Pembangunan *MAC* Berbasis *Cipher* Aliran (RC4)

Made Harta Dwijaksara [1)]

1) Program Studi Teknik Informatika, ITB, Bandung 40132, email: if14137@students.if.itb.ac.id

Abstraksi *– Pada makalah ini akan dibahas pembangunan MAC (Message Authentication Code) yang berbasiskan cipher aliran. Adapun algritma cipher aliran yang dipilih adalah algoritma RC4 (ARCFOUR). Algoritma ini dipilih karena relatif sederhana dan dari sisi komputasi yang diperlukan juga relatif sedikit. Pada cipher aliran terdapat keystream generator yang berfugsi untuk membangkitan rangkain kunci acak berdasarkan seeds yang diberikan. Untuk dapat menghasilkan nilai MAC suatu pesan maka dapat memanfaatkan keystream generator yang dipergunakan secara simultan antara proses enkripsi atau dekripsi pesan dengan pembentukan nilai MAC suatu pesan yang diproses. Oleh karena kesederhanaan dari algoritma RC4 maka penambahan sequence baru untuk pembentukan MAC tidak akan memiliki pengaruh besar terhadap performansi dari algoritma ini.*

Kata Kunci*:MAC, RC4, cipher aliran, key stream generator, enkripsi, dekripsi.*

1. PENDAHULUAN

Salah satu teknik untuk menjaga otentikasi pesan adalah dengan menggunakan *MAC*. *MAC* adalah fungsi hash satu-arah yang menggunakan kunci rahasia dalam pembangkitan nilai hash [RIN06]. Secara matematis, *MAC* dinyatakan sebagai

$$MAC = Ck(M) \qquad (1)$$

yang dalam hal ini, *MAC* = nilai hash, C = fungsi hash (atau algoritma *MAC*), dan *K* = kunci rahasia. Fungsi *C* memampatkan pesan *M* yang berukuran sembarang dengan menggunakan kunci *K*. Perumusan pembentukan *MAC* ini memberikan kemungkinan bahwa *MAC* dapat dibangun dari algoritma enkripsi pesan yang berbasiskan aliran (*cipher* aliran) pada suatu komunikasi jaringan. Karena pada prinsipnya *MAC* akan membangkitkan suatu nilai acak yang konstan untuk semua panjang pesan dengan suatu kunci rahasia.

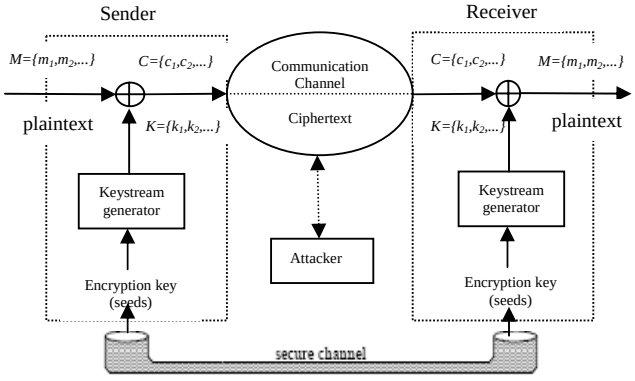
Adapun keuntungan penggunaan *MAC* berbasiskan *cipher* aliran selain dapat dipergunakan untuk mengotentikasi suatu pesan, *MAC* juga dapat dipergunakan sebagai *check sum* pengiriman pesan karena nilai yang dihasilkan akan memberikan informasi tentang paket yang dikirim. Hal ini sekaligus akan memberikan aspek *reliability* pada pada sistem komunikasi yang dibentuk.

2. ALGORITMA RC4 (*ARCFOUR*)

Algoritma RC4 adalah cipher aliran yang dipergunakan secara luas pada sistem keamanan seperti protokol SSL (*Secure Socket Layer*) [RIN06]. Agoritma ini sederhana dan mudah untuk diimplementasikan. RC4 membangkitkan aliran kunci (*keystream*) yang kemudian di-XOR-kan dengan plainteks pada waktu enkripsi (atau di-XOR-kan dengan bit-bit cipherteks pada waktu dekripsi). Tidak seperti cipher aliran yang memproses data dalam bit, RC4 memproses data dalam ukuran byte (1 byte = 8 bit). Untuk membangkitan aliran kunci, cipher menggunakan status internal yang terdiri dari dua bagian:

1. Permutasi angka 0 sampai 255 didalam larik $S_0, S_1, \ldots, S_{255}$. Permutasi merupakan fungsi dari kunci *U* dengan panjang variabel.
2. Dua buah pencacah indeks, i dan j

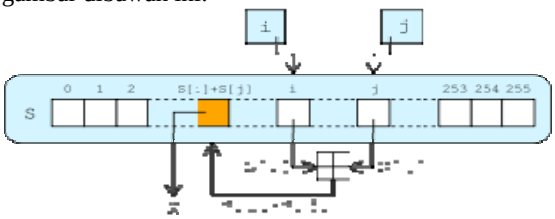
Proses enkripsi dan dekripsi dari RC4 dapat dilihat pada gambar berikut:



Gambar 1: Proses dekripsi dan enkripsi RC4

Gambar diatas memperlihatkan proses enkripsi dan denkripsi pesan pada pengirim dan penerima. Pesan yang dikirimkan akan di-XOR-kan dengan rangkaian kunci yang dibentuk oleh keystream generator yang mendapat umpan dari user. Sedangkan pada sisi penerima pembangkitan kunci acak oleh keystream generator juga dilakukan dengan umpan yang sama yang dikirimkan melalui suatu saluran khusus yang aman. Kemudian rangkaian kunci acak yang dihasilkan di-XOR-kan kembali dengan cipherteks yang diterima.

Untuk diagram *keystream generator* dapat dilihat pada gambar dibawah ini:



Gambar 2: Diagram pembangkitan kunci aliran RC4

Langkah-langkah dari algoritma RC4 adalah sebagai berikut:

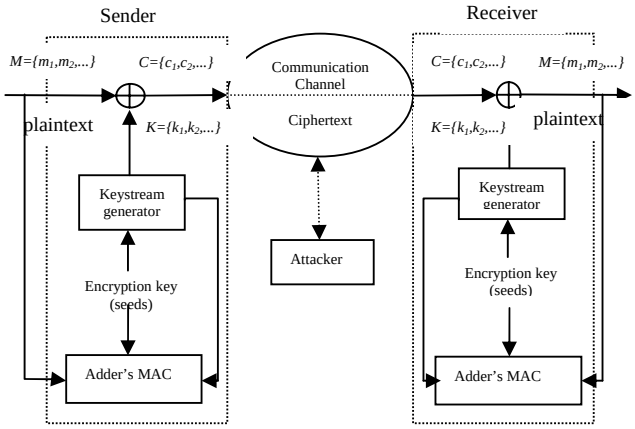
1. Inisialisasi larik S sehingga $S_0 = 0$, $S_1 = 1$, ..., $S_{255} = 255$
2. Jika panjang kunci $U < 256$, lakukan *padding* yaitu penambahan *byte* semu sehingga panjang kunci menjadi 256 *byte*. Misalnya jika U = "abc" yang hanya terdiri dari 3 *byte* maka lakukan padding dengan penambahan byte semu, misalnya U = "abcabcabc..." sampai panjang U mencapai 256 *byte*.
3. Lakukan permutasi terhadap nilai didalam larik S.
4. Bangkitkan aliran kunci (*keystream*) dan lakukan enkripsi dengan fungsi XOR.

$$\text{Cipherteks}_{(i)} = \text{Plainteks}_{(i)} \text{ XOR } Key_{(i)}$$

3. ALGORITMA RC4 DENGAN *MAC*

Seperti yang telah dijelaskan sebelumnya bahwa untuk menambahkan proses pembangkitan nilai *MAC* pada algoritma RC4 maka akan dilakukan proses konkuren antara proses enkripsi/dekripsi dengan proses pembangkitan *MAC*. Hal ini dikarenakan untuk membangkitan nilai *MAC* akan dipergunakan keystream generator dikombinasikan dengan pesan yang akan dienkripsi/dekripsi, dimana kedua proses ini akan dilakukan secara *realtime.*

Ide dibalik digunakannya *keystream generator* yang dikombinasikan dengan pesan sebagai sarana pembangkitan nilai *MAC* adalah pengertian tentang nilai *MAC* itu sendiri. Nilai *MAC* suatu pesan adalah suatu nilai unik untuk pesan yang bersangkutan dengan panjang tertentu. Dengan kata lain nilai *MAC* untuk pesan yang berbeda haruslah tidak sama. Disisi lain, *keystream generator* adalah suatu pembangkit kunci acak yang mana kunci acak tersebut akan dipergunakan untuk mengenkripsi atau mendekripsi pesan. Nilai dari rangkain kunci yang dihasilkan ini akan bergantung pada masukan umpan yang diberikan. Karena sifatnya ini maka nilai rangkaian kunci akan sangat bergantung pada nilai umpan yang diberikan. Jika rangkaian kunci yang dihasilkan ini dikombinasikan dengan pesan untuk memperoleh suatu nilai tertentu maka, nilai ini akan bersifat unik juga terhadap pesan yang lain. Untuk mendapatkan nilai serupa maka dibutuhkan umpan dan pesan yang benar-benar sama dengan yang dipergunakan untuk membentuk nilai tersebut. Hasil ini sesuai dengan konsep dari nilai *MAC* itu sendiri yaitu suatu nilai yang unik untuk suatu pesan tertentu sehingga dapat dipergunakan untuk menjaga integritas dari pesan tersebut. Konsep diatas dapat diperlihatkan sebagai berikut :



Gambar 3: RC4 dengan MAC

Dari gambar diatas dapat dilihat bahwa baik pada pihak pengirim atau penerima akan ditambahkan sebuah proses pembangkitan nilai *MAC* yang menerima masukan data berupa pesan yang akan dikirimkan dan rangkaian kunci yang dibentuka oleh *keystream generator*. Perlu diketahui bahwa nilai *MAC* dibangun secara simultan bersamaan dengan proses dekripsi/enkripsi pesan.

Untuk membentuk *MAC* yang berbasiskan algoritma cipher aliran maka terdapat dua langkah yang harus dilakukan yaitu : proses akumulasi dan proses finalisasi [PHI05]. Adapun kedua proses tersebut adalah sebagai berikut :

3.3 Proses Akumulasi

Proses akumulasi ini ditujukan untuk mengumpulkan nilai *MAC* sehingga panjang *MAC* yang dihasilkan sesuai denga yang dibutuhkan. Panjang *MAC* disini sebenarnya tidak memiliki standar namun untuk keseragaman akan dipergunakan nilai *MAC* dengan panjang 64-bits. Oleh karena sifat dari cipher aliran yang mengenkripsi pesan berdasarkan aliran dari pesan tersebut maka tidak mungkin dalam sekali enkripsi didapatkan suaut nilai *MAC* dengan panjang 64-bits, maka dari itulah dibutuhkan proses akumulasi untuk mengumpulkan nilai *MAC*, yang dibentuk secara *realtime* per *byte*, sehingga didapat panjang 64-bits.

Proses akumulasi yang dilakukan adalah dengan mengumpulkan nilai MAC yang didapat dari menerapkan persamaan berikut :

$$MAC_{(i)} = \left(\left(FK_{(i)} \oplus m_{(i)}\right) + MAC_{(i-1)}\right) \bmod 256 \quad (2)$$

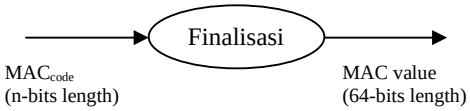
$$FK_{(i)} = \left(K_{(i)} + K_{(i-1)}\right) div 2 \quad (3)$$

$$MAC_{code} = \left\{MAC_{(1)}, MAC_2, ..., MAC_n\right\} \quad (4)$$

Keterangan:
$MAC_{(i)}$: nilai *MAC* ke-i
$FK_{(i)}$: *feed back key* ke-i, yaitu *key* yang dibentuk dari kombinasi key saat ini dengan *key* sebelumnya dari *keystream*
$K_{(i)}$: key ke-i dari *keystream*
MAC_{code} : adalah hasil akumulasi nilai *MAC*

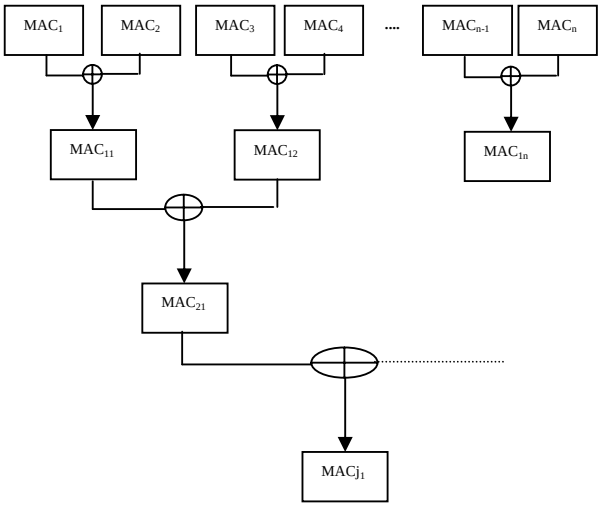
3.2 Proses Finalisasi

Dengan persamaan diatas akan didapatkan nilai *MAC* yang panjangnya mungkin saja lebih atau kurang dari 64-bits. Proses finalisasi berfungsi untuk melakukan penyesuaian terhadap panjang *MAC* yang telah dibentuk. Penyesuaian pajang dilakukan dengan cara penerapan prinsip XOR bertingkat sehingga akan sekaligus menimbulkan efek *diffusion* dan *confusion*, dengan demikian keunikan nilai *MAC* akhir akan didapat. Keluaran dari proses ini adalah nilai *MAC* dengan panjang 64-bits.



Gambar 4: Diagram proses finalisasi

Adapun proses XOR bertingkat yang diterapkan adalah sebagai berikut :



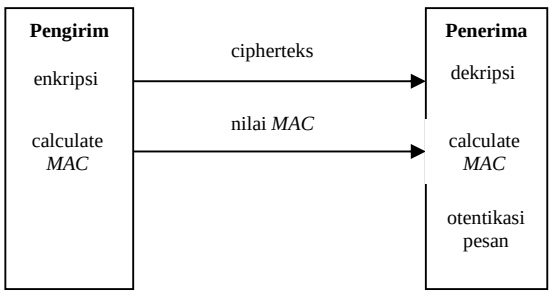
Gambar 5: Diagram XOR bertingkat

Dari gambar diatas terlihat bahwa MAC_{code} yang diperoleh dari proses akumulasi akan dibagi-bagi menjadi beberapa bagian yang kemudian akan dipasangkan untuk di-XOR-kan antar bagian, demikian juga dengan hasil XOR tersebut akan kembali dipasangkang untuk di-XOR-kan lagi sampai pada akhirnya tersisa satu bagian saja dengan panjang 64-bits. Jika ternyata panjang suatu bagian tidak sama dengan yang lain maka akan ditambhakan *byte-byte* pengganjal (*padding byte*). Nilai *MAC* inilah yang kemudian dikirimkan kepada penerima yang nantinya akan dipergunakan untuk mengotentikasi pesan apakah telah terjadi perubahan atau tidak.

4. PROTOKOL KOMUNIKASI

Dalam berkomunikasi menggunakan algoritma RC4 dengan *MAC* maka kedua belah pihak terlebih dahulu haruslah sudah melakukan pertukaran kunci rahasia mealuli suatu saluran khusus yang dipercayai benar-benar aman dari serangan. Kemudian pihak tersebut harus mengikuti protokol komunikasi berikut :

1. Pihak Pengirim
   a. Mengenkripsi pesan dengan algoritma RC4 sesuai kunci rahasia yang telah dipertukarkan sekaligus menghitung nilai *MAC*.
   b. Mengirimkan setiap hasil enkripsi pesan dan melakukan akumulasi nilai *MAC*.
   c. Setelah semua pesan terkirim maka nilai *MAC* yang telah melalui tahap finalisasi dikirimkan kepada pihak penerima.
   d. Mengirimkan setiap hasil enkripsi pesan dan melakukan akumulasi nilai *MAC*.
2. Pihak Penerima
   a. Melakukan dekripsi pesan sesuai dengan kunci rahasia yang telah dipertukarkan sekaligus menghitung nilai *MAC*.
   b. Setelah melalui tahap finalisasi kemudian nilai *MAC* yang diperoleh dibandingkan dengan nilai *MAC* yang diterima dari pengirim, jika hasilnya sama berarti pesan yang dikirim asli jika tidak berarti telah terjadi perubahan pada pesan yang dikirimkan.



Gambar 6 : Protokol komunikasi RC4 dengan MAC

5. IMPLEMENTASI

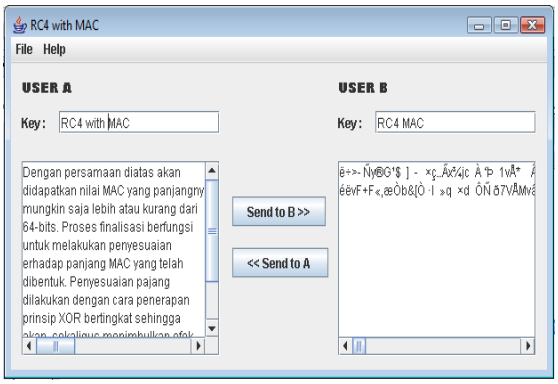
Proses yang dijelaskan diatas telah diimplementasikan dengan bahasa pemrograman java. Adapun algoritma hasil modifikasi pada algoritma RC4 untuk menambahkan proses penghitungan *MAC*, berikut potongan source code dari algoritma RC4.

```
for(X=0; X < Data.length(); ++X) {
     i = ((i+1) & 0xFF);
     Tmp = CypherBytes[i];
     Jump = ((Jump+Tmp) & 0xFF);
     T = ((Tmp+CypherBytes[Jump]) & 0xFF);
     CypherBytes[i] = CypherBytes[Jump];
     CypherBytes[Jump] = Tmp;
     Result += (char)(Data.charAt(X) ^
          CypherBytes[T]);

     //--------mac acumulation----------
     FK = (CypherBytes[T] + FK) / 2;
     MACi = ((FK^Data.charAt(X))+MACi) % 256;
     MAC += (char)(MACi);
}

//--------mac finalizatiton----------
ArrayChar = ArrayBlock[0].toCharArray();
for (i=1;i<m;i++){
    for (j=0;j<8;j++){
        ArrayChar[j] += (ArrayChar[j] ^
                    ArrayBlock[i].charAt(j));
    }
}
```

Program yang dibuat memerlukan parameter masukan yang didefinisikan oleh user berupa pesan yang akan dikirimkan dan masukan kunci rahasia.



Gambar 7 : Hasil implementasi program

Untuk implementasi maka dibuat dua buah user dengan masing-masing diimplementasikan dengan proses yang berbeda sehingga seolah-olah seperti terdapat dua pihak yang berkomunikasi. Masing-masing pihak akan berkomunikasi menggunakan protokol RC4 dengan *MAC*, dengan catatan disini kunci tidak dikirimkan melalui aplikasi ini, melainkan diasumsikan pihak yang berkomunikasi telah terlebih dahulu melakukan pertukaran kunci melalui suatu saluran yang aman.

6. PENGUJIAN

Untuk pengujian akan dipergunakan file tes yang tidak terlalu panjang demikian juga dengan ukuran kunci dipergunakan ukuran kunci 4 *byte* atau 32 *bit*. Adapun pengujian dilakukan dengan skenario sebagai berikut :

1. Skenario normal
   Pengirim
   pesan : ABCDEFGHIJKLMNOPQRSTUVWXYZ
   kunci : 1234
   cipherteks : D 2ë¶ª@EFQK ¨)K„òjnß¡AÎ]Ú
   MAC : _8¦¯?o[ì
   Penerima
   pesan : D 2ë¶ª@EFQK ¨)K„òjnß¡AÎ]Ú
   kunci : 1234
   cipherteks : ABCDEFGHIJKLMNOPQRSTUVWXYZ
   MAC : _8¦¯?o[ì
   Otentikasi pesan berhasil → pesan otentik
2. Kunci rahasia tidak sama
   Pengirim
   pesan : ABCDEFGHIJKLMNOPQRSTUVWXYZ
   kunci : 1234
   cipherteks : D 2ë¶ª@EFQK ¨)K„òjnß¡AÎ]Ú
   MAC : _8¦¯?o[ì
   Penerima
   pesan : D 2ë¶ª@EFQK ¨)K„òjnß¡AÎ]Ú
   kunci : 1289
   cipherteks : û]:ë”Ì ;¿ ¿.í'œ-´’[¿j
   MAC : U?µi¢??2?
   Otentikasi gagal
3. *Attacker* merubah isi cipherteks
   Pengirim
   pesan : ABCDEFGHIJKLMNOPQRSTUVWXYZ
   kunci : 1234
   cipherteks : D 2ë¶ª@EFQK ¨)K„òjnß¡AÎ]Ú
   MAC : _8¦¯?o[ì
   Attacker
   cipherteks : @EFQK ¨)K„òjnß¡AÎ]Ú
   Penerima
   pesan : @EFQK ¨)K„òjnß¡AÎ]Ú
   kunci : 1234
   cipherteks : E 7þ¸Ì¯$DŸò:‹¸¥•meç«
   MAC : ?°? ÑµÁ
   Otentikasi pesan gagal → pesan tidak otentik
4. *Attacker* mengganti pesan dan menghitung kembali nilai *MAC* lalu mengirimkan ke penerima
   Pengirim
   pesan : ABCDEFGHIJKLMNOPQRSTUVWXYZ
   kunci : 1234
   cipherteks : D 2ë¶ª@EFQK ¨)K„òjnß¡AÎ]Ú
   MAC : _8¦¯?o[ì
   Attacker
   pesan : 1234567890ABCDEFGHIJKLMNOP
   kunci : abcd
   cipherteks : ç[tv²v g7§uz- aTñnÉç•¦õŸ¾F
   MAC : ?fFbÖ¼ W
   Penerima
   pesan : ç[tv²v g7§uz- aTñnÉç•¦õŸ¾F

kunci : 1234
cipherteks : â ÙAšj8¼u*Èbe RVôla±lš=
MAC : ÝÁ.â +?ï
Otentikasi pesan gagal → pesan tidak otentik

Pada pengujian diatas terlihat bahwa panjang dari nilai *MAC* yang dihasilkan adalah 8 karakter (*byte*) atau sama dengan 64-bits. Setiap perubahan yang terjadi baik pada isi pesan ataupun kunci yang dipergunakan untuk melakukan dekripsi/enkripsi akan mengakibatkan gagalnya proses otentikasi pesan oleh pihak penerim, hal ini dikarenakan pembentukan nilai *MAC* melibatkan baik isi pesan semula dan kunci rahasia yang dipergunakan untuk enkripsi.

7. KESIMPULAN

Kriptografi berbasis aliran (*stream cipher*) ternyata dapat dipergunakan sebagai dasar pembentukan suatu nilai *MAC* untuk pesan yang akan dienkripsi yaitu dengan memanfaatkan *keystream* yang dibangkitkan dari keystream generator dan dikombinasikan dengan pesan dan kunci rahasia, sehingga nilai yang dihasilkan akan bersifat unik untuk suatu pesan.

Pembentukan *MAC* dilakukan perkarakter sesuai dengan mekanisme pada RC4 yang juga mengenkripsi pesan karakter demi karakter. Sebagai akibat dari hal ini yaitu bahwa pembangkitan kunci random untuk langkah selanjutnya harus menunggu sampai nilai *MAC* untuk karakter tersebut telah dihasilkan.

Aplikasi kriptografi berbasis aliran ini banyak dipergunakan pada komunikasi jaringan yang bersifat *realtime* maka sangat tepat bila sekaligus diaplikasikan mekanisme pembangunan MAC yang akan sekaligus menjaga integritas pesan dan sebagai *checksum* untuk memberikan aspek *reliability*.

DAFTAR REFERENSI

[JON98] Knudsen, Jonathan B, "*Java Cryptography*", 1998, O'Reilly.

[RIN06] Munir, Rinaldi, "*Dikat Kuliah IF5054 Kriptografi*" ,2006.

[PHI05] Hawkes, Philip, "*Primitive Spesification for Sober-128 "*, 2005.

Catatan : Source Code program dapat diambil pada link berikut → http://students.itb.ac.id/~harta-if135/Kripto/RC4withMAC