STUDI LIMA ALGORITMA FINALIS *ADVANCED ENCRYPTION STANDARD*

Nurkholis Madjid -- NIM 13503047

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13047@students.if.itb.ac.id

Abstrak

Makalah ini membahas tentang lima algoritma finalis *Advanced Encryotion Standard* (AES) yang merupakan pemilihan standard enkripsi yang diselenggarakan oleh NIST (*National Institute of Standards and Technology*) untuk menggantikan standard sebelumnya yaitu *Data Encryption Standard* (DES). Pada awalnya terseleksi lima belas kandidat, namun kemudian hanya lima algoritma saja yang berhasil masuk final, yang akhirnya dimenangkan oleh algoritma Rijndael.

Pada bagian pertama akan djelaskan secara singkat mengenai pemilihan AES sebagai pendahuluan. Di bagian kedua akan dibahas mengenai algoritma simetri *cipher block* yang merupakan salah satu syarat penting yang diajukan AES dalam pemilihannya. Kemudian di bagian ketiga, masing-masing algoritma akan dibahas satu per satu. Selanjutnya di bagian keempat, akan dibahas mengenai perbandingan kelima algoritma. Dan terakhir disimpulkan bahwa kelima algoritma memiliki kelebihan dan kekuarang masing-masing terbukti dengan walaupun sudah ditetapkan Rijndael sebagai standard baru, namun penggunaan kelima algoritma masih cukup luas sampai saat ini.

***Kata kunci**: Advanced Encryption Standard*, *cipher block,* Kriptografi, *MARS, RC6, Rijndael, Serpent, Twofish*.

1. Pendahuluan

Perkembangan teknologi saat ini telah mengubah cara masyarakat dalam berkomunikasi. Dengan adanya internet, pertukaran informasi sudah tidak terhambat lagi oleh aspek jarak dan waktu. Namun perkembangan teknologi informasi ini juga memiliki kelemahan dalam hal keamanan data. Penyadapan data dapat saja dilakukan oleh pihak-pihak yang tidak bertanggung jawab. Oleh karena itu aspek keamanan dalam pertukaran informasi menggunakan teknologi informasi menjadi sangat penting untuk diperhatikan.

Kriptografi adalah ilmu dan seni untuk menjaga keamanan pesan. Kriptografi menyediakan aspek kerahasiaan (menjaga isi pesan dari siapapun yang tidak berhak mengakses), integritas data (menjamin pesan belum pernah dimanipulasi selama pengiriman), otentikasi (identifikasi kebenaran pihak yang berkomunikasi dan sumber pesan), dan nirpenyangkalan (mencegah pihak yang berkomunikasi melakukan penyangkalan). Semua aspek tersebut sangat diperlukan dalam hal keamanan data pada teknologi informasi. Mengingat aspek keamanan data merupakan hal yang penting dalam pertukaran informasi, maka perkembangan teknologi informasi juga berimbas pada meningkatnya kebutuhan atas algoritma dan teknik kriptografi yang handal.

DES (*Data Encryption Standard*) yang diumumkan oleh NIST (*National Institute of Standards and Technology*) pada tahun 1972 sebagai standard algoritma kriptografi sudah dianggap tidak aman lagi. Dengan menggunakan perangkat keras tertentu, kunci dari DES dapat dipecahkan hanya dalam waktu yang relatif singkat. Oleh karena itu, pada tahun 1997 NIST mengumumkan adanya pemilihan standar baru sebagai pengganti DES yang diberi nama AES (*Advanced Encryption Standard*).

Persyaratan yang ditentukan oleh NIST dalam pemilihan algoritma standar enkripsi tersebut adalah sebagai berikut:

1. Merupakan algoritma kriptografi simetri *chipher* blok.
2. Seluruh rancangan algoritma harus publik (tidak dirahasiakan)
3. Panjang kunci fleksibel, yaitu sebesar 128 bit, 192 bit dan 256 bit.
4. Ukuran blok sebesar 128 bit.
5. Algoritma memungkinkan untuk diimplementasikan baik sebagai perangkat lunak maupun sebagai perangkat keras.

NIST mengumpulkan algoritma dari komunitas kriptografi, dengan tujuan memilih sebuah standar. Lima belas algoritma telah diajukan kepada NIST pada tahun 1998 dan kemudian pada tahun 1999 diseleksi menjadi lima finalis yang akan selanjutnya akan dipilih salah satu (atau lebih) algoritma untuk menjadi standar pada tahun 2000. Kelima finalis AES adalah sebagai berikut:

1. MARS, dari tim IBM
2. RC6, dari tim Laboratorium RSA
3. Rijndael, dari tim Vincent Rijmen dan Joan Daemen
4. Serpent, dari tim Ross Anderson, Eli Biham, dan Lars Knudsen
5. Twofish, dari tim Bruce Schneier

Lima algoritma finalis tersebut kemudian dipertandingkan dan pemenangnya akan menjadi standar enkripsi selanjutnya menggantikan DES. Kriteria utama yang dijadikan acuan untuk mempertandingkan kelima algoritma adalah keamanan, efisiensi, kebutuhan memori, dan fleksibilitas algoritma.

2. Algoritma Kriptografi Simetri *Cipher* Blok

Algoritma kriptografi akan disebut simetri apabila pasangan kunci untuk proses enkripsi dan dekripsinya sama. Algoritma enkripsi simetri inipun dibagi lagi menjadi dua kelas, yaitu *block-cipher* dan *stream-cipher*. Karena AES harus merupakan cipher blok, maka cipher stream tidak akan dibahas pada makalah ini.

Pada algoritma kriptografi simetri *cipher* blok, rangkaian bit-bit plainteks dibagi menjadi blok-blok bit dengan panjang sama [4]. Enkripsi dilakukan terhadap blok bit plainteks menggunakan bit-bit kunci (yang ukurannya sama dengan blok plainteks). Algoritma enkripsi menghasilkan blok cipherteks yang berukuran sama dengan blok plainteks. Dekripsi dilakukan dengan cara yang serupa seperti enkripsi.

Misalkan blok plainteks (P) yang berukuran m bit dinyatakan sebagai vektor

$$P = (p_1, p_2, ..., p_m)$$

yang dalam hal ini p_i adalah bit 0 atau bit 1 untuk i = 1, 2, …, m, dan blok cipherteks (C) adalah

$$C = (c_1, c_2, ..., c_m)$$

yang dalam hal ini c_i adalah bit 0 atau bit 1 untuk i = 1, 2, …, m. Bila plainteks dibagi menjadi n buah blok, barisan blok-blok plainteks dinyatakan sebagai

$$(P_1, P_2, \ldots, P_n)$$

Untuk setiap blok plainteks P_i, bit-bit penyusunnya dapat dinyatakan sebagai vektor

$$P_i = (p_{i1}, p_{i2}, ..., p_{im})$$

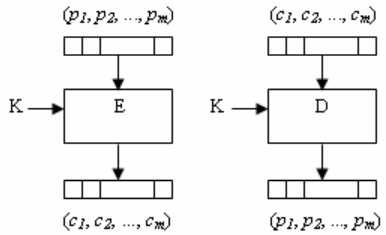
Enkripsi dengan kunci K dinyatakan dengan persamaan

$$E_k(P) = C,$$

sedangkan dekripsi dengan kunci K dinyatakan dengan persamaan

$$D_k(C) = P$$

Skema enkripsi dan dekripsi dengan *cipher* blok dapat dilihat pada Gambar 1. Fungsi E dan D dideskripsikan sendiri oleh kriptografer.



Gambar 1 Skema Enkripsi dan Dekripsi dengan *Cipher* Blok

2.1 Teknik Kriptografi yang Digunakan pada Chiper Blok

Algoritma blok cipher menggabungkan beberapa teknik klasik dalam proses enkripsi. Teknik kriptografi yang digunakan adalah:

1. **Substitusi**
   Teknik ini mengganti satu atau sekumpulan bit pada blok plainteks menjadi cipherteks tanpa mengubah urutannya.
2. **Transposisi atau Permutasi**
   Teknik ini memindahkan posisi bit pada blok plainteks untuk ditempatkan pada blok cipherteks berdasarkan aturan tertentu.
3. **Ekspansi**
   Teknik ini memperbanyak jumlah bit pada blok plainteks berdasarkan aturan tertentu, biasanya dinyatakan dengan tabel.
4. **Kompresi**
   Teknik ini kebalikan dari ekspansi, di mana jumlah bit pada blok plainteks dikompresi (diperkecil) berdasarkan aturan tertentu.

2.2 Prinsip Perancangan Cipher Blok

Perancangan algoritma kriptografi simetri cipher blok dapat mempertimbangkan beberapa prinsip berikut ini:

1. ***Confussion***
   Prinsip ini menyembunyikan hubungan apapun yang ada antara plainteks, cipherteks, dan kunci. Hal ini akan membuat kriptanalis frustasi untuk mencari pola-pola statistik yang muncul pada cipherteks.
2. ***Diffusion***
   Prinsip ini menyebarkan pengaruh satu bit plainteks atau kunci ke sebanyak mungkin cipherteks. Hal ini juga akan menyembunyikan hubungan statistik antara plainteks, cipherteks, dan kunci.
3. ***Iterated Cipher***
   Fungsi transformasi sederhana yang mengubah plainteks menjadi cipherteks diulang beberapa kali. Pada setiap putaran digunakan sub-kunci atau kunci putaran yang dikombinasikan dengan plainteks.
4. ***Feistel Network***
   Prinsip ini merupakan model jaringan yang membuat algoritma kriptografi menjadi *reversible*. Hal ini akan membuat kriptografer tidak perlu membuat algoritma baru untuk mendekripsi cipherteks menjadi plainteks.
5. ***Weak Key***
   Cipher blok yang bagus tidak memiliki kunci lemah (weak key). Kunci lemah adalah kunci yang menyebabkan tidak adanya perbedaan antara enkripsi dan dekripsi. Dekripsi terhadap cipherteks tetap menghasilkan plainteks semula, namun enkripsi dua kali berturut-turut pada plainteks akan menghasilkan kembali plainteksnya.
6. ***S-Box***
   Kotak-S (*S-Box*) adalah matriks yang berisi subtitusi sederhana yang memetakan satu atau lebih bit dengan satu atau lebih bit yang lain. Perancangan kotak-S menjadi isu penting dalam perancangan cipher blok karena kotak-S harus dirancang sedemikian rupa sehingga kekuatan kriptografinya bagus dan mudah untuk diimplementasikan.

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Gambar 2 Contoh S-Box pada DES

2.3 Mode Operasi *Cipher* Blok

Plainteks dibagi menjadi beberapa blok dengan panjang tetap. Beberapa mode operasi dapat diterapkan untuk melakukan enkripsi terhadap keseluruhan blok plainteks. Empat mode operasi yang lazim diterapkan pada sistem blok *cipher* adalah:

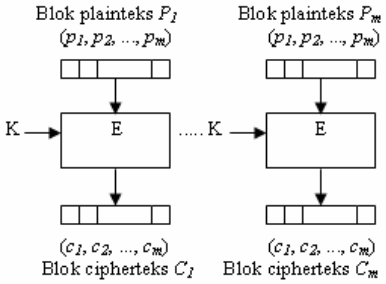
1. ***Electronic Code Book* (*ECB*)**
   Pada mode ini, setiap blok plainteks P_i dienkripsi secara individual dan independen menjadi blok cipherteks C_i. Secara matematis, enkripsi dengan mode *ECB* dinyatakan sebagai

   $$C_i = E_k(P_i)$$

   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. Skema enkripsi dengan mode *ECB* dapat dilihat pada Gambar 2.



Gambar 3 Skema Enkripsi dengan Mode *ECB*

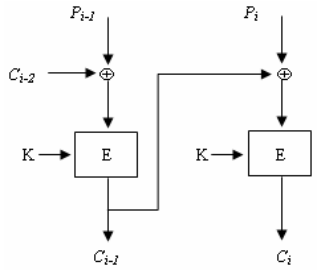
2. ***Cipher Block Chaining* (*CBC*)**
   Mode ini menerapkan mekanisme umpan balik (*feedback*) pada sebuah blok, yang dalam hal ini hasil enkripsi blok sebelumnya diumpanbalikkan ke dalam enkripsi blok yang *current*. Secara matematis, enkripsi dengan mode *CBC* dinyatakan sebagai

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

   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. Skema enkripsi dengan mode *CBC* dapat dilihat pada Gambar 3.



Gambar 4 Skema Enkripsi dengan Mode *CBC*

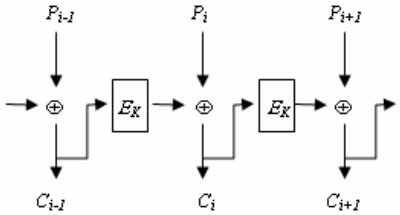
3. ***Cipher Feedback* (*CFB*)**
   Pada mode *CFB*, data dienkripsikan dalam unit yang lebih kecil daripada ukuran blok. Bila unit yang dienkripsikan satu karakter setiap kalinya, maka mode *CFB*-nya disebut *CFB* 8-bit. Secara umum *CFB* *n*-bit mengenkripsi plainteks sebanyak *n* bit setiap kalinya, yang mana $n \le m$ (m = ukuran blok). Mode *CFB* membutuhkan sebuah antrian (*queue*) yang berukuran sama dengan ukuran blok masukan. Baik enkripsi maupun dekripsi, algoritma *E* dan *D* yang digunakan sama. Secara matematis, enkripsi dengan mode *ECB* dinyatakan sebagai

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

   dan dekripsi sebagai

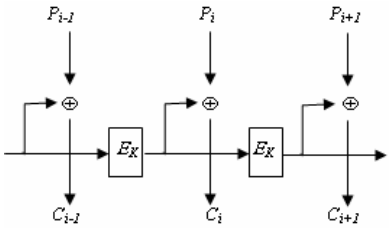
   $$P_i = C_i \oplus MSB_m(D_k(X_i))$$
   $$X_{i+1} = LSB_{m-n}(X_i) \,||\, C_i$$

   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



Gambar 5 Skema Enkripsi dengan Mode *CFB*

4. ***Output Feedback* (*OFB*)**
   Mode *OFB* mirip dengan mode *CFB*, kecuali *n*-bit dari hasil enkripsi terhadap antrian disalin menjadi elemen posisi paling kanan di antrian. Skema enkripsi dengan mode *OFB* dapat dilihat pada Gambar 5.



Gambar 6 Skema Enkripsi dengan Mode *OFB*

3. Studi Algoritma

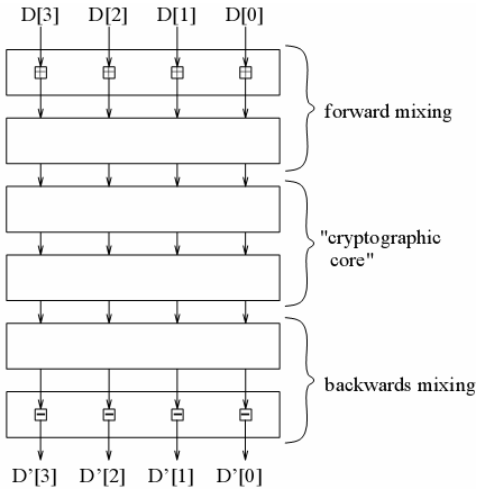
3.1 Algoritma MARS

MARS adalah cipher block yang didesain oleh tim dari IBM Corporation, yaitu Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M.Matyas, Jr., Luke O'Connor, Mohammad Peyravian, David Safford, dan Nevenko Zunic.

MARS merupakan block cipher dengan panjang blok 128 bit dan panjang kunci fleksible dari 128 bit sampai 499 bit. Algoritma ini bekerja dengan *word* 32 bit, dan menggunakan jaringan Feistel (Feistel network) tipe-3 [3].

MARS menerima input dan menghasilkan output empat *word* 32 bit. Cipher ini berorientasi pada *word*, di mana semua operasi internalnnya bekerja pada *word* 32 bit, sehingga struktur internal dari MARS adalah endian-neutral (code yang sama dapat bekerja pada mesin little-endian maupun big-endian). Jika input (atau output) cipher adalah byte stream, maka digunakan pengurutan little-endian byte untuk menginterpretasi masing-masing empat byte sebagai sebuah *word* 32 bit.

Gambar 5 memperlihatkan skema dari MARS yang dibagi menjadi 3 fase. Fase pertama, "*Forward Mixing*" berisi pengacakan besar-besaran dan "hujan kunci" untuk membuat frustasi serangan *chosen-plaintext*. Di dalam fase ini dilakukan penambahan *word* kunci ke *word* data, diikuti dengan delapan putaran berbasis S-Box. Fase kedua merupakan "*Cryptographic Core*" (inti kriptografi) dari cipher ini, yang berisi 16 putaran transformasi dengan jaringan Feistel. Untuk memastikan enkripsi dan dekripsi memiliki kekuatan yang sama, maka 8 putaran pertama dilakukan dengan "*forward mode*" sedangakan sisanya dengan "*backward mode*". Fase terakhir, "*Backward Missing*" operasi yang dilakukan hampir sama seperti fase pertama, namun merupakan kebalikan dari fase pertama. Fase ini bertujuan melindungi cipher dari serangan *chosen-ciphertext*.



Gambar 5 Skema MARS

3.1.1 Fase I: *Forward Mixing*

Pada fase ini, pertama kali yang dilakukan adalah menambahkan masing-masing *word* data dengan *word* kunci. Selanjutnya masuk ke dalam delapan putaran pengacakan jaringan Feistel tanpa kunci, dikombinasikan dengan beberapa operasi pengacakan tambahan. Masing-masing putaran, digunakan sebuah *word* data (disebut dengan *source word*) untuk modifikasi tiga *word* data lainnya (drisebut dengan *target words*). Keempat byte *source word* dijadikan indeks ke dua S-Box, S_0 dan S_1 yang berisi 256 *word* 32 bit, dan elemen S-Box yang berkoresponden di-XOR-kan atau ditambahkan kepada tiga *target words*.

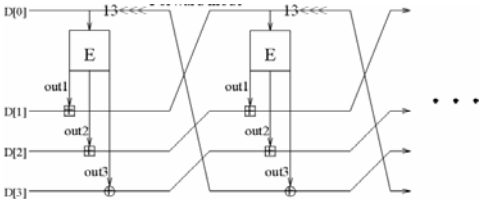
Misalkan *source word* dinotasikan dengan b_0, b_1, b_2, b_3, maka b_0 dan b_2 dijadikan indeks ke S-Box S_0, dan b_1 dan b_3 untuk S-Box S_1. Pertama, $S_0[b_0]$ di-XOR-kan dengan *target word* pertama, lalu $S_1[b_1]$ ditambahkan ke *target word* yang sama. Kemudian $S_0[b_2]$ ditambahkan kepada *target word* kedua dan $S_1[b_3]$ di-XOR-kan dengan *target word* ketiga. Selanjutnya *source word* dirotasi 24 bit ke kanan.

Untuk putaran selanjutnya, keempat *word* dirotasikan, sehingga *target word* pertama menjadi *source word* berikutnya, *target word* kedua menjadi *target word* pertama, *target word ketiga* menjadi *target word* kedua, dan *source word* menjadi *target word* ketiga.

Sebagai tambahan, setelah empat putaran, salah satu *target word* ditambahkan kepada *source word*. Contohnya, setelah putaran pertama dan kelima, *target word* ketiga ditambahkan kembali ke *source word*. Alasan dari pengacakan tambahan ini adalah untuk mengurangi beberapa serangan differential yang mudah, untuk memecah simetri pada fase mixing.

3.1.2 Fase II: *Main Keyed Transformasion*

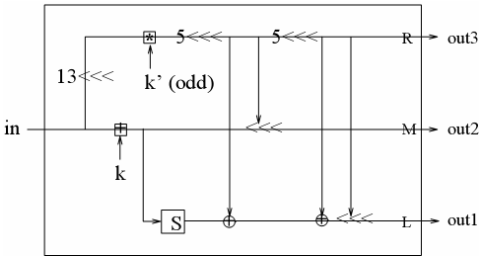
Inti dari cipher MARS adalah jaringan Feistel tipe-3 yang terdiri dari 16 putaran. Masing-masing putaran digunakan sebuah fungsi E (E dari kata *expansion*) yang berbasiskan pada sebuah kombinasi baru dari perkalian, rotasi dan S-Box. Fungsi ini menerima input sebuah *wordi* data dan mengembalikan tiga *word* data sebagai output.



Gambar 5 Jaringan Feistel pada fase II untuk mode *forward*

Gambar 7 memperlihatkan struktur jaringan Feistel. Pada setiap putaran, digunakan sebuah *word* data sebagai input fungsi E, dan tiga output *word* dari fungsi E ditambahkan atau di-XOR-kan dengan tiga *word* data lainnya. Sebagai tambahan, *source word* dirotasikan 13 posisi ke kiri.

Untuk memastikan bahwa cipher memiliki ketahanan terhadap serangan *chosen-ciphertext* yang sama dengan ketahanan terhadap serangan *chosen-plaintext*, maka tiga output fungsi E digunakan dalam urutan yang berbeda pada delapan putaran pertama dengan delapan putaran terakhir. Misalnya, pada delapan putaran pertama output pertama dan kedua fungsi E ditambahkan pada *target word* pertama dan kedua secara berurutan, dan output ketiga di-XOR-kan dengan *target word* ketiga. Pada delapan putaran berikutnya, output pertama dan kedua fungsi E ditambahkan pada *target word* ketiga dan kedua secara berurutan, dan output ketiga di-XOR-kan dengan *target word* pertama.



Gambar 5 Skema Fungsi E

Skema dari fungsi E dapat dilihat pada gambar 9. fungsi ini menerima input sebuah *word* data dan menggunakan dua *word* kunci untuk menghasilkan tiga *word* output. Pada fungsi ini digunakan tiga variable termporer, dinotasikan sebagai *L, M,* dan *R* (untuk *left, middle,* dan *right*). Selanjutnya tiga variable ini akan dirujuk sebagai tiga "*lines*" dalam fungsi.

Awalnya, set *R* dengan nilai dari *source word* setelah dirotasikan 13 posisi ke kiri, dan set *M* dengan jumlah dari *source word* dan *word* kunci pertama. Kemudian sembilan bit paling kanan dijadikan sebagai indek untuk sebuah E-Box dengan 512 elemen (yang didapatkan dengan konkatenasi S_1 dan S_1 dari fase sebelumnya), dan set *L* dengan nilai yang berkoresponden pada S-Box.

Selanjutnya *word* kunci kedua (diharuskan memiliki sebuah integer ganjil) dikalikan dengan *R* dan kemudian dirotasikan 5 posisi ke kiri. Setelah itu *R* di-XOR-kan dengan *L*, dan rotasikan *M* ke kiri sebanyak lima bit terkanan dari *R*. Kemudian rotasikan *R* 5 posisi ke kiri dan XOR-kan dengan *L*. Terakhir, rotasikan *L* ke kiri sebanyak lima bit terkanan dari *R*. Output pertama dari fungsi E adalah *L*, kedua adalah *M*, dan output ketiga adalah *R*.

3.1.1 Fase III: *Backward Mixing*

Fase ini sama dengan fase *forwars mixing*, hanya bedanya *word* data diproses dengan urutan yang berkebalikan. Sama seperti di *forward mixing*, digunakan juga sebuah *source word* untuk memodifikasi tiga *word target*. Untuk *source word* b_0, b_1, b_2, b_3, maka b_0 dan b_2 digunakan sebagai indeks ke S-Box S_1 dan b_1 dan b_3 indeks untuk S_0. $S_1[b_0]$ di-XOR-kan dengan *target word* pertama, kurangkan $S_0[b_3]$ dengan *target word* kedua, dan kurangkan $S_1[b_2]$ dengan *target word* ketiga dan XOR-kan $S_0[b_1]$ juga dengan *target word* ketiga. Terakhir, rotasikan *source word* 24 posisi ke kiri.

Untuk putaran selanjutnya, keempat *word* dirotasikan, sehingga *target word* pertama menjadi *source word* berikutnya, *target word* kedua menjadi *target word* pertama, *target word ketiga* menjadi *target word* kedua, dan *source word* menjadi *target word* ketiga.

3.2 Algoritma RC6

Algoritma RC6 didesain oleh Ronald L. Rivest dari laboratorium ilmu komputer MIT, dan tiga orang dari laboratorium RSA, yaitu M.J.B. Robshaw, R. Sidney, dan Y.L. Yin. RC6 ini merupakan pengembangan dari algoritma RC5, yang sengaja didesain untuk memenuhi persyaratan dari AES [6].

Seperti RC5, RC6 adalah algoritma enkripsi yang termasuk ke dalam *fully-parameterized familiy*. Sebuah versi RC6 lebih akurat dispesifikasikan sebagai RC6-*w*/*r*/*b* di mana *w* adalah ukuran *word*, *r* adalah jumlah putaran dalam proses enkripsi yang bernilai non-negatif, dan *b* adalah panjang kunci enkripsi dalam byte.

Untuk semua variannya, RC6-*w*/*r*/*b* beroperasi pada unit (blok) dengan empat *word* *w*-bit menggunakan keenam operasi dasar berikut ini.

$a + b$	penambahan integer mod 2^w
$a - b$	pengurangan integer mod 2^w
$a \oplus b$	bitwise XOR *word* *w*-bit
a x b	perkalian integer mod 2^w
$a <<< b$	rotasi *word* *w*-bit *a* ke kiri sejauh *least significant* $\log_2 w$ dari *b*
$a >>> b$	rotasi *word* *w*-bit *a* ke kanan sejauh *least significant* $\log_2 w$ dari *b*

Penting untuk diperhatikan bahwa RC6 menggunakan istilah "*round*" (putaran) adalah analog dengan istilah putaran yang biasa digunakan pada DES-*like* yaitu setengah data di-*update* oleh setengah lainnya, dan kemudian keduanya di-*swap* (tukar), atau yang lebih sering disebut dengan jaringan Feistel. Pada RC5, aksi seperti itu disebut dengan "*half-round*" (setengah-putaran), sedangkan satu putaran terdiri dari dua setengah-putaran. Ini mungkin akan membingungkan, sehingga RC6 menggunakan istilah "*round*" sesuai dengan istilah yang umum digunakan.

3.2.1 *Key Schedule*

Key schedule pada RC6-*w*/*r*/*b* secara praktiknya identik dengan *key schedule* pada RC5-*w*/*r*/*b*. Satu-satunya perbedaan dari keduanya adalah RC6-*w*/*r*/*b* menurunkan *word* lebih banyak dari kunci yang dimasukkan oleh *user* pada proses enkripsi dan dekripsi. *User* memasukkan sebuah kunci dengan panjang *b* byte. Sejumlah byte nol akan ditambahkan pada akhir kunci untuk mendapatkan panjang kunci yang equal dengan integral bukan-nol jumlah *word*. Bytes dari kunci ini kemudian diproses dalam mode little-endian menjadi array *word* *w*-bit sejumlah *c, L*[0], ..., *L*[*c*-1]. Kemudian byte pertama kunci disimpan sebagai byte *low-order* pada *L*[0], dan seterusnya. *L*[*c*-1] di-*padding* dengan *high-order* byte nol jika diperlukan. Jumlah *word* *w*-bit yang akan dibangkitkan untuk kunci putaran adalah sebanyak 2*r*+4 dan disimpan pada array *S*[0,..., 2*r*+3]. Berikut ini adalah pseudo-code implementasi *key schedule* RC6.

Key schedule for RC6-$w/r/b$

Input: User-supplied b byte key preloaded into the c-word array $L[0, \ldots, c-1]$
Number r of rounds

Output: w-bit round keys $S[0, \ldots, 2r+3]$

Procedure:
$S[0] = P_w$
for $i = 1$ **to** $2r + 3$ **do**
  $S[i] = S[i-1] + Q_w$
$A = B = i = j = 0$
$v = 3 \times \max\{c, 2r+4\}$
for $s = 1$ **to** v **do**
  {
    $A = S[i] = (S[i] + A + B) \lll 3$
    $B = L[j] = (L[j] + A + B) \lll (A + B)$
    $i = (i+1) \bmod (2r+4)$
    $j = (j+1) \bmod c$
  }

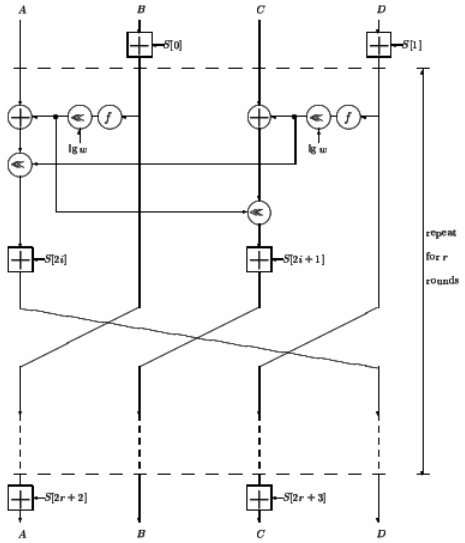
Gambar 5 *Pseudo-code* implementasi *key schedule* RC6

Konstanta P_{32} = B7E151613 dan Q_{32} = 9E3779B9 (heksadesimal) adalah "konstanta *magic*" yang sama yang digunakan dalam *key schedule* RC5. Nilai dari P_{32} ini diturunkan dari ekspansi biner dari *e*-2, di mana *e* adalah basis dari fungsi logaritma natural. Sedangkan nilai Q_{32} diturunkan dari ekspansi biner dari ϕ-1, di mana ϕ adalah *Golden Ratio.*

3.2.2 Proses Enkripsi dan Dekripsi

RC6 bekerja dengan empat *w*-bit register *A, B, C, D* yang berisi inisial input plainteks, dan di akhir proses enkripsi akan berisi output cipherteks, dan sebaliknya untuk proses dekripsi. Byte pertama dari plainteks atau cipherteks ditempatkan pada *least-significant* byte pada *A*, sedangkan byte terakhir dari plainteks atau cipherteks ditempatkan pada *most-significant* byte pada *D*. Kemudian *B* ditambahkan dengan *S*[0] dan *D* ditambahkan

dengan *S*[1]. Selanjutnya masuk pada putaran pertama algoritma.



Gambar 8 Skema Enkripsi RC6

Di dalam putaran, pertama kali *B* dan *D* masing-masing dimasukkan ke dalam fungsi f yang didefinisikan sebagai berikut.

$$f(X) = (X(2X+1)) \bmod 2^w$$

Hasil dari keduanya kemudian dirotasikan ke kiri dengan $\log_2 w$, (B <<< $\log_2 w$ dan D <<< $\log_2 w$) kemudian *B* di-XOR-kan dengan *A*, sedangkan *D* di-XOR-kan dengan *C*. Kemudian *A* dirotasikan ke kiri dengan *D*, sedangkan *C* dirotasikan ke kiri dengan *B* (*A* <<< *D* dan *C* <<< *B*). Setelah itu, *A* ditambahkan dengan *S*[2*i*] dan C ditambahkan dengan S[2*i*+1]. Di akhir putaran, nilai *A* dimasukkan ke dalam *D,* nilai *B* dimasukkan ke dalam *A*, nilai *C* dimasukkan ke dalam *B*, sedangkan nilai *D* dimasukkan ke dalam *C*. Gambar 8 mengilustrasikan skema enkripsi RC6.

3.3 Algoritma Rijndael

Rijndael merupakan block cipher yang didesain oleh Joan Daemen dan Vincent Rijmen. Desain dari Rijndael sangat dipengaruhi oleh desain dari block cipher Square yang juga merupakan hasil desain Daemen dan Rijmen. Nama algoritma ini merupakan kombinasi dari kedua penemunya tersebut.

Rijndael adalah cipher block yang berulang dengan panjang blok dan panjang kunci yang bervariasi. Panjang blok dan panjang kunci dapat dipilih secara independen antara 128, 192, dan 256 bit [2].

Tabel 1 Jumlah Putaran Setiap Blok pada Rijndael

Nr	Nb = 4	Nb = 6	Nb = 8
Nk = 4	10	12	14
Nk = 6	12	12	14
Nk = 8	14	14	14

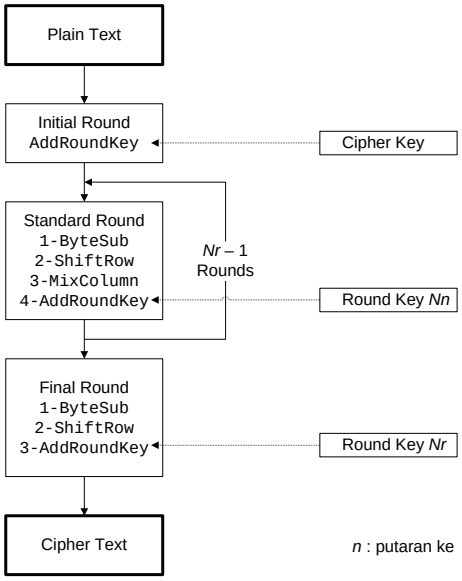
Tabel 1 memperlihatkan jumlah putaran (Nr) pada algoritma Rijndael yang tergantung pada panjang kunci (Nk = panjang kunci / 32 bit) dan panjang blok (Nb = panjang blok / 32 bit). Algoritma Rijndael menggunakan beberapa transformasi yang diulang pada setiap putaran. Masing-masing transformasi menghasilkan suatu *intermediate result* yang disebut dengan *state*. Pada gambar 7 diperlihtakan contoh representasi dari state dan key.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Gambar 7 Contoh state Nb=6 dan key Nk=4

Setiap putaran pada Rijndael menggunakan kunci internal yang berbeda yang disebut round key. Round key ini dibangkitkan dari cipher key yang merupakan parameter input algoritma. Proses pembangkitan round key ini disebut dengan Key Schedule.



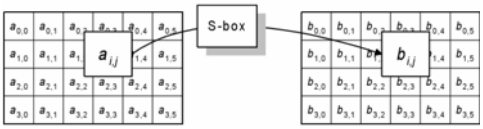
Gambar 7 Skema Rijndael

Seperti dapat dilihat pada gambar 8, secara umum, algoritma Rijndael terdiri dari:

1. *Initial Round Key Addition*:
   a. AddRoundKey
2. *Nr-1 Rounds*:
   a. ByteSub
   b. ShiftRow
   c. MixColumn
   d. AddRoundKey
3. *Final Round*:
   a. ByteSub
   b. ShiftRow
   c. AddRoundKey

3.3.1 Transformasi pada Rijndael

AddRoundKey merupakan proses bitwise XOR secara linear antara current state dengan roundkey. State hasil dari proses ini akan masuk ke tahap selanjutnya. Sedangkan ByteSub adalah proses substitusi byte secara non-linear dengan menggunakan sebuah tabel substitusi S-box. Berikut ini adalah ilustrasi dari ByteSub.



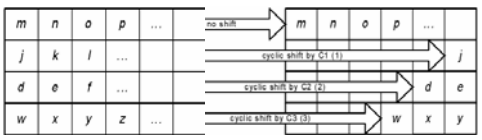
Gambar 8 Ilustrasi ByteSub

Pada ShiftRow, baris-baris pada current state digeser secara siklik. Baris ke-0 tidak digeser, baris ke-1 digeser sebesar C1 byte, baris ke-2 digeser sebesar C2 byte, dan baris ke-3 digeser sebesar C3 byte. Besarnya C1, C2, dan C3 tergantung pada panjang blok (Nb).

Tabel 2 Besar pergeseran byte pada ShiftRow

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

Gambar 6 mengilustrasikan bagaimana transformasi ShiftRow pada sebuah state.



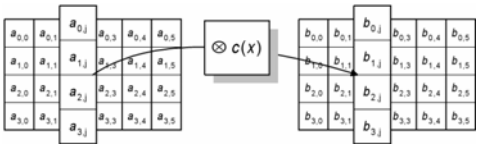
Gambar 9 Ilustrasi ShiftRow

Dalam transformasi MixColumn, setiap kolom pada state diperlakukan sebagai polinom pada GF(2^8) dan dikalikan dengan modulo x^4+1, dengan sebuah polinom c(x):

$$c(x) = \text{‘03’}\ x^3 + \text{‘01’}\ x^2 + \text{‘01’}\ x + \text{‘02’}$$

Persamaan ini dapat dinyatakan sebagai perkalian matriks $b(x) = c(x) \oplus a(x)$.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$



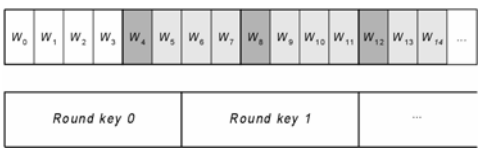
Gambar 10 Ilustrasi MixColumn

3.3.2 Pembangkitan kunci pada Rijndael

Seperti telah disebutkan sebelumnya bahwa pembangkitan kunci internal (round key) dari Ciper Key pada Rijndael disebut dengan Key Schedule. Prinsip dasar dari Key Schedule adalah:

1. Jumlah total bit round key adalah sama dengan panjang blok dikalikan dengan jumlah putaran ditambah 1 (misalnya untuk panjang blok 128 bit dengan 10 putaran, maka dibutuhkan 1408 bit round key)
2. Cipher Key diekspansi menjadi sebuah Expanded Key
3. Round key diambil dari Expanded key dengan aturan: round key pertama diambil dari Nb words pertama, round key kedua diambil dari Nb words kedua, dan seterusnya (1 word = 32 bit atau 4 byte)

Expanded key adalah sebuah array linear dari word 4 byte, dapat dinyatakan dengan W[Nb*(Nr+1)]. Nk word pertama merupakan cipher key, sedangkan word (W[i]) selanjutnya adalah hasil operasi XOR dari word sebelumnya (W[i-1]) dengan word Nk posisi sebelumnya (W[i-Nk]). Round key i didapat dari words W[Nb*i] sampai W[Nb*(i+1)]. Hal ini diilustrasikan pada gambar 11.



Gambar 10 Ilustrasi Key Schedule

3.4 Algoritma Serpent

Algoritma Serpent didesain oleh Ross Andersen dari Cambridge Universitiy Inggris, Eli Biham dar Technion Israel, dan Lars Knudsen dari University of Bergen Norwegia.

Serpent adalah algoritma dengan 32 putaran jaringan SP yang beroperasi pada empat *word* 32 bit, yang berarti ukuran bloknya adalah 128 bir. Semua nilai yang digunakan direpresentasikan sebagai *bitstream*. Untuk komputasi internal, semua nilai direpresentasikan dalam little-endian, di mana *word* pertama adalah *least-significant word*, dan *word* terakhir adalah *most-significant word*. Secara eksternal, setiap blok dituliskan sebagai plain hexadesimal 128 bit [1].

Serpent mengenkripsi plainteks *P* 128 bit menjadi cipherteks *C* 128 bit dalam 32 putaran dengan kontrol dari 33 sub-kunci 128 bit *K*0,...,K32. Panjang kunci masukan *user* fleksibel, namun untuk memenuhi persyaratan AES, maka ditetapkan 128, 192, dan 256 bit. Kunci yang lebih pendek dari 256 bit dipetakan menjadi kunci sepanjang 256 bit dengan menambahkan satu "1" bit pada akhir MSB, dan diikuti dengan "0" bit sampai mencapai 256 bit.

Cipher ini terdiri dari:
1. Permutasi inisial IP
2. 32 putaran, yang masing-masing terdiri dari sebuah operasi pengacakan kunci, operasi menggunakan S-Box, dan transformasi linear (kecuali pada putaran terakhir). Pada putaran terakhir, transformasi ini digantikan dengan penambahan operasi pengacakan kunci.
3. Permutasi akhir FP

Permutasi inisial IP diterapkan pada plainteks P menghasilkan B_0, yang merupakan input dari putaran pertama, yaitu putaran-0 (putaran diberi nomor dari 0 sampai 31). Hasil dari putaran pertama (putaran-0) dinamakan B_1, hasil putaran kedua (putaran-1) dinamakan B_2, dan seterusnya sampai B_{32}. Permutasi akhir akan menghasilkan cipherteks C.

Masing-masing fungsi putaran R_i (i = 0,...,31) hanya menggunakan sebuah S-Box ter-replikasi. Misalnya, R_0 menggunakan S_0, 32 copy yang diterapkan secara paralel, sehingga copy dari S_0 menggunakan bit 0,1,2, dan 3 dari $B_0 \oplus K_0$ sebagai input dan mengembalikan empat bit pertama dari vektor intermediate sebagai output, copy selanjutnya menerima masukan bit ke 4-7 dari $B_0 \oplus K_0$ dan mengembalikan empat bit selanjutnya dari vektor intemediate, dan seterusnya. Vektor intermediate kemudian ditransformasi menggunakan linear transformasi, menghasilkan B_1. Dengan cara yang sama, R1 menggunakan 32 copy S_1 secara paralel pada $B_1 \oplus K_1$ dan mentransformasi outputnya menggunakan transformasi linear, menghasilkan B_2.

Himpunan delapan S-Box digunakan sebanyak empat kali. Oleh karena itu, setelah menggunakan S_7 pada putaran-7, S_0 digunakan kembali pada putaran-8, kemudian S1 pada putaran-9, dan seterusnya. Putaran terakhir R_{31} sedikir berbeda dengan lainnya, yaitu dengan menggunakan S_7 pada $B_{31} \oplus K_{31}$, dan meng-XOR-kan hasilnya dengan K_{32}, bukan menggunakan transformasi linear. Hasil B_{32} kemudian dipermutasikan dengan FP, menghasilkan cipherteks.

Setiap putaran menggunakan 9 S-Box yang berbeda yang memetakan empat input bit ke empat output bit. Masing-masing S-Box digunakan tepat pada empat putaran, dan masing-masing digunakan 32 kali secara paralel.

Seperti DES, permutasi akhir adalah invers dari permutasi inisial. Dengan demikian, cipher secara formal dapat dideskripsikan sebagai berikut.

$$B_o := IP(P)$$
$$B_{i+1} := R_i(B_i)$$
$$C := FP(B_{32})$$

di mana:

$$R_i(X) = L(S(X \oplus K_i)) \qquad i = 0,...,30$$
$$R_i(X) = Si(S \oplus K_i)\, K_{32} \qquad i = 31$$

di mana Si adalah aplikasi S-Box $S_{i \bmod 8}$ 32 kali secara paralel, dan L adalah transformasi linear.

3.4.1 S-Box Serpent

S-Box Serpent adalah permutasi 4 bit dengan ketentuan sebagai berikut.
1. Masing-masing karakteristik diferensial memiliki probabilitas maksimal ¼, dan sebuah input dengan perbedaan satu bit tidak akan meghasilkan output dengan perbedaan satu bit
2. Masing-masing karakteristik linear memiliki probabilitas antara ½ ± ¼, dan hubungan linear antara sebuah bit pada input dan sebuah bit pada output memiliki probabilitas ½ ±1/8
3. Urutan non-linear bit output sebagai fungsi dari bit input maksimal 3

Pembangkitan S-Box terinspirasi dari EC4, yaitu menggunakan matriks dengan 32 array yang masing-masing memiliki 16 entri.

Matriks diinisialisasi dengan 32 baris S-Box DES dan ditransformasikan dengan menukar entri pada array ke-r bergantung pada nilai entri ke-(r+1) array dan pada inisial string yang merepresentasikan kunci. Jika array hasilnya memenuhi ketentuan yang telah disebutkan sebelumnya, maka simpan array sebagai Serpent S-Box. Ulangi prosedur tadi sampai 8 S-Box berhasil dibangkitkan.

S0:	3	8	15	1	10	6	5	11	14	13	4	2	7	0	9	12
S1:	15	12	2	7	9	0	5	10	1	11	14	8	6	13	3	4
S2:	8	6	7	9	3	12	10	15	13	1	14	4	0	11	5	2
S3:	0	15	11	8	12	9	6	3	13	1	2	4	10	7	5	14
S4:	1	15	8	3	12	0	11	6	2	5	4	10	9	14	7	13
S5:	15	5	2	11	4	10	9	12	0	3	14	8	13	6	7	1
S6:	7	2	12	5	8	4	6	11	14	9	1	15	13	3	10	0
S7:	1	13	15	0	14	8	2	11	7	4	12	10	9	3	5	6

Gambar 10 Skema Twofish

InvS0:	13	3	11	0	10	6	5	12	1	14	4	7	15	9	8	2
InvS1:	5	8	2	14	15	6	12	3	11	4	7	9	1	13	10	0
InvS2:	12	9	15	4	11	14	1	2	0	3	6	13	5	8	10	7
InvS3:	0	9	10	7	11	14	6	13	3	5	12	2	4	8	15	1
InvS4:	5	0	8	3	10	9	7	14	2	12	11	6	4	15	13	1
InvS5:	8	15	2	9	4	1	13	14	11	6	5	3	7	12	10	0
InvS6:	15	10	1	13	5	3	6	0	4	9	14	7	2	12	8	11
InvS7:	3	0	6	13	9	14	15	8	5	12	11	7	10	1	4	2

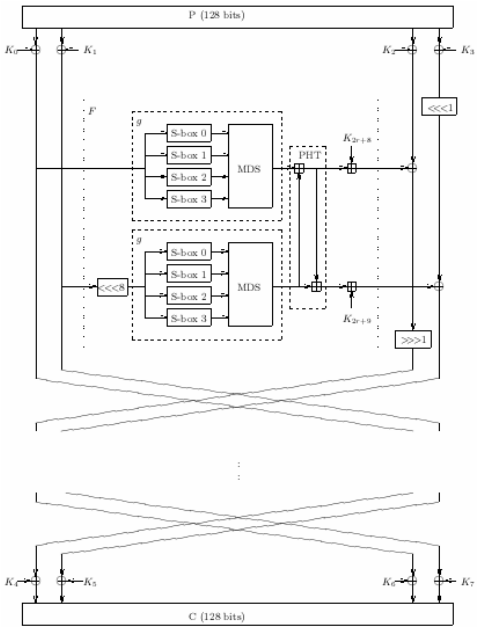
Gambar 10 Skema Twofish

3.5 Algoritma Twofish

Twofish didesain oleh Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, dan Niels Ferguson dari Laboratorium Counterpane System. Bruce Schneier juga mendesain algoritma Blowfish yang telah diimplementasi oleh lebih dari 130 aplikasi komersial. Twofish yang resmi, yaitu dengan 16 putaran, sampai saat ini belum terpecahkan. Namun untuk 5 putaran, telah berhasil dipecahkan dengan $2^{22,5}$ plainteks terpilih dan 2^{51} usaha (effort) [7].

Twofish adalah block cipher 126 bit yang menerima kunci dengan panjang yang fleksibel sampai 256 bit. Cipher ini menggunakan cukup banyak metode dalam implementasinya, meliput jaringan Feistel (Feistel network), S-Box, Matriks MDS, transformasi Pseudo-Hadamard, *whitening,* dan *key schedule*.

Gambar 1 memperlihatkan skema dari block cipher Twofish. Twofish menggunakan sebuah struktur seperti jaringan Feistel 16 putaran dengan tambahan whitening terhadap input dan output. Yang membuatnya bukan termasuk jaringan Feistel murni adalah adanya rotasi 1 bit. Sebenarnya rotasi ini dapat dimasukkan ke dalam fungsi F, agar dapat membentuk jaringan Feistel murni, namun hal ini membutuhkan rotasi tambahan terhadap *word* sebelum output dari tahap whitening.



Gambar 10 Skema Twofish

Plainteks dibagi menjadi empat *word* 32 bit. Pada tahap whitening input, *word* tersebut di-XOR-kan dengan empat *word* kunci, kemudian masuk ke dalam 16 putaran. Pada setiap putaran, dua *word* sebelah kiri digunakan sebagai masukan untuk dua fungsi g (salah satu *word* dirotasi 8 bit terlebih dulu). Fungsi g terdiri dari empat S-Box key-dependent, dan diikuti dengan tahap pengacakan linear menggunakan matriks MDS. Hasil dari dua fungsi g dikombinasikan menggunakan *Pseudo-Hadamard Transform* (PHT), dan penambahan dua *word* kunci. Kedua hasil ini kemudian di-XOR-kan dengan dua *word* plainteks sebelah kanan (salah satunya dirotasikan 1 bit ke kiri, dan yang lain dirotasikan 1 bit ke kanan terlebih dulu). Bagian kiri dan kanan ini kemudian di-*swap* (tukar) untuk masuk ke putaran selanjutnya. Setelah 16 putaran, proses *swap* terakhir dikembalikan lagi keempat *word* di-XOR-kan dengan *word* kunci untuk menghasilkan cipherteks.

Secara formal, 16 byte plainteks $p_0,...,p_{15}$ pertama kali dibagi menjadi empat *word* $P_0,...,P_3$ masing-masing 32 bit menggunakan konversi little-endian.

$$P_i = \sum_{j=0}^{3} p_{(4i+j)} \cdot 2^{8j} \qquad i = 0,...,3$$

Pada tahap input whitening, keempat *word* ini di-XOR-kan dengan 4 *word* kunci yang telah diekspan.

$$R_{0,i} = P_i \oplus K_i \qquad i = 0,...,3$$

Pada masing-masing 16 putaran, dua *word* pertama digunakan sebagai masukan fungsi F, yang juga menggunakan nomor putaran sebagai masukan. *Word* ke-3 di-XOR-kan dengan output pertama fungsi F dan kemudian dirotasikan 1 bit ke kanan. *Word* ke-4 dirotasikan 1 bit ke kiri kemudian di-XOR-kan dengan output ke-2 dari fungsi F. Setelah itu, kedua bagian ditukar, sehingga

$$(F_{r,0},F_{r,1}) = F(R_{r,0},R_{r,1},r)$$
$$R_{r+1,0} = ROR(R_{r,2} \oplus F_{r,0},1)$$
$$R_{r+1,1} = ROL(R_{r,3},1) \oplus F_{r,1}$$
$$R_{r+1,2} = R_{r,0}$$
$$R_{r+1,3} = R_{r,1}$$

untuk $r = 0,...,15$ dan di mana ROR dan ROL adalah fungsi rotasi kanan dan kiri. Tahap whitening output mengembalikan proses penukaran (*swap*) pada putaran terakhir dan meng-XOR-kan keempat *word* dengan kunci yang diekspan.

$$C_i = R_{16,(i+2) \bmod 4} \oplus K_{i+4} \qquad i = 0,...,3$$

Keempat *word* cipherteks kemudian dituliskan sebagai 16 byte $c_0,...,c_{15}$ menggunakan konversi little-endian.

$$c_i = \left\lfloor \frac{C_{\lfloor i/4 \rfloor}}{2^{8(i \bmod 4)}} \right\rfloor \bmod 2^8 \qquad i = 0,...,15$$

3.5.1 Fungsi F

Fungsi F adalah permutasi pada nilai 64 bit yang bergantung pada kunci (*key-dependent*). Fungsi ini membutuhkan 3 masukan, yaitu 2 *word* input R_0 dan R_1, dan nomor putaran r yang digunakan untuk mendapatkan kunci putaran yang sesuai. R_0 dimasukkan ke dalam fungsi g, menghasilkan T_0. R_1 dirotasikan 8 bit ke kiri kemudian dimasukkan ke dalam fungsi g menghasilkan T_1. T_0 dan T_1 dikombinasikan dalam PHT dan ditambahkan dengan dua *word* hasil ekspansi kunci.

$$T_0 = g(R_0)$$
$$T_1 = g(ROL(R_1,8))$$
$$F_0 = (T_0 + T_1 + K_{2r+8}) \bmod 2^{32}$$
$$F_1 = (T_0 + 2T_1 + K_{2r+9}) \bmod 2^{32}$$

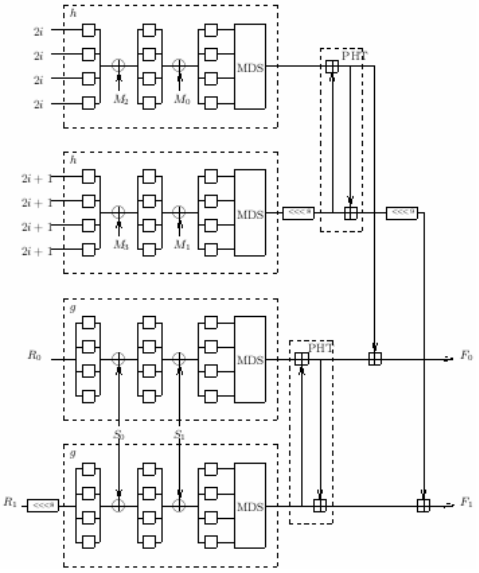
di mana (F_0, F_1) adalah hasil dari fungsi F.

3.5.2 Fungsi g

Fungsi g merupakan inti dari Twofish. *Word* input *X* (32 bit) dibagi menjadi empat byte. Masing-masing byte dimasukkan ke dalam S-Box *key-dependent*. S-Box tersebut bersifat bijektif, yaitu menerima masukan 8 bit dan menghasilkan keluaran 8 bit juga. Keempat hasil itu diperlakukan sebagai sebuah vektor dengan panjang 4 pada GF(2^8), dan dikalikan dengan matriks MDS 4x4 (menggunakan *field* GF(2^8) untuk komputasi). Kemudian hasil perkalian ini kembali diperlakukan sebagai *word* 32 bit yang merupakan output dari fungsi g ini.

$$x_i = \left\lfloor X / 2^{8i} \right\rfloor \bmod 2^8 \qquad i = 0,...,3$$
$$y_i = s_i\lfloor x_i \rfloor \qquad i = 0,...,3$$
$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} . & \cdots & . \\ . & MDS & . \\ . & \cdots & . \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$
$$Z = \sum_{i=0}^{3} z_i \cdot 2^{8i}$$

di mana s_i adalah S-Box *key-dependent* dan *Z* adalah hasil dari fungsi g. Untuk lebih jelas, berikut ini akan dibahas korespondensi antara nilai byte dengan elemen *field* GF(2^8).



Gambar 10 Skema Fungsi g

GF(2^8) direpresentasikan dengan GF(2)[x]/v(x) di mana $v(x) = x^8 + x^6 + x^5 + x^3 + 1$ adalah polinom primitif dengan derajat 8 pada GF(2). Elemen *field* $a = \sum_{i=0}^{7} a_i x^i$ dengan $a_i \in$ GF(2) diidentifikasi dengan nilai byte $\sum_{i=0}^{7} a_i 2^i$. Ini merupakan pemetaan "natural": penambahan dalam GF(2^8) berkoresponden dengan XOR pada byte.

Matriks MDS ditentukan sebagai berikut:

$$MDS = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix}$$

di mana seetiap elemen dituliskan dalam nilai heksadesimal.

3.5.3 *Key Schedule*

Key Schedule harus menyediakan 40 *word* kunci $K_0,...,K_{15}$ dan 4 S-Box *key-dependent* yang digunakan dalam fungsi g. Twofish didefinisikan untuk kunci dengan panjang N = 128, N = 192, dan N = 256. Kunci dengan panjang kurang dari 256 bit dapat digunakan dengan mem-*padding*-nya dengan nol sampai panjangnya mencapai salah satu dari ketiga panjang kunci tersebut yang terdekat.

Misalkan $k = N/64$. Kunci M terdiri dari $8k$ byte $m_0,...,m_{8k-1}$. Masing-masing byte dikonversi menjadi $2k$ *word* 32 bit.

$$M_i = \sum_{j=0}^{3} m_{(4i+j)} \cdot 2^{8j} \qquad i = 0,...,3$$

dan kemudian menjadi dua vektor *word* dengan panjang k.

$$M_e = (M_0 , M_2,, \dots , M_{2k-2})$$
$$M_o = (M_1 , M_3,, \dots , M_{2k-1})$$

Vektor *word* ketiga dengan panjang k juga diturunkan dari kunci dengan membagi kunci menjadi 8 grup, menginterpretasinya sebagai vektor pada GF(2^8), dan mengalikannya dengan matriks 4x8 yang diturunkan dari sebuah kode RS. Masing-masing 4 byte hasilnya kembali diinterpretasi sebagai *word* 32 bit. *Word* ini membentuk vektor ketiga.

$$\begin{pmatrix} s_{i,0} \\ s_{i,1} \\ s_{i,2} \\ s_{i,3} \end{pmatrix} = \begin{pmatrix} \cdot & \cdots & \cdot \\ \vdots & \text{RS} & \vdots \\ \cdot & \cdots & \cdot \end{pmatrix} \cdot \begin{pmatrix} m_{8i} \\ m_{8i+1} \\ m_{8i+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+6} \\ m_{8i+7} \end{pmatrix}$$

$$S_i = \sum_{j=0}^{3} s_{i,j} \cdot 2^{8j}$$

untuk i = 0,...,k-1, dan $S = (S_{k-1} , S_{k-2},, \dots , S_0)$. Perhatikan bahwa S berisi *word* dalam urutan terbalik. Untuk perkalian matriks RS, GF(2^8) direpresentasikan dengan GF(2)[x]/w(x) di mana $w(x) = x^8 + x^6 + x^5 + x^3 + 1$ adalah polinom primitif dengan derajat 8 pada GF(2) yang berbeda dari sebelumnya (pada matriks MDS). Namun pemetaan antara nilai byte dengan elemen GF(2^8) menggunakan definisi yang sama dengan yang digunakan pada matriks MDS.

$$\text{RS} = \begin{pmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 87 & 5A & 58 & DB & 9E & 03 \end{pmatrix}$$

Vektor M_e , M_o dan S membentuk basis dari *key schedule.*

3.5.2.1 Penambahan panjang kunci

Twofish dapat menerima kunci dengan panjang sampai 256 bit. Untuk panjang kunci yang tidak didefinisikan, kunci di-*padding* pada bagian akhir dengan byte nol sampai mencapai panjang terdekat yang didefinisikan. Misalnya, kunci dengan panjang 80 bit $m_0,..,m_9$ akan diperpanjang dengan men-set m_i=0 untuk i = 10,...,15 dan memperlakukan kunci ini sebagai kunci 128 bit.

3.5.2.2 Fungsi h

Fungsi ini menerima dua buah masukan (32 bit X dan list $L = (L_0,...,L_{k-1})$ berisi *word* 32 bit sebanyak) dan menghasilkan keluaran sebuah *word*. Funsi ini berlangsung dalam k tahap. Pada masing-masing tahap, keempat *word* akan dimasukkan ke dalam S-Box dan di-XOR-kan dengan byte yang didapatkan dari list. Selanjutnya, *word* sekali lagi dimasukkan ke dalam S-Box, dan dikalikan dengan matriks MDS seperti pada fungsi g.

3.5.2.3 S-Box *Key-dependent*

S-Box pada fungsi g didefinisikan sebagai

$$g(X) = h(X, S)$$

Sehingga, untuk i = 0,...,3, S-Box *key-dependent* dibentuk dengan pemetaan dari x_i ke y_i, dalam fungsi h, di mana list L equal dengan vektor S yang diturunkan dari kunci.

3.5.2.4 *Word* Kunci K_j

Word kunci didapatkan menggunakan fungsi h.

$$\begin{aligned} \rho &= 2^{24} + 2^{16} + 2^8 + 2^0 \\ A_i &= h(2i\rho, M_e) \\ B_i &= \text{ROL}(h((2i+1)\rho, M_o), 8) \\ K_{2i} &= (A_i + B_i) \bmod 2^{32} \\ K_{2i+1} &= \text{ROL}((A_i + 2B_i) \bmod 2^{32}, 9) \end{aligned}$$

Konstanta ρ digunakan untuk menduplikasi byte. Konstanta ini memiliki properti untuk i =

0,...,255, *word iρ* terdiri dari empat byte yang sama, masing-masing dengan nilai *i*. Fungsi h diterapkan untuk jenis *word* ini. Untuk A_i, nilai byte-nya adalah 2*i*, dan parameter kedua dari h adalah M_e. B_i dihitung dengan cara yang sama menggunakan 2*i*+1 sebagai nilai byte dan M_o sebagai parameter kedua, dengan rotasi tambahan 8 bit. Nilai A_i dan B_i dikombinasikan dalam PHT. Salah satu hasilnya kemudian dirotasi 9 bit. Kedua hasil tersebut membentuk dua *word* kunci.

3.5.2.5 Permutasi q_0 dan q_1

Permutasi q_0 dan q_1 adalah permutasi *fixed* dalam nilai 8 bit. Masing-masing dibangun dari empat permutasi 4 bit yang berbeda. Untuk input dengan nilai *x*, nilai output *y* yang berkoresponden didefinisikan sebagai berikut:

$$
\begin{aligned}
a_0, b_0 &= \lfloor x/16 \rfloor, x \bmod 16 \\
a_1 &= a_0 \oplus b_0 \\
b_1 &= a_0 \oplus \mathrm{ROR}_4(b_0, 1) \oplus 8a_0 \bmod 16 \\
a_2, b_2 &= t_0[a_1], t_1[b_1] \\
a_3 &= a_2 \oplus b_2 \\
b_3 &= a_2 \oplus \mathrm{ROR}_4(b_2, 1) \oplus 8a_2 \bmod 16 \\
a_4, b_4 &= t_2[a_3], t_3[b_3] \\
y &= 16\, b_4 + a_4
\end{aligned}
$$

di mana ROR_4 adalah fungsi ROR yang merotasikan nilai 4-bit. Pertama, *word* dibagi menjadi dua bagian. Kedua *word* dikombinasikan dalam tahap pengacakan bijektif. Masing-masing bagian kemudian dimasukkan ke dalam S-Box *fixed* 4-bit. Proses ini selanjutnya diikuti dengan tahap pengacakan lainnya dan S-Box. Terakhir, kedua bagian dikombinasikan kembali menjadi sebuah byte. Untuk permutasi q_0 S-Box-nya adalah sebagai berikut:

$$
\begin{aligned}
t_0 &= [\ 8\ 1\ 7\ \mathrm{D}\ 6\ \mathrm{F}\ 3\ 2\ 0\ \mathrm{B}\ 5\ 9\ \mathrm{E}\ \mathrm{C}\ \mathrm{A}\ 4\ ] \\
t_1 &= [\ \mathrm{E}\ \mathrm{C}\ \mathrm{B}\ 8\ 1\ 2\ 3\ 5\ \mathrm{F}\ 4\ \mathrm{A}\ 6\ 7\ 0\ 9\ \mathrm{D}\ ] \\
t_2 &= [\ \mathrm{B}\ \mathrm{A}\ 5\ \mathrm{E}\ 6\ \mathrm{D}\ 9\ 0\ \mathrm{C}\ 8\ \mathrm{F}\ 3\ 2\ 4\ 7\ 1\ ] \\
t_3 &= [\ \mathrm{D}\ 7\ \mathrm{F}\ 4\ 1\ 2\ 6\ \mathrm{E}\ 9\ \mathrm{B}\ 3\ 0\ 8\ 5\ \mathrm{C}\ \mathrm{A}\ ]
\end{aligned}
$$

Sedangkan S-Box pada q_1 adalah:

$$
\begin{aligned}
t_0 &= [\ 2\ 8\ \mathrm{B}\ \mathrm{D}\ \mathrm{F}\ 7\ 6\ \mathrm{E}\ 3\ 1\ 9\ 4\ 0\ \mathrm{A}\ \mathrm{C}\ 5\ ] \\
t_1 &= [\ 1\ \mathrm{E}\ 2\ \mathrm{B}\ 4\ \mathrm{C}\ 3\ 7\ 6\ \mathrm{D}\ \mathrm{A}\ 5\ \mathrm{F}\ 9\ 0\ 8\ ] \\
t_2 &= [\ 4\ \mathrm{C}\ 7\ 5\ 1\ 6\ 9\ \mathrm{A}\ 0\ \mathrm{E}\ \mathrm{D}\ 8\ 2\ \mathrm{B}\ 3\ \mathrm{F}\ ] \\
t_3 &= [\ \mathrm{B}\ 9\ 5\ 1\ \mathrm{C}\ 3\ \mathrm{D}\ \mathrm{E}\ 6\ 4\ 7\ \mathrm{F}\ 2\ 0\ 8\ \mathrm{A}\ ]
\end{aligned}
$$

4. Perbandingan Algoritma

Perbedaan yang paling mudah dilihat dari kelima cipher di atas adalah perbedaan jumlah putaran. Berikut ini adalah tabel perbandingan jumlah putaran dari kelima algoritma finalis AES. Namun perbandingan jumlah putaran tidak memperlihatkan perbandingan performansi algoritma. Oleh karena itu, lebih baik jika perbandingan dilakukan terhadap jumlah putaran maksimal yang tidak aman, karena berhasil dikriptanalisis dengan ketentuan tertentu [8].

Tabel Jumlah Putaran maksimal yang tidak aman

Algoritma	Jumlah putaran
MARS	9 dari 16
RC6	15 dari 20
Rijndael	8 dari 14
Serpent	9 dari 32
Twofish	6 dari 16

Perbedaan selanjutnya yang dapat dilihat adalah titik berat perancangan algoritma. Kelima algoritma menitikberatkan inti struktur dan pengembangan algoritma berbeda satu sama lainnya. Berikut ini adalah perbandingan mengenai struktur inti dan prinsip desain kelima algoritma.

Tabel 2 Perbandingan struktur inti dan prinsip desain

Algoritma	Struktur inti dan Prinsip desain
MARS	Jaringan Feistel yang dimodifikasi – Menggabungkan struktur DES
RC6	Jaringan Feistel – Memodifikasi RC5
Rijndael	Jaringan substitusi permutasi yang dimodifikasi – Square
Serpent	Jaringan substitusi dan permutasi – Bit lice
Twofish	Jaringan Feistel – Memodofikasi Blowfish

Kelima algoritma menggunakan inti Feistel konvensional dengan sedikit atau tanpa modifikasi – satu dengan Square dan satu lagi dengan bit lice. Kompleksitas algoritma berpengaruh pada ongkos implementasi baik dari segi pengembangan dengan *fixed resource* (batasan perangkat keras dan batasan code dan ukuran data perangkat lunak) maupun performansi *real time* untuk fixed resoource (*throughput*) [5].

Selain dari perbandingan di atas, sudah banyak percobaan yang telah dilakukan dalam membandingkan performansi kelima algoritma finalis AES. Pada umumnya perbandingan tersebut dilakukan dengan cara mengimplementasi kelima algoritma dalam suatu perangkat keras tertentu. Dari implementasi tersebut, diukur performansi implementasi kelima algoritma dengan mengacu pada kecepatan algoritma, kebutuhan

(*requirement*) terhadap resource, dan sebagainya.

Dari semua percobaan membandingkan kelima algoritma tersebut, hampis semuanya menghasilkan hasil yang berbeda dalam hal algoritma mana yang terbaik performansinya. Contohnya untuk implementasi dalam *smart card*, Serpent menjadi algoritma yang paling efisien, namun untuk implementasi pada suatu processor tertentu, Serpent justru menjadi algoritma yang membutuhkan waktu paling lama dalam proses enkripsi dan dekripsi, dan Rijndael menjadi algoritma yang paling cepat.

Perbedaan-perbedaan tersebut menunjukkan bahwa masing-masing algoritma memiliki kelebihan dan kekurangan masing-masing. Namun di samping itu, perbedaan ini mungkin disebabkan karena memang ada beberapa percobaan yang dilakukan oleh desainernya sendiri, jadi memungkinkan adanya bias dalam melakukan percobaan.

Walaupun banyak pihak yang membanding-bandingkan kelima algoritma, namun pemenang algoritma telah ditentukan. Rijndael mendapatkan 86 suara, mengungguli kelima algoritma finalis lainnya, yaitu Serpent dengan 59 suara, Twofish dengan 31 suara, diikuti oleh RC6 dengan 23 suara, dan terakhir adalah MARS dengan 13 suara. Tentunya pemilihan ini sudah melalui proses yang ketat untuk mendapatkan suatu standard baru menggantikan AES. Dalam prakteknya, kelima algoritma cukup luas digunakan dalam berbagai keperluan.

5. Kesimpulan

1. Algoritma simetri *cipher block* merupakan salah satu algoritma kriptografi yang cukup handal terbukti dengan dalam pemilihan AES, yang akan menjadi standard enkripsi, harus menggunakan algoritma kriptografi simetri *cipher block*.
2. Algoritma cipher block tidak harus didesain mengikuti prinsip perancangan yang ada, karena dapat didesain dengan memodifikasi prinsip-prinsip yang biasa digunakan dalam perancangan algoritma *cipher block*. Contohnya adalah dengan memodifikasi jaringan Feistel atau dengan memodifikasi pembangkitan S-Box.
3. Beberapa pembatasan pada persyaratan yang diajukan dalam pemilihan AES justru membuat beberapa algoritma menjadi tidak maksimal. Contohnya adalah penentuan panjang kunci, yang sebenarnya jika diserahkan kepada para kandidat untuk menentukan sendiri, tentunya akan lebih optimal karena desainernya tentu yang lebih mengetahui panjang kunci optimal bagi algoritmanya.
4. Kelima algoritma finalis AES memiliki kelebihan dan kekurangannya masing-masing, karena walaupun telah ditetapkan Rijndael sebagai standar enkripsi, namun penggunaan algoritma lainnnya masih cukup luas.

DAFTAR PUSTAKA

[1] Anderson, Ross, et. al. (1998) Serpent: A Proposal for the Advanced Encryption Standard

[2] Daemen, Joan, Vincent Rijmen. (2004). The *Rijndael* Specification.

[3] IBM Corporation Team. (1999). MARS – A Candidate Cipher for AES.Schneier,

[4] Munir, Rinaldi. (2004). Bahan Kuliah IF5054 Kriptografi. Departemen Teknik Informatika, Institut Teknologi Bandung.

[5] Patel, Dhiren R. (2000). The AES Winner.

[6] Rivest, Ronald, et. al. (1998). The RC6™ Block Cipher

[7] Bruce, et. al. (1998). Twofish: A 128-bit Block Cipher.

[8] Schneier, Bruce, Whiting Doug. (2000) A Performance Comparison of the AES Finalists