PERBANDINGAN ALGORITMA MARS DAN RIJNDAEL DALAM BEBERAPA MODE OPERASI CIPHER BLOK

Muhammad Bahari Ilmy – NIM : 13504062

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : bahari135@students.itb.ac.id

Abstrak

Makalah ini membahas tentang perbandingan dua algoritma enkripsi cipher block yang menjadi kandidat *Advanced Encryption Standard* (AES), yaitu MARS dan Rijndael yang akhirnya menjadi *AES*. MARS merupakan algoritma kunci simetri dengan 128-bit cipher block dan memiliki variasi panjang kunci diantara (128 s.d. 1248 bit). Sedangkan Rijndael merupakan sebuah algoritma kriptografi simetri yang beroperasi dalam bentuk blok 128-bit. Rijndael mendukung panjang kunci 128-bit, 192-bit, dan 256-bit.

Baik MARS maupun Rinjdael dibuat untuk memenuhi kebutuhan dalam keamanan data, terutama sebagai salah satu jawaban atas kurang memuaskannya *DES(Data Encryption Standard).* Kedua algoritma ini memiliki sejumlah teknik dan operasi yang hampir sama secara prinsip, karena pada dasarnya untuk algoritma kunci simetri memiliki sejumlah operasi yang standard, seperti substitusi dan transposisi. Namun dari segi kompleksitas ruang, waktu, dan tingkat keamanan terlihat adanya perbedaan. Itulah yang menjadi dasar perbandingan yang akan diulas pada makalah ini.

Sebuah perangkat lunak bernama *DEC V* yang dibuat oleh *Delphi Encryption Compendium* digunakan sebagai alat bantu untuk membandingkan kedua algoritma. *DEC V* dikembangkan dengan menggunakan kakas Borland Delphi dan lingkungan pengembangan berbasis Windows. Perangkat lunak ini menyediakan banyak pilihan algoritma kriptografi serta mendukung penyandian untuk segala jenis arsip.

Pengujian dan perbandingan dilakukan dengan menggunakan beberapa mode operasi cipher blok. Yakni menggunakan 3 mode cipher blok, yaitu mode operasi *ECB(Electronic Code Book), CBC (Cipher Block Chaining), CFB(Cipher Feedback),* dan *OFB(Output Feedback).*

Perbandingan pun dilakukan dengan menggunakan teknik *white box* . Perbandingan dilakukan dengan menganalisis operasi-operasi yang diterapkan dari berbagai sudut pandang. Keunggulan serta kelemahannya.

***Kata kunci**:* MARS, Rijndael,.*Advanced Encryption Standard, DEC* V,enkripsi, dekripsi, *Electronic Code Book, Cipher Block Chaining, Cipher Feedback, Output Feedback.*

I. Pendahuluan

Teknologi informasi masa kini sudah sangat mendukung pengiriman data dan informasi dari belahan dunia manapun, tidak lagi terbatas dengan ruang dan waktu. Informasi yang ada pada suatu belahan dunia dapat diakses dari belahan dunia manapun selama terhubung melalui *network* atau yang sekarang dikenal dengan *global network* atau *internet.* Transfer data dan informasi melalui internet sekarang ini sudah menjadi bagian dari kehidupan kita.

Transfer data dan informasi dalam media elektronik tentunya harus dilengkapi dengan tingkat keamanan akan hal tersebut. Apalagi bila informasi yang dikirimkan bersifat rahasia dan menyangkut kepentingan yang cukup besar. Oleh karena itu tentu diperlukan suatu proses dalam media elektronik tersebut yang dapat menjamin keamanan data dan informasi yang akan ditransfer. Salah satu bidang informatika atau ilmu komputer yang menangani hal itu adalah kriptografi.

Perkembangan bidang kriptografi sekarang ini sudah semakin pesat. Berbagai macam jenis algoritma dikembangkan, namun terdapat dua jenis algoritma yang berkembang cukup baik menurut tipe kunci yang digunakan, yaitu algoritma kunci simetri dan algoritma kunci publik. Keduanya memiliki kelebihan dan kekurangan masing-masing dalam mengamankan sebuah data atau informasi.

Algoritma kunci simetri berkembang cukup pesat menarik perhatian masyarakat banyak. *DES* (*Data Encyption Standard*) yang menjadi algoritma standard pertama yang digunakan untuk mengenkripsi data. *DES* beroperasi pada ukuran blok 64 bit. *DES* mengenkripsikan 64 bit plainteks menjadi 64 bit chiperteks dengan menggunakan 56 bit internal (*internal key*) atau upa-kunci (*subkey*). Kunci internal dibangkitkan dari kunci eksternal (*eksternal key*) yang panjangnya 64 bit.

Namun, *DES* dianggap tidak aman lagi, penyebabnya karena panjang kunci yang hanya 56 bit. Pada rancangan awal, panjang kunci yang diusulkan IBM adalah 128 bit, tetapi atas permintaan NSA, panjang kunci diperkecil menjadi 56 bit. Serangan yang mungkin terhadap *DES* adalah dengan *exhaustive key search*. Dengan panjang kunci 56 bit akan terdapat 2^{56} atau 72.057.594.037.927.936 kemungkinan kunci. Jika diasumsikan serangan *exhaustive key search* dengan menggunakan prosesor paralel mencoba setengah dari jumlah kemungkinan kunci itu, maka dalam satu detik dapat dilakukan satu juta serangan. Jadi total diperlukan 1142 tahun untuk menemukan kunci yang benar.

Namun, pada tahun 1998 *Electronic Frontier Foundation (EEF)* merancang dan membuat perangkat keras untuk menemukan kunci *DES* secara *exhaustive key search* dengan biaya $250.000 dan diharapkan dapat menemukan kunci kurang dari 5 hari. Tahun 1999, kombinasi perangkat keras *EEF* dengan kolaborasi internet yang melibatkan lebih dari 100.000 komputer dapat menemukan kunci *DES* kurang dari 1 hari.
Menanggapi hal tersebut *National Institute of Standards and Technology (NIST)*, sebagai agensi Departemen Perdagangan Amerika Serikat mengusulkan kepada pemerintah Federal AS untuk sebuah standard kriptografi yang baru. Kemudian NIST mengadakan sayembara terbuka untuk membuat standard algoritma kriptografi yang baru pengganti *DES*. Standard tersebut kelak diberi nama *Advanced Encryption Standard* (*AES*).

Persyaratan yang diajukan oleh *NIST* tentang algoritma yang baru tersebut adalah :

1. Algoritma termasuk ke dalam kelompok algoritma kunci simetri berbasis *cipher* blok.
2. Seluruh rancangan algoritma tidak dirahasiakan
3. Panjang kunci yang fleksibel : 128, 192, dan 256 bit.
4. Ukuran blok yang dienkripsi adalah 128 bit.
5. Algoritma dapat diimplementasikan dalam perangkat keras maupun perangkat lunak.

NIST menerima 15 proposal algoritma yang masuk. Konferensi umum pun diselenggarakan untuk menilai keamanan algoritma yang diusulkan. Pada bulan Agustus 1998, *NIST* memilih 5 finalis yang didasarkan pada aspek keamanan algoritma, kemangkusan (*efficiency*), fleksibilitas, dan kebutuhan memori (penting untuk *embedded system*). Finalis tersebut adalah *Rijndael, RC6, MARS, Twofish, Serpent*.

Masing-masing dari kelima kandidat di atas tentu memiliki kelebihan dan kekurangan masing-masing sebagai algoritma *cipher* blok, sehingga *NIST* memutuskan bahwa Rijndael yang menjadi *AES*.

2. Kriptografi

Kriptografi berasal dari dua kata Yunani, yaitu *Crypto* yang berarti rahasia dan *Grapho* yang berarti menulis. Secara umum kriptografi dapat diartikan sebagai ilmu dan seni penyandian yang bertujuan untuk menjaga keamanan dan kerahasiaan suatu pesan. Kriptografi pada dasarnya sudah dikenal sejak lama. Menurut catatan sejarah, kriptografi sudah digunakan oleh bangsa Mesir sejak 4000 tahun yang lalu oleh raja-raja Mesir pada saat perang untuk mengirimkan pesan rahasia kepada panglima perangnya melalui kurir-kurinya. Orang yang melakukan penyandian ini disebut *kriptografer*, sedangkan orang yang mendalami ilmu dan seni dalam membuka atau memecahkan suatu algoritma kriptografi tanpa harus mengetahui kuncinya disebut *kriptanalis*.

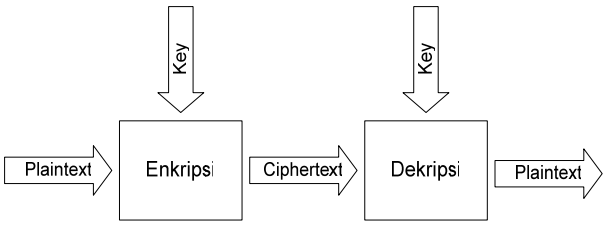
Seiring dengan perkembangan teknologi, algoritma kriptografi pun mulai berubah menuju ke arah algoritma kriptografi yang lebih rumit dan kompleks. Kriptografi mau tidak mau harus

diakui mempunyai peranan yang paling penting dalam peperangan sehingga algoritma kriptografi berkembang cukup pesat pada saat Perang Dunia I dan Perang Dunia II. Menurut catatan sejarah, terdapat beberapa algoritma kriptografi yang pernah digunakan dalam peperangan, diantaranya adalah ADFVGX yang dipakai oleh Jerman pada Perang Dunia I, Sigaba/M-134 yang digunakan oleh Amerika Serikat pada Perang Dunia II, Typex oleh Inggris, dan Purple oleh Jepang. Selain itu Jerman juga mempunyai mesin legendaris yang dipakai untuk memecahkan sandi yang dikirim oleh pihak musuh dalam peperangan yaitu, Enigma.

Algoritma yang baik tidak ditentukan oleh kerumitan dalam mengolah data atau pesan yang disampaikan. Yang penting, aklgoritma tersebut harus memenuhi 4 persyaratan berikut :

1. **Kerahasiaan**. Pesan (*plaintext*) hanya dapat dibaca oleh pihak yang memliki kewenangan.
2. **Autentikasi**. Pengirim pesan harus dapat diidentifikasi dengan pasti, penyusup harus dipastikan tidak bisa berpura-pura menjadi orang lain.
3. **Integritas**. Penerima pesan harus dapat memastikan bahwa pesan yang dia terima tidak dimodifikasi ketika sedang dalam proses transmisi data.
4. ***Non-Repudiation***. Pengirim pesan harus tidak bisa menyangkal pesan yang dia kirimkan.

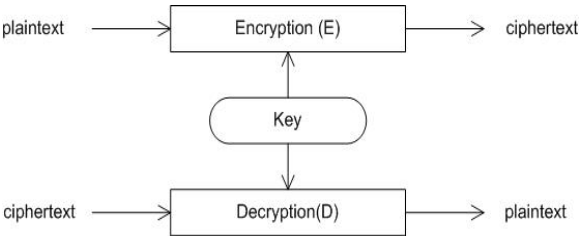
Kriptografi pada dasarnya terdiri dari dua proses, yaitu proses enkripsi dan proses dekripsi. Proses enkripsi adalah proses penyandian pesan terbuka menjadi pesan rahasia (*ciphertext*). *Ciphertext* inilah yang nantinya akan dikirimkan melalui saluran komunikasi terbuka. Pada saat *ciphertext* diterima oleh penerima pesan, maka pesan rahasia tersebut diubah lagi menjadi pesan terbuka melalui proses dekripsi sehingga pesan tadi dapat dibaca kembali oleh penerima pesan. Secara umum, proses enkripsi dan dekripsi dapat digambarkan sebagai berikut :



Gambar 1
Proses Enkripsi dan Dekripsi

2.1 Kriptografi *Secret Key* (Kunci simetri)

Kedua algoritma baik MARS maupun Rijndael merupakan algoritma kunci simetri (*secret key*). Kriptografi *secret key* adalah kriptografi yang hanya melibatkan satu kunci dalam proses enkripsi dan dekripsi. Proses dekripsi dalam kriptografi *secret key* ini adalah kebalikan dari proses enkripsi.



Gambar 2 Kriptografi simetris

Kriptografi *secret key* seringkali disebut sebagai kriptografi konvensional atau kriptografi simetris (*Symmetric Cryptography*) dimana proses dekripsi adalah kebalikan dari proses enkripsi dan menggunakan kunci yang sama.

Kriptografi simetris dapat dibagi menjadi dua, yaitu penyandian blok dan penyandian alir. Penyandian blok bekerja pada suatu data yang terkelompok menjadi blok-blok data atau kelompok data dengan panjang data yang telah ditentukan. Pada penyandian blok, data yang masuk akan dipecah-pecah menjadi blok data yang telah ditentukan ukurannya. Penyandian alir bekerja pada suatu data bit tunggal atau terkadang dalam satu byte. Jadi format data yang mengalami proses enkripsi dan dekripsi adalah berupa aliran bit-bit data.

Algoritma yang ada pada saat ini kebanyakan bekerja untuk penyandian blok karena kebanyakan proses pengiriman data pada saat ini menggunakan blok-blok data yang telah

ditentukan ukurannya untuk kemudian dikirim melalui saluran komunikasi.

Kelemahan dari algoritma jenis ini diantaranya kunci harus ditransfer lepada penerima kunci melalui jalar komunikasi yang aman. Biasanya jalar seperti itu memiliki harga yang mahal.

3. Cipher Blok

Penyandian blok pada dasarnya adalah proses penyandian terhadap blok data yang jumlahnya sudah ditentukan. Untuk sistem penyandian blok terdapat empat jenis mode operasi, yaitu *Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback(CFB), Output Feedback (OFB).*

3.1 *Electronic Code Book* (ECB*)*

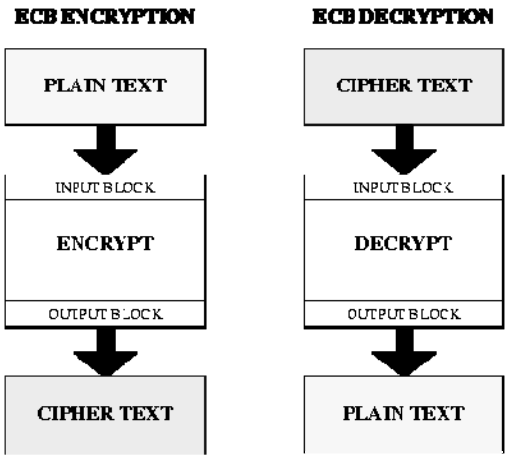
Mode ECB adalah mode yang paling umum dan paling mudah untuk diimplementasikan. Cara yang digunakan adalah dengan membagi data ke dalam blok-blok data terlebih dahulu yang besarnya sudah ditentukan. Blok-blok data inilah yang disebut *plaintext* karena blok data ini belum disandikan. Proses enkripsi akan langsung mengolah *plaintext* menjadi *ciphertext* tanpa melakukan operasi tambahan. Suatu blok *plaintext* yang dienkripsi dengan menggunakan kunci yang sama akan menghasilkan *ciphertext* yang sama. Secara matematis, enkripsi dengan mode *ECB* dinyatakan sebagai

$$C_i = E_k(P_i)$$

dan dekripsi sebagai

$$P_i = D_k(C_i)$$

yang dalam hal ini, P_i dan C_i masing-masing blok plainteks dan cipherteks ke-*i*.



Gambar 3
Mode operasi *Electronic Code Book* (ECB)

Keuntungan dari mode OBC ini adalah kemudahan dalam implementasi dan pengurangan resiko salahnya semua *plaintext* akibat kesalahan pada satu *plaintext*. Namun mode ini memiliki kelemahan pada aspek keamanannya. Dengan mengetahui pasangan *plaintext* dan *ciphertext*, seorang *kriptanalis* dapat menyusun suatu *code book* tanpa perlu mengetahui kuncinya.

3.2 *Cipher Block Chaining* (CBC)

Pada CBC digunakan operasi umpan balik atau dikenal dengan operasi berantai (*chaining*). Pada CBC, hasil enkripsi dari blok sebelumnya adalah *feedback* untuk enkripsi dan dekripsi pada blok berikutnya. Dengan kata lain, setiap blok *ciphertext* dipakai untuk memodifikasi proses enkripsi dan dekripsi pada blok berikutnya.

Mode ini menerapkan mekanisme umpan balik (*feedback*) pada sebuah blok, yang dalam hal ini hasil enkripsi blok sebelumnya diumpanbalikkan ke dalam enkripsi blok yang *current*. Caranya, blok plainteks yang *current* di-*XOR*-kan terlebih dahulu dengan blok cipherteks hasil enkripsi sebelumnya, selanjutnya hasil peng-*XOR*-an ini masuk ke dalam fungsi enkripsi. Dengan mode *CBC*, setiap blok cipherteks bergantung tidak hanya pada blok plainteksnya tetapi juga pada seluruh blok plainteks sebelumnya.

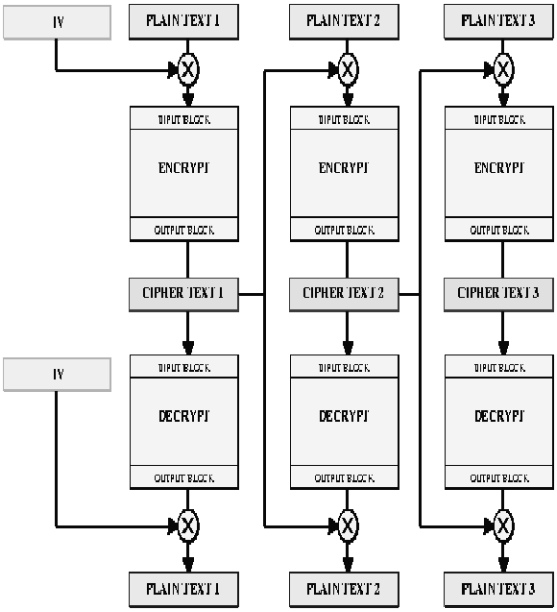
Secara matematis, enkripsi dengan mode *CBC* dinyatakan sebagai

$$C_i = E_k(P_1 \oplus C_{i-1})$$

dan dekripsi sebagai

$$P_i = D_k(C_i) \oplus C_{i-1}$$

Yang dalam hal ini, $C_0 = IV$ (*initialization vector*). *IV* dapat diberikan oleh pengguna atau dibangkitkan secara acak oleh program. Jadi, untuk menghasilkan blok cipherteks pertama (C_1), *IV* digunakan untuk menggantikan blok cipherteks sebelumnya, C_0. Sebaliknya pada dekripsi, blok plainteks diperoleh dengan cara meng-*XOR*-kan *IV* dengan hasil dekripsi terhadap blok cipherteks pertama.

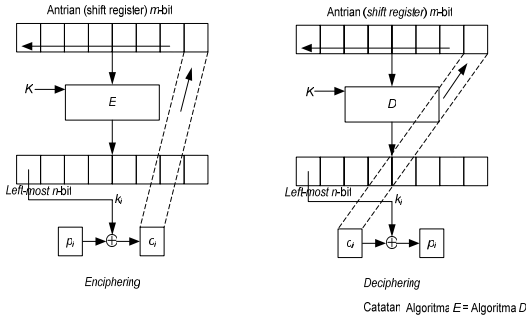


Gambar 4
Mode Operasi *Cipher Blok Chaining* (CBC)

Pada CBC diperlukan data acak sebagai blok pertama. IV digunakan hanya untuk membuat suatu pesan menjadi unik dan IV tidak mempunyai arti yang penting sehingga IV tidak perlu dirahasiakan.

3.3 *Cipher Feedback* (CFB)

Pada mode CBC, proses enkripsi atau dekripsi tidak dapat dilakukan sebelum blok data yang diterima lengkap terlebih dahulu. Masalah ini diatasi pada mode *Cipher Feedback* (CFB). Pada mode CFB, data dapat dienkripsi pada unit-unit yang lebih kecil atau sama dengan ukuran satu blok. Misalkan pada CFB 8 bit, maka data akan diproses tiap 8 bit.



Gambar 5
Mode Operasi *Cipher Feedback* (CFB)

Pada permulaan proses enkripsi, IV akan dimasukkan dalam suatu *register* geser. IV ini akan dienkripsi dengan menggunakan kunci yang sudah ada. Dari hasil enkripsi tersebut, akan diambil 8 bit paling kiri atau *Most Significant Bit* untuk di-XOR dengan 8 bit dari *plaintext*. Hasil operasi XOR inilah yang akan menjadi *ciphertext* dimana *ciphertext* ini tidak hanya dikirim untuk ditransmisikan tetapi juga dikirim sebagai *feedback* ke dalam *register* geser untuk dilakukan proses enkripsi untuk 8 bit berikutnya.

Tinjau mode *CFB n*-bit yang bekerja pada blok berukuran *m*-bit. Algoritma enkripsi dengan mode *CFB* adalah sebagai berikut:

1. Antrian diisi dengan *IV* (*initialization vector*).
2. Enkripsikan antrian dengan kunci *K*. *n* bit paling kiri dari hasil enkripsi berlaku sebagai *keystream* (k_i) yang kemudian di-*XOR*-kan dengan *n*-bit dari plainteks menjadi *n*-bit pertama dari cipherteks. Salinan (*copy*) *n*-bit dari cipherteks ini dimasukkan ke dalam antrian (menempati *n* posisi bit paling kanan antrian), dan semua *m*-*n* bit lainnya di dalam antrian digeser ke kiri

menggantikan *n* bit pertama yang sudah digunakan.

3. *m*-*n* bit plainteks berikutnya dienkripsikan dengan cara yang sama seperti pada langkah 2.

Sedangkan, algoritma dekripsi dengan mode *CFB* adalah sebagai berikut:

1. Antrian diisi dengan *IV* (*initialization vector*).
2. Dekripsikan antrian dengan kunci *K*. *n* bit paling kiri dari hasil dekripsi berlaku sebagai *keystream* (k_i) yang kemudian di-*XOR*-kan dengan *n*-bit dari cipherteks menjadi *n*-bit pertama dari plainteks. Salinan (*copy*) *n*-bit dari cipherteks dimasukkan ke dalam antrian (menempati *n* posisi bit paling kanan antrian), dan semua *m*-*n* lainnya di dalam antrian digeser ke kiri menggantikan *n* bit pertama yang sudah digunakan.
3. *m*-*n* bit cipherteks berikutnya dienkripsikan dengan cara yang sama seperti pada langkah 2.

Baik enkripsi maupun dekripsi, algoritma *E* dan *D* yang digunakan sama. Mode *CFB n*-bit yang bekerja pada blok berukuran *m*-bit dapat dilihat pada Gambar 4.

Secara formal, mode *CFB n*-bit dapat dinyatakan sebagai:

Proses Enkripsi:

$$C_i = P_i \oplus MSB_m(E_k(X_i))$$
$$X_{i+1} = LSB_{m-n}(X_i) \,||\, C_i$$

Proses Dekripsi:

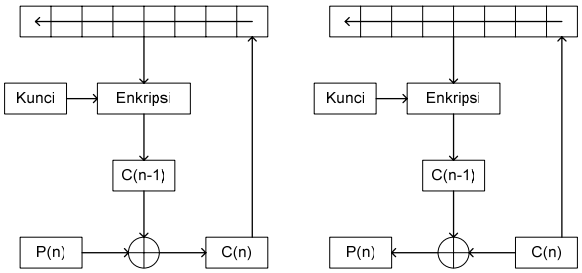
$$P_i = C_i \oplus MSB_m(D_k(X_i))$$
$$X_{i+1} = LSB_{m-n}(X_i) \,||\, C_i$$

yang dalam hal ini:

X_i = isi antrian dengan X_i adalah *IV*
E = fungsi enkripsi dengan algoritma *cipher* blok
D = fungsi dekripsi dengan algoritma *cipher* blok
K = kunci
m = panjang blok enkripsi/dekripsi
n = panjang unit enkripsi/dekripsi
|| = operator penyambungan (*concatenation*)
MSB = Most Significant Byte
LSB = Least Significant Byte

3.4 *Output Feedback* (OFB)

Sama pada mode CFB, mode OFB juga memerlukan sebuah *register* geser dalam pengoperasiannya. Pertama kali, IV akan masuk ke dalam *register* geser dan dilakukan enkripsi terhadap IV tersebut. Dari hasil proses enkripsi tersebut akan diambil 8 bit paling kiri untuk dilakukan XOR dengan *plaintext* yang nantinya akan menghasilkan *ciphertext*. *Ciphertext* tidak akan diumpan balik ke dalam *register* geser, tetapi yang akan diumpan balik adalah hasil dari enkripsi IV.



Gambar 6
Mode Operasi *Output Feedback* (OFB)

Tinjau mode *OFB n*-bit yang bekerja pada blok berukuran *m*-bit. Algoritma enkripsi dengan mode *OFB* adalah sebagai berikut (lihat Gambar 6):

1. Antrian diisi dengan *IV* (*initialization vector*).
2. Enkripsikan antrian dengan kunci *K*. *n* bit paling kiri dari hasil enkripsi dimasukkan ke dalam antrian (menempati *n* posisi bit paling kanan antrian), dan *m*-*n* bit lainnya di dalam antrian digeser ke kiri menggantikan *n* bit pertama yang sudah digunakan. *n* bit paling kiri dari hasil enkripsi juga berlaku sebagai *keystream* (k_i) yang kemudian di-*XOR*-kan dengan *n*-bit dari plainteks menjadi *n*-bit pertama dari cipherteks.
3. *m*-*n* bit plainteks berikutnya dienkripsikan dengan cara yang sama seperti pada langkah 2.

Sedangkan, algoritma dekripsi dengan mode *OFB* adalah sebagai berikut (lihat Gambar 6):

1. Antrian diisi dengan *IV* (*initialization vector*).

2. Dekripsikan antrian dengan kunci *K*. *n* bit paling kiri dari hasil dekripsi dimasukkan ke dalam antrian (menempati *n* posisi bit paling kanan antrian), dan *m*-*n* bit lainnya di dalam antrian digeser ke kiri menggantikan *n* bit pertama yang sudah digunakan. *n* bit paling kiri dari hasil dekripsi juga berlaku sebagai *keystream* (k_i) yang kemudian di-*XOR*-kan dengan *n*-bit dari cipherteks menjadi *n*-bit pertama dari plainteks.

3. *m*-*n* bit cipherteks berikutnya dienkripsikan dengan cara yang sama seperti pada langkah 2.

Baik enkripsi maupun dekripsi, algoritma *E* dan *D* yang digunakan sama.

4. MARS

4.1 Panjang Kunci dan Ukuran Blok MARS

MARS adalah *shared-key cipher* blok, dengan ukuran blok 128 bit dan ukuran kunci yang bervariasi antara 128 sampai 448 bit, sesuai dengan persyaratan yang ditetapkan oleh *NIST*.

4.2 Elemen Pembangun Algoritma MARS

4.2.1 Tipe – 3 *Feistel Network*

MARS memiliki panjang blok 128 bit dengan ukuran word 32 bit. Hal ini menunjukkan bahwa setiap blok terdiri dari 4 (empat) word. Dalam struktur network, yang mempunyai kemampuan untuk menangani empat word dalam satu blok adalah tipe 3-Feistal network.

Tipe-3 *Feistel network* terdiri dari banyak iterasi, dimana pada setiap iterasi terdapat satu word data (dan beberapa sub kunci) yang digunakan untuk memodifikasi ketiga word data yang lain. Hal ini berbeda dengan tipe-1 *Feistal network* yang pada setiap iterasinya terdapat satu word data yang digunakan untuk memodifikasi satu word data yang lain.

4.2.2 XOR, Penjumlahan, Pengalian

Operasi-operasi ini merupakan operasi-operasi sederhana yang digunakan untuk "*mix together*" atau mengkombinasikan data dan kunci.

4.2.3 *Fixed Rotation*

Rotasi berdasarkan nilai tertentu yang sudah ditetapkan. Dalam hal ini nilai rotasi untuk transformasi kunci adalah 13 posisi dcngan pergerakan rotasi ke kiri, untuk r -function adalah 5 dan 13 posisi dengan pergerakan rotasi ke kin, 24 posisi untuk *forward* mixing dengan pergerakan rotasi ke kin dan 24 posisi untuk *backward* mixing dengan pergerakan rotasi ke kanan.

4.2.4 *Data-dependent Rotation*

Rotasi berdasarkan nilai yang ditentukan bcrdasarkan 5 bit terendah (berkisar antara 0 dan 31) dari word data, misalkan nilai rotasi r = 5 bit terendah dari M maka nilai rotasi *r* akan sangat tergantung dengan nilai 5 bit terendah dari M.

4.2.5 S-box

MARS menggunakan tabel tunggal yang terdiri dari 512 32-bit word, yang disebut dengan S-box. Tabel S-box ini merupakan fixed tabel yang nilainya sudah tetap. Lihat lampiran.

4.3 Algoritma Eknkripsi MARS

Ukuran blok yang digunakan untuk enkripsi data pada algoritma MARS adalah 128 bit. Sebelum enkripsi blok dimulai, satu blok masukan dibagi menjadi empat word data dimana setiap word data terdiri dari 32-bit data. Untuk selanjutya keseluruhan operasi internal dilakukan pada 32-bit data atau satu word data.

4.3.1 Struktur *Cipher* Algoritma MARS

Struktur *cipher* pada MARS dibagi dalam 3 tahap yakni :

1. *Forward mixing*, berfungsi untuk mencegah serangan terhadap *chosen plainlext.* Terdiri dari penambahan sub kunci pada setiap word data atau *plaimezr*, diikuti dengan delapan iterasi

mixing tipe-3 *feitsal* (dalarn *forward mode)* dengan berbasis S-box.

2. *Cryptographic core* dan *cipher*, terdiri dari enam belas iterasi tranformasi kunci tipe-3 *feistal.* Untuk menjamin bahwa proses enkripsi dan dekripsi mempunyai kekuatan yang sama, delapan iterasi pertama ditunjukkan dalam "*forward mode"* dan delapan iterasi terakhir ditunjukkan dalam "backward mode".

3. *Backward mixing*, berfungsi untuk melindungi serangan kembali terhadap *chosen chipertext.* Tahap ini merupakan invers dari tahap pertama, terdiri dari delapan iterasi *mixing* tipe-3 *feistel* (dalam *backward mode)* dengan berbasis s-box, diikuti dengan pengurangan sub kunci dari word data. Hasil pengurangan inilah yang disebut dengan *ciphertext*.

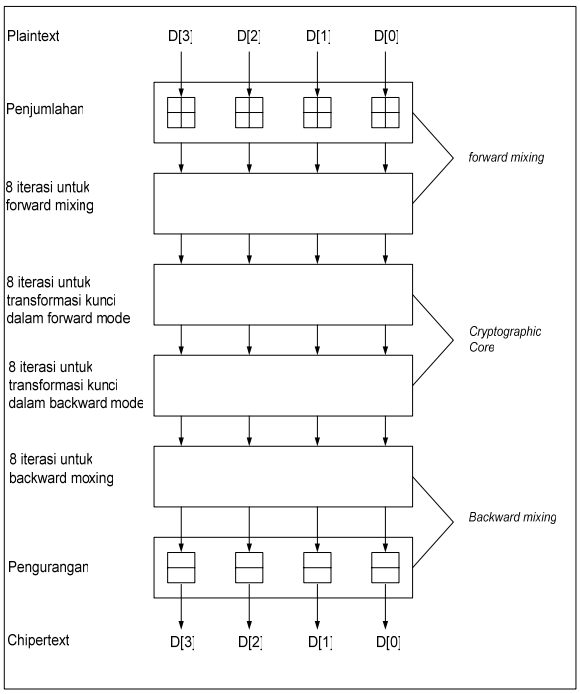
Notasi yang digunakan dalam cipher:

- D[ ] adalah sebuah array untuk 4 32-bit data. Inisial D berisi *plaintext* dan pada akhir proses enkripsi berisi *ciphertext.*

- K[ ] adalah array untuk expanded key, terdiri dari 40 32 bit word.

- S[ ] adalah sebuah S-box, terdiri dari 512 32-bit word.

Struktur umum dari *cipher* dapat dilihat pada gambar dibawah ini :



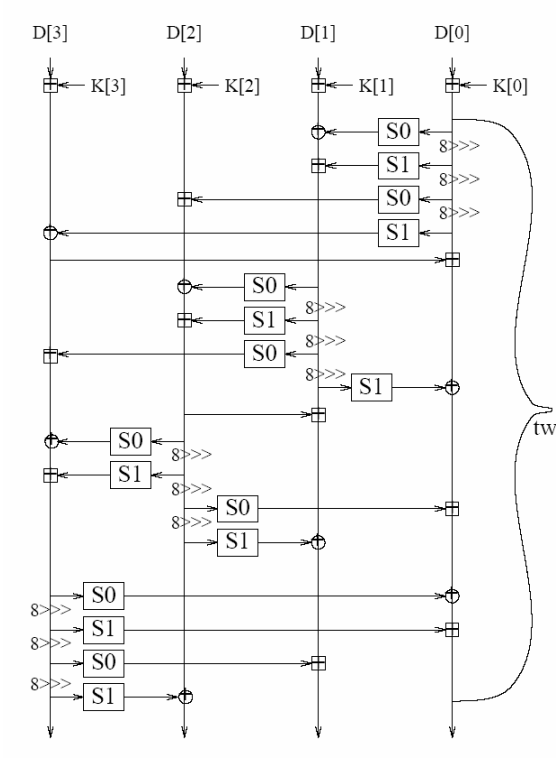
Gambar 7
***Struktur Cipher* algoritma MARS**

4.3.2 *Forward Mixing*

Dalam tahap ini, pertama-tama sebuah sub kunci ditambahkan pada setiap word data dari *plaintext*, dan kemudian dilakukan delapan iterasi *mixing* tipe-3 *feistel network* (dalam *forward mode)*, dikombinasikan dengan operasi *mixing* tambahan. Dalam setiap iterasi digunakan sebuah word data (*source* word) untuk memodifikasi tiga word data (target word). Keempat byte source word digunakan sebagai indeks untuk S.box, kemudian nilai S-box entri akan di-XOR-kan, atau ditambahkan pada ketiga word data yang lain.

Keempat byte dari source word dinotasikan dengan b0, b1, b2, b3 (dimana b0 adalah byte terendah dan b3 adalah byte tertinggi) dan digunakan sebagai indeks untuk S.box. S-box[b0] di-XOR-kan dengan target word pertama, dan S-box[b1+256] ditambahkan dengan target word yang sama. S-box[b2] ditambahkan dengan target word kedua dan S-box[b3+256] di-XOR-kan dengan target word ketiga. Terakhir source word dirotasikan sebanyak 24 posisi ke kanan.

Untuk iterasi berikutnya keempat word data dirotasikan, sehingga target word pertama saat ini menjadi source word berikutnya, target word kedua saat ini menjadi target word pertama berikutnya, target word ketiga saat ini menjadi target word kedua berikutnya dan source word saat ini menjadi target word ketiga benkutnya.



Gambar 8
Struktur Forward Mixing

4.3.3 *Cryptographic Core*

Cryptographic core pada MARS *cipher* menggunakan tipe 3 *-feistal network* yang terdiri dari enam belas iterasi. Dalam setiap iterasi digunakan E-*function (E* untuk *expansion)* yang mengkombinasikan penjumlahan, perkalian, fixed *rotation data dependent rotation* dan S _box lookup. Fungsi ini menerima input satu word data dan menghasilkan tiga word data sebagai output dengan notasi L, M dan K. Dalam setiap iterasi digunakan satu word data sebagai input untuk E-function dan ketiga output word data dari P *-function* ditambahkan atau di-XOR-kan ke ketiga word data yang lam. Kemudian Source word dirotasikan sebanyak 13 posisi ke kin.

Ketiga output dari E-*function* digunakan dengan cara yang berbeda dalam delapan iterasi pertama dibandingkan dengan delapan iterasi terakhir. Dalam delapan iterasi pertama, output pertama dan kedua dari E-*function* ditambahkan dengan target word pertama dan kedua, output ketiga di-XOR-kan dengan target word ketiga. Dalam delapan iterasi terakhir, output pertama dan kedua dari E *-function* ditambahkan dengan target word ketiga dan kedua, output ketiga di-XOR-kan dengan target word pertama.

4.3.3.1 *E-Function* (Fungsi Enkripsi)

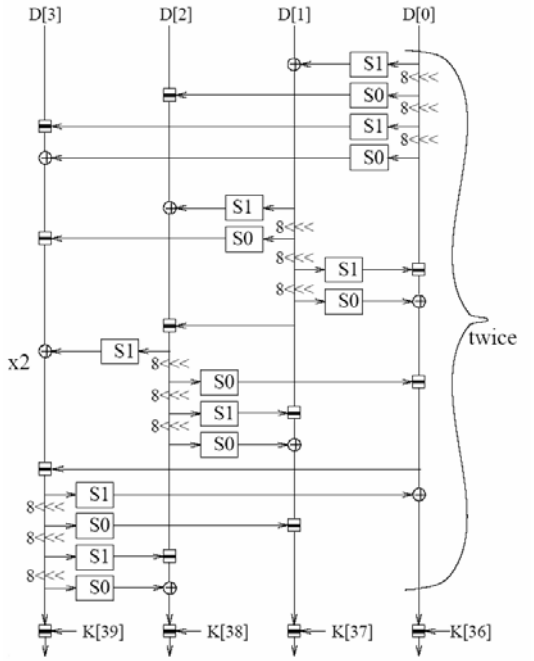
E -function menerima input satu word data dan menggunakan dua atau lebih sub kunci untuk menghasilkan tiga word data sebagai output. Dalam fungsi ini digunakan tiga variabel sementara, yang dinotasikan dengan L, M dan *R (left, middle, dan right).* R berfungsi untuk menampung nilai source word yang dirotasikan sebanyak 13 posisi ke kiri. M berfungsi untuk menampung nilai source word yang dijumlahkan dengan sub kunci pertama. Sembilan bit terendah dari M digunakan sebagai indeks untuk S.box. L berfungsi untuk menampung nilai yang sesuai dengan S-box entry. Sub kunci kedua akan dikalikan dengan *R* dan kemudian *R* dirotasikan sebanyak S posisi ke kiri. *L* di-XOR-kan dengan *R,* lima bit terendah dari *R* digunakan untuk nilai rotasi *r* dengan nilai antara 0 dan 31, dan M dirotasikan ke kiri sebanyak r posisi. *R* dirotasikan sebanyak 5 posisi ke kiri dan di-XOR-kan dengan *L.* Terakhir, lima bit terendah dari *R* diambil sebagai nilai rotasi *r* dan L dirotasikan kekiri sebanyak r posisi. Output word pertama dari E *-function* adalah M kedua adalah M dan ketiga adalah *R.*

4.3.4 *Backward Mixing*

Tahap ini merupakan invers dari tahap *forward mixing,* word data diproses dalam urutan yang berbeda dalam *backward mode.* Sama halnya pada *forward mixing,* pada *backward mixing* juga digunakan sebuah source word untuk memodifikasi tiga target word. Keempat byte dari source word dinotasikan dengan b0, bl, b2, b3 (dimana b0 adalah byte terendah dan b3 adalah byte tertinggi) dan digunakan sebagai indeks untuk S-box. S-box[b0+256] di-XOR-kan dengan target word pertama, dan S-box[b3] dikurangkan dengan target word kedua. S-box[b2+256] dikurangkan dengan target word ketiga dan S-box[b1] di-XOR-kan dengan target word ketiga

juga. Terakhir source word dirotasikan sebanyak 24 posisi ke kiri.

Untuk iterasi berikutnya keempat word data dirotasikan sehingga target word pertama saat ini menjadi source word berikutnya, target word kedua saat ini menjadi target word pertama berikutnya, target word ketiga saat ini menjadi target word kedua berikutnya dan source word saat ini menjadi target word ketiga berikutnya.



Gambar 9
Struktur Backward Mixing

4.3.4 Perluasan Kunci

Perluasan kunci berfungsi untuk membangkitkan sub kunci dari kunci yang diberikan oleh pemakai yakni k[] yang terdiri dari n 32-bit (word). Kunci diperluas menjadi 40 32-bit (word) sub kunci K[ ]. Dalam prosedur ini dibuluhkan 7 word data yang diambil dari S-box[0..6] dan digunakan untuk transformasi linier. Tabel temporeri T yang terdiri dari 47 word data digunakan untuk menampung nilai 7 word data dari S-box[0..6] dan nilai hasil transformasi linier, dimana 7 word pertama berisikan nilai S-box(0..6] dan 40 word terakhir akan diisikan melalui transformasi linier yang selanjutnya digunakan dalam iterasi untuk perluasan kunci. Dalam transformasi linier untuk T[0..38] diisikan dengan ketentuan T[i-7] di-XOR-kan dengan T[i-2] dan hasilnya dirotasikan sebanyak 3 posisi ke kin kemudian di-XOR-kan dengan k[i mod n], di-XOR-kan dengan i. Untuk T[39] diisikan dengan n.

Perluasan kunci dilakukan sebanyak 7 iterasi dan pada iterasi terakhir nilai temperori T[0..39] disubstitusikan menjadi nilai sub kunci K[0..39]. Dalam setiap iterasi T[1..39] didapat dengan cara menambahkan S-box[9 bit terendah dari T[i-1]] dengan T[i] dan kemudian dirotasikan sebanyak 9 posisi kekiri. Untuk T(0), S-box[9 *bit* terendah dari T[39] ditambahkan dengan T[0] dan kemudian dirotasikan sebanyak 9 posisi ke kiri. Dari hasil iterasi terakhir T[0..39] disubtitusikan ke *sub* kunci K[0..39] dengan cara : K[(7i) mod 40] diisikan dengan T[i].

Untuk nilsi K5 K7,.. K33 diubah dengan ketentuan: u digunakan untuk menampung nilai S-box[265+2 bit terendah dari K[1], j digunakan untuk menampung nilai 5 bit terendah dari K[i+3]. Kemudian nilai 2 bit terendah dari K[i] diset menjadi 1 dan ditampung dalam w. Bit mask M diset menjadi 1 jika dalam w1 terdapat 10 bit 1 atau bit 0 yang berurutan. U dirotasikan sebanyak j posisi ke kiri dan hasilnya ditampung dalam p. Terakhir p di-XOR-kan dengan w di bawah kontrol M dan disimpan dalam K[i].

5. Rijndael

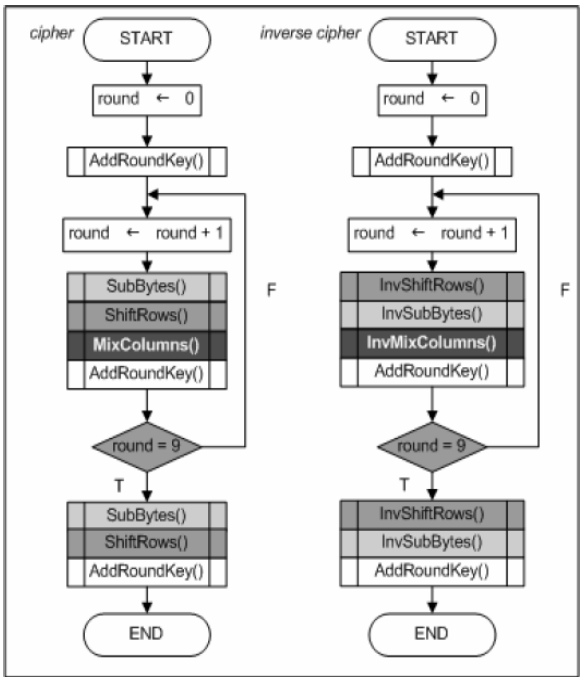
5.1 Panjang Kunci dan Ukuran Blok *Rijndael*

Rijndael menyediakan panjang kunci 128 bit sampai 256 bit dengan step 32 bit. Panjang kunci dan ukuran blok dapat dipilih secara independen. Karena *NIST* menetapkan persyaratan bahwa ukuran blok harus 128 bit, dan panjang kunci harus 128, 192, dan 256 bit. Setiap blok dienkripsi dalam sejumlah putaran tertentu bergantung pada panjang kuncinya. Karena *Rijndael* menjadi pemenang, maka *Rijndael* biasa disebut *AES*.

5.2 Algoritma

Algoritma Rijndael yang terpilih menjadi AES menyandikan data dalam empat langkah dasar yaitu, langkah nonlinear, langkah dispersi, langkah difusi, dan penambahan kunci [5]. Langkah-langkah tersebut dapat dideskripsikan lebih mudah dengan memvisualisasikan data yang akan dikonversi dalam array byte segi empat (Gambar 2-1). Nonlinearisasi diperoleh dengan memakai tabel subsitusi nonlinear. Dispersi dilakukan lewat permutasi byte-byte

data dari kolom array yang berbeda. Langkah difusi menyandikan data menjadi kombinasi linear dari byte-byte data dalam satu kolom array tersebut. Penambahan kunci dilakukan dengan operasi XOR antara data dengan kunci. Keempat langkah tersebut akan memiliki nama khusus dalam algoritma yang diterangkan AES.



Gambar 10
Diagram aliran cipher dan inverse cipher

5.2.1 Enkripsi

Cipher (Gambar 2-2) berlangsung dalam rentetan empat fungsi pembangun (primitif), SubBytes(), ShiftRows(), MixColumns(), dan AddRoundKey(). Rentetan tersebut dijalankan sebanyak Nr - 1 sebagai loop utama (Nr = 10 untuk AES-128). Setiap loop disebut round. AddRoundKey() dieksekusi sebagai round inisial sebelum loop utama. Setelah loop utama tersebut berakhir (sembilan round), SubBytes(), ShiftRows(), MixColumns() , dan AddRoundKey(), dieksekusi secara berturut-turut sebagai final round.

5.2.1.1 AddRoundKey()

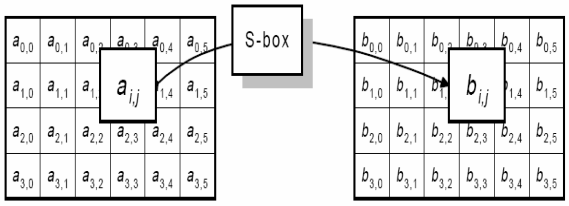
Melakukan *XOR* antara *state* awal (plainteks) dengan *cipher key*. Tahap ini disebut juga *initial round*.

5.2.1.2 SubBytes()

Substitusi byte dengan menggunakan tabel substitusi (*S-box*). Langkah ini disebut juga langkah transformasi nonlinear.

Tabel 1 Tabel *S-box* yang digunakan dalam transformasi *ByteSub() AES*

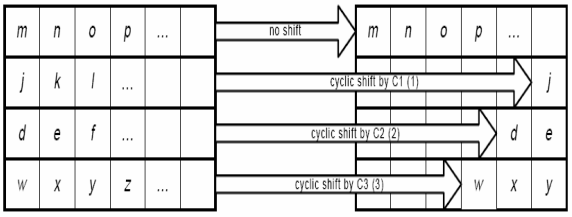
hex	y 0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x 0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16



Gambar 11
Transformasi Liner menggunakan S-box

5.2.1.3 ShiftRows()

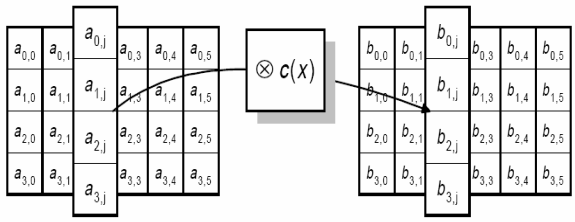
ShiftRows() merupakan langkah permutasi yang dieksekusi lewat pergeseran siklik tiga baris terakhir state (baris pertama, r = 0, tidak digeser). Baris ke dua digeser siklik ke kanan sekali, baris ke tiga dua kali, baris ke empat tiga kali.



Gambar 12
Operasi ShiftRows()

5.2.1.4 MixColomn()

Mengacak data di masing-masing kolom *array state*. Difusi diperoleh lewat transformasi MixColumns() yang mengoperasikan state kolom-demi-kolom.



Gambar 13
Operasi MixColumn()

5.2.2 Dekripsi

Setiap fungsi enkripsi di atas memiliki ebalikan, dekripsi berlangsung dengan kebalikan dari setiap primitif (inverse cipher). AddRoundKey() dieksekusi sebagai initial round, diikuti sembilan round rentetan InvShiftRows(), InvSubBytes(), InvMixColumns(), dan AddRoundKey(). Round ke-10 yang mengikutinya tidak menyertakan InvMixColumns serupa dengan final round enkripsi.

5.2.2.1 InvSubBytes()

Invers dari tabel S-Box yang digunakan untuk SubBytes tersedia sebagai $SBox^{-1}$.

Tabel 2 Tabel *S-box*$^{-1}$ yang digunakan dalam transformasi *InvByteSub()*

hex	y															
x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

5.2.2.2 InvShiftRows()

Kebalikan ShiftRows() ini berlangsung dengan menggeser siklik ke arah berlawanan. Baris ke dua digeser siklik ke kiri sekali, baris ke tiga dua kali, baris ke empat tiga kali.

5.2.2.3 InvMixColumns()

Operasi state ini merupakan operasi kebalikan dari MixColumns().

6. Pengujian dan Perbandingan Algoritma MARS dan Rijndael

6.1 Pengujian dengan Menggunakan Perangkat Lunak DEC V

6.1.1 Perancangan Kasus Uji

Pengujian dibagi-bagi menjadi beberapa kasus uji. Pembagian dilakukan berdasarkan sasaran pengujian. Pengujian di bawah ini lebih menekankan pada pengujian waktu enkripsi dan dekripsi. Sedangkan untuk tingkat keamanan akan diuji secara *white box*.

1. Kasus Uji 1
   Kasus Uji 1 bertujuan untuk menguji lama waktu proses enkripsi dan dekripsi dengan menggunakan *Rijndael* dengan mode operasi *ECB* untuk panjang kunci 128-bit.

2. Kasus Uji 2
   Kasus Uji 2 bertujuan untuk menguji lama waktu proses enkripsi dan dekripsi dengan menggunakan *MARS* dengan mode operasi *ECB*, untuk panjang kunci 128-bit.

3. Kasus Uji 3
   Kasus Uji 3 bertujuan untuk menguji lama waktu proses enkripsi dan dekripsi dengan menggunakan *Rijndael* dengan mode operasi *CBC*, untuk panjang kunci 128-bit.

4. Kasus Uji 4
   Kasus Uji 4 bertujuan untuk menguji lama waktu proses enkripsi dan dekripsi dengan menggunakan *MARS* dengan mode operasi *CBC*, untuk panjang kunci 128-bit.

5. Kasus Uji 5
   Kasus Uji 5 bertujuan untuk menguji lama waktu proses enkripsi dan dekripsi dengan menggunakan *Rijndael* dengan mode operasi *CFB,*, untuk panjang kunci 128-bit.

6. Kasus Uji 6
   Kasus Uji 5 bertujuan untuk menguji lama waktu proses enkripsi dan dekripsi dengan menggunakan *MARS* dengan mode operasi *CFB,*, untuk panjang kunci 128-bit.

7. Kasus Uji 7

Kasus Uji 7 bertujuan untuk menguji lama waktu proses enkripsi dan dekripsi dengan menggunakan *Rijndael* dengan mode operasi *OFB*, untuk panjang kunci 128-bit.

8. Kasus Uji 8
   Kasus Uji 5 bertujuan untuk menguji lama waktu proses enkripsi dan dekripsi dengan menggunakan *MARS* dengan mode operasi *OFB*, untuk panjang kunci 128-bit.

6.2 Evaluasi Hasil Uji

Dari hasil pengujian kasus uji 1, 2, 3, 4, 5, 6 , 7, 8 secara keseluruhan menunjukan bahwa waktu yang dibutuhkan dalam proses enkripsi dan dekripsi menggunakan algoritma Rijndael lebih cepat dibandingkan algoritma *MARS* untuk semua setiap tipe mode operasi cipher blok.

Hasil pengujian ini menunjukan secara pasti bahwa performansi *Rijndael* secara kompleksitas waktu lebih baik daripada *MARS*.

6.3 Perbandingan Metode yang Digunakan pada MARS dan Rijndael

Secara prinsip, seperti yang telah dijelaskan pada bagian abstraksi di atas, kedua algoritma baik *MARS* maupun *Rijndael* menggunakan metode-metode yang mirip. Seperti wajarnya algoritma cipher blok yang menggunakan operasi-operasi substitusi, transposisi, dan penambahan operasi lainnya.

Namun, yang mencolok dari perbedaan kedua algoritma diantaranya bahwa algoritma *MARS* mengimplementasian jaringan feistel dalam implementasinya. Sedangkan algoritma *Rijndael* tidak. Hal ini yang membuat perbedaan yang mencolok dalam segi kompleksitas waktu. Jadi usaha yang dilakukan pembuat *MARS* untuk lebih meningkatkan tingkat keamanan dengan menggunakan jaringan feistel harus dibayar mahal dengan performansinya yang buruk dalam segi kompleksitas waktu.

Kompleksitas ruang pada kedua algoritma ini dapat dilihat secara *white box*. Metode yang banyak meminta kapasitas memori komputer diantaranya adalah penggunaan S-box dalam algoritma. Kedua algoritma ini sama-sama mengimplementasikan S-box dalam algoritmanya. Jadi, kompleksitas ruang untuk panjang kunci yang sama untuk kedua algoritma tidak jauh berbeda.

Dilihat dari kemampuan kedua algoritma meneriama variasi panjang kunci, tentu saja algoritma *MARS* lebih unggul. *MARS* menerima kunci yang bervariasi antara 128 – 1248 bit. Sedangkan *Rijndael* hanya mampu menerima variasi panjang kunci 128 bit, 192 bit, dan 256 bit. Hal ini merupakan keunggulan tersendiri bagi algoritma *MARS*.

6.4 Perbandingan Tingkat Keamanan Kedua Algoritma Menghadapi Serangan Kriptanalisis

Kedua algoritma sangat mendukung keamanan terhadapa serangan kriptanalisis, baik linear maupun diferensial.

Sebenarnya algoritma *MARS* lebih unggul dalam tingkat keamanan dikarenakan penggunaan jaringan feistel sebagai salah satu metode operasi.

7. Kesimpulan

Baik algoritma *MARS* maupun *Rijndael* memenuhi syarat-syarat atau tuntutan sebagai algoritma blok cipher untuk era teknologi sekarang ini.

Namun bila dilihat lebih seksama, dari segi kompleksitas waktu algoritma *Rijndael* memiliki performansi yang lebih baik dibandingkan algoritma *MARS*. Sedangkan dalam segi kompleksitas ruang dan tingkat keamanan kedua algoritma tidak jauh berbeda. Mungkin itu yang menjadi salah satu pertimbangan *NIST* bahwa algoritma *Rijndael* pantas untuk menjadi *Advanced Encryption Standard* dibandingkan *MARS*.

Referensi

[1] Samosir, Henriko. *Animasi Algoritma Kunci Simetri MARS*. 2004.

[2] Wibowo, Wihartantyo Ari. *Advanced Encryption Standard, Algoritma Rijndael*. 2004.

[3] Lung, Chan. *Studi dan Implementasi Advanced Encryption Standard dengan Empat Mode Operasi Block Cipher*. 2004.

[4] IBM Corporation. *MARS – a candidate cipher for AES*. 1999.

[5] Vincent, Rijmen and Joan Daemen. *AES proposal : Rijndael*.

[6] Munir, Rinaldi. *Diktat Kuliah IF5054 : Kriptografi*. 2006. Institut Teknologi Bandung.

[7] *http://en.wikipedia.org/wiki/Cryptography*

Lampiran

WORD Sbox[ ] = {
0x09d0c479, 0x28c8ffe0, 0x84aa6c39,
0x9dad7287, 0x7dff9be3, 0xd4268361,
0xc96da1d4, 0x7974cc93, 0x85d0582e,
0x2a4b5705, 0x1ca16a62, 0xc3bd279d,
0x0f1f25e5, 0x5160372f, 0xc695c1fb,
0x4d7ff1e4, 0xae5f6bf4, 0x0d72ee46,
0xff23de8a, 0xb1cf8e83, 0xf14902e2,
0x3e981e42, 0x8bf53eb6, 0x7f4bf8ac,
0x83631f83, 0x25970205, 0x76afe784,
0x3a7931d4, 0x4f846450, 0x5c64c3f6,
0x210a5f18, 0xc6986a26, 0x28f4e826,
0x3a60a81c, 0xd340a664, 0x7ea820c4,
0x526687c5, 0x7eddd12b, 0x32a11d1d,
0x9c9ef086, 0x80f6e831, 0xab6f04ad,
0x56fb9b53, 0x8b2e095c, 0xb68556ae,
0xd2250b0d, 0x294a7721, 0xe21fb253,
0xae136749, 0xe82aae86, 0x93365104,
0x99404a66, 0x78a784dc, 0xb69ba84b,
0x04046793, 0x23db5c1e, 0x46cae1d6,
0x2fe28134, 0x5a223942, 0x1863cd5b,
0xc190c6e3, 0x07dfb846, 0x6eb88816,
0x2d0dcc4a, 0xa4ccae59, 0x3798670d,
0xcbfa9493, 0x4f481d45, 0xeafc8ca8,
0xdb1129d6, 0xb0449e20, 0x0f5407fb,
0x6167d9a8, 0xd1f45763, 0x4daa96c3,
0x3bec5958, 0xababa014, 0xb6ccd201,
0x38d6279f, 0x02682215, 0x8f376cd5,
0x092c237e, 0xbfc56593, 0x32889d2c,
0x854b3e95, 0x05bb9b43, 0x7dcd5dcd,
0xa02e926c, 0xfae527e5, 0x36a1c330,
0x3412e1ae, 0xf257f462, 0x3c4f1d71,
0x30a2e809, 0x68e5f551, 0x9c61ba44,
0x5ded0ab8, 0x75ce09c8, 0x9654f93e,
0x698c0cca, 0x243cb3e4, 0x2b062b97,
0x0f3b8d9e, 0x00e050df, 0xfc5d6166,
0xe35f9288, 0xc079550d, 0x0591aee8,
0x8e531e74, 0x75fe3578, 0x2f6d829a,
0xf60b21ae, 0x95e8eb8d, 0x6699486b,
0x901d7d9b, 0xfd6d6e31, 0x1090acef,
0xe0670dd8, 0xdab2e692, 0xcd6d4365,
0xe5393514, 0x3af345f0, 0x6241fc4d,
0x460da3a3, 0x7bcf3729, 0x8bf1d1e0,
0x14aac070, 0x1587ed55, 0x3afd7d3e,
0xd2f29e01, 0x29a9d1f6, 0xefb10c53,
0xcf3b870f, 0xb414935c, 0x664465ed,
0x024acac7, 0x59a744c1, 0x1d2936a7,
0xdc580aa6, 0xcf574ca8, 0x040a7a10,
0x6cd81807, 0x8a98be4c, 0xaccea063,
0xc33e92b5, 0xd1e0e03d, 0xb322517e,
0x2092bd13, 0x386b2c4a, 0x52e8dd58,
0x58656dfb, 0x50820371, 0x41811896,
0xe337ef7e, 0xd39fb119, 0xc97f0df6,
0x68fea01b, 0xa150a6e5, 0x55258962,
0xeb6ff41b, 0xd7c9cd7a, 0xa619cd9e,
0xbcf09576, 0x2672c073, 0xf003fb3c,
0x4ab7a50b, 0x1484126a, 0x487ba9b1,
0xa64fc9c6, 0xf6957d49, 0x38b06a75,
0xdd805fcd, 0x63d094cf, 0xf51c999e,
0x1aa4d343, 0xb8495294, 0xce9f8e99,
0xbffcd770, 0xc7c275cc, 0x378453a7,
0x7b21be33, 0x397f41bd, 0x4e94d131,
0x92cc1f98, 0x5915ea51, 0x99f861b7,
0xc9980a88, 0x1d74fd5f, 0xb0a495f8,
0x614deed0, 0xb5778eea, 0x5941792d,
0xfa90c1f8, 0x33f824b4, 0xc4965372,
0x3ff6d550, 0x4ca5fec0, 0x8630e964,
0x5b3fbbd6, 0x7da26a48, 0xb203231a,
0x04297514, 0x2d639306, 0x2eb13149,
0x16a45272, 0x532459a0, 0x8e5f4872,
0xf966c7d9, 0x07128dc0, 0x0d44db62,
0xafc8d52d, 0x06316131, 0xd838e7ce,
0x1bc41d00, 0x3a2e8c0f, 0xea83837e,
0xb984737d, 0x13ba4891, 0xc4f8b949,
0xa6d6acb3, 0xa215cdce, 0x8359838b,
0x6bd1aa31, 0xf579dd52, 0x21b93f93,
0xf5176781, 0x187dfdde, 0xe94aeb76,
0x2b38fd54, 0x431de1da, 0xab394825,
0x9ad3048f, 0xdfea32aa, 0x659473e3,
0x623f7863, 0xf3346c59, 0xab3ab685,
0x3346a90b, 0x6b56443e, 0xc6de01f8,
0x8d421fc0, 0x9b0ed10c, 0x88f1a1e9,
0x54c1f029, 0x7dead57b, 0x8d7ba426,
0x4cf5178a, 0x551a7cca, 0x1a9a5f08,
0xfcd651b9, 0x25605182, 0xe11fc6c3,
0xb6fd9676, 0x337b3027, 0xb7c8eb14,
0x9e5fd030,
0x6b57e354, 0xad913cf7, 0x7e16688d,
0x58872a69, 0x2c2fc7df, 0xe389ccc6,
0x30738df1, 0x0824a734, 0xe1797a8b,
0xa4a8d57b, 0x5b5d193b, 0xc8a8309b,
0x73f9a978, 0x73398d32, 0x0f59573e,
0xe9df2b03, 0xe8a5b6c8, 0x848d0704,
0x98df93c2, 0x720a1dc3, 0x684f259a,
0x943ba848, 0xa6370152, 0x863b5ea3,
0xd17b978b, 0x6d9b58ef, 0x0a700dd4,
0xa73d36bf, 0x8e6a0829, 0x8695bc14,
0xe35b3447, 0x933ac568, 0x8894b022,
0x2f511c27, 0xddfbcc3c, 0x006662b6,
0x117c83fe, 0x4e12b414, 0xc2bca766,
0x3a2fec10, 0xf4562420, 0x55792e2a,

0x46f5d857, 0xceda25ce, 0xc3601d3b,
0x6c00ab46, 0xefac9c28, 0xb3c35047,
0x611dfee3, 0x257c3207, 0xfdd58482,
0x3b14d84f, 0x23becb64, 0xa075f3a3,
0x088f8ead, 0x07adf158, 0x7796943c,
0xfacabf3d, 0xc09730cd, 0xf7679969,
0xda44e9ed, 0x2c854c12, 0x35935fa3,
0x2f057d9f, 0x690624f8, 0x1cb0bafd,
0x7b0dbdc6, 0x810f23bb, 0xfa929a1a,
0x6d969a17, 0x6742979b, 0x74ac7d05,
0x010e65c4, 0x86a3d963, 0xf907b5a0,
0xd0042bd3, 0x158d7d03, 0x287a8255,
0xbba8366f, 0x096edc33, 0x21916a7b,
0x77b56b86, 0x951622f9, 0xa6c5e650,
0x8cea17d1, 0xcd8c62bc, 0xa3d63433,
0x358a68fd, 0x0f9b9d3c, 0xd6aa295b,
0xfe33384a, 0xc000738e, 0xcd67eb2f,
0xe2eb6dc2, 0x97338b02, 0x06c9f246,
0x419cf1ad, 0x2b83c045, 0x3723f18a,
0xcb5b3089, 0x160bead7, 0x5d494656,
0x35f8a74b, 0x1e4e6c9e, 0x000399bd,
0x67466880, 0xb4174831, 0xacf423b2,
0xca815ab3, 0x5a6395e7, 0x302a67c5,
0x8bdb446b, 0x108f8fa4, 0x10223eda,
0x92b8b48b, 0x7f38d0ee, 0xab2701d4,
0x0262d415, 0xaf224a30, 0xb3d88aba,
0xf8b2c3af, 0xdaf7ef70, 0xcc97d3b7,
0xe9614b6c, 0x2baebff4, 0x70f687cf,
0x386c9156, 0xce092ee5, 0x01e87da6,
0x6ce91e6a, 0xbb7bcc84, 0xc7922c20,
0x9d3b71fd, 0x060e41c6, 0xd7590f15,
0x4e03bb47, 0x183c198e, 0x63eeb240,
0x2ddbf49a, 0x6d5cba54, 0x923750af,
0xf9e14236, 0x7838162b, 0x59726c72,
0x81b66760, 0xbb2926c1, 0x48a0ce0d,
0xa6c0496d, 0xad43507b, 0x718d496a,
0x9df057af, 0x44b1bde6, 0x054356dc,
0xde7ced35, 0xd51a138b, 0x62088cc9,
0x35830311, 0xc96efca2, 0x686f86ec,
0x8e77cb68, 0x63e1d6b8, 0xc80f9778,
0x79c491fd, 0x1b4c67f2, 0x72698d7d,
0x5e368c31, 0xf7d95e2e, 0xa1d3493f,
0xdcd9433e, 0x896f1552, 0x4bc4ca7a,
0xa6d1baf4, 0xa5a96dcc, 0x0bef8b46,
0xa169fda7, 0x74df40b7, 0x4e208804,
0x9a756607, 0x038e87c8, 0x20211e44,
0x8b7ad4bf, 0xc6403f35, 0x1848e36d,
0x80bdb038, 0x1e62891c, 0x643d2107,
0xbf04d6f8, 0x21092c8c, 0xf644f389,
0x0778404e, 0x7b78adb8, 0xa2c52d53,
0x42157abe, 0xa2253e2e, 0x7bf3f4ae,
0x80f594f9, 0x953194e7, 0x77eb92ed,
0xb3816930, 0xda8d9336, 0xbf447469,
0xf26d9483, 0xee6faed5, 0x71371235,
0xde425f73, 0xb4e59f43, 0x7dbe2d4e,
0x2d37b185, 0x49dc9a63, 0x98c39d98,
0x1301c9a2, 0x389b1bbf, 0x0c18588d,
0xa421c1ba, 0x7aa3865c, 0x71e08558,
0x3c5cfcaa, 0x7d239ca4, 0x0297d9dd,
0xd7dc2830, 0x4b37802b, 0x7428ab54,
0xaeee0347, 0x4b3fbb85, 0x692f2f08,
0x134e578e, 0x36d9e0bf, 0xae8b5fcf,
0xedb93ecf, 0x2b27248e, 0x170eb1ef,
0x7dc57fd6, 0x1e760f16, 0xb1136601,
0x864e1b9b, 0xd7ea7319, 0x3ab871bd,
0xcfa4d76f, 0xe31bd782, 0x0dbeb469,
0xabb96061, 0x5370f85d, 0xffb07e37,
0xda30d0fb, 0xebc977b6, 0x0b98b40f,
0x3a4d0fe6, 0xdf4fc26b, 0x159cf22a,
0xc298d6e2, 0x2b78ef6a, 0x61a94ac0,
0xab561187, 0x14eea0f0, 0xdf0d4164,
0x19af70ee
};