Studi dan Perbandingan Algoritma Blowfish dan Twofish

Adhitya Randy – NIM : 13503027

Laboratorium Ilmu dan Rekayasa Komputasi
Progam Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

E-mail: if13027@students.if.itb.ac.id

Abstrak

Tingginya kebutuhan informasi saat ini mendorong tumbuhnya metode pengamanan informasi. Seni penyandian dan pengamanan pesan yang disebut kriptografi ini berkembang sangat cepat saat ini. Penggunaan komputer dijital saat ini mendorong tumbuhnya algoritma kriptografi modern yang beroperasi dalam mode bit. Kriptografi modern terbagi menjadi algoritma kunci simetri dan kunci publik. Algoritma kunci simetri pun terbagi ke dalam *cipher* aliran dan *cipher* blok. Semua jenis algoritma ini digunakan dalam berbagai bidang aplikasi.

Blowfish dan Twofish merupakan algoritma *cipher* blok yang sangat luas dan banyak penggunaannya saat ini. Kedua algoritma ini dikembangkan oleh Bruce Scheneier. Twofish sendiri merupakan algoritma kriptografi yang dikembangkan dari Blowfish. Penggunaannya yang luas disebabkan kedua algoritma ini kuat terhadap serangan.

Studi dan perbandingan antara algoritma Blowfish dan Twofish dilakukan karena hubungan yang erat antara kedua algoritma tersebut. Studi dan perbandingan ini dilakukan dengan memandang kedua algoritma tersebut dari berbagai sisi. Sisi yang akan dibahas dalam studi ini adalah penjelasan algoritma, kriptanalisis terhadap algoritma, uji coba keamanan algoritma, dan penerapan algoritma.

***Kata kunci**: Blowfish, Twofish, perbandingan, kriptanalisis, keamanan, penerapan*

1 Pendahuluan

Perkembangan dunia informatika yang sangat pesat saat ini membawa pertumbuhan dunia ke dalam masa teknologi informasi menjadi ujung tombak kemajuan. Karena itulah nilai informasi saat ini sangat tinggi dan penting. Teknologi informasi yang telah terjalin saat ini berdiri di atas media komunikasi sebagai media penyampaian informasi dari suatu tempat ke tempat lainnya. Informasi-informasi yang ingin disampaikan berjalan melalui media komunikasi tersebut. Media komunikasi yang banyak digunakan tentu harus merupakan media yang mudah dijangkau dan digunakan oleh semua orang. Contoh media komunikasi yang saat ini sering digunakan adalah telepon dan jaringan internet.

Kemudahan pengaksesan media komunikasi oleh semua orang membawa dampak bagi keamanan informasi atau pesan yang menggunakan media komunikasi tersebut. Informasi menjadi sangat rentan untuk diketahui, diambil dan dimanipulasi oleh pihak-pihak yang tidak berkepentingan. Karena itulah dibutuhkan suatu metode yang dapat menjaga kerahasiaan informasi ini. Metode yang dimaksud adalah kriptografi yaitu sebuah seni dan bidang keilmuan dalam penyandian informasi atau pesan dengan tujuan menjaga keamanannya. Walaupun telah berkembang sejak zaman dahulu kala, teknik kriptografi yang dibutuhkan masa kini harus menyesuaikan dirinya terhadap meluasnya penggunaan komputer dijital pada masa kini.

Penggunaan komputer dijital mendorong berkembangnya kriptografi modern yang beroperasi dalam mode bit (satuan terkecil dalam dunia dijital) daripada dalam mode karakter yang biasa digunakan dalam kriptografi klasik. Akan tetapi, kriptografi modern tetap menggunakan prinsip substitusi dan transposisi yang sudah berkembang sejak kriptografi klasik. Algoritma kriptografi modern sendiri dapat dibagi ke dalam kelompok algoritma simetri dan kunci publik. Algoritma simetri merupakan algoritma kriptografi yang beroperasi dengan kunci enkripsi dan dekripsi yang sama. Sedangkan pada algoritma kunci publik, kunci yang digunakan untuk proses enkripsi dan dekripsi berbeda. Algoritma simetri yang beroperasi dalam mode bit dapat dikelompokkan menjadi dua kategori, yaitu *cipher* aliran dan *cipher* blok. *Cipher* aliran merupakan algoritma kriptografi yang beroperasi dalam bentuk bit tunggal. Sedangkan algoritma kriptografi kategori *cipher* blok beroperasi dalam bentuk blok bit.

Saat ini sudah banyak berkembang algoritma kriptografi kunci simetri baik untuk kategori *cipher* aliran maupun *cipher* blok. Di dalam dunia informatika dikenal algoritma Blowfish dan

Twofish. Kedua algoritma tersebut merupakan algoritma kriptografi kunci simetri kategori *cipher* blok yang banyak digunakan di dunia informatika hingga saati ini. Kedua algoritma ini beroperasi dalam bentuk blok bit, dengan ukuran blok sebesar 64 bit untuk Blowfish dan 128 bit untuk Twofish. Kunci yang digunakan dalam algoritma Blowfish sepanjang 32 sampai 388 bit. Sedangkan algoritma Twofish dapat bekerja dengan menerima panjang kunci yang beragam hingga sepanjang 128 bit.

Blowfish yang dirancang pada tahun 1993 ini ditujukan untuk menggantikan algoritma DES. Twofish yang dirancang lima tahun kemudian merupakan perkembangan dari algoritma Blowfish. Perancang kedua algoritma kriptografi ini adalah Bruce Schneier. Kedua algoritma tersebut menggunakan dua buah fitur yang terkenal yaitu kotak-s yang bergantung pada kunci dan penjadwalan kunci yang sangat kompleks. Kedua algoritma ini banyak digunakan karena merupakan algoritma kunci simetri yang kuat terhadap serangan. Hingga saat ini belum ditemukan metode kriptanalisis yang efektif untuk Blowfish maupun Twofish.

2 Dasar Teori *Cipher* Blok

Cipher blok merupakan algoritma kriptografi yang beroperasi dalam bentuk blok bit. Proses enkripsi dilakukan terhadap blok bit plainteks dengan mengguanakan kunci yang berukuran sama dengan ukuran blok plainteks. Algoritma ini akan menghasilkan cipherteks yang sama panjang dengan blok plainteks. Proses dekripsi terhadap cipherteks berlangsung dengan cara serupa seperti enkripsi. Hanya saja pada proses dekripsi, operasi berjalan kebalikan dari proses enkripsi.

Proses enkripsi dengan kunci K dinyatakan secara formal dengan persamaan

$$E_K(P) = C$$

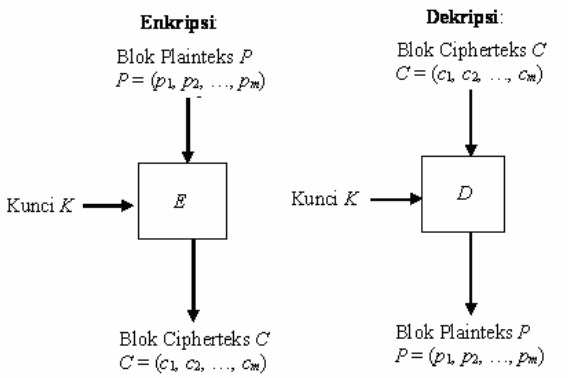
Sedangkan persamaan yang menyatakan proses dekripsi dengan kunci K adalah

$$D_K(C) = P$$

Fungsi E yang digunakan dalam proses enkripsi harus merupakan fungsi yang berkoresponden satu-ke-satu, sehingga

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

Skema enkripsi dan dekripsi cipher blok dapat dilihat pada Gambar 1.



Gambar 1 Skema Enkripsi dan Dekripsi *Cipher* Blok

Pada cipher blok plainteks dibagi menjadi beberapa blok dengan panjang tetap yaitu sepanjang kunci yang dimasukkan. Karena itulah terdapat kemungkinan panjang plainteks tidak habis dibagi dengan panjang ukuran blok yang ditetapkan atau panjang kunci. Hal ini mengakibatkan blok terakhir berukuran lebih pendek daripada ukuran blok-blok lainnya. Solusi permasalahan ini adalah dengan *padding*, yaitu dengan menambahkan blok terakhir dengan pola bit yang teratur hingga panjang blok sama dengan ukuran blok yang ditetapkan. Pola bit teratur ini misalnya ditambahkan bit 0 semua, atau bit 1 semua, atau bit 0 dan 1 secara bergantian. Setelah proses dekripsi hapus kembali *padding*, agar hasil dekripsi kembali menjadi plainteks.

Pada algoritma kriptografi yang beroperasi pada mode blok ini dikenal beberapa mode operasi. Empat mode operasi yang lazim digunakan pada cipher blok adalah:

1. *Electronic Code Book* (ECB)
2. *Cipher Block Chaining* (CBC)
3. *Cipher Feedback* (CFB)
4. *Output Feedback* (OFB)

Penggunaan mode-mode operasi tersebut tidak merubah fungsi enkripsi dan dekripsi yang telah didefinisikan.

2.1 Electronic Code Book (ECB)

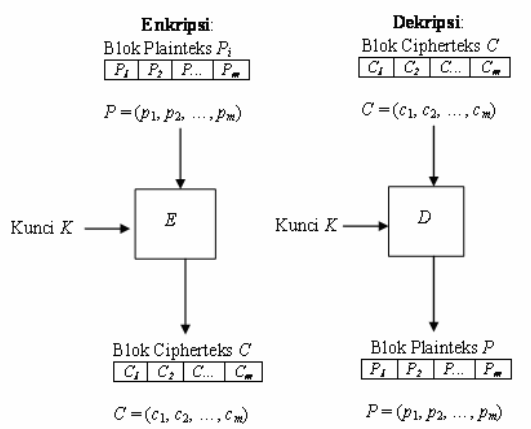
Pada mode ECB, setiap mode plainteks P_i dienkripsi secara individual dan independen menjadi blok cipherteks C_i. Secara matematis proses enkripsi dengan mode ECB dapat dinyatakan sebagai berikut:

$$Ci = E_K(Pi)$$

dan proses dekripsi sebagai berikut:

$$Pi = D_K(Ci)$$

Dalam hal ini, P_i dan C_i merupakan blok plainteks dan cipherteks ke-i. Skema enkripsi dan dekripsi dengan mode ECB dapat dilihat pada Gambar 2.



Gambar 2 Skema Enkripsi dan Dekripsi pada *Cipher* Blok dengan Mode ECB

Mode ini merupakan mode termudah dari keempat mode yang telah disebutkan di atas. Setiap blok plainteks dienkripsi secara independen, sehingga proses enkripsi tidak harus berlangsung secara linear atau proses enkripsi dapat dilakukan pada blok-blok secara tidak berurutan. Keuntungan mode ECB lainnya adalah kesalahan satu bit pada satu blok hanya akan mempengaruhi blok cipherteks yang berkoresponden pada proses dekripsi.

Tetapi, mode ECB lemah terhadap serangan. Dalam dunia nyata, kebanyakan bagian-bagian pesan cenderung akan muncul secara berulang di dalam sebuah teks. Dengan mode ECB, yang mengenkripsikan blok-blok secara independen, maka perulangan pesan pada plainteks mempunyai peluang yang besar untuk muncul berulang pula pada cipherteks. Dengan mengetahui informasi ini, kriptanalis dapat melakukan serangan dengan metode statistik secara mudah. Selain itu, mode ECB juga sangat rentan terhadap manipulasi cipherteks yang bertujuan untuk mengelabui pihak penerima pesan.

2.2 Cipher Block Chaining (CBC)

Pada mode CBC terdapat mekanisme umpan balik pada sebuah blok, yaitu blok plainteks *current* di-XOR-kan terlebih dahulu dengan dengan blok cipherteks hasil enkripsi sebelumnya. Selanjutnya hasil operasi XOR ini dimasukkan ke dalam fungsi enkripsi. Dengan demikian pada mode CBC, setiap blok cipherteks bergantung tidak hanya pada blok plainteksnya, tetapi juga pada seluruh blok plainteks sebelumnya. Dekripsi dilakukan dengan cara memasukkan blok cipherteks *current* ke dalam fungsi dekripsi, kemudian meng-XOR-kan hasilnya dengan blok cipherteks sebelumnya. Secara matematis proses enkripsi dapat dinyatakan sebagai berikut:

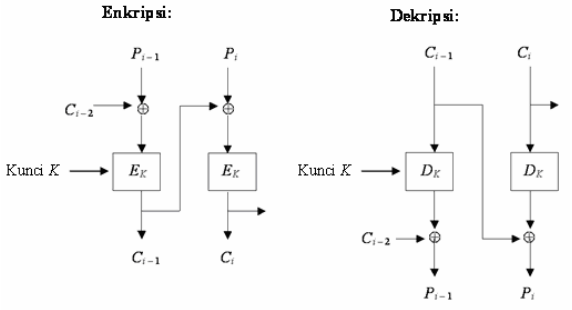
$$C_i=E_K(P_i \oplus C_{i-1})$$

sedangkan proses dekripsi dapat dinyatakan sebagai berikut:

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

Dalam hal ini C_0 merupakan IV (*Initialization Vactor*). IV dapat diberikan oleh pengguna atau dibangkitkan secara acak oleh aplikasi. IV ini merupakan rangkaian bit yang tidak bermakna dan hanya digunakan sebagai inisialisasi untuk membuat setiap blok cipherteks menjadi unik. Gambar 3 memperlihatkan skema enkripsi dan dekripsi dengan mode CBC.

Dengan mode CBC, kesalahan pada satu bit plainteks akan mempengaruhi blok cipherteks yang berkoresponden dan blok-blok cipherteks selanjutnya. Sedangkan kesalahan satu bit pada cipherteks hanya akan mempengaruhi satu blok plainteks yang berkoresponden dan satu bit pada blok berikutnya dengan posisi bit yang berkoresponden pula.



Gambar 3 Skema Enkripsi dan Dekripsi pada *Cipher* Blok dengan Mode CBC

Terlepas dari kekurangan yang telah disebutkan di atas, mode CBC dapat mengatasi kelemahan yang timbul pada mode ECB. Dengan mode ECB blok-blok plainteks yang sama tidak menghasilkan blok-blok cipherteks yang sama. Karena itulah kriptanalisis lebih sulit dilakukan pada mode CBC.

2.3 Cipher Feedback (CFB)

Mode CBC memiliki kelemahan yaitu proses enkripsi hanya dapat dilakukan pada ukuran blok yang utuh sehingga mode CBC tidak efisien jika diterapkan pada aplikasi komunikasi data. Permasalahan ini dapat diatasi pada mode CFB. Mode CFB mengenkripsikan data dalam unit yang lebih kecil daripada ukuran blok. Proses enkripsi pada unit yang lebih kecil daripada ukuran blok ini membuat mode CFB berlaku seperti *cipher* aliran. Karena hal inilah, mode CFB dapat diterapkan pada aplikasi komunikasi data.Unit yang dienkripsi dapat berupa bit per bit. Bila unit yang dienkripsi berupa satu karakter setiap kalinya, maka mode CFB ini disebut CFB 8-bit. Mode ini membutuhkan sebuah antrian yang berukuran sama dengan ukuran blok

masukan. Secara formal, proses enkripsi mode CFB *n*-bit dapat dinyatakan sebagai berikut:

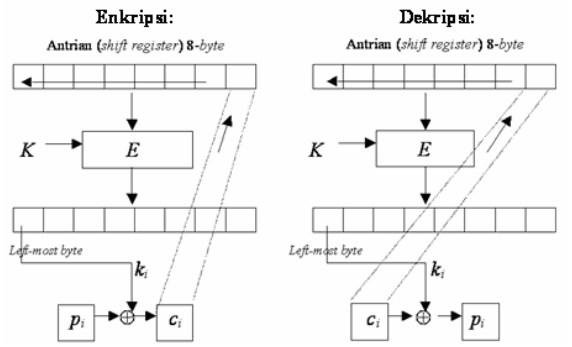
$$C_i = P_i \oplus MSB_m(E_K(X_i))$$
$$X_{i+1} = LSB_{m-n}(X_i) \,||\, C_i$$

sedangkan proses dekripsi dapat dinyatakan sebagai berikut:

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

Keterangan:
X_i = isi antrian dengan X_1 adalah IV
E = fungsi enkripsi
K = kunci
M = panjang blok enkripsi
N = panjang unit enkripsi
|| = operator penyambungan (*concatenation*)
MSB = Most Significant Byte
LSB = Least Significant Byte

Mode CFB mempunyai keunikan tersendiri, yaitu untuk proses enkripsi dan dekripsi digunakan fungsi yang sama. Skema enkripsi dan dekripsi dengan mode CFB 8-bit dapat dilihat pada Gambar 4.



Gambar 4 Skema Enkripsi dan Dekripsi pada *Cipher* Blok dengan Mode CFB 8-bit

Pada mode CFB pun terdapat perambatan kesalahan baik pada proses enkripsi maupun proses dekripsi. Pada proses enkripsi, kesalahan satu bit pada plainteks mempengaruhi blok cipherteks yang berkoresponden dan blok-blok cipherteks berikutnya. Sedangkan kesalahan satu bit pada blok cipherteks akan bit yang berkoresponden dan bit-bit lainnya selama bit *error* tersebut terdapat di dalam *shift register*. Pada mode CFB 8-bit dan ukuran blok 64 bit, maka kesalahan satu *byte* pada blok cipherteks akan mempengaruhi satu *byte* plainteks yang berkoresponden dan delapan *byte* berikutnya (lama *byte error* terdapat dalam *shift register* adalah delapan putaran).

2.4 Output Feedback (OFB)

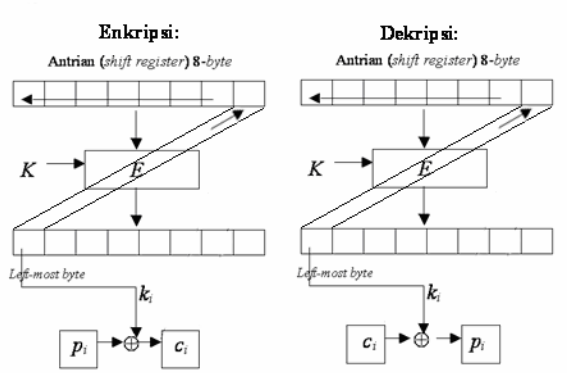
Mode OFB berkerja dengan cara yang mirip dengan mode CFB, kecuali n-bit dari hasil fungsi enkripsi terhadap antrian disalin menjadi elemen paling kanan antrian. Gambar 5 menunjukkan skema enkripsi dan dekripsi pada mode OFB 8-bit. Secara formal, proses enkripsi mode OFB *n*-bit dapat dinyatakan sebagai berikut:

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

sedangkan proses dekripsi dapat dinyatakan sebagai berikut:

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

Pada mode OFB tidak terdapat perambatan kesalahan. Kesalahan satu bit pada plainteks hanya mengakibatkan kesalahan satu bit yang berkoresponden pada cipherteks. Sebaliknya kesalahan satu bit pada cipherteks hanya mengakibatkan kesalahan satu bit yang berkoresponden pada plainteks.



Gambar 5 Skema Enkripsi dan Dekripsi pada *Cipher* Blok dengan Mode OFB 8-bit

3 Prinsip Perancangan *Cipher* Blok

Perancangan cipher blok harus memperhatikan beberapa prinsip perancangan yang telah biasa digunakan. Prinsip-prinsip yang dibahas pada upa-bab ini adalah prinsip cipher berulang (*iterated cipher*), jaringan Feistel (*Feistel network*), dan kotak-S (*S-box*).

3.1 Cipher Berulang (Iterated Cipher)

Cipher berulang merupakan fungsi transformasi sederhana yang mengubah plainteks menjadi cipherteks dengan proses perulangan sejumlah kali. Pada setiap putaran digunakan upa-kunci atau kunci putaran yang dikombinasikan dengan plainteks. Secara formal, *cipher* berulang dinyatakan sebagai berikut:

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

Keterangan:
i = 1, 2, ..., r (r adalah jumlah putaran)
K_i = upa-kunci (subkey) pada putaran ke-i

f = fungsi transformasi (di dalamnya terdapat fungsi substitusi, permutasi, dan/atau ekspansi, kompresi)

Plainteks dinyatakan dengan C_0 dan cipherteks dinyatakan dengan C_r.

3.2 Jaringan Feistel (Feistel Network)

Hampir semua algoritma cipher blok bekerja dalam model jaringan Feistel. Jaringan ini ditemukan oleh Horst Feistel pada tahun 1970. Secara formal, operasi transformasi pada jaringan *feistel* dapat dinyatakan sebagai:

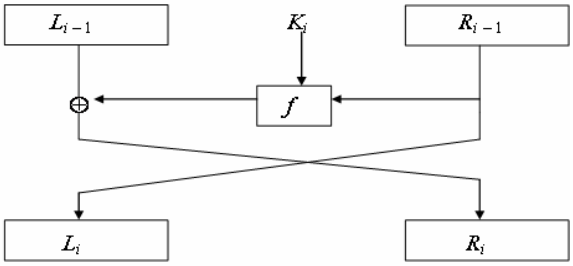
$$Li = Ri - 1$$
$$Ri = Li - 1 \oplus f(Ri - 1, Ki)$$

Proses enkripsi dan dekripsi dapat menggunakan model jaringan Feistel yang sama. Model jaringan *feistel* bersifat *reversible* untuk proses enkripsi dan dekripsi. Sifat *reversible* ini memungkinkan mendekripsi cipherteks menjadi plainteks tanpa membuat algoritma baru. Contoh:

$$Li - 1 \oplus f(Ri - 1, Ki) \oplus f(Ri - 1, Ki) = Li - 1$$

Selain itu, sifat *reversible* tidak bergantung pada fungsi f sehingga fungsi f dapat dibuat serumit mungkin.

Skema jaringan *feistel* dapat dilihat pada Gambar 6.



Gambar 6 Jaringan Feistel

3.3 Kotak-S (S-Box)

Kotak-S adalah suatu matriks sederhana 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 nirlanjad dalam algoritma, karena operasinya adalah *look-up table*. Masukan dari operasi *look-up table* dijadikan sebagai indeks kotak-S, dan keluarannya adalah *entry* di dalam kotak-S.

Contoh: Kotak-s di dalam algoritma DES adalah 6 x 4 *S-box* yang berarti memetakan 6 bit masukan menjadi 4 bit keluaran. Salah satu kotak-S yang terdapat di dalam algoritma DES dapat dilihat pada Tabel 1.

Tabel 1 Salah Satu Kotak-S pada Algoritma DES

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

Baris diberi nomor dari 0 sampai 3.
Kolom diberi nomor dari 0 sampai 15.

Masukan untuk proses substitusi adalh 6 bit, yaitu $b_1b_2b_3b_4b_5b_6$

Nomor baris dari tabel ditunjukkan oleh string bit b_1b_6 (menyatakan 0 sampai 3 desimal). Sedangkan nomor kolom ditunjukkan oleh string bit $b_2b_3b_4b_5$ (menyatakan 0 sampai 15 desimal).

Misalnya masukan adalah 110100
Nomor baris tabel = 10 (artinya baris 2 desimal)
Nomor kolom tabel = 1010 (artinya kolom 10 desimal)
Jadi, substitusi untuk 110100 adalah *entry* pada baris 2 dan kolom 10, yaitu 0100 (atau 4 desimal).

Perancangan kotak-s menjadi isu penting karena kotak-S harus dirancang sedemikian sehingga kekuatan kriptografinya bagus dan mudah diimplementasikan. Ada empat cara atau pendekatan yang dapat digunakan dalam pembuatan kotak-S. Keempat cara itu adalah:

1. Dipilih secara acak.
   Untuk kotak-S yang kecil, cara pengisian secara acak tidak aman, namun untuk kotak-s yang besar cara pemilihan acak ini cukup bagus untuk diterapkan.

2. Dipilih secara acak lalu diuji.
   Sama seperti cara nomor 1, namun nilai acak yang dibangkitkan diuji apakah memenuhi sifat tertentu.

3. Dibuat oleh orang (*man made*).
   Entry di dalam kotak-S dibangkitkan dengan teknik yang lebih intuitif. Perancang kotak-S mengisi kotak-S secara intuitif.

4. Dihitung secara matematis (*math made*).
   Entry di dalam kotak-S dibangkitkan berdasarkan prinsip matematika yang terbukti aman dari serangan kriptanalisis.

4 Algoritma Kriptografi Blowfish

4.1 Deskripsi Blowfish

Blowfish adalah sebuah algoritma kriptografi yang beroperasi pada mode blok. Algoritma Blowfish ditujukan untuk diimplementasikan pada sebuah mikroprosesor berskala besar. Algoritma Blowfish sendiri tidak dipatenkan sehingga dapat digunakan oleh orang banyak. Blowfish dirancang untuk memenuhi kriteria-kriteria sebagai berikut:

1. Cepat. Blowfish dirancang agar dapat mengenkripsikan data pada mikroprosesor 32 bit dengan kecepatan 26 *clock cycles per byte*.
2. Kompak. Blowfish dirancang agar dapat berjalan dengan penggunaan memori kurang dari 5kB.
3. Sederhana. Blowfish dirancang hanya menggunakan operasi-operasi sederhana. Operasi-operasi yang digunakan dalam Blowfish adalah penambahan, XOR, dan *table lookup* dalam operand 32 bit. Rancangan yang sederhana ini mempermudah proses analisa yang membuat Blowfish terhindar dari kesalahan implementasi.
4. Keamanan yang beragam. Panjang kunci yang digunakan dalam algoritma Blowfish bervariasi dengan panjang kunci maksimal 448 bit.

4.2 Algoritma Blowfish

Blowfish merupakan cipher blok yang beroprasi pada blok berukuran 64 bit dengan panjang kunci yang variatif. Alogoritma Blowfish terdiri dari dua bagian yaitu ekspansi kunci dan enkripsi data. Ekspansi kunci mengubah sebuah kunci dengan panjang maksimal 448 bit kepada beberapa *array* upa-kunci dengan ukuran total 4168 *byte*.

Enkripsi data terdiri dari sebuah fungsi sederhana yang mengalami putaran atau iterasi sebanyak 16 kali. Setiap putaran teridir dari sebuah permutasi yang bergantung pada kunci dan substitusi yang bergantung pada kunci-dan-data. Seluruh operasi berupa penambahan dan XOR dengan kata sepanjang 32 bit. Operasi tambahan yang digunakan hanya berupa *data lookup* terhadap *array* dengan empat indeks yang dilakukan setiap putaran.

Blowfish menggunakan sejumlah besar upa-kunci. Kunci-kunci tersebut harus dibangkitakan terlebih dahulu sebelum proses enkripsi dan dekripsi data dilakukan.

P-array terdiri dari 18 buat upa-kunci dengan ukuran 32 bit:

$$P_1, P_2, ..., P_{18}$$

Empat buat kotak-S dengan ukuran 32 bit mempunyai *entry* sebanyak 256 buah. Kotak-kotak tersebut adalah:

$$S_{1,0}, S_{1,1}, ..., S_{1,255}$$
$$S_{2,0}, S_{2,1}, ..., S_{2,255}$$
$$S_{3,0}, S_{3,1}, ..., S_{3,255}$$
$$S_{4,0}, S_{4,1}, ..., S_{4,255}$$

Upa-kunci dibangkitkan dengan menggunakan algoritma *Blowfish*. Metode pembangkitan upa-kunci ini dilakukan dengan langkah-langkah sebagai berikut:

1. Pertama-tama inisialisasi *P-array* dan kemudian keempat kotak-S, secara berurutan dengan sebuah string yang sama. String yang digunakan ini terdiri dari digit heksadesimal dari p.
2. Lakukan operasi XOR antara P_1 dengan 32 bit pertama dari kunci, lakukan operasi XOR antara P_2 dengan 32 bit kunci berikutnya, dan begitu seterusnya untuk semua bit pada kunci (hingga P_{18}). Ulangi langkah ini melalui perputaran bit-bit kunci hingga seluruh elemen pada *P-array* telah dilakukan operasi XOR dengan bit-bit kunci.
3. Lakukan proses enkripsi terhadap string dengan keseluruhan elemen nol dengan algoritma Blowfish, menggunakan upa-kunci yang dijelaskan pada langkah 1 dan 2.
4. Ubah nilai P_1 dan P_2 dengan hasil keluaran pada langkah 3.
5. Lakukan proses enkripsi terhadap hasil keluaran dari langkah 3 menggunakan algoritma Blowfish dengan upa-kunci yang telah dimodifikasi.
6. Ubah nilai P_3 dan P_4 dengan hasil keluaran pada langkah 5.
7. Lanjutkan proses mengubah semua elemen yang terdapat pada *P-array*, dan kemudian keempat kotak-S secara berurutan, dengan hasil keluaran algoritma Blowfish yang terus menerus berubah.

Secara keseluruhan terdapat 521 iterasi atau putaran yang dibutuhkan untuk membangkitkan seluruh upa-kunci yang dibutuhkan. Aplikasi kemudian dapat menyimpan upa-kunci yang telah dihasilkan. Proses pembangkitan kunci ini tidak perlu selalu dilakukan setiap saat.

Blowfish menggunakan jaringan Feistel yang terdiri dari 16 buah putaran. Skema jaringan Feistel pada

algoritma Blowfish dapat dilihat pada Gambar 7. Masukan terhadap jaringan Feistel ini adalah x, yang merupakan elemen data dengan ukuran 64 bit.

Proses enkripsi dilakukan dengan langkah-langkah sebagai berikut:

1. Bagi x menjadi setengah bagian, yaitu dengan ukuran 32 bit. Hasil pembagian ini adalah : x_L dan x_R.
2. Lakukan langkah-langkah berikut dalam 16 putaran:

$$x_L = x_L \oplus P_i$$
$$x_R = F(x_L) \oplus x_R$$
Tukar x_L dan x_R

Keterangan:
i = 1, 2, ..., 16 (menunjukkan nomor putaran)

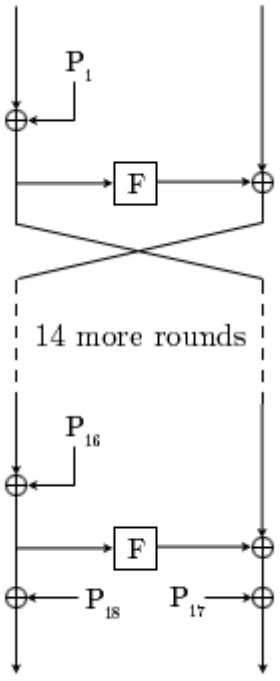
3. Tukar x_L dan x_R (membatalkan pertukaran terakhir).
4. $x_R = x_R \oplus P_{17}$
5. $x_L = x_L \oplus P_{18}$
6. Gabungkan kembali x_L dan x_R

Fungsi F yang terdapat pada jaringan Feistel difinisikan sebagai berikut:

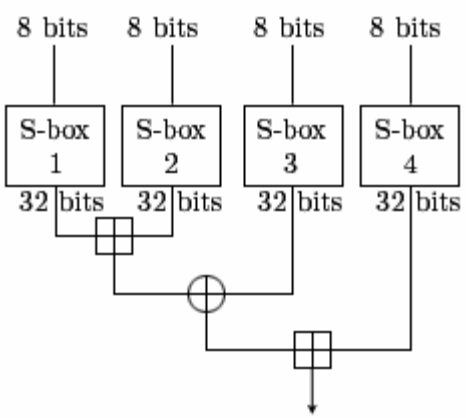
1. Bagi x_L menjadi empat bagian yang berukuran 8 bit. Keempat bagian yang dihasilkan adalah a, b,c, dan d.
2. Fungsi $F(x_L)$ didefinisikan sebagai berikut:

$$F(x_L) = ((S_{1,a} + S_{2,b} \bmod 2^{32}) \oplus S_{3,c}) + S_{4,d} \bmod 2^{32}$$

Skema fungsi F dapat dilihat pada Gambar 8.



Gambar 7 Skema Jaringan Feistel pada Algoritma Blowfish



Gambar 8 Skema Fungsi F pada Algoritma Blowfish

Proses dekripsi dilakukan dengan langkah yang sama dengan proses enkripsi, kecuali $P_1, P_2, ..., P_{18}$ digunakan dengan urutan terbalik dari proses enkripsi.

Implementasi algoritma Blowfish yang memerlukan waktu yang cepat harus mengurangi jumlah putaran dan memastikan bahwa semua upa-kunci tersimpan dalam *cache*.

4.3 Keamanan pada Algoritma Blowfish

Hingga saat ini telah terdapat beberapa proses kriptanalisis terhadap Blowfish. Walaupun begitu, kriptanalisis ini dilakukan terhadap algoritma Blowfish yang telah diperlemah atau disederhanakan.

John Kelsey mengembangkan sebuah metode serangan yang dapat memecahkan Blowfish dengan tiga putaran, tetapi tidak dapat mengembangkan lebih dari itu. Penyerangan ini mengeksploitasi fungsi F. Vikramjit Singh Chhabra juga telah mencari cara yang efisien untuk mengimplementasikan mesin pencarian kunci dengan cara lempang (*brute force*).

Serge Vaudenay melakukan pemeriksaan terhadap Blowfish dengan kotak-S diketahui dan putaran sebanyak r. Proses pemeriksaan yang dilakukan dengan serangan diferensial dapat menghasilkan *P-array* dengan 2^{8r+1} *chosen* plainteks [SCH95]. Untuk kunci lemah tertentu yang menghasilkan kotak-S yang buruk, serangan yang sama hanya membutuhkan 2^{4r+1} *chosen* plainteks untuk menghasilkan *P-array*. Kemungkinan untuk mendapatkan kunci lemah ini sendiri adalah 1 berbanding 2^{14}. Tanpa diketahuinya kotak-S yang digunakan, serangan ini dapat mendeteksi lemah tidaknya kunci yang digunakan, tetapi tidak dapat menentukan kunci, kotak-S, dan *P-array* yang digunakan. Cara penyerangan ini hanya efektif

terhadap aplikasi algoritma Blowfish dengan jumlah putaran yang dikurangi. Untuk algoritma Blowfish dengan jumlah putaran sebanyak 16 kali, cara serangan ini sama sekali tidak efektif dilakukan.

Penemuan kunci lemah sangatlah penting, tetapi kunci-kunci lemah ini mustahil untuk diketahui. Sebuah kunci lemah adalah kunci yang mempunyai dua *entry* yang sama untuk sebuah kotak-S. Tidak ada cara untuk memeriksa kunci lemah sebelum dilakukan ekspansi kunci. Pemeriksaan kunci lemah hanya dapat dilakukan ekspansi kunci terlebih dahulu. Tetapi cara ini tidak terlalu penting untuk dilakukan.

Di luar serangan diferensial yang telah disebutkan di atas, hingga saat ini belum ditemukan kriptanalisis yang dapat sukses dan efektif bekerja pada algoritma Blowfish.

4.4 Penggunaan Algoritma Blowfish

Blowfish adalah salah satu algoritma *cipher* blok yang tercepat dan digunakan secara luas di dunia, kecuali ketika pergantian kunci. Setiap kunci baru memerlukan pemrosesan awal yang sebanding dengan mengenkripsikan teks denga ukuran sekitar 4 kilobyte. Pemrosesan awal ini sangat lambat dibandingkan dengan algoritma *cipher* blok lainnya. Hal ini menyebabkan Blowfish tidak mungkin digunakan dalam beberapa aplikasi, tetapi tidak menimbulkan masalah dalam banyak aplikasi lainnya.

Pemrosesan awal yang lama pada Blowfish digunakan sebagai ide untuk metode *password-hasing* yang digunakan pada OpenBSD. Metode *password-hashing* ini menggunakan algoritma yang diuturnnkan dari algoritma Blowfish yang menggunakan penjadwalan kunci yang lambat. Algoritma ini digunakan dengan pertimbangan bahwa usaha komputasi ekstra yang harus dilakukan dapat memberikan proteksi lebih terhadap serangan terhadap *password* berbasiskan kamus (*dictionary attacks*).

Dalam beberapa implementasi, Blowfish memerlukan memori yang relatif besar, yaitu sekitar 4 kilobyte. Hal ini tidak menjadi masalah bahkan untuk komputer *desktop* dan *laptop* yang sudah berumur tua. Tetapi hal ini juga membuat implementasi Blowfish pada *embedded system* terkecil (seperti pada *smartcard* pada awal kemunculannya) tidak mungkin untuk dilakukan.

Penggunaan algoritma Blowfish antara lain terdapat pada:

1. 96Crypt, oleh fever.link.
   96Crypt merupakan aplikasi untuk enkripsi dan dekripsi arsip dan *folder*.
2. A-Lock, oleh Trillium Technology Group.
   A-Lock merupakan perangkat lunak enkripsi yang terintegrasi dengan aplikasi *e-mail* untuk Windows yang terkenal.
3. Access Manager, oleh Citi-Software Ltd.
   Aplikasi *password manager* untuk sistem operasi Windows.
4. AntiSnoop Password Dropper.
   Sebuah *password* basis data yang dirancang untuk melindungi terhadap *keystrokes loggers*, *mouse monitors*, dan sebagainya.
5. Canner, oleh Cinnabar Systems.
   Digunakan untuk melindungi kode Java dari *reverse engineering* dengan cara membuat *native Windows executable* yang mengandung kelas dan *resource* dan aplikasi dalam keadaan terenkripsi. Kelas dan *resource* ini kemudian akan didekripsikan pada memori ketika muncul permintaan dari mesin virtual Java.
6. Nautilus.
   Merupakan produk telepon berbasiskan internet yang aman. Produk ini dapat digunakan dan didapatkan secara gratis.
7. PuTTY, oleh Simon Tatham.
   Klien SSH untuk Win32. Aplikasi ini dapat digunakan dan didapatkan secara gratis.

5 Algoritma Kriptografi Twofish

5.1 Deskripsi Twofish

Twofish merupakan algoritma yang beroperasi dalam mode blok. Algoritma Twofish sendiri merupakan pengembangan dari algoritma Blowfish. Perancangan Twofish dilakukan dengan memperhatikan kriteria-kriteria yang diajukan National Institute of Standards and Technology (NIST) untuk kompetisi Advanced Encryption Standard (AES). Tujuan dari perancangan Twofish yang selaras dengan kriteria NIST untuk AES adalah sebagai berikut:

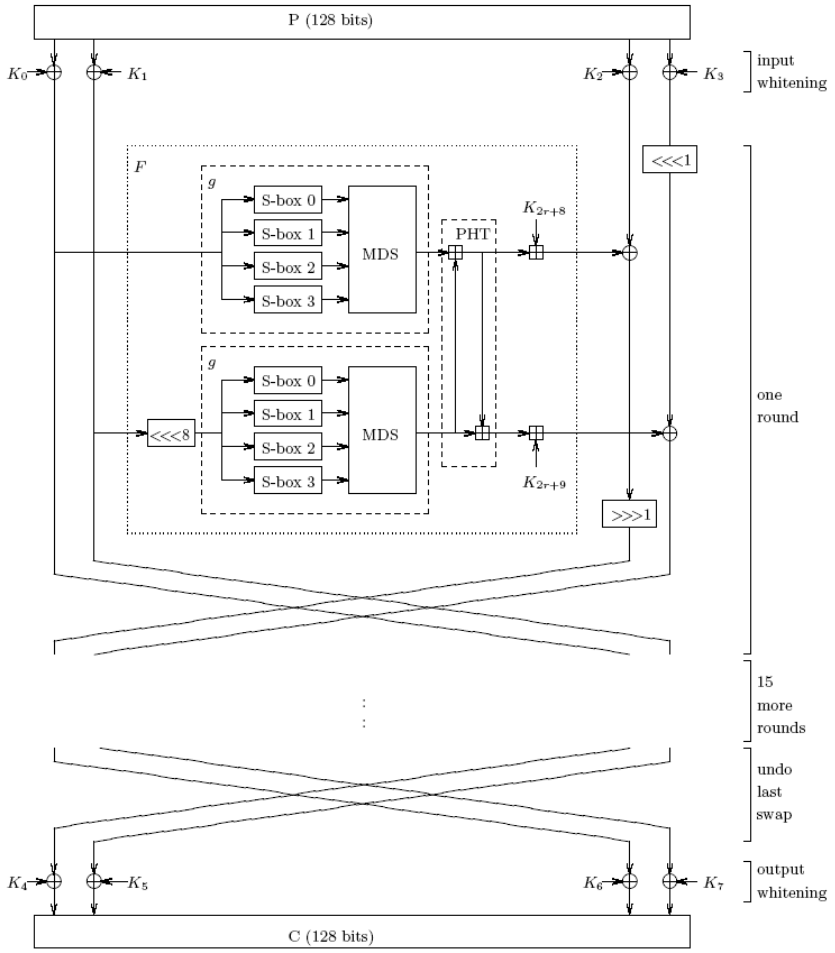
1. Merupakan *cipher* blok dengan kunci simetri dan blok sepanjang 128 bit.
2. Panjang kunci yang digunakan adalah 128 bit, 192 bit. Dan 256 bit.
3. Tidak mempunyai kunci lemah.
4. Efisiensi alogoritma, baik pada Intel Pentium Pro dan perangkat lunak lainnya dan *platform* perangkat keras.
5. Rancangan yang fleksibel. Rancangan yang fleksibel ini dapat diartikan misalnya dapat menerima panjang kunci tambahan, dapat diterapkan pada platform dan aplikasi yang sangat variatif, serta cocok untuk *cipher* aliran, fungsi hash, dan MAC.
6. Rancangan yang sederhana agar memudahkan proses analisis dan implementasi algoritma.

Selain kriteria-kriteria yang telah disebutkan di atas, pada Twofish ditambahkan kriteria performansi sebagai berikut:

1. Menerima kunci dengan panjang berapapun hingga 256 bit
2. Mengenkripsikan data dalam waktu kurang dari 500 *clock cycles* per blok pada Intel Pentium, Pentium Pro, dan Pentium II, untuk versi algoritma yang teroptimasi sepenuhnya.
3. Mampu untuk membentuk sebuah kunci 128 bit (untuk kecepatan enkripsi yang optimal) dalam waktu yang kurang dari waktu yang dibutuhkan untuk mengenkripsikan 32 blok pada Pentium, Pentium Pro, dan Pentium II.
4. Tidak menggunakan operasi-operasi yang membuat Twofish tidak efisien pada mikroprosesor selain 32 bit, mikroprosesor 8 bit, dan mikroprosesor 16 bit. Twofish juga tidak menggunakan operasi-operasi yang dapat mengurangi efisiensinya pada mikroprosesor 64 bit.

5.2 Algoritma Twofish

Twofish menggunakan struktur sejenis Feistel dalam 16 putaran dengan tambahan teknik *whitening* terhadap masukan dan hasil keluarannya. Teknik *whitening* sendiri adalah teknik melakukan operasi XOR terhadap materi kunci sebelum putaran pertama dan sesudah putaran akhir. Elemen di luar jaringan Feistel normal yang terdapat dalam algoritma Twofish adalah rotasi 1 bit. Proses rotasi ini dapat dipindahkan ke dalam fungsi F untuk membentuk struktur jaringan Feistel yang murni, tetapi hal ini membutuhkan tambahan rotasi kata sebelum langkah *whitening* hasil keluaran. Struktur algoritma Twofish dapat dilihat secara global pada Gambar 9.



Gambar 9 Struktur Algoritma Twofish Secara Global

Plainteks dibagi ke dalam empat bagian kata dengan ukuran 32 bit. Plainteks dengan ukuran 16 *byte* yaitu p_0, ..., p_{15} dibagi ke dalam empat bagian menjadi P_0, ..., P_3 dengan ukuran masing-masing 32 bit dengan menggunakan konversi *little endian*.

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

Dalam langkah *whitening* masukan, keempat bagian plainteks ini dilakukan operasi XOR dengan empat buat kata kunci dari kunci yang telah diekspansi.

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

Langkah ini kemudian diikuti dengan 16 putaran. Pada setiap putaran kedua kata yang pertama digunakan sebagai masukan untuk fungsi F, yang juga menerima masukan nomor putaran. Kata ketiga dilakukan operasi XOR dengan keluaran pertama fungsi F dan kemudian dirotasikan ke kanan sebanyak 1 bit. Kata keempat dirotasikan ke kanan sebanyak 1 bit dan kemudian dilakukan operasi XOR dengan keluaran kedua dari fungsi F. Kemudian tukarkan kedua bagian tersebut.

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

Keterangan:
r = 0, ..., 15
ROR dan ROL = fungsi yang merotasikan argumen pertamanya ke kanan atau kiri dengan jumlah bit sesuai dengan argumen keduanya

Langkap whitening hasil keluaran dilakukan dengan membatalkan proses penukaran pada putaran terakhir serta melakukan operasi XOR kata-kata data dengan empat kata dari kunci yang telah diekspansi.

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

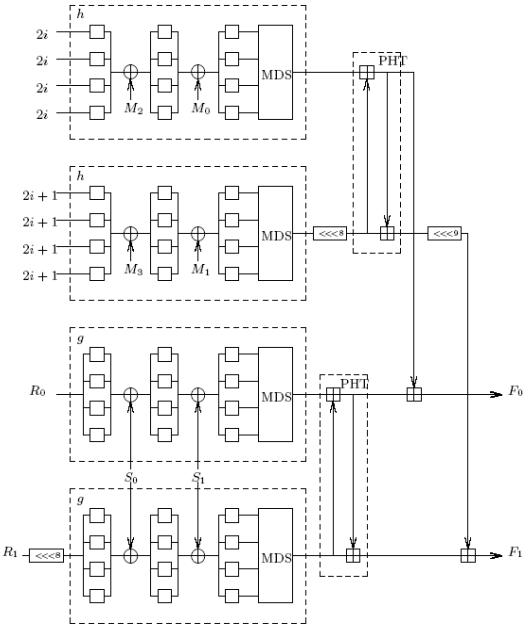
Empat buah kata cipherteks ini kemudian dituliskan sebagai 16 byte c0, ..., c15 dengan menggunakan konversi *little endian*, sama seperti yang digunakan pada plainteks.

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

Fungsi F adalah sebuah permutasi yang bergantung pada kunci dan beroperasi pada nilai 64 bit. Fungsi F meminta masukan tiga argumen, yaitu dua buah masukan kata R_0 dan R_1, dan nomor putaran r yang digunakan untuk memilih upa-kunci yang bersesuaian. R_0 kemudian dilewatkan kepada fungsi g yang menghasilkan T_0. R1 kemudian dirotasikan ke kiri sebanyak 8 bit dan kemudian dilewatkan kepada fungsi g untuk menghasilkan T_1. Hasil T_0 dan T_1 kemudian dikombinasikan di dalam Pseudo Hadamard Transform (PHT) dan dua buah kunci dari kunci yang telah diekspansi ditambahkan. Skema fungsi F dapat dilihat pada Gambar 10.

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

Keterangan:
F_0, F_1 = hasil keluaran dari fungi F



Gambar 10 Skema Fungsi F pada Algoritma Twofish

Fungsi g merupakan inti dari algoritma Twofish. Masukan kata X dibagi menjadi empat *byte*. Setiap byetiap byte dilewatkan kete dilewatkan ke kotak-S masing-masing yang bergantung dengan kunci. Setiap kotak-S bijektif, menerima masukan 8 bit dan menghasilkan keluaran 8 bit. Keempat hasil kemudian diinterpretasikan sebagai vektor dengan panjang 4 terhadap GF(2^8), dan kemudian dikalikan dengan 4 x 4 matriks MDS (menggunakan bidang GF(2^8) untuk proses komputasi). Vektor hasilnya kemudian diinterpretasikan sebagai kata berukuran 32 bit sebagai hasil fungsi g.

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

Keterangan:
s_i = kotak-S yang bergantung terhadap kunci
Z = hasil dari fungsi g

Penjadwalan kunci harus menyediakan 40 buah kata dari kunci yang sudah diekspansi K_0, ..., K_{39}, dan empat buah kotak-S yang bergantung terhadap kunci

untuk digunakan di dalam fungsi *g*. Twofish didefinisikan untuk menerima kunci dengan panjang 128 bit, 192 bit, dan 256 bit. Kunci yang lebih pendek dari 256 bit dapat digunakan dengan melakukan padding nol hingga mencapai panjang yang telah didefinisikan.

Kemudian didefinisikan $k = N/64$. Kunci M terdiri dari 8*k byte* $m_0, ..., m_{8k-1}$. *Byte-byte ini* pertama-tama dikonversikan menjadi kata-kata berukuran 2*k* dari setiap 32 bit.

$$M_i = \sum_{j=0}^{3} m_{(4i+j)} \cdot 2^{8j} \qquad i = 0, \ldots, 2k-1$$

Kemudian *byte-byte* tersebut juga dikonversikan menjadi dua vektor kata-kata dengan panjang *k*.

$$\begin{aligned} M_e &= (M_0, M_2, \ldots, M_{2k-2}) \\ M_o &= (M_1, M_3, \ldots, M_{2k-1}) \end{aligned}$$

Kata ketiga dari vektor dengan panjang *k* juga diturunkan dari kunci. Proses ini dilakukan dengan mengambil *byte-byte* kunci dalam kelompok yang berjumlah 8, kemudian interpretasikan kelompok-kelompok tersebut sebagai sebuah vektor melalui GF(2^8), dan mengalikannya dengan matriks berukuran 4 x 8 yang diturunkan dari sebuah kode RS. Setiap hasilnya yang berupa 4 *byte* ini kemudian diinterpretasikan sebagai satu kata berukuran 32 bit. Kata-kata ini membentuk vektor ketiga.

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

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

Keterangan:
$i = 0, ..., k-1$

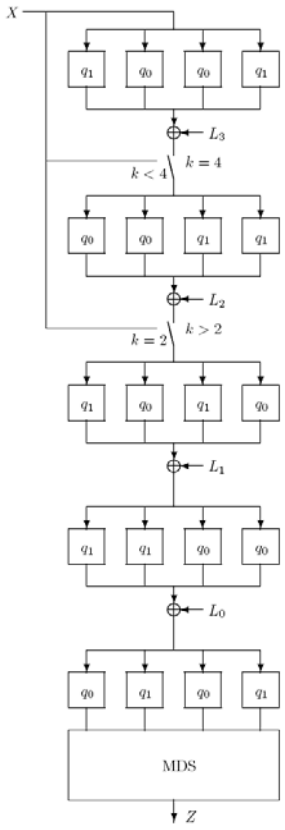
$$S = (S_{k-1}, S_{k-2}, \ldots, S_0)$$

Perhatikan bahwa urutan kata-kata pada *S* berada dalam keadaan urutan yang terbalik. Untuk perkalian matriks RS, GF(2^8) direpresentasikan dengan GF(2)[*x*]/*w*(*x*), dengan $w(x) = x^8 + x^6 + x^3 + x^2 + 1$ adalah sebuah primitih polinomial berderajat 8 terhadap GF(2). Pemetaan antara nialai byte dengan elemen-elemen dari GF(2^8) menggunakan definisi yang sama dari perkalian matriks MDS. Menggunakan pengertian pemetaan ini, maka matriks RS didefinisikan dengan:

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

Ketiga vektor Me, Mo, dan S membentuk basis untuk penjadwalan kunci.

Algoritma Twofish dapa menerima kunci dengan panjang berapapun hingga 256 bit. Untuk ukuran kunci yang tidak didefinisikan di atas, kunci ini mendapatkan *padding* pada akhir kunci dengan *byte* 0 sampai mencapai nilai kunci terdekat yang telah didefinisikan.



Gambar 11 Skema Fungsi *h*

Fungsi *h* adalah fungsi yang menerima dua buah masukan, yaitu satu kata 32 bit *X* dan sebuah *list* $L = (L_0, ..., L_{k-1})$ dari kata-kata 32 bit dengan panjang *k*, dan menghasilkan satu buah kata sebagai hasil keluaran. Skema fungsi *h* dapat dilihat pada Gambar 11. Fungsi ini bekerja dalam *k* tingkatan. Pada setiap tingkatan, setiap keempat *byte* dilewatkan kepada kotak-S statis, dan dilakukan operasi XOR dengan sebuah *byte* yang diturunkan dari *list*. Akhirnya, *byte-byte* tersebut kemdian sekali lagi dilewatkan kepada kotak-S statis, dan keempat byte tersebut dikalikan dengan matriks MDS seperti pada fungsi *g*. Secara formal, pembagian kata-kata ke dalam *byte* dedefinisikan sebagai berikut:

$$\begin{aligned} l_{i,j} &= \lfloor L_i / 2^{8j} \rfloor \bmod 2^8 \\ x_j &= \lfloor X / 2^{8j} \rfloor \bmod 2^8 \end{aligned}$$

Keterangan:
$i = 0, ..., k-1$
$j = 0, ..., 3$

Kemudian dilakukan urutan substitusi dan XOR diaplikasikan.

$$y_{k,j} = x_j \qquad j = 0, \ldots, 3$$

Jika $k = 4$, maka:

$$\begin{aligned} y_{3,0} &= q_1[y_{4,0}] \oplus l_{3,0} \\ y_{3,1} &= q_0[y_{4,1}] \oplus l_{3,1} \\ y_{3,2} &= q_0[y_{4,2}] \oplus l_{3,2} \\ y_{3,3} &= q_1[y_{4,3}] \oplus l_{3,3} \end{aligned}$$

Jika $k \geq 3$, maka:

$$\begin{aligned} y_{2,0} &= q_1[y_{3,0}] \oplus l_{2,0} \\ y_{2,1} &= q_1[y_{3,1}] \oplus l_{2,1} \\ y_{2,2} &= q_0[y_{3,2}] \oplus l_{2,2} \\ y_{2,3} &= q_0[y_{3,3}] \oplus l_{2,3} \end{aligned}$$

Untuk semua kasus berlaku:

$$\begin{aligned} y_0 &= q_1[q_0[q_0[y_{2,0}] \oplus l_{1,0}] \oplus l_{0,0}] \\ y_1 &= q_0[q_0[q_1[y_{2,1}] \oplus l_{1,1}] \oplus l_{0,1}] \\ y_2 &= q_1[q_1[q_0[y_{2,2}] \oplus l_{1,2}] \oplus l_{0,2}] \\ y_3 &= q_0[q_1[q_1[y_{2,3}] \oplus l_{1,3}] \oplus l_{0,3}] \end{aligned}$$

Dalam hal ini q_0 dan q_1 merupakan permutasi statis pada nilai 8 bit yang akan didefinisikan kemjudian. Hasil dari vektor y_i kemudian dikalikan dengan matriks MDS, seperti pada fungsi *g*.

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} . & \cdots & . \\ \vdots & \text{MDS} & \vdots \\ . & \cdots & . \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

$$Z = \sum_{i=0}^{3} z_i \cdot 2^{8i}$$

Dengan Z adalah hasil dari *h*.

Kotak-S pada Twofish merupakan kotak-S yang bergantung pada kunci. Kotak-S pada fungsi g dapat didefinisikan sebagai berikut:

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

Keterangan:
$i = 0, ..., 3$

Kotak-S s_i kemudian dibentuk dengan pemetaan dari x_i ke y_i pada fungsi *h*, dengan list *L* sama dengan vektor *S* yang diturunkan dari kunci.

Kata-kata hasil ekspansi kunci didefinisikan dengan menggunakan fungsi *h* sebagai berikut:

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

Konstanta ρ digunakan untuk menduplikasikan *byte-byte*. ρ mempunyai properti $i = 0, ..., 255$, kata $i\rho$ teridiri dari empat *byte* dengan ukuran sama, dengan nilai *i*. Fungsi *h* kemudian diaplikasikan ke kata-kata untuk tipe ini. Untuk A_i nilai *byte*-nya adalah $2i$, dan argumen kedua dari *h* adalah M_e. B_i dihitung dengan cara yang sama menggunakan $2i + 1$ sebagai nilai *byte* dan M_o sebagai argumen kedua, dengan tambahan operasi rotasi sebanyak 8 bit. Nilai dari A_i dan B_i kemudian dikombinasikan di dalam PHT. Salah satu hasilnya kemudian dirotasikan sebanyak 9 bit. Kedua hasil ini membentuk dua kata sebagai hasil ekspansi kunci.

Permutasi q_0 dan q_1 adalah permutasi statis dengan nilai 8 bit. Permutasi q_0 dan q_1 dibentuk dari empat buat permutasi 4 bit yang berbeda satu sama lain. Untuk nilai masukan *x*, dapat didefinisikan nilai keluaran *y* yang berkoresponden sebagai berikut:

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

ROR_4 adalah fungsi yang mirip dengan ROR yang merotasikan nilai 4 bit. Pertama-tama, *byte* dibagi manjadi dua bagian. Hasilnya kemudian dikombinasikan di dalam sebuah langkah pencampuran bijektif (*bijective mixing*). Setiap bagian kemudian dilewatkan kepada kotak-S statis 4-bit masing-masing. Proses ini diikuti dengan langkah pencampuran lainnya dan lookup *kotak*-S. Permutasi q_0 dengan kotak-S 4 bit didefinisikan sebagai berikut:

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

Setiap 4 bit pada kotak-S direpresentasikan dengan sebuah *list entry-entry* menggunakan notasi heksadesimal (*Entry* untuk masukan 0, 1, ..., 15 didaftarkan secara terurut). Permutasi q_1 dengan kotak-S 4 bit didefinisikan sebagai berikut:

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

5.3 Keamanan pada Algoritma Twofish

Studi mengenai keamanan dan kriptanalisis algoritma Twofish sudah banyak dilakukan. Walaupun begitu semua studi ini dilakukan terhadap

algoritma Twofish yang sudah disederhanakan atau diperlemah.

Untuk Twofish dengan lima putaran, tanpa proses *whitening* pada awal dan akhir proses dibutuhkan $2^{22,5}$ pasangan *chosen* plainteks dan 2^{51} usaha. Metode kriptanalisis untuk Twofish ini masih terus dikembangkan. Tetapi tidak ada metode kriptanalisis untuk Twofish dengan jumlah putaran di atas sembilan buah.

Selain dengan serangan *chosen* plainteks, telah dilakukan juga kriptanalisis dengan serangan *related-key*. Metode kriptanalisis yang dilakukan adalah serangan *chosen-key* parsial pada Twofish dengan 10 putaran tanpa proses *whitening* pada awal dan akhir proses. Untuk melakukan proses kriptanalisis ini harus disiapkan pasangan kunci-kunci yang berhubungan. Kemudian pilih 20 dari 32 *byte* dari setiap kunci. Dua puluh *byte* yang dipilih berada di bawah kendali kriptanalisis. Sedangkan dua belas *byte* sisanya tidak dikatahui, tetapi kriptanalisis dapat mengetahui bahwa keduanya adalah sama untuk kedua kunci. Proses yang harus dilakukan sebanyak 2^{64} *chosen* plainteks untuk setiap kunci yang dipilih dan dilakukan sekitar 2^{34} usaha, untuk mendapatkan 12 *byte* kunci yang belum diketahui.

Selain studi-studi di atas, dilakukan juga serangan terhadap algoritma yang telah dikurangi jumlah putarannya dengan penyederhanaan fitur-fitur tertentu, seperti: Twofish dengan kotak-S statis, Twofish tanpa rotasi 1 bit, dan yang lainnya. Tetapi hasil studi ini menunjukkan bahwa kriptanalisis tidak dapat memecahkan Twofish walaupun dengan penyederhanaan-penyederhanaan yang telah dilakukan [SCH01].

Studi mengenai serangan diferensial terhadap Twofish pun telah dilakukan. Walaupun begitu serangan diferensial ini sudah tidak efektif pada Twofish dengan tujuh putaran. Estimasi jumlah usaha yang harus dilakukan untuk memecahkan Twofish tujuh putaran dengan melakukan serangan diferensial adalah sebanyak 2^{131}.

Teknik serangan interpolasi sangatlah efektif jika digunakan untuk menyerang algoritma kriptografi yang bekerja dengan menggunakan fungsi-fungsi aljabar sederhana. Prinsip penyerangan yang dilakukan sangatlah sederhana, yaitu jika cipherteks dapat dirpresentasikan sebagai ekspresi polinomial atau rasional (dengan koefisien N) dari sebuah plainteks, maka ekspresi polinomial atau rasional tersebut dapat dibangun dari N pasangan plainteks dan cipherteks. Akan tetapi, serangan interpolasi sering hanya berhasil terhadap *cipher* dengan jumlah putaran yang sangat sedikit atau terhadap *cipher* dengan fungsi putaran dengan derajat aljabar yang rendah. Kotak-S pada Twofish mempunyai derajat aljabar yang cukup tinggi, dan kombinasi dari operasi-operasi dari kelompok aljabar yang berbeda-beda meningkatkan derajat tersebut semakin tinggi. Karena itulah, Twofish dipercaya aman dan kuat terhadap serangan interpolasi bahkan jika hanya menggunakan jumlah putaran yang sedikit.

Studi-studi di atas menunjukkan bahwa algoritma Twofish sangat kuat terhadap serangan, bahkan untuk versi Twofish yang sudah diperlemah. Untuk algoritma Twofish yang sebenarnya atau tidak diperlemah sama sekali belum ditemukan metode kriptanalisis atau serangan yang efisien untuk memecahkannya.

5.4 Penggunaan Algoritma Twofish

Implementasi algoritma Twofish harus memperhatikan kecepatan komputasi yang diinginkan. Twofish mempunyai karakteristik melakukan persiapan kunci dalam waktu yang lama. Karena itulah untuk menjami kecepatan, semua proses penjadwalan kunci dapat dilakukan terlebih dahulu dan disimpan. Penjadwalan kunci ini tidak harus selalu dilakukan. Dengan melakukan proses penjadwalan kunci di awal, proses enkripsi dan dekripsi Twofish dapat diselesaikan dengan sangat cepat. Tetapi, jika penerapan algoritma dilakukan dengan memori yang minimum, maka Twofish dapat melakukan penjadwalan kunci sejalan dengan berjalannya proses enkripsi dan dekripsi.

Perancangan algoritma Twofish yang benar memperhatikan domain penerapan aplikasi, membuat penerapan Twofish sangatlah luas. Twofish dapat dengan mudah diterapkan dalam berbagai mesin, bahkan untuk mesin dengan kapasitas memori yang minimal sekalipun.

Penggunaan algoritma Twofish antara lain terdapat pada:

1. Away32 Deluxe dan Away IDS Deluxe oleh BMC Engineering.
   Kedua aplikasi di atas merupakan perangkat lunak untuk enkripsi arsip dan *folder* pada Windows.
2. CleverCrypt oleh Quantum Digital Security.
   Perangkat lunak enkripsi drive virtual *on-the-fly* untuk Windows. Selain menggunakan algoritma Twofish, aplikasi ini juga menggunakan Rijndael dan Blowfish.
3. cryptcat oleh Farm9.
   Versi aplikasi netcat buatan L0pht dengan algoritma Twofish. Aplikasi ini memungkinkan pembangunan tunnel sederhana yang terenrkipsi antar mesin, melalui jaringan internet, dan dalam kasus-kasus tertentu dapat melewati *firewalls*. Aplikasi ini gratis dan *open source*.

4. DigiSecret oleh TamoSoft.
   Aplikasi berbasiskan Windows untuk membuat *archive* yang terenkripsi dan arsip *self-extracting EXE*, *shredding*, dan *file sharing* melalui internet.
5. FoxTrot oleh Roth Systems
   Sebuah *server* HTTP yang dirancang sebagai *server* aplikasi profesional. Dengan menggunakan aplikasi ini, pengguna dapat mengeksekusi perintah SQL langsung melalui *Address line* pada *browser*. Dengan mengguanakan algoritma Twofish, memungkinkan dilakukan transaksi yang aman dan terjaga.
6. Secure Information Courier, SecExMail, dan SecExFile oleh Bytefusion.
   Secure Information Courier memungkinkan pengunjung untuk mengunjungi *website* dengan mengirimkan *secure e-mail* kepada organisasi pengelolanya. SecExMail menyediakan proses enkripsi *e-mail* yang transparan, sedangkan SecExFile menyediakan fitur enkripsi dengan menekan satu buat tombol.
7. SecurePad oleh NeuralAbyss Software.
   Apliksi ini menawarkan alternatif dari Notepad dan Wordpad pada sistem operasi Windows. Aplikasi ini menggunakan Blowfish dan Twofish secara bersamaan dengan beberapa algoritma lainnya.

6 Perbandingan Blowfish dan Twofish

6.1 Kecepatan

Pada Tabel 2 dan

Tabel 3 berikut terdapat data hasil uji coba enkripsi dan dekripsi algoritma Blowfish dan Twofish dengan mode CFB. Percobaan pertama dikakukan dengan panjang kunci 10 karakter dan percobaan kedua dilakukan dengan panjang kunci 25 karakter. Percobaan ini dilakukan dengan menggunakan progam "Part I from Delphi Encryption Compendium."

Percobaan ini dilakukan untuk mengukur kecepatan proses enkripsi dan dekripsi dari kedua algoritma tersebut. Arsip-arsip dipilih dari jenis yang berbeda-beda. Hal terpenting dalam pemilihan arsip adalah terdapat perbedaan ukuran arsip. Sehingga dapat dilihat kecepatan proses enkripsi dan dekripsi dengan data yang berbeda-beda ukurannya. Ukuran kunci yang berbeda digunakan untuk mengetahui pengaruh panjang kunci terhadap kecepatan proses ini. Sedangkan mode yang dipilih CFB karena mode CFB berlangsung lebih lambat dibandingkan dengan mode-mode lainnya. Dengan lebih lambatnya mode CFB diharapkan perbedaan antara kedua algoritma dapat terlihat dengan lebih mudah. Selain itu kecepatan proses mode CFB dengan mode lainnya berbanding lurus. Sehingga pengambilan mode CFB dapat mewakili mode-mode lainnya.

Tabel 2 Hasil Percobaan Dengan Kunci 10 Karakter

No	Nama Arsip	Ukuran Arsip	Panjang Kunci	Waktu Enkripsi Mode CFB (ms)		Waktu Dekripsi Mode CFB (ms)	
				Blowfish	Twofish	Blowfish	Twofish
1	01 - Welcome To My Living Room.mp3	928 kB	10	254	733	259	733
2	flower.jpg	53,8 kB	10	16	43	17	44
3	bdg-map1.png	692 kB	10	191	544	198	543
4	Ceklist APBO.xls	26 kB	10	8	21	8	21
5	Drawing1.vsd	69 kB	10	20	54	20	56
6	Introduction To Multicasting.PDF	40,3 kB	10	12	32	13	33
7	putty.exe	412 kB	10	114	320	116	322
8	SDL-1.0-english-intro.zip	44,8 kB	10	13	36	13	36
9	test.txt	36 byte	10	1	1	1	1
10	Tugas-TA1.doc	64,5 kB	10	19	51	19	53
11	Colibri.mp3	5761 kB	10	1516	4986	1548	4882

Tabel 3 Hasil Percobaan Dengan Kunci 25 Karakter

No	Nama Arsip	Ukuran Arsip	Panjang Kunci	Waktu Enkripsi Mode CFB (ms)		Waktu Dekripsi Mode CFB (ms)	
				Blowfish	Twofish	Blowfish	Twofish
1	01 - Welcome To My Living Room.mp3	928 kB	25	205	748	204	730
2	flower.jpg	53,8 kB	25	13	43	13	44
3	bdg-map1.png	692 kB	25	153	544	153	547
4	Ceklist APBO.xls	26 kB	25	7	21	6	21
5	Drawing1.vsd	69 kB	25	16	56	16	56
6	Introduction To Multicasting.PDF	40,3 kB	25	10	33	10	52
7	putty.exe	412 kB	25	91	327	90	338
8	SDL-1.0-english-intro.zip	44,8 kB	25	11	36	11	36
9	test.txt	36 byte	25	1	1	1	1
10	Tugas-TA1.doc	64,5 kB	25	15	53	16	51
11	Colibri.mp3	5761 kB	25	1887	4903	1874	5066

Berdasarkan hasil percobaan terlihat bahwa proses enkripsi dan dekripsi algoritma yang sama, akan menghabiskan waktu yang kurang lebih sama juga. Walaupun begitu pada penggunaan algoritma Twofish, terkadang terdapat rentang waktu yang cukup besar antara proses enkripsi dan dekripsi. Misalnya pada data ke-1 dan ke-7 pada

Tabel 3. Dapat terlihat juga bahwa arsip dengan ukuran yang lebih besar akan diproses lebih lama dibandingkan dengan arsip yang berukuran lebih kecil. Sedangkan perbedaan panjang kunci tidak mempengaruhi waktu proses enkripsi dan dekripsi secara signifikan.

Berdasarkan keseluruhan data hasil eksperimen dapat ditarik kesimpulan bahwa algoritma Twofish selalu berjalan lebih lambat daripada algoritma Blowfish, kecuali untuk arsip yang berukuran sangat kecil.

6.2 Keamanan

Tingkat keamanan suatu algoritma kunci simetri tipe *cipher* blok dapat diukur dari tingkat kerumitan algoritma, panjang blok yang digunakan, panjang kunci yang digunakan, dan tingkat pengacakan plainteks terhadap cipherteks. Algoritma Blowfish dan Twofish menggunakan jaringan Feistel dan kotak-S dalam implementasinya. Karena itu tingkat keamanan algoritma ini juga dipengaruhi oleh cara penjadwalan kunci internal dan cara pembangkitan kotak-S.

Semakin tinggi tingkat kerumitan suatu algoritma maka algoritma tersebut semakin sulit dipecahkan. Semakin besar ukuran blok yang digunakan juga akan memepersulit penyerangan pada algoritma tersebut. Semakin besar ukuran blok yang digunakan akan mengakibatkan semakin jarangnya terdapat cipherteks berulang yang berasal dari plainteks yang sama. Hal ini menyebabkan hubungan antara plainteks dan cipherteks menjadi kabur, sehingga mempersulit kriptanalisis untuk melakukan penyerangan terhadap algoritma kriptografi yang digunakan. Ukuran panjang suatu kunci juga berpengaruh pada kekuatan algoritma. Biasanya semakin panjang dan acak suatu kunci akan mempersulit penyerangan algoritma kriptografi. Tingkat pengacakan plainteks terhadap cipherteks yang tinggi mengakibatkan sulitnya mencari hubungan antara plainteks dan cipherteks. Hal ini akan mempersulit penyerangan terhadap algoritma.

Khusus untuk algoritma Blowfish dan Twofish yang menggunakan jaringan Feistel dan kotak-S, terdapat beberapa hal tambahan yang dapat mempengaruhi kekuatan algoritma. Kekuatan jaringan Feistel pada dasarnya didukung oleh penjadwalan kunci internal yang sangat acak dan rumitnya fungsi f yang didefinisikan. Semakin acak hasil penjadwalan kunci internal yang dilakukan akan mempersulit kriptanalisis untuk melakukan penyerangan.

Kotak-S digunakan algoritma Blowfish dan Twofish dalam fungsi f pada jaringan Feistel. Cara pembangkitan kotak-S ini mempengaruhi tingkat kerumitan pada fungsi f tersebut. Ada dua macam pendekatan dalam pembangkitan kotak-S ini. Pertama adalah pembangkitan kotak-S secara statis.

Pembangkitan secara statis ini berarti kotak-S yang digunakan tidak bergantung pada plainteks dan kunci yang dimasukkan. Pendekatan kedua adalah pembangkitan kotak-S secara dinamis. Pembangkitan dinamis ini biasanya diimplementasikan dengan menggunakan fungsi bilangan acak. Pada algoritma Blowfish dan Twofish, kotak-S dibangkitkan berdasarkan kunci yang dimasukkan.

Tabel Tabel 4 berikut adalah perbandingan komponen-komponen keamanan suatu algoritma dalam satuan kulitatif.

Tabel 4 Perbandingan Komponen Keamanan Algoritma Blowfish dan Twofish

Komponen	Blowfish	Twofish
Algoritma	16 putaran; menggunakan jaringan Feistel dan kotak-S; menggunakan operasi penambahan, XOR, dan *lookup table*	16 putaran; menggunakan jaringan Feistel, kotak-S, matriks MDS, teknik Pseudo-Hadamard Transform (PHT), dan teknik *whitening*; menggunakan operasi penambahan, XOR, *lookup table*, aljabar vektor, aljabar polinom dan aljabar medan ;menggunakan fungsi dengan derajat aljabar yang tinggi (merupakan gabungan dari banyak kelompok aljabar)
Panjang blok	64 bit	128 bit
Panjang kunci	Bervariasi, panjang maksimal 448 bit	Bervariasi, panjang maksimal 256 bit
Tingkat pengacakan	Rumit, karena menggunakan jaringan Feistel 16 putaran dan 4 kotak-S yang bergantung pada kunci.	Sangat rumit, karena menggunakan jaringan Feistel 16 putaran, 4 kotak-S yang bergantung pada kunci, matriks MDS, teknik PHT, teknik *whitening*.
Penjadwalan kunci internal	Penjadwalan kunci internal bergantung pada kunci eksternal; cara pembangkitan dengan menggunakan operasi penambahan dan XOR	Penjadwalan kunci internal bergantung pada kunci eksternal; cara pembangkitan kunci dengan menggunakan operasi penambahan, XOR, aljabar vektor, aljabar polinom, dan matriks MDS
Pembangkitan kotak-S	Cara pembangkitan kotak-S dilakukan dengan operasi penambahan dan XOR; kotak-S bergantung pada kunci	Cara pembangkitan kotak-S dilakukan dengan operasi matriks MDS dan aljabar medan; kotak-S bergantung pada kunci

Berdasarkan tabel di atas, algoritma Twofish dirancang jauh lebih rumit dibandingkan dengan algoritma Blowfish. Algoritma Blowfish hanya lebih unggul dalam hal panjang kunci dibandingkan dengan algoritma Twofish. Dari perbandingan pada tabel di atas dapat ditarik kesimpulan bahwa rancangan Twofish jauh lebih aman dibandingkan dengan Blowfish.

6.3 Penggunaan

Pengguanaan kedua algoritma ini telah dibahas pada bagian 4.4 dan bagian 5.4. Contoh-contoh yang diberikan pada bagian tersebut hanyalah sebagian kecil dari keseluruhan penerapan algoritma Blowfish dan Twofish. Hal ini menunjukkan bahwa kedua algoritma ini dipakai secara luas di seluruh dunia. Walaupun algoritma Blowfish tergolong algoritma "tua", kekuatan algoritma ini masih digunakan untuk berbagai macam keperluan. Sedangkan algoritma Twofish yang jauh lebih kuat dan baru dibandingkan dengan Blowfish semakin banyak aplikasinya di dunia nyata.

7 Kesimpulan

Berdasarkan hasil studi dan percobaan dapat disimpulkan bahwa:

1. Algoritma Twofish jauh lebih kuat dibandingkan dengan Blowfish.
2. Sampai saat ini belum ditemukan metode kriptanalisis yang efisien untuk algoritma Twofish dan Blowfish yang tidak diperlemah.
3. Kedua algoritma ini masih banyak dipakai dalam aplikasi dunia nyata.
4. Algoritma Blowfish berjalan lebih cepat dibandingkan dengan Twofish.
5. Algoritma Blowfish dapat digunakan untuk aplikasi yang membutuhkan waktu enkripsi dan dekripsi yang cepat serta tingkat keamanan data tidak terlalu penting.
6. Algoritma Twofish dapat digunakan untuk aplikasi yang membutuhkan tingkat keamanan data yang sangat penting serta waktu enkripsi dan dekripsi tidak terlalu dipersoalkan.

8 Daftar Pustaka

[MUN06] Munir, Rinaldi. 2006. Diktat Kuliah IF5054 Kriptografi. Program Studi Teknik Informatika - Institut Teknologi Bandung.

[SCH93] Schneier, Bruce. 1993. Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish). Springer-Verlag.

[SCH95] Schneier, Bruce. 1995. The Blowfish Encryption Algorithm -- One Year Later. Dr. Dobb's Journal.

[SCH96] Schneier, Bruce. 1996. Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C. John Wiley and Sons.

[SCH98] Schneier, Bruce. 1998. Twofish: A 128-Bit Block Cipher.

[SCH99] Schneier, Bruce. 1999. Performance Comparison of the AES Submission.

[SCH01] Schneier, Bruce. 2001. The Twofish Encryption Algorithm - Block encryption for the 21st century. Dr. Dobb's Journal.