STUDI DAN IMPLEMENTASI ALGORITMA MAC BERBASIS FUNGSI HASH SATU ARAH DAN BERBASIS CIPHER BLOK

Raden Fitri Indriani – NIM : 13503020

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13020@students.if.itb.ac.id

Abstrak

MAC adalah fungsi *hash* satu arah yang menggunakan kunci rahasia. MAC digunakan untuk otentikasi pesan tanpa mengenkripsi pesan. Algoritma MAC dapar dirancang dengan dua pendekatan, yaitu :

1. algoritma MAC berbasis cipher blok
2. algoritma MAC berbasis fungsi *hash* satu arah.

Makalah ini membahas mengenai algoritma MAC berbasis fungsi *hash* satu arah dan algoritma MAC berbasis cipher blok. Untuk algoritma MAC berbasis fungsi *hash* satu arah, fungsi *hash* yang digunakan adalah SHA1. Sedangkan untuk algoritma MAC berbasis cipher blok, algoritma cipher blok yang digunakan adalah algoritma cipher blok yang berasal dari tugas 2 mata kuliah Kriptografi yang telah dibuat sebelumnya. Pembahasan ini mencakup teori dan implementasi. Dari hasil implementasi dan pembahasan teori akan didapatkan suatu kesimpulan dengan membandingkan MAC berbasis fungsi *hash* satu arah menggunakan SHA1 dan MAC berbasis cipher blok.

Kata kunci : MAC, fungsi *hash*, cipher blok

1. Dasar Teori

A. Algoritma Cipher Blok

Algoritma cipher blok tergolong ke dalam algoritma simetri. Algoritma ini melakukan operasi terhadap blok-blok bit yang panjangnya tetap. Plainteks akan dibagi menjadi n buah blok bit dengan panjang yang sama. Sebagai contoh, ketika melakukan enkripsi pada suatu blok plainteks yang panjangnya 128 bit, maka akan menghasilkan suatu blok cipherteks sepanjang 128 bit juga. Begitu juga halnya ketika melakukan proses dekripsi pada suatu blok cipherteks berukuran 128 bit, akan menghasilkan suatu blok plainteks sepanjang 128 bit. Ukuran blok-blok yang digunakan untuk melakukan proses dekripsi/enkripsi sama dengan ukuran kunci yang digunakan.

Algoritma cipher blok terdiri dari dua buah algoritma yang berpasangan. Satu digunakan untuk melakukan proses enkripsi, E. Sedangkan satu lagi digunakan untuk melakukan proses dekripsi, D. Dimana :

$$D=E^{-1}$$

Dikarenakan fungsi D merupakan fungsi invers dari E, maka algoritma cipher blok dapat dirumuskan sebagai berikut.

$$E_k^{-1}(E_k(P)) = P$$

P = sebuah blok plainteks
k = kunci

Kedua buah algortima tersebut menerima input berupa blok input plainteks/cipherteks berukuran n bit dan sebuah kunci berukuran k bit, serta akan menghasilkan blok keluaran sepanjang n bit.

Pada proses enkripsi, algoritma cipher blok akan menerima sebuah blok plainteks berukuran n bit dan sebuah kunci berukuran k bit. Proses pertama yang dilakukan adalah mengecek apakah n merupakan kelipatan dari k, atau dengan kata lain n mod k = 0. Jika n mod k <> 0, maka akan dilakukan penambahan bit yang dinamakan dengan *padding bits*. *Padding bits* akan ditambahkan sejumlah (k - (n mod k)) bit ke dalam

blok plainteks. Akan tetapi jika n mod k = 0, maka penambahan *padding bits* tidak perlu dilakukan.

Pemrosesan blok plainteks tersebut selain bergantung pada algoritma yang digunakan, juga bergantung pada mode operasi yang digunakan. Setiap mode operasi yang ada memiliki karakteristik tersendiri untuk menghindari serangan-serangan yang mungkin terjadi. Adapun mode-mode tersebut adalah sebagai berikut.

a) Electronic Code Book (ECB)
Mode operasi ini merupakan mode yang paling sederhana dimana plainteks dibagi menjadi beberapa blok yang berukuran sama dengan kunci yang digunakan dan setiap blok dienkripsi secara terpisah. Mode ini mempunyai kelemahan yaitu sebuah *plainteks* akan dienkripsikan menjadi sebuah *cipherteks* yang identik. Dengan demikian, mode EBC ini tidak dapat menyembunyikan pola dari data. Hal ini mengakibatkan mode EBC tidak mendukung *message confidentiality*. Oleh karena itulah mode ini tidak disarankan untuk digunakan sebagai protokol kriptografi.

b) Cipher Block Chaining (CBC)
Mode CBC hampir sama dengan ECB. Hanya saja pada mode ini setiap blok bergantung dengan blok-blok (blok sebelum dan blok sesudah blok yang bersangkutan) yang berdekatan dengannya. Hal ini dikarenakan setiap blok plainteks dilakukan operasi xor dengan blok cipherteks sebelumnya. Pada blok pertama dilakukan operasi xor dengan *Initialitation Vector* (IV) sebagai penggantin blok sebelumnya. Dengan adanya *Initialitation Vector* (IV) maka pesan menjadi unik. Walaupun demikian CBC mempunyai kelemahan yaitu jika terjadi kesalahan pada satu buah bit dalam sebuah blok plainteks pada proses enkripsi, maka blok-blok cipherteks berikutnya dan blok cipherteks dari plainteks yang bersangkutan akan mengalami kesalahan pula. Akan tetapi jika kesalahan terjadi pada sebuah bit dalam sebuah cipherteks pada proses dekripsi, maka hanya blok planteks yang bersangkutan dan sebuah blok plainteks berikutnya saja yang terkena dampaknya

c) Cipher Feedback (CFB)
Mode CFB memperlakukan blok plainteks seolah seperti cipher aliran yang dilakukan dengan cara blok plainteks dienkripsikan menjadi ukuran yang lebih kecil dari blok seperti 2 bit, 8 bit, dan seterusnya. Kelemahan dari mode CFB adalah jika terdapat kesalahan pada satu buah bit yang dienkripsi, maka bit-bit selanjutnya juga akan salah. Hal ini juga berlaku untuk proses dekripsi.

d) Propagating Cipher Block Chaining (CPBC)
Mode CPBC didesain untuk mengatasi kekurangan yang terdapat di dalam CBC. Akan tetapi mode ini sangat jarang sekali digunakan. Mode ini hanya digunakan pada Kerberos dan WASTE.

e) Output Feedback (OFB)
Perlakuan blok input pada mode ini mirip dengan perlakuan blok input pada mode CFB. Hanya saja pada OFB yang dimasukkan ke dalam antrian adalah elemen antrian terdepan yang telah dikenakan fungsi enkripsi. Sedangkan pada mode CFB, yang dimasukkan ke dalam antrian adalah hasil enkripsi (cipherteks).

B. Fungsi *Hash*

Fungsi *hash* adalah suatu fungsi yang menerima masukan suatu string sembarang dan menghasilkan string keluaran yang panjangnya tetap yang pada umumnya string keluaran ini berukuran lebih kecil daripada ukuran string masukan. String keluaran fungsi *hash* ini dapat juga disebut sebagai nilai *hash* atau *message digest*. Fungsi *hash* ini biasanya digunakan untuk memverifikasi suatu dokumen atau arsip dengan arsip asli.

Secara umum, persamaan fungsi *hash* dapat dirumuskan sebagai berikut.

$$h = H(P)$$

keterangan :
h : nilai *hash* atau *message digest*
H : fungsi *Hash*
P : string / pesan masukan

Fungsi *hash* satu arah atau *One Way Hash* adalah fungsi *hash* yang bekerja secara satu arah yaitu suatu pesan yang sudah diubah menjadi nilai *hash* atau *message digest* tidak dapat dikembalikan lagi menjadi pesan semula. Secara umum, dua buah pesan akan menghasilkan *message digest* yang berbeda. Walaupun tidak menutup kemungkinan bahwa akan ada dua buah pesan yang dapat menghasilkan *message digest* yang sama. Akan tetapi kemungkinan ini sangatlah kecil sekali (secara komputasi).

Sifat-sifat fungsi *hash* satu arah atau *One Way Hash* adalah sebagai berikut [MUN06] :

1) Fungsi H dapat diterapkan pada blok data berukuran apa saja.
2) H menghasilkan nilai *hash* (h) dengan panjang yang tetap.
3) H(x) mudah dihitung untuk setiap nilai x yang diberikan.
4) Untuk setiap nilai h yang diberikan, tidak mungkin menemukan x sedemikian sehingga H(x) = h. Itulah sebabnya fungsi H dikatakan fungsi *Hash* satu arah atau *One Way Hash.*
5) Untuk setiap x yang diberikan, tidak mungkin mencari y ≠ x sedemikian sehingga H(y) = H(x).
6) Tidak mungkin (secara komputasi) mencari pasangan x dan y sedemikian sehingga H(x) = H(y).

Sebuah fungsi *Hash* dianggap tidak aman jika [MUN06] :

1) Secara komputasi dimungkinkan menemukan pesan yang bersesuaian dengan pesan ringkasnya (nilai *hash*-nya atau *message digest*), dan
2) Terjadi kolisi (*collision*), yaitu terdapat beberapa pesan berbeda yang mempunyai pesan ringkas yang sama.

C. SHA-1

Secure Hash Algorithm (SHA), dibuat oleh NIST bersamaan dengan NSA, yang akan digunakan secara berbarengan dengan Digital Signature Standard (DSS) untuk menspesifikasikan Secure Hash Standard (SHS). SHA-1 merupakan revisi dari SHA-0 yang dipublikasikan pada tahun 1994.

SHA merupakan suatu fungsi *hash* yang mirip dengan MD4 yang dibuat oleh Rivest. Perbedaannya terletak pada SHA menambahkan operasi ekspansi tambahan, putaran ekstra, dan seluruh transformasi dirancang untuk mendukung ukuran blok DSS demi efisiensi.

SHA mengambil input pesan yang panjangnya kurang dari 2^{64} bit dan menghasilkan nilai *hash* atau *message digest* yang panjangnya 160 bit yang dirancang sedemikian rupa sehingga akan sangat susah secara komputasi untuk mendapatkan dua buah pesan yang akan menghasilkan nilai *hash* yang sama. Sebagai contoh, jika kita mempunyai sebuah dokumen A, h(A), akan sangat sulit menemukan dokumen B dengan nilai *hash* yang sama. Akan sangat sulit juga bagi kita untuk membuat dokumen B sedemikian sehingga akan menghasilkan nilai *hash* yang sama dengan dokumen A tersebut. Oleh karena itulah SHA-1 dianggap aman. Perubahan sekecil apapun pada pesan masukan akan menghasilkan *message digest* yang berbeda dengan probabilitas yang sangat besar.

SHA-1 dapat digunakan bersamaan dengan DSA dalam surat elektronik (*e-mail*), transfer uang secara elektronikm distribusi perangkat lunak, penyimpanan data, dan aplikasi lainnya yang membutuhkan jaminan data integritas dan autentikasi keaslian data. SHA-1 juga dapat digunakan kapanpun ketika dibutuhkan untuk menghasilkan *condesed version* dari suatu pesan. [ITL06]

SHA-1 dapat diimpelementasikan pada perangkat lunak (*software*), perangkat keras (*hardware*), *firmware*, maupun kombinasi diantara ketiganya.[ITL06]

Langkah-langkah pembuatan *message digest* dengan algoritma SHA-1 adalah sebagai berikut [MUN06].

1) Penambahan bit-bit pengganjal (*padding bits*).
   Pesan ditambah dengan bit-bit pengganjal sedemikian rupa sehingga panjang pesan (dalam satuan bit) kongruen dengan 448 modulo 512. Ini berarti panjang pesan setelah ditambah dengan bit-bit pengganjal adalah 64 bit kurang dari kelipatan 512. Angka 512 ini muncul karena SHA-1 melakukan pemrosesan pesan dalam blok-blok berukuran 512 bit. Jika ada pesan berukuran 448 bit pun akan tetap ditambahi bit-bit pengganjal. Pesan dengan panjang 448 bit tersebut akan ditambahi bit-bit pengganjal sepanjang 512 bit sehingga panjangnya menjadi 960 bit. Dengan demikian, panjang bit-bit pengganjal adalah antara 1-512 bit. Bit-bit pengganjal terdiri dari satu buah bit 1 diikuti dengan sisanya bit 0.

2) Penambahan nilai panjang pesan semula.
   Pesan yang telah ditambah dengan bit-bit pengganjal selanjutnya ditambah lagi dengan 64 bit yang menyatakan panjang pesan semula. Setelah ditambah dengan 64 bit ini, maka panjang pesan sekarang menjadi kelipatan 512 bit.

3) Inisialisasi penyangga (*buffer*) MD.
   SHA membutuhkan 5 buah penyangga (*buffer)* yang masing-masing panjangnya 32 bit. Dengan demikian total panjang penyangga adalah $5 \times 32 = 160$ bit. Kelima buah penyangga ini akan menampung hasil akhir dan hasil antara. Kelima buah penyangga tersebut diberi nama A, B, C, D, dan E. Setiap penyangga diinisialisasi dengan nilai-nilai sebagai berikut :

   A = 67452301
   B = EFCDAB89
   C = 98BADCFE
   D = 10325476
   E = C3D2E1F0

4) Pengolahan pesan dalam blok berukuran 512 bit.
   Pesan dibagi menjadi L buah blok yang masing-masing panjangnya 512 bit (Y_0 sampai Y_{L-1}). Setiap blok 512-bit tersebut diproses bersama dengan penyangga MD menjadi keluaran 128-bit. Proses ini dinamakan proses H_{SHA-1}.

   Proses H_{SHA-1} terdiri dari 80 buah putaran, dan masing-masing putaran menggunakan bilangan penambah K_t, yaitu :

   Putaran $0 \le t \le 19$: K_t=5A827999

   Putaran $20 \le t \le 39$: K_t=6ED9EBA1

   Putaran $40 \le t \le 59$: K_t=8F1BBCDC

   Putaran $60 \le t \le 79$: K_t=CA62C1D6

   Fungsi lojik untuk setiap putaran f_0, f_1, ..., f_{79} (setiap f_t, $0 \le t \le 79$, mengoperasikan 32 bit words B, C, D dan menghasilkan 32 bit words sebagai keluarannya)yang digunakan dalam SHA-1 adalah sebagai berikut. [ITL06]

   f_t(B,C,D) = (B AND C) OR ((NOT B) AND D) (0 <= t <= 19)

   f_t(B,C,D) = B XOR C XOR D (20 <= t <= 39)

   f_t(B,C,D) = (B AND C) OR (B AND D) OR (C AND D) (40 <= t <= 59)

   f_t(B,C,D) = B XOR C XOR D (60 <= t <= 79).

Berikut ini adalah sebuah contoh sebuah pesan dan *message digest* yang dihasilkan dari proses SHA-1. [ITL06]

Pesan masukan : “ABC”
Binary code dari pesan tersebut adalah :
01100001 01100010 01100011
Pesan ini mempunyai panjang 24 bit.

Proses pertama yang dilakukan adalah penambahan bit-bit pengganjal yaitu, menambahkan 1 buah bit 1 dan 423 buah bit 0.

Setelah penambahan bit-bit pengganjal, proses selanjutnya adalah menambahkan panjang pesan semula. Dalam kasus ini, ditambahkan 00000000 00000018, yang merupakan representasi dari angka 24 dengan panjang 64 bit. Dengan demikian, pesan hanya terdiri dari 1 buah blok 512 bit. Maka n = 1.

Kemudian dilakukan inisialiasi penyangga, yaitu :
H_0=67452301
H_1=EFCDAB89
H_2=98BADCFE
H_3=10325476
H_4=C3D2E1F0

Setelah itu blok dibagi menjadi 16 buah sub-blok yang masing-masing berukuran 32 bit (1 words). Blok-blok dalam words tersebut adalah sebagai berikut.
W[0]=61626380
W[1]=00000000
W[2]=00000000
W[3]=00000000
W[4]=00000000
W[5]=00000000
W[6]=00000000
W[7]=00000000
W[8]=00000000
W[9]=00000000
W[10]=00000000
W[11]=00000000
W[12]=00000000
W[13]=00000000
W[14]=00000000
W[15] = 00000018

Nilai heksa dari A, B, C, D, dan E setelah melalui 80 putaran yaitu :

	A	B	C	D	E
t = 0:	0116FC33	67452301	7BF36AE2	98BADCFE	10325476
t = 1:	8990536D	0116FC33	59D148C0	7BF36AE2	98BADCFE
t = 2:	A1390F08	8990536D	C045BF0C	59D148C0	7BF36AE2
t = 3:	CDD8E11B	A1390F08	626414DB	C045BF0C	59D148C0
t = 4:	CFD499DE	CDD8E11B	284E43C2	626414DB	C045BF0C
t = 5:	3FC7CA40	CFD499DE	F3763846	284E43C2	626414DB
t = 6:	993E30C1	3FC7CA40	B3F52677	F3763846	284E43C2
t = 7:	9E8C07D4	993E30C1	0FF1F290	B3F52677	F3763846
t = 8:	4B6AE328	9E8C07D4	664F8C30	0FF1F290	B3F52677
t = 9:	8351F929	4B6AE328	27A301F5	664F8C30	0FF1F290
t = 10:	FBDA9E89	8351F929	12DAB8CA	27A301F5	664F8C30
t = 11:	63188FE4	FBDA9E89	60D47E4A	12DAB8CA	27A301F5
t = 12:	4607B664	63188FE4	7EF6A7A2	60D47E4A	12DAB8CA
t = 13:	9128F695	4607B664	18C623F9	7EF6A7A2	60D47E4A
t = 14:	196BEE77	9128F695	1181ED99	18C623F9	7EF6A7A2
t = 15:	20BDD62F	196BEE77	644A3DA5	1181ED99	18C623F9
t = 16:	4E925823	20BDD62F	C65AFB9D	644A3DA5	1181ED99
t = 17:	82AA6728	4E925823	C82F758B	C65AFB9D	644A3DA5
t = 18:	DC64901D	82AA6728	D3A49608	C82F758B	C65AFB9D
t = 19:	FD9E1D7D	DC64901D	20AA99CA	D3A49608	C82F758B
t = 20:	1A37B0CA	FD9E1D7D	77192407	20AA99CA	D3A49608
t = 21:	33A23BFC	1A37B0CA	7F67875F	77192407	20AA99CA
t = 22:	21283486	33A23BFC	868DEC32	7F67875F	77192407
t = 23:	D541F12D	21283486	0CE88EFF	868DEC32	7F67875F
t = 24:	C7567DC6	D541F12D	884A0D21	0CE88EFF	868DEC32
t = 25:	48413BA4	C7567DC6	75507C4B	884A0D21	0CE88EFF
t = 26:	BE35FBD5	48413BA4	B1D59F71	75507C4B	884A0D21
t = 27:	4AA84D97	BE35FBD5	12104EE9	B1D59F71	75507C4B
t = 28:	8370B52E	4AA84D97	6F8D7EF5	12104EE9	B1D59F71
t = 29:	C5FBAF5D	8370B52E	D2AA1365	6F8D7EF5	12104EE9

	A	B	C	D	E
t = 30:	1267B407	C5FBAF5D	A0DC2D4B	D2AA1365	6F8D7EF5
t = 31:	3B845D33	1267B407	717EEBD7	A0DC2D4B	D2AA1365
t = 32:	046FAA0A	3B845D33	C499ED01	717EEBD7	A0DC2D4B
t = 33:	2C0EBC11	046FAA0A	CEE1174C	C499ED01	717EEBD7
t = 34:	21796AD4	2C0EBC11	811BEA82	CEE1174C	C499ED01
t = 35:	DCBBB0CB	21796AD4	4B03AF04	811BEA82	CEE1174C
t = 36:	0F511FD8	DCBBB0CB	085E5AB5	4B03AF04	811BEA82
t = 37:	DC63973F	0F511FD8	F72EEC32	085E5AB5	4B03AF04
t = 38:	4C986405	DC63973F	03D447F6	F72EEC32	085E5AB5
t = 39:	32DE1CBA	4C986405	F718E5CF	03D447F6	F72EEC32
t = 40:	FC87DEDF	32DE1CBA	53261901	F718E5CF	03D447F6
t = 41:	970A0D5C	FC87DEDF	8CB7872E	53261901	F718E5CF
t = 42:	7F193DC5	970A0D5C	FF21F7B7	8CB7872E	53261901
t = 43:	EE1B1AAF	7F193DC5	25C28357	FF21F7B7	8CB7872E
t = 44:	40F28E09	EE1B1AAF	5FC64F71	25C28357	FF21F7B7
t = 45:	1C51E1F2	40F28E09	FB86C6AB	5FC64F71	25C28357
t = 46:	A01B846C	1C51E1F2	503CA382	FB86C6AB	5FC64F71
t = 47:	BEAD02CA	A01B846C	8714787C	503CA382	FB86C6AB
t = 48:	BAF39337	BEAD02CA	2806E11B	8714787C	503CA382
t = 49:	120731C5	BAF39337	AFAB40B2	2806E11B	8714787C
t = 50:	641DB2CE	120731C5	EEBCE4CD	AFAB40B2	2806E11B
t = 51:	3847AD66	641DB2CE	4481CC71	EEBCE4CD	AFAB40B2
t = 52:	E490436D	3847AD66	99076CB3	4481CC71	EEBCE4CD
t = 53:	27E9F1D8	E490436D	8E11EB59	99076CB3	4481CC71
t = 54:	7B71F76D	27E9F1D8	792410DB	8E11EB59	99076CB3
t = 55:	5E6456AF	7B71F76D	09FA7C76	792410DB	8E11EB59
t = 56:	C846093F	5E6456AF	5EDC7DDB	09FA7C76	792410DB
t = 57:	D262FF50	C846093F	D79915AB	5EDC7DDB	09FA7C76
t = 58:	09D785FD	D262FF50	F211824F	D79915AB	5EDC7DDB
t = 59:	3F52DE5A	09D785FD	3498BFD4	F211824F	D79915AB
t = 60:	D756C147	3F52DE5A	4275E17F	3498BFD4	F211824F
t = 61:	548C9CB2	D756C147	8FD4B796	4275E17F	3498BFD4
t = 62:	B66C020B	548C9CB2	F5D5B051	8FD4B796	4275E17F
t = 63:	6B61C9E1	B66C020B	9523272C	F5D5B051	8FD4B796
t = 64:	19DFA7AC	6B61C9E1	ED9B0082	9523272C	F5D5B051
t = 65:	101655F9	19DFA7AC	5AD87278	ED9B0082	9523272C
t = 66:	0C3DF2B4	101655F9	0677E9EB	5AD87278	ED9B0082
t = 67:	78DD4D2B	0C3DF2B4	4405957E	0677E9EB	5AD87278
t = 68:	497093C0	78DD4D2B	030F7CAD	4405957E	0677E9EB
t = 69:	3F2588C2	497093C0	DE37534A	030F7CAD	4405957E
t = 70:	C199F8C7	3F2588C2	125C24F0	DE37534A	030F7CAD
t = 71:	39859DE7	C199F8C7	8FC96230	125C24F0	DE37534A
t = 72:	EDB42DE4	39859DE7	F0667E31	8FC96230	125C24F0
t = 73:	11793F6F	EDB42DE4	CE616779	F0667E31	8FC96230
t = 74:	5EE76897	11793F6F	3B6D0B79	CE616779	F0667E31
t = 75:	63F7DAB7	5EE76897	C45E4FDB	3B6D0B79	CE616779
t = 76:	A079B7D9	63F7DAB7	D7B9DA25	C45E4FDB	3B6D0B79
t = 77:	860D21CC	A079B7D9	D8FDF6AD	D7B9DA25	C45E4FDB
t = 78:	5738D5E1	860D21CC	681E6DF6	D8FDF6AD	D7B9DA25
t = 79:	42541B35	5738D5E1	21834873	681E6DF6	D8FDF6AD

Dengan demikian, setelah blok 1 selesai diproses, maka nilai dari H_i adalah :

H_0=67452301+42541B35
=A9993E36

H_1=EFCDAB89+5738D5E1
=4706816A

H_2=98BADCFE+21834873
=BA3E2571

H_3=10325476+681E6DF6
=7850C26C

H_4=C3D2E1F0+D8FDF6AD
= 9CD0D89D

Maka, *message digest* dari pesan "ABC" ini adalah A9993E36 4706816ABA3E25717850C26C9CD 0D89D.

D. MAC

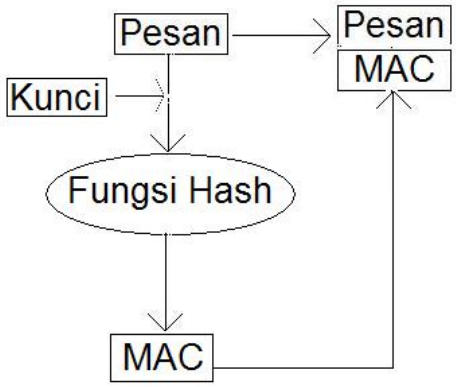
Hasil dari fungsi *hash* yang dikombinasikan dengan pesan dan kunci disebut dengan Message Autentication Code (MAC). MAC merupakan *fingerprint* atau *message digest* dari pesan masukan yang dikombinasikan dengan sebuah kunci. [LIN06]

Secara matematis, MAC dinyatakan sebagai [MUN06]

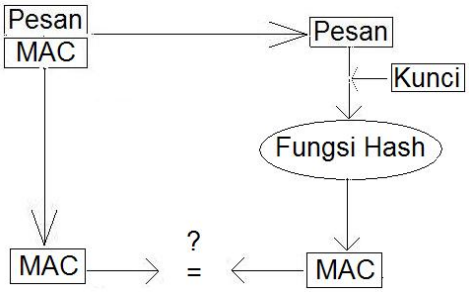
$$MAC = C_K(M)$$

dengan,
MAC = nilai *hash*,
C = fungsi *hash* atau algoritma MAC,
K = kunci rahasia.
Fungsi C memampatkan pesan M yang berukuran sembarang dengan menggunakan kunci K.

MAC digunakan untuk melakukan otentikasi pesan tanpa perlu merahasiakan pesan tersebut (tanpa mengenkripsi pesan yang bersangkutan). Mula-mula pengirim pesan akan menghitung MAC dari pesan yang hendak dikirimkan dengan kunci rahasia K (asumsi bahwa pengirim dan penerima pesan telah berbagi kunci rahasia). Kemudian MAC dilekatkan (*embedded*) pada pesan. Selanjutnya pesan dikirim bersama-sama dengan MAC kepada penerima. Penerima kemudian menggunakan kunci yang sama untuk menghitung MAC ke pesan dan membandingkannya dengan MAC yang diterimannya. Jika kedua MAC tersebut sama, maka penerima dapat menyimpulkan bahwa pesan dikirim oleh orang yang sesungguhnya dan isi pesan tidak diubah oleh pihak ketiga selama transmisi. Jika pesan tidak dikirim oleh pengirim yang asli, maka MAC yang dihitung oleh penerima akan berbeda dengan MAC yang diterima sebab pihak ketiga tidak mengetahui kunci rahasia. Begitu juga jika pesan sudah diubah selama transmisi, maka MAC yang dihitung oleh penerima tidak sama dengan MAC yang diterima.



Gambar 1-D-1 Pemberian MAC pada pesan yang akan dikirim oleh pengirim pesan



Gambar 1-D-2 verifikasi MAC yang oleh penerima pesan

Algoritma MAC dapat dirancang dengan dua pendekatan, yaitu : [MUN06]

1) Algoritma MAC berbasis cipher blok (*Block Cipher*)
   MAC dapat dibangkitkan dengan menggunakan algoritma cipher blok dengan mode CBC atau CFB. Nilai *hash* yang akan menjadi MAC adalah hasil enkripsi blok terakhir.

2) Algoritma MAC berbasis fungsi *hash* satu arah
   Fungsi *hash* satu arah seperti MD5 dapat digunakan sebagai MAC. Yang akan menjadi MAC adalah nilai *hash* dari pesan M ditambah dengan kunci K.
   MAC = H(M,K)

2. Implementasi

A. Algoritma Cipher Blok

Algoritma cipher blok yang digunakan untuk membangkitkan MAC kali ini merupakan algoritma

cipher blok sederhana. Ukuran setiap blok yang digunakan adalah sama dengan ukuran kunci. Algoritma ini hanya membangkitkan kunci internal untuk setiap blok, kemudian dilakukan operasi XOR antara kunci internal yang dibangkitkan dan blok yang bersangkutan. Setelah proses tersebut selesai untuk semua blok, kemudia dilakukan transposisi blok. Mode yang digunakan kali ini adalah mode CBC.

Untuk melakukan enkripsi, hal pertama yang dilakukan adalah mengubah pesan ke dalam struktur array bit agar lebih mudah diproses. Kemudian menambahkan bit-bit pengganjal jika dibutuhkan. Bit-bit pengganjal yang diperlukan pasti merupakan kelipatan dari 8 bit karena 1 karakter mempunyai panjang 8 bit. Oleh karena itu, jika dibutuhkan bit-bit pengganjal maka akan terdiri dari minimal 1 buah karakter “Q” dan beberapa buah (atau tidak sama sekali) karakter “W”. Karakter-karakter ini dipilih dengan alasan sangat kecil sekali kemungkinan suatu pesan yang mempunyai arti akan memiliki karakter “QW” pada akhir pesan.

Adapun algoritma untuk melakukan penambahan bit-bit pengganjal tersebut adalah sebagai berikut.

```
private int addPadding()
{
 int si,ccd,i,mo,g;
 BitArray tm;
 //penambahan bit pengganjal jika diperlukan
 if (buffP.Length % buffK.Length == 0)
 {
   si = buffP.Length / buffK.Length;
   buffPad = new BitArray(buffP);
 }
 else
  {
    si = (buffP.Length / buffK.Length) + 1;
    buffPad = new BitArray(buffP.Length + (buffK.Length -
     (buffP.Length % buffK.Length)));
     for (i = 0; i < buffP.Length; i++)
     {
      buffPad[i] = buffP[i];
     }
     ccd = i;
     mo = buffK.Length - (buffP.Length % buffK.Length);
     if (mo == 8)
      {
       byte[] qbyte = new byte[] { Convert.ToByte('Q') };
       tm = new BitArray(qbyte);
       g = 0;
       i = 0;
       while (i < mo)
       {
        buffPad[ccd] = tm[g];
        g++;
        ccd++;
        i++;
        }
      }
      else
      {
       string st = "QW";
       int xx = 16;
```

```
       while (xx <= mo)
        {
        st = st + Convert.ToString('W');
        xx = xx + 8;

        }
         byte[] bet = new byte[xx / 8];
         char[] ch = st.ToCharArray();
         for (i = 0; i < bet.Length; i++)
         {
          bet[i] = Convert.ToByte(ch[i]);
         }
         tm = new BitArray(bet);
         g = 0;
         i = 0;
         while (i < mo)
         {
          buffPad[ccd] = tm[g];
          g++;
          i++;
          ccd++;
         }
      }
  }
  return (si);
}
```

Setelah penambahan bit-bit pengganjal selesai dilakukan, selanjutnya adalah membangkitkan kunci internal untuk setiap blok dengan cara membentuk jaringan feistel dan melakukan operasi XOR antara kunci internal dan blok bersangkutan. fungsi untuk membangkitkan kunci internal adalah sebagai berikut.

```
public BitArray Feistel()
{
 BitArray kl;
 BitArray kr;
 BitArray tem;
 BitArray f;
 BitArray ki=new BitArray(buffK);
 int s, i;

 s = ki.Length / 2;
 kl = new BitArray(s);
 kr = new BitArray(s);
 f = new BitArray(buffK.Length / 2);
 int x = (buffK.Length  / 2) / 2;
 for (int j = 0; j < buffK.Length / 2; j++)
 {
  f[j] = buffK[x];
  x++;
 }
 for (i = 0; i < s; i++)
 {
  kl[i] = ki[i];
  kr[i] = ki[s + i];
 }
 tem = new BitArray(kr);
 kr = new BitArray(kl.Xor(kr.Xor(f)));
 kl = tem;
```

```
 for (i = 0; i < s; i++)
 {
  ki[i] = kl[i];
  ki[s + i] = kr[i];
 }
 return (ki);
}
```

Fungsi pembangkitan kunci internal di atas digunakan di dalam prosedur di bawah ini.

```
public BitArray feisNet(BitArray c, int y)
{//membangkitkan kunci internal sebanyak round dan melakukan xor
   dengan cipherteks
 int i;
 int round = y;
 BitArray ki=new BitArray (c.Length);
 for (i = 1; i <= round; i++)
 {
  ki=new BitArray (Feistel()); }
  BitArray ba = new BitArray(c.Xor(ki));
  return (ba);
 }
```

Banyaknya putaran yang dilakukan sejumlah nomor urutan blok tersebut dimod dengan 8.Sebagai contoh, jika suatu plainteks terdiri dari 18 buah blok. Untuk blok pertama, putaran dilakukan sebanyak satu kali, sedangkan untuk blok kesepuluh putaran dilakukan sebanyak dua kali. Adapun prosedur enkripsi secara keseluruhan adalah sebagai berikut.

```
 private BitArray encryptionCBC(BitArray bay, int si)
 {
    int y, i, ix, d;
    BitArray ciper = new BitArray(bay);
    BitArray IV = new BitArray(buffK.Length);
    BitArray enc = new BitArray(buffK.Length);
    byte[] bi = new byte[] { 217 };
    BitArray tmp = new BitArray(bi);
    for (i = 0; i < (buffK.Length / 8); i++)
    {
     IV[i + 0] = tmp[0];
     IV[i + 1] = tmp[1];
     IV[i + 2] = tmp[2];
     IV[i + 3] = tmp[3];
     IV[i + 4] = tmp[4];
     IV[i + 5] = tmp[5];
     IV[i + 6] = tmp[6];
     IV[i + 7] = tmp[7];
    }
    i = 0;
    ix = 1;       //ix = nomor posisi blok
    d = 0;
```

```
        while (ix <= si)
        {
         BitArray tempo = new BitArray(buffK.Length);
         for (y = 0; y < buffK.Length; y++)
         {
          tempo[y] = bay[i];
          i++;
         }
         tempo = tempo.Xor(IV);
         if (ix % 8 == 0)
         { enc=new BitArray ( feisNet(tempo,8)); }
         else if (ix > 8)
         { enc=new BitArray (feisNet(tempo,ix % 8)); }
         else
         {enc=new BitArray (feisNet(tempo,ix));}
         int yy;
         IV = enc;
         for (yy = 0; yy < tempo.Length; yy++)
         {
           ciper[d] = enc[yy];
           d++;
         }
         ix++;
        }
       return ciper;
  }
```

Adapun prosedur untuk menghitung nilai MAC yang dihasilkan adalah berikut.

```
public BitArray nilaiMAC()
{
 int d;
 BitArray macCBC = new BitArray(buffK.Length);
 int si=addPadding();
 BitArray ciper = new BitArray(encryptionCBC(buffPad,si));
 d = ciper.Length - buffK.Length; /*posisi bit pertama pada blok
                                    terakhir*/
 for (int i = 0; i < buffK.Length; i++)
 {
  macCBC[i] = ciper[d];
  d++;
 }
 return macCBC;
```

Nilai MAC yang dihasilkan dengan menggunakan pendekatan algoritma cipher blok ini berupa bit array.

Sebagai contoh, untuk pesan “Ibu pergi ke pasar.” Dan kunci yang digunakan yaitu “kriptografi”, dengan menggunakan algoritma cipher blok di atas akan menghasilkan nilai MAC “<MAC>[285]1100101111010010001100000100000010110000000001000000000000001001001000011100111011100011110 0</MAC>”. MAC ini ditambahkan pada akhir pesan yang terletak diantara *tag* “<MAC>” dan “</MAC>”. Kode “[285]” untuk menandakan bahwa hasil MAC ini didapat dengan menggunakan algoritma berbasis cipher blok. Hal ini dilakukan agar memudahkan pada saat melakukan otentikasi karena pada saat otentikasi, pengguna hanya diminta memasukkan kunci saja, tidak diminta memilih pendekatan algoritma mana yang digunakan.

B. Fungsi *Hash*

Untuk MAC dengan pendekatan algoritma berbasis fungsi *hash*, fungsi *hash* yang digunakan adalah SHA-1. Adapun implementasi algoritma SHA-1 adalah sebagai berikut.

```
public string Hash(string s)
{
 byte[] arrByte;
 BitArray arrBit;
 int arSizePad, x, y, i, z, j, k;
 uint f, g, temp;

 uint A = 0x67452301;
 uint B = 0xEFCDAB89;
 uint C = 0x98BADCFE;
 uint D = 0x10325476;
 uint E = 0xC3D2E1F0;
 f = 0;
 g = 0;

 arrByte = UTF8Encoding.UTF8.GetBytes(s);
 arrBit = new BitArray(arrByte);
 x = arrBit.Length / 512;
 y = arrBit.Length;
 // Penambahan padding bits
 x++;
 z = (x * 512) - 64;
 arrBit.Length += 1;
 arrBit[arrBit.Length - 1] = true;
 j = z - arrBit.Length;
 for (i = 1; i <= j; i++)
 {
  arrBit.Length += 1;
  arrBit[arrBit.Length - 1] = false;
 }
 // Penambahan panjang pesan
 int[] arrInt = new int[1];
 arrInt[0] = y;
 BitArray arrPjg = new BitArray(arrInt);
 x = arrBit.Length;
 j = 64 - arrPjg.Length;
 arrBit.Length += 64;
 z = arrPjg.Length;
 for (i = 0; i < j; i++)
 {
  arrBit[x + i] = false;
 }
 for (i = 0; i < z; i++)
 {
   arrBit[x - z + i] = arrPjg[i];
 }

  x = arrBit.Length / 512;
  // Membagi pesan dalam ukuran blok 512 bit
  for (i = 0; i < x; i++)
  {
    BitArray arr512 = new BitArray(512);
    BitArray arr32;
    BitArray[] arr16blok = new BitArray[80];
    int i512;
```

```
i512 = 0;
for (int idx = 512 * i; idx < 512 * i + 512; idx++)
{
 arr512[i512] = arrBit[idx];
 i512++;
}
// membagi tiap blok ke dalam 16 buah sub blok 32 bit
for (y = 0; y < 16; y++)
{
 arr32 = new BitArray(32);
 int i32 = 0;
 for (int idx = 32 * y; idx < 32 * y + 32; idx++)
 {
   arr32[i32] = arr512[idx];
   i32++;
  }
 arr16blok[y] = arr32;
 }
 for (int id = 16; id <= 79; id++)
 {
  arr16blok[id] = (((arr16blok[id-3].Xor(arr16blok[id-
                  8])).Xor(arr16blok[id -
                  14])).Xor(arr16blok[id - 16]));
  arr16blok[id] = leftrotate(arr16blok[id], 1);
 }
 //inisialisasi peubah penyangga
 uint a = A;
 uint b = B;
 uint c = C;
 uint d = D;
 uint e = E;

 for (k = 0; k <= 79; k++)
 {
  if ((k >= 0) && (k <= 19))
  {
   f = (b & c) | ((~b) & d);
   g = 0x5A827999;
  }
  else if ((k >= 20) && (k <= 39))
  {
   f = (b ^ c ^ d);
   g = 0x6ED9EBA1;
  }
  else if ((k >= 40) && (k <= 59))
  {
   f = (b & c) | (b & d) | (c & d);
   g = 0x8F1BBCDC;
  }
  else if ((k >= 60) && (k <= 79))
  {
   f = (b ^ c ^ d);
   g = 0xCA62C1D6;
  }
  temp = ((a << 5) | (a >> 27)) + f + e + g +
         ToHex(arr16blok[k]);
  e = d;
  d = c;
  c = (b << 30) | (b >> 2);   //c=b<<<30
  b = a;
  a = temp;
```

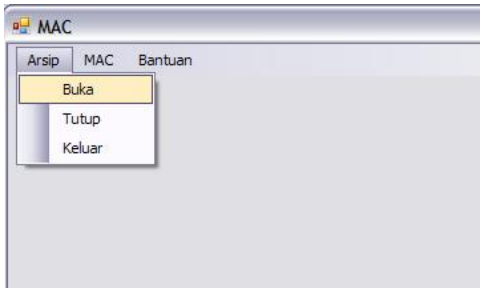
```
        }
        A = A + a;
        B = B + b;
        C = C + c;
        D = D + d;
        E = E + e;

       }
       //melakukan append
       string hashStr = Convert.ToString(A) +
                        Convert.ToString(B) +
                        Convert.ToString(C) +
                        Convert.ToString(D) +
                        Convert.ToString(E);
        return hashStr;
      }
```

Sebagai contoh, pesan "Ibu pergi ke pasar." Dengan kunci "kriptografi" akan menghasilkan "MAC <MAC>[590]29220B6FE9DA47CD090BC4DE393BBF3C6F8835C33</MAC>". MAC ini ditambahkan pada akhir pesan yang terletak diantara *tag* "<MAC>" dan "</MAC>". Kode "[590]" untuk menandakan bahwa hasil MAC ini didapat dengan menggunakan algoritma berbasis fungsi *hash* satu arah. Hal ini dilakukan agar memudahkan pada saat melakukan otentikasi karena pada saat otentikasi, pengguna hanya diminta memasukkan kunci saja, tidak diminta memilih pendekatan algoritma mana yang digunakan.

C. Pemberian MAC

Ketika pengguna ingin membangkitkan nilai MAC dari suatu pesan, maka pengguna harus membuka arsip pesan tersebut terlebih dahulu. Pembukaan arsip pesan ini dilakukan dengan memilih menu 'Buka'.

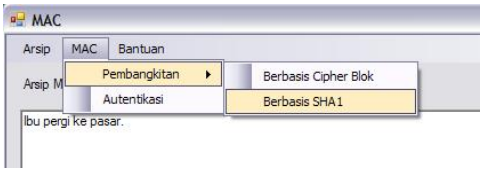


Gambar 2-C-1. Menu Buka Arsip

Sebagai contoh, pesan yang ingin diberi MAC adalah arsip 'Ibu.txt' yang berisi kalimat "Ibu pergi ke pasar.". Maka setelah pengguna memilih menu buka dan aplikasi telah membuka arsip tersebut, tampilan aplikasi seperti gambar di bawah ini.

Gambar 2-C-2. Aplikasi Menampilkan Arsip Masukan

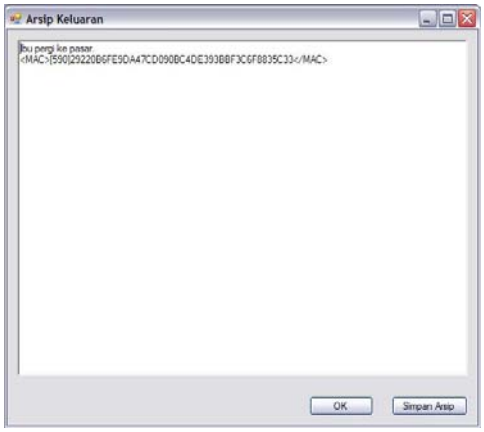
Setelah arsip masukan dibuka, maka selanjutnya adalah memilih pendekatan algoritma yang digunakan untuk membangkitkan MAC. Misalkan pendekatan algoritma yang digunakan adalah fungsi *hash* seperti gambar yang di bawah ini.



Gambar 2-C-3. Menu Memilih Pendekatan Algoritma yang Digunakan untuk Membangkitkan MAC

Setelah memilih pendekatan algoritma yang digunakan untuk

membangkitkan MAC, selanjutnya aplikasi menampilkan *form* untuk meminta masukan kunci rahasia yang akan digunakan. Jika kunci rahasia yang dimasukkan valid, maka aplikasi akan membangkitkan MAC dari arsip masukan. Misalkan kunci yang digunakan adalah "kriptografi". Ketika MAC selesai dihitung, aplikasi akan menampilkan arsip masukan yang telah dilekatkan (*embedded*) dengan MAC yang dihasilkan.



Gambar 2-C-4. Pesan yang Telah Dilekatkan dengan MAC

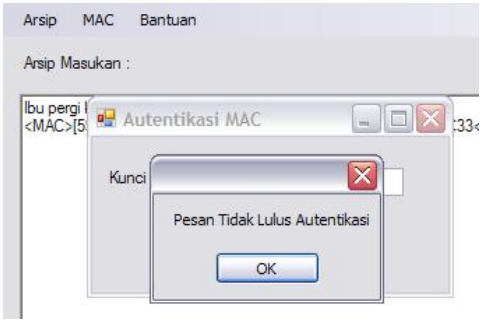
D. Autentikasi MAC

Jika pengguna ingin melakukan autentikasi MAC yang terdapat di dalam suatu pesan, maka pertama-tama pengguna harus membuka arsip yang terdapat MAC tersebut. Menu untuk membuka arsip sama dengan ketika membuka arsip masukan untuk pemberian MAC. Sebagai contoh, arsip yang ingin diautentikasi adalah "ibu.rfi" yang merupakan arsip hasil pembangkitan MAC pada contoh sebelumnya. Adapun tampilan aplikasi yang menampilkan isi arsip adalah sebagai berikut.

Gambar 2-D-1. Tampilan Isi Pesan yang Berisi MAC untuk Diautentikasi

Setelah arsip dibuka, aplikasi akan meminta masukan kunci rahasia yang digunakan. Kemudian aplikasi akan membandingkan nilai MAC yang dibangkitkan menggunakan kunci yang tadi dimasukkan dengan nilai MAC yang terdapat di dalam arsip masukan. Jika kedua buah MAC tersebut sama, maka aplikasi akan mengeluarkan *alert* yang memberitahukan bahwa pesan lulus autentikasi. Akan tetapi jika kedua buah MAC tersebut tidak sama, aplikasi akan menampilkan *alert* yang memberitahukan bahwa pesan tidak lulus autentikasi.

Gambar 2-D-2. *Alert* 'Pesan Lulus Autentikasi'



Gambar 2-D-3. *Alert* 'Pesan Tidak Lulus Autentikasi'

3. Hasil Eksperimen

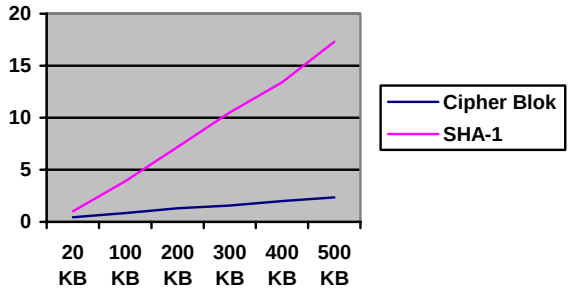
A. Perbandingan Kecepatan

Berikut ini adalah tabel kecepatan pembangkitan MAC untuk kedua buah pendekatan algoritma yang digunakan.

Ukuran File	Pendekatan Algoritma	Waktu
20 KB	Cipher blok	0.44 detik
	SHA-1	01.01 detik
100 KB	Cipher blok	0.84 detik
	SHA-1	03.90 detik
200 KB	Cipher blok	01.30 detik
	SHA-1	07.20 detik

300 KB	Cipher blok	01.56 detik
	SHA-1	10.50 detik
400 KB	Cipher blok	01.98 detik
	SHA-1	13.40 detik

500KB	Cipher blok	02.35 detik
	SHA-1	17.32 detik



Gambar 3-A-1. Diagram Perbandingan Kecepatan Antara SHA-1 dengan Cipher Blok

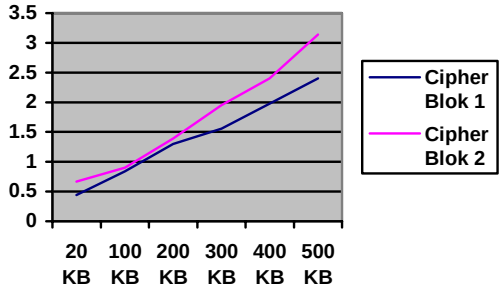
Dikarenakan algoritma cipher blok yang digunakan cukup sederhana, maka algoritma tersebut akan sedikit diperumit dengan melakukan transposisi blok-blok setelah blok-blok dienkripsi. Fungsi transposisi ini akan dipanggil setelah pemanggilan fungsi encryptionCBC. Adapun algoritma untuk transposisi adalah sebagai berikut.

```
Public BitArray transposition(int keyL, BitArray bay)
{//transposisi sepanjang keyL (panjang kunci)
 BitArray bar = new BitArray(bay.Length);
 int i, j, k, l;
 l = 0;
 for (i = 0; i < (bay.Length / keyL); i++)
 {
  j = (keyL * (keyL / 8)) + l;
  if (j > bay.Length - 1)
  {
    while (j > bay.Length - 1)
    { j = j - bay.Length; }
  }
  for (k = 0; k < keyL; k++)
  {
    bar[j] = bay[l];
    j++;
    l++;
  }
 }
 return bar;
}
```

Berikut ini adalah tabel waktu untuk membangkitkan MAC dengan menggunakan algoritma cipher blok yang telah sedikit diperumit.

Ukuran File	Waktu
20 KB	0.67 detik
100 KB	0.90 detik

200 KB	1.39 detik
300 KB	1.95 detik
400 KB	2.40 detik
500 KB	3.14 detik



Gambar 3-A-2. Diagram Perbandingan Kecepatan Cipher Blok Pertama dengan Cipher Blok Kedua

B. Perbandingan Tingkat Keamanan

Pengetesan tingkat keamanan dilakukan dengan melakukan perubahan pada kunci yang digunakan dan pada pesan, kemudian dilakukan autentikasi. Pengetesan ini dilakukan terhadap pesan arsip 'Ibu.txt' yang berisi kalimat "Ibu pergi ke pasar." Dengan kunci yang digunakan adalah "kriptografi". Jika pembangkitan MAC dilakukan dengan menggunakan pendekatan algoritma cipher blok, maka MAC yang benar (asli) adalah "11001011101001000110000010000010110000000001000000000000001001001000011100111011100011110 0". Sedangkan jika menggunakan pendekatan algoritma SHA-1 maka MAC yang dihasilkan adalah "29220B6FE9DA47CD090BC4DE393BBF3C6F8835C33". Berikut ini adalah tabel pengetesan yang telah dilakukan.

Pendekatan Algoritma	Jenis Pengetesan	Hasil
Cipher Blok	Penambahan 1 buah karakter pada pesan	Pesan tidak lulus autentikasi
	Pengurangan 1 buah karakter pada pesan	Pesan tidak lulus autentikasi
	Penggantian 1 buah karakter pada pesan	Pesan tidak lulus autentikasi
	Penambahan 1 buah karakter pada kunci	Pesan tidak lulus autentikasi
	Pengurangan 1 buah karakter pada kunci	Pesan tidak lulus autentikasi
	Penggantian 1 buah karakter pada kunci	Pesan tidak lulus autentikasi
	Penambahan 1 buah karakter pada MAC	Pesan tidak lulus autentikasi
	Pengurangan 1 buah karakter pada MAC	Pesan tidak lulus autentikasi
	Penggantian 1 buah karakter pada MAC	Pesan tidak lulus autentikasi
SHA-1	Penambahan 1 buah karakter pada pesan	Pesan tidak lulus autentikasi
	Pengurangan 1 buah karakter pada pesan	Pesan tidak lulus autentikasi
	Penggantian 1 buah karakter pada pesan	Pesan tidak lulus autentikasi
	Penambahan 1 buah karakter pada kunci	Pesan tidak lulus autentikasi
	Pengurangan 1 buah karakter pada kunci	Pesan tidak lulus autentikasi
	Penggantian 1 buah karakter pada kunci	Pesan tidak lulus autentikasi
	Penambahan 1 buah karakter pada MAC	Pesan tidak lulus autentikasi
	Pengurangan 1 buah karakter pada MAC	Pesan tidak lulus autentikasi
	Penggantian 1 buah karakter pada MAC	Pesan tidak lulus autentikasi

4. Penutup

A. Kesimpulan

1) Pembangkitkan nilai MAC dengan menggunakan algoritma cipher blok sederhana seperti pada makalah ini lebih cepat dibandingkan dengan pembangkitan MAC menggunakan fungsi *hash* SHA-1.
2) Semakin rumit algoritma cipher blok (semakin banyak operasi yang digunakan) maka waktu yang dibutuhkan untuk membangkitkan nilai MAC akan semakin lama.

3) MAC dengan menggunakan pendekatan algoritma cipher blok lebih aman karena ketergantungan keamanan bergantung pada dua hal, yaitu algoritma cipher blok (dengan catatan algoritma cipher blok bukan merupakan algoritma yang telah dipublikasikan) yang dipakai dan kunci rahasia yang digunakan. Sedangkan SHA-1 hanya bergantung pada kunci rahasia yang digunakan.

B. Saran

Jika ingin membangkitkan MAC dengan menggunakan algoritma cipher blok disarankan menggunakan algoritma cipher blok yang cukup kompleks tetapi dengan jumlah operasi yang tidak terlalu banyak agar waktu yang dibutuhkan untuk membangkitkan MAC tidaklah lama.

5. Daftar Referensi

[ITL06] http://www.itl.nist.gov/fipspubs
Diakses tanggal 6 November 2006

[LIN06] http://www.linktionary.com
Diakses tanggal 6 November 2006

[MUN06] Munir, R., (2006), Diktat Kuliah IF5054 Kriptografi.