# KRIPTANALISIS PADA BLOCK CIPHER

Satria Putra Sajuthi – NIM : 13503099

*Program Studi Teknik Informatika, Institut Teknologi Bandung*
*Jl. Ganesha 10, Bandung*
E-mail : if13099@students.if.itb.ac.id

**Abstrak**

Makalah ini membahas mengenai berbagai macam serangan pada block cipher seperti AES, rijndael, serpent. Serangan terhadap blok cipher yang dibahas di sini dimulai dari differential attack dan liniear cryptanalisis sampai serangan-serangan yang lebih baru yang melibatkan pembuatan sistem persamaan dari cipher seperti XLS *attack*, XL *attack*, MQ *attack* dan *algebraic attack*. Untuk melakukan analsis serangan, akan dilakukan studi kasus terhadap dua algoritma blok cipher yaitu Rijndael dan Serpent
**Kata kunci**: *AES, Rijndael, Serpent, MQ Problem, multivariate polinomial equations,algebraic attack*

## 1. Pendahuluan

Bapak dari teori informasi, Claude Shannon pernah berkata untuk memecahkan suatu cipher yang baik, membutuhkan "as much work as solving a system of simultaneous equations in a large number of unknowns of a complex type." usaha yang sebanding dengan memecahkan suatu sistem dari persamaan simultan dengan banyak variabel yang tidak diketahui'.

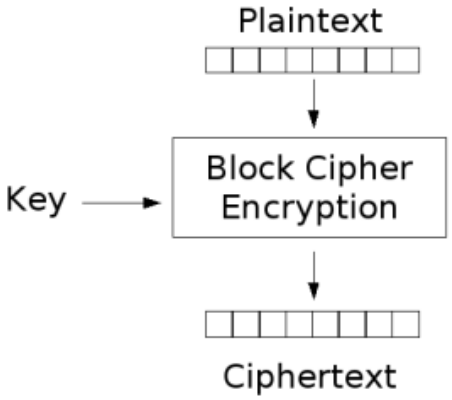
Para peneliti mengenai kunci simetrik kriptografi berfokus kepada aspek local dan aspek statistic pada cipher, dan melupakan aspek global dari masalah tersebut. Kuncinya didefinisikan sebagai solusi dari sistem persamaan aljabar yang mendeskripsikan keseluruhan cipher. Sistem ini tidak boleh terlalu sederhana sehingga memungkinkan seesorang untuk memecahkannya.

Yang menjadi permasalahan adalah munculnya sistem persamaan untuk merepresentasikan cipher. AES yang merupakan standar yang dipakai untuk enkripsi pada saat ini juga didasarkan pada sistem sederhana yang berisi fungsi-fungsi aljabar. Hal ini menimbulkan kekhawatiran karena setiap kunci simetrik(blok cipher, stream cipher dan hash function) dapat diserang dengan algebraic attack.

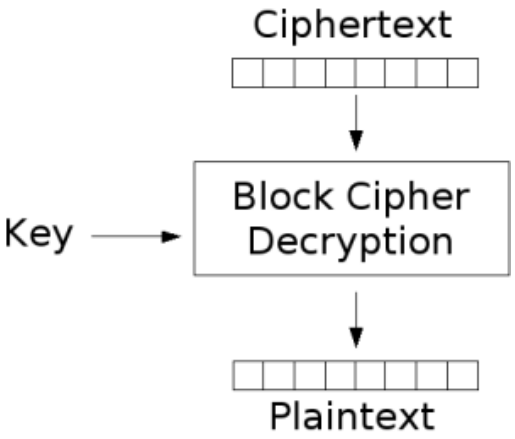
## 2. Block Cipher

Dalam kriptografi, cipher blok merupakan cipher kunci simetrik yang beroperasi pada kumpulan bit dengan panjang yang tetap yang disebut sebagai blok dengan berbagai macam transformasi. Pada saat enkripsi, cipher blok akan menggunakan n-bit blok plaintext sebagai input dan menghasilkan n-bit blok sebagai output. Perubahan ini dikontrol dengan menggunakan mekanisme kunci rahasia.

Secara umum, cipher blok mempunyai 2 algoritma yang berpasangan. Algortima untuk enkripsi (E) dan algoritma untuk dekripsi (D). Kedua algoritma tersebut menerima 2 masukan yaitu blok dengan ukuran n-bit dan kunci dengan ukuran k bit yang akan menghasilkan output dengan ukuran n bit.




**Gambar 1 Enkripsi pada blok cipher**




**Gambar 2 Dekripsi pada blok cipher**

Untuk membangun suatu algoritma blok cipher, maka diperlukan beberapa teknik berikut:

a. Substitusi.
Teknik ini mengganti satu atau sekumpulan bit ada blok plainteks tanpa mengubah urutannya. Secara matematis, teknik substitusi ini ditulis sebagai $c_i = E(p_i)$, i=1,2,... yang dalam hal ini $c_i$ adalah bit cipherteks, $p_i$ adalah bit plainteks, dan f adalah fungsi substitusi. Dalam praktek, E dinyatakan sebagai fungsi matematis atau dapat merupakan tabel substitusi (S-box).

b. Transposisi atau permutasi.
Teknik ini memindahkan posisi bit pada blok plainteks berdasarkan aturan tertentu. Secara matematis, teknik transposisi ini ditulis sebagai C=PM uang dalam hal ini C adalah blok cipherteks, P adalah blok plainteks dan M adalah fungsi transposisi. Dalam praktek, M dinyatakan sebagai tabel atau matriks permutasi (P=box)

c. Ekpansi dan kompresi
Teknik ekspansi memperbanyak jumlah bit semula sedangkan teknik kompresi menciutkan jumlah bit semula. Dalam prakteknya, kedua teknik ini direpresentasikan dalam bentuk tabel.

Pada tahun 1949, Claude shannon mempublikasikan makalahnya yang berjudul ”Communication Theory of Secrecy Systems”. Di dalam makalah ini terdapat 2 prinsip penyandian yang sering digunakan dalam blok cipher. Kedua prinsip tersebut adalah:

a. *Confusion*
Prinsip ini menyembunyikan hubungan apapun yang ada antara plainteks, cipherteks dan kunci. Prinsip *confusion* akan membuat kriptanalis frustasi untuk mencari pola-pola statistic yang muncul pada cipher. Penerapan prinsip ini secara benar akan membuat hubungan statistic antara plainteks, cipherteks dan kunci menjadi sangat rumit.

b. *Diffusion*
Prinsip ini menyebarkan pengaruh satu bit plainteks atau kunci ke sebanyak mungkin cipherteks. Contoh : pengubahan kecil pada plainteks sebanyak satu atau dua bit akan menghasilkan perubahan pada cipherteks yang tidak dapat diprediksi. Prinsip diffusion juga menyembunyikan hubungan statistik antara plainteks, cipherteks, dan kunci untuk membuat kriptanalisis menjadi sulit. Untuk mendapatkan keamanan yang bagus, prinsip confusion dan diffusion diulang berkali-kali pada sebuah blok tunggal dengan kombinasi yang berbeda-beda.

### 2.1 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*)
2. *Cipher Block Chaining* (*CBC*)
3. *Cipher Feedback* (*CFB*)
4. *Output Feedback* (*OFB*)

#### 2.1.1 *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

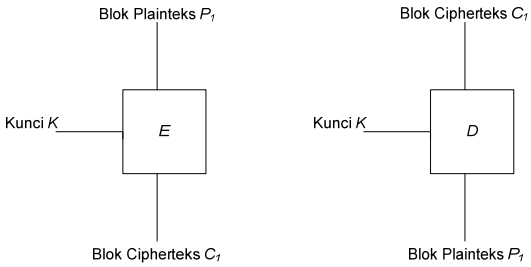
$$C_i = E_k(P_i)$$

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 dan dekripsi dengan mode *ECB* dapat dilihat pada Gambar 2.



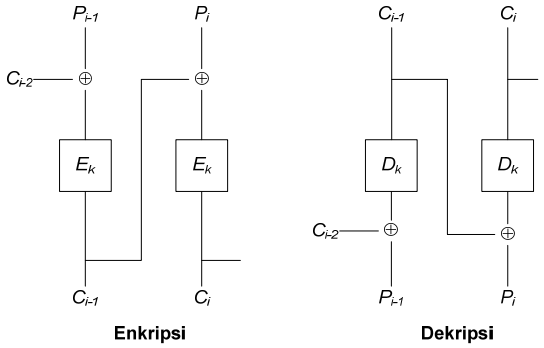

**Gambar 3 Skema Enkripsi dan Dekripsi dengan Mode *ECB***

Ada kemungkinan panjang plainteks tidak habis dibagi dengan panjang ukuran blok yang ditetapkan. Hal ini mengakibatkan blok terakhir berukuran lebih pendek daripada blok-blok lainnya. Satu cara untuk mengatasi hal ini adalah dengan *padding*, yaitu menambahkan blok terakhir dengan pola bit yang teratur agar panjangnya sama dengan ukuran blok yang ditetapkan.

### 2.1.2 *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*. 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.

Dekripsi dilakukan dengan memasukkan blok cipherteks yang *current* ke fungsi dekripsi, kemudia meng-*XOR*-kan hasilnya dengan blok cipherteks sebelumnya. Dalam hal ini, blok cipherteks sebelumnya berfungsi sebagai umpan maju (*feedforward*) pada akhir proses dekripsi. Skema enkripsi dan dekripsi dengan mode *CBC* dapat dilihat pada Gambar 3.




**Gambar 4 Enkripsi dan Dekripsi dengan Mode *CBC***

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.

Pada mode *CBC*, blok plainteks yang sama menghasilkan blok cipherteks yang berbeda hanya jika blok-blok plainteks sebelumnya berbeda.

### 2.1.3 *Cipher-Feedback* (*CFB*)

Pada mode *CFB*, data dienkripsikan dalam unit yang lebih kecil daripada ukuran blok. Unit yang dienkripsikan dapat berupa bit per bit, 2 bit, 3 bit, dan seterusnya. 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 $n \leq m$ ($m$ = ukuran blok). Mode *CFB* membutuhkan sebuah antrian (*queue*) yang berukuran sama dengan ukuran blok masukan.

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

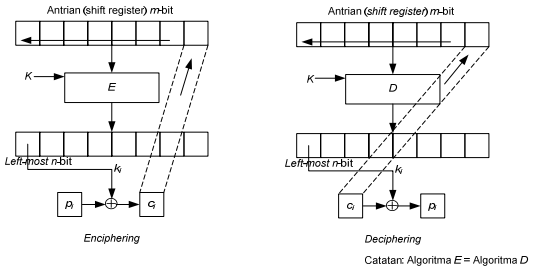
1. Antrian diisi dengan *IV* (*initialization vector*).
2. Enkripsikan antrian dengan kunci *K*. *n* bit paling kiri dari hasil enkripsi berlaku sebagai *keystream* ($k_i$) yang kemudian di-*XOR*-kan dengan *n*-bit dari plainteks menjadi *n*-bit pertama dari cipherteks. Salinan (*copy*) *n*-bit dari 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.




**Gambar 5 Mode *CFB* n-bit**

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:

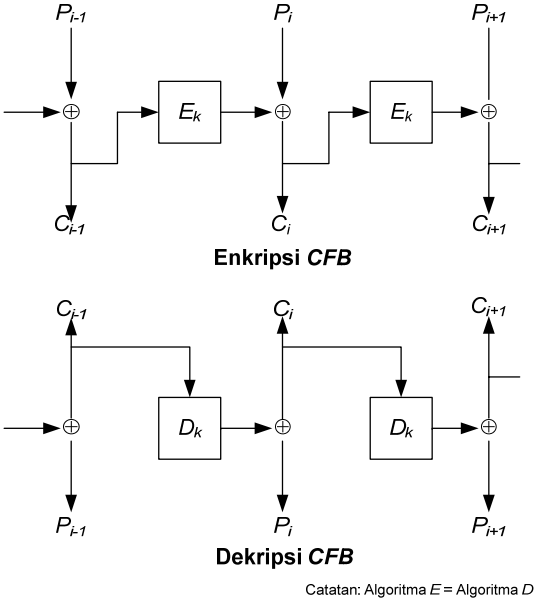
$$C_i = P_i \oplus MSB_m(E_k(X_i))$$
$$X_{i+1} = LSB_{m-n}(X_i) \,||\, C_i$$

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*




**Gambar 6 Enkripsi dan Dekripsi Mode *CFB* *n*-bit untuk blok *n*-bit**

Jika $m = n$, maka mode *CFB* *n*-bit adalah seperti pada Gambar 5. *CFB* menggunakan skema umpan balik dengan mengaitkan blok plainteks bersama-sama sedemikian sehingga cipherteks bergantung pada semua blok plainteks sebelumnya. Skema enkripsi dan dekripsi dengan mode *CFB* dapat dilihat pada Gambar 5.
Dari Gambar 5 dapat dilihat bahwa:

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

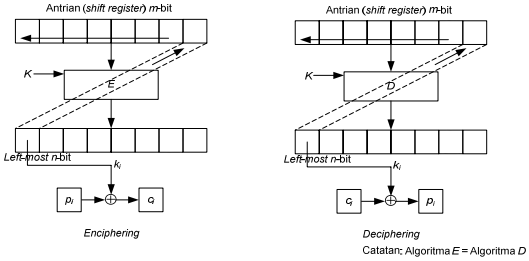
*IV* pada *CFB* tidak perlu dirahasiakan. *IV* harus unik untuk setiap pesan, sebab *IV* yang sama untuk setiap pesan yang berbeda akan menghasilkan *keystream* $k_i$ yang sama.

### 2.1.4 *Output-Feedback* (*OFB*)

Pada mode *OFB*, data dienkripsikan dalam unit yang lebih kecil daripada ukuran blok. Unit yang dienkripsikan dapat berupa bit per bit, 2 bit, 3 bit, dan seterusnya. Bila unit yang dienkripsikan satu karakter setiap kalinya, maka mode *OFB*-nya disebut *OFB* 8-bit.

Secara umum *OFB n*-bit mengenkripsi plainteks sebanyak *n* bit setiap kalinya, yang mana $n \leq m$ ($m$ = ukuran blok). Mode *OFB* membutuhkan sebuah antrian (*queue*) yang berukuran sama dengan ukuran blok masukan.
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.

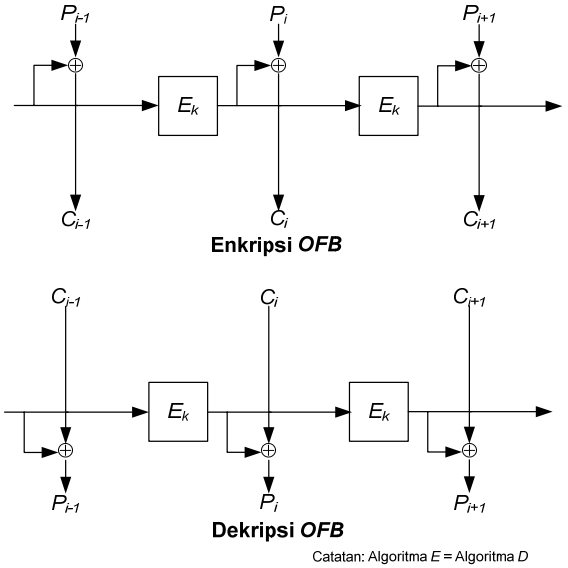



**Gambar 7 Mode *OFB* n-bit**

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. Mode *OFB n*-bit yang bekerja pada blok berukuran *m*-bit dapat dilihat pada Gambar 6.




**Gambar 8 Enkripsi dan Dekripsi *OFB n*-bit untuk blok n-bit**

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

Proses Enkripsi:

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

Proses Dekripsi:

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

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*

Jika $m = n$, maka mode *OFB n*-bit adalah seperti pada Gambar 6. *OFB* menggunakan skema umpan balik dengan mengaitkan blok plainteks bersama-sama sedemikian sehingga cipherteks bergantung pada semua blok plainteks sebelumnya. Skema enkripsi dan dekripsi dengan mode *OFB* dapat dilihat pada Gambar 7.

## 3.Rijndael

### 3.1 Latar Belakang Rijndael

Rijndael merupakan blok cipher yang diadopsi menjadi standar enkripsi oleh pemerintahan Amerika Serikat untuk menggantikan standard enkripsi yang lama (Data Encryption Standard). Rijndael merupakan pemenang dari sayembara yang dilakukan oleh NIST(National Institute of Standards and Technology) untuk dijadikan standar enkripsi baru yang bernama AES (Advanced Encryption Standard).

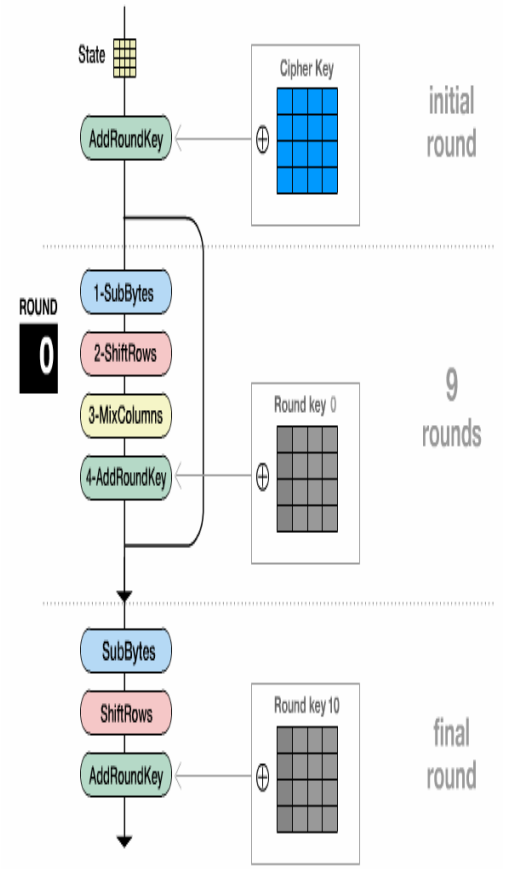
Algoritma Rjindael ini diciptakan oleh dua orang kriptografer asal Belgia yang bernama John Daemen dan Vincent Rijmen. Nama Rijndael pun berasal diambil dari gabungan nama belakang kedua ilmuwan tersebut. Algoritma ini terinspirasi dari algoritma sejenis yang bernama Square.

### 3.2 Deskripsi Cipher Rijndael

Ukuran blok pada algoritma rjindael bervariasi. Minimum ukuran blok adalah 128 bits sedangkan maksimumnya adalah 256 bits. Ukuran kunci juga bervariasi antara 128-256 bits dengan kelipatan 32 bits.

Tidak seperti DES yang menggunakan jaringan feister, struktur yang digunakan dalam melakukan iterasi blok oleh Rjindael adalah jaringan substitusi-permutasi. Jumlah roundnya 10,12,14 tergantung dari ukuran kunci.

### 3.3 Langkah-langkah algoritma Rijndael

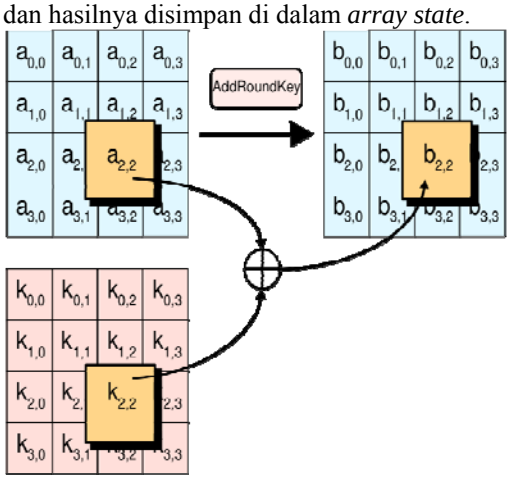



**Gambar 9 Skema utama algoritma Rijndael**

Seperti terlihat pada gambar di atas, algoritma ini terdiri dari 3 tahap yaitu tahap awal(*initial round*), tahap putaran, dan tahap akhir(*final round*). Pada tahap awal dilakukan XOR antara blok plaintext dengan *cipher key*. Setelah itu dilakukan *subBytes*, *shiftRows*, *MixColumns*, dan *AddRoundKey* sebanyak jumlah putaran dikurangi satu. Pada tahap terakhir dilakukan *subBytes*, *shiftRows*, dan *MixColumns*.

#### 3.3.1 *AddRoundKey*

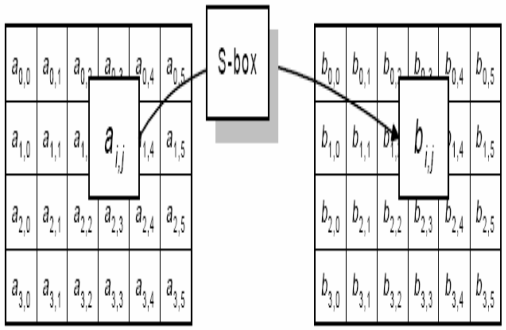
Transformasi ini melakukan operasi XOR terhadap sebuah round key dengan *array state* dan hasilnya disimpan di dalam *array state*.



**Gambar 10 *AddRoundKey***

### 3.3.2 *SubBytes*

Transformasi ini memetakan setiap byte dari array state dengan menggunakan table substitusi S-box. Tidak seperti DES yang mempunyai banyak S-box untuk menangani setiap putaran, Rijndael hanya mempunyai satu buah S-box untuk semua putaran.

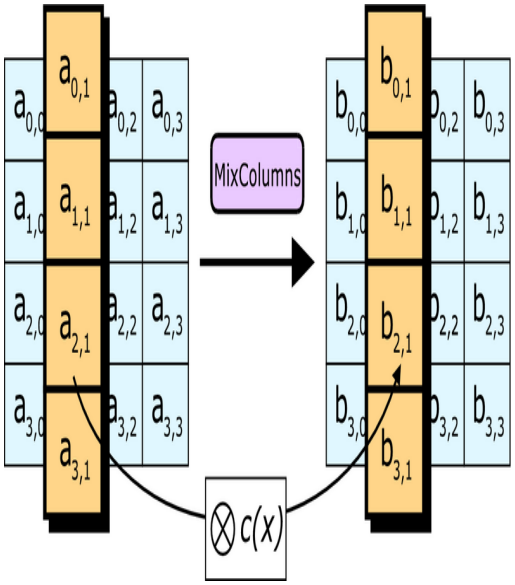



**Gambar 11 *SubBytes***

### 3.3.3 *MixColumns*

Transformasi MixColumns mengalikan setiap kolom dari array state dengan polinom a(x) mod ($x^4 + 1$). MixColumns memberikan efek difusi pada cipher. Setiap kolom diperlakukan sebagai polinom 4-suku pada GF($2^8$). Polinom a(x) yang ditetapkan adalah:
$a(x) = \{03\}x^3 + \{01\}\ x^2 + \{01\}x + \{02\}$

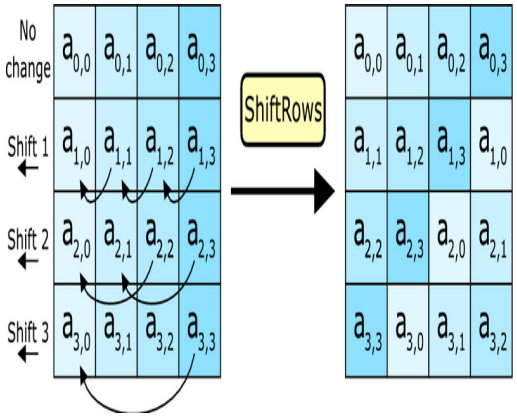
Transformasi ini dinyatakan sebagai perkalian matriks $s'(x) = a(x)$ x $s(x)$.




**Gambar 12 *Mix Columns***

### 3.3.4 *ShiftRows*

Transformasi *ShiftRows* melakukan pergeseran secara wrapping(siklik) pada 3 baris terakhir dari array state. Jumlah pergeseran bergantung dari nilai baris(r). Baris r=1 digeser sejauh 1 byte, baris r=2 digeser sejauh 2 byte, dan baris r=3 digeser sejauh 3 byte. Baris r=0 tidak digeser.




**Gambar 13 *ShiftRows***

### 3.4 Pembangkitan Round Key

*Round key* diturunkan dari kunci luar(*Cipher Key*) dengan cara penjadwalan kunci(*key schedule*). Proses penjadwalan ini terdiri dari dua komponen yaitu ekspansi kunci dan seleksi *round key*. Prinsip dasarnya adalah sebagai berikut :
- Jumlah bit dari *Round Key* sama dengan panjang blok dikalikan dengan jumlah round ditambah satu.
- *Cipher Key* diekspansi menjadi *expanded key*
- Round key diambil dari expanded key dengan cara mengambil seukuran blok sesuai dengan nomor roundnya.

### 3.5 Formula aljabar dari Rijndael

S-box pada rijndael dapat ditulis dengan persamaan

$$S(x) = w_8 + \sum_{d=0}^{7} w_d x^{255-2^d}$$

untuk konstanta tertentu wo,..,w8.

Penyederhanaan pertama yang dapat dilakukan adalah dengan menghilangkan konstanta w8 dari formula tersebut. Dalam putaran normal, output dari keempat S box dikalikan dengan matriks MDS dan 4 kunci putaran ditambahkan ke hasil kali matriks tersebut. Karena kunci dan MDS matriks bersifat linier, w8 dalam S-box dapat digantikan dengan penambahan suatu konstanta dengan kunci

putaran. Pada putaran terakhir tidak terdapat MDS matriks, tetapi terdapat penambahan kunci putaran sehingga trik di atas tetap berlaku. Hal ini mengubah formula di atas menjadi

$$S(x) = \sum_{d=0}^{7} w_d x^{255-2^d}$$

dan yang harus diperhatikan lagi adalah jadwal kunci yang harus dimodifikasi dimana setiap expanded key ditambahkan dengan suatu konstanta yang cocok.

Penyederhanaan selanjutnya yang dapat dilakukan adalah dengan mengubah persamaan di atas menjadi

$$S(x) = \sum_{d=0}^{7} w_d x^{-2^d}$$

dengan mengasumsikan $x^{255}$ =1 untuk semua x kecuali x=0.

Persamaan di atas dapat dijelaskan dengan struktur dari S-box. S-box terdiri dari pembalikan dalam GF($2^8$) dengan 0 dipetakan ke 0, diikuti dengan 1 bit fungsi linier, yang diikuti lagi dengan penambahan suatu konstanta. Penambahan konstanta ini dapat dipindahkan ke subsistem penjadwalan kunci sehingga dapat diindahkan. Fungsi linier dapat diekspresikan sebagai GF($2^8$) dimana setiap exponen merupakan pangkat dua. Cara termudah untuk melihat ini adalah dengan mengamati bahwa kuadrat dalam GF($2^8$) merupakan 1 bit operasi linier. $(a+b)^2 = a^2+b^2$ dalam GF($2^8$).

### 3.5.1 Persamaan satu putaran

$a_{i,j}^{(r)}$ menandakan byte pada posisi(i,j) dengan input pada putaran ke-r. Seperti biasanya, nilai state dalam rijndael direpresentasikan dengan matriks 4 X 4 bytes yang setiap baris dan kolomnya dimulai dari 0 sampai 3.

Langkah pertama dalam putaran normal adalah SubBytes menggunakan S-box.

$$s_{i,j}^{(r)} = S[a_{i,j}^{(r)}] = \sum_{d_r=0}^{7} w_{d_r} (a_{i,j}^{(r)})^{-2^{d_r}}$$

dimana $s_{i,j}^{(r)}$ merupakan state setelah dilakukan subByte.

Langkah selanjutnya adalah operasi shiftRow yang dapat dituliskan sebagai berikut

$$t_{i,j}^{(r)} = s_{i,i+j}^{(r)} = \sum_{d_r=0}^{7} w_{d_r} (a_{i,i+j}^{(r)})^{-2^{d_r}}$$

Langkah ketiga adalah MixColumn yang dapat dituliskan sebagai berikut

$$m_{i,j}^{(r)} = \sum_{e_r=0}^{3} v_{i,e_r} t_{e_r,j}^{(r)}$$

dimana $v_{i,j}$ merupakan koefisien dari matriks MDS. Substitusi sederhana akan memberikan

$$\begin{aligned} m_{i,j}^{(r)} &= \sum_{e_r=0}^{3} v_{i,e_r} \sum_{d_r=0}^{7} w_{d_r} (a_{e_r,e_r+j}^{(r)})^{-2^{d_r}} \\ &= \sum_{e_r=0}^{3} \sum_{d_r=0}^{7} w_{i,e_r,d_r} (a_{e_r,e_r+j}^{(r)})^{-2^{d_r}} \end{aligned}$$

untuk beberapa konstanta $w_{i,j,k}$.

Langkah terakhir adalah penambahan dengan kunci putaran yang outputnya merupakan masukan untuk putaran selanjutnya.

$$\begin{aligned} a_{i,j}^{(r+1)} &= m_{i,j}^{(r)} + k_{i,j}^{(r)} \\ &= k_{i,j}^{(r)} + \sum_{e_r=0}^{3} \sum_{d_r=0}^{7} w_{i,e_r,d_r} (a_{e_r,e_r+j}^{(r)})^{-2^{d_r}} \end{aligned}$$

dimana $k_{i,j}^{(r)}$ merupakan kunci putaran r pada posisi (i,j). Formula di atas tersebut dapat ditulis dengan beberapa cara

$$a_{i,j}^{(r+1)} = k_{i,j}^{(r)} + \sum_{\substack{e_r \in \mathcal{E} \\ d_r \in \mathcal{D}}} w_{i,e_r,d_r} (a_{e_r,e_r+j}^{(r)})^{-2^{d_r}} \quad (1)$$

$$a_{i,j}^{(r+1)} = k_{i,j}^{(r)} + \sum_{f_r=0}^{31} w_{i,f_r} (a_{\lfloor f_r/8 \rfloor, \lfloor f_r/8 \rfloor + j}^{(r)})^{-2^{f_r}} \quad (2)$$

$$a_{i,j}^{(r+1)} = k_{i,j}^{(r)} + \sum_{f_r=0}^{31} w_{i,f_r} (a_{f_r,f_r+j}^{(r)})^{-2^{\lfloor f_r/4 \rfloor}} \quad (3)$$

Persamaan (1) merupakan hasil penulisan yang lebih ringkas dari persamaan sebelumnya dengan ε = {0,..,3} dan D = {0,..,7}. Persamaan (2) diturunkan dengan mengubah $f_r = 8e_r + d_r$, $w_{i,j}$ merupakan konstanta yang cocok. Persamaan (3) diturunkan dengan cara yang mirip, dengan mengubah $f_r = 4d_r + e_r$. Dari ketiga persamaan tersebut, persamaan (1) merupakan persamaan yang paling elegan dan akan digunakan untuk representasi selanjutnya.

### 3.5.2 Persamaan banyak putaran

Ekspresi dari persamaan banyak putaran dapat diturunkan dengan melakukan substitusi. Contohnya untuk 2 putaran Rijndael maka persamaannya adalah

$$a_{i,j}^{(3)} = k_{i,j}^{(2)} + \sum_{\substack{e_2\in\mathcal{E}\\ d_2\in\mathcal{D}}} \frac{w_{i,e_2,d_2}}{\left(k_{e_2,e_2+j}^{(1)} + \sum_{\substack{e_1\in\mathcal{E}\\ d_1\in\mathcal{D}}} \frac{w_{e_2,e_1,d_1}}{\left(a_{e_1,e_1+e_2+j}^{(1)}\right)^{2^{d_1}}}\right)^{2^{d_2}}} \qquad (4)$$

dan untuk 3 putaran persamaannya adalah

$$a_{i,j}^{(4)} = k_{i,j}^{(3)} + \sum_{\substack{e_3\in\mathcal{E}\\ d_3\in\mathcal{D}}} \frac{w_{i,e_3,d_3}}{\left(k_{e_3,e_3+j}^{(2)} + \sum_{\substack{e_2\in\mathcal{E}\\ d_2\in\mathcal{D}}} \frac{w_{e_3,e_2,d_2}}{\left(k_{e_2,e_2+e_3+j}^{(1)} + \sum_{\substack{e_1\in\mathcal{E}\\ d_1\in\mathcal{D}}} \frac{w_{e_2,e_1,d_1}}{\left(a_{e_1,e_1+e_2+e_3+j}^{(1)}\right)^{2^{d_1}}}\right)^{2^{d_2}}}\right)^{2^{d_3}}}$$

Penerapan Freshman's Dream pada persamaan (4) akan menghasilkan

$$a_{i,j}^{(3)} = k_{i,j}^{(2)} + \sum_{\substack{e_2\in\mathcal{E}\\ d_2\in\mathcal{D}}} \frac{w_{i,e_2,d_2}}{\left(k_{e_2,e_2+j}^{(1)}\right)^{2^{d_2}} + \sum_{\substack{e_1\in\mathcal{E}\\ d_1\in\mathcal{D}}} \frac{w_{e_2,e_1,d_1}^{2^{d_2}}}{\left(a_{e_1,e_1+e_2+j}^{(1)}\right)^{2^{d_1+d_2}}}}$$

dimana setiap perpangkatan bersifat individu. Setiap *subscript* menjadi semakin kompleks setiap kali masuk lebih dalam ke dalam struktur rekursif, tetapi setiap *subscript* diketahui dan independen terhadap kunci maupun plainteks. Untuk kemudahan penulisan maka digunakan notasi K untuk kunci ekpansi, notasi C untuk mengkodekan semua konstanta walaupun tidak bernilai sama, semua subscript digantikan dengan notasi *. Penerapan notasi tersebut sebagai berikut

$$a_{i,j}^{(3)} = K + \sum_{\substack{e_2\in\mathcal{E}\\ d_2\in\mathcal{D}}} \frac{C}{K^* + \sum_{\substack{e_1\in\mathcal{E}\\ d_1\in\mathcal{D}}} \frac{C}{p_*^* + K^*}}$$

Untuk rumus 5 putaran dapat dituliskan sebagai berikut

$$a_{i,j}^{(6)} = K + \sum_{\substack{e_5\in\mathcal{E}\\ d_5\in\mathcal{D}}} \frac{C}{K^* + \sum_{\substack{e_4\in\mathcal{E}\\ d_4\in\mathcal{D}}} \frac{C}{K^* + \sum_{\substack{e_3\in\mathcal{E}\\ d_3\in\mathcal{D}}} \frac{C}{K^* + \sum_{\substack{e_2\in\mathcal{E}\\ d_2\in\mathcal{D}}} \frac{C}{K^* + \sum_{\substack{e_1\in\mathcal{E}\\ d_1\in\mathcal{D}}} \frac{C}{K^* + p_*^*}}}}} \qquad (5)$$

## 4.Serpent

### 4.1 Latar Belakang Serpent

Seperti halnya Rjindael, serpent merupakan algoritma yang dilombakan untuk menjadi standar enkripsi AES(Advanced Encryption Standard).

Algoritma ini dirancang oleh Ross Anderson, Eli Biam dan Lars Knudsen.

### 4.2 Deskripsi Cipher Serpent

Blok Cipher dari Serpent memiliki panjang 128 bits dan mendukung panjang kunci 128 bit, 192 bit, dan 256 bit. Semua nilai dalam cipher ini direpresentasikan sebagai aliran bit dengan index yang dimulai dari 0, 0 sampai 31 untuk 32 bit word, 0 sampai 127 untuk 128 bits blok, 0 sampai 255 untuk 256 bits kunci dan seterusnya. Semua komputasi internal dilakukan dengan mode little-endian.

Seperti halnya Rijndael, serpent juga menggunakan struktur jaringan substitusi-permutasi. Jumlah putarannya adalah 32 dan beroperasi pada 4 * 32 bit words.

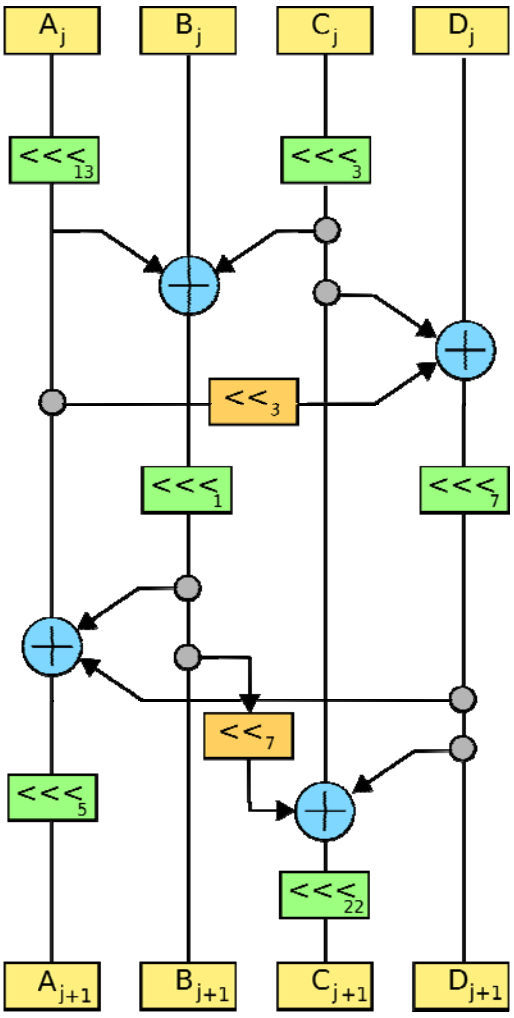
### 4.3 Langkah-langkah algoritma Serpent

Serpents mengenkripsikan 128 bit plaintext P menjadi 128 bit ciphertext C dalam 32 putaran di bawah kendali 33 *subkeys* yang masing-masing berukuran 128 bit. Ada 3 macam ukuran kunci yaitu 128 bit, 192 bit dan 256 bit;

kunci dengan ukuran tidak pas dengan 3 macam kunci tersebut dan kurang dari 256 bits akan ditambahkan *padding*. Penambahan padding dilakukan dengan menambahkan bit '1' pada posisi *Most Significant Bits* yang diikuti dengan penambahan 0 untuk memenuhi 256 bits.

Algoritmanya sendiri terdiri dari:

- Permutasi awal (initial permutation) IP
- 32 putaran, yang masing-masing putaran mengandung operasi perpaduan kunci(key mixing), substitusi dengan S-box, dan untuk setiap putaran kecuali putaran terakhir akan dilakukan transformasi linier. Pada putaran terakhir transformasi linier ini digantikan dengan tambahan operasi *key mixing*.
- Permutasi Akhir(Final permutation) FP




**Gambar 14 Transformasi Linier**

Permutasi awal dan akhir tidak mempunyai arti yang signifikan dalam kriptografi. Kedua permutasi ini digunakan untuk optimisasi implementasi dari cipher dan untuk meningkatkan efisiensi dari komputasi

## 5.Kriptanalisis

Serangan adalah setiap usaha atau percobaan yang dilakukan oleh kriptanalis untuk menemukan kunci atau menemukan plainteks dari cipherteksnya. Dalam membahas setiap serangan terhadap kriptografi, kita selalu mengasumsikan kriptanalis mengetahui algoritma kriptografi yang digunakan.

Berdasarkan ketersediaan data, serangan ini dapat dikelompokkan dengan beberapa cara

1. *Chipertext-only attack*
   Kriptanalis memiliki beberapa cipherteks dari beberapa pesan, semuanya dienkripsi dengan algoritma yang sama.
2. *Known-plaintext attack*
   Beberapa pesan yang formatnya terstruktur membuka peluang kepada kriptanalis untuk menerka plainteks dari cipherteks yang bersesuaian
3. *Chosen-plaintext attack*
   Serangan jenis ini lebih hebat daripada *known plainteks attack*, kerna kriptanalis dapat memilih plainteks tertentu untuk dienkripsikan, yaitu plainteks-plainteks yang lebih mengarahkan penemuan kunci
4. *Adaptive-chosen-plaintex attack*
   Kasus khusus dari jenis serangan nomor 3 di atas. Misalnya, kriptanalis memilih blok plainteks yang besar, lalu dienkripsi, kemudian memilih blok lainnya yang lebih kecil berdasarkan hasil serangan sebelumnya, begitu seterusnya
5. *Chosen-ciphertext attack*
   Kriptanalis memiliki akses terhadap cipherteks yang didekripsi (misalnya terhadap mesin elektronik yang melakukan dekripsi secara otomatis)
6. *Chosen-text attack*
   Gabungan *chosen-plaintext* attack dan *chosen-ciphertext* attack.

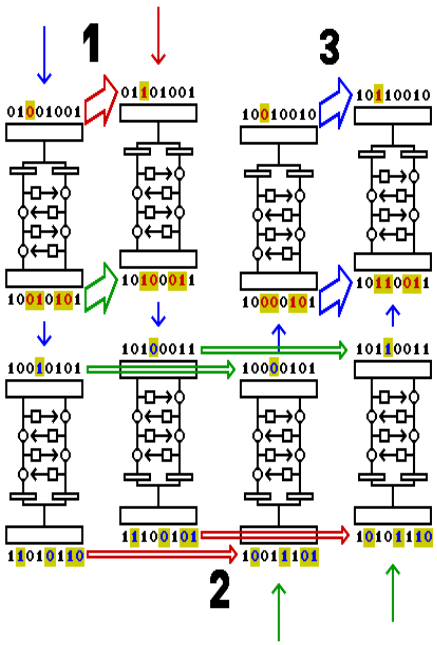
### 5.1 Differential kriptanalisis

Differential kriptanalisis merupakan salah satu teknik kriptanalisis yang digunakan pada blok cipher. Teknik ini pertama kali ditemukan oleh Eli Biham dan Adi Shamir pada tahun 1980 yang mempublikasikan berbagai macam

serangan terhadap berbagai macam blok cipher dan fungsi hash, termasuk kelemahan terhadap DES. Differential kriptanalisis termasuk jenis chosen plaintext attack, yang berarti bahwa kriptanalis harus dapat mengenkripsi plainteks sesuai dengan pilihannya. Skema ini dapat mengenkripsi DES dengan orde $2^{47}$ chosen plaintext. Metodenya adalah dengan cara menggunakan sepasang plaintext yang mempunyai perbedaan yang konstan. Umumnya perbedaan tersebut didefinisikan dengan operasi XOR. Kriptanalis kemudian menghitung perbedaan dari cipherteks yang bersangkutan untuk melihat pola-pola statistic apa yang dapat diturunkan.

### 5.1.1 Boomerang Attack

Salah satu varian dari Differential kripanalisis yang terkenal adalah boomerang attack. Teknik ini ditemukan oleh David A. Wagner pada tahun 1999.




Angka pada diagram di atas menunjukkan urutan langkah-langkah yang terjadi pada serangan.

1. Dipilih blok secara acak dari plainteks. Berdasarkan karakteristik dari separuh pertama cipher, jika di XOR kan vector tertentu, yang disebut d1(00100000 dalam diagram), hasilnya setelah melakukan half enciphering terhadap 2 blok plaintext sebelum dan sesudah XOR mempunyai perbedaan c1(00110110 dalam diagram).
2. Karena karakteristik tersebut hanya berlaku pada separuh pertama dari cipher, hasilnya pada keseluruhan cipher menjadi tidak berhubungan. Ambil kedua hasil tersebut kemudian XOR kan setiap hasil dengan d2(01001011 dalam diagram), yang merupakan vector yang berkorespondensi terhadap karakteristik dari paruh kedua dari cipher. Dalam setiap kasus, d2 XOR ciphertext blok diharapkan akan mengubah hasil setelah melakukan deciphering dengan c2(00010000 dalam diagram).
3. Dengan 2 result penampung yang mempunyai perbedaan c1, jika setiap result di XORkan dengan c2, hasil XOR tersebut tetap mempunyai perbedaan c1. Perbedaan ini sekarang berelasi terhadap separuh dari karakteristik.

## 5.2 XLS Attack

Prinsip yang banyak digunakan dalam membentuk cipher adalah dengan mengikuti paradigmanya Claude Shannon yang memadukan prinsip confusion dan diffusion. Contohnya adalah jaringan substitusi-permutasi yang merupakan kombinasi dari berbagai lapisan S box dengan permutasi bit. Bila struktur tersebut dipandang secara umum, setiap substitusi yang mengandung persamaan linier dapat disebut juga SA(Substitution Affine)-cipher.

XLS-cipher merupakan komposisi dari sejumlah putaran yang mirip:

- X: putaran pertama i=1 dimulai dengan XOR dengan kunci putaran Ki-1
- S : proses transformasi dengan menggunakan S-box secara paralel sebanyak B untuk setiap s bit.
- L : Proses yang menggunakan persamaan linier untuk mendukung prinsip difusi
- X : langkah selanjutnya XOR dengan kunci Ki. Jika i=jumlah putaran maka selesai, jika tidak maka majukan i dan kembali ke langkah S.

### 5.2.1 Struktur paling atas dari rijndael

Rijndael termasuk jenis XSL-cipher dengan s=8, B=4*Nb. Nr = 10..14 putaran. Data dari rijndael dimodelkan sebagai states yang berbentuk persegi yang terdiri dari Nb kolom, yang masing-masing mempunyai ukuran 4 * S-

box (4*s = 32 bits). Jumlah kolom yang mungkin adalah 4, 6, atau 8 sehingga ukuran blok yang memungkinkan adalah Nb * 32 yaitu 128bt, 192bit, dan 256 bit secara berurutan.

Proses enkripsi rjindael dalam mode XLS dapat digambarkan sebagai berikut :

- X : XOR plaintext dengan kunci putaran $K_{i-1}$
- S : B = Nb*4 Sbox pada s = 8 bits
- L : Permutasi shiftRow yang diikuti transformasi linier $GF(256)^4$ -> $GF(256)^4$ yang disebut MixColumn yang dilakukan secara parallel pada setiap Nb kolom kecuali pada putaran terakhir.
- X : pada fase selanjutnya dilakukan XOR lagi, bila putaran terakhir maka selesai, jika tidak maka kembali ke fase S.

### 5.2.2 Struktur paling atas dari serpent

Serpent dideskripsikan sebagai XLS cipher dengan s =4, B=32, Nr =32. Ukuran blok selalu 128 bit. Ukuran kunci $H_k$ = 128bits, 192 bits, atau 256 bits dan diekpansi menjadi $E_k$ = $(N_r+1)$ * s * B = 1056 bits.

### 5.2.3 S Box dan Persamaan Algebraic yang overdefined

Satu-satunya bagian yang tidak bersifat linier dari XSL cipher adalah pada S box. Dimisalkan $F:GF(2)_s$ -> $GF(2)_s$ adalah S box $F:x=(x_1..x_s)$ -> $y = (y_1..y_s)$. Pada Rijndael dan Serpent S-box dibentuk dengan fungsi Boolean yang baik. Salah satu kriteria fungsi Boolean yang baik adalah setiap $y_i$ mempunyai tingkat algebraic yang tinggi ketika diekspresikan sebagai polynomial peubah banyak dalam $x_i$. Bagaimanapun juga, semua ini tidak dapat memastikan bahwa tidak ada persamaan peubah banyak dalam bentuk $P(x_1,\ldots,x_s,y_1,\ldots,y_s)$ yang mempunyai tingkat algebraic yang rendah.

Untuk tingkat tertentu dari equation d (biasanya d=2), hal yang menarik adalah angka sebenarnya r dari persamaan $P(x_1,\ldots,x_s,y_1,\ldots,y_s)$. Untuk persamaan yang eksplisit $y_i = f(x_1,..,x_s)$, angka r dapat lebih besar dari s. Hal menarik lainnya adalah banyaknya monomial yang muncul pada persamaan yang dinotasikan sebagai t. Secara umum $t \approx c(s,d)$. Jika $t<<c(s,d)$, dikatakan bahwa persamaan tersebut jarang(*sparse*). Jika r>>s, maka sistemnya disebut ***overdefined.***

### 5.2.4 Kualitas dari S boxes acak

Ketika nilai r dekat dengan nilai t, kita dapat mengeliminasi kebanyakan terminologi tersebut dengan eliminiasi linier dan mendapatkan persamaan sederhana yang sparse dan mungkin linier. Karena itu, hal ini memungkinkan untuk mengukur kualitas sistem persamaan dengan rasio t/r >= 1. Jika t/r mendekati 1, S-box tersebut jelek. Dari sudut pandang ini, sistem yang overdefined ditandai dengan nilai r yang besar dan sistem yang sparse ditandai dengan t yang kecil merupakan sistem yang jelek. Selain daripada kedua hal tersebut, sistem tersebut dapat disebut sebagai sistem yang bagus kecuali jika s bernilai sangat kecil.

### 5.2.5 Persamaan yang overdefined pada S-box Serpent

Untuk membuktikan bahwa S box Serpent overdefined, dilakukan percobaan terhadap 4 bit S box. Untuk melakukan hal itu terdapat matriks yang berukuran 16 X 37 yang masing-masing row berisi nilai dari t = 37 monomials {1,x1,..,x4,y1,..,y4,x1x1,…,x1y1,…,y3,y4} untuk setiap dari $2^s$ = 16 kemungkinan entri x=(x1,..x4). Rank dari matriks tersebut paling tinggi adalah 16, maka dari itu apapun yang berada di dalam S box, akan ada paling sedikit r>= 37-16=21 persamaan kuadrat. Hal ini akan menghasilkan sistem yang sangat overdefined karena 21 jauh lebih besar dari 4. $t/r \approx 1.75$.

### 5.2.6 Persamaan yang overdefined pada S-box Rijndael

Untuk rijndael, s bernilai 8. Nilai itu cukup besar jika dibandingkan dengan Serpent: $(2^8)! \approx 2^{1684}$ S-box bijektif 8 bit dibandingkan dengan $(2^4)! \approx 2^{44}$ untuk s =4. Untuk alasan ini, tidak diharapkan akan muncul sifat yang berguna. Contohnya mudah dilihat bahwa dengan metode yang dideskripsikan pada Serpent di atas, S box acak 8 bits akan menghasilkan r=0 karena $2^s$ = 256 lebih besar dari 137.

S box rjindael merupakan komposisi dari inverse yang dimodifikasi dalam GF(256) dengan tidak ada bit yang dipetakan ke dirinya sendiri, dengan transformasi affine yang multivariate $GF(2)^8$ -> $GF(2)^8$ . Kedua fungsi tersebut dapat dimisalkan g dan f dan S=f o g. Dimisalkan x = nilai input dan y=g(x) yang berkoresponden dengan nilai output. z = S(x) = f(g(x)) = f(y). Menurut definisi S-box :

$$\forall x \neq 0 \qquad 1 = xy$$

Persamaan ini akan menghasilkan 8 multivariate persamaan bi-linier dalam 8 variabel dan akan mengarah kepada 8 persamaan bi-affine antara $x_i$ dan $z_j$. Tujuh dari 8 persamaan di atas bernilai true dengan probabilitas 1, dan persamaan ke 8 bernilai benar dengan probabilitas 255/256.

### 5.2.7 XSL Attack

Serangan ini pertama kali dipublikasikan oleh Nicolas Courtois dan Josep Pieprzyk dalam makalah mereka yang berjudul "*Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*". Teknik ini diklaim dapat memecahkan AES lebih cepat dari cara *exhaustive search*. XSL attack mengandalkan pada keberhasilan menganalisis subsistem internal dari cipher untuk menurunkan persamaan kuadrat secara simultan. Kumpulan persamaan ini umumnya sangat besar. Contohnya pada 128 bit AES terdapat 8000 persamaan dengan jumlah variable 1600. Metode untuk memecahkan persamaan ini disebut XSL (*eXtended Sparse Linearisation*). Jika persamaan tersebut dapat dipecahkan, maka kunci dapat diperoleh.

Pemecahan persamaan tersebut menjadi masalah jika ditemukan persamaan yang bersifat MQ(*Multivariate quadratic*). Persamaan MQ merupakan permasalahan yang bersifat NP-hard(Non Polinomial). XSL attack membutuhkan algoritma yang efisien untuk menyelesaikan MQ. Salah satu teknik untuk menyelesaikan sistem MQ adalah dengan linearisasi, yang mengubah setiap persamaan quadratic menjadi variable yang independent yang akan menghasilkan persamaan linier dengan menggunakan algoritma seperti Gaussian elimination.

Tahun 2000, Courtois mengajukan algoritma untuk MQ yang bernama XL(eXtended Linearisastion). Algoritma ini meningkatkan jumlah persamaan dengan mengalikan dengan monomial derajat tertentu. Algoritma ini akan menghasilkan suatu bentuk struktur yang disebut XSL. Algoritma XSL dibentuk dari algoritma XL dengan memilih monomial secara selektif.

## 6. Kesimpulan

Kesimpulan yang dapat diambil dari studi literatur mengenai berbagai macam serangan pada blok cipher ini adalah :

1. *Advanced Encryption Standard(AES)* masih merupakan suatu standard yang layak pakai untuk mengatasi masalah keamanan data.
2. Penemuan dari serangan algebraic membuka kesempatan baru untuk melakukan riset dan mengubah cara pandang dalam melakukan rancangan dan kriptanalisis dari suatu cipher ke level selanjutnya.
3. Cipher Rijndael dapat diekpresikan dengan rumus aljabar yang ringkas dan rapi.
4. XLS *attack* mengurangi waktu yang diperlukan untuk melakukan pemecahan block cipher daripada pemecahan blok cipher menggunakan *exhaustive search*

## 2. Daftar Referensi


[1]. Ross Anderson, Eli Biham and Lars Knudsen: Serpent: A Proposal for the Advanced Encryption Standard. Available from http://www.cl.cam.ac.uk/~rja14/ serpent.html

[2]. Nicolas Courtois and Josef Pieprzyk: Cryptanalysis of Block Ciphers with Overdefined Systems of Equations, Asiacrypt 2002, LNCS 2501, pp.267-287, Springer.

[3]. Nicolas Courtois and Josef Pieprzyk: Cryptanalysis of Block Ciphers with Overdefined Systems of Equations, Available at http://eprint.iacr.org/2002/044/.

[4]. Joan Daemen, Vincent Rijmen: AES proposal: Rijndael, The latest revised version of the proposal is available on the internet, http://csrc.nist.gov/encryption/ aes/rijndael/Rijndael.pdf

[5]. Joan Daemen, Vincent Rijmen: The Design of Rijndael. AES - The Advanced Encryption Standard, Springer-Verlag, Berlin 2002. ISBN 3-540-42580-2.

[6]. Nicolas Courtois: The security of Hidden Field Equations (HFE); Cryptographers' Track Rsa Conference 2001, LNCS 2020, Springer, pp. 266-281.

[7]. Jacques Patarin: Cryptanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt'88; Crypto'95, Springer, LNCS 963, pp. 248-261, 1995.

[8]. Adi Shamir, Jacques Patarin, Nicolas Courtois, Alexander Klimov, Efficient Algorithms

for solving Overdefined Systems of Multivariate Polynomial Equations, Eurocrypt'2000, LNCS 1807, Springer, pp. 392-407.

[9]. Claude Elwood Shannon: Communication theory of secrecy systems, Bell System Technical Journal 28 (1949), see in particular page 704.

[10]. Anne Canteaut, Marion Videau: Degree of composition of highly nonlinear functions and applications
to higher order differential cryptanalysis; Eurocrypt 2002, LNCS 2332, Springer.

[11]. Don Coppersmith, Shmuel Winograd: "Matrix multiplication via arithmetic progressions"; J.
Symbolic Computation (1990), 9, pp. 251-280.

[12]. Nicolas Courtois, Louis Goubin, Willi Meier, Jean-Daniel Tacier: Solving Underdefined Systems
of Multivariate Quadratic Equations; PKC 2002, LNCS 2274, Springer, pp. 211-227.

[13]. Nicolas Courtois: The security of Hidden Field Equations (HFE); Cryptographers' Track Rsa
Conference 2001, San Francisco 8-12 April 2001, LNCS2020, Springer-Verlag, pp. 266-281.

[14]. Horst Feistel: Cryptography and computer privacy; Scientific American, vol. 228, No. 5, pp. 15-23,
May 1973.

[15]. Niels Ferguson, Richard Schroeppel and Doug Whiting: A simple algebraic representation of
Rijndael; Draft 2001/05/16, presented at the rump session of Crypto 2000 and available at http://www.macfergus.com/niels/pubs/rdalgeq.html.

[16]. Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Mike Stay, David Wagner, Doug
Whiting: Improved Cryptanalysis of Rijndael, FSE 2000, Springer.

[17]. J.B. Kam and G.I. Davida: Structured design of substitution-permutation encryption networks;
IEEE Trans. on Computers, Vol. C-28, 1979, pp.747-753.

11. Lars R. Knudsen, Vincent Rijmen: On the Decorrelated Fast Cipher (DFC) and its Theory;
FSE'99, Springer, LNCS 1636, pp. 81-94.

[18]. Michael Luby, Charles W. Rackoff, How to construct pseudorandom permutations from pseudorandom
functions; , SIAM Journal on Computing, vol. 17, n. 2, pp. 373-386, April 1988.

[19]. T.T. Moh: On The Method of XL and Its Inefficiency Against TTM, available at http://eprint.iacr.org/2001/047/.

[20]. Moni Naor and Omer Reingold: On the construction of pseudo-random permutations: Luby-Rackoff revisited; Journal of Cryptology, vol

[21]. Kaisa Nyberg: Differentially Uniform Mappings for Cryptography; Eurocrypt'93, LNCS 765,
Springer, pp. 55-64.

[22]. Jacques Patarin: Cryptanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt'88;Crypto'95, Springer-Verlag, pp. 248-261.

[23]. Jacques Patarin: Generic Attacks on Feistel Schemes ; Asiacrypt 2001, LNCS 2248, Springer, pp.222-238.

[24]. Jacques Patarin: Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of Asymmetric Algorithms; in Eurocrypt'96, Springer Verlag, pp. 33-48.

[25]. Jacques Patarin, Nicolas Courtois, Louis Goubin: Improved Algorithms for Isomorphism of Polynomials;
Eurocrypt 1998, Springer-Verlag.

[26]. Adi Shamir, Alex Biryukov: Structural Cryptanalysis of SASAS; Eurocrypt 2001, LNCS 2045,
Springer, pp. 394-405.

[27]. Adi Shamir, Aviad Kipnis: Cryptanalysis of the HFE Public Key Cryptosystem; In Advances in Cryptology, Proceedings of Crypto'99, Springer-Verlag, LNCS.

[28]. Adi Shamir, Jacques Patarin, Nicolas Courtois, Alexander Klimov, Efficient Algorithms for solving
Overdefined Systems of Multivariate Polynomial Equations, Eurocrypt'2000, LNCS 1807, Springer, pp. 392-407.

[29]. Robert D. Silverman: A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths; RSA Lab. report, http://www.rsasecurity.com/rsalabs/bulletins/bulletin13.html.

[30]. Claude Elwood Shannon: Communication theory of secrecy systems; , Bell System Technical Journal 28 (1949), see in patricular page 704.

[31]. Serge Vaudenay: Provable Security for Block Ciphers by Decorrelation; Technical Report LIENS-98-8 of the Laboratoire d'Informatique de l'Ecole Normale Sup´erieure, 1998. Available at http://lasecwww.epfl.ch/query.msql?ref=Vau98b.

[32]. Serge Vaudenay, Shiho Moriai: On the Pseudorandomness of Top-Level Schemes of Block Ciphers; Asiacrypt 2000, LNCS 1976, Springer, pp. 289-302.

[33] Munir, Rinaldi, Diktat Kuliah Kriptografi, Insitut Teknologi Bandung, 2006.