Pembangunan Algoritma MAC Berbasis Cipher Aliran

Rika Safrina – NIM : 13503053

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesha 10 Bandung
2007
E-mail : if13053@students.if.itb.ac.id

Abstrak

MAC (Message Authentication Code) adalah sebuah tanda pengenal untuk membuktikan keaslian suatu dokumen yang didapatkan dengan menggunakan pesan tak bermakna yang diperoleh dari pemrosesan sebagian isi dokumen menggunakan sebuah kunci privat. Secara teknis, (setengah) dokumen diproses menggunakan kunci privat sehingga menghasilkan pesan *MAC*, yang lebih sederhana dari isi dokumen. Pesan *MAC* ini kemudian dilekatkan dengan dokumen dan dikirim ke penerima. Penerima kemudian menggunakan kunci yang sama untuk memperoleh pesan *MAC* dari dokumen yang diterima dan membandingkannya dengan pesan *MAC* yang ia terima.

Oleh karena itu, *MAC* dapat menjamin 2 macam keamanan data: integritas dan otentikasi. Integritas dapat terdeteksi jika isi dokumen diubah, karena pesan *MAC* yang dihasilkan dan diterima akan berbeda. Sedangkan otentikasi terdeteksi dengan penggunaan kunci privatnya. Hanya pengirim dan penerima yang berhak lah yang mengetahui kunci privat tersebut. Bedanya dengan tanda tangan digital adalah pada kunci ini. *MAC* menggunakan kunci yang sama (algoritma simetri), sedangkan tanda tangan digital menggunakan kunci privat dan kunci publik (algoritma kunci publik).

Algoritma *MAC* dapat dibangun dengan menggunakan dua cara, yaitu fungsi hash dan algoritma simetri. Jika menggunakan *hash*, fungsi ini diberlakukan bagi isi dokumen yang telah dilekatkan dengan kunci privat. Sedangkan algoritma simetri yang banyak diimplementasikan dalam *MAC* sejauh ini adalah dengan mengambil satu blok dari isi dokumen dan dilakukan enkripsi menggunakan algoritma *cipher* blok.

Algoritma *cipher* aliran bisa juga digunakan untuk membangun sebuah algoritma *MAC*, yaitu dengan mengambil sebagian dari isi dokumen yang sekiranya unik dan penting (sehingga setiap pesan dapat memiliki pesan *MAC* yang berbeda). Makalah ini akan menjelaskan proses pembangunan algoritma *MAC* dengan menggunakan algoritma simetri *cipher* aliran.

Kata kunci: *Mesage Authentication Code,* fungsi *hash,* kunci simetri, *cipher* aliran.

1. Pendahuluan

Ada 4 jenis keamanan yang diinginkan oleh manusia dalam mengirim dan/atau menerima pesan, yaitu kerahasiaan, integritas, otentikasi dan nirpenyangkalan.

Setiap pesan belum tentu membutuhkan semua jenis keamanan di atas. Adakalanya suatu pesan sah-sah saja dibaca oleh pihak lain (yang bukan pengirim ataupun penerima pesan yang berhak), asalkan memiliki bukti yang kuat bahwa pesan itu berasal dari pengirim pesan yang benar dan isi pesan nya tidak diubah-ubah oleh pihak yang tidak berkepentingan. Hal ini berarti bukan kerahasiaan yang dipentingkan, melainkan integritas pesan dan otentikasi pengirim. Contoh pesan seperti ini adalah obrolan di internet antara 2 pihak, dimana isi obrolan bersifat umum dan boleh dibaca oleh banyak pihak.

Berawal dari itu, muncullah aplikasi kriptografi selain algoritma penyandian pesan, yaitu teknik untuk menjaga keaslian pesan dan kebenaran pengirim, yang salah satu tekniknya adalah *Message Authentiation Code* (MAC). Teknik lainnya dikenal dengan *Digital Signature*, yang akan dibahas perbandingannya dengan *MAC* pada poin 4.

Jadi *MAC* adalah sebuah tanda pengenal untuk membuktikan keaslian suatu dokumen berupa pesan tak bermakna yang diperoleh dari pemrosesan isi dokumen menggunakan sebuah kunci privat.

Atau secara matematis dapat dinyatakan sebagai berikut:

$$MAC = G_k(x) \qquad ...[1]$$

Keterangan:

- G: fungsi untuk mengambil intisari/rangkuman dari *x*
- x : dokumen yang ingin diberikan sebuah *MAC*
- k : kunci rahasia yang diketahui oleh kedua belah pihak, yang akan digunakan dalam pembangkitan *MAC*

Fungsi untuk menghasilkan *MAC* harus memiliki sifat-sifat sebagai berikut:

a. Kemudahan komputasi

Jika diberikan masukan kunci *k* dan dokumen *x*, proses menjalankan fungsi $G_k(x)$ harus mudah dan cepat, dengan *MAC* sebagai hasilnya **[1]**.

b. Kompresi

Karena *MAC* berperan sebagai tanda pengenal, sebaiknya *MAC* tidak menambah ukuran dokumen secara signifikan. Oleh karena itu, *MAC* yang dihasilkan harus merupakan intisari/ringkasan dari dokumen *x*, yang panjangnya lebih kecil dari panjang dokumen.

c. Kekuatan Komputasi

Sebagai penyerang, untuk memanipulasi dokumen *MAC* harus mengetahui kunci *k*. Hal ini tidak terlalu sulit dilakukan jika komputasi fungsi *G* yang diimplementasikan tidak kuat untuk melawan serangan-serangan yang mungkin terjadi pada *MAC*, yaitu:

1. *Known-text attack* → penyerang dapat menentukan pola *MAC* dari dua atau lebih pasang $(x,G_k(x))$

2. *Chosen-text attack* → penyerang dapat menentukan pola *MAC* dari pasangan $(x,G_k(x))$ yang dipilihnya sendiri

3. *Adaptive chosen-text attack* → penyerang dapat menentukan pola *MAC* dari pasangan $(x,Gk(x))$ yang mengarah ke penemuan kunci

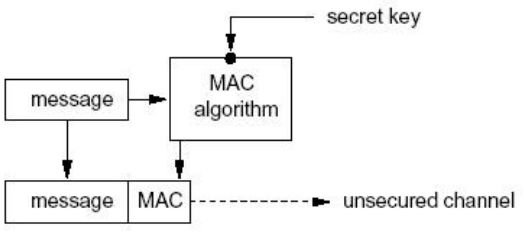
Fungsi *G* harus bersifat seefisien mungkin (seperti yang disebutkan pada poin no.1 di

atas), namun perlu juga diperhatikan bahwa fungsi ini harus kuat untuk memperkecil kemungkinan menghasilkan *MAC* yang sama atau berpola. Jadi, dibutuhkan fungsi *G* yang rumit namun tetap cepat.

Fungsi *G* memampatkan dokumen *x* yang berukuran sembarang dengan menggunakan kunci *k*. Fungsi *G* adalah fungsi *many-to-one*, yang berarti beberapa dokumen berbeda mungkin memiliki *MAC* yang sama, tetapi menemukan pesan-pesan semacam itu sangat sulit [MUN06], tergantung dari kerumitan fungsi *G* yang diimplementasikan. Walaupun disebut fungsi, namun pada kenyataannya untuk membangkitkan suatu *MAC* membutuhkan beberapa fungsi/metode dan prosedur. Untuk selanjutnya fungsi *G* disebut dengan algoritma *MAC*.

Oleh karena itu, dapat disimpulkan bahwa secara fungsional dibutuhkan 2 buah parameter untuk membangun sebuah *MAC*: kunci rahasia dan dokumen masukan.

Kemudian *MAC* akan dilekatkan pada dokumen dan dikirimkan ke penerima. Begitu juga di sisi penerima, *MAC* dari dokumen dibangkitkan dengan kunci *k* yang sama, lalu dicocokkan apakah *MAC* yang dihasilkan sama dengan *MAC* yang terlekat pada dokumen. Jika tidak, maka terdapat dua kemungkinan, yaitu kunci *k* yang dimasukkan berbeda atau dokumen yang dikirimkan tidak asli (telah dimanipulasi). Untuk lebih jelasnya, lihat diagram *MAC* pada **Gambar 1** di bawah ini.



Gambar 1 Diagram *MAC*

MAC juga digunakan dalam pengiriman pesan melalui *SSL* (*Secure Socket Layer*). *SSL* adalah protokol yang digunakan untuk *browsing web* secara aman. Dalam hal ini, *SSL* bertindak sebagai protokol yang mengamankan komunikasi antara *client* dan *server*. Protokol ini memfasilitasi penggunaan enkripsi untuk data yang rahasia dan membantu menjamin integritas informasi yang dipertukarkan antara *website* dan *web browser*.

MAC sendiri digunakan di *SSL* dalam proses pembungkusan pesan oleh *SSL record*. Algoritma yang digunakan adalah algoritma *MAC* yang berbasis fungsi *hash* MD5. Untuk penjelasan mengenai pendekatan (basis) dalam merancang algoritma *MAC* akan dijelaskan pada poin 5.

2. Jaminan Keamanan dari *MAC*

Seperti yang telah dijelaskan sebelumnya, ada dua segi keamanan yang dijamin oleh *MAC*, yaitu integritas pesan dan keabsahan pengirim. Berikut penjelasan lebih rinci bagaimana *MAC* dengan prosedur kerja seperti yang dijelaskan di atas dapat memenuhi kebutuhan akan dua segi keamanan tersebut:

a. Otentikasi Pengirim

Otentikasi adalah keamanan yang memeriksa identifikasi pihak-pihak yang saling berkomunikasi (pengirim dan penerima dokumen).

Dalam *MAC* terdapat kode identifikasi perseorangan yang terenkripsi dengan sebuah kunci rahasia. Jika *MAC* yang dihasilkan sama dengan yang diterima, tandanya kunci rahasia yang digunakan benar. Karena kunci rahasia tersebut hanya diketahui oleh dua orang, maka penerima merasa yakin bahwa pesan yang ia dapat benar berasal dari pengirim yang tepat.

b. Integritas Pesan

Integritas adalah keamanan yang menjamin isi dokumen asli dan tidak diubah saat proses transmisi. Idealnya, program juga dapat mengidentifikasi bagian mana dan jenis manipulasi seperti apa yang dilakukan terhadap dokumen. Namun jika adanya ketidakcocokan antara *MAC* yang dibangkitkan dengan *MAC* yang diterima, maka penerima tidak perlu mempedulikan isi dokumen karena dokumen tersebut kemungkinan besar sudah dimanipulasi.

Integritas data adalah properti dimana data tidak diubah oleh perlakuan yang tidak diharapkan sejak data tersebut dibuat, dikirim atau disimpan di tempat yang sesuai.

Verifikasi integritas data membutuhkan hanya sekumpulan data *item* yang memenuhi kriteria yang telah ditetapkan. Selain dari itu, maka data dikatakan tidak valid lagi.

Kriteria pengenalan integritas data ini termasuk redundansi dan format. Fokus spesifik dari integritas data adalah komposisi data dalam satuan bit (*bitwise*). Operasi dikatakan invalid dalam integritas data jika terjadi penambahan bit, penghapusan, perubahan susunan bit, inversi atau substitusi bit, dan kombinasi dari itu semua.

Karena *MAC* juga menyediakan layanan keamanan ini, maka sebenarnya tidak adil jika algoritma ini disebut *Message Authentication Code* (kode untuk mengotentikasi pesan) saja. Makanya algoritma ini juga memiliki nama lain, yaitu *Message Integration Code (MIC)* yang memenuhi sifat dalam menjaga keamanan integritas pesan.

3. Perbedaan *MAC* dan *hash*

Di dalam dunia kriptografi, kita mengenal istilah *message digest* (intisari). *Message digest* adalah sebuah intisari yang unik dari sebuah pesan yang membantu penerima pesan untuk mengetahui apakah pesan yang diterimanya sama persis dengan pesan yang dikirim untuknya.

Setiap *message digest* adalah unik, namun tidak bisa dikembalikan ke pesan semula, atau dengan kata lain *non-reversible* (satu arah). Di sisi pengirim, *message digest* dikalkulasi dan dikirimkan bersama dengan pesan. Sedangkan di sisi penerima, *message digest* juga dikalkulasi lalu dicocokkan apakah hasilnya sama dengan yang dilekatkan pada pesan yang diterimanya atau tidak.

Melihat dari pengertian ini, jelas bahwa *MAC* termasuk ke dalam *message digest*. Fungsi *message digest* lainnya selain *MAC* adalah *hash*.

Banyak yang menyamakan antara *hash* dan *MAC*. Bahkan dari referensi lain disebutkan bahwa *MAC* merupakan bagian dari hash (dimana bagian lainnya adalah *MDC* atau *Manipulation Detection Code*). Namun jika dilihat dari fungsinya, ketiganya sama-sama mengkompresi isi dokumen menjadi suatu ringkasan unik dengan panjang tertentu sehingga dapat menjamin integritas pesan yang terkirim.

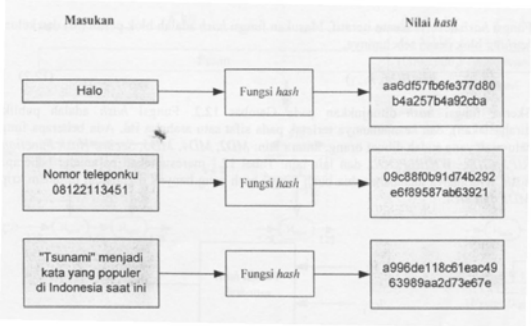
Fungsi *hash* adalah fungsi matematika yang mengkompresi dokumen yang panjangnya tidak menentu, menjadi sebuah pesan pendek dengan panjang tertentu.

Atau secara matematis dapat dinyatakan sebagai berikut:

$$h = H(x) \quad \text{...[2]}$$

Keterangan:
- H: fungsi *hash* untuk mengambil intisari/rangkuman dari *x*
- x: dokumen yang intisari nya ingin dibangkitkan

- h : nilai *hash* atau intisari yang dihasilkan



Gambar 2. Contoh nilai *hash* dari pesan yang panjangnya berbeda-beda [MUN06]

Fungsi *hash* satu arah (*one way hash*) adalah fungsi *hash* yang bekerja dalam satu arah: pesan yang sudah diubah menjadi *message digest* tidak dapat dikembalikan lagi menjadi pesan semula. Dua pesan yang berbeda akan selalu menghasilkan nilai *hash* yang berbeda pula. Sifat-sifat fungsi *hash* satu arah adalah sebagai berikut [SCH98]:

1. Fungsi H dapat diterapkan pada blok data berukuran berapa saja
2. H menghasilkan nilai (h) dengan panjang tetap (*fixed-length output*)
3. H(x) mudah dihitung untuk setiap nilai x yang diberikan
4. Untuk setiap h yang diberikan, tidak mungkin menemukan x sedemikian sehingga H(x) = h. Itulah sebabnya fungsi H dikatakan fungsi *hash* satu-arah (*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).

Keenam sifat di atas penting sebab sebuah fungsi hash seharusnya berlaku seperti fungsi acak. Sebuah fungsi *hash* dianggap tidak aman jika secara komputasi dimungkinkan menemukan pesan yang bersesuaian dengan pesan ringkasnya, dan terjadi kolisi, yaitu terdapat beberapa pesan berbeda yang mempunyai pesan ringkas yang sama.

Tidak seperti *MAC*, parameter fungsi *hash* hanya 1 masukan, yaitu dokumen. Karena perbedaan *MAC* dan *hash* hanya terletak pada penggunaan kunci rahasia, makanya *MAC* sering disebut juga dengan *keyed-hash function*, atau fungsi *hash* berkunci.

Kelebihan *MAC* dibanding hash adalah hanya pihak-pihak yang berkepentingan saja yang bisa membangkitkan *message digest MAC* karena hanya mereka lah yang mengetahui kunci rahasia. Sedangkan *message digest hash* dapat dibangkitkan oleh siapapun (virus sekalipun) jika ia tau fungsi *hash* yang digunakan.

4. Perbedaan *MAC* dan tanda tangan *digital*

Sebuah tanda tangan *digital* (*digital signature)* merupakan cipherteks dari nilai *hash* dokumen. Di sisi pengirim, dokumen ditanda tangan dengan menggunakan kunci privat pengirim. Sedangkan di sisi penerima, dokumen diverifikasi dengan menggunakan kunci publik pengirim.

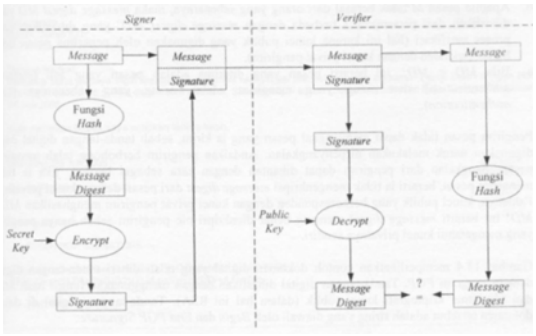
Jika proses verifikasi sukses, itu berarti pengirim dan penerima memang pihak yang berhak serta pesan tidak dimanipulasi selama transmisi.

Dari pengertian di atas, digital signature tampak sama dengan *MAC*, kecuali kunci yang digunakan. Tanda tangan *digital* dikomputasi dan diverifikasi dengan menggunakan dua buah kunci, kunci publik dan kunci privat.

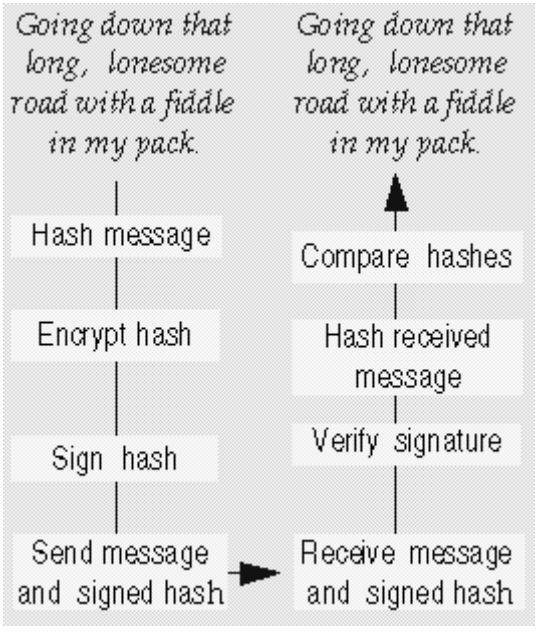
Pada **Gambar 3 dan 4** dapat dilihat bahwa nilai *hash* merupakan hasil enkripsi yang kemudian dilekatkan (ditandatangankan) sehingga penerima dapat memverifikasi tanda tangan yang ia dapatkan dengan mendekripsikannya menggunakan kunci publik pengirim lalu dibandingkan dengan

nilai *hash* yang bisa ia komputasikan dari pesan asli.

Jadi yang dibutuhkan untuk membangun sebuah tanda tangan *digital* adalah fungsi *hash* (seperti MD5, SHA-1, dan lain-lain) dan algoritma kunci publik (seperti RSA, ElGamal, dan lain-lain).



Gambar 3. Otentikasi dengan tanda tangan *digital*



Gambar 4. Prosedur pemberian tanda tangan *digital*

Hal ini tentu berbeda dengan *MAC* yang hanya membutuhkan salah satu di antaranya (alih-alih algoritma kunci publik, *MAC* menggunakan algoritma kunci simetri, yang akan dijelaskan lebih lanjut pada poin 5 di bawah mengenai pendekatan algoritma *MAC*).

MAC bukan tanda tangan *digital* karena tidak dapat digunakan untuk menjamin keamanan nirpenyangkalan.

MAC tidak dapat menghasilkan kepercayaan (trust) yang penuh antara pengirim dan penerima. Tiga aspek keamanan di dalam kriptiografi (integritas data, otentikasi dan nirpenyangkalan) dapat diselesaikan dengan menggunakan tanda tangan *digital*. Pesan ditandatangani oleh pengirim dan tanda tangan diverifikasi oleh penerima. Pesan yang sudah ditandatangani menunjukkan bahwa pesan tersebut otentik (baik otentik isi maupun otentik pengirim) dan tidak dapat disangkal [MUN06].

5. Pendekatan algoritma *MAC*

Ada 3 pendekatan dalam merancang sebuah algoritma *MAC*, yaitu: (1) berbasis fungsi *hash*, (2) berbasis *cipher* aliran, atau (3) berbasis *cipher* blok.

1. Algoritma MAC berbasis fungsi hash

Pendekatan *hash* dilakukan dengan mengganti fungsi $G_k(x)$ pada *MAC* dengan fungsi $H(x)$ pada *hash*, namun dengan tambahan kunci k yang ikut beroperasi di dalam fungsinya. Algoritma ini disebut juga dengan *HMAC (hash function-based MAC*).

Ukuran kunci yang digunakan dalam *HMAC* harus lebih besar atau sama dengan $L/2$, dimana L adalah ukuran nilai *hash* yang dihasilkan. Untuk aplikasi yang membolehkan kunci k lebih besar dari ukuran dokumen, maka k harus di-*hash* terlebih dahulu [NAT02].

Algoritma *MAC* berbasis fungsi *hash* sangat luas digunakan adalah yang menggunakan fungsi *hash* MD5.

2. Algoritma MAC berbasis cipher aliran

Pendekatan cipher aliran didasarkan pada operasi bit, yaitu mengambil sebagian bit dari dokumen untuk dienkripsi menjadi *MAC*.

Contohnya saja algoritma *MAC* berbasis *cipher* aliran yang dirancang oleh peneliti RSA *Laboratory*, yang mana algoritmanya menghasilkan bit *MAC* sebanyak setengah dari jumlah bit dokumen asli. Secara umum, langkah-langkah nya adalah membagi dokumen menjadi dua, lalu masing-masing dilakukan operasi LFSR baru kemudian keduanya digabungkan melalui proses tertentu.

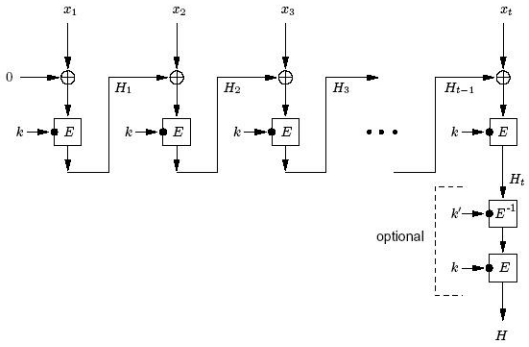
Selain itu, tujuan dari makalah ini adalah mencoba merancang suatu algoritma MAC berbasis cipher aliran, yang rancangannya akan dijelaskan lebih lanjut pada poin 7.

3. Algoritma MAC berbasis cipher blok

Pendekatan *cipher* blok biasanya menggunakan mode *Cipher Block Chaining (CBC)*, yaitu mode yang menghasilkan banyak plainteks yang saling bergantung hingga akhirnya terbentuk sebuah cipherteks.

Karena operasinya per blok, maka bagian dokumen yang diambil adalah per blok, apakah itu blok terakhir, pertama, dan sebagainya. Blok inilah yang kemudian akan dienkripsi dengan menggunakan *cipher* blok seperti DES (*Data Encryption Standard*), RC5, RC6, AES (*Advanced Encryption Standard*), dan lain-lain.

Skema pada **Gambar 5** di bahwa menunjukkan langkah-langkah pembangkitan *MAC*. Awalnya isi *file* di-XOR[1]-kan dengan 0 dan hasil XOR tersebut dienkripsi menggunakan kunci *k* dan fungsi *E*. Hasil enkripsi ini di-XOR-kan dengan isi *file* dan hasil XOR tersebut dienkripsi menggunakan kunci *k* dan fungsi *E*, begitu seterusnya hingga pemrosesan terakhir akan diambil blok tertentu untuk dijadikan sebagai *MAC*.



Gambar 5. Skema *MAC* berbasis *cipher* blok mode *CBC*

Algoritma *MAC* berbasis *cipher* blok yang sangat luas digunakan dan juga sebagai standar internasional adalah yang menggunakan algoritma *cipher DES*.

6. Perbandingan pendekatan algoritma *MAC*

Penggunaan fungsi *hash* bisa dibilang lebih rumit dibanding menggunakan algoritma *cipher*. Alasannya adalah karena fungsi *hash* yang telah umum (seperti MD5, SHA-1, dan lain-lain) harus diubah dulu karena harus mengikutsertakan kunci simetri *k*. Sedangkan pada algoritma *cipher*, algoritmanya sendiri tidak perlu diubah. Yang perlu dipikirkan hanya bagian mana dari dokumen yang akan dienkripsi untuk dijadikan *MAC*.

[1] Exclusive OR

Pada *cipher* blok, jika enkripsi dilakukan hanya sekali (tidak menggunakan mode *CBC*), maka kemungkinan besar akan terdapat hasil *MAC* yang sama untuk beberapa dokumen, jika blok yang dienkripsi sama. Makanya pada *MAC* berbasis *cipher* blok dibutuhkan algoritma yang rumit seperti *CBC* yang harus dienkripsi berkali-kali.

Sedangkan dengan *cipher* aliran, kemungkinan menghasilkan *MAC* yang sama sangat kecil, karena bit-bit *MAC* terdiri dari bit-bit pesan yang tersebar di seluruh dokumen, yang membuatnya sangat unik.

Selain itu, ada juga pendekatan lain yang digunakan dalam membangun algoritma MAC, yaitu menggunakan teknik kriptografi kuno namun tidak dapat terpecahkan, yaitu *one time pad* yang diperkenalkan oleh Simmons and Stinson pada tahun 1995. *MAC* dibangkitkan dengan menggunakan *one time pad* yang disimpan di suatu program. Kunci ini hanya diketahui oleh pihak yang menyimpan *one time pad* tersebut ke dalam program.

Untuk mengecek integritas pesan, penerima harus menggunakan program yang didalamnya terdapat *one time pad* yang sama. Hal ini akan menjadi redundansi apabila program tersebut harus dikirimkan bersamaan dengan pesan (misalnya hanya pengirim yang mengetahui *one time pad* itu).

Jika telah selesai digunakan, kunci ini langsung dimusnahkan agar tidak digunakan untuk mengenkripsi pesan lain. Cara ini sangat aman karena hanya program (dan pengirim pesan) yang mengetahui kunci dan program dapat membangkitkan *MAC* secara otomatis. Namun, *effort* yang dikeluarkan tidak sebanding dengan keamanan yang didapat apalagi jika dibandingkan dengan aplikasi *MAC* lain yang lebih sederhana.

Lagipula, ada cara lain yang lebih sederhana untuk memperoleh tingkat keamanan seperti itu. Yaitu dengan menggunakan kunci rahasia *k* yang sekali pakai (*one time key*).

7. Algoritma *MAC* berbasis *cipher* aliran

Ada banyak sekali algoritma *MAC* berbasis *cipher* aliran yang dibangun sampai saat ini, di antaranya *Tail-MAC: An Efficient Message Authentication Scheme for Stream Ciphers (2004), VMPC-MAC: A Stream Cipher Based Authenticated Encryption Scheme - Zoltak (2004), Salsa20 with Poly1305*, dan algoritma MAC berbasis *cipher* aliran yang dirancang oleh peneliti-peneliti dari RSA *Laboratory* [RSA05].

Algoritma-algoritma di atas memiliki kelebihan dan kekurangannya masing-masing. Berikut akan dijelaskan sebuah rancangan algoritma *MAC* yang sangat sederhana, berbasis *cipher* aliran RC4. Algoritma/program ini diberi nama **MARC**.

RC4 adalah algoritma *cipher* aliran yang didasarkan pada kegunaan dari hasil permutasi acak dengan menggunakan variabel berisi ukuran kunci. Langkah-langkah nya adalah sebagai berikut:

1. Melakukan inisialisasi pada tabel S satu dimensi:

```
S[0] = 0
S[1] = 1
    :
    :   ...dst
    :
S[255] = 255

for i← 0 to 255 do
     S[i] ← i
endfor
```

2. Jika panjang kunci U (masukan dari *user*) < 256 karakter (*byte*), kunci tersebut di-*padding*, yaitu penambahan bit-bit isian pada akhir kunci. Contoh: kunci dari user adalah `'kripto'`. Karena kurang dari 256 *byte*, maka kunci tersebut di-*padding* menjadi

`'kriptokriptokripto.....'`
hingga 256 karakter.

3. Melakukan permutasi terhadap setiap nilai tabel S:

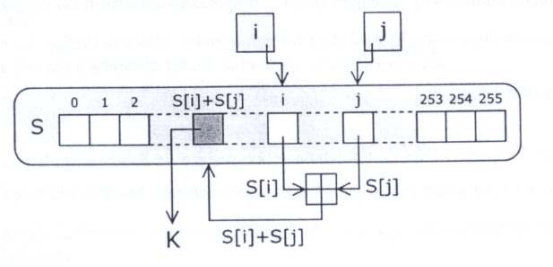
```
j ← 0
for i ← 0 to 255 do
    j←(j+ S[i] + U[i]) mod 256
    swap(S[i],S[j])
endfor
```

Dimana fungsi `swap` adalah proses pertukaran nilai `S[i]` dan `S[j]`.

4. Membangkitkan aliran kunci (*keystream*) dan mengenkripsi plainteks dengan *keystream* tersebut:

```
i ← 0
j ← 0
for        idx←0        to PanjangPlainteks – 1 do
    i ← (i+ 1) mod 256
    j ← (j+S[i]) mod 256
    swap(S[i],S[j])
    t ← (S[i]+S[j]) mod 256
    K ← S[t]
    C ← K XOR P[idx]
endfor
```

Keterangan:
`P` : *array* of karakter plainteks
`K` : *keystream* yang dibangkitkan
`C` : cipherteks



Gambar 6. Pembangkitan kunci aliran K pada RC4

Proses pembangkitan kunci aliran K dalam algoritma RC4 ini dapat dilihat lebih jelas pada **Gambar 6** di atas. Pada *cipher* aliran, semakin acak fungsi pembangkitan kunci alirannya, maka akan semakin baik pula performansi *cipher*-nya. Hal itu disebabkan karena algoritma menjadi tidak mudah dipahami oleh penyerang [AND95].

Seperti terlihat pada langkah 4 algoritma RC4 di atas, yang dienkripsi adalah keseluruhan plainteks. Sedangkan *MAC* hanya berupa intisari dari pesan, makanya yang akan dienkripsi hanya sebagian bit plainteks.

Satu *byte* terdiri dari 8 bit, dimana bit pertama selalu 1 dan bit terakhir (ke-8) biasanya 0. Maka, untuk mendapatkan sifat keunikan dari *MAC* yang dihasilkan, bit yang akan diambil adalah bit yang berada di tengah saja. Dalam hal ini, **MARC** mengambil bit ke-4 (skala 0-7) untuk setiap *byte* dokumen.

Jadi, panjang *MAC* yang akan didapat (dalam bit) adalah panjang plainteks (dalam bit) dibagi 8. Atau dengan kata lain banyaknya *byte* dari plainteks tersebut. Jika hasil pembagiannya tidak genap (bersisa), maka kumpulan bit yang tidak mencapai 1 *byte* itu di-padding dengan `0 (false)` hingga genap 1 *byte*.

Contohnya, BitArray dari dokumen adalah 010**0**1011 000**1**0011 101**0**1111. Maka, bit-bit yang diambil dari dokumen adalah 010. Karena bit yang terkumpul kurang dari 8 buah (tidak mencapai 1 *byte*), maka bit-bit tersebut di-padding menjadi **00000**010. Bit ini dienkripsi dengan K yang dibangkitkan menggunakan algoritma RC4.

Untuk lebih jelasnya, algoritma **MARC** adalah sebagai berikut:

```
function buildMac(byte[]
bits, int[] S) → byte[]

{
BitArray bi, ba, tmp
byte[] C
div ← panjang_bits / 8

if (panjang_bits mod 8 = 0)
then
    panjangC ← div
else
    panjangC ← div + 1

i ← 0
j ← 0
idx ← 0
l ← 0

while (idx < panjang_bits)
do
   i ← (i + 1) mod 256
   j ← (j + S[i]) mod 256
   ba←IntToBin(ToInt(bits[idx
   ]))
   tmp[l] ← ba[4]

   if (idx = panjang_bits - 1)
   then
      while (l < 7)
      do
            l++
            tmp(l) ← false

   if (l = 7)
   then
      swap(S[i],S[j])
      t ← (S[i] + S[j]) mod
   256
      K ← S[t]
      bi ← IntToBin(K)
      C[m]←ToByte(BinToInt(bi
   XOR tmp))
      m++
      l ← 0

   l++

return C
}
```

Keterangan:

- byte[] : tipe *array of byte*
- tmp : tempat menampung sementara bit-bit yang dikumpulkan dari bit awal tiap *byte*, bertipe *array of bit*
- K : *keystream* yang dibangkitkan
- C : *MAC* yang dihasilkan
- IntToBin : fungsi yang mengubah *integer* menjadi *BitArray*
- BinToInt : fungsi yang mengubah *BitArray* menjadi *integer*
- ToByte : fungsi yang mengubah parameter masukan (tipe *integer*) menjadi tipe *byte*
- ToInt : fungsi yang mengubah parameter masukan (tipe *byte*) menjadi tipe *integer*

Penggunaan RC4 sebagai algoritma *cipher* aliran pada pembangunan *MAC* ini disebabkan karena kekuatannya. Sampai saat ini *cipher* RC4 belum bisa dipecahkan kecuali menggunakan *exhaustive search (brute force)*[2]. Batas kunci RC4 yang bisa dipecahkan adalah 40 bit [SAF06].

8. Implementasi *MAC* berbasis *cipher* aliran

MARC adalah algoritma *MAC* berbasis RC4. Untuk menyediakan layanan otentikasi dan integritas, sebuah program *MAC* setidaknya harus menyediakan 2 bagian, yaitu membangkitkan *MAC* dari dokumen dan melekatkannya sehingga siap untuk dikirim, dan mengecek *MAC* dari dokumen tersebut.

Pada **MARC**, dua fungsi itu diwujudkan dalam menu **Give a MAC** dan

[2] *Exhaustive key search* atau *brute force search* adalah suatu teknik dasar yang digunakan kriptoanalis untuk mencoba setiap kunci yang mungkin sampai ditemukan kunci yang sebenarnya.

Authentication. Untuk lebih jelasnya, dapat dilihat pada **Gambar 7** di bawah ini.

Gambar 7. Dua menu utama pada program MARC

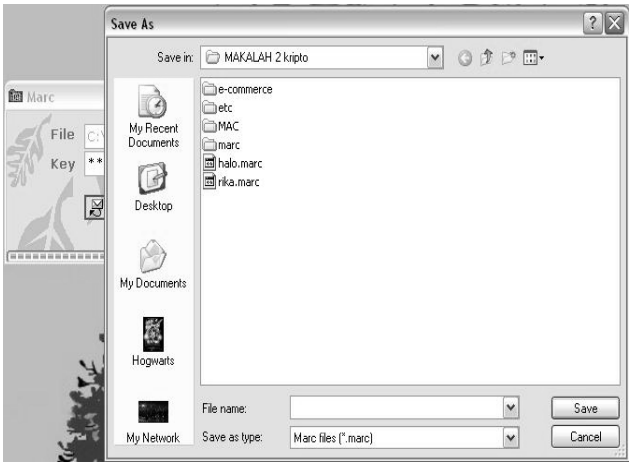
Gambar 8. Pilihan bantuan pada aplikasi MARC

1. Give a MAC

Dengan masukan *file* (dokumen) dan *key* (kunci rahasia), menu ini membangkitkan *MAC* dari *file* dengan menggunakan algoritma **MARC** yang telah dijelaskan pada bagian sebelumnya, melekatkannya pada isi *file* dengan menambahkan panjang isi *file* asli setelah MAC, lalu menyimpan hasilnya yang berupa sebuah *file* **MARC** (lihat **Gambar 9**!).

Jadi, susunan dari sebuah *file* **MARC** adalah:

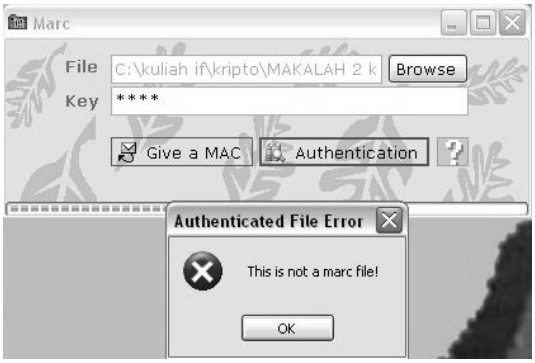
isi file + MAC + panjang isi file



Gambar 9. Menyimpan *file* MARC

2. Authentication

Masukannya sama dengan menu *Give a MAC*, yaitu *file* dan *key*. Hanya saja *file* masukannya lebih spesifik, yaitu yang berekstensi **MARC** (lihat **Gambar 10!**). *File* **MARC** yang terdiri dari isi *file* asli dan *MAC* yang dilekatkan, akan dipisahkan berdasarkan panjang yang tersimpan di *byte* terakhir *file* **MARC**. Maka akan didapatkan *file* asli dan *MAC*-nya.



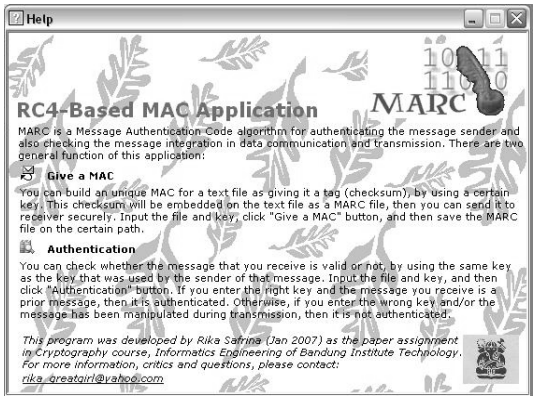
Gambar 10. Pengecekan *file* MARC

Kemudian dengan menggunakan *key*, *MAC* dari *file* asli dibangkitkan lalu dicocokkan dengan *MAC* yang tadi didapatkan dari hasil pemisahan file **MARC**. Jika cocok, akan muncul

window pop-up yang bertuliskan **"Authenticate!"** (lihat **Gambar 11** di bawah ini!). Sebaliknya jika tidak cocok, akan muncul *window pop-up* bertuliskan **"Not Authenticate!"** (lihat **Gambar 12** di bawah ini!).

Gambar 11. *File* MARC dan kunci yang cocok

Gambar 12. *File* MARC dan kunci yang tidak cocok



Gambar 13. Penjelasan MARC pada aplikasi

Program **MARC** dapat diunduh di http://students.if.itb.ac.id/~If13053/kriptografi.

9. Pengujian algoritma *MAC* berbasis *cipher* aliran

Pengujian meliputi otentikasi dengan kasus-kasus berikut:

1. Karakter di dalam teks dihapus, ditambah dan diubah

teks semula (beserta *MAC*):
`rika cantik sekalicèá`

teks dihapus:
`ika cantik sekalicèá`
hasil verifikasi:
`This is not a marc file!`

teks ditambah:
`hm rika cantik sekalicèá`
hasil verifikasi:
`This is not a marc file!`

teks diubah:
`lika cantik sekalicèá`
hasil verifikasi:
`Not Authenticate!`

Jika panjang pesan telah berubah, maka panjang pesan yang tersimpan di akhir *byte* tidak cocok dengan panjang pesan sekarang. Hal itu akan dikarakterisasikan sebagai bukan *file* **MARC**. Sedangkan jika panjang pesan sama namun karakter berubah, akan keluar *message* **Not Authenticate!**

2. Karakter di dalam MAC diubah

MAC semula ():
`cèá`

MAC diubah:
`aèá`
hasil verifikasi:
`Not Authenticate!`

3. Kunci simetri yang dimasukkan tidak valid

teks semula (beserta *MAC*):
`rika cantik sekalicèá`

Kunci simetri seharusnya:
`1234`
Kunci simetri yang dimasukkan untuk verifikasi:
`rikas`
Hasil verifikasi:
`Not Authenticate!`

4. MAC dihapus dari dokumen

hasil verifikasi terhadap dokumen tanpa MAC:
`This is not a marc file!`

10. Kesimpulan

Message Authentication Code (MAC) adalah intisari yang dihasilkan dari pemrosesan dokumen dan kunci rahasia tertentu menggunakan suatu algoritma otentikasi pesan (*message authentication algorithm*).

Bersama-sama dengan fungsi *hash*, aplikasi *message digest* ini menjamin integrasi pesan, yaitu mengetahui apakah pesan telah dimanipulasi saat transmisi.

Keamanan lainnya yang dijamin oleh MAC adalah otentikasi pengirim, yaitu memeriksa si pengirim pesan melalui kunci rahasia yang hanya diketahui oleh pengirim dan penerima pesan.

Tujuan umum *MAC* dan *digital signature* secara umum adalah sama, yaitu memberi tanda pengenal pada pesan. Namun, karena *digital signature* menggunakan kunci publik (dibanding *MAC* yang menggunakan kunci simetri), maka *digital signature* lebih unggul dalam menjamin keamanan nirpenyangkalan yang tidak disediakan oleh *MAC*.

Ada beberapa jenis pendekatan dalam membangun algoritma *MAC*, yaitu berbasis fungsi *hash*, berbasis *cipher* aliran dan berbasis *cipher* blok. Untuk yang ekstrim, dapat menggunakan teknik kriptografi *one time pad*.

Penggunaan fungsi *hash* lebih rumit dibanding menggunakan algoritma *cipher*. Hal itu disebabkan karena fungsi *hash* yang telah umum (seperti MD5, SHA-1, dan lain-lain) harus diubah terlebih dahulu karena harus mengikutsertakan kunci simetri *k* dalam komputasinya. Sedangkan pada algoritma *cipher*, algoritmanya sendiri tidak perlu diubah. Yang perlu dipikirkan hanya bagian mana dari dokumen yang akan dienkripsi untuk dijadikan *MAC*.

Pada *cipher* blok, jika enkripsi dilakukan hanya sekali (tidak menggunakan mode *CBC*), maka kemungkinan besar akan terdapat hasil *MAC* yang sama untuk beberapa dokumen, jika blok yang dienkripsi sama. Makanya pada *MAC* berbasis *cipher* blok dibutuhkan algoritma yang rumit seperti *CBC* yang harus dienkripsi berkali-kali.

Sedangkan dengan *cipher* aliran, kemungkinan menghasilkan *MAC* yang sama sangat kecil, karena bit-bit *MAC* terdiri dari bit-bit pesan yang tersebar di seluruh dokumen, yang membuatnya sangat unik.

Selain itu, *cipher* ini sangat cocok untuk aplikasi yang melakukan pemrosesan data per bit nya, mungkin dikarenakan untuk menghemat isi memori atau *buffer* [MEN96].

Dengan kelebihan *cipher* aliran seperti itu, maka penulis mencoba merancang suatu algoritma *MAC* berbasis *cipher* aliran. Algoritma *cipher* aliran yang dipilih adalah RC4 karena unggul dalam mencegah serangan.

Satu *byte* terdiri dari 8 bit, dimana bit pertama selalu 1 dan bit terakhir (ke-8) biasanya 0. Maka, untuk mendapatkan sifat keunikan dari *MAC* yang dihasilkan, bit yang akan diambil adalah bit yang berada di tengah saja. Dalam hal ini, **MARC**

mengambil bit ke-4 (skala 0-7) untuk setiap *byte* dokumen.

Jadi, panjang *MAC* yang akan didapat (dalam bit) adalah panjang plainteks (dalam bit) dibagi 8. Atau dengan kata lain banyaknya *byte* dari plainteks tersebut. Jika hasil pembagiannya tidak genap (bersisa), maka kumpulan bit yang tidak mencapai 1 *byte* itu di-padding dengan `0 (false)` hingga genap 1 *byte*.

Contohnya, BitArray dari dokumen adalah 010**0**1011 000**1**0011 101**0**1111. Maka, bit-bit yang diambil dari dokumen adalah 010. Karena bit yang terkumpul kurang dari 8 buah (tidak mencapai 1 *byte*), maka bit-bit tersebut di-padding menjadi **00000**010. Bit ini dienkripsi dengan K yang dibangkitkan menggunakan algoritma RC4.

Aplikasi MARC terbukti efisien, cepat, sederhana dan dapat memverifikasi *text file* dengan tepat. Aplikasi ini dapat dikembangkan lebih lanjut untuk menangani *file-file* selain *file* teks, karena telah diberi pondasi dasar ke arah sana. Hanya tinggal *debugging* saja.

DAFTAR PUSTAKA

[SAF06] Safrina, Rika. 2006. *Studi dan Perbandingan Sistem Penyandian Pesan dengan Algoritma RC2, RC4, RC5 dan RC6*. Paper. Bandung: Program Studi Teknik Informatika, Institut Teknologi Bandung.

[MUN06] Munir, Rinaldi. 2006. *Diktat Kuliah IF5054 Kriptografi*. Bandung: Program Studi Teknik Informatika, Institut Teknologi Bandung.

[MEY82] Meyer, Carl H., dan Matyas, Stephen M. 1982. *Cryptography: A New Dimension in Computer Data Security*. New York: John Wiley & Sons.

[RSA05] RSA Securiti, Inc. 2005. *RSA Laboratories*. http://www.rsasecurity.com/. [14 Desember 2006].

[GUR03] Guritman, Sugi., dkk. 2003. *Algoritma Blowfish untuk Penyandian Pesan*. Jurnal Ilmiah Ilmu Komputer vol.1 no.1, edisi September 2003, hal 9-19. Bogor: Institut Pertanian Bogor.

[RIV03] Rivest, Ronald L.,. *Ronald L. Rivest's Homepage*. http://theory.csail.mit.edu/%7Erivest/homepage.html. [28 September 2006].

[SCH98] Schneier, Bruce., dkk. 1998. *Related-Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA*. Paper.

[KRE00] KremlinEncrypt. 2000. *Concepts of Cryptography*. http://www.kremlinencrypt.com/crypto.html. [28 September 2006].

[SMA05] SmartComputing. 2005. *RC2, RC4, RC5 and RC6*. USA: Sandhills Publishing Company. http://www.smartcomputing.com. [28 September 2006].

[RIV98] Rivest, Ronald L., dkk. 1998. *The RC4 Stream Cipher*. Paper.

[SUP03] Suprapti, Iswanti. 2003. *Studi Sistem Keamanan Data dengan Metode Public Key Cryptography*. Paper.

[MEN96] Menezes, A., dkk. 1996. *Handbook of Applied Cryptography*. CRC Press, Inc.

[IBM05] IBM. 2005. *IBM Cryptographic Toolkits*. http://www.ibm.com [28 September 2003].

[SUR05] Surfpack. 2005. *File Encryption for Your Important Files*.

http://www.surfpack.com [28 September 2003].

[AND95] Anderson, R. J. 1995. *A Faster Attack on Certain Stream Cipher*. Cambridge: University Computer Laboatory.

[NAT02] National Institute of Standards and Technology. 2002. *The Keyed-Hash Message Authentication Code (HMAC)*. Federal Information Processing Standards Publication. USA: NIST.