BAB 2
TINJAUAN PUSTAKA

2.1 Kriptografi

Kriptografi merupakan ilmu sekaligus seni untuk menjaga kerahasiaan pesan (data atau informasi) dengan cara menyamarkannya (*to crypt*) menjadi bentuk tersandi yang tidak bermakna [2].

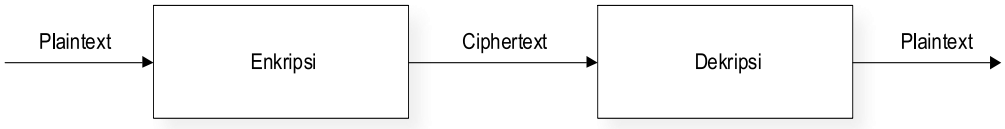
Pesan yang dirahasiakan dinamakan plainteks, sedangkan pesan hasil penyamaran dinamakan chiperteks. Proses penyamaran dari plainteks ke chiperteks disebut enkripsi (*encryption*) dan proses pembalikan dari chiperteks ke plainteks disebut dekripsi (*decryption*) [2].

2.1.1 Teori DasarKriptografi

Setiap hari aktivitas *surfing* di *internet*, *login* untuk melihat *email* atau *chatting* melalui jaringan *internet* ternyata dapat ditembus dengan menggunakan piranti lunak semacam *Packet Sniffer*. Hal ini terjadi bukan hanya komunikasi sipil yang disadap tetapi juga komunikasi militer dengan resiko yang akan terjadi adalah data-data rahasia negara menjadi bocor ke tangan negara lain. Untuk mencegah hal itu terjadi, diperlukan proses penyandian data yang dikenal dengan enkripsi (*encryption*) yang merupakan implementasi dari teknik kriptografi (*Criptography*) [6].

Kriptografi adalah suatu ilmu yang mempelajari bagaimana cara menjaga agar data atau pesan tetap aman saat dikirimkan, dari pengirim ke penerima tanpa mengalami gangguan dari pihak ketiga [6]. Menurut *Bruce Scheiner* dalam bukunya "*Applied Cryptography*", Kriptografi adalah ilmu pengetahuan dan seni

menjaga *message* agar tetap aman (*secure*) [4]. Kriptografi merupakan salah satu cabang ilmu algoritma matematika. Para penggemar kriptografi sering disebut *cryptographer*, sedangkan kebalikannya adalah *cryptanalyst* yang berusaha memecahkan sandi kriptografi. Kriptografi diilhami dengan teknik enkripsi atau teknik penyandian yang mengubah sebuah pesan dari pesan yang dapat dibaca (*plaintext*) menjadi sebuah pesan yang acak dan yang sulit diartikan (*ciphertext*). Untuk dapat membaca pesan yang terenkripsi diperlukan proses terbalik dari enkripsi yang disebut dekripsi (*decryption*).



Gambar 2.1 Proses Enkripsi-Dekripsi Sederhana

Plaintext dinotasikan dengan M (*Message*), yang dapat berupa *bit stream*, *file text*, *digitized voice stream*, *digital video image* atau lebih singkatnya M adalah data *binary*.

Ciphertext dinotasikan dengan C, juga berupa data *binary* yang kadang-kadang mempunyai ukuran yang sama dengan M, lebih kecil dari M atau lebih besar dari M.

Fungsi enkripsi E, berfungsi untuk mengubah M menjadi C, dalam matematika dinotasikan dengan :

$$E(M) = C \qquad (2,1)$$

Fungsi dekripsi D, berfungsi untuk mengubah C menjadi M, dalam matematika dinotasikan dengan :

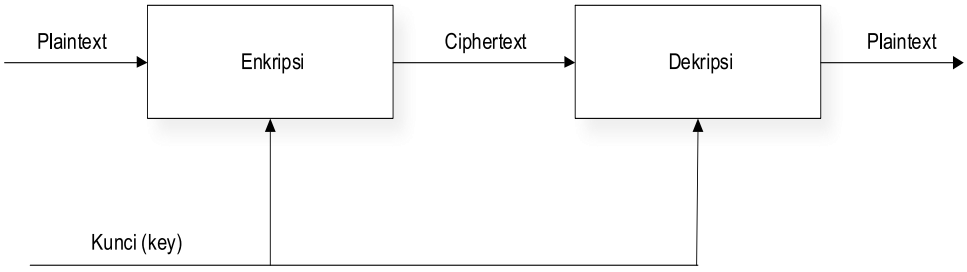
$$D(C) = M \qquad (2,2)$$

Apabila fungsi enkripsi E ditambahkan sebuah kunci K maka notasi matematikanya menjadi :

$$E_K(M) = C \qquad (2,3)$$

Dan untuk fungsi dekripsi D maka notasi matematikanya menjadi :

$$D_K(C) = M \qquad (2,4)$$



Gambar 2.2 Enkripsi dan Dekripsi Dengan Sebuah Kunci

2.1.2 Kriptanalisis

Kriptografi menyebabkan timbulnya kriptanalisis, yaitu ilmu pengetahuan dan seni untuk membongkar data acak. Praktisi dari kriptanalisis disebut kriptanalis. Kriptanalis berkembang sejalan dengan kriptografi. Setiap ada algoritma kriptografi baru yang dibuat oleh kriptografer langsung diikuti oleh adanya upaya percobaan kriptanalisis. Percobaan kriptanalisis ini disebut *attack*

(serangan). Kriptanalisis mencoba mengembalikan data jelas tanpa menggunakan akses ke kunci kriptografi. Ukuran keberhasilan suatu upaya kriptanalisis adalah sampai sejauh mana keberhasilan diketahuinya data jelas atau kunci kriptografi [6].

Asumsi dasar dari suatu kriptosistem adalah bahwa seorang kriptanalis mengetahui keseluruhan mekanisme enkripsi, terkecuali kuncinya. Berdasarkan hal itu maka serangan terhadap suatu kriptografi dapat diklasifikasikan menjadi empat, yaitu [2] :

1. *Ciphertext-only attack*

   Pada jenis serangan ini, kriptanalis mempunyai *ciphertext* dari beberapa data yang dienkripsikan dengan algoritma kriptografi yang sama. Tujuan kriptanalis adalah mendapatkan *plaintext* dari *ciphertext* atau lebih baik lagi menarik kesimpulan mengenai kunci yang digunakan.

2. *Known-plaintext attack*

   Pada jenis serangan ini, kriptanalis tidak hanya memiliki *ciphertext*, tetapi juga *plaintext* dari *ciphertext* tersebut.Tujuan kriptanalis adalah untuk menarik kesimpulan mengenai kunci yang digunakan untuk mengenkripsi data atau algoritma untuk mendekripsikan *ciphertext*.

3. *Chosen-plaintext attack*

   Pada jenis serangan ini, kriptanalis selain mengetahui *ciphertext* dan *plaintext*, juga dapat memilih *plaintext* yang diinginkan yang biasanya

memiliki lebih banyak informasi tentang kunci.Tujuan kriptanalis adalah menarik kesimpulan mengenai kunci yang digunakan untuk mengenkripsi data.

4. *Adaptive-chosen-plaintext attack*

Dalam hal ini kriptanalis tidak hanya dapat memilih *plaintext* yang telah dienkripsi, tetapi juga dapat memodifikasi pilihan tersebut berdasarkan hasil enkripsi sebelumnya. Kriptanalis mengetahui blok *plaintext* yang lebih kecil dan kemudian memilih yang lain berdasarkan hasil enkripsi pertama, kedua, dan seterusnya.

2.2 Kriptografi Klasik dan Kriptografi *Modern*

2.2.1 Kriptografi Klasik

Sebelum komputer ada, kriptografi dilakukan dengan algoritma berbasis karakter. Algoritma yang digunakan termasuk ke dalam sistem kriptografi simetri dan digunakan jauh sebelum sistem kriptografi kunci public ditemukan.

Terdapat sejumlah algoritma yang tercatat dalam sejarah kriptografi (sehingga dinamakan algoritma kriptografi klasik), namun sekarang algoritma tersebut sudah usang karena sangat mudah dipecahkan.

Tiga alasan mempelajari algoritma kriptografi klasik [2] :

1. Untuk memberikan pemahaman konsep dasar kriptografi.

2. Dasar dari algoritma kriptografi *modern*.

3. Dapat memahami potensi-potensi kelemahan sistem *chiper*.

Algoritma kriptografi klasik ada 2, yaitu [2] :

1. *Chiper* Substitusi (*Substitution Chipers*)

Ini adalah algoritma kriptografi yang mula-mula digunakan oleh kaisar Romawi, *Julius Caesar* (sehingga dinamakan juga *caesar chiper*), untuk menyandikan pesan yang ia kirim kepada para gubernurnya. Caranya adalah dengan mengganti (mensubstitusi) setiap karakter dengan karakter lain dalam susunan abjad (alfabet).

Misalnya, tiap huruf disubstitusi dengan huruf ketiga berikutnya dan susunan abjad. Dalam hal ini kuncinya adalah jumlah pergeseran huruf (yaitu k = 3).

Tabel substitusi :

p_i : ABCDEFGHIJKLMNOPQRSTUVWXYZ

c_i: DEFGHIJKLMNOPQRSTUVWXYZABC

Dengan mengkodekan setiap huruf abjad dengan *integer* sebagai berikut : A = 0,5 = 1, ...,Z=25, maka secara matematis *Caesar chipper* menyandikan *plainteks* p_i menjadi c_i, dengan aturan :

$$c_i = E(p_i) = (p_i + 3) \bmod 26 \qquad (2,5)$$

Dan dekripsi *chiperteks* c_i menjadi p_i, dengan aturan :

$$P_i = D(c_i) = (c_i - 3) \bmod 26 \qquad (2,6)$$

Karena hanya ada 26 huruf abjad, maka pergeseran huruf yang mungkin dilakukan adalah dari 0 sampai 25. Secara umum, untuk pergeseran huruf sejauh *k* (dalam hal ini *k* adalah kunci enkripsi dandekripsi), fungsi enkripsi adalah :

$$C_j = E(p_i) = (p_i + k) \bmod 26 \qquad (2,7)$$

Dan fungsi dekripsi adalah :

$$P_i = D(C_j) = (c_i - k) \bmod 26 \qquad (2,8)$$

Jenis-jenis *chiper* substitusi ada4macam, yaitu [6] :

1. *Chiper* abjad-tunggal (*monoalphabetic chiper* atau *chiper* substitusi sederhana-*simple substitution chiper*)

Satu karakter di *plainteks* diganti dengan satu karakter yang bersesuaian. Jadi, fungsi *chipering*-nya adalah fungsi satu-ke-satu. Jika *plainteks* terdiri dari huruf-huruf abjad, maka jumlah kemungkinan susunan huruf-huruf *chiperteks* yang dapat dibuat adalah sebanyak

$$26! = 403.291.461.126.605.635.584.000.000$$

Caesar chiper adalah kasus khusus dari *chiper* abjad tunggal di mana susunan huruf*chiperteks* diperoleh dengan menggeser huruf-huruf alfabet sejauh 3 karakter.

ROT13 adalah program enkripsi sederhana yang ditemukan pada sistem *UNIX*. ROT13 menggunakan *chiper* abjad-tunggal dengan pergeseran k = 13 (jadi, huruf A diganti dengan N, B diganti dengan O, dan seterusnya).

Enkripsi arsip dua kali dengan ROT13 menghasilkan arsip semula :

P = ROT13(ROT13(P)) (2,9)

2. *Chiper* substitusi homofonik (*Homophonic substitution chiper*)

Seperti *chiper* abjad-tunggal, kecuali bahwa setiap karakter di dalam *plainteks* dapat dipetakan ke dalam salah satu dari karakter *chiperteks* yang mungkin. Misalnya huruf A dapat berkoresponden dengan 7, 9, atau 16, huruf B dapat berkoresponden dengan 5, 10, atau 23 dan seterusnya.

Fungsi *chipering*-nya memetakan satu-ke-banyak (*one-to-many*). *Chiper* substitusi homofonik digunakan pertama kali pada tahun 1401 oleh wanita bangsawan Mantua. *Chiper* substitusi homofonik lebih sulit dipecahkan daripada *chiper* abjad-tunggal. Namun, dengan *known-plaintext attack*, *chiper* ini dapat dipecahkan, sedangkan dengan *chipertext-only attack* lebih sulit.

3. *Chiper* abjad-majemuk (*Polyalpabetic substitution chiper*)

Merupakan *chiper* substitusi-ganda (*multiple-substitution chiper*) yang melibatkan penggunaan kunci berbeda. *Chiper* abjad-majemuk dibuat dari sejumlah *chiper* abjad-tunggal, masing-masing dengan kunci yang berbeda.Kebanyakan *chiper* abjad-majemuk adalah *chiper* substitusi periodik yang didasarkan pada periodem.

Misalkan *plainteks* P adalah :

$$P = p_1 p_2 \ldots p_m p_{m+1} \ldots p_{2m} \ldots \quad (2,10)$$

Maka *chiperteks* hasil enkripsi adalah :

$$E_k(P) = f_1(P_1) f_2(P_2) \ldots f_m(P_m) f_{m+1}(P_{m+1}) \ldots f_{2m}(P_{2m}) \ldots \quad (2,11)$$

Yang dalam hal ini p_i , adalah huruf-huruf di dalam *plainteks.* Untuk m= 1, *chipernya ekivalen* dengan *chipper* abjad-tunggal.

Contoh *chiper* substitusi periodik adalah *chiper Vigenere* yang ditemukan oleh kriptologi *Perancis, Blaise de Vigenere* pada abad 16. Misalkan K adalah deretan kunci

$$K = ki\ k2\ \ldots\ km \quad (2,12)$$

Yang dalam hal ini k_i untuk $1 \leq i \leq m$ menyatakan jumlah pergeseran pada huruf ke-i. Maka, karakter *chiperteks*$y_i(p)$ adalah

$$y_i(p) = (P+k_i) \bmod n \quad (2,13)$$

Misalkan periode m = 20, maka 20 karakter pertamadienkripsi dengan persamaan (5), dimana setiap karakter ke-i menggunakan kunci k_i. Untuk 20 karakter berikutnya, kembali menggunakan pola enkripsi yang sama. *Chiper* abjad-majemuk ditemukan pertama kali oleh *Leon Battista* pada tahun 1568. Metode ini digunakan oleh tentara *AS* selama Perang Sipil *Amerika*. Meskipun *chiper* abjad-majemuk dapat dipecahkan dengan mudah (dengan bantuan

komputer), namun anehnya banyak program keamanan komputer (*computer security*) yang menggunakan *chipper* jenis ini.

4. *Chiper* substitusi poligram (*Polygram substitution chiper*)

Blok karakter disubstitusi dengan blok *chiperteks*. Misalnya ABA diganti dengan RTQ, ABB diganti dengan SLL, dan lain-lain. *Playfair chiper*, ditemukan pada tahun 1854, termasuk ke dalam *chiper* substitusi poligram dan digunakan oleh Negara *Inggris* selama Perang Dunia I.

2. *Chiper* Transposisi

Pada *chiper* transposisi, *plainteks* tetap sama, tetapi urutannya diubah. Dengan kata lain, algoritma ini melakukan *transpose* terhadap rangkaian karakter di dalam teks. Nama lain untuk metode ini adalah permutasi, karena *transpose* setiap karakter di dalam teks sama dengan mempermutasikan karakter-karakter tersebut.

Misalkan *plainteks* adalah

TEKNIK INFORMATIKA UNIKOM BANDUNG

Untuk meng-enkripsi pesan, *plainteks* ditulis secara *horizontal* dengan lebar kolom tetap, misal selebar 6 karakter (kunci k=6) :

TEKNIK

INFORM

ATIKAU

NIKOMB

ANDUNG

Maka *chiperteks*nya dibaca secara *vertikal* menjadi :

TIANAENTINKFIKDNOKOUIRAMNKMUBG

Untuk mendekripsi pesan, kita membagi panjang *chiperteks* dengan kunci. Pada contoh ini, kita membagi 30 dengan 6 untuk mendapatkan 5. Algoritma dekripsi identik dengan algoritma enkripsi. Jadi, untuk contoh ini, kita menulis *chiperteks* dalam baris-baris selebar 5 karakter menjadi :

TIANA

ENTIN

KFIKD

NOKOU

IRAMN

KMUBG

Dengan membaca setiap kolom kita memperoleh pesan semula :

TEKNIK INFORMATIKA UNIKOM BANDUNG

2.2.2 Kriptografi *Modern*

Algoritma kriptografi *modern* umumnya beroperasi dalam *mode bit* ketimbang *mode* karakter (seperti yang dilakukan pada *cipher* substitusi atau *cipher* transposisi dari algoritma kriptografi klasik). Operasi dalam *mode bit* berarti semua data dan informasi (baik kunci, *plainteks*, maupun *cipherteks*) dinyatakan dalam rangkaian (*string*) *bit biner*, 0 dan 1. Algoritma enkripsi dan dekripsi memproses semua data dan informasi dalam bentuk rangkaian *bit*. Rangkaian *bit* yang menyatakan *plainteks* dienkripsi menjadi *cipherteks* dalam bentuk rangkaian *bit*, demikian sebaliknya [6].

Enkripsi *modern* berbeda dengan enkripsi konvensional. Enkripsi *modern* sudah menggunkan komputer untuk pengoperasiannya, berfungsi untuk mengamankan data baik yang di*transfe r*melalui jaringan komputer maupun yang bukan. Hal ini sangat berguna untuk melindungi *privacy* ,*data integrity*, *authentication* dan *non-repudiation*. Perkembangan algoritma kriptografi *modern* berbasis *bit* didorong oleh penggunaan komputer *digital* yang merepresentasikan data dalam bentuk *biner*.

Macam-macam Algoritma kriptografi *modern* [6] :

Kriptografi *modern* merupakan suatu perbaikan yang mengacu pada kriptografi klasik. Pada kriptogarfi *modern* terdapat berbagai macam algoritma yang dimaksudkan untuk mengamankan informasi yang dikirim melalui jaringan komputer. Algoritma kriptografi *modern* terdiri dari tiga jenis :

1. Algoritma *Simetris*

Algoritma *simetris* adalah algoritma yang menggunakan kunci yang sama untuk enkripsi dan dekripsinya. Algoritma kriprografi *simetris* sering disebut algoritma kunci rahasia, algoritma kunci tunggal, atau algoritma satu kunci, dan mengharuskan pengirim dan penerima menyetujui suatu kunci tertentu. Kelebihan dari algoritma kriprografi *simetris* adalah waktu proses untuk enkripsi dan dekripsi relatif cepat. Hal ini disebabkan efesiensi yang terjadi pada pembangkit kunci. Karena prosesnya relatif cepat maka algoritma ini tepat untuk digunakan pada sistem komunikasi digital secara *real time* seperti *GSM.*

2. Algoritma *Asimetris*

Algoritma *Asimetris* adalah pasangan kunci kriptografi yang salah satunya digunakan untuk proses enkripsi dan satu lagi deskripsi. Semua orang yang mendapatkan kunci publik dapat menggunakannya untuk mengenkripsi suatu pesan, sedangkan hanya satu orang saja yang memiliki rahasia itu, yang dalam hal ini kunci rahasia, untuk melakukan pembongkaran terhadap kode yang dikirim untuknya. Contoh algoritma terkenal yang menggunakan kunci *Asimetris* adalah *RSA* (merupakan singkatan dari nama penemunya, yakni *Rivest, Shamir* dan *Adleman*).

3. Algoritma *Hibrida*

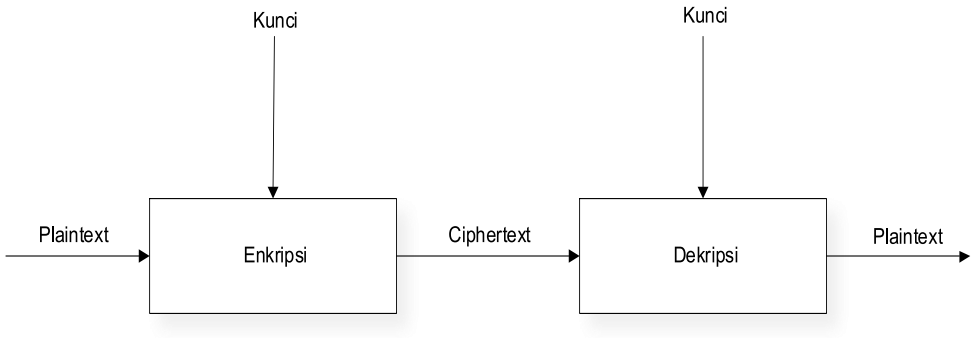
Algoritma *hibrida* adalah algoritma yang memanfaatkan dua tingkatan kunci, yaitu kunci rahasia (*simetri*) – yang disebut juga *session key* (kunci sesi) –

untuk enkripsi data dan pasangan kunci rahasia- kunci publik untuk pemberian tanda tangan *digital* serta melindungi kunci *simetri*.

2.3 Jenis-Jenis Algoritma Kriptografi

2.3.1 Algoritma *Simetris*

Algoritma *simetris* melakukan proses enkripsi dan dekripsi dengan memakai kunci yang sama (seperti ilustrasi pada Gambar 2.3).



Gambar 2.3 Mekanisme Kriptografi *Simetris*

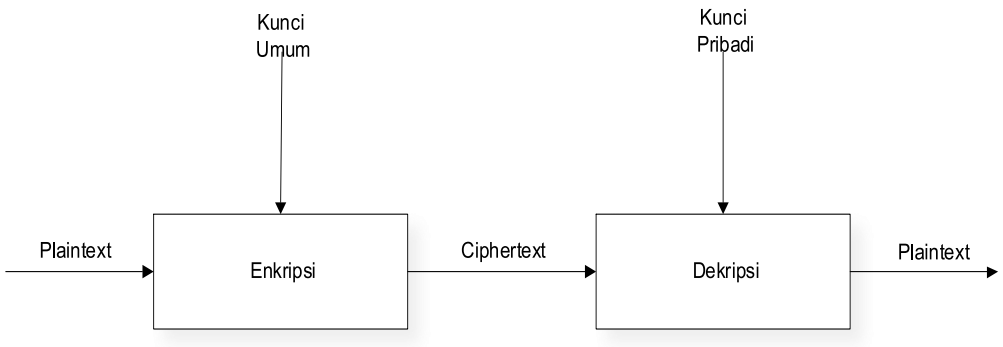
Algoritma *simetris* dapat beroperasi dengan mekanisme *block chipper* ataupun mekanisme *stream chiper*. Pada mekanisme *block chiper*, fungsi enkripsi beroperasi pada blok-blok *plaintext* yang berukuran tetap. Jika blok yang digunakan mempunyai panjang *bit*. Umumnya panjang blok yang digunakan adalah 64 *bit*. Sedangkan pada mekanisme *stream chipper*, enkripsi dilakukan pada data *plaintext* yang ukurannya tidak dibatasi. *Stream chipper* memproses data sebagai deretan karakter-karakter, dengan menggunakan karakter berupa satu *bit* data ataupun sejumlah kecil *bit* data. *Stream chipper* juga dapat

dikombinasikan dengan *block chiper*, yaitu dengan menggunakan blok-blok sebagai karakter penyusunnya [2].

2.3.2 Algoritma *Asimetris*

Berbeda dengan algoritma *simetris*, algoritma *Asimetris* menggunakan dua kunci yang berbeda dalam proses enkripsi dan dekripsi data. Algoritma *Asimetris* juga dikenal dengan nama Kriptografi Kunci Publik. Sistem ini menggunakan dua kunci/*key* (berupa kode angka), satu kunci digunakan untuk mengenkripsi data, dan kunci lainnya untuk mendekripsi data tersebut. Kedua kunci tersebut terhubung secara matematis dengan rumus tertentu, sehingga data yang telah dienkripsi oleh suatu kunci hanya biasa didekripsi dengan menggunakan kunci pasangannya.

Setiap pengguna mempunyai dua kunci, yaitu kunci publik dan kunci *privat*. Pengguna dapat menyebarkan kunci publik secara bebas. Karena adanya hubungan yang khusus antara kedua kunci, penggunadan siapapun yang menerima kunci publik tersebut mendapat jaminan bahwa data yang telah dienkripsi dengan kunci tersebut hanya biasa didekripsi oleh kunci pasangannya, berapa kunci *privat* milik pengguna yang sama. Keamanan ini terjamin selama pengguna dapat menjaga kerahasiaan kunci privat. Pasangan kunci ini harus dibuat secara khusus oleh pemiliknya. Algoritma yang biasanya digunakan untuk pembuatan pasangan kunci adalah algoritma *RSA* (dinamakan berdasarkan inisial pembuatnya, yaitu :*Rivest, Shamir,* dan *Adleman*) [2].



Gambar 2.4 Mekanisme Kriptografi *Asimetris*

Karena algoritmanya yang lebih kompleks, komputasi yang diperlukan untuk sistem kripto *Asimetris* jauh lebih besar dibandingkan dengan sistem kripto *simetris*. Oleh karena itu, biasanya data yang dikomunikasikan tetap dienkrip dengan menggunakan system kripto *simetris*. Kunci *simetris* yang digunakan adalah kunci sesi (session key) bersifat unik untuk setiap sesi komunikasi yang dilakukan. Kunci sesi itu merupakan kunci *simetris* yang digunakan untuk komunikasi data pada jangka waktu tertentu. Di pihak lain, sistem kripto *Asimetris* digunakan pada awal sesi komunikasi untuk pengiriman kunci sesi yang diperlukan.

2.4 *Rijndael (AES)*

2.4.1 Deskripsi *Rijndael*

Rijndael termasuk dalam jenis algoritma kriptografi yang sifatnya simetri dan *cipher block*. Dengan demikian algoritma ini rnempergunakan kunci yang sama saat enkripsi dan dekripsi serta masukan dan keluarannya berupa blok dengan jumlah *bit* tertentu [5].

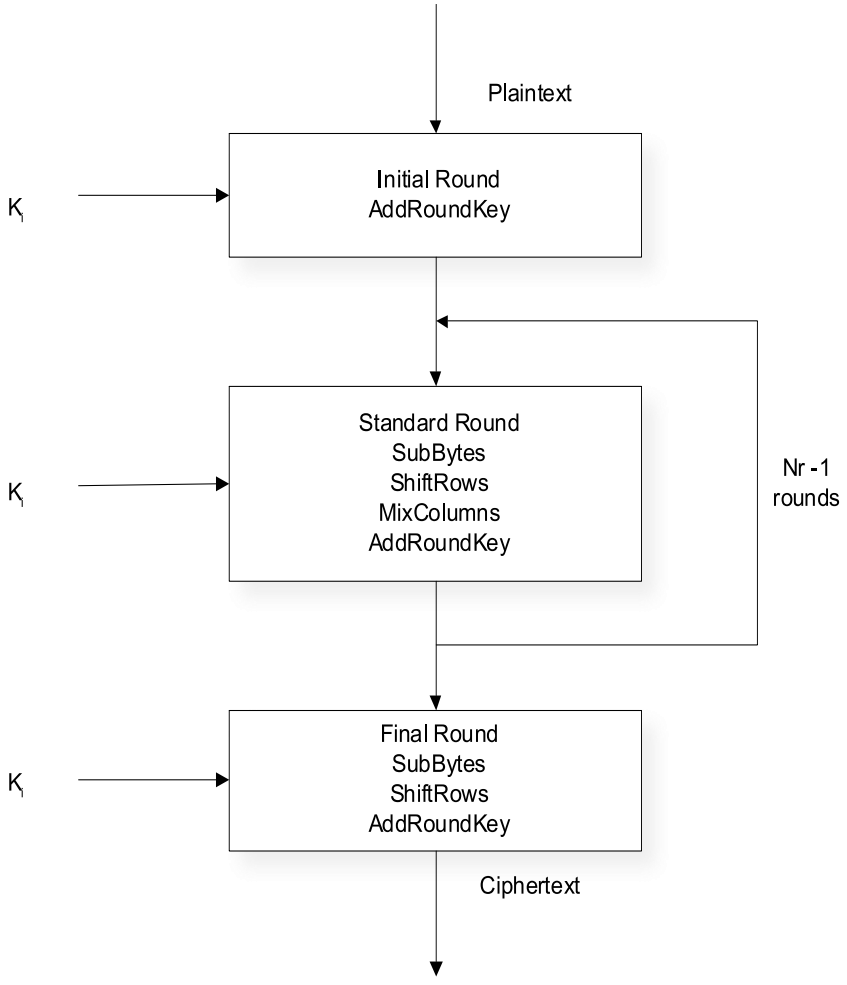
Rijndael mendukung berbagai variasi ukuran blok dan kunci yang akan digunakan. Namun *Rijndael* mempunyai ukuran blok dan kunci yang tetap sebesar 128, 192, 256 *bit*. Pemilihan ukuran blok data dan kunci akan menentukan jumlah proses yang harus dilalui untuk proses enkripsi dan dekripsi [5].

2.4.2 Algoritma *Rijndael*

Algoritma kriptografi bernama *Rijndael* yang didesain oleh *Vincent Rijmen* dan *John Daemen* asal *Belgia* keluar sebagai pemenang kontes algoritma kriptografi pengganti *DES* yang diadakan oleh *NIST* (*National Institutesof Standards and Technology*) milik pemerintah *Amerika Serikat* pada 26 November 2001. Algoritma *Rijndael* inilah yang kemudian dikenal dengan *Advanced Encryption Standard (AES).* Setelah mengalami beberapa proses standardisasi oleh *NIST, Rijndael* kemudian diadopsi menjadi *standard* algoritma kriptografi secara resmi pada 22 Mei 2002. Pada 2006, *AES* merupakan salah satu algoritma terpopuler yang digunakan dalam kriptografi kunci simetrik [5].

Algoritma *Rijndael* menggunakan *substitusi, permutasi*, dan sejumlah putaran yang dikenakan pada tiap blok yang akan dienkripsi atau dekripsi. Untuk setiap putarannya, *Rijndael* menggunakan kunci yang berbeda. Kunci setiap putaran disebut *round key*. Tetapi tidak seperti *DES* yang berorientasi *bit*, *Rijndael* beroperasi dalam orientasi byte sehingga memungkinkan untuk implementasi algoritma yang efisien kedalam software dan hardware. Ukuran blok untuk algoritma *Rijndael* adalah 128*bit* (16 *byte*).

Algoritma *Rijndael* dapat mendukung panjang kunci 128 *bit* sampai 256 *bit* dengan step 32 *bit*. Panjang kunci berpengaruh pada jumlah putaran yang dikenakan pada tiap blok. Misalnya, untuk ukuran blok dan panjang kunci sebesar 128 *bit* ditentukan 10 putaran, sedangkan untuk ukuran blok 128 *bit* dan panjang kunci 256 *bit* jumlah putaran yang ditentukan adalah 14 putaran.



Gambar 2.5 Struktur Algoritma *Rijndael* Secara *Global*

Algoritma *Rijndael* mempunyai 3 parameter sebagai berikut [5] :

1. *Plainteks* : *array* yang berukuran 16 *byte*, yang berisi data masukan.
2. *Chiperteks* : *array* yang berukuran 16 *byte*, yang berisi hasil enkripsi.
3. *Key* : *array* yang berukuran 16 *byte* (untuk panjang kunci 128 *bit*), yang berisi kunci *ciphering* (disebut juga *cipher key*).

Blok-blok data masukan dan kunci dioperasikan dalam bentuk *array*. Setiap anggota *array* sebelum menghasilkan keluaran *ciphertext* dinamakan dengan *state*. Setiap *state* akan mengalami proses yang secara garis besar terdiri dari empat tahap yaitu, *AddRoundKey*, *SubBytes*, *ShiftRows*, *dan MixColumns*. Kecuali tahap *MixColumns*, ketiga tahap lainnya akan diulang pada setiap proses sedangkan tahap *MixColumns* tidak akan dilakukan pada tahap terakhir. Proses dekripsi adalah kebalikkan dari dekripsi.

Karena terjadi beberapa tahap dalam proses enkripsi, maka diperlukan *subkey* yang akan dipakai pada tiap tahap. Pengembangan jumlah kunci yang akan dipakai diperlukan karena kebutuhan *subkey* yang akan dipakai dapat mencapai ribuan *bit*, sedangkan kunci yang disediakan secara *default* hanya128-256 *bit*. Jumlah total kunci yang diperlukan sebagai *subkey* adalah sebanyak Nb(Nr+l), dimana Nb adalah besarnya blok data dalam satuan *word*. Sedangkan Nr adalah jumlah tahapan yang harus dilalui dalam satuan *word*. Sebagai contoh, bilamana digunakan 128 *bit* (4 *word*) blok data dan 128 *bit* (4 *word*) kunci maka akan dilakukan 10 kali proses (lihat Tabel 2.1). Dengan demikian dari rumus

didapatkan 4(10+l) = 44 *word* = 1408 *bit* kunci. Untuk melakukan pengembangan jumlah kunci yang akan dipakai dari kunci utama maka dilakukan *key schedule*.

Tabel 2.1 Jumlah Proses Berdasarkan *Bit* Blok dan Kunci

Panjang Kunci (NK) Dalam *words*	Ukuran Blok Data (NB) Dalam *words*	Jumlah Proses (NR)
4	4	10
6	4	12
8	4	14

2.4.2.1 *Key Schedule*

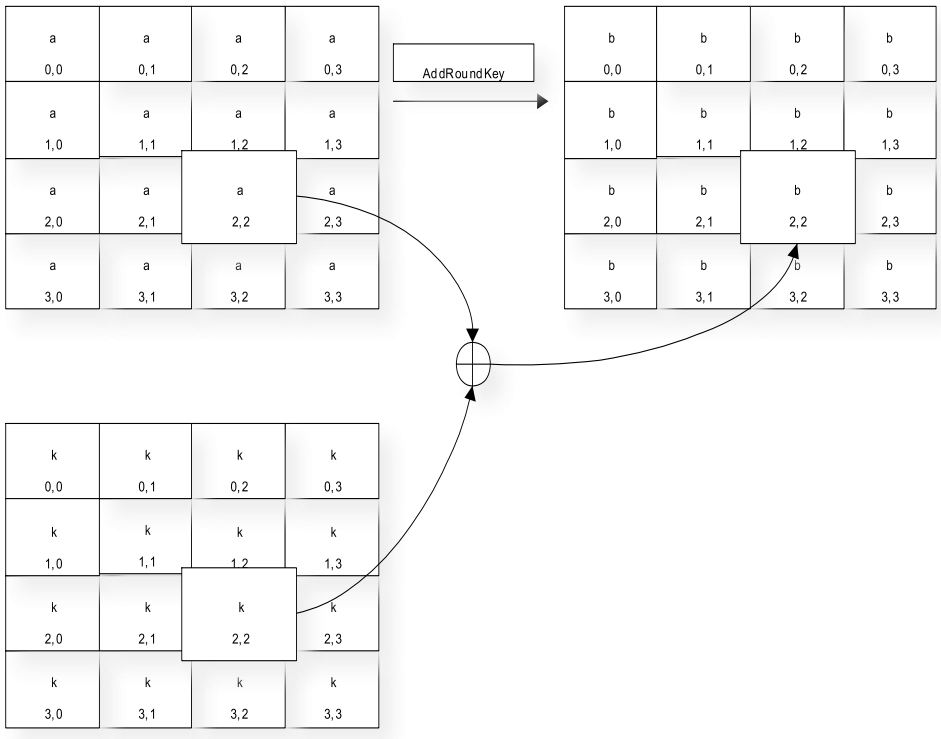
Proses *key schedule* diperlukan untuk mendapatkan *subkey* dari kunci utama agar cukup untuk melakukan enkripsi dan dekripsi. Proses ini terdiri dari beberapa operasi, yaitu [5] :

1. Operasi *Rotate*, yaitu operasi perputaran 8 *bit* pada 32 *bit* dari kunci.
2. Operasi *Sub Bytes*, padaoperasi ini 8*bit* dari *subkey* disubstitusikan dengan nilai dari *S-Box*.
3. Operasi *Rcon*,operasi ini dapat diterjemahkan sebagai operasi pangkat 2 nilai tertentu dari *user*. Operasi ini menggunakan nilai-nilai dalam *Galoisfield*. Nilai-nilai dari *Rcon* kemudian akan di-*XOR* dengan hasil operasi *Sub Bytes*.

4. Operasi *XOR* dengan w[i-Nk] yaitu *word* yang berada pada Nk sebelumnya.

2.4.2.2 *AddRoundKey*

Pada proses ini *subkey* digabungkan dengan *state*. Proses penggabungan ini menggunakan operasi *XOR* untuk setiap *byte* dari *subkey* dengan *byte* yang bersangkutan dari *state*. Untuk setiap tahap, *subkey* dibangkitkan dari kunci utama dengan menggunakan proses *key schedule*. Setiap *subkey* berukuran sama dengan *state* yang bersangkutan [5]. Proses *AddRoundKey* diperlihatkan pada Garnbar 2.6.



Gambar 2.6 Proses *AddRoundKey*

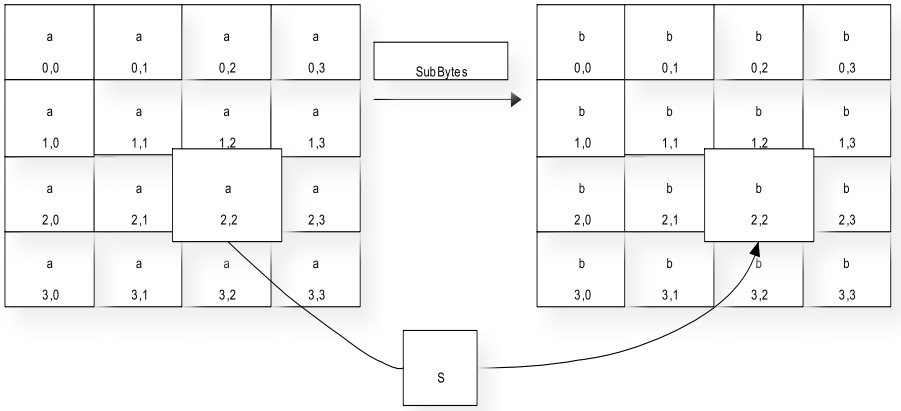
2.4.2.3 *Sub Bytes*

Proses *Sub Bytes* adalah operasi yang akan melakukan substitusi tidak *linear* dengan cara mengganti setiap *byte state* dengan *byte* pada sebuah tabel yang dinamakan tabel *S-Box* [5].

Sebuah tabel *S-Box* terdiri dari 16 baris dan 16 kolom dengan masing masing berukuran 1 *byte*. Tabel *S-Box* diperlihatkan pada Gambar 2.2, sedangkan proses *Sub Bytes* diperlihatkan pada Gambar 2.7.

Tabel 2.2 *AES S-Box*

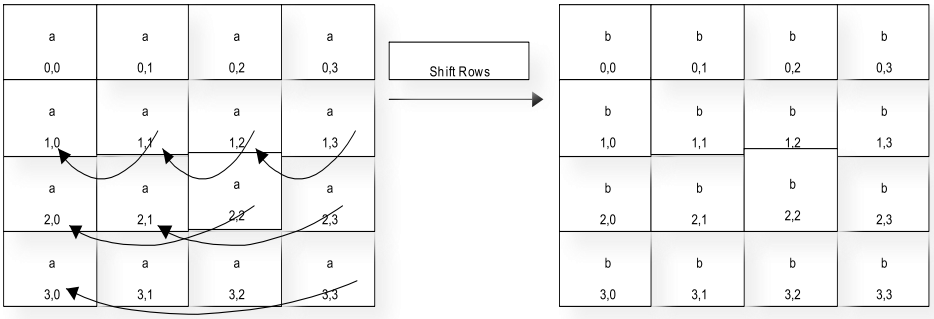
		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	53	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	de	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	38	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	37	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	98	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16



Gambar 2.7 Proses *Sub Bytes*

2.4.2.4 *Shift Rows*

Proses *Shift Rows* akan beroperasi pada tiap baris dari tabel *state*. Proses ini akan bekerja dengan cara memutar *byte-byte* pada 3 baris terakhir (baris1, 2, dan 3) dengan jumlah perputaran yang berbeda-beda. Baris 1 akan diputar sebanyak 1 kali, baris 2 akan diputar sebanyak 2 kali, dan baris 3 akan diputar sebanyak 3 kali. Sedangkan baris 0 tidak akan diputar [5]. Proses *Shift Rows* diperlihatkan pada Gambar 2.8.



Gambar 2.8 Proses *Shift Rows*

2.4.2.5 *Mix Columns*

Proses*Mix Columns* akan beroperasi pada tiap kolom dari tabel *state*. Operasi ini menggabungkan 4 *bytes* dari setiap kolom tabel *state* dan menggunakan transformasi *linier*. Operasi *Mix Columns* memperlakukan setiap kolom sebagai polinomial 4 suku dalam *Galois field* dan kemudian dikalikan dengan $c(x)$*modulo*(x^4+1), dimana $c(x)=3x^3+x^2+x+2$. Kebalikan dari *polynomial* ini adalah $c(x) = 1x^3+13x^2+9x+14$. Operasi *Mix Columns* juga dapat dipandang sebagai perkalian matriks.

Langkah *Mix Columns* dapat ditunjukkan dengan mengalikan 4 bilangan di dalam *Galois field* oleh matriks berikut ini.

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a0 \\ a1 \\ a2 \\ a3 \end{bmatrix} \qquad (2,14)$$

Atau bila dijabarkan :

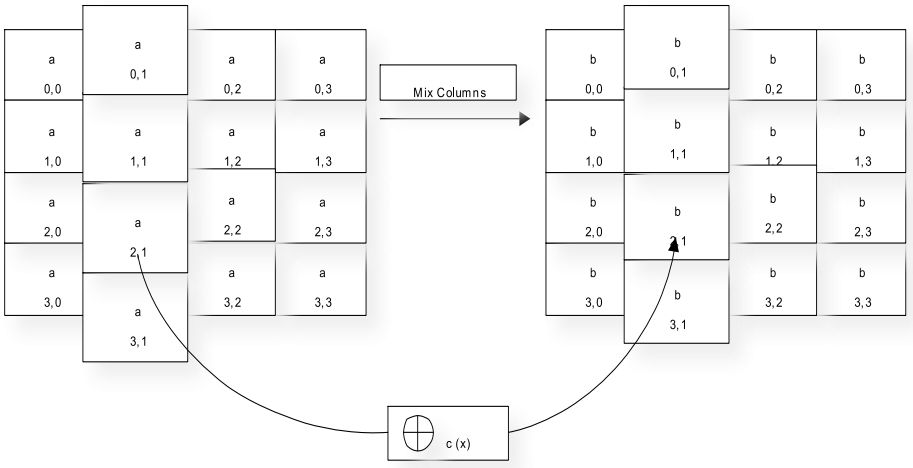
$$r0 = 2a0 + a3 + a2 + 3a1 \quad (2,15)$$

$$r1 = 2a1 + a0 + a3 + 3a2 \quad (2,16)$$

$$r2 = 2a2 + a1 + a0 + 3a3 \quad (2,17)$$

$$r3 = 2a3 + a2 + a1 + 3a0 \quad (2,18)$$

Operasi penjumlahan di atas dilakukan dengan operasi *XOR*, sedangkan operasi perkalian dilakukan dalam *Galois field*.



Gambar 2.9 Proses *Mix Columns*

2.4.3 Keamanan Algoritma *Rijndael*

Untuk *Rijndael*, tipe serangan *square attacks* cukup menjadi dikenal sebagai serangan terbaik terhadap *Rijndael*. *Square attacks* adalah serangan yang

memanfaatkan struktur orientasi *byte*. Algoritma ini bekerja dengan baik pada *square cipher* yang bekerja dalam 6 putaran. Untuk *Rijndael* dengan kunci sepanjang128 *bit*, maka serangan ini lebih cepat dari pada *exhaustive search*, hingga 6 kali iterasi *Rijndael*. Namun untuk *AES*, jelas bahwa serangan ini tidak mungkin dipraktekkan karena jumlah putaran pada *Rijndael*, mengakibatkan batas keamanan untuk algoritma ini menjadi lebih besar [5].

2.5 Twofish

2.5.1 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 [4] :

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 algoritma, baik pada *Intel Pentium Pro* dan perangkat lunak lainnya dan *platform* perangkat keras.

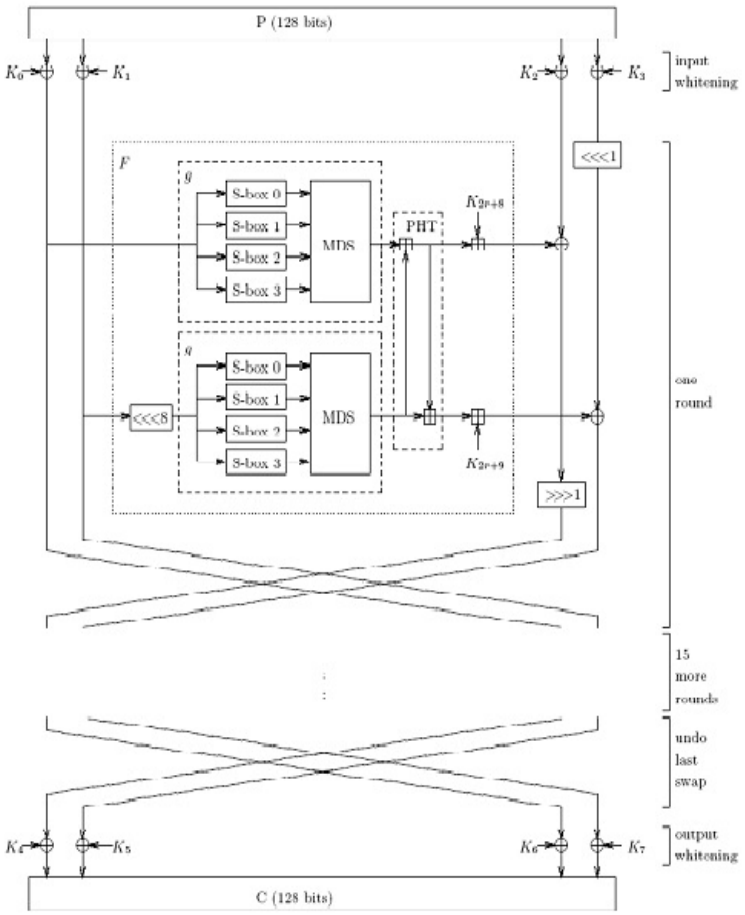
5. Rancangan yang fleksibel. Rancangan yang fleksibel ini dapat diartikan misalnya dapat menerima panjang kunci tambahan, dapat diterapkan pada *platform* dan aplikasi yang sangat *variatif,* serta cocok untuk *cipher* aliran, fungsi *hash*, dan *MAC*.

6. Rancangan yang sederhana agar memudahkan proses analisis dan implementasi algoritma.

2.5.2 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 [4].

Plaintext dipecah menjadi empat buah *word* 32-*bit*. Pada *whitening* input, keempat *word* itu di *XOR*-kan dengan empat *keyword.* Dan diikuti dengan keenam belas *round*. Dalam tiap *round,* dua *word* kiri digunakan sebagai *input* fungsi G (salah satunya dirotasikan dengan 8 *bit* terlebih dahulu). Struktur algoritma *Twofish* yang dikutip dari sebuah jurnal yang berjudul " *Implementation of the Twofish chipper using FPGA devices*" yang ditulis oleh *Pawel Chodoweic, Kris Gaj* dapat dilihat secara global pada Gambar 2.10.



Gambar 2.10 Struktur Algoritma *Twofish* Secara *Global*

Langkah-langkah algoritma *Twofish* adalah sebagai berikut :

1. Masukan satu blok *plaintext t*adalah 128*bit*. Satu blok tersebut dibagi menjadi 4 buah sub-blok yang masing-masing sepanjang 32 *bit* (A, B, C, danD).

2. Masing-masing sub-blok tersebut melalui proses *whitening* dengan meng-*XOR*-kan dengan kunci K0, Kl , K2, danK3.

$$Ro,i = Pi \oplus Ki \quad i = 0...3 \qquad (2,19)$$

Proses pada *input whitening* :

R0,I=Pi XORKi i =0...3 (2,20)

(Fr,0,Fr,l)=F(Rr,0,Rr,l ,r) (2,21)

Rr+ 1,0=ROR(Rr,2XORFr,0,l) (2,22)

Rr+1,1 =ROL (Rr,3,l)XORFr,l (2,23)

Rr+l,2=Rr,0 (2,24)

Rr+l,3=Rr,l (2,25)

Proses pada *outpu twhitening* :

Ci = Rl6 (i+2)mod 4 XOR Ki +4 $\overrightarrow{\ }$ I =0...3 (2,26)

Langkah-langkah 1 putaran adalah sebagai berikut :

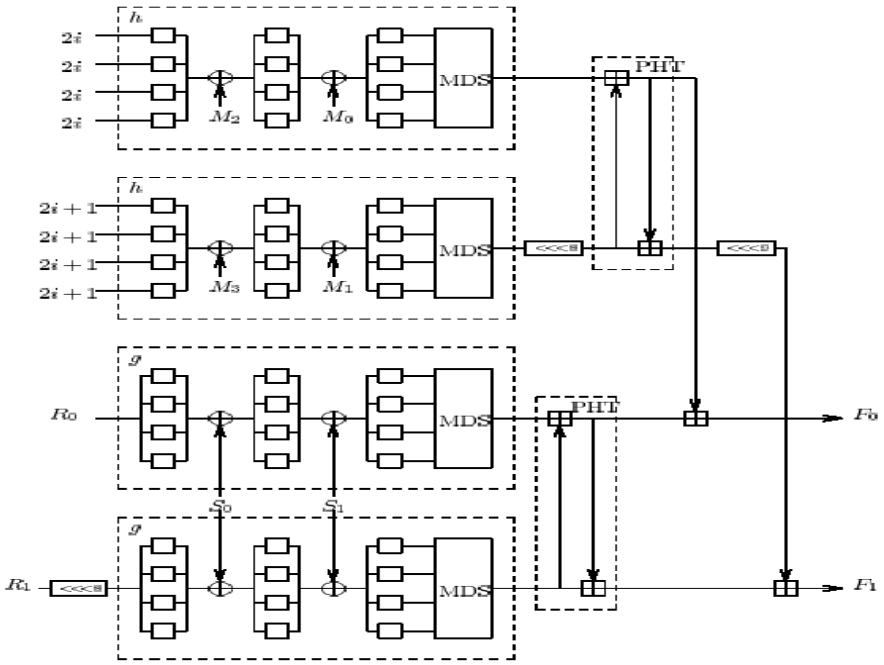
1. 2 buah 32 *bit* yang kiri (A dan B) merupakan input dari fungsi *G* (yang merupakan bagian dari fungsi *F*), yang salah satunya (B) di geser kekiri sejauh 8 *bit* dahulu.
2. Fungsi *G* memiliki 4 buah kotak substitusi yang dibangkitkan oleh kunci.
3. Keluaran fungsi kotak substitusi dilakukan percampuran linear menggunakan kotak *Most Distance Separable*.
4. Keluaran fungsi *G* dimasukkan ke fungsi transformasi *pseudo-Hadamard*, kemudian ditambahkan dengan 2 buah 32 *bit* dari kunci.

5. Dua buah 32 *bit* hasil kemudian di *XOR* kan dengan C dan D. Hasil *XOR* dengan C digeser ke kanan sejauh 1 *bit*. Dan untuk D sebelum di *XOR* kan digeser ke kiri sejauh1 *bit*.

6. 2 buah 32 *bit* kiri dan kanan dipertukarkan (A dan B dipertukarkan dengan C dan D).

2.5.2.1 Fungsi F

Fungsi F adalah permutasi yang bergantung pada kunci dengan nilai 64 *bit*. Fungsi ini menerima 3 argumen, dua buah 32 *bit* RO dan Rl, dan nomor putaranyang akan mengembalikan T0. Rl akan digeser sejauh 8 *bit* yang kemudian dikombinasikan ulang menggunakan transformasi *pseudo-hadamard* yang kemudian nantinya akan dipertukarkan dan dimasukkan kembali keputaran selanjutnya [4].



Gambar 2.11 Skema Fungsi F Pada Algoritma *Twofish*

Fungsi *G* merupakan inti dari algoritma *Twofish*. Masukan kata X dibagi menjadi empat *byte*. Setiap *byte* tiap *byte* 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 4x4 matriks *MDS* (menggunakan bidang GF(2^8) untuk proses komputasi). Vektor hasilnya kemudian diinterpretasikan sebagai kata berukuran 32 *bit* sebagai hasil fungsi *G* [4].

2.5.3 Proses *Whitening*

Whitening sebuah teknik meng *XOR*kan material kunci sebelum *round* pertama dan setelah *round* terakhir, digunakan oleh *Merkle* dalam *Khufu/Khafre*,

dan ditemukan oleh *Rivest* untuk *DES-X*. *Whitening* menambah tingkat kesulitan serangan pencarian kunci terhadap *ciphertext*, dengan menyembunyikan masukkan spesifik terhadap *round* pertama dan *round* terakhir dari fungsi F [4].

Twofish meng*XOR*kan128-*bit* sub-kunci sebelum *round Feistel* yang pertama, dan 128 *bit* lagi setelah *round Feistel* terakhir. Sub-kunci ini diperhitungkan dengan cara yang sama seperti sub-kunci *round*, tetapi tidak digunakan di tempat lain dalam *cipher*.

2.5.4 Keamanan 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 222,5 pasangan *chosen plainteks* dan 251 usaha. Metode kriptanalisis untuk *Twofish* ini masih terus dikembangkan. Tetapi tidak ada metode kriptanalisis untuk *Twofish* dengan jumlah putaran di atas Sembilan buah [4].

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 diketahui, tetapi kriptanalisis dapat mengetahui bahwa keduanya adalah sama untuk kedua kunci. Proses yang harus dilakukan sebanyak 264 *chosen plainteks* untuk setiap kunci yang dipilih dan dilakukan sekitar 234 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.

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 2131. 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 dipresentasikan 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 [4].

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.

2.6 Keamanan Data

Kata aman dapat didefinisikan sebagai terhindar dari serangan atau kegagalan. Jadi ada beberapa ancaman yang dapat mengacaukan sistem, yang sering tanpa kita sadari telah dikelilingi oleh berbagai bentuk ancaman. Suatu sistem baru dapat dikatakan aman apabila dalam segala keadaan sumber daya yang digunakan dan yang diakses adalah sesuai dengan kehendak pengguna. Untuk melaksanakan tujuan tersebutlah dirancang suatu sistem keamanan yang berfungsi melindungi sistem informasi.

Salah satu upaya pengamanan sistem informasi yang dapat dilakukan adalah kriptografi. Kriptografi sesungguhnya merupakan studi terhadap teknik

matematis yang terkait dengan aspek keamanan suatu sistem informasi, antara lain seperti kerahasiaan, integritas data, otentikasi, dan ketiadaan penyangkalan.

Keempat aspek tersebut merupakan tujuan fundamental dari suatu sistem kriptografi [2] :

1. Otoritas (*Authority*)

   Informasi yang berada pada sistem seharusnya dapat dimodifikasi oleh pihak yang berwenang. Modifikasi yang tidak diinginkan, dapat berupa penulisan tambahan pesan, pengubahan isi, pengubahan status, penghapusan, pembuatan pesan baru.

2. Integritas Data (*Data Integrity*)

   Penerima harus dapat memeriksa apakah pesan telah dimodifikasi ditengah jalan atau tidak. Seseorang penyusup seharusnya tidak memasukan tambahan kedalam pesan, mengurangi atau mengubah pesan selama data di perjalanan.

3. Otentikasi (*Authentication*)

   Otentikasi merupakan layanan yang terkait dengan identifikasi terhadap pihak-pihak yang ingin mengakses sistem informasi (*entity authentication*) maupun keaslian data dari sistem informasi itu sendiri (*data origin authentication*).

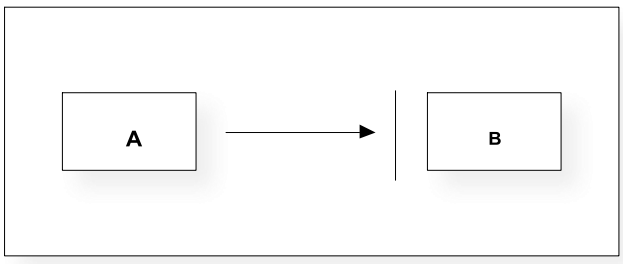
4. Ketiadaan Penyangkalan (*non-repudation*)

Ketiadaan penyangkalan adalah layanan yang berfungsi untuk mencegah terjadinya penyangkalan terhadap suatu aksi yang dilakukan oleh pelaku sistem informasi.

Melihat pada kenyataan semakin banyak data yang diproses dengan komputer dan dikirim melalui perangkat komunikasi elektronik, maka ancaman terhadap pengamanan data akan semakin meningkat. Beberapa pola ancaman terhadap komunikasi data dalam komputer dapat diterangkan sebagai berikut [6] :

a. *Interruption*

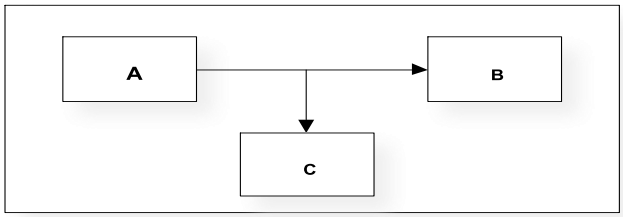
Terjadi bila data yang dikirimkan dari A tidak sampai pada orang yang berhak B. *Interruption* merupakan pola penyerangan terhadap sifat *availability* (ketersedian data). Contohnya adalah kerusakan pada *hardware*, kegagalan *operating system* sehingga sistem tidak dapat menemukan *file* yang dicari.



Gambar 2.12 *Interuption*

b. *Interception*

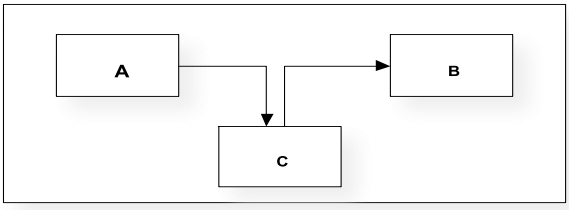
Terjadi bila pihak ketiga C berhasil membaca data yang dikirimkan. *Interception* merupakan pola penyerangan terhadap sifat *confidentiality/secrecy* (kerahasian data), contohnya adalah penggandaan program atau *file* data yang tidak terlihat, atau pencurian data pada jaringan dengan cara *wireteapping*.



Gambar 2.13 *Interception*

c. *Modification*

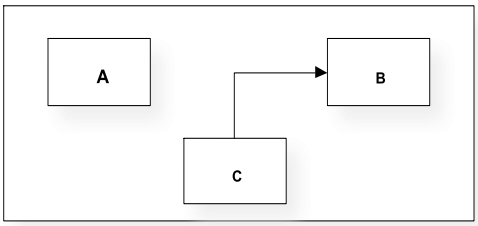
Pada serangan *modification* pihak ketiga C berhasil merubah pesan yang dikirimkan. *Modification* merupakan pola penyerangan terhadap sifat *integrity* (keasliandata).



Gambar 2.14 *Modification*

d. *Fabrication*

Pada serangan *fabrication* penyerang berhasil mengirimkan data ketujuan dengan memanfaatkan identitas orang lain. *Fabrication* merupakan pola penyerangan terhadap sifat *authenticity* (autentifikasi data).



Gambar 2.15 *Fabrication*

Untuk mengantisipasi ancaman-ancaman tersebut di atas perlu dilakukan usaha untuk melindungi data yang dikirim melalui saluran komunikasi, salah satu usaha tersebut adalah dengan menyandikan informasi yang ada pada data yang akan dikirim. Penyembunyian informasi yang ada dalam suatu bentuk tertentu yang tidak dapat dimengerti pihak lain (yang tidak berkepentingan) merupakan bagian kriptografi.

2.7 *Local Area Network (LAN)*

Dalam suatu sistem jaringan, dimana seluruh komputer saling berbagi data dan resources satu sama lain sehingga tercapai efisiensi dalam pemanfaatan teknologi, sangat dibutuhkan perangkat-perangkat khusus dan instalasi tertentu [8].

Tujuan dari suatu jaringan adalah menghubungkan jaringan-jaringan yang telah ada dalam jaringan tersebut sehingga informasi dapat ditransfer dari satu lokasi ke lokasi yanglain. Struktur geometrik ini disebut dengan *LAN Topologies* [8].

Terdapat 6 jenis topologi yaitu :

1. *Bus*

Topologi ini memiliki karakteristik sebagai berikut :

a. Merupakan satu kabel yang kedua ujungnya ditutup, dimana sepanjang kabel terdapat *node-node*
b. Umum digunakan karena sederhana dalam instalasis
c. *Signal* melewati kabel dalam dua arahdan mungkin terjadi *collision*
d. Problem terbesar pada saat kabel putus. Jika salah satu segmen kabel putus, maka seluruh jaringan akan terhenti.

2. *Ring*

Topologi ini mempuyai karakteristik sebagai berikut :

a. Lingkaran tertutup yang berisi *node-node*
b. Sederhana dalam layout
c. *Signal* mengalir dalam satu arah, sehingga dapat menghindarkan terjadinya *collision* (dua paket data bercampur), sehingga memungkinkan pergerakan data yang cepat dan *collision detection* yang lebih sederhana

d. Problem sama dengan topologi *bus*

e. Biasanya topologi *ring* tidak dibuat secara fisik melainkan direalisasikan dengan sebuah *consentrator* dan kelihatan seperti topologi *star*

3. *Star*

Topologi ini mempunyai karakteristik sebagai berikut :

a. Setiap *node* berkomunikasi langsung dengan *central node*, *traffic data* mengalir dari *node* ke *central node* dan ketnbali lagi.

b. Mudah dikembangkan, karena setiap *node* hanya memiliki kabel yang langsung terhubung ke *central node*.

c. Keunggulannya adalah jika satu kabel *node* terputus yang lainnya tidak terganggu.

d. Dapat digunakan kabel yang "*lower grade*" karena hanya menangani satu *traffic node*, biasanya digunakan kabel *UTP*.

4. *Extended Star*

Topologi *Extended Star* merupakan perkembangan lanjutan dari topologi *star* dimana karakteristiknya tidak jauh berbeda dengan topologi *star* yaitu :

a. Setiap *node* berkomunikasi langsung dengan *subnode*, sedangkan *subnode* berkomunikasi dengan *central node*. *Traffic data* mengalir dari *node* ke *subnode* lalu diteruskan ke *central node* dan kembali lagi.

b. Digunakan pada jaringan yang besar dan membutuhkan penghubung yang banyak atau melebihi dari kapasitas maksimal penghubung.

c. Keunggulannya jika satu kabel *sub node* terputus maka *sub node* yang lainnya tidak terganggu, tetapi apabila *central node* terputus maka semua *node* di setiap *sub node* akan terputus

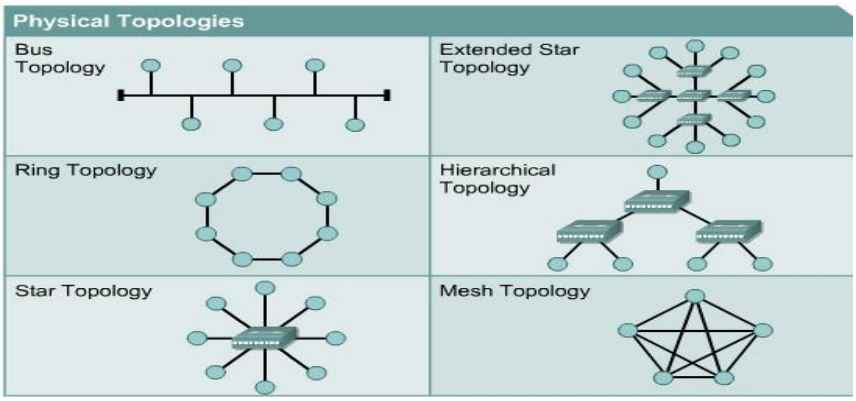
d. Tidak dapat digunakan kabel yang "*lowergrade*" karena hanya menangani satu *traffic node*, karena untuk berkomunikasi antara satu *node* ke *node* lainnya membutuhkan beberapa kali *hops*.

5. *Hierarchical topology*

Topologi ini biasa disebut sebagai topologi *tree.* Dibangun oleh seperti halnya topologi *extended star* yang dihubungkan melalui *sub node* dalam satu *central node*. Topologi ini dapat men*support* baik *baseband* maupun *broadband signaling* dan juga men*support* baik *contention* maupun *token bus access*.

6. *Mesh*

Mesh topologi dibangun dengan memasang *link* diantara *station-station*. Sebuah '*fully-connected mesh'* adalah sebuah jaringan dimana setiap terminal terhubung secara langsung ke semua terminal-terminal yang lain. Biasanya digunakan pada jaringan komputer kecil. Topologi ini secara teori memungkinkan akan tetapi tidak praktis dan biayanya cukup tinggi untuk di-implementasikan. *Mesh* topologi memiliki tingkat redundansi yang tinggi. Sehingga jika terdapat satu *link* yang rusak maka suatu *station* dapat mencari *link* yang lainnya.



Gambar 2.16 Jenis-jenis Topologi

2.8 Jaringan *Peer to Peer*

Jaringan komputer P2P termasuk sebuah cabang (subset) dari bidang komputasi terdistribusi. Namun komputasi terdistribusi sendiri bukanlah cabang dari P2P. Sebutan "*peer-to-peer*" mengisyaratkan sebuah hubungan kesetaraan (*egalitarian relationship*) diantara para *peer* (baca : pengguna satu dengan yang lainnya). Dan yang terpenting, hubungan ini haruslah menghasilkan interaksi langsung antara komputer pengguna yang satu dengan komputer pengguna lainnya. Tanpa embel-embel ada komputer yang berstatus sebagai *client* dan berstatus sebagai *server* [8].

Secara teknis, jaringan P2P (*peer-to-peer*) adalah sebuah jaringan yang memungkinkan semua komputer dalam lingkungannya bertindak/berstatus sebagai *server* yang memiliki kemampuan untuk mendistribusikan sekaligus menerima berkas-berkas atau sumber daya (*resource*) yang ada dalam komputer mereka ke komputer lainnya.

Jaringan bertipe ini sangat banyak dijumpai di kantor-kantor yang tidak membutuhkan sebuah sentral pengaturan laiknya jaringan *client-server*. Di *internet*, jaringan P2P hidup dan berkembang melalui aplikasi-aplikasi populer seperti *Napster* dan *Gnutella* [8].



Gambar 2.17 Model Jaringan *Peer to Peer*

Keuntungan menggunakan jaringan *peer to peer* adalah :

1. Tidak memerlukan investasi tambahan untuk pembelian *hardware* dan *software server*.
2. Tidak diperlukan seorang *network administrator* dan *setup*nya. mudah serta meminta biaya yang murah.

Kerugian menggunakan jaringan *peer to peer* adalah :

1. *Sharing* sumber daya pada suatu komputer di dalam jaringan akan sangat membebani komputer tersebut.
2. Masalah lain adalah kesulitan dalam mengatur *file-file*. *User* harus menangani komputernya sendiri jika ditemui masalah keamanan sangat lemah.

2.9 Kompeksitas Algoritma

Algorima adalah urutan logis langkah-langkah penyelesaian masalah secara sistemastis. Sebuah algoritma tidak saja harus benar, tetapi juga harus mangkus (efisien). Algoritma yang benar sekalipun mungkin tidak berguna untuk menjalankan algoritma tersebut atau ruang memori yang diperlukan untuk struktur datanya terlalu besar. Jika kita tulis aloritmanya, maka algoritma harus menguji semua himpunan bagian dan memeriksa apakah *x* merupakan anggota himpunan bagian tersebut [10].

2.9.1 Kebutuhan Waktu dan Ruang

Kebutuhan waktu suatu algoritma biasanya dihitung dalam satuan *detik/mikrodetik*, dan sebagainya, sedangkan ruang memori yang digunakannya dapat dihitung dalam satuan *byte* atau *kilobyte*.

Biasanya orang mengukur kebutuhan waktu sebuah algoritma degan mengeksekusi langsung algoritma tersebut pada sebuah komputer, lalu dihitung berapa lama durasi waktu yang dibutuhkan untuk menyelesaikan sebuah persoalan dengan *n* yang berbeda-beda.

Sebagai ilustrasi, tinjau masalah menghitung rata-rata dari *n* buah data bilanan bulat. Kita mengasumsikan data masukan sudah dibaca dan disimpan di dala elemen-elemen larik (tabel) a_1, a_2, ..., a_n [10].

2.9.2 Kompleksitas Waktu dan Ruang

Secara teoritis, model abstrak pengukuran waktu/ruang harus independen dari pertimbangan mesin dan *compiler* apapun. Model abstrak seperti itu dapat dipakai untuk membandingkan algoritma yang berbeda. Besaran yang dipakai

untuk menerangkan model abstrak pengukuran waktu/ruang ini adalah kompleksitas algoritma. Ada dua macam kompleksitas algoritma, yaitu kompleksitas waktu dan kompleksitas ruang [10].

2.9.3 Terminologi Kompleksitas Waktu/Ruang

Terminologi yang diperlukan dalam membahas kompleksitas waktu dan kompleksitas ruang suatu algoritma adalah [10]:

1. Ukuran besar masukan data untuk suatu algoritma, *n*.
2. Kompleksitas waktu, *T(n)*, adalah jumlah operasi yang dilakukan untuk melaksanakan algoritma sebagai fungsi dari ukuran masukan *n*.
3. Kompleksitas ruang, *S(n)*, adalah ruang memori yang dibutuhkan algoritma sebagai fungsi dari ukuran masukan *n*.

2.9.3.1 Kompleksitas Waktu

Setelah menetapkan ukuran masukan, maka langkah selanjutnya dalam mengukur kompleksitas waktu adalah menghitung banyaknya operasi yang dilakukan oleh algoritma. Di dalam sebuah algoritma mungkin terdapat banyak sekali jenis-jenis operasi, misalnya operasi penjumlahan (termasuk pengurangan), operasi perbandingan, operasi pembagian, operasi pembacaan, pemanggilan prosedur dan sebagainya [10].

Jenis-jenis operasi yang terdapat di dalam algoritma HitungRerata adalah :

1. Operasi pengisian nilai (dengan operator "←")

jumlah ← 0,	1 kali
k ← 1,	1 kali
jumlah ← jumlah + $a_{k,}$	*n* kali

k ← k + 1, *n* kali

r ← jumlah/n), 1 kali

jumlah seluruh operasi pengisian nilai adalah

$t_1 = 1 + 1 + n + n + 1 = 3 + 2n$

2. Operasi penjumlahan (dengan operator "+")

jumlah + $a_{k,}$ *n* kali

k + 1, *n* kali

jumlah seluruh operasi penjumlahan adalah

$t_2 = n + n = 2n$

3. Operasi pembagian (dengan operator "/")

Jumlah seluruh operasi pembagian adalah

Jumlah/*n*) 1 kali

Dengan demikian, kompleksitas waktu algoritma diitung berdasarkan jumlah operasi aritmetika dan operasi pengisian nilai adalah

$T(n) = t_1 + t_2 + t_3 = 3 + 2n + 2n + 1 = 4n + 4$

2.10 Pendekatan Rekayasa Perangkat Lunak

Proses perangkat lunak telah menjadi perhatian yang serius selama decade terakhir. Proses perangkat lunak menentukan pendekatan yang digunakan ketika perangkat lunak dikembangkan, tetapi pengembangan perangkat lunak juga meliputi teknologi yang mempopulasikan proses, metode, teknis, serta alat-alat otomatis.

Pada tugas akhir ini akan dibangun suatu perangkat lunak melalui pendekatan rekayasa perangkat lunak dengan mengambil suatu metode tertentu untuk pengembangannya.

2.10.1 Pengertian Rekayasa Perangkat Lunak

Pengertian Rekayasa Perangkat Lunak menurut *Roges Pressman S.* pada konferensi seminar, Rekayasa Perangkat Lunak didefinisikan sebagai berikut :

> Rekayasa Perangkat Lunak adalah pengembangan suara untuk memperoleh perangkat lunak secara ekonomis yang *reliable* dan bekerja secara efisien pada mesin nyata.

Setelah ada pengembangan yang lebih komprehensif definisi Rekayasa Perangkat Lunak adalah dinyatakan sebagai berikut (*Pressman*:1977) :

1. Aplikasi dari sebuah pendekatan kuantifiabel, disiplin, dan sistematis kepada pengembangan, operasi, dan pemeliharaan perangkat lunak, yaitu aplikasi dari Rekayasa Perangkat Lunak
2. Studi tentang pendekatan-pendekatan seperti pada (1).

Rekayasa perangkat lunak merupakan teknologi yang terdiri dari beberapa elemen (*Pressman*:1977) yaitu :

1. Fokus Kualitas

Manajemen kualitas total serta filosofinya mengangkat budaya pengembangan proses yang terus-menerus, dan budaya itu sendiri membawa

kepada perkembangan pendekatan yang semakin mantap terhadap rekayasa perangkat lunak. Baru landasan yang menopang rekayasa perangkat lunak merupakan fokus kepada kualitas.

2. Proses

Proses bertindak sebagai perekat yang mempertahankan teknologi dan memungkinkan pengembangan perangkat lunak menjadi rasional dan tepatwaktu. Selain itu, proses juga mendefinisikan kerangka yang harus didirikan agar penyerahan teknologi rekayasa perangkat lunak menjadi efektif.

3. Metode

Menyediakan teknik bagaimana membangun perangkat lunak, meliputi tugas-tugas yang mencakup analisis, perancangan, implementasi (pemrograman), pengujian, dan pemeliharaan.

4. Alat Bantu

Alat bantu yang digunakan saling terintegrasi sehingga informasi yang dihasilkan oleh suatu alat bantu dapat digunakan alat bantu lainnya.

2.10.2 Model Sekuensial Linear

Sekuensial linear mengusulkan sebuah pendekatan kepada pengembangan perangkat lunak yang sistematik dan sekuensial yang mulai pada tingkat dan kemajuan sistem pada seluruh analisis, desain, kode, pengujian, dan pemeliharaan.

Dimodelkan setelah siklus rekayasa konvensional, model sekuensial linear melingkupi aktifitas-aktifitas sebagai berikut :

1. Rekayasa dan pemodelan sistem informasi

   Karena perangkat lunak selalu merupakan bagian dari sebuah istem yang lebih besar, kerja dimulai dengan membangun syarat dari semua elemen sistem dan menggunakan beberapa subset dari kebutuhan ke perangkat lunak tersebut. Pandangan sistem ini penting ketika perangkat lunak harus berhubungan dengan elemen-elemen lain seperti perangkat lunak, manusia, dan *database*.

2. Analisis kebutuhan perangkat lunak

   Proses pengumpulan kebutuhan diintensifkan dan difokuskan, khususnya pada perangkat lunak. Untuk memahami sifat program yang dibangun,rekayasa perangkat lunak (analisis) harus memahami domain informasi, tingkah laku, unjuk kerja, dan antarmuka (interface) yang diperlukan. Kebutuhan baik sistem maupun perangkat lunak didokumentasikan dan dilihat lagi.

3. Desain

   Desain perangkat lunak sebenarnya adalah proses multi langkah yang berfokus pada empat atribut sebuah program yang berbeda, struktur data, arsitektur perangkat lunak, representasi *interface*, dan *detail* (algoritma) *object oriented programming* (*OOP*). Proses desain menterjemahkan

syarat atau kebutuhan kedalam sebuah representasi perangkat lunak yang dapat diperkirakan demi kualitas sebelum dimulai pemunculan kode.

4. Generasi kode

Desain harus diterjemahkan kedalam bentuk mesin yang bisa dibaca. Langkah pembuatan kode melakukan tugas ini. Jika desain dilakukan dengan cara yang lengkap, pembuatan kode dapat diselesaikan secara mekanis.

5. Pengujian

Sekali kode dibuat, pengujian program dimulai. Proses pengujian berfokus pada logika internal perangkat lunak, memastikan bahwa semua persyaratan sudah diuji, dan pada eksternal fungsional yaitu mengarahkan pengujian untuk menemukan kesalahan-kesalahan dan memastikan bahwa *input* yang dibatasi akan memberikan hasil aktual yang sesuai dengan yang dibutuhkan. Seperti pengujian dengan menggunakan metode *Black Box testing*. Metode *Black Box Testing* ini merupakan pengujian program berdasarkan fungsi dari program. Tujuan dari metode *Black Box Testing* ini adalah untuk menemukan kesalahan fungsi pada program.

6. Pemeliharaan

Perangkat lunak akan mengalami perubahan setelah dipakai sekian lama seiring dengan kebutuhan. Perubahan akan terjadi karena perangkat lunak harus disesuaikan dengan mengakomodir perubahan-perubahan

lingkungan eksternal atau karena perkembangan fungsional serta unjuk kerjanya.

2.11 Metode Analisis Perancangan Terstruktur

Berikut ini akan akan dipaparkan metode analisis perancangan terstruktur dalam pembangunan sistem kriptografi ini.

2.11.1 *Data Flow Diagram* (DFD)

Data Flow Diagram(DFD) adalah suatu model logika atau proses yang dibuat untuk menggambarkan darimana asal data dan kemana tujuan data yang keluar dari sistem, yang mana data disimpan, proses apa yang menghasilkan data tersebut dan interaksi antara data yang tersimpan dan proses yang dikenakan pada data tersebut [9].

Data Flow Diagram(DFD) menggambarkan proses penyimpanan data dan proses yang mentransformasikan data. *Data Flow Diagram*(DFD) menunjukan hubungan antara data pada sistem dan proses pada sistem (Kristanto, 2008).

2.11.2 *Flowchart*

Flowchart (bagan aliran) adalah gambar yang menggunakan lambang-lambang baku untuk menggambarkan sistem atau proses. *Flowchart* memiliki beberapa lambang yang sudah biasa digunakan dalam pengembangan sistem, baik dalam sistem manual maupun sistem komputerisasi [9].

Flowchart sistem menunjukkan asal dokumen, tujuan dokumen, kegunaan dokumen, dan berbagai tindakan yang diperlukan sehubungan dengan aliran dokumen tersebut. Flowchart ini hanya menggunakan lambang dokumen.

Flowchart dokumen bermanfaat untuk menganalisis pengendalian suatu sistem atau menganalisis pemisahan wewenang dan tanggung jawab.

Flowchart dokumen dibuat secara berkolom dan masing-masing kolom mewakili suatu unit atau entitas. Hubungan antar unit yang sekaligus menunjukkan arus dokumen ditunjukkan dengan tanda panah.

2.12 Pemilihan BahasaPemrograman

Pemrograman merupakan kegiatan menulis kode program yang akan dikompiler oleh bahasa pemrograman. Penulisan kode program merupakan kegiatan yang paling utama dalam tahap implementasi sistem. Untuk mengimplementasikan dalam versi *windows*. Versi *windows* akan menggunakan bahasa menggunakan bahasa pemrograman terstruktur yaitu *Visual Basic* [1].

2.12.1 Ruang Lingkup *Visual Basic*

Visual Basic adalah salah suatu *developement tools* untuk membangun aplikasi dalam lingkungan *Windows*. Dalam pengembangan aplikasi, *Visual Basic* menggunakan pendekatan *visual* untuk merancang *user interface* dalam bentuk *form*, sedangkan untuk kodingnya menggunakan dialek bahasa *basic* yangcenderung mudah dipelajari. *Visual Basic* telah menjadi *tools* yang terkenal bagi para pemula maupun para *developer*. *Visual Basic* adalah bahasa pemrograman berbasis *Microsoft Windows* yang merupakan terstruktur, yaitu pemrograman berorientasi prosedural, *Visual Basic* menyediakan objek objek yang sangat kuat, berguna dan mudah [1].

Dalam lingkungan *Windows, User-interface* sangat memegang peranan penting, karena dalam pemakaian aplikasi yang kita buat, pemakai senantiasa

berinteraksi dengan *User-interface* tanpa menyadari bahwa dibelakangnya berjalan instruksi-instruksi program yang mendukung tampilan dan proses yang dilakukan.

Pada pemrograman *Visual*, pengembangan aplikasi dimulai dengan pembentukkan *user interface*, kemudian mengatur properti dari objek-objek yang digunakan dalam *user interface*, dan baru dilakukan penulisan kode program untuk menangani kejadian-kejadian (*event*). Tahap pengembangan aplikasi demikian dikenal dengan istilah pengembangan aplikasi dengan pendekatan *BottomUp*.