilai pi	85	71	163	35	221	81	175	58
Indeks	168	169	170	171	172	173	174	175
Nilai pi	195	92	249	206	186	197	234	38

Indeks	176	177	178	179	180	181	182	183
Nilai pi	44	83	13	110	133	40	132	9
Indeks	184	185	186	187	188	189	190	191
Nilai pi	211	223	205	244	65	129	77	82
Indeks	192	193	194	195	196	197	198	199
Nilai pi	106	220	55	200	108	193	171	250
Indeks	200	201	202	203	204	205	206	207
Nilai pi	36	225	123	8	12	189	177	74
Indeks	208	209	210	211	212	213	214	215
Nilai pi	120	136	149	139	227	99	232	109
Indeks	216	217	218	219	220	221	222	223
Nilai pi	233	203	213	254	59	0	29	57
Indeks	224	225	226	227	228	229	230	231
Nilai pi	242	239	183	14	102	88	208	228
Indeks	232	233	234	235	236	237	238	239
Nilai pi	166	119	114	248	235	117	75	10
Indeks	240	242	242	243	244	245	246	247
Nilai pi	49	68	80	180	143	237	31	26
Indeks	248	249	250	251	252	253	254	255
Nilai pi	219	153	141	51	159	17	131	20

Untuk melakukan penambahan *Checksum* di lakukan sebanyak 16 kali sebagai berikut:

Iterasi 0 : I=0, J=0, L=0

Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]
Set C to M [0 * 16+0]	Set C[0] to S [37 xor 0]	Set L to C[0]
Set C to M [0]	Set C[0] to S [37]	Set L to 212
Set C to 37	Set C[0] to 212	

Iterasi 1 : I=0, J=1, L=212

Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]
Set C to M [0 * 16+1]	Set C[1] to S [80 xor 212]	Set L to C[1]
Set C to M [1]	Set C[1] to S [132]	Set L to 100
Set C to 80	Set C[1] to 100	

Iterasi 2 : I=0, J=2, L=100

Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]
Set C to M [0 * 16+2]	Set C[2] to S [68 xor 100]	Set L to C[2]
Set C to M [2]	Set C[2] to S [32]	Set L to 30
Set C to 68	Set C[2] to 30	

Iterasi 3 : I=0, J=3, L=30

Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]
Set C to M [0 * 16+3]	Set C[3] to S [70 xor 30]	Set L to C[3]
Set C to M [3]	Set C[3] to S [88]	Set L to 53
Set C to 70	Set C[3] to 53	

Iterasi 4 : I=0, J=4, L=53

Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]
Set C to M [0 * 16+4]	Set C[4] to S [45 xor 53]	Set L to C[4]
Set C to M [4]	Set C[4] to S [24]	Set L to 152
Set C to 45	Set C[4] to 152	

Iterasi 5 : I=0, J=5, L=152

Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]
Set C to M [0 * 16+5]	Set C[5] to S [49 xor 152]	Set L to C[5]
Set C to M [5]	Set C[5] to S [169]	Set L to 92
Set C to 49	Set C[5] to 92	

Iterasi 6 : I=0, J=6, L=92

Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]

Set C to M [0 * 16+6]	Set C[6] to S [46 xor 92]	Set L to C[6]
Set C to M [6]	Set C[6] to S [114]	Set L to 56
Set C to 46	Set C[6] to 56	
Iterasi 7 : I=0, J=7, L=56		
Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]
Set C to M [0 * 16+7]	Set C[7] to S [50 xor 56]	Set L to C[7]
Set C to M [7]	Set C[7] to S [10]	Set L to 84
Set C to 50	Set C[7] to 84	
Iterasi 8 : I=0, J=8, L=84		
Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]
Set C to M [0 * 16+8]	Set C[8] to S [10 xor 84]	Set L to C[8]
Set C to M [8]	Set C[8] to S [94]	Set L to 151
Set C to 10	Set C[8] to 151	
Iterasi 9 : I=0, J=9, L=151		
Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]
Set C to M [0 * 16+9]	Set C[9] to S [37 xor 151]	Set L to C[9]
Set C to M [9]	Set C[9] to S [178]	Set L to 13
Set C to 37	Set C[9] to 13	
Iterasi 10 : I=0, J=10, L=13		
Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]
Set C to M [0 * 16+10]	Set C[10] to S [199 xor 13]	Set L to C[10]
Set C to M [10]	Set C[10] to S [202]	Set L to 123
Set C to 199	Set C[10] to 123	
Iterasi 11 : I=0, J=11, L=123		
Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]
Set C to M [0 * 16+11]	Set C[11] to S [236 xor 123]	Set L to C[11]
Set C to M [11]	Set C[11] to S [151]	Set L to 246
Set C to 236	Set C[11] to 246	
Iterasi 12 : I=0, J=12, L=246		
Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]
Set C to M [0 * 16+12]	Set C[12] to S [12 xor 246]	Set L to C[12]
Set C to M [12]	Set C[12] to S [250]	Set L to 141
Set C to 12	Set C[12] to 141	
Iterasi 13 : I=0, J=13, L=141		
Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]
Set C to M [0 * 16+13]	Set C[13] to S [13 xor 141]	Set L to C[13]
Set C to M [13]	Set C[13] to S [128]	Set L to 69
Set C to 13	Set C[13] to 69	
Iterasi 14 : I=0, J=14, L=69		
Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]
Set C to M [0 * 16+14]	Set C[14] to S [14 xor 69]	Set L to C[14]
Set C to M [14]	Set C[14] to S [75]	Set L to 137
Set C to 14	Set C[14] to 137	
Iterasi 15 : I=0, J=15, L=137		
Set C to M [I * 16+J]	Set C[J] to S [C xor L]	Set L to C[J]
Set C to M [0 * 16+15]	Set C[15] to S [15 xor 137]	Set L to C[15]
Set C to M [15]	Set C[15] to S [134]	Set L to 135
Set C to 15	Set C[15] to 135	

Setelah selesai melakukan perhitungan hingga 16 iterasi maka hasilnya dapat dilihat pada tabel 4. berikut ini:

Tabel 4. Hasil Penambahan *Checksum*

M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15
37	80	68	70	45	49	46	50	10	37	199	236	12	13	14	15
M16	M17	M18	M19	M20	M21	M22	M23	M24	M25	M26	M27	M28	M29	M30	M31
212	100	30	53	152	92	56	84	151	13	123	26	141	69	137	135

3. Inisialisasi *MD Buffer*
   Sebuah *Buffer* atau penyangga X yang berukuran 48 *Bytes* digunakan untuk menghitung *message digest*. *Buffer* atau penyangga ini di inisialisasikan atau di beri nilai awal dengan nilai 0. Untuk hasil inisialisasi *MD buffer* dapat di lihat pada tabel 5. berikut ini:

Tabel 5. Inisialisasi *MD Buffer*

X0	X1	X2	X3	X4	X5	X6	X7
0	0	0	0	0	0	0	0
X8	X9	X10	X11	X12	X13	X14	X15
0	0	0	0	0	0	0	0
X16	X17	X18	X19	X20	X21	X22	X23
0	0	0	0	0	0	0	0
X24	X25	X26	X27	X28	X29	X30	X31
0	0	0	0	0	0	0	0
X32	X33	X34	X35	X36	X37	X38	X39
0	0	0	0	0	0	0	0
X40	X41	X42	X43	X44	X45	X46	X47
0	0	0	0	0	0	0	0

4. Memproses Pesan dalam 16 Bytes Blok
   Untuk memproses pesan dalam 16 blok, maka lakukan subsitusi dari blok i ke blok x dengan jumlah iterasi sebanyak 16 kali, dapat dilihat pada perhitungan di bawah ini:

Iterasi 0 : I=0, J=0

Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+0] to M [0 * 16+0]	Set X [32+0] to (X [16+0] xor X[0])
Set X[16] to M [0]	Set X [32] to (X [16] xor X[0])
Set X[16] to 37	Set X [32] to (37 xor 0)
	Set X [32] to 37

Iterasi 1 : I=0, J=1

Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+1] to M [0 * 16+1]	Set X [32+1] to (X [16+1] xor X[1])
Set X[17] to M [1]	Set X [33] to (X [17] xor X[1])
Set X[17] to 80	Set X [33] to (80 xor 0)
	Set X [33] to 80

Iterasi 2 : I=0, J=2

Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+2] to M [0 * 16+2]	Set X [32+2] to (X [16+2] xor X[2])
Set X[18] to M [2]	Set X [34] to (X [18] xor X[2])
Set X[18] to 68	Set X [34] to (68 xor 0)
	Set X [34] to 68

Iterasi 3 : I=0, J=3

Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+3] to M [0 * 16+3]	Set X [32+3] to (X [16+3] xor X[3])
Set X[19] to M [3]	Set X [35] to (X [19] xor X[3])
Set X[19] to 70	Set X [35] to (70 xor 0)
	Set X [35] to 70

Iterasi 4 : I=0, J=4

Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+4] to M [0 * 16+4]	Set X [32+4] to (X [16+4] xor X[4])
Set X[20] to M [4]	Set X [36] to (X [20] xor X[4])
Set X[20] to 45	Set X [36] to (45 xor 0)
	Set X [36] to 45

Iterasi 5 : I=0, J=5

Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+5] to M [0 * 16+5]	Set X [32+5] to (X [16+5] xor X[5])
Set X[21] to M [5]	Set X [37] to (X [21] xor X[5])
Set X[21] to 49	Set X [37] to (49 xor 0)
	Set X [37] to 49

Iterasi 6 : I=0, J=6

Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+6] to M [0 * 16+6]	Set X [32+6] to (X [16+6] xor X[6])
Set X[22] to M [6]	Set X [38] to (X [22] xor X[6])
Set X[22] to 46	Set X [38] to (46 xor 0)
	Set X [38] to 46

Iterasi 7 : I=0, J=7

Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+7] to M [0 * 16+7]	Set X [32+7] to (X [16+7] xor X[7])
Set X[23] to M [7]	Set X [39] to (X [23] xor X[7])
Set X[23] to 50	Set X [39] to (50 xor 0)
	Set X [39] to 50
Iterasi 8 : I=0, J=8	
Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+8] to M [0 * 16+8]	Set X [32+8] to (X [16+8] xor X[8])
Set X[24] to M [8]	Set X [40] to (X [24] xor X[8])
Set X[24] to 10	Set X [40] to (10 xor 0)
	Set X [40] to 10
Iterasi 9 : I=0, J=9	
Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+9] to M [0 * 16+9]	Set X [32+9] to (X [16+9] xor X[9])
Set X[25] to M [9]	Set X [41] to (X [25] xor X[9])
Set X[25] to 37	Set X [41] to (37 xor 0)
	Set X [41] to 37
Iterasi 10 : I=0, J=10	
Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+10] to M [0 * 16+10]	Set X [32+10] to (X [16+10] xor X[10])
Set X[26] to M [10]	Set X [42] to (X [26] xor X[10])
Set X[26] to 199	Set X [42] to (199 xor 0)
	Set X [42] to 199
Iterasi 11 : I=0, J=11	
Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+11] to M [0 * 16+11]	Set X [32+11] to (X [16+11] xor X[11])
Set X[27] to M [11]	Set X [43] to (X [27] xor X[11])
Set X[27] to 236	Set X [43] to (236 xor 0)
	Set X [43] to 236
Iterasi 12 : I=0, J=12	
Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+12] to M [0 * 16+12]	Set X [32+12] to (X [16+12] xor X[12])
Set X[28] to M [12]	Set X [44] to (X [28] xor X[12])
Set X[28] to 12	Set X [44] to (12 xor 0)
	Set X [44] to 12
Iterasi 13 : I=0, J=13	
Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+13] to M [0 * 16+13]	Set X [32+13] to (X [16+13] xor X[13])
Set X[29] to M [13]	Set X [45] to (X [29] xor X[13])
Set X[29] to 13	Set X [45] to (13 xor 0)
	Set X [45] to 13
Iterasi 14 : I=0, J=14	
Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+14] to M [0 * 16+14]	Set X [32+14] to (X [16+14] xor X[14])
Set X[30] to M [14]	Set X [46] to (X [30] xor X[14])
Set X[30] to 14	Set X [46] to (14 xor 0)
	Set X [46] to 14
Iterasi 15 : I=0, J=15	
Set X [16+J] to M [I * 16+J]	Set X [32+J] to (X [16+J] xor X[J])
Set X [16+15] to M [0 * 16+15]	Set X [32+15] to (X [16+15] xor X[15])
Set X[31] to M [15]	Set X [47] to (X [31] xor X[15])
Set X[31] to 15	Set X [47] to (15 xor 0)
	Set X [47] to 15

Tabel 6. Hasil Proses 16 Bytes Word

X0	X1	X2	X3	X4	X5	X6	X7
0	0	0	0	0	0	0	0
X8	X9	X10	X11	X12	X13	X14	X15
0	0	0	0	0	0	0	0
X16	X17	X18	X19	X20	X21	X22	X23
37	80	68	70	45	49	46	50

X24	X25	X26	X27	X28	X29	X30	X31
10	37	199	236	12	13	14	15
X32	X33	X34	X35	X36	X37	X38	X39
37	80	68	70	45	49	46	50
X40	X41	X42	X43	X44	X45	X46	X47
10	37	199	236	12	13	14	15

5. *Output*

Iterasi 0 : K=0, T=0
Set T and X [K] to x [K] xor S[T]
Set T and X [0] to x [0] xor S[0]
Set T and X [0] to x [0] xor S[0]
Set T and X [0] to 0 xor 41
Set T and X [0] to 41

Iterasi 1 : K=1, T=41
Set T and X [K] to x [K] xor S[T]
Set T and X [1] to x [1] xor S[41]
Set T and X [1] to x [1] xor S[41]
Set T and X [1] to 0 xor 66
Set T and X [1] to 66

Iterasi 2 : K=2, T=66
Set T and X [K] to x [K] xor S[T]
Set T and X [2] to x [2] xor S[66]
Set T and X [2] to x [2] xor S[66]
Set T and X [2] to 0 xor 121
Set T and X [2] to 121

Iterasi 3 : K=3, T=121
Set T and X [K] to x [K] xor S[T]
Set T and X [3] to x [3] xor S[121]
Set T and X [3] to x [3] xor S[121]
Set T and X [3] to 0 xor 252
Set T and X [3] to 252

Iterasi 4 : K=4, T=252
Set T and X [K] to x [K] xor S[T]
Set T and X [4] to x [4] xor S[252]
Set T and X [4] to x [4] xor S[252]
Set T and X [4] to 0 xor 159
Set T and X [4] to 159

Iterasi 5 : K=5, T=159
Set T and X [K] to x [K] xor S[T]
Set T and X [5] to x [5] xor S[159]
Set T and X [5] to x [5] xor S[159]
Set T and X [5] to 0 xor 15
Set T and X [5] to 15

Iterasi 6 : K=6, T=15
Set T and X [K] to x [K] xor S[T]
Set T and X [6] to x [6] xor S[15]
Set T and X [6] to x [6] xor S[15]
Set T and X [6] to 0 xor 19
Set T and X [6] to 19

Iterasi 7 : K=7, T=19
Set T and X [K] to x [K] xor S[T]
Set T and X [7] to x [7] xor S[19]
Set T and X [7] to x [7] xor S[19]
Set T and X [7] to 0 xor 243
Set T and X [7] to 243

Iterasi 8 : K=8, T=243
Set T and X [K] to x [K] xor S[T]
Set T and X [8] to x [8] xor S[243]
Set T and X [8] to x [8] xor S[243]
Set T and X [8] to 0 xor 180
Set T and X [8] to 180

Iterasi 9 : K=9, T=180
Set T and X [K] to x [K] xor S[T]
Set T and X [9] to x [9] xor S[180]
Set T and X [9] to x [9] xor S[180]
Set T and X [9] to 0 xor 133
Set T and X [9] to 133

Iterasi 10 : K=10, T=133
Set T and X [K] to x [K] xor S[T]
Set T and X [10] to x [10] xor S[133]
Set T and X [10] to x [10] xor S[133]
Set T and X [10] to 0 xor 113
Set T and X [10] to 113

Iterasi 11 : K=11, T=113
Set T and X [K] to x [K] xor S[T]
Set T and X [11] to x [11] xor S[113]
Set T and X [11] to x [11] xor S[113]
Set T and X [11] to 0 xor 184
Set T and X [11] to 184

Iterasi 12 : K=12, T=184
Set T and X [K] to x [K] xor S[T]
Set T and X [12] to x [12] xor S[184]
Set T and X [12] to x [12] xor S[184]
Set T and X [12] to 0 xor 211
Set T and X [12] to 211

Iterasi 13 : K=13, T=211
Set T and X [K] to x [K] xor S[T]
Set T and X [13] to x [13] xor S[211]
Set T and X [13] to x [13] xor S[211]
Set T and X [13] to 0 xor 139
Set T and X [13] to 139

Iterasi 14 : K=14, T=139
Set T and X [K] to x [K] xor S[T]
Set T and X [14] to x [14] xor S[139]
Set T and X [14] to x [14] xor S[139]
Set T and X [14] to 0 xor 101

Iterasi 15 : K=15, T=101
Set T and X [K] to x [K] xor S[T]
Set T and X [15] to x [15] xor S[101]
Set T and X [15] to x [15] xor S[101]
Set T and X [15] to 0 xor 165

Set T and X [14] to 101

Set T and X [15] to 165

Iterasi 16 : K=16, T=165
Set T and X [K] to x [K] xor S[T]
Set T and X [16] to x [16] xor S[165]
Set T and X [16] to x [16] xor S[165]
Set T and X [16] to 37 xor 81
Set T and X [16] to 116

Iterasi 17 : K=17, T=116
Set T and X [K] to x [K] xor S[T]
Set T and X [17] to x [17] xor S[116]
Set T and X [17] to x [17] xor S[116]
Set T and X [17] to 80 xor 150
Set T and X [17] to 198

Iterasi 18 : K=18, T=198
Set T and X [K] to x [K] xor S[T]
Set T and X [18] to x [18] xor S[198]
Set T and X [18] to x [18] xor S[198]
Set T and X [18] to 68 xor 171
Set T and X [18] to 239

Iterasi 19 : K=19, T=239
Set T and X [K] to x [K] xor S[T]
Set T and X [19] to x [19] xor S[239]
Set T and X [19] to x [19] xor S[239]
Set T and X [19] to 70 xor 185
Set T and X [19] to 255

Iterasi 20 : K=20, T=255
Set T and X [K] to x [K] xor S[T]
Set T and X [20] to x [20] xor S[255]
Set T and X [20] to x [20] xor S[255]
Set T and X [20] to 45 xor 20
Set T and X [20] to 57

Iterasi 21 : K=21, T=57
Set T and X [K] to x [K] xor S[T]
Set T and X [21] to x [21] xor S[57]
Set T and X [21] to x [21] xor S[57]
Set T and X [21] to 49 xor 251
Set T and X [21] to 202

Iterasi 22 : K=22, T=202
Set T and X [K] to x [K] xor S[T]
Set T and X [22] to x [22] xor S[202]
Set T and X [22] to x [22] xor S[202]
Set T and X [22] to 46 xor 123
Set T and X [22] to 85

Iterasi 23 : K=23, T=85
Set T and X [K] to x [K] xor S[T]
Set T and X [23] to x [23] xor S[85]
Set T and X [23] to x [23] xor S[85]
Set T and X [23] to 50 xor 144
Set T and X [23] to 162

Iterasi 24 : K=24, T=162
Set T and X [K] to x [K] xor S[T]
Set T and X [24] to x [24] xor S[162]
Set T and X [24] to x [24] xor S[162]
Set T and X [24] to 10 xor 163
Set T and X [24] to 169

Iterasi 25 : K=25, T=169
Set T and X [K] to x [K] xor S[T]
Set T and X [25] to x [25] xor S[169]
Set T and X [25] to x [25] xor S[169]
Set T and X [25] to 37 xor 92
Set T and X [25] to 121

Iterasi 26 : K=26, T=121
Set T and X [K] to x [K] xor S[T]
Set T and X [26] to x [26] xor S[121]
Set T and X [26] to x [26] xor S[121]
Set T and X [26] to 199 xor 252
Set T and X [26] to 59

Iterasi 27 : K=27, T=59
Set T and X [K] to x [K] xor S[T]
Set T and X [27] to x [27] xor S[59]
Set T and X [27] to x [27] xor S[59]
Set T and X [27] to 236 xor 92
Set T and X [27] to 176

Iterasi 28 : K=28, T=176
Set T and X [K] to x [K] xor S[T]
Set T and X [28] to x [28] xor S[176]
Set T and X [28] to x [28] xor S[176]
Set T and X [28] to 12 xor 44
Set T and X [28] to 32

Iterasi 29 : K=29, T=32
Set T and X [K] to x [K] xor S[T]
Set T and X [29] to x [29] xor S[32]
Set T and X [29] to x [29] xor S[32]
Set T and X [29] to 13 xor 30
Set T and X [29] to 19

Iterasi 30 : K=30, T=19
Set T and X [K] to x [K] xor S[T]
Set T and X [30] to x [30] xor S[19]
Set T and X [30] to x [30] xor S[19]
Set T and X [30] to 14 xor 243
Set T and X [30] to 253

Iterasi 31 : K=31, T=253
Set T and X [K] to x [K] xor S[T]
Set T and X [31] to x [31] xor S[253]
Set T and X [31] to x [31] xor S[253]
Set T and X [31] to 15 xor 17
Set T and X [31] to 30

Iterasi 32 : K=32, T=30
Set T and X [K] to x [K] xor S[T]
Set T and X [32] to x [32] xor S[30]
Set T and X [32] to x [32] xor S[30]

Iterasi 33 : K=33, T=167
Set T and X [K] to x [K] xor S[T]
Set T and X [33] to x [33] xor S[167]
Set T and X [33] to x [33] xor S[167]

Set T and X [32] to 37 xor 130
Set T and X [32] to 167

Iterasi 34 : K=34, T=106
Set T and X [K] to x [K] xor S[T]
Set T and X [34] to x [34] xor S[106]
Set T and X [34] to x [34] xor S[106]
Set T and X [34] to 68 xor 146
Set T and X [34] to 214

Iterasi 36 : K=36, T=174
Set T and X [K] to x [K] xor S[T]
Set T and X [36] to x [36] xor S[174]
Set T and X [36] to x [36] xor S[174]
Set T and X [36] to 45 xor 234
Set T and X [36] to 199

Iterasi 38 : K=38, T=203
Set T and X [K] to x [K] xor S[T]
Set T and X [38] to x [38] xor S[203]
Set T and X [38] to x [38] xor S[203]
Set T and X [38] to 46 xor 8
Set T and X [38] to 38

Iterasi 40 : K=40, T=210
Set T and X [K] to x [K] xor S[T]
Set T and X [40] to x [40] xor S[210]
Set T and X [40] to x [40] xor S[210]
Set T and X [40] to 10 xor 149
Set T and X [40] to 159

Iterasi 42 : K=42, T=42
Set T and X [K] to x [K] xor S[T]
Set T and X [42] to x [42] xor S[42]
Set T and X [42] to x [42] xor S[42]
Set T and X [42] to 199 xor 111
Set T and X [42] to 168

Iterasi 44 : K=44, T=47
Set T and X [K] to x [K] xor S[T]
Set T and X [44] to x [44] xor S[47]
Set T and X [44] to x [44] xor S[47]
Set T and X [44] to 12 xor 18
Set T and X [44] to 30

Iterasi 46 : K=46, T=143
Set T and X [K] to x [K] xor S[T]
Set T and X [46] to x [46] xor S[143]
Set T and X [46] to x [46] xor S[143]
Set T and X [46] to 14 xor 2
Set T and X [46] to 12

Set T and X [33] to 80 xor 58
Set T and X [33] to 106

Iterasi 35 : K=35, T=214
Set T and X [K] to x [K] xor S[T]
Set T and X [35] to x [35] xor S[214]
Set T and X [35] to x [35] xor S[214]
Set T and X [35] to 70 xor 232
Set T and X [35] to 174

Iterasi 37 : K=37, T=199
Set T and X [K] to x [K] xor S[T]
Set T and X [37] to x [37] xor S[199]
Set T and X [37] to x [37] xor S[199]
Set T and X [37] to 49 xor 250
Set T and X [37] to 203

Iterasi 39 : K=39, T=38
Set T and X [K] to x [K] xor S[T]
Set T and X [39] to x [39] xor S[38]
Set T and X [39] to x [39] xor S[38]
Set T and X [39] to 50 xor 224
Set T and X [39] to 210

Iterasi 41 : K=41, T=159
Set T and X [K] to x [K] xor S[T]
Set T and X [41] to x [41] xor S[159]
Set T and X [41] to x [41] xor S[159]
Set T and X [41] to 37 xor 15
Set T and X [41] to 42

Iterasi 43 : K=43, T=168
Set T and X [K] to x [K] xor S[T]
Set T and X [43] to x [43] xor S[168]
Set T and X [43] to x [43] xor S[168]
Set T and X [43] to 236 xor 195
Set T and X [43] to 47

Iterasi 45 : K=45, T=30
Set T and X [K] to x [K] xor S[T]
Set T and X [45] to x [45] xor S[30]
Set T and X [45] to x [45] xor S[30]
Set T and X [45] to 13 xor 130
Set T and X [45] to 143

Iterasi 47 : K=47, T=12
Set T and X [K] to x [K] xor S[T]
Set T and X [47] to x [47] xor S[12]
Set T and X [47] to x [47] xor S[12]
Set T and X [47] to 15 xor 236
Set T and X [47] to 227

Tabel 7. *Output*

X0	X1	X2	X3	X4	X5	X6	X7
41	66	121	252	159	15	19	243
X8	X9	X10	X11	X12	X13	X14	X15
180	133	113	184	211	139	101	165
X16	X17	X18	X19	X20	X21	X22	X23
116	198	239	255	58	202	85	162
X24	X25	X26	X27	X28	X29	X30	X31
169	121	59	176	32	19	253	30

X32	X33	X34	X35	X36	X37	X38	X39
167	106	214	174	199	203	38	210
X40	X41	X42	X43	X44	X45	X46	X47
159	42	168	47	30	143	12	227

Message Digest dihasilkan dari *output* X0-X15 sehingga hasilnya adalah 41 66 121 252 159 15 19 243 180 133 113 184 211 139 101 165 jika di konversi menjadi bilangan heksadesimal maka hasilnya adalah sebagai berikut : 294279FC9F0F13F3B48571B8D38B65A5 nilai inilah yang menjadi identitas *file pdf* yang merepresentasikan isi dari *file pdf* sehingga jika ada *file pdf* yang memiliki nilai *hash* sama maka *file pdf* tersebut ganda atau duplikat.

4. KESIMPULAN

Dari hasil analisa dan penulisan 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 Proses pencarian file yang duplikat atau ganda sebagian besar menggunakan atau memanfaatkan algoritma fungsi hash. Algoritma fungsi hash berguna sebagai enkripsi satu arah dan menghasilkan identitas dari sebuah file Pdf dan setiap file Pdf menghasilkan nilai hash yang sama. Pencarian jenis ini akan mencari file dengan nama, ekstensi dan ukuran dengan panjang checksum 128 bit. Proses Algoritma MD2 dihitung secara infisibel untuk mencari *string* yang sesuai untuk menghasilkan nilai hash atau juga digunakan untuk mencari dua string yang berbeda yang akan menghasilkan nilai hash yang sama. Penerapan algoritma MD2 pada aplikasi Duplicate Pdf Scanner adalah untuk membangkitkan identitas setiap *file Pdf* yang ada pada sebuah media penyimpanan dengan melakukan pencarian *file Pdf* yang berekstensi Pdf, selanjutnya membangkitkan nilai *hash* dari setiap *file* Pdf tersebut, setelah setiap *file* pdf tersebut memiliki nilai *hash* masing-masing maka *file* pdf tersebut akan di urutkan berdasarkan nilai *hash*nya dan pengguna dapat menghapus *file* pdf yang memiliki nilai hash yang sama karena *file* pdf tersebut merupakan *file* pdf yang sama atau duplikat

REFERENCES

[1] R. Sadikin, Kriptografi Untuk Kemanan Jaringan dan Implementasinya Dalam Bahasa Java, Yogyakarta: Andi, 2012.
[2] R. Munir, Kriptografi, Bandung: R. Munir, 2006.
[3] Y. Kurniawan, Kriptografi Keamanan Internet dan Jaringan Komunikasi, Bandung: Informatika, 2017.
[4] R. Munir, Kriptografi, Bandung: Informatika Bandung, 2006.
[5] F. I. P. Standards Publications, "Secure Hash Standard," National Institute of Standards and Technology, Amerika Serikat, 2002.
[6] S. Patil, N. Jagtap, S. Rajput and R. Sangore, "A Duplicate File Finder System," International Journal of Science Spirituality Business and Technology, pp. 10-14, 2017.
[7] D. Irwanto, Perancangan Object Oriented Software Dengan UML, Yogyakarta: Andi, 2007.
[8] Ketut Darmayuda, Pemograman Aplikasi Database dengan Microsoft Visual Basic. NET 2008, Bandung, 2010.
[9] C. Solution and S. Community, Membangun Aplikasi Database Dengan Visual Basic 2008 dan SQL Server 2008, Jakarta: PT. Elex Media Komputindo, 2010.