BAB II
TINJAUAN PUSTAKA

2.1 Kriptografi

Kriptografi adalah ilmu sekaligus seni untuk menjaga keamanan pesan (*message*). Kata *cryptography* berasal dari kata Yunani yaitu ”kryptos” yang artinya tersembunyi dan ”graphein” yang berarti menulis.

Algoritma kriptografi memuat data proses penting yaitu:

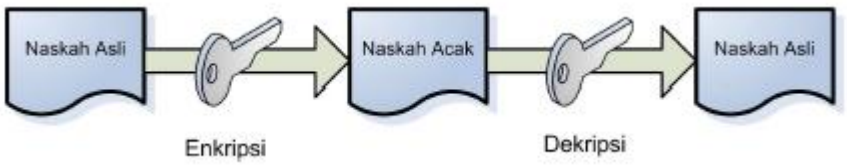
- aturan untuk *enciphering* dan *deciphering*,
- fungsi matematika yang digunakan untuk enkripsi dan dekripsi (Wirdasari,2008).

2.2 Enkripsi dan Dekripsi

Proses menyandikan *plaintext* menjadi *ciphertext* disebut enkripsi (*encryption*) atau *enciphering* (standard nama menurut ISO 7498-2), dan proses mengembalikan *ciphertext* menjadi *plaintext*-nya disebut dekripsi (*decryption*) atau *deciphering* (standard nama menurut ISO 7498-2) (Wirdasari, 2008).

Teknik enkripsi dalam kriptografi klasik yang digunakan adalah enkripsi simetris dimana kunci dekripsi sama dengan kunci enkripsi. Untuk *public key*

cryptography, diperlukan teknik enkripsi asimetris dimana kunci dekripsi tidak sama dengan kunci enkripsi. Enkripsi, dekripsi dan pembuatan kunci untuk teknik enkripsi asimetris memerlukan komputasi yang lebih intensif dibandingkan enkripsi simetris, karena enkripsi asimetris menggunakan bilangan-bilangan yang sangat besar (Kromodimoeljo, 2009).



Gambar 1. Proses enkripsi dan dekripsi

2.3 Kunci Kriptografi

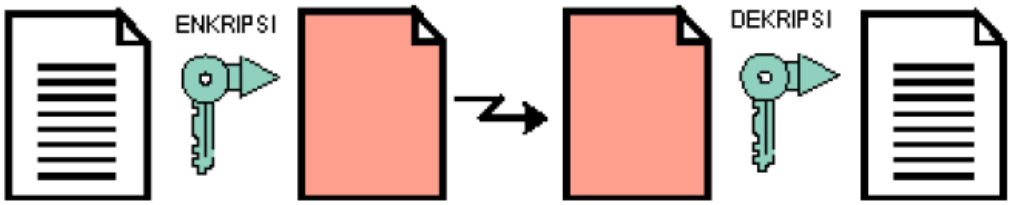
Enkripsi dan dekripsi pada umumnya membutuhkan penggunaan sejumlah informasi rahasia, disebut sebagai kunci. Kunci adalah parameter yang digunakan untuk transformasi *enciphering* dan *deciphering*. Kekuatan kriptografi ditentukan dengan menjaga kerahasiaan algoritmanya, maka algoritma kriptografinya dinamakan algoritma *restricted*. Algoritma *resricted* tidak cocok lagi saat ini. Kriptografi modern mengatasi masalah ini dengan menggunakan kunci. Kunci bersifat rahasia (*secret*), sedangkan algoritma kriptografi tidak rahasia (*public*) (Wirdasari, 2008).

2.4 Jenis Algoritma Kriptografi Berdasarkan kunci

Dua jenis algoritma kriptografi berdasarkan kuncinya, yaitu:

2.4.1 Algoritma Simetris

Algoritma simetris menggunakan kunci yang sama untuk melakukan enkripsi dan dekripsi. Contoh algoritma simetris antara lain *Data Encryption Standard* (DES), RC2, RC4, RC5, RC6, *Tiny Encryption Algorithm(*TEA*),* LOKI, *International Data Encryption Algorithm* (IDEA), *Advanced Encryption Standard* (AES), *One Time Pad* (OTP), A5 dan sebagainya(Simarmata, 2013).



Gambar 2. Kunci Algoritma Simetri

Algoritma simetris baik pengirim maupun penerima memiliki kunci rahasia yang umum. Kunci-kunci ini harus dirahasiakan, oleh karena itulah sistem ini sering disebut sebagai *secret-key ciphersystem*. Jumlah kunci yang dibutuhkan umumnya adalah :

$$nC_2 \frac{n(n-1)}{2} \quad (1)$$

dengan *n* menyatakan banyaknya pengguna (Wirdasari, 2008).

Kelebihan kunci simetris:

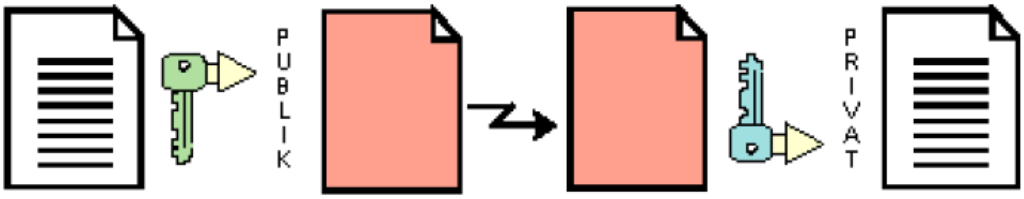
a. Kecepatan operasi lebih tinggi bila dibandingkan dengan algoritma asimetris.
b. Karena kecepatannya yang cukup tinggi, maka dapat digunakan pada sistem *real-time*

Kelemahan kunci simetris:

a. Pengiriman pesan dengan pengguna yang berbeda dibutuhkan kunci yang berbeda juga, sehingga akan terjadi kesulitan dalam manajemen kunci tersebut.
b. Permasalahan dalam pengiriman kunci itu sendiri yang disebut "*key distribution problem*" .

2.4.2 Algoritma Asimetris

Algoritma asimetris menggunakan kunci yang berbeda untuk enkripsi dan dekripsi. Beberapa contoh algoritma asimetris antara lain *Digital Signature Algorithm* (DSA), RSA, Diffie-Hellman, Pohlig-Hellman, *Elliptic Curve Crypthography* (ECC), Kriptografi *Quantum* dan sebagainya (Simarmata, 2013).



Gambar 3 Kunci Algoritma Asimetris

Kelebihan kunci asimetris:

a. Masalah keamanan pada distribusi kunci dapat lebih baik.
b. Masalah manajemen kunci yang lebih baik karena jumlah kunci yang lebih sedikit.

Kelemahan kunci asimetris:

a. Kecepatan yang lebih rendah bila dibandingkan dengan algoritma simetris.
b. Kunci yang digunakan lebih panjang dibandingkan dengan algoritma simetris untuk tingkat keamanan sama.

2.5 Tujuan Kriptografi

Empat tujuan mendasar dari kriptografi yaitu,

1. *Confidentiality,* adalah layanan yang ditujukan untuk menjaga agar pesan tidak dapat dibaca oleh pihak-pihak yang tidak berhak.
2. *Data integrity*, adalah layanan yang menjamin bahwa pesan masih asli/utuh atau belum pernah dimanipulasi selama pengiriman.
3. *Authentication*, adalah layanan yang berhubungan dengan identifikasi, baik mengidentifikasi kebenaran pihak-pihak yang berkomunikasi (*user authentication or entity authentication*) maupun mengidentifikasi kebenaran sumber pesan (*data origin authentication*).
4. *Non-repudiation*, adalah layanan untuk mencegah entitas yang berkomunikasi melakukan penyangkalan, yaitu pengirim pesan menyangkal melakukan pengiriman atau penerima pesan menyangkal telah menerima pesan (Stalling, 1998).

2.6 Notasi Matematis

Misalkan: *C = ciphertext*

P = plaintext

Fungsi enkripsi *E* memetakan *P* ke *C*,

$$E(P) = C \quad (2)$$

Fungsi dekripsi *D* memetakan *C* ke *P*,

$$D(C) = P \quad (3)$$

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

$$D(E(P)) = P \quad (4)$$

Kekuatan algoritma kriptografi diukur dari banyaknya kerja yang dibutuhkan untuk memecahkan data *ciphertext* menjadi *plaintext*-nya. Kerja ini dapat diekuivalenkan dengan waktu, semakin keras usaha yang dilakukan, yang berarti juga semakin lama waktu yang dibutuhkan, maka semakin kuat algoritma kriptografinya, yang berarti semakin aman digunakan untuk menyandikan pesan. Kekuatan kriptografi ditentukan dengan menjaga kerahasiaan algoritmanya, maka algoritma kriptografinya dinamakan algoritma *restricted*. Algoritma *restricted* tidak cocok lagi saat ini. Sistem kriptografi modern menggunakan kekuatan kriptografinya, yang terletak pada kunci. Kunci tersebut berupa deretan karakter atau bilangan bulat, dijaga kerahasiaannya, dengan menggunakan kunci *K*, maka fungsi enkripsi dan dekripsi menjadi:

$$E_K(P) = C \quad (5)$$

$$D_K(C) = P \quad (6)$$

dan kedua fungsi ini memenuhi:

$$D_K (E_K (P)) = P \quad (7)$$

2.7 Algoritma *Cipher Block*

Algoritma *cipher block* termasuk ke dalam algoritma simetris yang mengoperasikan bit-bit yang panjangnya tetap dengan sebuah transformasi yang sama. *Plaintext* dibagi menjadi blok-blok bit dengan panjang yang sama, ketika melakukan enkripsi, misalnya pada sebuah blok *plaintext* yang panjangnya 64 bit maka akan menghasilkan blok *ciphertext* sepanjang 64 bit juga. Proses dekripsi pun sama halnya, sebuah blok *ciphertext* sepanjang 128 bit akan menghasilkan sebuah blok *plaintext* sepanjang 128 bit. Kunci yang digunakan untuk melakukan enkripsi dan dekripsi mempunyai panjang yang sama dengan blok bit *plaintext* (Indriani, 2007).

Cipher block terdiri dari dua buah algoritma yang berpasangan. Satu digunakan untuk melakukan enkripsi, *E*, dan satu lagi digunakan untuk melakukan dekripsi, E^{-1}. Kedua algoritma tersebut menerima dua buah input, yaitu sebuah blok input berukuran *n* bit dan sebuah kunci berukuran *k* bit, menghasilkan sebuah blok yang berukuran *n* bit.

2.8 Algoritma LOKI

LOKI dirancang oleh kriptografer Australia yaitu Lawrie Brown, Josef Pieprzyk, dan Jennifer Seberry. LOKI didesain sebagai hasil dari analisis yang dilakukan secara detail terhadap blok *cipher* yang standar digunakan pada saat itu, yaitu DES (*Data Encryption Standar*). Ada versi terbaru dari LOKI ini, maka LOKI

yang dibuat pertama kali lebih dikenal dengan nama LOKI89 sesuai dengan tahun pembuatannya, walaupun LOKI pertama kali diperkenalkan pada tahun 1990 (Indriani, 2007).

LOKI didesain untuk menggantikan DES sehingga strukturnya pun dibuat hampir sama dengan DES. LOKI menggunakan blok data sepanjang 64 bit dan kunci sepanjang 64 bit pula.

2.9 Algoritma *Tiny Encryption Algorithm* (TEA)

Tiny Encription Algorithm (TEA) merupakan suatu algoritma sandi yang diciptakan oleh David Wheeler dan Roger Needham dari *Computer Laboratory*, *Cambridge University*, *England* pada bulan November 1994. Algoritma ini merupakan algoritma mengenkripsi suatu blok *plaintext* dengan jumlah bit tertentu dan menghasilkan blok *ciphertext* yang dirancang untuk penggunaan *memory* yang seminimal mungkin dengan kecepatan proses yang maksimal.

Sistem penyandian TEA menggunakan proses *feistel network* dengan menambahkan fungsi matematik berupa penambahan dan pengurangan sebagai operator pembalik selain XOR. Proses *feistel network* adalah membagi *plaintext* ke dalam beberapa blok dan melakukan penukaran letak blok dalam setiap *round*, yang akan memberikan efek konsep konfusi dan difusi. Konfusi adalah mengaburkan hubungan *plaintext* dan *ciphertext* yang menimbulkan kesulitan dalam usaha untuk mencari keteraturan *plaintext* dan *ciphertext*, sedangkan difusi adalah menyebarkan redudansi *plaintext* dengan menyebarkan masukan ke seluruh *ciphertext*. Hal ini dimaksudkan untuk menciptakan pergeseran dua arah (ke kiri dan ke kanan) menyebabkan semua bit kunci dan data bercampur secara

berulang ulang. Algoritma TEA merupakan algoritma kriptografi simeteris atau disebut juga algoritma kriptografi konvensional yaitu algoritma yang menggunakan kunci untuk proses enkripsi sama dengan kunci untuk proses dekripsi (Nurdin, 2013).

Cara Kerja Algoritma TEA

1. Pergeseran (*shift*)

   Blok teks terang pada kedua sisi yang masing masing sebanyak 32-bit akan digeser kekiri sebanyak empat (4) kali dan digeser ke kanan sebanyak lima (5) kali.

2. Penambahan

   Langkah selanjutnya setelah digeser kekiri dan kekanan, maka *Y* dan *Z* yang telah digeser akan ditambahkan dengan kunci *k[0]-k[3]*. Sedangkan *Y* dan *Z* awal akan ditambahkan dengan *sum* (*delta*).

3. Dilakukan Proses XOR

   Proses selanjutnya setelah dioperasikan dengan penambahan pada masing-masing *register* maka akan dilakukan proses XOR dengan rumus untuk satu *round* adalah sebagai berikut:

   $$y = y + (((z<<4)+k[0])\wedge z+sum\wedge((z>>5)+k[1])) \quad (8)$$

   $$z = z + (((y<<4)+k[2]\wedge y+sum\wedge((y>>5)+k[3])) \quad (9)$$

   Hasil penyandian dalam satu *cycle* satu blok teks terang 64-bit menjadi 64-bit teks sandi adalah dengan menggabungkan *Y* dan *Z*. Untuk penyandian pada

cycle berikutnya *Y* dan *Z* ditukar posisinya, sehingga *Y 1* menjadi *Z1* dan *Z1* menjadi *Y1* lalu dilanjutkan proses seperti langkah langkah di atas sampai dengan 16 *cycle* (32 *round*).

4. *Key Schedule*

Algoritma TEA menggunakan *key schedule* sangat sederhana, yaitu kunci *k[0]* dan *k[1]* konstan digunakan untuk *round* ganjil sedangkan kunci *k[2]* dan *k[3]* konstan digunakan untuk *round* genap.

5. Dekripsi dan Enkripsi

Proses dekripsi sama halnya seperti pada proses penyandian yang berbasis *feistel cipher* lainnya, yaitu pada prinsipnya adalah sama pada saat proses enkripsi. Hal yang berbeda adalah penggunaan teks sandi sebagai input dan kunci yang digunakan urutannya dibalik. Proses dekripsi semua *round* ganjil menggunakan *k[1]* terlebih dahulu kemudian *k[0]*, demikian juga dengan semua *round* genap digunakan *k[3]* terlebih dahulu kemudian *k[2]*. Rumus untuk enkripsi dekripsi seperti di bawah ini:

Proses enkripsi digunakan rumus :

$$L0 = L0 + f\,(R0, k[0], k[1], sum) \quad (10)$$

$$R0 = R0 + f\,(L0, k[2], k[3], sum) \quad (11)$$

Jadi *L0* merupakan hasil penjumlahan dari *L 0* ditambahkan dengan *f (R0 , k[0], k[1], sum).*

Proses enkripsi untuk satu *round* digunakan rumus:

$$y = y + (((z<<4)+k[0]) \wedge z+sum \wedge ((z>>5)+k[1])) \quad (12)$$

$$z = z + (((y<<4)+k[2]) \wedge y+sum \wedge ((y>>5)+k[3])) \quad (13)$$

Proses dekripsi digunakan rumus :

$$L0 = L0 + f\ (\ R0\ ,\ k[1],\ k[0],\ sum\ ) \quad (14)$$

$$R0 = R0 + f\ (\ L0,\ k[3],\ k[2],\ sum\ )\ . \quad (15)$$

Jadi *L0* merupakan hasil penjumlahan dari *L0* ditambahkan dengan *f (R0 , k[0], k[1], sum).*

Proses dekripsi untuk satu *round* digunakan rumus:

$$y = y + (((z<<4)+k[1])\hat{}z+sum\hat{}((z>>5)+k[0])) \quad (16)$$

$$z = z + (((y<<4)+k[3]\hat{}y+sum\hat{}((y>>5)+k[2])) \quad (17)$$

Rumus *Y* di atas menjelaskan bahwa *Y* merupakan hasil dari *Y* yang ditambahkan dengan *Z* yang yang digeser kekiri sebanyak empat kali dengan penambahan kunci *k[1]*. Kemudian hasil penjumlahan tadi di XOR kan dengan *Z* yang dijumlahkan dengan *sum*(*delta*). Hasil dari proses XOR dari kedua penjumlahan tadi di XOR lagi dengan *Z* yang digeser kekanan sebanyak lima kali dengan penambahan kunci *k[0]*. Rumus *Z* sama halnya dengan rumus *Y*, hanya kunci yang digunakan menggunakan kunci *k[3]* dan *k[2]* (Nurdin, 2013).

2.10 *Running Time*

Running time merupakan waktu yang dibutuhkan untuk mengeksekusi setiap instruksi di dalam program sampai selesai. Program memiliki beberapa operasi yaitu penambahan (+), pengurangan (-), perkalian (*), pembagian (/), *return* (pengembalian nilai dari fungsi), inisialisasi, dan perbandingan dimana setiap operasi ini *running time*-nya masing-masing 1 unit (Weiss, 2007).

2.11 Kompleksitas Algoritma

Kompleksitas dari suatu algoritma merupakan ukuran seberapa banyak komputasi yang dibutuhkan algoritma tersebut untuk menyelesaikan masalah. Secara informal, algoritma yang dapat menyelesaikan suatu permasalahan dalam waktu yang singkat memiliki kompleksitas yang rendah, sementara algoritma yang membutuhkan waktu lama untuk menyelesaikan masalahnya mempunyai kompleksitas yang tinggi. Nilai *n* yang besar atau bahkan tidak terbatas perlu dilakukan analisis efisiensi dari suatu algoritma untuk menentukan kompleksitas waktu yang sesuai atau disebut juga kompleksitas waktu asimptotik dengan melihat waktu tempuh (*running time*). Kompleksitas waktu asimptotik terdiri dari tiga macam. Pertama, keadaan terbaik (*best case*) dinotasikan dengan Ω *(g(n))* (*Big-Omega*), keadaan rata-rata (*average case*) dinotasikan dengan Θ *(g(n))* (*Big-Theta*) dan keadaan terburuk (*worst case*) dinotasikan dengan *O(g(n))* (*Big-O*). Kompleksitas waktu algoritma dihitung dengan menggunakan notasi *O(f(n))*, dibaca ”*big-O* dari *f(n)*” (Weiss, 2007).

Notasi *O* menyatakan *running time* dari suatu algoritma untuk memungkinkan kasus terburuk. Notasi *O* memiliki beberapa bentuk :

1. *O(1)*, merupakan algoritma konstan yang artinya *running time* algoritma tersebut tetap, tidak bergantung pada *n*.
2. *O(n)*, disebut algoritma linier yang artinya bila *n* menjadi 2*n* maka *running time* algoritma tersebut akan menjadi dua kali semula.

3. $O(n^2)$, disebut algoritma kuadratik yang biasanya hanya digunakan untuk kasus dengan *n* yang berukuran kecil. Sebab, bila *n* dinaikkan menjadi dua kali semula, maka *running time* algoritma akan menjadi empat kali semula.
4. $O(n^3)$, disebut algoritma kubik dimana bila *n* dinaikkan menjadi dua kali semula, maka *running time* algoritma akan menjadi delapan kali semula.
5. $O(2^n)$, disebut algoritma eksponensial dimana bila *n* dinaikkan menjadi dua kali semula, maka *running time* algoritma akan menjadi kuadrat kali semula.
6. *O(log n)*, disebut algoritma logaritmik dimana laju pertumbuhan waktu lebih lambat dari pada pertumbuhan *n*. Algoritma yang termasuk algoritma logaritmik adalah algoritma yang memecahkan persoalan besar dengan mentransformasikannya menjadi beberapa persoalan yang lebih kecil dengan ukuran sama. Basis algoritma tidak terlalu penting, sebab bila misalkan n dinaikkan menjadi dua kali semula, log *n* meningkat sebesar jumlah tetapan.
7. *O(n log n)*, terdapat pada algoritma yang membagi persoalan menjadi beberapa persoalan yang lebih kecil, menyelesaikan setiap persoalan secara independen, kemudian menggabungkan solusi masing-masing persoalan.
8. *O(n!)*, diebut algoritma faktorial dimana algoritma jenis ini akan memproses setiap masukan dan menghubungkannya dengan *n*-1 masukan lainnya. Bila *n* menjadi dua kali semula, maka *running time* algoritma akan menjadi faktorial dari 2*n*.

Analisis kompleksitas waktu algoritma dihitung dengan menggunakan notasi *O(f(n))* dimana Notasi *O* menyatakan *running time(T(n))* dari suatu algoritma untuk memungkinkan kasus terburuk (*worst case*) (Weiss, 2007).