BAB II

LANDASAN TEORI

2.1 Kriptografi

Kriptografi adalah sebuah studi tentang teknik matematika yang berhubungan dengan aspek-aspek keamanan informasi seperti kerahasiaan data, integritas data, otentikasi data dan asal data. Kriptografi membahas tentang pencegahan dan deteksi kecurangan dan aktivitas mencurigakan lainnya. Kriptografi memiliki beberapa tujuan, yaitu (Menezes dkk, 1996):

1. *Confidentiality*, yaitu menjaga agar isi dari informasi tetap bersifat rahasia dari semua kecuali mereka yang berwenang memilikinya.
2. *Data integrity*, yaitu memasikan bahwa informasi tidak dimanipulasi oleh pihak yang tak berwenang. Yang termasuk manipulasi data adalah penambahan, penghapusan, dan substitusi data.
3. *Authentication*, yaitu mengidentifikasi informasi serta pengirimnya.
4. *Non-repudiation*, yaitu mencegah sebuah kesatuan menyangkal tindakan yang dilakukan olehnya pada informasi terkait.

Penggunaan kriptografi mempunyai sejarah yang cukup panjang. Julius Caesar diketahui menggunakan *substitution cipher* untuk mengenkripsi pesan rahasia. Di jaman yang lebih modern, kriptografi digunakan oleh militer untuk merahasiakan perintah strategis melalui jalur komunikasi radio. Pesan rahasia antara pemerintahan suatu negara dengan misi diplomatiknya juga diamankan menggunakan kriptografi.

Menurut Kromodimoeljo, kriptografi tidak hanya dibutuhkan oleh militer dan misi diplomatik. Kemajuan teknologi komunikasi dan komputer membuat kriptografi dibutuhkan oleh kalangan yang lebih luas, bahkan boleh dikatakan bahwa semua orang yang menggunakan internet menggunakan kriptografi, meski sering tanpa disadari (Kromodimoeljo, 2010).

2.1.1 Enkripsi

Enkripsi merupakan salah satu cara praktis yang dapat digunakan untuk mencapai kerahasiaan informasi. Teknik enkripsi modern menggunakan transformasi matematika yang memperlakukan informasi sebagai angka atau elemen aljabar dalam sebuah ruang dan mengubahnya menjadi pesan tak bermakna. Untuk mengembalikan informasi menjadi pesan bermakna, transformasi enkripsi harus bersifat reversibel dan transformasi pembalikan ini disebut sebagai dekripsi (Mao, 2003).

Sistem kriptografi dapat dibedakan berdasarkan 3 dimensi yang independen (Stallings, 2010):

- Tipe operasi yang digunakan untuk melakukan proses transformasi. Semua algoritma enkripsi dibangun berdasarkan 2 prinsip dasar, yaitu substitusi, dimana tiap elemen pada *plaintext* akan diubah menjadi elemen lain, dan transposisi, dimana tiap elemen pada *plaintext* akan diubah urutannya. Kebutuhan utama yang harus dipenuhi adalah tidak ada data yang hilang dalam proses atau seluruh operasi bersifat reversibel.

- Jumlah *key* yang digunakan. Apabila *key* yang digunakan *sender* dan *receiver* sama, maka sistem tersebut disebut sebagai enkripsi simetris. Apabila *key* yang digunakan berbeda, sistem disebu sebagai enkripsi asimetris.
- Cara *plaintext* diproses. Sebuah *block cipher* akan memroses masukan dalam bentuk blok kumpulan elemen pada satu waktu, dan menghasilkan keluaran blok berukuran sama untuk tiap blok masukan. Sedangkan stream cipher memroses elemen masukan secara kontinu dan menghasilkan sebuah elemen pada satu waktu.

Sebuah enkripsi simetris memiliki 5 unsur, yaitu (Stallings, 2010):

- *Plaintext* : Data atau pesan asli yang akan dijadikan masukan pada algoritma enkripsi yang digunakan.
- *Encryption Algorithm* : Algoritma enkripsi yang akan melaksanakan transformasi dan substitusi pada *plaintext*.
- *Secret Key* : Masukan lainnya untuk algoritma enkripsi. *Key* tersebut bersifat independen dari *plaintext* maupun algoritma yang digunakan. Algoritma akan mampu menghasilkan keluaran yang berbeda tergantung dari *key* yang digunakan.
- *Ciphertext* : Keluaran dari proses enkripsi yang merupakan pesan yang sudah diacak berdasarkan *plaintext* dan *secret key*. Untuk sebuah *plaintext* yang sama, penggunaan 2 *key* yang berbeda akan menghasilkan 2 *ciphertext* yang berbeda. *Ciphertext* terlihat sebagai aliran data yang acak dan tidak dapat dimengerti.

– *Decryption algorithm* : Pada dasarnya, ini merupakan algoritma enkripsi yang berjalan terbalik. Menggunakan *ciphertext* dan *key* sebagai masukan dan menghasilkan *plaintext*.

Penggunaan *key* pada metode enkripsi sangatlah penting. Apabila transformasi enkripsi berhasil diketahui, seorang pengguna tidak perlu merancang ulang seluruh metode, cukup mengubah *key* yang digunakan. Penggunaan *key* yang berbeda merupakan praktik yang baik pada kriptografi. Sebuah metode enkripsi disebut *breakable* apabila pihak ketiga mampu mendapatkan *plaintext* dari sebuah *ciphertext* tanpa memerlukan *key* dalam jangka waktu tertentu (Menezes dkk, 1996).

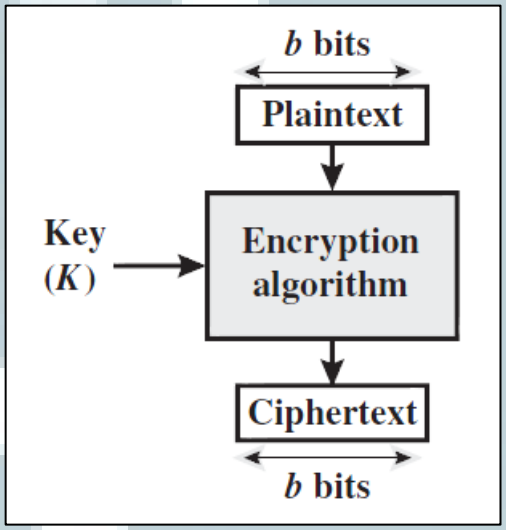
Terdapat 2 pendekatan umum dalam menyerang sebuah sistem enkripsi konvensional, yaitu (Stallings, 2010):

– Kriptanalisis : Serangan kriptanalisis bergantung pada sifat dari algoritma ditambah dengan sedikit pengetahuan akan karakteristik umum dari *plaintext* ataupun sepasang *plaintext* dan *ciphertext* terkait. Jenis serangan ini memanfaatkan karakteristik dari algoritma untuk menyimpulkan sebuah *plaintext* yang spesifik atau *key* yang digunakan.

– *Brute force attack* : Penyerang mencoba seluruh kombinasi *key* yang memungkinkan pada sebuah *ciphertext* sampai *plaintext* hasil terjemahan yang dapat dipahami berhasil diperoleh. Pada umumnya, setengah dari seluruh kombinasi *key* yang memungkinkan harus dicoba untuk berhasil memecahkannya.

2.1.2 Block Cipher

Block cipher adalah metode enkripsi dimana sebuah blok *plaintext* diperlakukan sebagai satu kesatuan dan digunakan untuk menghasilkan sebuah blok *ciphertext* dengan ukuran yang sama. Umumnya, ukuran blok yang digunakan adalah 64 dan 128 bit (Stallings, 2010).



Gambar 2.1 Block Cipher (Stallings, 2010)

Sebuah *block cipher* beroperasi pada blok *plaintext* berukuran n *bits* untuk menghasilkan blok *ciphertext* berukuran n *bits*. Terdapat 2^n blok *plaintext* yang unik dan agar enkripsi tersebut bersifat reversibel, tiap blok harus mampu menghasilkan sebuah blok *ciphertext* yang unik.

Block cipher seperti DES/3DES, CAST, IDEA dan AES menggunakan efek *diffusion* dan *confusion* untuk mempersulit analisa statistik. Ini dilakukan menggunakan apa yang disebut *Feistel Network* atau *substitution permutation network*. DES merupakan standar enkripsi pertama yang menggunakan konsep

Feistel network dan algoritma DES masih digunakan oleh 3DES yang dianggap cukup kuat untuk penggunaan masa kini (Kromodimoeljo, 2010).

Feistel mengusulkan bahwa *block cipher* ideal dapat dicapai dengan menggunakan konsep dari *product cipher*, yaitu penggunaan 2 atau lebih *cipher* sederhana dalam suatu urutan dan menghasilkan produk yang lebih kuat dibanding produk dari komponen *cipher* yang membangunnya. Secara khusus, Feistel mengusulkan penggunaan *cipher* yang melakukan substitusi dan permutasi secara bergantian. Kedua istilah tersebut dapat didefinisikan sebagai berikut:

- Substitusi : Setiap elemen *plaintext* akan ditukar dengan elemen *ciphertext* yang sesuai.
- Permutasi : Sebuah rangkaian elemen *plaintext* ditukar dengan permutasi dari rangkaian tersebut. Dengan demikian, tidak ada elemen yang ditambahkan, dihapus, atau ditukar pada rangkaian tersebut, hanya urutan kemunculan elemen yang berubah.

Feistel cipher merupakan aplikasi praktis dari proposal oleh Claude Shannon untuk membangun *product cipher* yang memiliki fungsi *confusion* dan *diffusion*. Pada *diffusion*, struktur statistik dari sebuah *plaintext* diacak. Ini dicapai dengan membuat tiap digit *plaintext* mempengaruhi nilai dari banyak digit *ciphertext*. Pada *block cipher*, *diffusion* dapat diperoleh dengan melakukan permutasi pada data secara berulang dan diikuti dengan penerapan sebuah fungsi pada permutasi tersebut.

Setiap block cipher melibatkan sebuah transformasi blok *plaintext* menjadi blok *ciphertext*, dimana transformasi tersebut bergantung pada nilai *key*. *Diffusion*

bertujuan untuk membuat hubungan statistik antara *plaintext* dan *ciphertext* serumit mungkin agar mencegah usaha untuk menyimpulkan *key*. Pada sisi lain, *confusion* bertujuan untuk membuat hubungan statistik antara *ciphertext* dan nilai *key* serumit mungkin, untuk menggagalkan usaha menemukan *key*. Hal ini dicapai dengan menggunakan algoritma substitusi yang kompleks.

Realisasi dari sebuah *feistel network* tergantung pada pilihan parameter berikut: (Stallings, 2010)

- Ukuran blok: Semakin besar ukuran blok, semakin baik tingkat keamanan yang dihasilkan, namun memperlambat kecepatan enkripsi dan dekripsi.
- Ukuran *key*: Semakin besar ukuran *key*, semakin baik tingkat keamanan yang dihasilkan.
- Jumlah putaran: Semakin banyak putaran, semakin baik tingkat keamanan yang diberikan. Jumlah yang umum digunakan adalah 16 putaran.
- *Subkey generation algorithm*: Semakin rumit algoritma yang digunakan, semakin sulit untuk melakukan kriptanalisis.
- *Round function F*: Semakin rumit fungsi yang digunakan, semakin resisten terhadap kriptanalisis.

2.1.3 Mode operasi Block Cipher

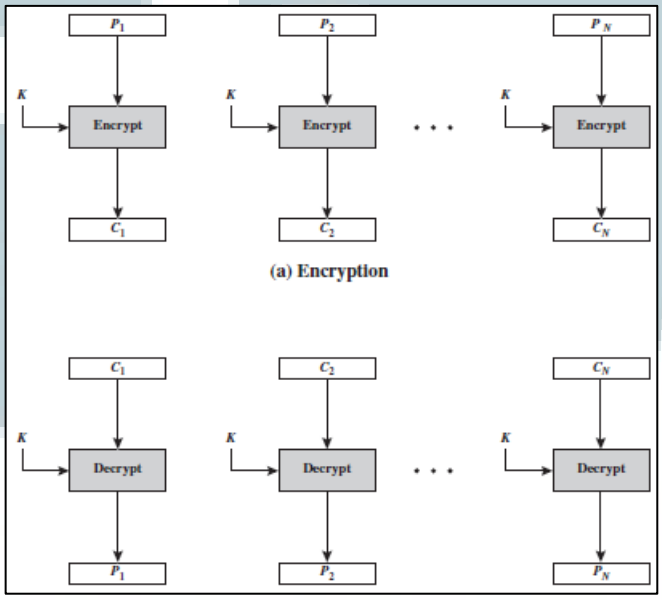
Menurut Dworkin (2001), terdapat 5 mode operasi standar dalam menggunakan *block cipher*, yaitu sebagai berikut.

A. Mode Electronic Codebook

Mode *Electronic Codebook* (ECB) adalah sebuah mode operasi *block cipher* yang melakukan operasi terhadap tiap blok *plaintext* secara independen ke tiap blok *ciphertext* masing-masing. Mode ECB dapat didefinisikan seperti pada rumus 2.1 dan 2.2.

Enkripsi ECB: $Cj = CIPH_K(Pj)$ untuk j = 1… n … rumus 2.1

Dekripsi ECB: $Pj = CIPH^{-1}_K(Cj)$ untuk j = 1… n … rumus 2.2



Gambar 2.2 Mode Operasi ECB (Stallings, 2010)

Pada enkripsi dan dekripsi ECB, fungsi *cipher* dapat dilakukan secara paralel. Penggunaan mode ECB membuat sebuah *plaintext* tertentu yang dienkripsi dengan *key* tertentu akan selalu menghasilkan *ciphertext* yang sama (Dworkin, 2001).

Mode ECB ideal untuk digunakan pada data berukuran kecil, seperti *key*. Karakteristik utama dari mode ECB adalah jika blok yang sama muncul lebih dari

sekali, *cipher* akan selalu menghasilkan blok *ciphertext* yang sama. Hal ini membuat mode ECB tidak cukup aman untuk pesan yang panjang dan memiliki struktur tertentu, sehingga memberikan kriptanalis kesempatan untuk mengubah atau menyusun ulang blok. (Stallings, 2010)

B. Mode Cipher Block Chaining

Mode *Cipher Block Chaining* (CBC) merupakan sebuah mode operasi yang melaksanakan proses enkripsi dengan melibatkan hasil XOR dari blok *plaintext* dengan blok *ciphertext* sebelumnya. Mode CBC membutuhkan sebuah *Initialization Vector* (IV), blok data sebagai masukan yang akan di-XOR dengan blok plaintext pertama sebelum dienkripsi. IV tersebut tidak harus bersifat rahasia dan dapat ditransmisikan bersama *ciphertext*. Mode CBC dapat didefinisikan seperti pada rumus 2.3 dan 2.4.

Enkripsi CBC: $C_1 = CIPH_K (P_1 \oplus IV)$;

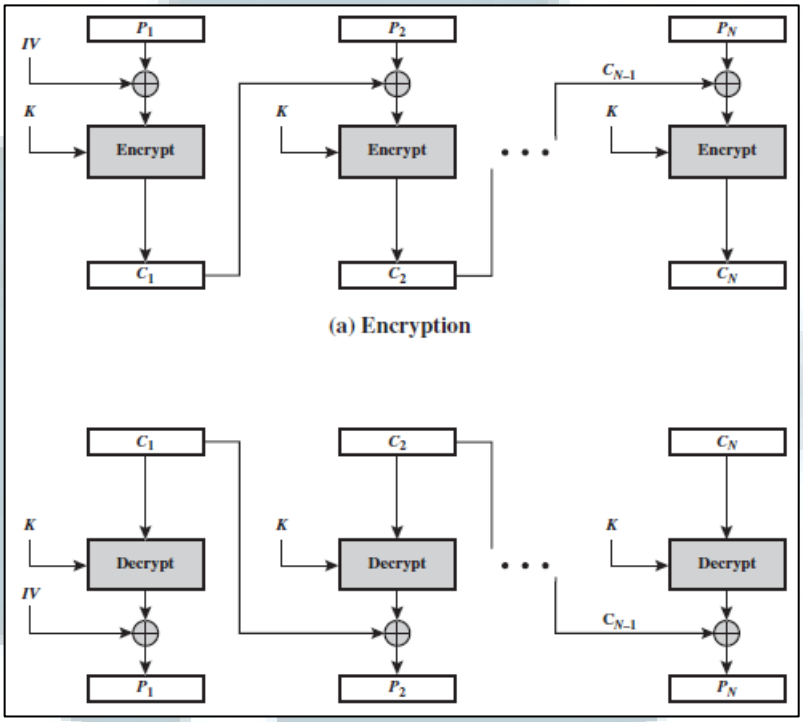
$Cj = CIPH_K(Pj \oplus C_{j-1})$ untuk j = 2…n; …rumus 2.3

Dekripsi CBC: $P1 = CIPH^{-1}_K(C_1) \oplus IV$;

$Pj = CIPH^{-1}_K(Cj) \oplus C_{j-1}$ untuk j = 2…n; …rumus 2.4

IV yang digunakan harus diketahui oleh *sender* dan *receiver*, namun tidak dapat diprediksi oleh pihak ketiga sebelum IV diciptakan. Terdapat 2 metode untuk menciptakan sebuah IV yang tidak dapat ditebak. Metode pertama adalah dengan menggunakan fungsi *forward cipher* terhadap sebuah *nonce* dengan menggunakan *key* yang sama seperti yang digunakan untuk mengenkripsi *plaintext*. *Nonce* adalah nilai yang hanya digunakan sekali dan bersifat unik pada

setiap operasi enkripsi. Sedangkan metode kedua adalah dengan menciptakan sebuah data *random* menggunakan *random number generator* yang disetujui oleh FIPS (*Federal Information Processing Standard*) (Dworkin, 2001).



Gambar 2.3 Mode Operasi CBC (Stallings, 2010)

C. Mode Cipher Feedback

Cipher Feedback (CFB) adalah mode operasi yang menggunakan *feedback* dari sebagian *ciphertext* sebagai masukan forward cipher untuk menghasilkan blok *output* yang akan di XOR dengan *plaintext* untuk menghasilkan *ciphertext*, dan sebaliknya. Mode ini juga memerlukan parameter *integer*, dinotasikan sebagai *s*, dimana $1 \leq s \leq b$ (ukuran blok). Rumus 2.5 dan 2.6 adalah definisi enkripsi dan dekripsi dari mode CFB.

Enkripsi CFB : $I_1 = IV;$

$I_j = LSB_{b-s}(I_{j-1}) \mid C^{\#}_{j-1}$ untuk j = 2 … n;

$O_j = CIPH_K(I_j)$ untuk j = 1, 2 … n;

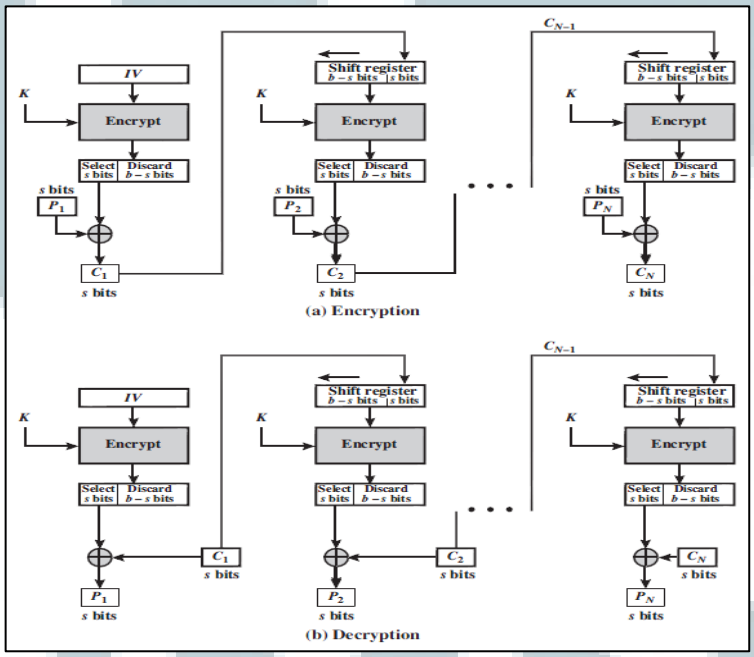
$C^{\#}_j = P^{\#}_j \oplus MSB_s(O_j)$ untuk j = 1, 2 … n; … rumus 2.5

Dekripsi CFB : $I_1 = IV;$

$I_j = LSB_{b-s}(I_{j-1}) \mid C^{\#}_{j-1}$ untuk j = 2 … n;

$O_j = CIPH_K(I_j)$ untuk j = 1, 2 … n;

$P^{\#}_j = C^{\#}_j \oplus MSB_s(O_j)$ untuk j = 1, 2 … n; … rumus 2.6



Gambar 2.4 Mode operasi CFB (Stallings, 2010)

Pada CFB, IV digunakan sebagai *input* pertama dan operasi cipher dijalankan menggunakan IV untuk menghasilkan blok *output* pertama. Ciphertext pertama dihasilkan dengan melakukan XOR terhadap bagian *plaintext* pertama dengan *s most significant bits* dari blok *output* pertama. *Least significant bits* dari

IV berukuran b-s digabungkan dengan s bit *ciphertext* pertama untuk membentuk blok *input* kedua. Proses ini diulang secara berurutan terhadap blok *input* sampai seluruh bagian *plaintext* menghasilkan bagian *ciphertext* masing-masing.

D. Mode Output Feedback

Mode *Output Feedback* (OFB) menggunakan iterasi *cipher* terhadap sebuah IV untuk menciptakan blok *output* yang di XOR dengan *plaintext* untuk menghasilkan *ciphertext*, dan sebaliknya. Mode OFB membutuhkan sebuah IV yang bersifat unik pada setiap eksekusi (*nonce*). OFB dapat didefinisikan seperti pada rumus 2.7 dan 2.8.

Enkripsi OFB: $I_1 = IV;$

$I_j = O_{j-1}$ untuk j = 2 … n;

$O_j = CIPH_K(I_j)$ untuk j = 1, 2 … n;

$C_j = P_j \oplus O_j$ untuk j = 1, 2 … n-1;

$C^*_n = P^*_n \oplus MSB_u(O_n);$ … rumus 2.7

Dekripsi OFB: $I_1 = IV;$

$I_j = O_{j-1}$ untuk j = 2 … n;

$O_j = CIPH_K(I_j)$ untuk j = 1, 2 … n;

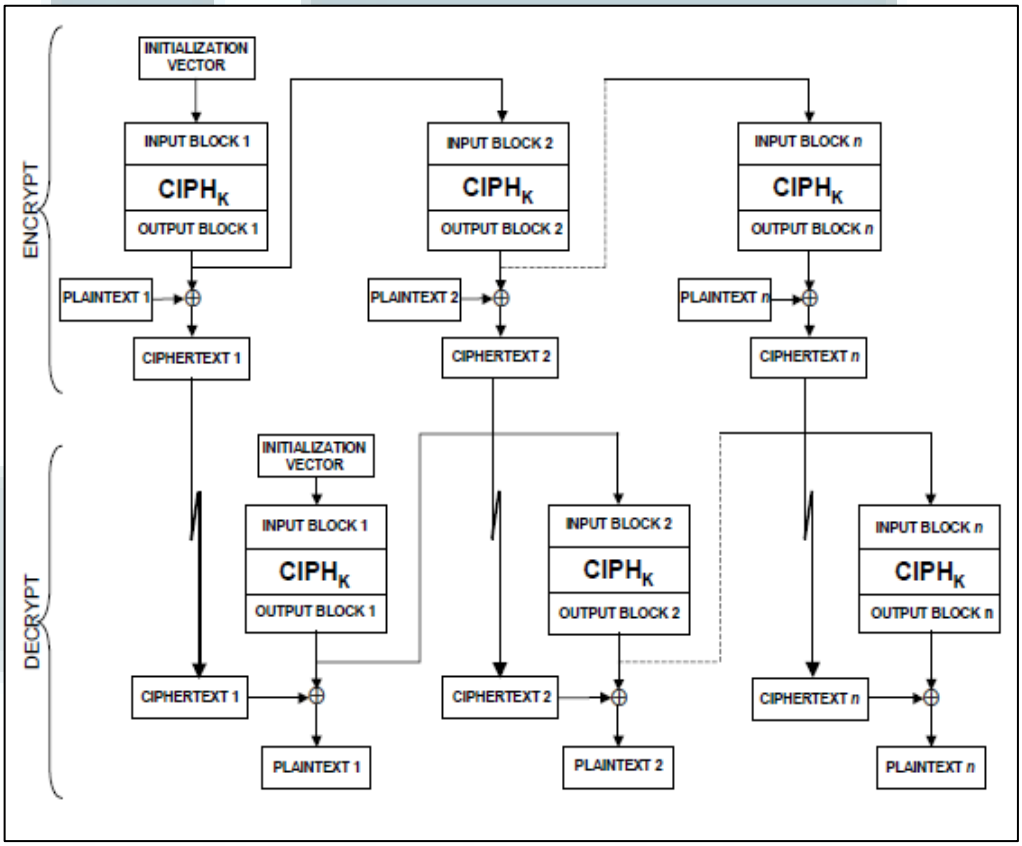
$P_j = C_j \oplus O_j$ untuk j = 1, 2 … n-1;

$P^*_n = C^*_n \oplus MSB_u(O_n);$ … rumus 2.8

Pada proses enkripsi OFB, IV akan ditransformasi menggunakan *forward cipher* untuk menghasilkan blok output pertama, yang kemudian akan di XOR dengan blok *plaintext* pertama untuk menghasilkan blok *ciphertext* pertama. Blok

output pertama akan diperlakukan sebagai masukan untuk *forward cipher* dan menghasilkan blok *output* kedua. Proses ini diulang hingga mencapai blok terakhir. Untuk blok terakhir yang kemungkinan hanya merupakan sebagian blok terdiri dari *u* bit, hanya *u* bit yang *most significant* dari blok output terakhir digunakan untuk operasi XOR.

Sedangkan pada proses dekripsi, blok output pertama akan di XOR dengan blok *ciphertext* pertama untuk menghasilkan blok *plaintext* pertama. Proses dekripsi mirip dengan proses enkripsi, dimana *plaintext* dan *ciphertext* bertukar posisi. Untuk menjamin kerahasiaan pesan, IV yang digunakan untuk mengenkripsi harus unik untuk tiap pesan.



Gambar 2.5 Mode Operasi OFB (Dworkin, 2001)

E. Mode Counter

Mode *Counter* CTR merupakan mode yang menggunakan *forward cipher* terhadap sekumpulan blok *input*, disebut *counters*, untuk menghasilkan blok *output* yang akan di XOR dengan *plaintext* dan menghasilkan *ciphertext*, dan sebaliknya. Rangkaian *counter* yang digunakan harus memiliki blok yang unik satu sama lain. Kondisi ini tidak dibatasi untuk satu pesan saja, seluruh pesan yang dienkripsi menggunakan *key* yang sama harus memiliki *counter* yang berbeda. Berikut adalah definisi dari mode CTR (*counter* dinotasikan dengan T):

Enkripsi CTR: $O_j = CIPH_K(T_j)$ untuk j = 1, 2 … n;

$C_j = P_j \oplus O_j$ untuk j = 1, 2 … n-1;

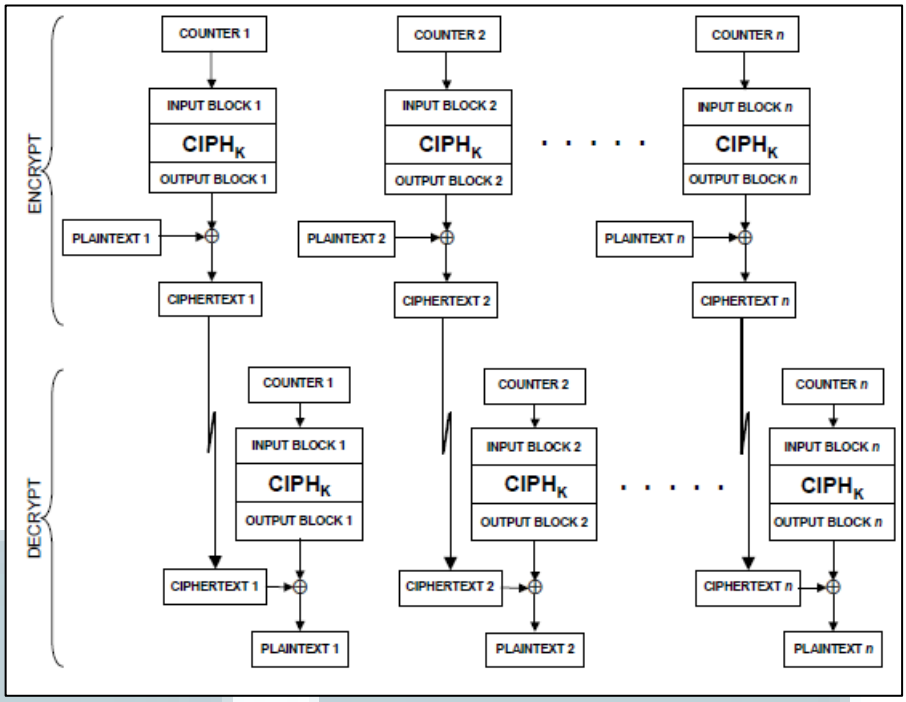
$C^*_n = P^*_n \oplus MSB_u(O_n)$; … rumus 2.9

Dekripsi CTR: $O_j = CIPH_K(T_j)$ untuk j = 1, 2 … n;

$P_j = C_j \oplus O_j$ untuk j = 1, 2 … n-1;

$P^*_n = C^*_n \oplus MSB_u(O_n)$; … rumus 2.10

Dalam enkripsi mode CTR, tiap blok *counter* diproses dengan *forward cipher*, dimana hasilnya akan di XOR dengan *plaintext* untuk menghasilkan *ciphertext*. Untuk blok terakhir yang mungkin hanya terdiri dari *u* bit, hanya *u* bit yang *most significant* dari blok *output* terakhir digunakan untuk operasi XOR. Hal ini juga berlaku untuk proses dekripsi, dimana posisi *plaintext* dan *ciphertext* ditukar.



Gambar 2.6 Mode Operasi CTR (Dworkin, 2001)

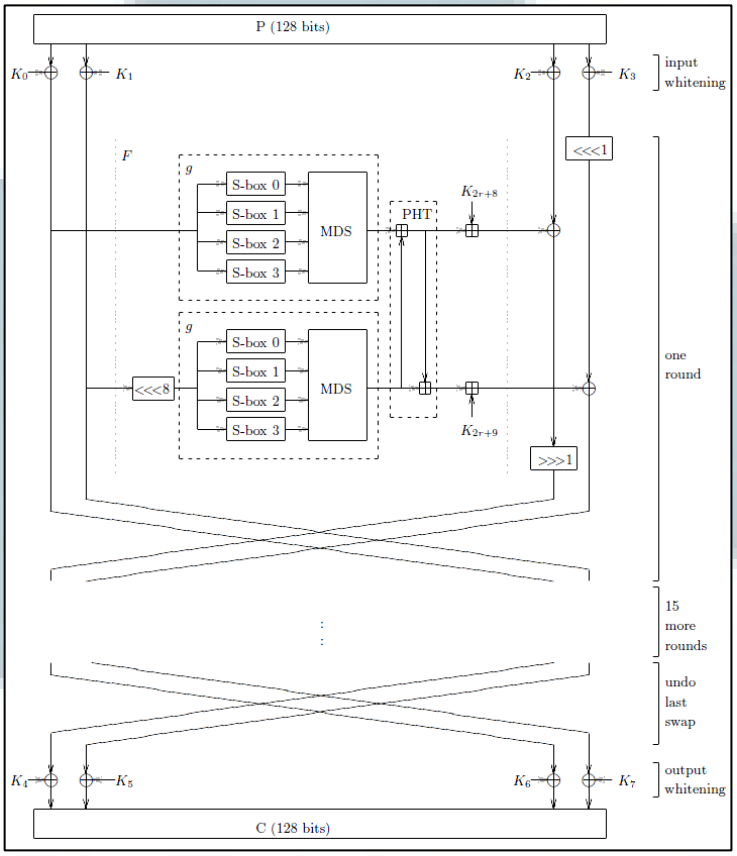
2.2 Algoritma Twofish

Algoritma twofish adalah salah satu algoritma yang menjadi finalis dalam pemilihan AES oleh NIST. Twofish merupakan sebuah *block cipher* berukuran 128-bit yang menggunakan *key* dengan ukuran yang bervariasi, sampai dengan 256-bit dan dioptimalkan untuk CPU 32-bit. Sampai saat ini masih belum ditemukan kriptanalisis yang berhasil memecahkan twofish. Twofish tidak dipatenkan sehingga dapat digunakan pada aplikasi enkripsi tanpa biaya.

2.2.1 Desain Twofish

Twofish menggunakan struktur jaringan Feistel dengan 16 putaran. *Plaintext* dibagi menjadi 4 blok berukuran 32-bit. Pada tahap *input whitening*, blok *plaintext* tersebut di XOR dengan 4 blok *subkey*, yang kemudian diikuti

dengan 16 putaran. Pada tiap putaran, dua blok di kiri akan digunakan sebagai masukan untuk *g function*, salah satunya dirotasi sebanyak delapan bit terlebih dahulu.



Gambar 2.7 Twofish (Schneier, 1998)

Hasil dari 2 *g function* akan dikombinasikan menggunakan *Pseudo Hadamard Transform* (PHT), yang kemudian akan ditambahkan dengan 2 *subkey* putaran. Kedua hasil tersebut akan di-XOR dengan 2 blok di kanan. Kemudian 2 blok bagian kiri dan 2 blok bagian kanan akan bertukar posisi untuk putaran berikutya. Setelah melewati seluruh putaran, pertukaran pada putaran terakhir akan dikembalikan dan ke-4 blok tersebut akan di-XOR dengan 4 blok *subkey*, disebut *output whitening* untuk menghasilkan *ciphertext*.

S-box merupakan operasi substitusi non-linear berdasarkan sebuah tabel. S-box dapat bervariasi baik dari ukuran masukan dan keluaran, dan dapat dibuat secara *random* atau menggunakan algoritma tertentu. Twofish menggunakan 4 S-box yang bersifat *bijective* dan bergantung pada nilai *key*.

Transformasi Pseudo-Hadamard (PHT) adalah operasi matematis sederhana yang mampu bekerja dengan cepat dalam implemetasi *software*. Twofish menggunakan PHT 32-bit untuk mencampur hasil dari dua *g function*. Dengan dua masukan, *a* dan *b*, PHT 32 bit dapat dinotasikan seperti pada rumus 2.11.

$$A_0 = a + b \bmod 2^{32}$$

$$B_0 = a + 2b \bmod 2^{32} \qquad \text{… rumus 2.11}$$

Whitening merupakan teknik yang melakukan XOR terhadap blok *input* dengan material *key* sebelum putaran pertama dan setelah putaran terakhir. Twofish melakukan XOR dengan subkey berukuran 128 bit sebelum putaran pertama dan setelah putaran terakhir. Subkey diperoleh dengan cara yang sama untuk mendapatkan subkey untuk tiap putaran, namun tidak digunakan ditempat lain pada *cipher*. Pada sebuah uji coba serangan terhadap varian Twofish dengan putaran yang dikurangi, ditemukan bahwa proses *whitening* menambah tingkat kesulitan untuk menyerang *cipher* (Schneier, 1998).

A. F Function

F function menerima 3 argumen sebagai masukan, yaitu dua *blok* R_0 dan R_1 yang berukuran 32-bit, serta nomor putaran *r* yang akan digunakan untuk

menentukan *subkey* yang akan digunakan. R_0 (blok paling kiri) akan diproses menggunakan *g function*, yang akan menghasilkan T_0. Sedangkan R_1 akan dirotasi ke kiri sebanyak 8 bit terlebih dahulu sebelum diproses menggunakan *g function* untuk menghasilkan T_1. T_0 dan T_1 digabungkan menggunakan PHT dan ditambahkan dengan *subkey* yang sesuai dan menghasilkan dua blok 32 bit, F_0 dan F_1(Schneier, 1998). Notasi dari *f function* dapat dilihat pada rumus 2.12.

$$T_0 = g(R_0)$$

$$T_1 = g(ROL(R_1, 8))$$

$$F_0 = (T_0 + T_1 + K_{2r+8}) \bmod 2^{32}$$

$$F_1 = (T_0 + 2T_1 + K_{2r+9}) \bmod 2^{32} \qquad \text{… rumus 2.12}$$

B. G Function

G function merupakan dasar dari algoritma Twofish. Masukan X akan dipecah menjadi empat blok berukuran satu *byte*. Tiap *byte* akan diproses menggunakan *key dependent* S-box masing-masing. Setiap S-box bersifat *bijective*, menerima masukan sebesar delapan bit dan menghasilkan keluaran sebesar delapan bit. Hasil dari operasi S-box akan dikalikan dengan matriks MDS berukuran 4x4. Rumus 2.13 menggambarkan notasi dari *g function*:

$$x_i = [X/2^{8i}] \bmod 2^8 \qquad \text{untuk } i = 0, \dots, 3$$

$$y_i = s_i[x_i] \qquad \text{untuk } i = 0, \dots, 3$$

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

$$\boldsymbol{Z} = \sum_{i=0}^{3} \boldsymbol{z_i} \cdot \mathbf{2}^{\boldsymbol{8i}} \qquad \text{… rumus 2.13}$$

dimana s_i adalah *key dependent* S-box dan Z adalah hasil dari *g function.*

Kode MDS (*Maximum Distance Separable*) adalah sebuah pemetaan linear dari elemen *field a* ke elemen *field b*, menghasilkan sebuah vektor gabungan dari elemen a+b, dengan sifat bahwa jumlah elemen berbeda antara dua vektor berbeda yang dihasilkan oleh pemetaan MDS setidaknya sejumlah *b* + 1. Pemetaan MDS dapat direpresentasikan dengan matriks MDS yang terdiri dari elemen *a* x *b*. Twofish menggunakan matriks MDS tunggal berukuran 4x4 (Schneier, 1998). MDS merupakan salah satu elemen *diffusion* yang digunakan oleh Twofish. Penggunaan MDS mampu menghasilkan *diffusion* yang signifikan dalam mencegah serangan kriptanalisis. (Singh, 2000)

C. Key Scheduling

Key schedule adalah sebuah proses untuk merubah *key* menjadi *round keys* yang akan digunakan oleh *cipher*. *Key schedule* Twofish harus menghasilkan 40 *expanded key* K_0, … , K_{39} dan empat *key-dependent* S-box untuk digunakan dalam *g function*. Twofish menggunakan *key* dengan ukuran N = 128, N = 192, dan N = 256-bit. *Key* di bawah ukuran tersebut dapat di *padding* dengan 0 sampai panjang *key* terdekat yang telah didefinisikan.

Key M terdiri dari 8k *byte* yaitu m_0, … , m_{8k-1}, dimana k = N/64. *Byte* tersebut diubah menjadi 2k *words* berukuran 32 bit, lalu dipecah menjadi dua vektor *words* dengan panjang k, berdasarkan ganjil dan genap.

$M_e = (M_0, M_2, \ldots, M_{2k-2})$

$M_o = (M_1, M_3, \ldots, M_{2k-1})$ … rumus 2.14

Selain dua vektor tersebut, dibuat juga vektor ketiga S berdasarkan *key*. Vektor ini diisi dengan mengambil *bytes* dari key dalam ukuran delapan, dan dikalikan dengan matriks 4x8 yang dibuat berdasarkan *Reed-Solomon code*. Tiap empat bytes dari hasil diperlakukan sebagai *word* 32-bit.

Setelah membentuk *s-box keys*, vektor *words* Me dan Mo digunakan untuk membuat *round subkeys*. Rumus 2.15 menggambarkan proses pembuatan *round keys* pada *cipher* Twofish.

$$p = 2^{24} + 2^{16} + 2^{8} + 2^{0}$$

$$A_i = h(2ip, \mathrm{Me})$$

$$B_i = \mathrm{ROL}(h((2i + 1)p, \mathrm{Mo}), 8)$$

$$K_{2i} = (A_i + B_i) \bmod 2^{32}$$

$$K_{2i+1} = \mathrm{ROL}((A_i + 2B_i) \bmod 2^{32}, 9) \qquad \text{… rumus 2.15}$$

Fungsi *h* merupakan sebuah fungsi yang menerima dua masukan, yaitu sebuah *word* 32-bit X dan sebuah daftar *word* 32-bit L dengan ukuran k. Fungsi h memiliki k tahap dan pada setiap tahap, empat *bytes* diproses melalui S-box dan di-XOR dengan sebuah *byte* dari daftar L. Kemudian, setiap *bytes* diproses kembali menggunakan S-box dan dikalikan dengan matriks MDS seperti dalam *g function.*

2.2.2 Kinerja Twofish

Twofish dirancang dari awal untuk memiliki performa yang baik. Twofish dapat diimplementasikan secara efisien pada berbagai macam platform. Desain Twofish memungkinkan berbagai macam *performance tradeoff*, tergantung pada

kepentingan kecepatan enkripsi, *key setup*, penggunaan memori, dan parameter implementasi lainnya (Schneier, 1998).

Tabel 2.1 *Maximal Insecure Variants* finalis AES

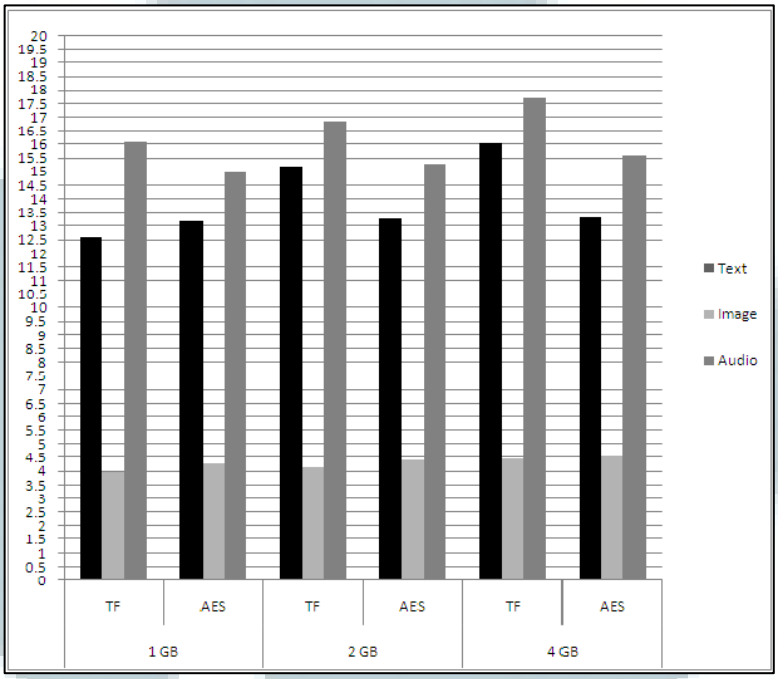
Algoritm Name	Rounds
MARS	9 of 16
RC6	15 of 20
Rijndael	8 of 14
Serpent	9 of 32
Twofish	6 of 16

Berdasarkan pada tabel 2.1, *round function* twofish masih merupakan yang terkuat dibanding finalis AES lainnya. Berdasarkan *maximal insecure variants*, serangan terbaik terhadap twofish yang diketahui dapat memecahkan 6 putaran, sedangkan pada algoritma lainnya dapat dipecahkan sebanyak 9 putaran. Twofish memiliki performa yang cukup baik pada implementasi *hardware*, dan memiliki performa yang baik pada implementasi *software* (Schneier, 2000).

Berdasarkan laporan atas finalis AES oleh James Nechvatal (2000), dkk, Twofish memiliki *security margin* yang tinggi, namun memiliki performa *key setup* yang lambat. Twofish memiliki performa yang bervariasi dalam performa enrkripsi dan dekripsi, namun secara umum memiliki performa yang cukup baik. Twofish membutuhkan sedikit area tambahan untuk mengimplementasi enkripsi dan dekripsi pada hardware, karena kedua proses tersebut hampir identik.

Dapat dilihat pada gambar 2.8, ukuran RAM juga ikut memengaruhi performa algoritma Twofish. Twofish dapat melebihi performa algoritma AES

pada komputer dengan ukuran RAM yang lebih besar. Dampak ini paling terlihat paling jelas pada berkas berjenis *text* dan *audio* (Rizvi, 2011).



Gambar 2.8 Performa AES dan Twofish berdasarkan RAM (Rizvi, 2011)

2.2.3 Kriptanalisis Twofish

Berdasarkan penelitian yang dilakukan oleh Niels Ferguson (1999), 6 dari 16 putaran Twofish dapat dipecah menggunakan *impossible-differential attack*. Apabila pada implementasi Twofish tidak diberikan tahap *whitening*, maka serangan ini memungkinkan untuk memecahkan tujuh putaran. Ini menunjukkan bahwa proses *whitening* setara dengan menambahkan satu putaran pada *round function.*

Pada tahun 2000, Shiho Moriai dan Yiquin Lisa Yin mempublikasikan kriptanalisis terbaik akan twofish yang menemukan karakteristik *truncated*

differential pada versi 16 putaran. Namun hingga saat ini, belum dapat digunakan dalam tipe serangan apapun.

2.3 Random Number Generator

Nilai *random* berperan penting dalam berbagai macam tujuan, seperti menghasilkan *data encryption key*. Dengan kemajuan pada bidang komputer, *programmer* mulai menyadari akan pentingnya kebutuhan akan *random number*. Namun, komputer hanya melakukan instruksi yang telah diberikan sehingga dapat dengan mudah diprediksi. Agar komputer dapat menghasilkan random number, terdapat 2 pendekatan yang dapat digunakan, yaitu *Pseudo-Random Number Generation* (PRNG) dan *True Random Number Generator* (TRNG).

PRNG adalah teknik yang menggunakan formula matematika atau tabel tertentu untuk menghasilkan nilai yang terlihat random. Sedangkan TRNG menggunakan fenomena fisik untuk menghasilkan random number dan mengimplementasikannya ke dalam komputer. PRNG mampu bekerja dengan efisien dan bersifat deterministik. (Random.org, tanpa tahun)

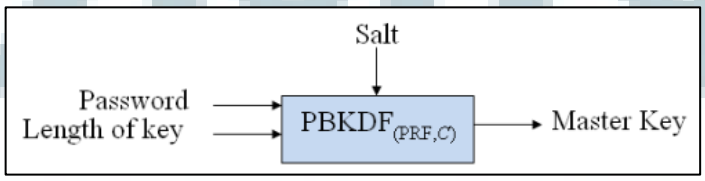
Pada umumnya, aplikasi kriptografi menggunakan teknik tertentu untuk menghasilkan nilai *random*. Teknik ini biasanya bersifat deterministik, sehingga hasil yang diberikan tidak bernilai *random* jika dilihat secara statistik. Namun jika teknik yang digunakan baik, hasil yang didapatkan akan mampu melewati berbagai uji *randomness*. Nilai yang bersifat seperti ini disebut sebagai *pseudorandom numbers*. (Stallings, 2010)

Class RNGCryptoServiceProvider dari .NET Framework menggunakan implementasi *Random Number Generator* yang dapat digunakan untuk digunakan aplikasi kriptografi. *Random number* yang dihasilkan dari *class* ini dapat digunakan sebagai *key*, IV, dan *salt* yang dibutuhkan sebuah aplikasi kriptografi. (Microsoft, tanpa tahun)

2.4 Password-Based Key Derivation Function

Password adalah sebuah kalimat yang biasa dibuat dan digunakan oleh pengguna untuk mengautentikasi hak akses pengguna tersebut terhadap sumber daya tertentu. Pada umumnya, *password* yang dibuat oleh pengguna memiliki sifat *random* yang lemah, sehingga sebaiknya tidak digunakan sebagai *key* untuk kriptografi. Namun, pada aplikasi tertentu, seperti proteksi data pada media penyimpanan, *password* dapat merupakan informasi rahasia satu-satunya yang dapat digunakan untuk memproteksi data.

Password-Based Key Derivation Function adalah algoritma deterministik yang dapat digunakan untuk menghasilkan materi *key* dari *password*. Setiap PBKDF didefinisikan menggunakan sebuah *Pseudorandom Function* dan sejumlah iterasi tertentu. Masukan untuk PBKDF meliputi *password*, *salt*, dan ukuran *key* yang diinginkan (Turan, 2010).



Gambar 2.9 Diagram generik PBKDF

Salt pada kriptografi berbasis *password* digunakan untuk menghasilkan kumpulan *keys* berdasarkan *password*, dimana salah satunya akan dipilih secara acak berdasarkan *salt*. Penggunaan *salt* memberikan 2 keuntungan, yaitu:

a. Penyerang akan kesulitan untuk melakukan prekomputasi terhadap seluruh *key* berdasarkan sebuah *password dictionary*.
b. Kemungkinan akan penggunaan *key* yang sama sangatlah kecil.

Jumlah iterasi meningkatkan biaya produksi *key*, sehingga meningkatkan tingkat kesulitan untuk melakukan serangan. Secara matematis, jumlah iterasi *c* akan meningkatkan kekuatan *password* sebanyak $\log_2(c)$ bit terhadap serangan berbasis percobaan, seperti *brute force attack*. Pemilihan jumlah iterasi bergantung pada keadaan dan bervariasi antar aplikasi. Jumlah iterasi minimum yang direkomendasikan adalah 1.000 kali (RSA Laboratories, 2011).

2.5 Message Authenticaton Code

Message Authentication adalah sebuah mekanisme yang digunakan untuk memverifikasi integritas sebuah pesan. *Message authentication* memastikan bahwa data yang diterima dan data yang dikirim adalah sama. Salah satu teknik autentikasi adalah menggunakan *key* untuk menghasilkan blok data berukuran tetap yang ditambahkan pada pesan, disebut sebagai *cryptographic checksum* atau MAC (*Message Authentication Code*). MAC dapat didefinisikan seperti pada rumus 2.16.

$$MAC = MAC(K, M) \quad \text{… rumus 2.16}$$

dimana M = pesan

C = fungsi MAC

K = *key*

MAC = *Message Authentication Code*

Dengan menggunakan MAC, pesan dapat dipastikan tidak mengalami perubahan oleh pihak ketiga, dengan asumsi bahwa *key* yang digunakan pada fungsi MAC tidak diketahui. Karena penyerang tidak mengetahui *key* yang digunakan, penyerang tidak dapat mengubah MAC sebuah pesan untuk menutupi perubahan yang dilakukannya terhadap isi pesan. Fungsi MAC tidak perlu bersifat reversibel (Stallings, 2010).