BAB 2

LANDASAN TEORI

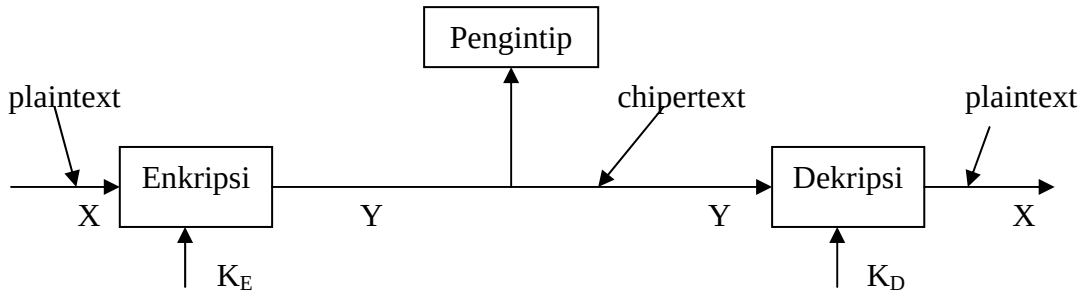
2.1 Konsep Dasar Kriptografi

Kriptografi (*cryptography*) berasal dari Bahasa Yunani, yaitu *cryptos* yang berarti *secret* atau rahasia dan *graphein* yang berarti *writing* atau tulisan. Jadi, kriptografi berarti tulisan rahasia. Pada zaman dulu, kriptografi memiliki definisi yaitu ilmu dan seni untuk menjaga kerahasiaan pesan dengan cara menyandikannya ke dalam bentuk yang tidak dapat dimengerti lagi maknanya. Definisi ini mungkin cocok pada masa lalu dimana kriptografi digunakan untuk keamanan komunikasi penting seperti komunikasi di kalangan militer, diplomat, dan mata-mata. Namun saat ini kriptografi lebih dari sekedar *privacy*, tetapi juga untuk tujuan *data integrity, authentication,* dan *non-repudiation.*

Pada saat ini, definisi kriptografi berkembang menjadi ilmu dan seni untuk menjaga kemanan pesan (*SCH96)*. Atau pada definisi yang lainnya kriptografi adalah ilmu yang mempelajari teknik-teknik matematika yang berhubungan dengan aspek keamanan informasi seperti kerahasiaan, integritas data, serta otentifikasi (*MEN96*). Kata “seni” pada definisi di atas berasal dari fakta sejarah bahwa pada masa-masa awal sejarah kriptografi, setiap orang mungkin mempunyai cara yang unik untuk merahasiakan pesan. Cara-cara unik tersebut mungkin berbeda-beda pada setiap pelaku kriptografi sehingga setiap cara menulis pesan rahasia, pesan mempunyai nilai estetika tersendiri sehingga kriptografi berkembang menjadi sebuah seni merahasiakan pesan. Pada perkembangan selanjutnya, kriptografi berkembang menjadi sebuah disiplin ilmu

sendiri karena teknik-teknik kriptografi dapat diformulasikan secara matematik sehingga menjadi sebuah metode yang formal.

Konsep dasar kriptografi dapat dilihat pada gambar berikut :



Gambar 2.1 Konsep Kriptografi secara umum

$Y = E_{K_E}(X)$ (enkripsi)

$X = D_{K_D}(Y)$ (dekripsi)

dimana: X = *plaintext*, Y = *chipertext*

K_E = *key* enkripsi, K_D = *key* dekripsi

Kriptografi sangat erat kaitannya dengan beberapa istilah berikut :

- Pesan, Plainteks, dan Cipherteks

  Pesan (*message*) adalah data atau informasi yang dapat dibaca dan dimengerti maknanya. Nama lain untuk pesan adalah plainteks (*plaintext*) atau teks jelas (*cleartext*). Pesan dapat berupa data atau informasi yang dikirim (melalui kurir, saluran telekomukasi, dsb) atau yang disimpan di dalam media perekaman (kertas, *storage*, dsb). Pesan yang tersimpan tidak hanya berupa teks, tetapi juga dapat

berbentuk citra (*image*), suara / bunyi (audio), dan video, serta berkas biner lainnya.

Agar pesan tidak dapat dimengerti maknanya oleh pihak lain, maka pesan perlu disandikan ke bentuk lain yang tidak dapat dipahami. Bentuk pesan yang tersandi disebut cipherteks (*ciphertext*) atau kriptogram (*cryptogram*). Cipherteks harus dapat ditransformasikan kembali menjadi plainteks semula agar pesan yang diterima bisa dibaca.

- Pengirim dan Penerima

  Komunikasi data melibatkan pertukaran pesan antara dua entitas. Pengirim (*sender*) adalah entitas yang mengirim pesan kepada entitas lainnya. Penerima (*receiver*) adalah entitas yang menerima pesan. Entitas di sini dapat berupa orang, mesin (komputer), kartu kredit, dan sebagainya. Jadi, orang bisa bertukar pesan orang lainnya (contoh : Alice berkomunikasi dengan Bob), sedangkan di dalam jaringan komputer, mesin (komputer) berkomunikasi dengan mesin (contoh : mesin ATM berkomunikasi dengan computer *server* di bank).

- Enkripsi dan Dekripsi

  Proses menyandikan plainteks menjadi cipherteks disebut enkripsi (*encryption*) atau *enciphering*. Sedangkan proses mengembalikan cipherteks menjadi plainteks semula dinamakan dekripsi (*decryption*) atau *deciphering*. Enkripsi dan dekripsi dapat diterapkan baik pada pesan yang dikirim maupun pada pesan tersimpan. Istilah *encryption of data in motion* mengacu pada enkripsi pesan yang ditransmisikan melalui saluran komunikasi, sedangkan *encryption of data at-rest* mengacu pada enkripsi dokumen yang disimpan di dalam *storage*. Contoh

encryption of data in motion adalah pengiriman nomor PIN dari mesin ATM ke momputer *server* di kantor bank pusat. Contoh *encryption of data at-rest* adalah enkripsi file basis data di dalam hard disk.

- Cipher dan Kunci

  Algoritma kriptografi disebut juga *cipher* yaitu aturan untuk enkripsi dan dekripsi, atau fungsi matematika yang digunakan untuk enkripsi dan dekripsi. Beberapa cipher memerlukan algoritma yang berbeda untuk enkripsi dan dekripsi. Sedangkan kunci / *key* adalah parameter yang digunakan untuk transformasi enkripsi dan dekripsi.

Istilah *cipher* sering disamakan dengan kode (*code*). Kode mempunyai sejarah tersendiri di dalam kriptografi. Sebenarnya kedua istilah ini tidak sama pengertiannya. Jika *cipher* adalah transformasi karakter ke karakter atau bit ke bit tanpa memperhatikan struktur bahasa pesan, maka kode sering diacu sebagai prosedur yang mengganti setiap plainteks dengan kata kode, misalnya :

kapal api datang dikodekan menjadi hutan bakau hancur

Kode juga dapat berupa deretan angka dan huruf yang tidak bermakna, seperti :

Kapal api datang dikodekan menjadi xyztvq bkugbf hjqpot

Keamanan algoritma kriptografi sering diukur dari banyaknya kerja (*work*) yang dibutuhkan untuk memecahkan cipherteks menjadi plainteksnya tanpa mengetahui kunci yang digunakan. Kerja ini dapat diekivalenkan dengan waktu, memori, uang, dan lain-lain. Semakin banyak kerja yang diperlukan, yang berarti juga semakin lama waktu yang

dibutuhkan, maka semakin kuat algoritma kriptografi tersebut, yang berarti semakin aman digunakan untuk menyandikan pesan.

Jika keamanan kriptografi ditentukan dengan menjaga kerahasiaan algoritmanya, maka algoritma kriptografinya dinamakan algoritma *restricted*. Algoritma *restricted* mempunyai sejarah tersendiri di dalam kriptografi. Algoritma *restricted* biasanya digunakan oleh sekelompok orang untuk bertukar pesan satu sama lain. Mereka membuat suatu algoritma enkripsi dan algoritma enkripsi tersebut hanya diketahui oleh anggota kelompok itu saja. Tetapi, algoritma *restricted* tidak cocok lagi saat ini, sebab setiap kali ada anggota kelompok keluar, maka algoritma kriptografi harus diganti lagi. Kriptografi modern mengatasi masalah ini dengan penggunaan kunci, yang dalam hal ini algoritma tidak lagi dirahasiakan, tetapi kunci harus dijaga kerahasiannya.

2.2 Konsep Matematis dalam Kriptografi

Konsep matematis yang mendasari algoritma kriptografi adalah relasi antara dua buah himpunan yaitu himpunan yang berisi elemen-elemen plainteks dan himpunan berisi cipherteks. Enkripsi dan dekripsi merupakan fungsi yang memetakan elemen-elemen antara kedua himpunan tersebut. Misalkan P menyatakan plainteks dan C menyatakan cipherteks, maka fungsi enkripsi E memetakan P ke C,

$$E(P) = \mathrm{C}$$

Dan fungsi dekripsi D memetakan C ke P,

$$D(C) = P$$

Karena proses enkripsi kemudian dekripsi mengembalikan pesan ke pesan asal, maka kesamaan berikut harus benar,

$$D(E(P)) = P$$

Dengan menggunakan kunci *K*, maka fungsi enkripsi dan dekripsi dapat ditulis sebagai,

$E_K(P) = C$ dan $D_K(C) = P$

Dan kedua fungsi ini memenuhi,

$D_K(E_K(P)) = P$

2.3 Tujuan Kriptografi

Dari penjelasan di atas, maka dapat disimpulkan bahwa tujuan dari kriptografi adalah sebagai berikut :

- Kerahasiaan (*confidentiality*)

  Adalah layanan yang ditujuan untuk menjaga agar pesan tidak dapat dibaca oleh pihak-pihak yang tidak berhak. Di dalam kriptografi, layanan ini direalisasikan dengan menyandikan pesan menjadi cipherteks. Misalnya pesan “Harap datang pukul 8” disandikan menjadi “TrxC#45motyptre!%”. Istilah lain yang senada dengan *confidentiality* adalan *secrecy* dan *privacy*.

- Integritas Data (*data integrity*)

  Adalah layanan yang menjamin bahwa pesan masih asli / utuh atau belum pernah dimanipulasi selama pengiriman. Dengan kata lain, aspek keamanan ini dapat diungkapkan sebagai pertanyaan : “Apakah pesan yang diterima masih asli atau tidak mengalami perubahan (modifikasi) ?”. Untuk menjaga integritas data, sistem harus memiliki kemampuan untuk mendeteksi manipulasi pesan oleh pihak-pihak yang tidak berhak, antara lain penyisipan, penghapusan, dan pensubstitusian data lain ke dalam pesan yang sebenarnya. Di dalam kriptografi, layanan ini

direalisasikan dengan menggunakan tanda tangan digital (*digital signature*). Pesan yang telah ditandatangani menyiratkan bahwa pesan yang dikirim adalah asli.

- Otentikasi (*authentication*)

  Adalah layanan yang berhubungan dengan identifikasi, baik mengidentifikasi kebenaran pihak-pihak yang berkomunikasi (*user authentication* atau *entity authentication*) maupun mengidentifikasi sumber pesan (*data origin authentication*). Dia pihak yang saling berkomunikasi harus dapat mengotentikasi satu sama lain sehingga ia dapat memastikan sumber pesan. Pesan yang dikirim melalui saluran komunikasi juga harus diotentikasi asalnya. Dengan kata lain, aspek keamanan ini dapat diungkapkan sebagai pertanyaan : “Apakah pesan yang diterima benar-benar berasal dari pengirim yang benar ?”. Otentikasi sumber pesan secara implicit juga memberikan kepastian integritas data, sebab jika pesan telah dimodifikasi berarti sumber pesan sudah tidak benar. Oleh karena itu, layanan integritas data selalu dikombinasikan dengan menggunakan tanda tangan digital (*digital signature*). Tanda tangan digital menyatakan sumber pesan.

- Nirpenyangkalan (*non repudiation*)

  Adalah layanan untuk mencegah entitas yang berkomunikasi melakukan penyangkalan, yaitu pengirim pesan menyangkal melakukan pengiriman atau penerima pesan menyangkal telah menerima pesan. Sebagai contoh misalkan pengirim pesan memberi otoritas kepada penerima pesan untuk melakukan pembelian, namun kemudian ia menyangkal telah memberikan otoritas tersebut. Contoh lainnya, misalkan seorang pemilik emas mengajukan tawaran kepada toko mas bahwa ia akan menjual emasnya. Tetapi, tiba-tiba harga emas turun drastis,

lalu ia membantah telah mengajukan tawaran menjual emas. Dalam hal ini, pihak toko emas perlu prosedur nirpenyangkalan untuk membuktikan bahwa pemilik emas telah melakukan kebohongan.

2.4 Algoritma Simetrik

Model enkripsi konvensional yang juga disebut algoritma simetrik. Proses enkripsi terdiri dari sebuah algoritma dan sebuah *key* (kunci). *Key* adalah sebuah nilai yang bebas dari *plaintext* yang mengontrol algoritma tersebut. Algoritma akan menghasilkan *output* yang berbeda tergantung pada *key* khusus yang digunakan pada waktu tersebut. Merubah *key* akan merubah *output* algoritma. Untuk menghasilkan kembali *plaintext* yang aslinya, kita menggunakan algoritma dekripsi dengan kunci yang sama dengan enkripsi. Pada gambar 2.1, K_E adalah sama dengan K_D.

2.5 Algoritma *Public-Key*

Algoritma *public-key* juga disebut algoritma *asymmetric* yang dirancang sehingga *key* yang digunakan untuk enkripsi berbeda dengan *key* yang digunakan untuk dekripsi. Selanjutnya *key* dekripsi tidak dapat dihitung dari *key* enkripsi. Algoritma tersebut disebut *public-key* karena *key* enkripsi dapat dibuat secara *public*. Orang asing dapat menggunakan *key* enkripsi tersebut untuk mengenkripsi sebuah *message*, tetapi hanya seorang tertentu dengan *key* dekripsi sepadan dapat mendekripsi *message* tersebut. Dalam sistem ini *key* enkripsi sering disebut *public key* dan *key* dekripsi disebut *private key*.

Enkripsi dengan *public key* (K) dinotasikan dengan :

$E_K(M) = C$

Dan didekripsi dengan *private key* dengan notasi sebagai berikut:

$D_K(C) = M$

Kadang-kadang *message* akan dienkripsi dengan *private key* dan didekripsi dengan *public key*, seperti yang digunakan dalam *digital signatures*.

2.6 Algoritma *Blowfish*

2.6.1 Enkripsi Algoritma *Blowfish*

Blowfish adalah *cipher* blok 64-bit yang memiliki sebuah kunci yang panjangnya variabel. Algoritma *Blowfish* terdiri dari dua bagian yaitu *key expansion* dan enkripsi data. Blok diagram enkripsi algoritma *Blowfish* dapat dilihat pada gambar 2.2.

Key expansion mengkonversikan sebuah kunci sampai 448 bit ke dalam beberapa *array subkey* dengan total 4168 byte.

Enkripsi data terdiri dari sebuah fungsi yang sederhana dengan iterasi 16 kali. Setiap *round* mempunyai sebuah permutasi *key-dependent* dan sebuah subsitusi *key* dan *data-dependent*. Semua operasi, penjumlahan dan XOR pada word 32-bit. Hanya operasi tambahan di indeks empat yaitu *lookup data array per round*.

Blowfish menggunakan sejumlah *subkey* yang besar. *Key* ini harus dihitung awal sebelum enkripsi atau dekripsi.

P-*array* mempunyai 18 *subkey* 32-bit :

$P_1, P_2, P_3, \ldots\ldots\ldots\ldots\ldots\ldots P_{18}$

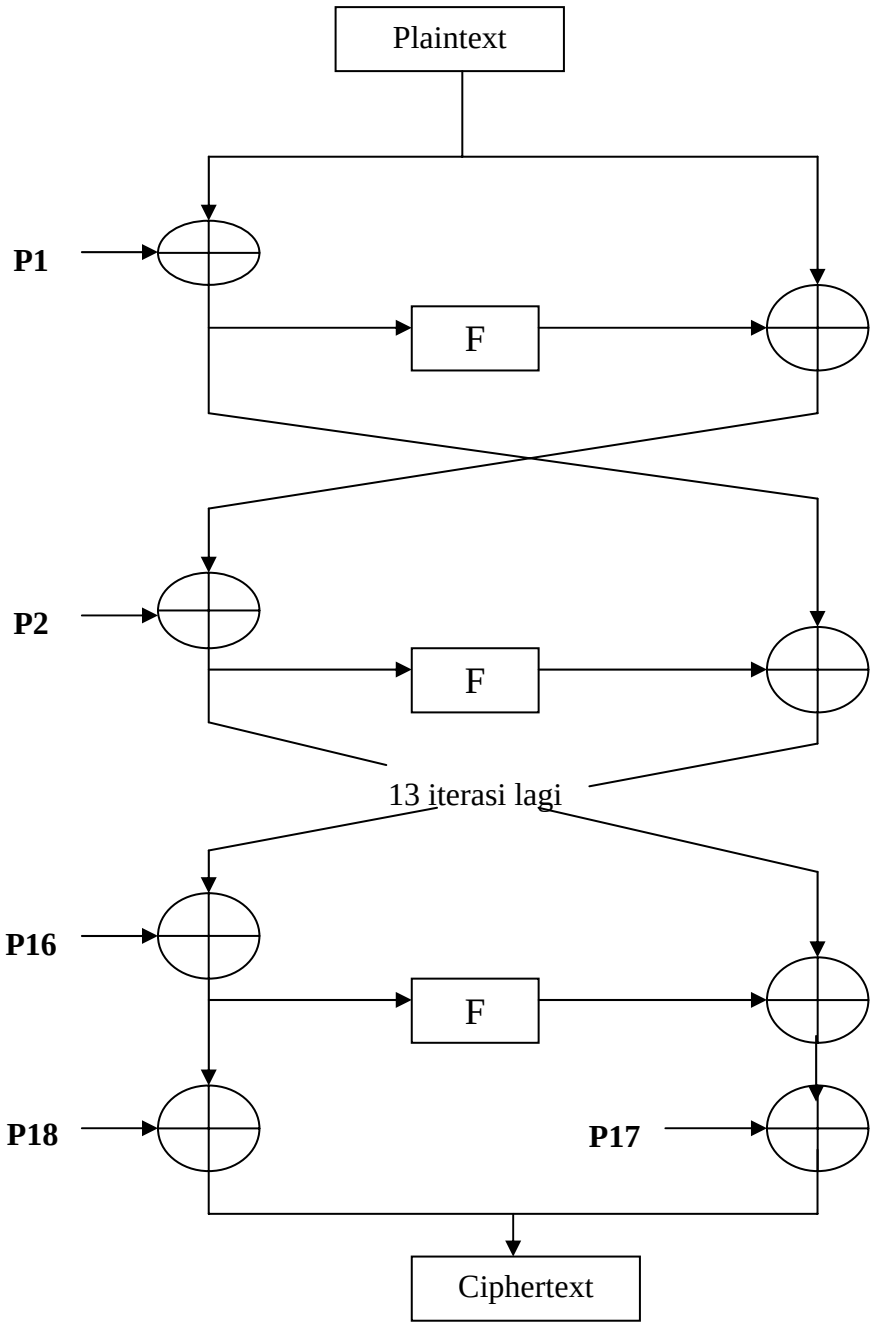
Empat *S-box* 32- bit mempunyai masing-masing 256 *entry* yaitu :

$S_{1,0}, S_{1,1}, S_{1,2}, S_{1,3}, \ldots\ldots\ldots\ldots\ldots S_{1,255}$

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

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

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



Gambar 2.2 Blok diagram Algoritma Enkripsi *Blowfish*

Blowfish adalah sebuah jaringan *Feistel* yang mempunyai 16 *round*. Inputnya adalah (x) element data 64-bit. Untuk mengenkripsi (x) yaitu :

Bagi (x) dalam dua bagian 32-bit menghasilkan (x_L) dan (x_R).

Untuk i = 1 sampai 16 maka :

$x_L = x_L \oplus P_i$

$x_R = F(x_L) \oplus x_R$

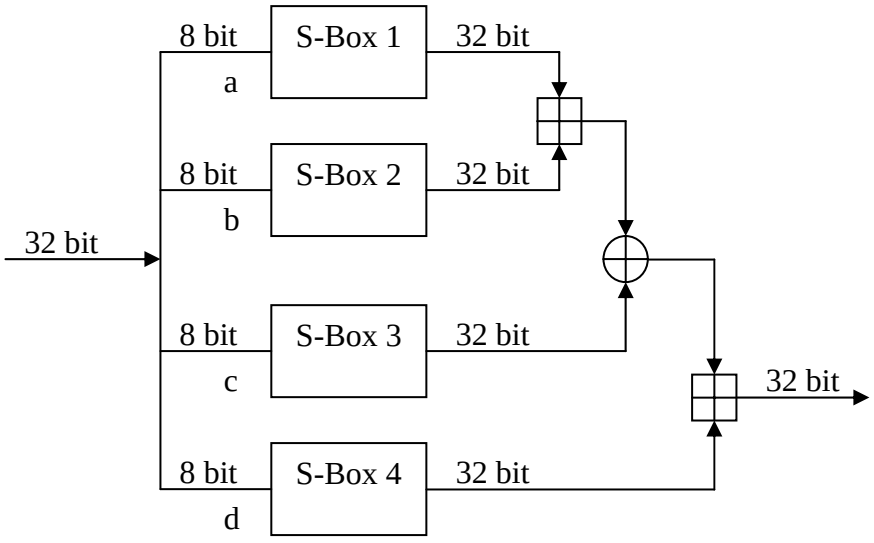
Swap (tukar) x_L dan x_R

Swap (tukar) x_L dan x_R (mengulang *swap* yang lalu)

$x_R = x_R \oplus P_{17}$

$x_L = x_L \oplus P_{18}$

Gabungkan kembali x_L dan x_R



Gambar 2.3 Fungsi F (Bruce Schneier: 1996)

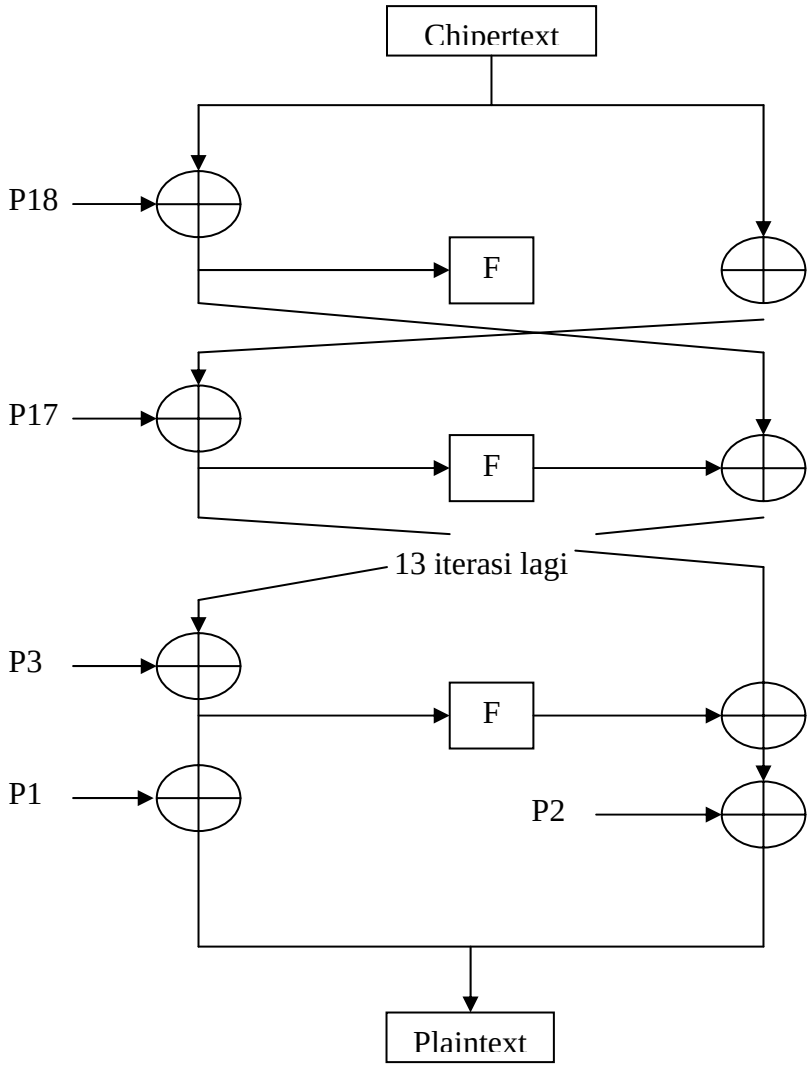
Fungsi F di atas adalah sebagai berikut :

Bagi x_L dalam empat kuarter 8-bit yaitu a, b, c dan d seperti gambar 2.3 maka :

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

2.6.2 Dekripsi Algoritma *Blowfish*

Dekripsi sama persis dengan enkripsi, kecuali bahwa P_1, P_2,…, P_{18} digunakan pada urutan yang berbalik (*reverse*). Blok diagram dekripsi seperti pada gambar 2.4.



Gambar 2.4 Blok Diagram dekripsi *Blowfish*

Dengan membalikkan 18 *subkey* untuk medekripsi metode algoritma *Blowfish*. Pertama, masalah ini nampak tidak dapat dipercaya, karena ada dua XOR operasi yang mengikuti pemakaian f-fungsi yang sebelumnya, dan hanya satu yang sebelumnya pemakaian pertama f-fungsi. Meskipun jika kita memodifikasi algoritma tersebut sehingga pemakaian *subkey* 2 sampai 17 menempatkan sebelum *output* f-fungsi yang di-XOR-kan ke sebelah kanan blok dan dilakukan ke data yang sama sebelum XOR itu, walaupun itu berarti ia sekarang berada di sebelah kanan blok, karena XOR *subkey* tersebut telah dipindahkan sebelum *swap* (tukar) kedua belah blok tersebut (tukar separuh blok kiri dan separuh blok kanan). Kita tidak merubah suatu apapun karena informasi yang sama di-XOR-kan ke separuh blok kiri antara setiap waktu, informasi ini digunakan sebagai *input* f-fungsi. Kenyataannya, kita mempunyai kebalikan yang pasti dari barisan dekripsi.

2.7 Jaringan *Feistel*

Banyak algoritma menggunakan jaringan *Feistel*. Ide ini muncul pada tahun 1970. Bila kita ambil sebuah blok dengan panjang n dan bagilah blok tersebut dengan dua bagian sepanjang n/2 yaitu bagian L (kiri) dan bagian R (kanan). Tentu saja panjang n adalah genap. Kita dapat mendefinisikan sebuah cipher blok iterasi dimana *output round* ke-I ditentukan dari *output round* sebelumnya :

$$L_i = R_{i-1}$$

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

K_i adalah *subkey* yang digunakan dalam *round* ke-I dan f adalah fungsi *round arbitrary*.

2.8 Bidang – Bidang Aplikasi

Suatu standar algoritma enkripsi harus cocok pada bidang-bidang beberapa aplikasi yaitu:

- Enkripsi Bulk

  Algoritma tersebut harus efisien dalam mengenkripsi file data atau *stream* data kontinyu.

- Generation Bit Acak (*random bit generation*)

  Algoritma tersebut harus efisien dalam memproduksi singel bit acak

- Enkripsi Paket

  Algoritma tersebut harus efisien dalam mengenkripsi *data packet-size*. (Sebuah paket ATM mempunyai data field 48-byte). Ia harus dapat diimplimentasikan dalam sebuah aplikasi dimana paket berurut akan dienkripsi atau didekripsi dengan kunci yang berbeda.

- Hashing

  Algoritma tersebut harus efisien dalam dikonversikan ke fungsi *hash one-way*.

2.9 *Platform*

Sebuah algoritma enkripsi standar harus dapat diimplimentasikan pada bermacam-macam platform yang berbeda, yang masing-masing mempunyai syarat-syaratnya sendiri. Ini termasuk :

- *Hardware* Khusus

  Algoritma tersebut dapat diimplimentasikan secara efisien dalam *hardware* VLSI sesuai dengan keinginan (*custom VLSI hadware*).

- Prosesor Yang Besar

  Sementara *hardware* yang terdedikasi akan selalu digunakan untuk aplikasi yang tercepat, sedangkan implimentasi *software* adalah lebih umum (*common*). Algoritma tersebut harus efisien pada mikroprosesor 32-bit yang mempunyai 4 k byte program dan *cache data*.

- Prosesor *Medium-size*

  Algoritma tersebut harus berjalan pada mikrocontroler dan prosesor *medium-size* yang lain, seperti 68HC11

- Prosesor Kecil

  Ia harus memungkinkan untuk mengimplimentasikan algoritma pada *smart card*, bahkan tidak efisien

Syarat-syarat untuk prosesor yang kecil adalah lebih susah. Batas RAM dan ROM adalah beberapa bagi *platform* ini. Juga, efisiensi lebih penting pada mesin kecil ini. *Workstation* mengandakan kapasitasnya hampir setiap tahun. *Embedded system* yang kecil adalah sama tahun ke tahun, dan ada sedikit kapasitas untuk *spare*. Jika ada suatu pilihan, maka burden komputasi extra haruslah pada prosesor yang besar lebih baik daripada yang kecil.

2.10 Membangkitkan *Sub-Key* (*Generating the Subkeys*)

Subkey dihitung menggunakan algoritma *Blowfish*. Metode yang pasti sebagai berikut :

1. Pertama inisial P-array dan kemudian S-boxes, agar mempunyai string yang tetap. String ini terdiri dari digit hexadesimal pi (kurang inisial 3). Contoh :

   P1 = 0x243f6a88

   P2 = 0x85a308d3

   P3 = 0x13198a2e

   P4 = 0x03707344

2. XOR P1 dengan 32 bit key pertama , XOR P2 dengan 32- bit key yang kedua dan seterusnya untuk semua bit key (mungkin sampai P14). Siklus yang berulang melalui key bit sampai semua P-array yang telah di-XOR-kan dengan key bit. (Untuk setiap key yang pendek, maka ada paling sedikit satu key ekivalen yang panjang; contoh jika A adalah 64-bit key maka kemudian AA, AAA, dan lain-lain adalah key yang ekivalen).
3. Enkripsikan string all-zero dengan algoritma *Blowfish* menggunakan subkey yang dijelaskan pada langkah 1 dan langkah 2.
4. Gantikan P1 dan P2 dengan output langkah 3.
5. Enkripsikan output langkah 3 menggunakan algoritma *Blowfish* dengan subkey yang termodifikasi.
6. Gantikan P3 dan P4 dengan output langkah 5.
7. Teruskan proses tersebut, gantikan semua entry P-array, dan kemudian semua empat S-boxes supaya mempunyai output algoritma *Blowfish* secara kontinyu berubah (continuously-changing).

Total iterasi 521 dibutuhkan untuk membangkitkan semua subkey yang diinginkan dari pada mengeksekusi proses turunan ini beberapa kali.

2.11 Kriteria Rancangan Algoritma *Blowfish*

Blowfish dirancang oleh Bruce Schneier dengan kriteria rancangan sebagai berikut:

1. Cepat. *Blowfish* mengenkripsi data pada 32-bit microprosesor dengan rate 26 clock siklus per byte.
2. Tersusun rapi (*compact*). *Blowfish* dapat running kurang lebih 5 K memory.
3. Sederhana (*simple*). *Blowfish* hanya menggunakan operasi yang sederhana yaitu penjumlahan, XOR dan tabel dalam mencari operand 32-bit. Rancangannya mudah dianalisa yang membuatnya tahan terhadapat kesalahan (*errors*) implimentasi.
4. Keamanannya variabel. Panjang *key* (kunci) adalah variabel dan dapat menjadi sepanjang 448 bit.

Blowfish dioptimasi bagi aplikasi-aplikasi dimana kunci tersebut tidak sering berubah, seperti jaringan komunikasi atau sebuah *encriptor* file otomatis. *Blowfish* ini siknifikan lebih cepat dari pada DES bila implimentasi pada 32-bit mikroprosesor yang mempunyai cache data yang besar, seperti Pentium dan PowerPC. *Blowfish* tidak cocok untuk aplikasi seperti paket *switching* yang kuncinya sering berubah, atau sebagai sebuah fungsi *hash one-way*. Syarat memorinya besar yang membuatnya tidak mungkin untuk aplikasi *smart card*.