STUDI ALGORITMA CIPHER BLOK KUNCI SIMETRI BLOWFISH CIPHER

Yoseph Suryadharma – NIM. 13504037

Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jalan Ganesha 10 Bandung
Email: if14037@students.if.itb.ac.id

Abstraksi

Makalah ini akan membahas mengenai studi algoritma *cipher* blok kunci simetri Blowfish *cipher* untuk pengamanan data. Blowfish *cipher* diciptakan oleh Bruce Schneier pada tahun 1993 sebagai pengganti *Data Encryption Standard (DES)*. Algoritma ini tidak dipatenkan oleh Bruce Schneier sehingga semua orang bebas untuk menggunakan algoritma ini untuk pengamanan data. Blowfish *cipher* memiliki panjang kunci sampai 448 bit dan memiliki blok data sebesar 64 bit. Blowfish *cipher* merupakan jaringan Feistel dengan 16 putaran dan menggunakan *key-dependent S-boxes* yang besar. Blowfish *cipher* relatif aman karena kunci dibangun secara acak diambil dari heksadesimal digit bilangan pi (selain angka 3 di depan koma). Blowfish *cipher* dirancang sedemikian rupa sehingga algoritma untuk enkripsi dan dekripsi datanya sama kecuali penggunaan *subkey*nya. *Subkey* untuk dekripsi merupakan kebalikan dari *subkey* untuk enkripsi. Hingga saat ini belum ditemukan kriptanalisis yang bisa memecahkan seluruh 16 putaran algoritma blowfish. Blowfish terbukti memiliki tingkat keamanan yang lebih baik dibandingkan dengan algoritma DES.

Kata kunci: *cipher* blok, kunci simetri, Blowfish *cipher*, jaringan Feistel, DES, enkripsi, dekripsi, *subkey, S-boxes*.

1. Pendahuluan

Pada saat ini keamanan data menjadi suatu hal yang sangat penting dalam suatu proses pertukaran data.
Proses pengiriman data tidak bisa lepas dari metode enkripsi dan dekripsi data untuk menjamin krahasiaan data. Dengan proses pengenkripsian data, informasi tidak akan bisa diketahui oleh pihak yang tidak berwenang. Enkripsi dilakukan oleh pengirim dengan cara mengubah data asli menjadi suatu kode dengan suatu kunci tertentu yang tidak bisa dipecahkan oleh orang lain. Data yang telah dienkripsi ini kemudian dikirimkan kepada pihak penerima. Penerima kemudian mendekripsi data yang telah dienkripsi oleh pihak pengirim untuk mengetahui isi pesan yang asli. Apabila algoritma kriptografi yang dipakai untuk pengenkripsian data tidak cukup kuat untuk menangkal serangan terhadap data yang telah dienkripsi, sia-sialah pengenkripsian data tersebut. Oleh karena itu, para kriptografer berusaha untuk menemukan algoritma kriptografi yang dipandang cukup aman untuk mengenkripsikan data.

Pada tahun 1972 IBM di bawah pimpinan W.L Tuchman mengembangkan algoritma kriptografi yang diberi nama Data Encryption Standard (DES). DES kemudian menjadi algoritma yang populer karena dijadikan standar algoritma enkripsi data kunci simetri sejak tahun 1977. DES termasuk ke dalam sistem kriptografisimetri dan tergolong jenis cipher blok. DES memiliki panjang kunci internal 56 bit untuk mengenkripsi 64 bit plainteks menjadi 64 bit plainteks. Kunci internal dibangkitkan dari kunci eksternal yang panjangnya 64 bit. Saat ini DES sudah dianggap tidak aman lagi untuk pengenkripsian data karena panjang kunci 64 bit dianggap pendek. Tahun 1999 kombinasi perangkat keras Electronic Frontier Foundation (EFE) dengan kolaborasi internet yang melibatkan lebih dari 100.000 komputer dapat menemukan kunci DES kurang dari satu hari.

Bruce Schneier menciptakan algoritma Blowfish *cipher* pada tahun 1993 sebagai pengganti algoritma *Data Encryption Standard (DES)*. Blowfish *cipher* ciptaan Bruce Schneier mirip dengan algoritma DES dengan sejumlah penyempurnaan. Blowfish *cipher* menggunakan kunci yang relatif panjang yaitu hingga 448 bit. Kunci ini terdiri dari 18 subkunci. Blowfish *cipher* merupakan jaringan Feistel dengan 16 putaran dan menggunakan key-*dependent S-boxes* yang besar (32 bit *S-boxes* yang memiliki 256 *entry*). Pembangkitan

18 subkunci dan *S-boxes* ini didasarkan atas digit-digit heksa desimal yang terdapat dalam bilangan pi (selain angka 3 di depan koma). Digit-digit bilangan pi merupakan suatu bilangan yang benar-benar acak yang diketahui saat ini. Oleh karena hal-hal di atas, algoritma blowfish ini lebih aman dibandingkan dengan algoritma DES. Sampai saat ini (tahun 2006) belum diketahui cara untuk memecahkan 16 putaran algoritma blowfish *cipher*.

2. Algoritma Kriptografi Kunci Simetri

Algoritma kriptografi simetri yang beroperasi dalam mode bit dibagi menjadi 2 kategori:

1. Cipher aliran (stream cipher)
   Algoritma ini beroperasi pada plainteks/cipherteks dalam bentuk bit tunggal yang dalam hal ini rangkaian bit dienkripsikan/didekripsikan bit per bit.

2. Cipher blok (block cipher)
   Algoritma kriptografi beroperasi pada plainteks/cipherteks dalam bentuk blok bit yang dalam hal ini rangkaian bit dibagi menjadi blok-blok bit yang panjangnya sudah ditentukan sebelumnya.

Algoritma kunci simetri menggunakan kunci yang sama untuk proses pengenkripsian maupun pendekripsian data.

2.1 Algoritma Cipher Blok

Karakteristik dari algoritma cipher blok adalah:

1. Pada algoritma cipher blok rangkaian bit-bit plainteks dibagi menjadi blok-blok bit yang panjangnya sama, pada umumnya 64 bit .

2. Enkripsi dilakukan terhadap blok bit plainteks menggunakan bit-bit kunci yang ukurannya sama dengan ukuran blok plainteks. Algoritma enkripsi menghasilkan blok cipherteks yang berukuran sama dengan blok plainteks.

3. Dekripsi dilakukan dengan cara yang sama dengan enkripsi.

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

$$P = (p_1, p_2, \ldots, p_m)$$

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

$$C = (c_1, c_2, \ldots, c_m)$$

yang dalam hal ini c_i adalah 0 atau 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_n bit-bit penyusunnya dapat dinyatakan sebagai vektor

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

Enkripsi dan dekripsi dengan kunci K dinyatakan berturut-turut dengan persamaan

$$E_K(P) = C$$

untuk enkripsi, dan

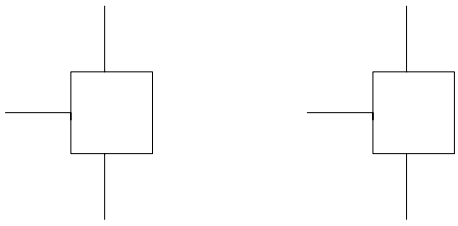
$$D_K(C) = P$$

untuk dekripsi.

Fungsi E haruslah suatu fungsi satu ke satu sehingga

$$E^{-1} = D$$

Skema enkripsi dan dekripsi dengan cipher blok digambarkan pada gambar di bawah ini. Fungsi E dan D dispesifikasikan oleh kriptografer.



2.2 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 cipher blok adalah:

1. Electronic Code Book (ECB).
2. Cipher Block Chaining (CBC).

3. Cipher Feedback (CFB).
4. Output Feedback (OFB).

2.2.1 Electronic Code Book (ECB)

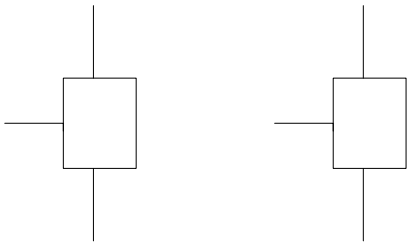
Pada model ini setiap blok plainteks P_i dienkripsi secara individual dan independen menjadi blok cipherteks C_i. Secara matematis, enkripsi dengan metode ECB dinyatakan sebagai

$$C_i = E_K(P_i)$$

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. Enkripsi dan dekripsi dengan mode ECB diperlihatkan dalam gambar di bawah ini.



Dalam hal ini E menyatakan fungsi enkripsi dan D menyatakan fungsi dekripsi menggunakan kunci K.

2.2.2 Cipher Block Chaining

Mode ini menerapkan mekanisme umpan-balik (feedback) pada sebuah blok, yang salam hal ini hasil enkripsi blok sebelumnya diumpan-balikkan ke dalam enkripsi blok 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 kemudian meng-XOR-kan hasilnya dengan blok cipherteks sebelumnya. Dalam hal ini, blok cipherteks sebelumnya berfungsi sebagai umpan-maju (feedforward) pada akhir proses dekripsi.

Secara matematis enkripsi dengan mode CBC dinyatakan sebagai

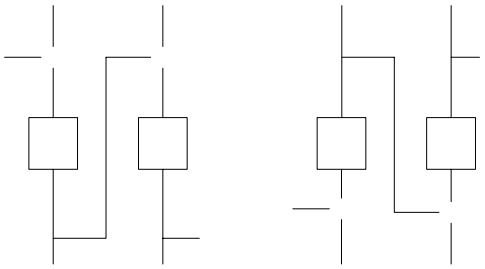
$$C_i = E_K(P_i \oplus C_{i-1})$$

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, IV digunakan untuk menggantikan blok cipherteks sebelumnya. Sebaliknya pada dekripsi blok plainteks diperoleh dengan cara meng-XOR-kan IV dengan hasil dekripsi terhadap blok cipherteks yang pertama. Perhatikan bahwa enkripsi terhadap blok i adalah fungsi dari semua plainteks dari blok 0 sampai blok i-1, sehingga blok plainteks yang sama menghasilkan blok cipherteks yang berbeda hanya jika blok-blok plainteks sebelumnya berbeda. Jika blok-blok plainteks sebelumnya ada yang sama, ada kemungkinan cipherteksnya sama. Untuk mencegah hal ini, digunakan IV yang merupakan data acak sebagai blok pertama. IV tidak mempunyai makna, ia hanya digunakan untuk membuat tiap blok cipherteks menjadi unik.



2.2.3 Cipher Feedback (CFB)

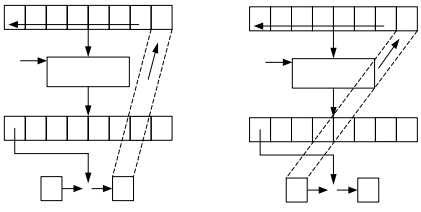
Pada mode *CFB*, data dienkripsikan dalam unit yang lebih kecil daripada ukuran blok. Unit yang dienkripsikan dapat berupa bit per bit, 2 bit, 5 bit, dan seterusnya. Bila unit yang dienkripsikan satu karakter setiap kalinya, maka mode *CFB*-nya disebut *CFB* 8-bit. Secara umum *CFB n*-bit mengenkripsi 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.



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 di atas.

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

Proses Enkripsi:

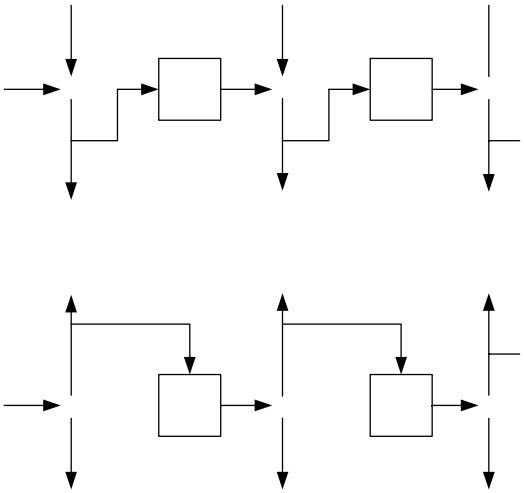
$$C_i = P_i \oplus MSB_m(E_k(X_i))$$
$$X_{i+1} = LSB_{m-n}(X_i) \,||\, C_i$$

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



Jika *m* = *n*, maka mode *CFB n*-bit adalah seperti pada gambar di atas. *CFB* menggunakan skema umpan balik dengan mengaitkan blok plainteks bersama-sama sedemikian sehingga cipherteks bergantung pada semua blok plainteks sebelumnya. Dari gambar dapat dilihat bahwa:

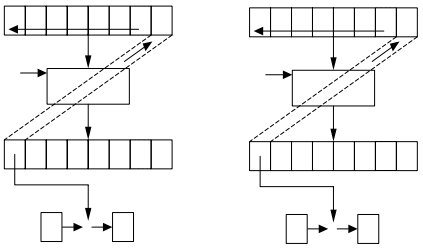
$$C_i = P_i \oplus E_k(C_{i-1})$$
$$P_i = C_i \oplus D_k(C_{i-1})$$

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.

2.2.4 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:

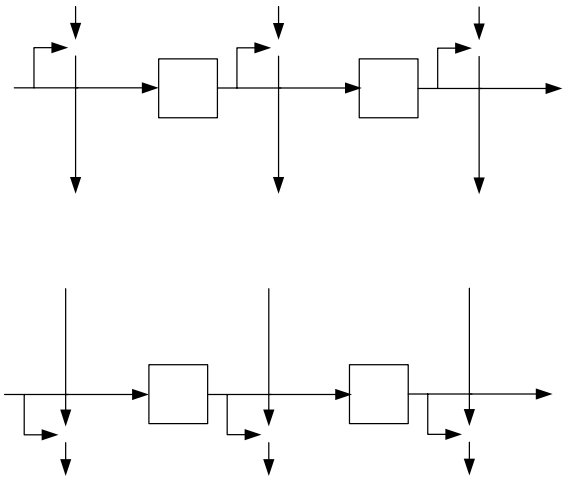
1. Antrian diisi dengan *IV* (*initialization vector*).
2. Enkripsikan antrian dengan kunci *K*. *n* bit paling kiri dari hasil enkripsi dimasukkan ke dalam antrian (menempati *n* posisi bit paling kanan antrian), dan *m*-*n* bit lainnya di dalam antrian digeser ke kiri menggantikan *n* bit pertama yang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



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

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.



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 di atas. *OFB*

menggunakan skema umpan balik dengan mengaitkan blok plainteks bersama-sama sedemikian sehingga cipherteks bergantung pada semua blok plainteks sebelumnya.

2.3 Prinsip Perancangan *Cipher* Blok

Perancangan algoritma kriptografi yang berbasis blok mempertimbangkan beberapa prinsip berikut:

1. Prinsip *confusion* dan *diffusion* Shannon.
2. *Cipher* berulang.
3. Jaringan Feistel.
4. Kunci lemah.
5. Kotak –S (*S-box*)

2.3.1 Prinsip *Confusion* dan *Diffusion* Shannon

1. *Confusion*

Prinsip ini menyembunyikan hubungan apapun yang ada antara plainteks, cipherteks, dan kunci. Sebagai contoh, pada *cipher* subtitusi seperti *caesar cipher*, hubungan antara cipherteks dan plainteks mudah diketahui, karena satu huruf yang sama pada plainteks diganti dengan satu huruf yang sama pada cipherteks.

Prinsip *confusion* akan membuat kriptanalis frustasi untuk mencari pola-pola statistik yang muncul pada cipherteks. *Confusion* yang bagus membuat hubungan statistik antara plainteks, cipherteks, dan kunci menjadi rumit.

2. *Diffusion*

Prinsip ini menyebarkan pengaruh satu bit plainteks atau kunci ke sebanyak mungkin cipherteks. Sebagai contoh, pengubahan kecil pada plainteks sebanyak satu atau dua bit menghasilkan perubahan pada cipherteks yang tidak dapat diprediksi.

Prinsip *diffusion* juga menyembunyikan hubungan statistik antara plainteks, cipherteks, dan kunci dan 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.

2.3.2 *Cipher* Berulang

Fungsi transformasi sederhana yang mengubah plainteks menjadi cipherteks diulang sejumlah kali. Pada setiap putaran digunakan subkunci atau kunci putaran yang dikombinasikan dengan plainteks.

Secara formal, *cipher* berulang dinyatakan sebagai:

$$C_i = f(C_{i-1}, K_i)$$

yang dalam hal ini:

$i = 1,2,\ldots, r$ (r adalah jumlah putaran)
K_i = subkunci pada putaran ke-i
f = fungsi transformasi

Plainteks dinyatakan dengan C_0 dan cipherteks dinyatakan dengan C_r .

2.3.3 Jaringan Feistel

Hampir semua algoritma *cipher* blok bekerja dalam model jaringan Feistel. Jaringan Feistel ditemukan oleh Horst Feistel pada tahun 1970.

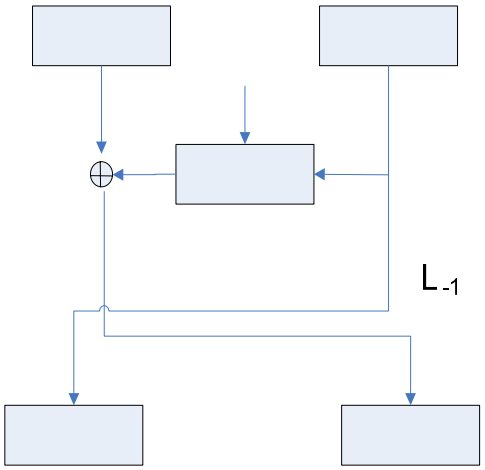
Model jaringan Feistel adalah sebagai berikut:

1. Bagi blok yang panjangnya n bit menjadi dua bagian kiri dan kanan yang panjangnya masing-masing n/2 (n harus genap).
2. Definisikan *cipher* blok berulang di mana hasil dari putaran ke-i ditentukan dari hasil putaran sebelumnya, yaitu:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

yang dalam hal ini,

$i = 1,2,\ldots,r$ (r adalah jumlah putaran)
K_i = subkunci pada putaran ke-i
f = fungsi transformasi.



Plainteks adalah gabungan L dan R awal, atau secara formal dinyatakan dengan (L_0,R_0), sedangkan cipherteks didapatkan dari L dan R hasil putaran terakhir setelah terlebih dahulu dipertukarkan, atau secara formal dinyatakan sebagai (R_r, L_r).

Jaringan Feistel ini bersifat *reversible* sehingga tidak perlu membuat algoritma baru untuk mendekripsi cipherteks menjadi plainteks. Karena operator Xor mengkombinasikan setengah bagian kiri dengan hasil dari fungsi transformasi f, maka kesamaan berikut pasti benar:

$$L_{i-1} \oplus f(R_{i-1},K_i) \oplus f(R_{i-1},K_i) = L_{i-1}$$

Sifat *reversible* tidak bergantung pada fungsi f sehingga fungsi f dapat dibuat serumit mungkin.

2.3.4 Kunci Lemah

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 terhadap plainteks akan menghasilkan kembali plainteksnya.

Misalkan K_L adalah kunci lemah, E adalah fungsi enkripsi, D adalah fungsi dekripsi, P adalah plainteks, dan C adalah cipherteks, maka persamaan berikut menunjukkan fenomena kunci lemah:

$E_{KL}(P) = C$
$D_{KL}(C) = E_{KL}(C) = P$

Cipher blok yang bagus tidak mempunyai kunci lemah.

2.3.5 Kotak-S (*S-box*)

Kotak-S adalah matriks yang berisi substitusi sederhana yang memetakan satu atau lebih bit dengan satu atau lebih bit yang lain. Pada kebanyakan algoritma *cipher* blok, kotak-S memetakan m bit masukan menjadi n bit keluaran, sehingga kotak-S tersebut dinamakan kotak m x n *S-box*.

Kotak-S merupakan satu-satunya langkah nirlanjar di dalam algoritma, karena operasinya adalah *look-up table*. Masukan dari operasi *look-up table* dijadikan sebagai indeks kotak-S dan keluarannya adalah entri di dalam kotak-S. Perancangan kotak-S menjadi isu yang sangat penting karena kotak-S harus dirancang sedemikian rupa sehingga kekuatan kriptografinya bagus dan mudah diimplementasikan. Ada empat cara (pendekatan) yang dapat digunakan dalam mengisi kotak-S:

1. Dipilih secara acak
   Untuk kotak-S yang kecil, cara pengisian secara acak tidak aman, namun untuk kotak-S yang besar cara ini cukup bagus.
2. Dipilih secara acak lalu diuji.
   Sama seperti cara nomor 1 tetapi nilai acak yang dibangkitkan diuji apakah memenuhi sifat tertentu.
3. Dibuat oleh orang.
   Entri di dalam kotak-S dibangkitkan dengan teknik yang lebih intuitif
4. Dihitung secara matematis.
   Entri di dalam kotak-S dibangkitkan berdasarkan prinsip matematika yang terbukti aman dari serangan kriptanalis.

3. Data Encryption Standard (DES)

DES adalah algoritma *cipher* blok yang populer karena dijadikan standar algoritma kunci simetri, meskipun saat ini standar tersebut telah digantikan dengan algoritma yang baru, AES, karena DES sudah dianggap tidak aman lagi. Algoritma DES dikembangkan di IBM di bawah kepemimpinan W.L Tuchman pada tahun 1972. Algoritma ini didasarkan pada algoritma Lucifer yang dibuat oleh Horst Feistel.

DES merupakan sistem kriptografi *cipher* blok simetri yang beroperasi pada ukuran blok 64 bit. DES mengenkripsikan 64 bit plainteks menjadi 64 bit cipherteks dengan menggunakan 56 bit kunci internal atau

subkunci. Kunci internal dibangkitkan dari kunci eksternal yang panjangnya 64 bit.

Skema global dari algoritma DES adalah sebagai berikut:

1. Blok plainteks dipermutasi dengan matriks permutasi awal.
2. Hasil permutasi awal kemudian dienkripsi sebanyak 16 kali. Setiap putaran menggunakan kunci internal yang berbeda.
3. Hasil enkripsi kemudian dipermutasi dengan matriks permutasi balikan menjadi blok cipherteks.

Di dalam proses enkripsi, blok plainteks terbagi menjadi dua bagian, kiri (L) dan kanan (R), yang masing-masing panjangnya 32 bit. Kedua bagian ini masuk ke dalam 16 putaran DES. pada setiap putaran i, blok R merupakan masukan untuk fungsi transformasi yang disebut f.

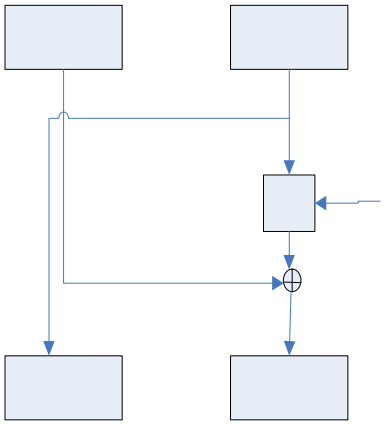
Pada fungsi f, blok R dikombinasikan dengan kunci internal K_i. Keluaran dari fungsi f di-Xor-kan dengan blok L untuk mendapatkan blok R yang baru. Sedangkan blok L yang baru langsung diambil dari blok R sebelumnya. Ini adalah satu putaran DES.

Secara matematis, satu putaran DES dinyatakan sebagai:

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus f(R_{i-1},K_i)$

Satu putaran DES merupakan model jaringan Feistel. Gambar jaringan Feistel untuk satu putaran DES adalah sebagai berikut:



Setiap putaran pertama terhadap blok plainteks dilakukan permutasi awal (*initialization permutation* atau IP). Tujuan permutasi awal adalah mengacak plainteks sehingga urutan bit-bit di dalamnya berubah.

Karena ada 16 putaran maka dibutuhkan kunci internal sebanyak 16 buah, yaitu K_1, K_2, …, K_{16}. Kunci-kunci internal ini dapat dibangkitkan sebelum proses enkripsi atau bersamaan dengan proses enkripsi. Kunci internal dibangkitkan dari kunci eksternal yang diberikan oleh pengguna. Kunci eksternal panjangnya adalah 64 bit atau 8 karakter.

Misalkan kunci eksternal yang tersusun dari 64 bit adalah K. Kunci eksternal ini menjadi masukan untuk permutasi dengan menggunakan matriks permutasi kompresi PC-1.

Dalam permutasi ini, tiap bit ke delapan (*parity bit*) dari delapan byte kunci diabaikan. Hasil permutasinya adalah 56 bit, sehingga dapat dikatakan bahwa panjang kunci DES adalah 56 bit. Selanjutnya, 56 bit ini dibagi menjadi 2 bagian kanan dan kiri, yang masing-masing panjangnya 28 bit, yang masing-masing disimpan di dalam C_0 dan D_0.

C_0 = berisi bit-bit pada posisi
57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18,
10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36

D_0 = berisi bit-bit pada posisi
63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22
14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4

Selanjutnya, kedua bagian digeser ke kiri (*left shift*) sepanjang satu atau dua bit bergantung pada tiap putaran. Operasi pergeseran bersifat *wrapping* atau *round shift*. Jumlah pergeseran setiap putaran ditunjukkan oleh tabel berikut:

Putaran	Jumlah pergeseran
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2

13	2
14	2
15	2
16	1

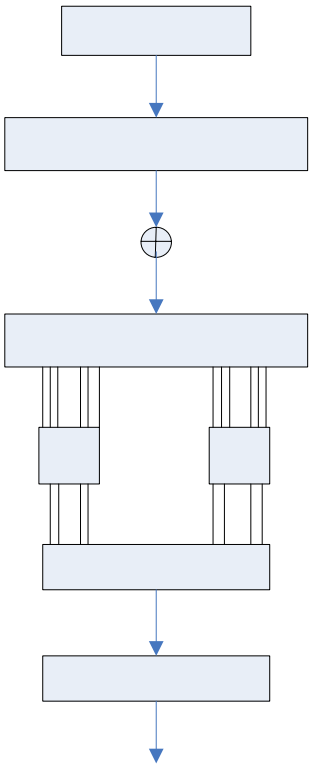
Misalkan (C_i,D_i) menyatakan penggabungan C_i dan D_i. (C_{i+1},D_{i+1}) diperoleh dengan menggeser C_i dan D_i satu atau dua bit. Setelah pergeseran bit, (C_i,D_i) mengalami permutasi kompresi dengan menggunakan matriks PC-2.

Dengan permutasi ini, kunci internal K_i diturunkan dari (C_i, D_i) yang dalam hal ini K_i merupakan penggabungan bit-bit C_i dan D_i. Jadi, setiap kunci internal K_i mempunyai panjang 48 bit.

Proses enkripsi terhadap blok plainteks dilakukan setelah permutasi awal. Setiap blok plainteks mengalami 16 putaran enkripsi. Setiap putaran enkripsi merupakan jaringan Feistel yang secara matematis dinyatakan sebagai

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1},K_i)$$

Diagram komputasi fungsi f dapat dilihat dari gambar berikut ini:



E adalah fungsi ekspansi yang memperluas blok R_{i-1} yang panjangnya 32-bit menjadi blok 48 bit. Fungsi ekspansi direalisasikan menggunakan matriks permutasi ekspansi.

Selanjutnya, hasil dari ekspansi, yaitu $E(R_{i-1})$ yang panjangnya 48 bit di-Xor-kan dengan K_i yang panjangnya 48 bit menghasilkan vektor A yang panjangnya 48 bit:

$$E(R_{i-1}) \oplus K_i = A$$

Vektor A dikelompokkan menjadi 8 kelompok, masing-masing 6 bit, dan menjadi masukan dalam proses substitusi. Proses substitusi dilakukan dengan menggunakan delapan buah kotak-S. Setiap kotak-S menerima masukan 6 bit dan menghasilkan keluaran 4 bit.

Keluaran proses substitusi adalah vektor B yang panjangnya 48 bit. Vektor B menjadi masukan untuk proses permutasi. Tujuan proses permutasi ini adalah untuk mengacak hasil substitusi kotak-S. Permutasi dilakukan dengan menggunakan matriks permutasi P (*P-box*).

Bit-bit P(B) merupakan keluaran dari fungsi f. Akhirnya, bit-bit P(B) di-Xor-kan dengan L_{i-1} untuk mendapatkan R_i.

$$R_i = L_{i-1} \oplus P(B)$$

Jadi, keluaran dari putaran ke-i adalah

$$(L_i,R_i) = (R_{i-1},L_{i-1} \oplus P(B))$$

Permutasi terakhir dilakukan setelah 16 kali putaran terhadap gabungan blok kiri dan blok kanan. Proses permutasi menggunakan matriks permutasi awal balikan (*invers initial permutation* atau IP^{-1}).

Proses dekripsi terhadap cipherteks merupakan kebalikan dari proses enkripsi. DES menggunakan algoritma yang sama untuk enkripsi dan dekripsi. Jika pada proses enkripsi urutan kunci internal yang digunakan adalah $K_1,K_2,...,K_{16}$ maka pada proses dekripsi adalah kebalikannya.

4. Blowfish C*ipher*

Blowfish cipher adalah algoritma cipher blok kunci simetri yang diciptakan oleh Bruce Schneier pada tahun 1993. Algoritma kriptografi ini dimaksudkan sebagai pengganti DES atau IDEA. Blowfish merupakan suatu jaringan Feistel dengan jumlah putaran

sebanyak 16 kali. Panjang blok yang dipakai dalam algoritma ini adalah 64 bit. Panjang kunci yang dipakai bisa bervariasi antara 32 bit sampai 448 bit. Blowfish dibuat oleh Bruce Schneier memenuhi kriteria algoritma kriptografi sebagai berikut:

1. Cepat. Blowfish mengenkripsi data pada mikroprosesor 32 bit dalam 26 *clock cycles per byte*.
2. Padat. Blowfish dapat berjalan pada memori kurang dari 5K.
3. Sederhana. Blowfish hanya mengunakan operasi sederhana: penjumlahan, XOR, dan tabel lookups pada operand 32 bit. Desainnya mudah dianalisis sehingga mudah menemukan jika terjadi kesalahan implementasi.
4. Aman. Panjang kunci Blowfish dapat bervariasi dari 32 bit sampai 448 bit.

4.1 Algoritma Blowfish

Algoritma Blowfish terdiri atas dua bagian: bagian ekspansi kunci dan bagian enkripsi data. Ekspansi kunci mengubah kunci sampai maksimal 448 bit menjadi beberapa *array* subkunci total sampai 4168 byte.

Enkripsi data dilakukan melalui suatu susunan jaringan Feistel dengan jumlah putaran 16 kali. Setiap putaran terdiri dari *key-dependent permutation dan key-and-data-dependent substitution*. Semua operasi adalah operasi Xor dan penjumlahan pada 32-bit *words*. Operasi tambahan adalah empat buah *lookups data array* untuk setiap putaran. Blowfish memiliki:

1. Delapan belas buah *P-array* yang berisi 32 bit subkunci.
2. Empat buah 32-bit *S-boxes* dengan 256 entri.

4.1.1 Subkunci

Blowfish menggunakan subkunci yang berjumlah banyak. Subkunci ini harus dikomputasi sebelum enkripsi maupun dekripsi data.

Algoritma pembangkitan subkuncinya adalah sebagai berikut:

1. Inisialisasi *P-array* pertama dan 4 buah *S-boxes* dengan suatu angka konstan. Angka ini harus mengandung digit heksadesimal dari bilangan pi (kecuali angka 3 di depan koma). Contohnya:

P1 = 0x243f6a88
P2 = 0x85a308d3
P3 = 0x13198a2e
P4 = 0x03707344

2. Lakukan operasi Xor P1 dengan 32 bit pertama dari kunci, P2 dengan 32 bit berikutnya, dan seterusnya.
3. Enkripsi semua *string* yang bernilai 0 dengan algoritma Blowfish menggunakan subkunci pada langkah 1 dan 2.
4. Ganti P1 dan P2 dengan hasil dari langkah 3.
5. Enkripsi hasil dari langkah 3 menggunakan algoritma Blowfish dengan subkunci yang telah dimodifikasi.
6. Ganti P3 dan P4 dengan hasil dari langkah 5.
7. Lakukan langkah-langkah di atas berulang kali, ganti semua entri *P-array* dan keempat *S-boxes* dengan hasil dari algoritma Blowfish.

Total ada 521 iterasi yang diperlukan untuk membangkitkan semua subkunci.
Dalam notasi algoritmik, pembangkitan subkuncinya adalah:

```
Input:
   K: The key - 32 bits or more
   PI: The binary
representation of the
fractional portion of "pi"
      = 3.1415927... - 3.0
      = 2/16 + 4*/16**2 +
3/16**3 + 15/16**4 + ...
      =
0x243f6a8885a308d313198a2e03707
344...

Output:
   P1, P2, ..., P18: 18 32-bit
sub-keys
   S1[], S2[], S3[], S4[]: 4 S-
boxes, 32-bit 256-element
arrays

Algorithm:
   (P1, P2, ..., P18, S1[],
S2[], S3[], S4[]) = PI
   K' = (K, K, K, ...), Repeat
the key to 18*32 bits long
   (P1, P2, ..., P18) = (P1,
P2, ..., P18) XOR K'
```

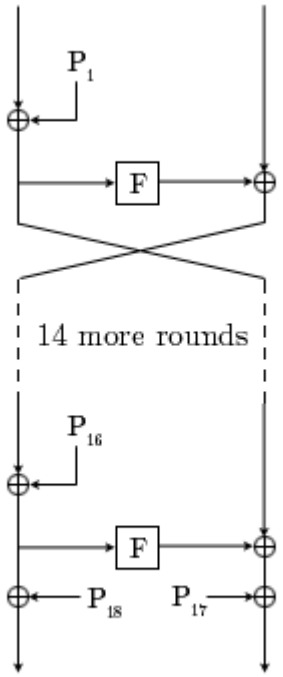
```
   T = (0x000000, 0x000000),
Setting initial clear text
   T = Blowfish(T), Applying
Blowfish algorithm
   (P1, P2) = T, Updating first
two sub-keys
   T = Blowfish(T), Applying
Blowfish again
   (P3, P4) = T
   ......
   T = Blowfish(T)
   (P17, P18) = T
   T = Blowfish(T)
   (S1[0], S1[1]) = T
   T = Blowfish(T)
   (S1[2], S1[3]) = T
   ......
   T = Blowfish(T)
   (S1[254], S1[255]) = T
   T = Blowfish(T)
   (S2[0], S2[1]) = T
   ......
   T = Blowfish(T)
   (S4[254], S4[255]) = T
```

4.1.2 Enkripsi dan Dekripsi

Blowfish merupakan suatu jaringan Feistel dengan 16 putaran. Inputnya adalah data 64 bit, x. Algoritma enkripsinya adalah sebagai berikut:

```
Bagi x menjadi xL dan xR yang
berukuran 32 bit.
for i=1 to 16
      xL = xL⊕ Pi
      xR = F(xL)⊕ xR
      Tukar xL dengan xR
Tukar xL dengan xR /*Batalkan
pertukaran terakhir*/
xR = xR⊕P17
xL = xL⊕P18
Gabung xL dengan xR
```

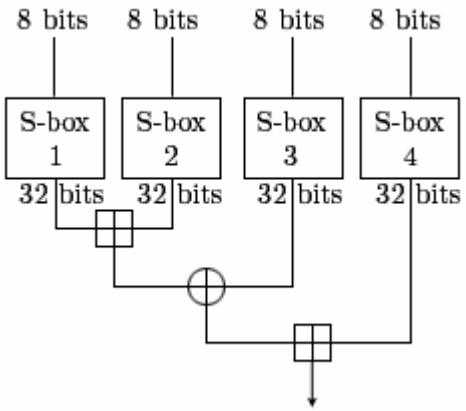
Ilustrasi gambarnya adalah sebagai berikut:



Fungsi F yang dipakai adalah sebagai berikut:

- Bagilah xL menjadi 4 dengan ukuran masing-masing 8 bit a,b,c,d.
- $F(xL) = ((S_{1,a} + S_{2,b} \bmod 2^{32}) \oplus S_{3,c}) + S_{4,d} \bmod 2^{32}$

Ilustrasi gambarnya adalah sebagai berikut:



Algoritma dekripsi sama saja dengan algoritma untuk enkripsi data. Hanya saja penggunaan P_1, P_2, P_3, ..., P_{18} adalah kebalikan dari enkripsi.

4.2 Analisis Desain Blowfish

Blowfish didesain dengan sangat sederhana sehingga mudah untuk dipahami dan mudah pula untuk diimplementasikan. Dengan

penggunaan jaringan Feistel, *S-box* sederhana, dan P-*array* algoritma Blowfish tidak mengandung kecacatan dan aman. Penggunaan operasi penjumlahan, Xor, dan subtitusi sangat efisien saat diimplementasikan pada komputer dan tidak membutuhkan memori yang besar.

Penggunaan fungsi F, fungsi *non-reversible*, menjadikan Blowfish memiliki efek saling mempengaruhi antarbit melalui jaringan Feistel: setiap bit pada xL mempengaruhi setiap bit pada xR untuk setiap putaran. Selain itu, karena setiap bit subkunci dipengaruhi oleh bit kunci, fungsi F ini juga memberikan dampak saling mempengaruhi antara kunci dan xR untuk setiap putaran. Penggunaan digit heksadesimal bilangan pi untuk inisialisasi *P-array* juga menambah keamanan algoritma blowfish. Digit-digit bilangan pi merupakan bilangan yang benar-benar acak dan tidak berpola yang diketahui saat ini.

Penggunaan 4 *S-boxes* yang berbeda berguna untuk menghindari kesimetrian saat *byte* input adalah sama, atau saat 32 bit input fungsi F adalah permutasi dari 32 bit input yang lain. *S-box* didesain dengan cepat, mudah diprogram, dan aman.

4.3 Keamanan Blowfish

Sampai saat ini (tahun 2006) belum ada publikasi mengenai kriptanalisis yang efektif untuk memecahkan semua putaran algoritma Blowfish. meskipun saat ini panjang blok 64 bit dianggap terlalu pendek. Panjang blok 64 bit dianggap pendek dan kurang aman karena enkripsi lebih dari 2^{32} blok data akan melemahkan informasi mengenai plainteks pada beberapa mode operasi dan mudah diserang dengan *birthday attack*.

Pada tahun 1996, Serge Vaudenay mengetahui bahwa *known-plaintext attack* memerlukan 2^{8r+1} *known-plaintext* untuk memecahkan, di mana r adalah jumlah putaran. Lebih jauh lagi, dia juga menemukan *weak-keys* yang dapat dideteksi dan dipecahkan hanya dengan 2^{4r+1} *known-plaintext*. Akan tetapi serangan ini tidak bisa digunakan untuk memecahkan keseluruhan 16 putaran Blowfish; Vaudenay menggunakan varian Blowfish dengan pengurangan jumlah putaran. Vincent Rijmen dalam thesis Ph.D-nya memperkenalkan metode *second-order differential atttack* yang bisa memecahkan 4 putaran blowfish, tidak lebih. Sampai sekarang belum ada cara yang diketahui yang bisa memecahkan keseluruhan 16 putaran Blowfish, selain *brute-force attack*.

4.4 Simplifikasi Blowfish

Untuk mengurangi kebutuhan memori dan waktu eksekusi, implementasi algoritma Blowfish bisa dilakukan penyederhanaan. Penyederhanaan yang mungkin dilakukan adalah sebagai berikut:

1. Pengurangan ukuran dan jumlah *S-box*.
   Jumlah *S-box* bisa dikurangi dari empat menjadi satu. Selain itu bisa juga dengan meng-*overlap* entri pada *S-box* tunggal: entri 0 mengandung byte 0 sampai 3, entri 1 mengandung byte 1 sampai 4, dll. Simplifikasi yang pertama akan mengurangi kebutuhan memori dari 4096 bytes menjadi 1024 bytes. Simplifikasi yang kedua akan mengurangi kebutuhan memori dari 1024 bytes menjadi 259 bytes. Beberapa langkah tambahan diperlukan untuk menghindari kesimetrian.

2. Pengurangan jumlah iterasi.
   Jumlah iterasi dapat dikurangi dari 16 putaran menjadi 8 putaran. Penentuan jumlah iterasi untuk keamanan data bergantung pada panjang kunci yang digunakan.

3. Kalkulasi subkunci secara *on-the-fly*.
   Untuk menyederhanakan kalkulasi subkunci, subkunci bisa dibangkitkan saling bebas (tidak saling bergantung) dengan subkunci yang lain.

5. Perbandingan Algoritma Blowfish dengan DES

Algoritma Blowfish mirip dengan algoritma DES. Algoritma blowfish adalah penyempurnaan dari algoritma DES. Blowfish lebih baik dari DES karena hal-hal berikut ini:

1. Kunci Blowfish bisa bervariasi dari 32 bit sampai 448 bit sedangkan DES hanya memiliki panjang kunci 64 bit (bahkan yang digunakan hanya 56 bit).
2. Subkunci Blowfish lebih banyak dari DES (Blowfish memiliki 18 subkunci sedangkan DES hanya 16) sehingga lebih sulit dipecahkan daripada DES.
3. Subkunci Blowfish benar-benar acak dan saling mempengaruhi antara satu subkunci dengan subkunci yang lain.
4. Algortima Blowfish selain aman juga sederhana sehingga mudah diimplementasikan, lain halnya dengan DES yang memiliki algoritma yang cukup rumit

6. Kesimpulan

1. Blowfish lebih baik daripada DES karena panjang kuncinya yang bervariasi dan subkunci yang benar-benar acak dan saling mempengaruhi antara subkunci yang satu dengan yang lain.
2. Blowfish memiliki desain yang sederhana sehingga mudah untuk diimplementasikan, selain itu algoritma ini tidak dipatenkan sehingga semua orang bisa memakainya.
3. Blowfish adalah algoritma *cipher* blok yang aman karena hingga saat ini belum ada metode serangan yang mampu memecahkan keseluruhan 16 putaran algoritma ini.

7. Daftar Pustaka

[1] Goots, Nik, Boris Isotov, Alex Moldovyan and Nik Moldovyan. Modern Cryptography: Protect your Data with Fast Block Cipher. A-LIST Publishing. 2003

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

[3] Schneier, Bruce. Applied Cryptography 2nd Edition. John Wiley & Sons, Inc. 1999

[4] http://www.herongyang.com/crypto/blowfish_cipher_1.html

[5] http://www.schneier.com/blowfish.html

[6] http://www.schneier.com/paper-blowfish-fse.html

[7] http://en.wikipedia.org/wiki/Blowfish_(cipher)

.