Studi dan Analisis One-Key CBC MAC

YulieAnneria Sinaga – NIM :13504085 [1)]
1) Jurusan Teknik Informatika ITB, Bandung 40116, email: if14085@students.if.itb.ac.id

Abstract – OMAC adalah *Message Authentication Code* berbasis *block chiper* yang didesain oleh Kaoru Kurosawa. OMAC adalah variansi dari *Chiper Block Chaining Message Authentication Code* (CBC MAC). OMAC merupakan singkatan dari One-key CBC MAC. OMAC memiliki dua varian yaitu OMAC1 dan OMAC2

OMAC dapat bekerja pada *block chiper* dengan ukuran blok sebesar 64 bit dan 128 bit, dengan semua ukuran kunci. OMAC akan menghasilkan sebuah MAC dengan dimensi lebih kecil atau sama dengan ukuran blok.

Pada makalah yang akan dibuat, akan dilakukan studi dari OMAC. Studi yang dilakukan mencakup pengenalan dan pembahasan OMAC serta perbandingan antara OMAC1 dan OMAC2, kemudian dilakukan implementasi OMAC dan ujicoba dengan menggunakan AES.

Kata Kunci*: MAC, OMAC,OMAC1, OMAC2.*

1. PENDAHULUAN

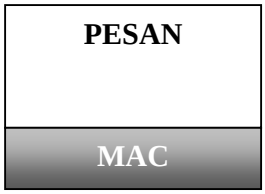
MAC *(Message Authentication Code)* adalah fungsi *hash* satu arah yang menggunakan kunci rahasia (secret key) dalam pembangkitan nilai *hash.* Dengan kata lain nilai *hash* adalah fungsi dari pesan dan kunci. MAC disebut juga *keyed hash function* atau *key-dependent one-way hash function.* MAC memiliki sifat dan fungsi yang sama seperti fungsi *hash* satu arah seperti MD5 dan SHA, hanya saja ada komponen kuni. Kunci digunakan oleh penerima pesan untuk menverifikasi nilai *hash*. [1]

Secara matematis, MAC dinyatakan sebagai

$$MAC = C_k(M) \quad (1)$$

Yang dalam hal ini, MAC = nilai *hash*, C = fungsi *hash* (atau algoritma MAC), dan K = kunci rahasia. Fungsi C memanpatkan pesan M yang berukuran sembarang dengan menggunakan kunci rahasia. Fungsi F adalah fungsi many-to-one, yang berarti beberapa pesan berbeda mungkinmemiliki MAC yang sama.

MAC digunakan untuk otentikasi pesan tanpa perlu merahasiakan (mengenkripsi) pesan. Mula-mula pengirim pesan menghitung MAC dari pesan yang ingin ia kirim dengan menggunakan kunci rahasia K menggunakan persamaan (1) (diasumsikan sebelum melakukan transmisi pengrim pesan dan penerima pesan sudah berbagi kunci rahasia), kemudian MAC ini dilekatkan (*embedded*) pada pesan (gambar 1.1).



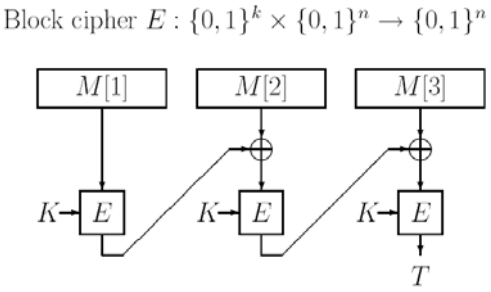
Gambar 1: MAC dilekatkan pada pesan untuk keperluan otentikasi

Selanjutnya pesan dikrim bersama-sama dengan MAC ke penerima. Penerima kemudian menggunakan kunci yang sama untuk menghitung MAC pesan dan membandingkannya dengan MAC yang diterimanya. Jika MAC ini sama maka penerima dapat menyimpulkan bahwa pesan ini dikirim oleh orang yang sesungguhnya, dan isi pesan tidak diubah oleh orang ke tiga selama mengalami transmisi. Jika pesan tidak berasal dari pengirim yang asli, maka MAC yang ia hitung tidak sama dengan MAC yang ia terima, sebab pihak ketiga tidak mengetahui kunci rahasia. Begitu juga apabila pesan sudah diubah selama transmisi, maka MAC yang ia hitung akan tidak sama dengan MAC yang ia terima.

Aplikasi MAC lainnya adalah untuk menjaga otentikasi arsip yang digunakan oleh dua atu lebih pengguna. Selain itu MAC juga digunakan utnuk menjamin integritas (keaslian) isi arsip terhadap perubahan, misalnya karena serangan virus. Caranya : hitung MAC dari arsip, kemudian simpan MAC di dalam sebuah tabel basisdata. Jika pengguna menggunakan fungsi *hash* satu arah biasa (seperti MD5), maka virus dapat menghitung *hash* yang baru dari arsip yang sudah diubah lalu mengganti nilai *hash* yang lama di dalam table. Tetapi jika digunakan mAC virus tidak dapat melakukan hal ini karena ia tidak mengetahui kunci.

Algoritma MAC dapat dirancang dengan dua pendekatan. Pendekatan pertama menggunakan algoritma kriptografi kunci-simetri berbasis blok (*chiper block*) dan pendekatan kedua adalah menggunaka fungsi *hash* satu arah.

Pendekatan yang akan dibahas pada makalah ini adalah algoritma MAC berbasis chiper block dengan mode CBC. Nilai *hash*-nya (yang menjadi MAC) adalah hasil enkripsi blok terakhir.



Gambar 2: CBC MAC

CBC MAC adalah metode yang sangat terkenal untuk meng-generate sebuah *message athentication code* (MAC) berbasis *block chiper*. Bellare, Kilian dan Rogaway telah membuktikan keamanan dari CBC MAC untuk pesan dengan panjang yang tetap *mn* bit, dimana *n* adalah panjang blok dari the *block chiper* yang mendasari E.[2]

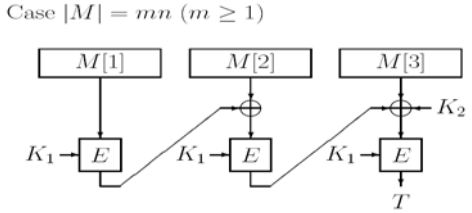
Bagaimanapun, diketahui bahwa CBC MAC tidak aman kecuali kalau panjang pesan tetap. Oleh karena itu, beberapa varian dari CBC MAC telah diajukan untuk variabel panjang pesan.

Pertama, Encrypted MAC (EMAC) telah diajukan. EMAC diperoleh dengan mengenkripsi nilai CBC MAC oleh E lagi dengan kunci baru K2. Sehingga,

$$\mathrm{EMAC}_{K1,K2}(M) = E_{K2}(\mathrm{CBC}_{K1}(M)) \qquad (2)$$

Dimana M adalah pesan, K1 adalah kunci dari CBC MAC dan CBC_{K1}(M) adalah nilai CBC MAC dari M. Petrank dan Rackoff kemudian membuktikan bahwa EMAC akan aman apabila panjang pesan adalah kelipatan positif dari *n* [2]. Bagaimanapun EMAC membutuhkan penjadwalan dua buah kunci dari the underlying *block chiper* E.

Kemudian Black dan Rogaway mengajukan XCBC yang hanya membutuhkan sebuah penjadwalan kunci dari *block chiper* yang mendasari E [2]. XCBC mengambil tiga buah kunci : satu buah kunci *block chiper* K1, dan dua n-bit kunci K2 dan K3. XCBC dideskripsikan sebagai berikut (lihat gambar 3)



Gambar 3 : XCBC

– apabila $|M| = mn$ untuk beberapa $m > 0$, maka XCBC menghitung tepat sama dengan CBC MAC, kecuali untuk melakuakan XOR sebuah n-bit kunci K2 sebelum mengenkripsi blok terakhir.

– Sebaliknya, 10*i* pengganjal ($i = n-1-|M| \bmod n$) disisipkan ke *M* dan XCBC menghitung sama persis dengan CBC MAC untuk pesan yang di-ganjal, kecuali untuk melakukan XOR kunci n-bit K3 lainnya sebelum mengenkripsi blok terakhir

Bagaimanapun, kekurangan dari XCBC adalah membutuhkan tiga buah kunci, dengan total (k+2n) bit. Akhirnya Kurosawa dan Iwata mengajukan dua-kunci CBC MAC (TMAC) [2]. TMAC menggunakan dua buah kunci , dengan total (k+n) bit: sebuah *block chiper* kunci K1 dan sebuah n-bit kunci K2. TMAC diperoleh dari XCBC dengan mengganti (K2,K3) dengan (K2.u, K2), dimana u adalah beberapa konstanta tidak nol dan "." Berarti multiplikasi dalam GF(2n).

Permasalahan dalam CBC MAC adalah : CBC MAC tidak memungkinkan pesan dengan panjang bit yang berubah-ubah (semua pesan haruslah kelipatan dari n bit) dan CBC MAC tidak memungkinkan pesan dengan panjang yang bervariasi (kecuali ingin keamanannya menjadi berkurang)

OMAC adalah MAC atau *message authentication code* yang berbasis *block chiper* yang didesain oleh Kaoru Kurosawa dan T. Iwata. OMAC merupakan varian sederhana dari CBC MAC (MAC berbasis Chiper Block Code). OMAC merupakan singkatan dari One-Key CBC MAC [2].

OMAC aman untuk pesan dengan panjang bit berapapun (padahal CBC MAC hanya aman untuk pesan dengan sebuah panjang bit yang pasti, dan panjangnya harus kelipatan dari panjang blok). Selain itu, efisiensi OMAC di-optimisasi cukup tinggi. Sehingga OMAC hampir seefisien CBC MAC.

2. PEMBAHASAN OMAC

OMAC adalah nama umum dari OMAC1 dan OMAC2. OMAC1 akan dijelaskan lebih dulu. OMAC1 ekuivalen dengan CMAC [4].

2.1. OMAC1

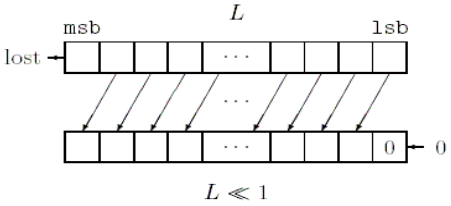
Sebelum menggunakan OMAC1, harus dipilih sebuah blockchiper E dan sebuah panjang tag t. E dapat berupa AES, Camellia, Triple-DES, atau *block chiper* apapun. Anggap panjang kunci E adalah k-bit dan panjang blok adalah n-bit.[4]

Panjang tag t dapat berupa integer apapun antara l sampai n tergantung pada aplikasi yang digunakan (tetapi dipastikan untuk mendapatkan tingkat keamanan yang diinginkan). Kemudian kunci rahasia dapat dibagi dengan partner lainnya. K adalah kunci k-bit yang merupakan satu-satunya kunci untuk E.

Pre-processing: Tahap-tahap berikut dapat dilakukan

tanpa pesan yang akan dikirim. Pertama, enkripsi n-bit 0 (dinotasikan dengan 0*) untuk menghitung *L*. Sehingga, *L* menjadi E (K , 0*). Periksa apakah bit yang paling signifikan dari *L* adalah 0.

Apabaila iya, maka *L.u* menjadi *L<<1* , dimana *L<<1* menunjukan sebuah shift yang bit-bitnya berkembang secara signifikan dengan bit yang paling signifikan menjadi hilang dan sebuah nol datang ke bit yang paling tidak signifikan (lihat gambar 4).



Gambar 4: Pergeseran *L*

Apabila tidak, maka *L.u* menjadi *(L<<1) xor Constant*, dimana *Constant* adalah konstanta *n*-bit. Apabila *n=128* , maka *Constant* adalah 0x00000000000000000000000000000087, dan apabila *n=64* , maka *Constant* adalah 0x000000000000001b, dimana bit-bit ditampilkan sebagai nilai heksadesimal dengan bit yang paling signifikan ke kiri.

Periksa apabila bit yang paling signifikan dari *L.u* adalah *0* . Jika ya, maka $L.u^{2}$ menjadi *(L.u)<<1* . Sebaliknya, $L.u^{2}$ menjadi *((L.u)<<1) xor Constant* , dimana *Constant* sama dengan diatas. Simpan *L.u* dan $L.u^{2}$.

Tag-generation: *M* adalah message atau pesan. Pecah *M* menjadi block-block *M[1], M[2], ..., M[m]* , dimana setiap *M[i] (i = 1 ,..., m-1)* terdiri atas *n* bit. Block pesan yang terakhir *M[m]* munkin memiliki lebih sedikit dari n bit (tetapi memiliki 0 bit apabila pesan M kosong).

1. *Y[0]* menjadi 0^{n} .
2. For *i = 1* to *m-1* do : *Y[i]* menjadi *E(K, M[i] xor Y[i-1])* .
3. Cek apakah panjang bit dari blok pesan terakhir adalah *n* bit.
    a. Jika ya, maka *X[m]* menjadi *M[m] xor Y[m-1] xor L.u* .
    b. Jika tidak, maka *M[m]* menjadi *M[m] 1* $0^{(n\text{-}1\text{-}(\text{bit length of } M[m]))}$. Sisipkan sebuah 1 dan kemudian sisipkan jumlah minimum dari 0, sehingga panjang total menjadi *n* bit. *X[m]* menjadi *M[m] xor Y[m-1] xor* $L.u^{2}$.
4. *T* menjadi *E(K, X[m])* .
5. Tag menjadi pemotongan *t*-bit dari *T* .
6. kembalikan nilai Tag.

Catat bahwa jika panjang pesan adalah kelipatan positif dari *n* , maka *L.u* digunakan. Apabila tidak, 10^{i} pengganjal dan $L.u^{2}$ digunakan. Jika pesan adalah string kosong, maka anda harus menyisipkan 10^{n-1} dan menggunakan $L.u^{2}$.

Berikut adalah deskripsi algoritmik dalam pseudocode (lihat gambar 5).

```
Algorithm OMAC1(K, M)
L ← E(K, 0^n)
if msb(L) = 0 then L · u ← L ≪ 1
              else L · u ← (L ≪ 1) ⊕ Constant
if msb(L · u) = 0 then L · u^2 ← (L · u) ≪ 1
                  else L · u^2 ← ((L · u) ≪ 1) ⊕ Constant
                       /* Constant is 0x0...087 (when n = 128),
                                  and 0x0...01b (when n = 64) */
Y[0] ← 0^n
Break M into blocks M[1], M[2], ..., M[m]
                  /* |M[i]| = n for i = 1, ..., m − 1, and |M[m]| ≤ n */
for i ← 1 to m − 1 do
      Y[i] ← E(K, M[i] ⊕ Y[i − 1])
if |M[m]| = n then X[m] ← M[m] ⊕ Y[m − 1] ⊕ L · u
              else X[m] ← (M[m]10^(n−1−|M[m]|)) ⊕ Y[m − 1] ⊕ L · u^2
T ← E(K, X[m])
Tag ← t-bit truncation of T
return Tag
```

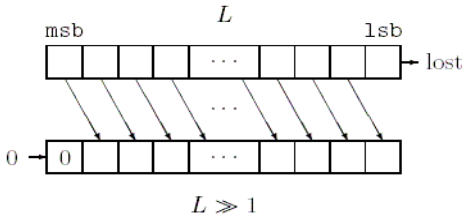
Gambar 5 : pseudocode dari OMAC1

Tag-verification: Anggap telah diterima sebuah pasangan tag pesan *(M, Tag ').* Untuk memeriksa apakah *(M, Tag ')* otentik, pertama, hitung tag *Tag* untuk pesan M dengan menggunakan generasi Tag diatas dan kunci rahasia pribadi. Jika *Tag ' = Tag* maka *M* otentik. Jika tidak, *M* tidak otentik.

2.2. OMAC 2

Selanjutnya, OMAC2 akan dibahas. Pada OMAC1, digunakan *L.u* dan $L.u^{2}$. Pada OMAC2, kita menggunakan *L.u* dan $L.u^{-1}$. Untuk menghitung $L.u^{-1}$, pertama, periksa apakah bit yang paling tidak signifikan dari *L* adalah 0.[4]

Apabila ya, maka $L.u^{-1}$ menjadi *L>>1* , dimana *L>>1* menunjukan sebuah shift dimana bit-bitnya menurun dengan bit yang paling tidak signifikan menjadi hilang dan sebuah 0 datang ke bit yang paling signifikan (lihat gambar 6).



Gambar 6 : pergeseran *L*

Jika tidak, maka $L.u^{-1}$ menjadi *(L>>1) xor Constant '* , dimana *Constant '* adalah konstanta n-bit. Jika *n=128* , maka *Constant '* adalah 0x80000000000000000000000000000043, dan jika *n=64* , maka *Constant '* adalah 0x800000000000000d

Berikut adalah deskripsi algoritmik OMAC2 dalam pseudocode

```
Algorithm OMAC2(K, M)
L ← E(K, 0^n)
if msb(L) = 0 then L · u ← L ≪ 1
              else L · u ← (L ≪ 1) ⊕ Constant
                        /* Constant is 0x0...087 (when n = 128),
                                      and 0x0...01b (when n = 64) */
if lsb(L) = 0 then L · u^-1 ← L ≫ 1
              else L · u^-1 ← (L ≫ 1) ⊕ Constant'
                        /* Constant' is 0x80...043 (when n = 128),
                                      and 0x80...0d (when n = 64) */
Y[0] ← 0^n
Break M into blocks M[1], M[2], ..., M[m]
                    /* |M[i]| = n for i = 1, ..., m − 1, and |M[m]| ≤ n */
for i ← 1 to m − 1 do
    Y[i] ← E(K, M[i] ⊕ Y[i − 1])
if |M[m]| = n then X[m] ← M[m] ⊕ Y[m − 1] ⊕ L · u
             else X[m] ← (M[m]10^(n−1−|M[m]|)) ⊕ Y[m − 1] ⊕ L · u^-1
T ← E(K, X[m])
Tag ← t-bit truncation of T
return Tag
```

Gambar 7 : pseudocode dari OMAC2

3. UJICOBA OMAC

Dilakukan pengujian OMAC yang menggunakan AES sebagai *block chiper* yang mendasari. Mengambil pesan $M \in \{0, 1\}^*$ dan mengembalikan sebuah string dalam $\{0,1\}^{128}$ sebagai tag [3]. Kunci OMAC adalah kunci AES. Akan diberikan empat buah contoh untuk setiap ukuran kunci (kunci 128-, 192-, dan 256-bit). Sehingga berjumlah dua belas contoh. Setiap string diekspresikan dalam notasi heksadesimal.

Sebagai kunci dari AES, akan digunakan nilai sebagai berikut :

$$K = \begin{cases} \texttt{2b7e151628aed2a6abf7158809cf4f3c} & \text{for 128-bit key,} \\ \texttt{8e73b0f7da0e6452c810f32b809079e5}\ \texttt{62f8ead2522c6b7b} & \text{for 192-bit key, and} \\ \texttt{603deb1015ca71be2b73aef0857d7781}\ \texttt{1f352c073b6108d72d9810a30914dff4} & \text{for 256-bit key.} \end{cases}$$

Pesan yang akan digunakan adalah 0-(string kosong), 16-, 40-, dan 64—bytes pertama dari :

6bc1bee22e409f96e93d7e117393172a
ae2d8a571e03ac9c9eb76fac45af8e51
30c81c46a35ce411e5fbc1191a0a52ef
f69f2445df4f9b17ad2b417be66c3710

Untuk pesan akan ditunjukan oleh ‘Msg’ dan output yang dihasilkan ditunjukan oleh ‘Tag’.

3.1 Test Vector dengan menggunakan AES-128

Test Vector untuk String Kosong :
K 2b7e151628aed2a6abf7158809cf4f3c
Msg <empty string>
Tag bb1d6929e95937287fa37d129b756746

Test Vector untuk pesan 16-Byte :
K 2b7e151628aed2a6abf7158809cf4f3c
Msg 6bc1bee22e409f96e93d7e117393172a
Tag 070a16b46b4d4144f79bdd9dd04a287c

Test Vector untuk pesan 40-Byte :
K 2b7e151628aed2a6abf7158809cf4f3c
Msg 6bc1bee22e409f96e93d7e117393172a
ae2d8a571e03ac9c9eb76fac45af8e51
30c81c46a35ce411
Tag dfa66747de9ae63030ca32611497c827

Test Vector untuk pesan 64-Byte :
K 2b7e151628aed2a6abf7158809cf4f3c
Msg 6bc1bee22e409f96e93d7e117393172a
ae2d8a571e03ac9c9eb76fac45af8e51
30c81c46a35ce411e5fbc1191a0a52ef
f69f2445df4f9b17ad2b417be66c3710
Tag 51f0bebf7e3b9d92fc49741779363cfe

3.2 Test Vector dengan menggunakan AES-192

Test Vector untuk String kosong :
K 8e73b0f7da0e6452c810f32b809079e5
62f8ead2522c6b7b
Msg <empty string>
Tag d17ddf46adaacde531cac483de7a9367

Test Vector untuk pesan 16-Byte :
K 8e73b0f7da0e6452c810f32b809079e5
62f8ead2522c6b7b
Msg 6bc1bee22e409f96e93d7e117393172a
Tag 9e99a7bf31e710900662f65e617c5184

Test Vector untuk pesan 40-Byte :
K 8e73b0f7da0e6452c810f32b809079e5
62f8ead2522c6b7b
Msg 6bc1bee22e409f96e93d7e117393172a
ae2d8a571e03ac9c9eb76fac45af8e51
30c81c46a35ce411
Tag 8a1de5be2eb31aad089a82e6ee908b0e

Test Vector untuk pesan 64-Byte :
K 8e73b0f7da0e6452c810f32b809079e5
62f8ead2522c6b7b
Msg 6bc1bee22e409f96e93d7e117393172a
ae2d8a571e03ac9c9eb76fac45af8e51
30c81c46a35ce411e5fbc1191a0a52ef
f69f2445df4f9b17ad2b417be66c3710
Tag a1d5df0eed790f794d77589659f39a11

3.3 Test Vector dengan menggunakan AES-256

Test Vector untuk String kosong :
K 603deb1015ca71be2b73aef0857d7781
1f352c073b6108d72d9810a30914dff4
Msg <empty string>
Tag 028962f61b7bf89efc6b551f4667d983

Test Vector untuk pesan 16-Byte :
K 603deb1015ca71be2b73aef0857d7781
1f352c073b6108d72d9810a30914dff4
Msg 6bc1bee22e409f96e93d7e117393172a

Tag 28a7023f452e8f82bd4bf28d8c37c35c

Test Vector untuk pesan 40-Byte :

K 603deb1015ca71be2b73aef0857d7781
1f352c073b6108d72d9810a30914dff4

Msg 6bc1bee22e409f96e93d7e117393172a
ae2d8a571e03ac9c9eb76fac45af8e51
30c81c46a35ce411

Tag aaf3d8f1de5640c232f5b169b9c911e6

Test Vector untuk pesan 64-Byte :

K 603deb1015ca71be2b73aef0857d7781
1f352c073b6108d72d9810a30914dff4

Msg 6bc1bee22e409f96e93d7e117393172a
ae2d8a571e03ac9c9eb76fac45af8e51
30c81c46a35ce411e5fbc1191a0a52ef
f69f2445df4f9b17ad2b417be66c3710

Tag e1992190549f6ed5696a2c056c315410

4. KESIMPULAN

Keamanan OMAC telah terbukti, bahwa ia memiliki panjang kunci yang optimal yaitu satu buah k-bit kunci K tanpa kehilangan unsur keamanan. Akan tetapi kekurangan yang dimiliki OMAC adalah satu *block chiper* meminta untuk menghitung L.

DAFTAR REFERENSI

[1] Munir, Rinaldi. *Diktat Kuliah IF5054 Kriptografi.* Program Studi Teknik Informatika, Institut Teknologi Bandung. 2006.

[2] T. Iwata, K. Kurosawa. *OMAC: One-Key CBC MAC.* Department of Computer and Information Sciences, Ibaraki University. 2003.

[3] T. Iwata, K. Kurosawa. . *OMAC: One-Key CBC MAC – Addendum.* Department of Computer and Information Sciences, Ibaraki University. 2003.

[4] T. Iwata, K. Kurosawa. *OMAC: One-Key CBC MAC.* http://www.nuee.nagoyau.ac.jp/labs/titawa/omac/omac.html