CIPHER BLOCK MENGGUNAKAN LOKI

Raden Fitri Indriani – NIM : 13503020

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13020@students.if.itb.ac.id

Abstrak

Di dalam kriptografi ada berbagai macam algoritma yang dapat digunakan, baik yang termasuk algoritma kriptografi modern maupun algoritma kriptografi klasik. Yang termasuk di dalam algoritma kriptografi modern adalah algoritma *cipher block*. Chiper block merupakan sebuah algoritma simetri yang mengoperasikan bit-bit yang mempunyai panjang yang sama. Sebagai contoh, ketika melakukan enkripsi, sebuah 128-bit blok *plaintext* akan menghasilkan sebuah 128-bit blok *ciphertext.* Begitu pula dengan proses deskripsi. Sebuah 128-bit blok *ciphertext* akan menghasilkan 128-bit blok *plaintext.*

Sudah banyak sekali algoritma *cipher block* yang dikembangkan. Salah satunya adalah LOKI. Nama LOKI diambil dari salah satu nama dewa dalam mitos Skandinavia. LOKI pertama kali dipublikasikan oleh kriptografer Australia yaitu Lawrie Brown, Josef Pieprzyk, dan Jennifer Seberry pada tahun 1990. LOKI dibuat dengan tujuan untuk menggantikan DES (*Data Encryption Standard*). LOKI dibuat sangat mirip dengan struktur DES. Sejak pertama kali dibuat, LOKI terus diperbaiki yang menghasilkan beberapa versi LOKI seperti LOKI91 dan LOKI97.

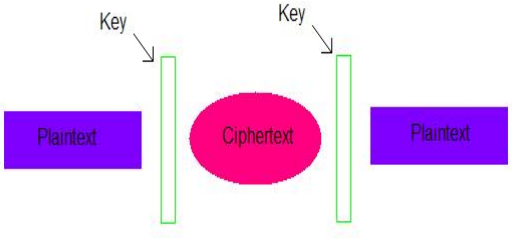
***Kata kunci**:* kriptografi modern,blok *cipher*, LOKI89, LOKI91, LOKI97

1. Pendahuluan

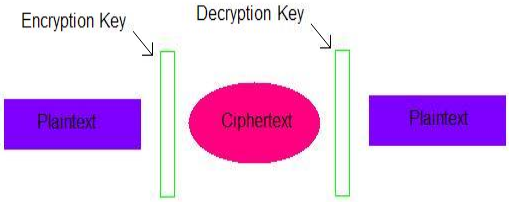
Kriptografi adalah sebuah ilmu dan seni yang memungkinkan pesan dikirim dalam bentuk yang aman sehingga hanya penerima yang berhaklah yang dapat membaca informasi di dalamya. Kriptografi sudah ada sejak jaman bangsa Mesir sekitar 4000 tahun yang lalu. Semakin berkembangnya jaman maka semakin berkembang pula algoritma kriptografi yang digunakan. Apalagi dengan jaman yang serba komputer sekarang ini, algoritma yang digunakan untuk mengamankan pesan pun semakin kompleks sehingga sulit dipecahkan oleh para kriptanalis. Berbeda dengan kriptografi klasik, kriptografi modern menggunakan algoritma yang canggih dan rumit serta kunci rahasia untuk mengekripsi dan mendekripsi data.

Kriptografi modern mempunyai basis pada kerahasiaan kunci yang digunakan, bukan pada kerahasiaan metode enkripsinya. Kunci rahasia haruslah tidak tampak ketika plainteks dan cipherteks dibandingkan. Kriptografi modern didasarkan pada persoalan matematika yang rumit seperti pemfaktoran bilangan prima, dan sebagainya. Desain sistem pengamanan difokuskan pada keamanan kunci yang digunakan. Contohnya adalah dengan menyembunyikan kunci yang digunakan dengan mengenkripsinya dengan kunci yang lain.

Ada dua macam enkripsi yang berdasarkan pada kunci yaitu algoritma simetri dan algoritma asimetri. Algoritma simetri menggunakan kunci yang sama untuk dekripsi dan enkripsi (kunci dekripsi dapat diturunkan dari kunci enkripsi). Sedangkan algoritma asimetri menggunakan kunci yang berbeda antara enkripsi dan dekripsi (kunci dekripsi tidak dapat diturunkan dari kunci dekripsi). Algoritma asimetri disebut juga algoritma kunci publik. Secara umum, algoritma simetri melakukan eksekusi lebih cepat dibandingkan dengan algoritma asimetri [ECO06].



Gambar 1 Algoritma Simetri



Gambar 2 Algoritma Asimetri

Algoritma simetri terbagi menjadi dua jenis yaitu *stream cipher algorithm* dan *block chiper algorithm. Stream cipher algorithm* dapat melakukan enkripsi sebuah bit single dari plainteks setiap waktunya. Sedangkan *block cipher algorithm* melakukan enkripsi terhadap beberapa bit dari plainteks (biasanya 64 bit) dan mengenkripsinya sebagai sebuah kesatuan.

2. Algoritma *Cipher* Blok

Algoritma *cipher* blok termasuk ke dalam algoritma simetri yang mengoperasikan bit-bit yang panjangnya tetap dengan sebuah transformasi yang sama. Biasanya plainteks dibagi menjadi blok-blok bit dengan panjang yang sama. Ketika melakukan enkripsi, misalnya pada sebuah blok plainteks yang panjangnya 64 bit maka akan menghasilkan blok cipherteks sepanjang 64 bit juga. Begitu pula ketika dilakukan dekripsi, sebuah blok cipherteks sepanjang 128 bit akan menghasilkan sebuah blok plainteks sepanjang 128 bit. Kunci yang digunakan untuk melakukan enkripsi dan dekripsi mempunyai panjang yang sama dengan blok bit plainteks

Cipher blok terdiri dari dua buah algoritma yang berpasangan. Satu digunakan untuk melakukan enkripsi, E. Satu lagi digunakan untuk melakukan dekripsi, E^{-1}. Kedua algoritma tersebut menerima dua buah input, yaitu sebuah blok input berukuran n bit dan sebuah kunci berukuran k bit, menghasilkan sebuah blok yang berukuran n bit.

Untuk kunci apapun yang digunakan, dekripsi merupakan fungsi invers dari enkripsi. Dengan demikan, algoritma enkripsi dan dekripsi dapat dirumuskan sebagai berikut.

$$E_k^{-1}(E_k(P)) = P$$

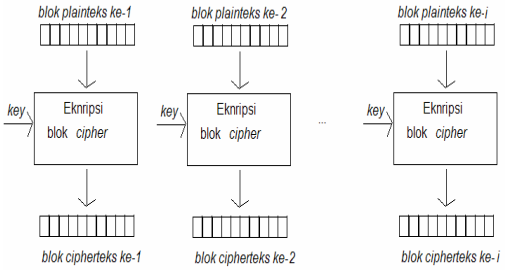
P = sebuah blok plainteks
k = key

Ukuran blok yang digunakan umumnya 64 bit atau 128 bit. Akan tetapi ada beberapa algoritma *cipher* blok yang menggunakan ukuran blok yang berbeda (bukan 64 bit maupun 128 bit). Ukuran blok 64 bit biasanya digunakan untuk desain yang dirancang sampai pertengahan tahun 1990. Desain-desain baru umumnya menggunakan blok berukuran 128 bit. Jika plainteks akan dibagi menjadi blok-blok berukuran 64 bit misalnya, tetapi ukuran plainteks tersebut bukan kelipatan dari 64, maka digunakan skema *padding* agar plainteks tersebut tetap bisa dienkripsi.

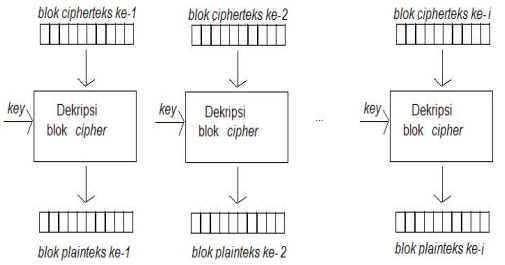
Ada beberapa mode operasi yang digunakan dalam blok *cipher*. Setiap mode operasi memiliki karakteristik tersendiri untuk menghindari serangan-serangan yang mungkin muncul. Ukuran kunci (*key*) yang umumnya digunakan adalah 40, 56, 64, 80, 128, 192 dan 256 bit. Beberapa mode operasi yang digunakan untuk blok *cipher* adalah sebagai berikut.

- *Electronic Code Book (ECB)*

  Mode operasi ECB merupakan mode enkripsi yang paling sederhana dimana plainteks dibagi menjadi beberapa blok yang berukuran sama dan setiap blok dienkripsi secara terpisah.



Gambar 3 Enkripsi EBC



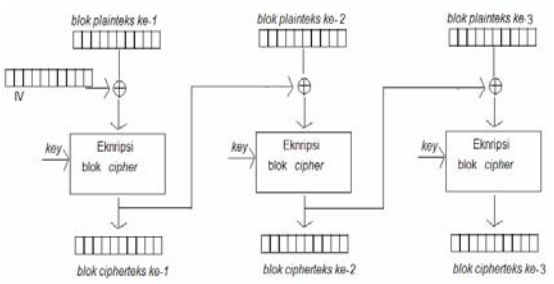
Gambar 4 Dekripsi EBC

Kelemahan mode ini adalah sebuah *plainteks* akan dienkripsikan menjadi sebuah *cipherteks* yang identik. Dengan demikian, mode EBC ini tidak dapat menyembunyikan pola dari data. Hal ini mengakibatkan mode EBC tidak mendukung *message confidentiality*. Oleh karena itulah mode ini tidak disarankan untuk digunakan sebagai protokol kriptografi.

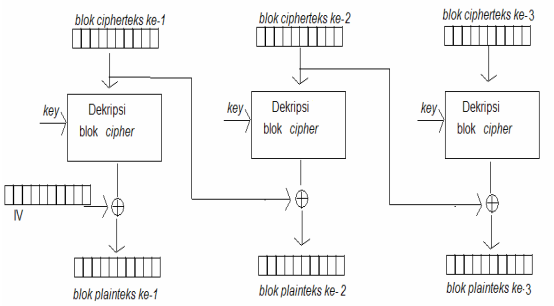
Berikut ini adalah sebuah contoh yang menunjukkan bahwa enkripsi dengan menggunakan mode EBC tidak dapat menyembunyikan pola data.

- *Cipher Block Chaining (CBC)*

Cipher block chaining hampir serupa dengan EBC dalam hal pembagian plainteks menjadi blok-blok bit yang berukuran sama. Hanya saja, dalam *cipher block chaining* setiap blok bergantung dengan blok yang berdekatan dengannya. Hal ini disebabkan karena setiap blok plainteks dilakukan operasi xor dengan blok cipherteks sebelumnya. Untuk blok pertama, digunakan *Initialitation Vector* (IV) sebagai pengganti cipherteks sebelumnya. Dengan adanya intialitation vector, maka pesan menjadi unik. Akan tetapi, CBC jarang digunakan. Mode CBC mempunyai kelemahan yaitu jika terjadi kesalahan satu bit dalam sebuah blok plainteks pada proses enkripsi, maka blok-blok cipherteks berikutnya dan blok cipherteks dari plainteks yang bersangkutan akan mengalami kesalahan pula. Akan tetapi jika kesalahan terjadi pada sebuah bit dalam sebuah cipherteks pada proses dekripsi, maka hanya blok planteks yang bersangkutan dan sebuah blok plainteks berikutnya saja yang terkena dampaknya.



Gambar 5 Enkripsi menggunakan mode CBC



Gambar 6 Dekripsi menggunakan mode CBC

- *Propagating Cipher Block Chaining (PCBC)*

Propagating cipher block chaining didesain untuk melengkapi kekurangan yang terdapat pada *cipher block chaining* (CBC). Proses enkripsi dan dekripsi dilakukan dengan cara sebagai berikut.

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

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

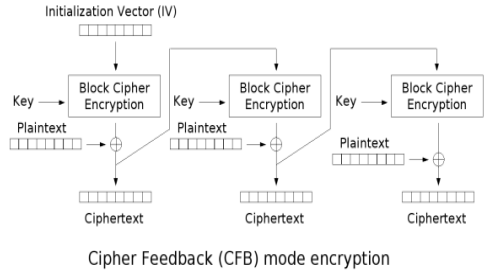
Initialitation vector yang digunakan adalah $m_o \oplus c_o$. PCPB biasanya digunakan pada Kerberos dan WASTE. Secara umum PCBC ini jarang digunakan.

- *Cipher FeedBack (CFB)*

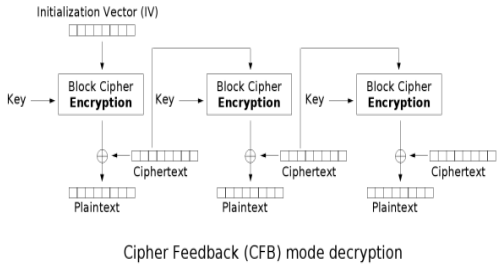
Mode CFB memperlakukan blok *cipher* seolah-olah seperti *cipher* aliran. Hal ini dilakukan dengan cara data dienkripsikan menjadi ukuran yang lebih kecil dari blok seperti 2 bit, 8 bit, dan seterusnya. Dengan demikian, mode CFB bekerja bukan pada blok lagi tetapi pada unit-unit yang terdiri dari beberapa buah bit.

Mode CFB membutuhkan sebuah queue dimana elemen dari queue

inilah yang akan dikenai fungsi enkripsi/dekripsi.



Gambar 7 Enkripsi menggunakan mode CFB

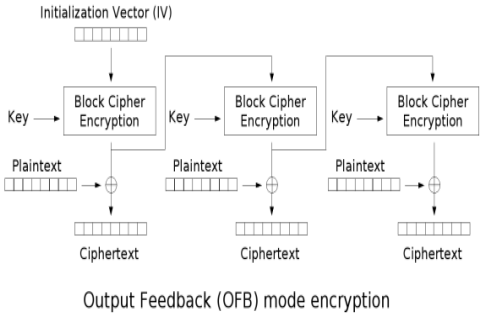


Gambar 8 Dekripsi menggunakan mode CFB

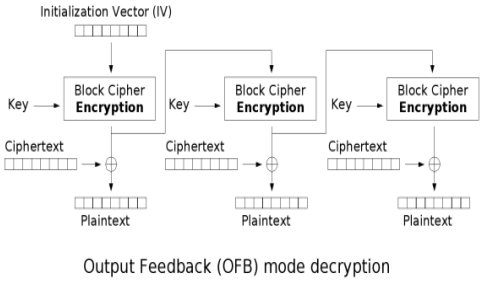
Kelemahan dari mode ini adalah jika terdapat kesalahan pada satu buah bit yang dienkripsi, maka bit-bit selanjutnya juga akan salah. Hal ini juga berlaku untuk proses dekripsi.

- *Output FeedBack(OFB)*

  Seperti halnya CFB, OFB juga memperlakukan blok *cipher* seperti layaknya *cipher* aliran. Pada prinsipnya mode OFB mirip dengan CFB. Perbedaannya adalah jika pada CFB yang dimasukkan ke dalam antrian (queue) adalah hasil enkripsi (cipherteks). Sedangkan pada OFB yang dimasukkan ke dalam antrian adalah elemen antrian terdepan yang telah dikenakan fungsi enkripsi.



Gambar 9 Enkripsi menggunakan mode OFB



Gambar 10 Dekripsi menggunakan mode OFB

Jika terjadi kesalahan pada 1 bit blok plainteks maka hanya akan mempengaruhi blok cipherteks saja. Hal ini juga berlaku untuk proses dekripsi.

3. LOKI89

LOKI dirancang oleh kriptografer Australia yaitu Lawrie Brown, Josef Pieprzyk, dan Jennifer Seberry. LOKI didesain sebagai hasil dari analisis yang dilakukan secara detail terhadap blok *cipher* yang standar digunakan pada saat itu, yaitu DES (*Data Encryption Standar*). Dikarenakan ada versi terbaru dari LOKI ini, maka LOKI yang dibuat pertama kali lebih dikenal dengan nama LOKI89 sesuai dengan tahun pembuatannya, walaupun LOKI pertama kali diperkenalkan pada tahun 1990.

LOKI didesain untuk menggantikan DES sehingga strukturnya pun dibuat hampir sama dengan DES. LOKI menggunakan blok data sepanjang 64 bit dan kunci sepanjang 64 bit pula.

❖ Desain LOKI89

Secara garis besar, desain dari LOKI89 sangat mirip dengan desain DES, termasuk :
– Permutasi P dan penggunaannya dalam membentuk cipherteks tergantung daripada plainteks
– Permutasi PC2 dan perputaran kunci dalam membentuk cipherteks tergantung dari kunci.

Desain S-Boxes dari LOKI89 berdasarkan pada kriteria nonlinier yang dibuat oleh Josef Pieprzyk, dan Jennifer Seberry. LOKI89 merupakan sebuah kelas dari *Feistel cipher*, maksudnya adalah sebuah bentuk dari subtitusi dan permutasi (S-P) dan jaringan (Shannon).

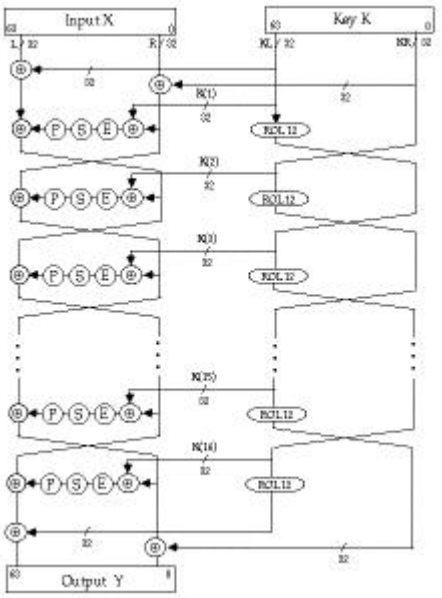
Properti inti dari jaringan S-P adalah sebagai berikut.
– Properti *avalanche*
– Properti *completeness*
Kedua buah properti di atas diimplementasikan dalam desain yang terdiri dari dua buah fase.

➢ Ekspansi Fungsi E
Ekspansi fungsi E dilakukan dengan cara menduplikasi input dan kunci untuk mendukung *autoclaving*. Ekspansi fungsi E berdasarkan pada pemilihan bit-bit yang dilakukan secara regular. Ekspansi fungsi E didesain menggunakan input 32 bit untuk memudahkan implementasi pada perangkat lunak.

➢ Permutasi P
Permutasi dilakukan dengan mengirimkan output dari sebuah S-Box untuk menjadi input pada S-Box lain yang akan diproses. Permutasi ini dapat dibangkitkan dengan fungsi yang berbeda-beda yang terdapat pada nomor S-Box.

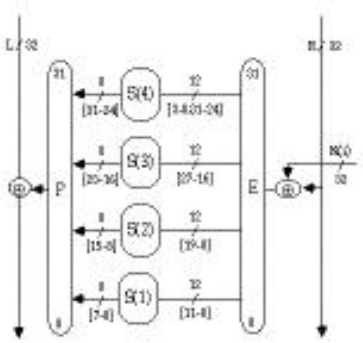
➢ Jadwal penggunaan kunci
Proses ini dilakukan dengan merotasikan kunci berukuran besar. Putaran dilakukan sebanyak 16 kali berdasarkan hasil dari Bilham-Shamir karena jika kurang dari itu dianggap tidak aman.

❖ Struktur LOKI89



Gambar 7 Struktur LOKI89

❖ Fungsi *f* LOKI89



Gambar 8 Fungsi f LOKI89

❖ Kunci LOKI89

Kunci yang digunakan oleh LOKI89 adalah blok yang mempunyai panjang 64 bit yang tidak menggunakan bit paritas. Dengan panjang 64 bit, maka jika kripanalisis menggunakan *brute force* harus mencoba kemungkinan sebanyak 2^{64} kali.

❖ Desain S-Box LOKI89

Desain S-Box yang digunakan oleh LOKI pada dasarnya sama dengan S-Box biasa dengan menggunakan properti *completeness* dan *avalanche*. Prinsip utama dari desain S-Box ini adalah fungsi boolean dengan nonlinier yang maksimum yang dicanangkan oleh Josef Pieprzyk, dan

Jennifer Seberry. Nonlinear dari sebuah fungsi *f* pada sebuah *set* F_n dari seluruh fungsi boolean dengan n variable adalah minimum jarak Hamming (Hamming *distance*) antara *f* dan kumpulan semua fungsi boolean linier L_n yang terdapat di F_n.

S-Box LOKI89 didesain untuk menerima masukan sebanyak 12 dan mengeluarkan output sebanyak 8. Hal ini dilakukan dengan alasan untuk memaksimalkan ketergantungan antar bit dan karena kedua angka tersebut merupakan nilai terbesar yang dapat disimpan di tabel biasa (tidak terkomputerisasi). S-Box ini dipecah menjadi fungsi yang terdiri dari 16 baris dan 256 kolom. Dimana setiap fungsinya merupakan eksponensial GF (2^8) sebagai berikut :

$$Sfn_r = (c+r)^{er} \bmod gen_r$$

Rumus tersebut mempunyai 30 polinomial yang tidak dapat diperkecil lagi, 24 perpangkatan yang membentuk fungsi nonlinier maksimum. Pangkat 31 dipilih karena dianggap merupakan nilai terkecil yang menghasilkan nilai nonlinier yang maksimal.

❖ Implementasi LOKI89

Berikut ini adalah salah satu implementasi LOKI89 64 bit dalam bahasa C yang dites pada mesin *little indian*. Kode ini dibuat oleh Lawrence Brown.

Fungsi f dideklarasikan dengan :

```
static Long f();
```

S-box dideklarasikan dengan :

```
static short s();
```

Kunci 64 bit dideklarasikan dengan cara :

```
Long  lokikey[2]= {0, 0};
```

Fungsi perputaran didefinisikan dengan cara sebagai berikut.

```
ROL12(b) b = ((b << 12) |
(b >> 20));

ROR12(b) b = ((b >> 12) |
(b << 20));
```

ROL12(b) digunakan pada saat perputaran untuk proses enkripsi yaitu untuk menggeser 32 bit blok b ke kiri sepanjang 12 bit. Sedangkan ROR12(b) digunakan pada saat perputaran untuk proses dekripsi yaitu untuk menggeser 32 bit blok b ke kanan sepanjang 12 bit.

Proses Enkripsi

Proses enkripsi berikut ini akan melakukan enkripsi sebuah 64 bit blok b dengan menggunakan kunci lokikey.

Proses ini akan melakukan operasi XOR blok input dengan kunci, implementasi perputaran LOKI sebanyak 16 kali yang bertujuan untuk menjamin bahwa blok keluaran merupakan hasil fungsi yang rumit antara blok masukan dan kunci, dan yang terakhir adalah melakukan XOR kembali blok masukan dengan kunci.

Setiap putaran akan melakukan perhitungan dengan menggunakan subkunci sepanjang 48 bit dengan rincian sebagai berikut :

L(i) = R(i-1)
R(i) = L(i-1) XOR f(R(i-1),K(i))

Untuk diingat bahwa 64 bit blok b masukan dianggap sebagai dua buah longword dimana bit-bit tersebut diberi nomor 0..63. ([63 62 ... 33 32] [31 30 ... 1 0]). Dengan demikian L adalah b[0] dan R adalah b[1].

Dikarenakan blok masukan dibagi menjadi dua bagian (L dan R), maka kunci juga harus dibagi menjadi dua bagian yang sama, yaitu KL dan KR dengan prinsip yang sama dalam pembagian blok masukan. Kemudian setelah kunci dibagi menjadi dua bagian, sekarang dilakukan operasi XOR antara kunci dengan blok masukan dengan cara sebagai berikut.

```
L = ((Long *)b)[0] ^ KL;
R = ((Long *)b)[1] ^ KR;
```

Kemudian dilakukan putaran sebanyak 16 kali menggunakan subkunci dengan cara sebagai berikut.

```
L ^= f(R, KL);
ROL12(KL);
R ^= f(L, KR);
ROL12(KR);
```

```
L ^= f(R, KL);
ROL12(KL);
R ^= f(L, KR);
ROL12(KR);
L ^= f(R, KL);
ROL12(KL);
R ^= f(L, KR);
ROL12(KR);
L ^= f(R, KL);
ROL12(KL);
R ^= f(L, KR);
ROL12(KR);
L ^= f(R, KL);
ROL12(KL);
R ^= f(L, KR);
ROL12(KR);
L ^= f(R, KL);
ROL12(KL);
R ^= f(L, KR);
ROL12(KR);
L ^= f(R, KL);
ROL12(KL);
R ^= f(L, KR);
ROL12(KR);
L ^= f(R, KL);
ROL12(KL);
R ^= f(L, KR);
ROL12(KR);
```

Operasi terakhir dari proses enkripsi ini adalah dengan melakukan XOR antara blok masukan yang telah diputar dengan kunci. Operasinya adalah sebagai berikut.

```
((Long *)b)[0] = R ^ KR;
((Long *)b)[1] = L ^ KL;
```

Proses Dekripsi

Seperti pada proses enkripsi, bagi kunci menjadi dua bagian yaitu KL dan KR. Begitu pula dengan blok yang akan dienkripsi dibagi menjadi dua bagian yaitu L dan R.

Operasi yang dilakukan pada proses dekripsi adalah melakukan XOR antara blok masukan dengan kunci.

```
L = ((Long *)b)[0] ^ KR;
R = ((Long *)b)[1] ^ KL;
```

Setelah itu dilakukan putaran sebanyak 16 kali dengan cara sebagai berikut.

```
ROR12(KR);
L ^= f(R, KR);
ROR12(KL);
R ^= f(L, KL);
ROR12(KR);
L ^= f(R, KR);
ROR12(KL);
R ^= f(L, KL);
ROR12(KR);
L ^= f(R, KR);
ROR12(KL);
R ^= f(L, KL);
ROR12(KR);
L ^= f(R, KR);
ROR12(KL);
R ^= f(L, KL);
ROR12(KR);
L ^= f(R, KR);
ROR12(KL);
R ^= f(L, KL);
ROR12(KR);
L ^= f(R, KR);
ROR12(KL);
R ^= f(L, KL);
ROR12(KR);
L ^= f(R, KR);
ROR12(KL);
R ^= f(L, KL);
```

Setelah putaran selesai dilakukan, tahap selanjutnya adalah dengan melakukan XOR kembali antara kunci dan blok masukan yang telah diproses sebelumnya.

```
((Long *)b)[0] = R ^ KL;
((Long *)b)[1] = L ^ KR;
```

Fungsi f(r,k)

f(r,k) merupakan sebuah fungsi LOKI89 yang kompleks dan nonlinier yang menghasilkan keluaran yang merupakan hasil dari fungsi kompleks dari masukan dan kuncinya. Caranya adalah data R(i-1) yang menjadi masukan akan melalui proses :

a. Penambahan modulo 2 pada K(i)
b. Perpanjangan menjadi 48 bit melalui fungsi ekspansi fn E
c. Dilakukan subtitusi dengan menggunakan S-Box
d. Dilakukan permutasi oleh P.

Data R(i-1) dideklarasikan dengan cara :

```
register Long r;
```

Kunci K(i) dideklarasikan dengan cara:

```
Long k;
```

12 bit untuk perluasan E dideklarasikan dengan cara:

```
#define MASK12 0x0fff;
```

Proses pertama yang dilakukan pada fungsi f ini adalah dengan melakukan operasi XOR antara data masukan R(i-1) dan kunci K(i).

```
a = r ^ k;
```

Setelah itu dilakukan perpanjangan dan subtitusi menggunakan S-box dengan cara :

```
b = ((Long)s((a&MASK12)))
    |
    ((Long)s(((a>>8)&
    MASK12)) <<  8)
    |
    ((Long)s(((a>>16)&
    MASK12)) << 16)
    |
    ((Long)s((((a>>24)
    |
    (a  <<  8))  &  MASK12))
    << 24);
```

Sejauh ini, proses yang dilakukan adalah :

```
B = S(E(R(i-1))^K(i))
```

Setelah ekspansi dan subtitusi selesai dilakukan, maka selanjutnya adalah dengan melakukan permutasi.

```
perm32(&c, &b, P);
```

Dengan demikian, keseluruhan proses yang dilakukan adalah sebagai berikut.

```
C=P(S(E(R(i-1))XOR K(i)))
```

Maka, nilai yang dikembalikan oleh fungsi f ini adalah C.

S-Box : s(i)

Sebagai catatan, nilai masukan 12 bit (i) untuk S-box direpresentasikan sebagai [11 10 9 ... 1 0] dimana :
Bit [11 10 1 0] akan memilih sebuah baris dari 16 buah baris yang terdapat di setiap S-box. Sedangkan bit [9 8 … 3 2] akan menentukan kolom pada baris yang telah dipilih oleh bit [11 10 1 0].

Proses pertama yang dilakukan adalah menentukan nilai baris.

```
r = ((i>>8) & 0xc) | (i &
0x3);
```

Setelah nilai baris ditentukan, maka selanjutnya adalah menentukan nilai kolom.

```
c = (i>>2) & 0xff;
```

Setelah itu menentukan nilai basis sfn.

```
t = c ^ r;
```

Proses yang terakhir adalah melakukan eksponensial.

```
v = exp8(t, sfn[r].exp,
sfn[r].gen);
/* Sfn[r] = t ^ exp mod
gen */
```

Dengan demikian nilai yang dikembalikan oleh proses S-box ini adalah v.

Permutasi : perm32(out,in,perm)

perm32 adalah sebuah operasi permutasi biasa yang mendapat masukan 32 bit blok in, menghasilkan 32 bit blok out di bawah kontrol lanjar permutasi perm. Setiap elemen dari perm menspesifikasikan bit masukan mana yang harus dipermutasikan menjadi bit keluaran dengan indeks yang sama dengan elemen perm.

Fungsi permutasi ini dilakukan dengan cara :

```
for (o=0; o<32; o++)
 {
  i =(int)*p++;
  b = (*in >> i) & 01;
  if (b)
      *out |= mask;
  mask >>= 1;
 }
```

/* Untuk setiap bit keluaran di posisi o, ambil nilai masukan yang akan dipermutasikan ke bit keluaran o. Hitung nilai dari untuk bit masukan i. Jika i telah diset, lakukan OR pada mask untuk bit keluaran i. Geser mask ke bit selanjutnya.
*/

Dengan :
mask = 0x80000000L
*p = perm
o = indeks bit keluaran
i = indeks bit masukan

Eksponensial : exp8(base, eksponen, gen)

Fungsi ini menerima tiga buah masukan yaitu base (basis dari

eksponensial), eksponen (nilai eksponen), dan gen (polinomial terkecil yang dapat digunakan untuk membangkitkan Galois Field). Fungsi ini mengembalikan hasil perhitungan dari :

```
exp = base ^ exp mod gen
```

Proses dari fungsi ini adalah sebagai berikut.

```
if (base == 0)
  return(0);

while (exponent != 0)
{
if((exponent&0x0001)== 0x0001)
   result = mult8(result,accum,
   gen);
exponent >>= 1;
accum = mult8(accum, accum,
gen);
}
return(result);
```

/*
jika nilai base adalah 0 maka fungsi akan otomatis mengembalikan nilai 0. jika tidak 0 maka dilakukan proses perulangan hingga exponent menjadi 0. Eksponen akan dikenai fungsi mult8 jika nilai eksponen adalah 1. Lalu nilai eksponen bergeser ke dijit selanjutnya.
*/

Multiply : mult8(a, b, gen)

Fungsi ini mengembalikan hasil perkalian dari dua buah string biner a dan b serta menggunakan generator gen sebagai modulus.

```
mult = a * b mod gen
```

Proses dari fungsi ini adalah sebagai berikut.

```
while(b != 0)
{
if (b & 01)
   product ^= a;
a <<= 1;
if (a >= SIZE)
   a ^= gen;
b >>= 1;
}
return(product);
```

/*
selama pengali b tidak 0, lakukan proses berikut ini : jika LSB dari b merupakan 01, tambahkan hasil perkalian dengan a. Geser a sebanyak satu dijit. Jika a lebih besar dari 256, maka a dikurangi modulo. Geser a sebanyak satu dijit.
*/

❖ Kelemahan LOKI89

Kelemahan-kelemahan LOKI89 yang berhasil ditemukan para ahli adalah sebagai berikut.

1. LOKI89 yang menggunakan putaran hingga 10 kali memungkinkan terjadinya serangan dengan menggunakan *differential cryptanalisis*. Hal ini ditemukan oleh Biham dan Knudsen.
2. LOKI89 yang menggunakan putaran hingga 14 kali memungkinkan adanya celah serangan menggunakan 3 *round characteristic*. Hal ini ditemukan oleh Kwan dan Knudsen.

4. LOKI91

LOKI91 dibuat sebagai revisi dari LOKI89. Oleh karena itulah terdapat beberapa kesamaan antara LOKI91 dan LOKI89.

❖ Desain LOKI91

Tambahan-tambahan yang terdapat pada LOKI91 dari LOKI89 adalah sebagai berikut.

a. Jadwal pembangkitan kunci diperbaiki sehingga meminimalkan pembangkitan kunci yang sama ataupun pasangan kunci dan plainteks yang berhubungan.
b. Meminimalkan probabilitas f(x) menghasilkan 0
c. Menjamin bahwa blok telah diputar dengan jumlah yang cukup sehingga serangan yang mungkin ada hanya *exhaustive search*.
d. Menjamin bahwa S-box tidak akan menghasilkan 0 sehingga meningkatkan keamanan untuk mode *hashing*.

❖ Spesifikasi LOKI91

Perubahan-perubahan yang terdapat pada LOKI91 adalah sebagai berikut.

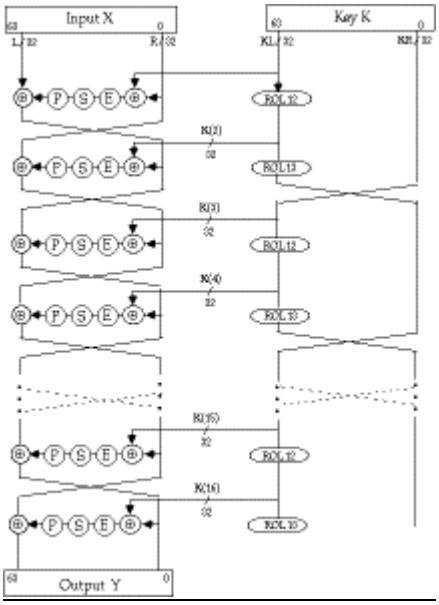
a. Merubah jadwal pembangkitan kunci sedemikian sehingga

kunci kanan KR dan kunci kiri KL ditukar setiap dua putaran.

b. Merubah jadwal rotasi kunci dengan menggunakan dua buah cara yaitu antara ROT12 (putaran ganjil) dan ROT13 (putaran genap).
c. Menghilangkan operasi XOR di awal dan akhir antara kunci dan blok yang akan dienkripsi/didekripsi.
d. Merubah fungsi S-box menjadi sebagai berikut.

$$Sfn\ (r,c) = (c + ((r*17)\ (+)\ ff_{16})\ \&\ ff_{16})\ 31\ \ mod\ gen_r$$

❖ Struktur LOKI91



Gambar 10 Struktur LOKI91

❖ Implementasi LOKI91

Berikut ini adalah salah satu implementasi LOKI91 64 bit dalam bahasa C yang dites pada mesin *little indian*. Kode ini dibuat oleh Matthew Kwan beserta Lawrence Brown.

Kunci masukan disimpan di dalam *array of long*. Sedangkan subkunci untuk 16 putaran dideklarasikan dengan cara :

```
Long loki_subkeys[ROUNDS];
```

Fungsi f dideklarasikan dengan cara :

```
static Long f();
```

S-box dideklarasikan dengan cara :

```
static short s();
```

Fungsi perputaran didefinisikan sebagai berikut.

```
ROL12(b) b = ((b << 12) |
(b >> 20));
ROL13(b) b = ((b << 13) |
(b >> 19));
```

ROL12(b) digunakan untuk menggeser 32 bit blok b ke kiri sepanjang 12 bit. Sedangkan ROL13(b) digunakan untuk menggeser 32 bit blok b ke kiri sepanjang 13 bit.

Sebelum melakukan proses enkripsi/dekripsi, subkunci sebanyak 16 buah harus dibangkitkan terlebih dahulu. Sebelum pembangkitan kunci dilakukan, kunci masukan harus dibagi dua bagian yaitu KL dan KR seperti pada pembagian kunci di LOKI89. Pembangkitan subkunci dilakukan dengan cara :

```
for (i=0;i<ROUNDS;i+=4)
 {
    loki_subkeys[i]=KL;
    ROL12 (KL);
    loki_subkeys[i+1]=KL;
    ROL13 (KL);
    loki_subkeys[i+2]=KR;
    ROL12 (KR);
    loki_subkeys[i+3]=KR;
    ROL13 (KR);
 }
```

Dengan ROUNDS=16.

Proses Enkripsi

Seperti pada halnya LOKI89, blok masukan dibagian menjadi dua buah bagian yang sama besar, yaitu R dan L.

```
L = ((Long *)b)[0];
R = ((Long *)b)[1];
```

Kemudian dilakukan proses enkripsi dengan menggunakan 16 buah subkunci yang telah dibangkitkan sebelumnya. Proses enkripsi dengan menggunakan subkunci ini dilakukan dengan cara sebagai berikut.

```
for (i=0; i<ROUNDS; i+=2)
{
L^=f(R,loki_subkeys[i]);
R^=f(L,loki_subkeys[i+1]);
}
```

Operasi yang terakhir dilakukan dalam proses enkripsi adalah dengan menukar L dan R :

```
((Long *)b)[0] = R;
((Long *)b)[1] = L;
```

Proses Dekripsi

Pada dasarnya proses dekripsi mirip dengan proses enkripsi, hanya saja putaran dilakukan dengan urutan yang terbalik dari proses enkripsi.

Pertama-tama, blok yang akan didekripsi dibagi menjadi dua buah bagian yang sama, yaitu L dan R.

```
L = ((Long *)b)[0];
R = ((Long *)b)[1];
```

Kemudian dilakukan putaran dengan cara sebagai berikut.

```
for (i=ROUNDS; i>0; i-=2)
{
L^=f(R, loki_subkeys[i-1]);
R^=f(L, loki_subkeys[i-2]);
}
```

Proses terakhir yang dilakukan pada dekripsi adalah dengan melakukan penukaran antara L dengan R.

```
((Long *)b)[0] = R;
((Long *)b)[1] = L;
```

Fungsi f(r,k)

f(r,k) merupakan sebuah fungsi LOKI91yang kompleks dan nonlinier yang menghasilkan keluaran yang merupakan hasil dari fungsi kompleks dari masukan dan subkunci. Hal ini dilakukan dengan cara sebagai berikut.
Operasi XOR dilakukan terhadap data masukan (r) dengan subkunci(k). Setelah itu data diperpanjang menjadi 4x12 bit yang akan dimasukkan ke dalam S-box. Keluaran 4x8 bit dari S-box akan dipermutasikan untuk menghasilkan 32 bit keluaran dari fungsi f ini.

Pertama-tama data masukan r dikenai fungsi XOR dengan subkunci masukan dan hasilnya disimpan dalam a.

```
a = r ^ k;
```

Sejauh ini operasi yang dilakukan adalah :

```
A = R(i-1) XOR K(i)
```

Setelah itu dilakukan ekspansi dan hasilnya dimasukkan ke dalam S-box.

```
b = ((Long)s((a&MASK12)))
    |
     ((Long)s(((a>>8)&
     MASK12))<<8)
    |
     ((Long)s(((a>>16)&
     MASK12))<<16)
    |
     ((Long)s((((a>>24)
    |
     (a<<8))&MASK12))<< 24);
```

Dimana MASK12 adalah 0x0fff. Sejauh ini operasi yang telah dilakukan adalah.

```
B = S(E(R(i-1))^K(i))
```

Operasi yang terakhir dilakukan pada fungsi f ini adalah melakukan permutasi dengan cara :

```
perm32(&c, &b, P);
```

Dengan demikian, keseluruhan proses fungsi f ini adalah :

```
C=P(S(E(R(i-1)) XOR K(i)))
```

Dengan C adalah nilai yang dihasilkan oleh fungsi f ini.

S-box : s(i)

Proses yang dilakukan dalam S-box ini adalah sebagai berikut.

Pertama-tama penentuan nilai baris yang ditentukan oleh bit ke 11 10 dan 1 0.

```
r=((i>>8)&0xc)|(i&0x3);
```

Setelah penentuan baris selesai, selanjutnya adalah penentuan kolom dengan menggunakan bit [9 8 .. 3 2].

```
c=(i>>2)&0xff;
```

Kemudian menentukan nilai basis untuk Sfn.

```
t=(c+((r*17)^0xff))&0xff;
```

Tahapan terakhir yang dilakukan adalah menghitung nilai Sfn[r] = t ^ exp mod gen.

```
v=exp8(t,sfn[r].exp,sfn[r].gen);
```

Dengan demikian nilai yang dihasilkan dari fungsi S-box ini adalah v.

Permutasi : perm32(out, in, perm)

Fungsi permutasi pada LOKI91 sama dengan fungsi permutasi pada LOKI89 dimana melakukan perasi permutasi biasa yang mendapat masukan 32 bit blok in, menghasilkan 32 bit blok out di bawah kontrol lanjar permutasi perm. Setiap elemen dari perm menspesifikasikan bit masukan mana yang harus dipermutasikan menjadi bit keluaran dengan indeks yang sama dengan elemen perm.

Fungsi permutasi ini dilakukan dengan cara :

```
for (o=0; o<32; o++)
 {
  i =(int)*p++;
  b = (*in >> i) & 01;

  if (b)
      *out |= mask;
  mask >>= 1;
 }
```

/* Untuk setiap bit keluaran di posisi o, ambil nilai masukan yang akan dipermutasikan ke bit keluaran o. Hitung nilai dari untuk bit masukan i. Jika i telah diset, lakukan OR pada mask untuk bit keluaran i. Geser mask ke bit selanjutnya.
*/

Dengan :
mask = 0x80000000L
*p = perm
o = indeks bit keluaran
i = indeks bit masukan

Eksponensial : exp8(base, exponent, gen)

Fungsi eksponensial LOKI91 sama dengan LOKI89, tidak ada perubahan sama sekali. Fungsi ini tetap melakukan operasi :

```
exp = base ^ exp mod gen
```

Proses dari fungsi ini adalah sebagai berikut.

```
if (base == 0)
  return(0);

while (exponent != 0)
{
if((exponent&0x0001)== 0x0001)
   result = mult8(result,accum,
   gen);
exponent >>= 1;
accum = mult8(accum, accum,
gen);
}
return(result);
```

/*
jika nilai base adalah 0 maka fungsi akan otomatis mengembalikan nilai 0. jika tidak 0 maka dilakukan proses perulangan hingga exponent menjadi 0. Eksponen akan dikenai fungsi mult8 jika nilai eksponen adalah 1. Lalu nilai eksponen bergeser ke dijit selanjutnya.
*/

Multiply : mult8(a, b, gen)

Fungsi mult8 pada LOKI91 sama dengan fungsi mult8 pada LOKI89 yaitu mengembalikan hasil perkalian dari dua buah string biner a dan b serta menggunakan generator gen sebagai modulus.

```
mult = a * b mod gen
```

Proses dari fungsi ini adalah sebagai berikut.

```
while(b != 0)
{
if (b & 01)
   product ^= a;
a <<= 1;
if (a >= SIZE)
   a ^= gen;
b >>= 1;
}
return(product);
```

/*
selama pengali b tidak 0, lakukan proses berikut ini : jika LSB dari b merupakan 01, tambahkan hasil perkalian dengan a. Geser a sebanyak satu dijit. Jika a lebih besar dari 256, maka a dikurangi modulo. Geser a sebanyak satu dijit.
*/

- Kelemahan LOKI91

Berikut ini adalah beberapa kelemahan yang terdapat pada LOKI91.

1. Dengan dihilangkannya operasi XOR di awal dan akhir proses mengakibatkan adanya ketergantungan cipherteks pada kunci di putaran 3 sampai 5.

5. LOKI97

LOKI97 merupakan sebuah algoritma blok *cipher* menggunakan kunci privat yang mengoperasikan data sebesar 128 bit

dengan kunci sebesar 256 bit. LOKI97 merupakan evolusi dari LOKI89 dan LOKI91 dengan memperkuat penjadwalan kunci dan memperbesar ukuran kunci yang digunakan. LOKI97 diajukan sebagai kandidat untuk menggantikan DES (Data Encryption Standard).

Struktur dan Desain LOKI97 secara keseluruhan dibuat oleh Dr. Lawrie Browns. Sedangkan desain dari S-box yang digunakan dibuat oleh Prof. Pieprzyk.

❖ Spesifikasi LOKI97

LOKI97 melakukan proses enkripsi/dekripsi data 128 bit menggunakan kunci sepanjang 128, 192, atau 256 bit.

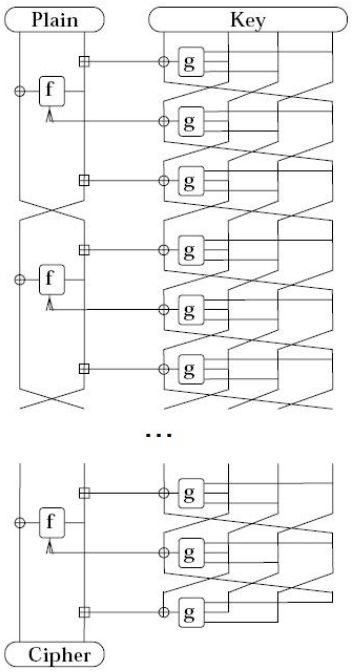
Karena LOKI97 menerima masukan data sepanjang 128 bit dan dikarenakan LOKI97 merupakan algoritma blok *cipher* maka LOKI97 akan menghasilkan data keluaran sepanjang 128 bit pula.

Proses pertama yang dilakukan dalam melakukan enkripsi adalah membagi blok masukan menjadi dua buah blok yang sama panjang yaitu L dan R. Hal ini juga dilakukan dalam LOKI89 dan LOKI91.

Setelah data dibagi menjadi dua buah bagian yang sama besar, dilakukan proses putaran sebanyak 16 kali dengan menggunakan jaringan Feistel. Proses yang dilakukan dalam setiap putaran adalah sebagai berikut.

$R_i = L_{i-1}$ XOR $f(R_{i-1} + SK_{3i-2}, SK_{3i-1})$
$L_i = R_{i-1} + SK_{3i-2} + SK_{3i}$

Setiap putaran menggunakan operasi XOR dan + (modulo 2^{64}) terhadap 64 bit data masukan bersamaan dengan hasil fungsi f yang merupakan fungsi nonlinier yang rumit sehingga memaksimalkan *avalanche* antara bit-bit masukan.



Gambar 11 Proses Enkripsi LOKI97

Sedangkan untuk proses dekripsi, proses putaran hanya dibalik. Dengan demikian perhitungannya adalah sebagai berikut.

$L_{i-1} = R_i$ XOR $f(L_i - SK_{3i}, SK_{3i-1})$
$R_{i-1} = L_i - SK_{3i} - SK_{3i-2}$

❖ Pembangkitan Subkunci LOKI97

LOKI97 menggunakan jadwal pembangkitan subkunci berdasarkan pada jaringan Feistel yang tidak seimbang yang mengoperasikan 4 buah blok sebesar 64 bit $[K4_0|K3_0|K2_0|K1_0]$. Dikarenakan ukuran kunci yang digunakan bisa mempunyai 3 buah ukuran, maka inisialisasi kunci untuk setiap ukuran pun berbeda-beda. Untuk kunci sepanjang 256 bit [Ka|Kb|Kc|Kd], maka inisialisasinya adalah :
$[K4_0|K3_0|K2_0|K1_0]$ = [Ka|Kb|Kc|Kd]

Untuk kunci sepanjang 192 [Ka|Kb|Kc] bit, inisialisasinya adalah dengan cara :
$[K4_0|K3_0|K2_0|K1_0]$=
[Ka|Kb|Kc|f(Ka,Kb)]

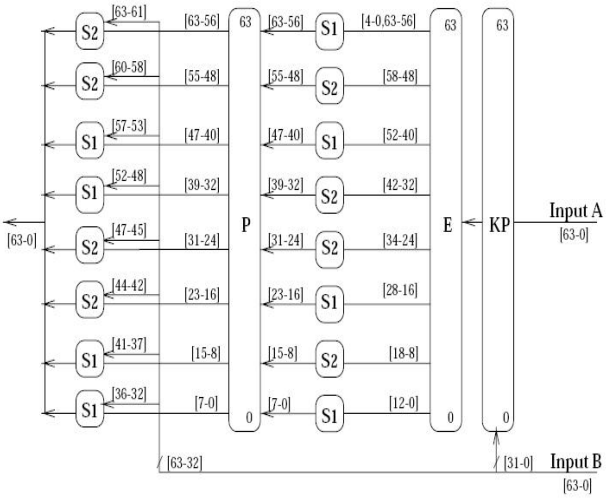
Sedangkan untuk kunci sepanjang 128 bit [Ka|Kb], proses inisialisasi dilakukan dengan cara :
$[K4_0|K3_0|K2_0|K1_0]$=
[Ka|Kb|f(Kb,Ka)|f(Ka,Kb)]

Berikut ini adalah proses untuk mendapatkan subkunci SK_i dengan melakukan putaran sebanyak 48 kali.
$SK_i = K1_i = K4_{i-1}$ XOR $g_i(K1_{i-1}, K3_{i-1}, K2_{i-1})$
$K4_i = K3_{i-1}$
$K3_i = K2_{i-1}$
$K2_i = K1_{i-1}$
Dengan i mulai dari 1 sampai dengan 48 dan :
$g_i(K1, K3, K2_i)=$
$f(K1+K3+ (Delta * i), K2)$
$Delta = \lfloor (sqrt(5)-1) * 2^{63} \rfloor$

❖ Fungsi f LOKI97

Fungsi f ini menerima dua buah masukan sebesar 64 bit dan menghasilkan sebuah blok berukuran 64 bit. Cara pemrosesannya adalah dengan menggunakan S-box dua lapis dengan nilai nonlinier maksimal yang memungkinkan untuk menghasilkan sebuah blok berukuran 64 bit. Selain menggunakan S-box, digunakan pula permutasi sebanyak dua kali untuk memaksimalkan nilai *avalanche* pada keseluruhan bit. Fungsi f ini didefinisikan sebagai berikut.

F(A,B) = Sb(P(Sa(E(KP(A,B)))),B)



Gambar 12 Fungsi f LOKI97

Pada deskripsi fungsi f diatas, terdapat beberapa fungsi antara, yaitu:

a. KP(A,B)
Fungsi ini merupakan fungsi permutasi biasa yang memecah A menjadi dua buah bagian masinng-masing sebesar 32 bit dan menggunakan 32 bit yang paling kanan dari B untuk menentukan apakah menukar bit pada posisi ini.

b. E()
Merupakan fungsi perluasan seperti yang terdapat dalam LOKI91 namun pergeseran yang ada lebih disederhanakan.

c. Sa(), Sb()
Sa() dan Sb() merupakan dua buah kolom dari S-box yang dibentuk dengan cara menggabungkan S1 dan S2 dimana Sa() = [S1,S2,S1,S2,S2,S1,S2,S1] dan Sb() = [S2,S2,S1,S1,S2,S2,S1,S1].

d. P()
Merupakan sebuah fungsi permutasi untuk menyebarkan hasil keluaran dari S-box untuk dibentuk menjadi blok berukuran 64 bit.

❖ S-Box LOKI97

LOKI97 menggunakan S-box berbentuk kotak memakai nilai Galois Field yang ganjil $GF(2^n)$. Supaya masukan yang didapat berjumlah ganjil maka S1 menggunakan masukan sebanyak 13 bit, sedangkan S2 menggunakan masukan sebanyak 11 bit. Nilai masukan yang dimasukan dibalik terlebih dahulu agar nilai masukan 1 tidak akan menghasilkan 1, begitu pula nilai masukan 0 tidak akan menghasilkan 0.

S1[x]=
$((x \text{ XOR } 1FFF^{3}) \bmod 2911) \& FF$,
dalam $GF(2^{13})$.

S2[x]=
$((x \text{ XOR } 7FF^{3}) \bmod AA7) \& FF$,
dalam $GF(2^{11})$.

❖ Kelemahan LOKI97
Menurut [KNU99], kelemahan LOKI97 adalah :
a. jumlah masukan yang tetap pada S-Box lapis kedua memungkinkan terjadinya *linier attack*.
b. Fungsi putaran yang terdapat di dalama LOKI97 memungkinkan terjadinya *differential attack*.

Kedua hal diatas pula yang dianggap menjadi penyebab mengapa LOKI97 menjadi kandidat lemah untuk AES.

6. Kesimpulan

Kesimpulan yang dapat ditarik dari ketiga buah generasi LOKI diatas adalah sebagai berikut.

1. Setiap versi LOKI pada umumnya terdiri dari :
   a. Permutasi
   b. Ekspansi
   c. Pembangkitan kunci yang dilakukan sebanyak 16 putaran.
   d. Pembagian blok kunci dan blok masukan menjadi dua buah blok yang sama besar.
   e. Penggunaan S-Box untuk membangkitkan subkunci.
2. Pada dasarnya keseluruhan proses LOKI adalah sama, khususnya untuk LOKI89 dan LOKI91. Perbedaan yang ada hanya pada pendeklarasian fungsi f.
3. Evolusi yang dilakukan dari versi satu ke versi lain hanyalah merubah fungsi S-box, permutasi, dan sebagainya, bukan merubah cara kerja itu sendiri.
4. Untuk mendapatkan algoritma yang aman, perlu adanya subkunci-subkunci yang dibangkitkan. Semakin rumit cara pembangkitannya, semakin kecil kemungkinan algoritma tersebut dapat diserang.

7. Daftar Referensi

[BRO98] Brown, L., Pieprzyk. Introducing the new LOKI97 Block Cipher. 1998.

[ECO06] http://eco.utexas . Diakses pada tanggal 25 September 2006.

[KNU99] Knudsen, L. R., Rijmen, V. Weaknesses in LOKI97

[WIK06] http://wikimirror.com. Diakses pada tanggal 25 September 2006.