BAB II

LANDASAN TEORI

II.1. Perancangan

Menurut Nataniel Dengen dan Heliza Rahmania Hatta (2009), Perancangan adalah suatu penggambaran, perencanaan dan pembuatan sketsa atau pengaturan dari beberapa elemen yang terpisah ke dalam satu kesatuan yang utuh dan berfungsi untuk memenuhi kebutuhan kepada pemakai sistem dan untuk memberikan gambaran yang jelas kepada pemogram komputer dan ahli-ahli teknik lainnya yang terlibat.

II.1.1 UML

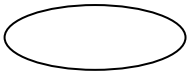
Unified modeling language (UML) adalah keluarga notasi grafis yang didukung oleh meta-model tunggal, yang membantu pendeskripsian dan desain sistem perangkat lunak, khususnya sistem yang dibangun menggunakan pemrograman berorientasi objek (OOP).

Unified modeling language (UML) adalah sebuah "bahasa" yang telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. (Alfera Lesmana, S.Kom, et all, 2010).

II.1.1.1 *Use Case*

Use case merupakan fungsionalitas yang diharapkan dari sebuah sistem. *Use case* merepresentasikan deskripsi lengkap tentang interaksi yang terjadi antara para actor dengan sistem/perangkat lunak yang sedang dikembangkan (Alfera Lesmana, S.Kom, et all, 2010).

Tabel II.1 Simbol *Use Case Diagram*

Nama Komponen	Keterangan	Simbol
Use Case	*Use case* digambarkan sebagai lingkaran elips dengan nama *use case* dituliskan di dalam elips tersebut.	
Actor	*Actor* adalah pengguna sistem. *Actor* tidak terbatas hanya manusia saja, jika sebuah sistem berkomunikasi dengan aplikasi lain dan membutuhkan *input* atau memberikan *output*, maka aplikasi tersebut juga bisa dianggap sebagai *actor*.	
Association	Asosiasi digunakan untuk menghubungkan *actor* dengan *use case*. Asosiasi digambarkan dengan sebuah garis yang menghubungkan antara *Actor* dengan *Use Case*.	

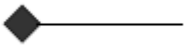
(Sumber : Alfera Lesmana, S.Kom, et all; 2010)

II.1.1.2 *Class Diagram*

Class diagram adalah suatu diagram yang memperlihatkan struktur sebuah sistem. *Class diagram* yang menggambarkan struktur *class, package,* dan objek beserta hubungan satu sama lain seperti dinamis, pewarisan, asosiasi, dan agregasi (Alfera Lesmana, S.Kom, et all, 2010).

Tabel II.2 Simbol *Class Diagram*

Nama Komponen	Keterangan	Simbol
Class	*Class* adalah blok - blok pembangun pada pemrograman berorientasi objek. Sebuah *class* digambarkan sebagai sebuah kotak yang terbagi atas 3 bagian. Bagian atas adalah bagian nama dari *class*. Bagian tengah mendefinisikan *property*/atribut *class*. Bagian akhir mendefinisikan *method-method* dari sebuah *class*.	Nama *Class* + atribut + atribut + atribut + *method* + *method*

Association	Sebuah asosiasi merupakan sebuah *relationship* paling umum antara 2 *class* dan dilambangkan oleh sebuah garis yang menghubungkan antara 2 *class*. Garis ini bisa melambangkan tipe-tipe *relationship* dan juga dapat menampilkan hukum-hukum multiplisitas pada sebuah *relationship*. (Contoh: *One-to-one, one-to-many, many-to-many*).	
Composition	Jika sebuah *class* tidak bisa berdiri sendiri dan harus merupakan bagian dari *class* yang lain, maka *class* tersebut memiliki relasi *Composition* terhadap *class* tempat dia bergantung tersebut. Sebuah *relationship composition* digambarkan sebagai garis dengan ujung berbentuk jajaran genjang berisi/*solid*.	
Dependency	Kadangkala sebuah *class* menggunakan *class* yang lain. Hal ini disebut *dependency*. Umumnya penggunaan *dependency* digunakan untuk menunjukkan operasi pada suatu *class* yang menggunakan *class* yang lain. Sebuah *dependency* dilambangkan sebagai sebuah panah bertitik-titik.	
Aggregation	*Aggregation* mengindikasikan keseluruhan bagian *relationship* dan biasanya disebut sebagai relasi	

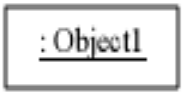
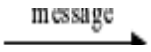
(Sumber : Alfera Lesmana, S.Kom, et all; 2010)

II.1.1.3 *Sequence Diagram*

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa message yang digambarkan terhadap waktu. *Sequence diagram* terdiri atas dimensi *vertical* (waktu) dan *horizontal* (objek-objek terkait).

Sequence diagram digunakan untuk menggambarkan *scenario* atau langkah-langkah yang dilakukan sebagai respon dari sebuah *event* untuk menghasilkan output tertentu (Alfera Lesmana, S.Kom, et all, 2010).

Tabel II.3 Simbol *Sequence Diagram*

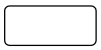
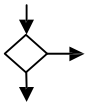
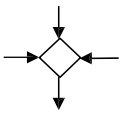
Nama Komponen	Keterangan	Simbol
Object	*Object* merupakan *instance* dari sebuah *class* dan dituliskan tersusun secara *horizontal*. Digambarkan sebagai sebuah *class* (kotak) dengan nama *object* di dalamnya yang diawali dengan sebuah titik koma.	
Actor	*Actor* juga dapat berkomunikasi dengan *object* , maka *actor* juga dapat diurutkan sebagai kolom. Simbol *Actor* sama dengan simbol pada *Actor Use Case Diagram.*	
Lifeline	*Lifeline* mengindikasikan keberadaan sebuah *object* dalam basis waktu. Notasi untuk *lifeline* adalah garis putus-putus *vertikal* yang ditarik dari sebuah *object* .	
Activation	*Activation* dinotasikan sebagai sebuah kotak segi empat yang digambar pada sebuah *lifeline*. mengindikasikan sebuah *object* yang akan melakukan sebuah aksi.	
Message	*Message*, digambarkan dengan anak panah *horizontal* antara *Activation Message* mengindikasikan komunikasi antara *object – object.*	

(Sumber : Alfera Lesmana, S.Kom, et all; 2010)

II.1.1.4 *Activity Diagram*

Activity Diagram adalah representasi grafis dari alur kerja tahapan aktivitas. Pada pemodelan UML, activity diagram dapat digunakan untuk menjelaskan bisnis dan alur kerja operasional secara tahap demi tahap dari komponen suatu sistem. *Activity Diagram* menunjukkan keseluruhan dari aliran *control* (Alfera Lesmana, S.Kom, et all, 2010).

Tabel II.4 Simbol *Activity Diagram*

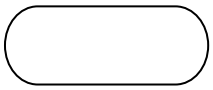
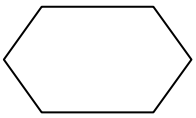
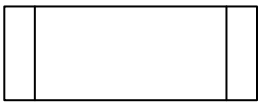
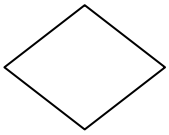
Nama Komponen	Keterangan	Simbol
Initial Node	Simbol awal proses	
Actions	Simbol aktivitas yang terjadi di sistem	
Flow	Simbol aksi atau proses selanjutnya	
Decision	Simbol satu *flow* datang dan 2 *flow* keluar dimana ada pilihan *activity*	
Merge	Pertemuan arah yang dimulai dari *decision*	
Fork	Simbol awal action atau proses paralel yang terjadi di urutan atau waktu yang sama	
Join	Simbol akhir dari *fork*	
Activity Final	Simbol proses akhir	

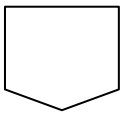
(Sumber : Alfera Lesmana, S.Kom, et all; 2010)

II.1.2. *Flowchart*

Flowchart adalah penyajian yang sistematis tentang proses dan logika dari kegiatan penanganan informasi atau penggambaran secara grafik dari langkah-langkah dan urut-urutan prosedur dari suatu *program*. *Flowchart* menolong analis dan *programmer* untuk memecahkan masalah ke dalam segmen-segmen yang lebih kecil dan menolong dalam menganalisis alternatif-alternatif lain dalam pengoprasian (Novi Dian Nathasia dan Anang Eko Wicaksono, 2011**)**.

Tabel II.5 Simbol *Flowchart*

Nama Komponen	Keterangan	Simbol
Terminator	Permulaan/akhir *program*	
Garis Alir (*Flow Line*)	Arah aliran *program*	
Preparation	Proses inisialisasi / pemberian harga awal	
Process	Proses penghitungan / proses pengolahan *data*	
Input/Output Data	Proses *input / output data*, parameter, informasi	
Predefined Process (Sub Program)	Permulaan *sub program* / proses menjalankan *sub program*	
Decision	Perbandingan pernyataan, penyeleksian *data* yang memberikan pilihan untuk langkah selanjutnya	

One Page Connector	Penghubung bagian-bagian *flowchart* yang berada pada satu halaman	
Off Page Connector	Penghubung bagian-bagian *flowchart* yang berada pada halaman yang berbeda	

(Sumber : Novi Dian Nathasia dan Anang Eko Wicaksono; 2011)

II.2. *Chatting*

Menurut Aini Yuwanisa (2012), *Chatting* adalah percakapan interaktif antar sesama pengguna komputer yang terhubung dalam suatu jaringan. Percakapan ini bisa dilakukan dengan saling berinteraktif melalui *teks*, maupun suara.

Sedangkan menurut Mohamad Fauzi Haryadi (2010), *Chatting* dalam bahasa Indonesia berarti ngobrol atau berbicara dua arah antara satu atau beberapa orang. Di dalam dunia komputer, *chatting* berarti berbicara dengan orang lain dengan menggunakan komputer.

Dengan demikian dapat disimpulkan bahwa *Chatting* adalah percakapan yang terjadi antara satu atau beberapa orang pengguna komputer yang saling terhubung dalam suatu jaringan.

II.3. Keamanan Jaringan

Menurut Syariful Ikhwan dan Ikhwana Elfitri (2014), Keamanan jaringan pada intinya adalah mengendalikan akses terhadap sumberdaya jaringan. Akses jaringan dikontrol agar bisa diakses oleh siapa saja yang berhak dan menghalangi

orang atau subjek yang tidak terdaftar untuk mengaksesnya. Prinsip keamanan jaringan di klasifikasikan menjadi 3 bagian :

1. *Confidentiality* (Kerahasiaan)

   Confidentiality mengacu pada kerahasiaan sebuah objek, dimana sebuah objek dijaga agar tidak diakses oleh subjek yang tidak berhak. Istilah ini juga mengacu pada data pribadi yang diberikan kepada pihak lain untuk keperluan tertentu dan hanya digunakan untuk keperluan tersebut. Contoh data-data yang sifatnya pribadi itu adalah nama, nomor kartu kredit, nomor paspor, nomor telepon, *password* komputer, agama, status perkawinan dan lain-lain Ada banyak *tool* yang digunakan untuk menjaga agar kerahasiaan sebuah subjek terjaga dengan baik, diantaranya Enkripsi, Akses Kontrol, otentifikasi, otorisasi dan keamanan fisik.

2. *Integrity* (Integritas)

   Integrity mengacu pada objek yang tetap asli (*original*), dimana objek tidak berubah di perjalanan hingga sampai ke tujuan dari objek tersebut. Sebagai contoh, *email* yang dikirim oleh seseorang bisa dicegat ditengah jalan kemudian diubah isinya dan selanjutnya baru dikirim ke penerima sebenarnya sehingga data yang diterima oleh penerima telah berubah dari yang diinginkan oleh pengirim. Bentuk serangan terhadap aspek *integrity* diantaranya adalah *virus*, *trojan horse*, atau pemakai lain yang berada ditengah komunikasi. Untuk mengatasi hal tersebut, maka perlu dibuat mekanisme proteksi agar data tidak bisa diubah oleh pihak-pihak yang tak diizinkan. *Tool* yang digunakan untuk

menjaga hal itu terlaksana diantaranya adalah *Checksums*, *Data correcting codes* dan *backup*.

3. *Availability* (Ketersediaan)

Availability mengacu pada ketersediaan *resource* dengan tepat, dimana user mempunyai hak akses tepat waktu dan tidak terkendala apapun.

II.4. Ancaman (*Threat*)

II.4.1. *Sniffing*

Menurut Rialda Annisya dan Maynina Norshela Hastuti (2012), *Snifing* merupakan sebuah aksi penyadapan paket data yang dikirimkan sebuah komputer ke *server* tertentu. Terdapat dua jenis aksi *sniffing*, yaitu *passive* dan *active.* Perbedaannya hanyalah jika *active* melakukan aksi perubahan paket data dalam melakukan *sniffing*, sedangkan *passive* tidak.

II.4.2. *Brute Force*

Menurut Rialda Annisya dan Maynina Norshela Hastuti (2012), *Brute force attack* atau dalam bahasa Indonesia disebut juga dengan serangan brute force ini adalah sebuah teknik serangan terhadap sebuah sistem keamanan komputer yang menggunakan percobaan terhadap semua kunci password yang memungkinkan atau istilah gampangnya mungkin menggunakan *Random password* atau *password* acak. Pendekatan ini pada awalnya merujuk pada sebuah program komputer yang mengandalkan kekuatan pemrosesan komputer dibandingkan kecerdasan manusia.

II.5. Kriptografi

Menurut Suriski Sitinjak, et al (2010), "Kriptografi berasal dari bahasa Yunani yaitu *cryptós* yang artinya "*secret*" (yang tersembunyi) dan *gráphein* yang artinya "*writting*" (tulisan)". Jadi, kriptografi berarti "*secret writting*" (tulisan rahasia). Definisi yang dikemukakan oleh Bruce Schneier (1996), "*Cryptography is the art and science of keeping messages secure"* (kriptografi adalah ilmu dan seni untuk menjaga keamanan pesan).

II.5.1. Sejarah Kriptografi

Menurut Budiantoro dan Nanan Rohman (2010), berdasarkan catatan sejarah, kriptografi sudah digunakan oleh bangsa Mesir sejak 4000 tahun yang lalu oleh raja-raja Mesir pada saat perang untuk mengirimkan pesan rahasia kepada panglima perangnya melalui kurir-kurirnya. Orang yang melakukan penyandian ini disebut *kriptografer*, sedangkan orang yang mendalami ilmu dan seni dalam membuka atau memecahkan suatu algoritma kriptografi tanpa harus mengetahui kuncinya disebut *kriptonalis*.

II.6. Algoritma

II.6.1. Pengertian Algoritma

Menurut Rinaldi Munir (2007), Algoritma adalah urutan langkah-langkah untuk memecahkan suatu masalah.

II.6.2. Sejarah Algoritma

Menurut Rinaldi Munir (2007), Algoritma adalah jantung ilmu komputer atau informatika. Banyak cabang dari ilmu komputer yang diacu dalam terminologi Algoritma, misalnya Algoritma perutean (*routing*) pesan di dalam

jaringan komputer, Algoritma *berensenham* untuk menggambar garis lurus (bidang grafik kumputer), Algoritma *Knuth-Morris-Pratt* untuk mencari suatu pola di dalam teks (bidang information *retrievel*), dan sebagainya.

Ditinjau dari asal usul kata, kata "Algoritma" sendiri mempunyai sejarah yang cukup aneh. Kata ini tidak muncul di dalam kamus *Webster* sampai akhir tahun 1957. Orang hanya menemukan kata *Algorism* yang berarti proses menghitung dengan angka Arab. Anda dikatakan *Algorist* jika Anda menggunakan angka Arab. Para ahli bahasa berusaha menemukan asal kata *Algorism* ini, namun hasilnya kurang memuaskan. Akhirnya para ahli sejarah matematika menemukan asal mula kata tersebut. Kata *Algorism* berasal dari nama penulis buku arab yang terkenal, yaitu Abu Ja'afar Muhammad Ibnu Musa al-Khuwarizmi (al-Khuwarizmi dibaca orang barat menjadi *Algorism*). Al-Khuwarizmi menulis buku yang berjudul *Kitab al jabar wal-muqabala*, yang artinya "Buku pemugaran dan pengurangan" (*The book of restoration and reduction*). Dari judul buku ini kita juga memperoleh akar kata "aljabar" (*algebra*). Perubahan dari kata *Algorism* menjadi *Algoritm* muncul karena kata *Algorism* sering dikelirukan dengan *arithmetic*, sehingga akhiran –sm berubah menjadi –thm. Karena perhitungan dengan angka Arab sudah menjadi hal yang biasa/lumrah, maka lambat laun kata *Algorithm* berangsur-angsur dipakai sebagai metode perhitungan (komputasi) secara umum, sehingga kehilangan makna aslinya. Dalam bahasa Indonesia, kata *Algorithm* diserap menjadi "Algoritma".

II.6.3. Algoritma Kriptografi Klasik

Menurut Novi Dian Nathasia dan Anang Eko Wicaksono (2011), Algoritma kriptografi klasik adalah algoritma kriptografi yang berbasis karakter, yaitu enkripsi dan dekripsi dilakukan pada setiap karakter pesan. Algoritma kriptografi klasik dapat dikelompokkan ke dalam dua macam *cipher*:

1. *Cipher* Substitusi

Di dalam *cipher* substitusi setiap unit *plainteks* diganti dengan satu unit *cipherteks*. Algoritma substitusi tertua yang diketahui adalah *Caesar Cipher* yang digunakan oleh kaisar Romawi yang bernama Julius Caesar.

Pada *Caesar Cipher*, tiap huruf disubstitusi dengan huruf ketiga berikutnya dari susunan alfabet yang sama. Dalam hal ini kuncinya adalah jumlah pergeseran huruf yaitu 3. Susunan alfabet setelah digeser sejauh 3 huruf membentuk sebuah tabel substitusi sebagai berikut:

Tabel II.6 Subtitusi *Caesar Cipher*

Plainteks	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipherteks	DEFGHIJKLMNOPQRSTUVWXYZABC

(Sumber : Novi Dian Nathasia dan Anang Eko Wicaksono; 2011)

Jadi, huruf A pada *pleinteks* disubstitusi dengan D, huruf B disubstitusi dengan E, demikian seterusnya. Dengan mengkodekan setiap huruf alfabet dengan *integer* secara matematis pergeseran 3 huruf alfabet ekivalen dengan melakukan operasi *modulo* terhadap *plainteks* P menjadi *cipherteks* C dengan persamaan: C = E(P) = (P+3) mod 26 karena ada 26 huruf di dalam alfabet.

Penerima pesan mengembalikan lagi *cipherteks* (dekripsi) dengan operasi kebalikan, yang secara matematis dapat dinyatakan dengan persamaan: P = D(C)

= (C-3) mod 26. Untuk mengenkripsi pesan yang disusun oleh 256 karakter ASCII, maka persamaan tersebut dapat diperluas menjadi: C = E(P) = (P+3) mod 256 dan fungsi dekripsi adalah: P = D(C) = (C-3) mod 256.

2. *Cipher* Transposisi

Pada *cipher* tranposisi, huruf-huruf di dalam *plainteks* tetap sama, hanya saja urutannya diubah. Nama lain untuk metode ini adalah permutasi atau pengacakan karena metode yang digunakan adalah dengan cara mempermutasikan karakter-karakter yang ada dalam suatu *teks*. Misalkan *plainteks*-nya adalah: ”JURUSAN TEKNIK INFORMATIKA” Untuk mengenkripsi pesan, plainteks ditulis secara horizontal dengan lebar kolom tetap, misal selebar 6 karakter (kunci k=6):

Tabel II.7 Enkripsi *Cipher* Transposisi

Horizontal	*Vertical*
JURUSA NTEKNI KINFOR MATIKA	JNKMUTIARENTUKFISNOKAIRA

(Sumber : Novi Dian Nathasia dan Anang Eko Wicaksono; 2011)

Untuk mendekripsi, kita membagi panjang *cipherteks* dengan jumlah baris (4):

Tabel II.8 Dekripsi *Cipher* Transposisi

Horizontal	*Vertical*
JNKM UTIA RENT UKFI SNOK AIRA	JURUSAN TEKNIK INFORMATIKA

(Sumber : Novi Dian Nathasia dan Anang Eko Wicaksono; 2011)

II.6.4. Algoritma Krigtografi Modern

Menurut Novi Dian Nathasia dan Anang Eko Wicaksono (2011), Kriptografi modern menggunakan gagasan dasar yang sama seperti kriptografi klasik tetapi penekanannya berbeda. Pada kriptografi klasik, kriptografer menggunakan algoritma yang sederhana yang memungkinkan *cipherteks* dapat dipecahkan dengan mudah, antara lain dengan penggunaan statistik, terkaan, teknik analisis frekuensi, dan lain-lain. Algoritma kriptografi modem dibuat sedemikian kompleks sehingga kriptoanalis sangat sulit memecahkan *cipherteks* tanpa mengetahui kunci.

Algoritma kriptografi modem umumnya beroperasi dalam mode *bit* ketimbang mode karakter. Operasi dalam mode *bit* berarti semua data dan informasi baik kunci, *plainteks* maupun *cipherteks* dinyatakan dalam rangkaian *bit biner* 0 dan 1. Algoritma enkripsi dan dekripsi memproses semua data dan informasi dalam bentuk rangkaian *bit*. Rangkaian *bit* yang menyatakan *plainteks* di-enkripsi menjadi *cipherteks* dalam bentuk rangkaian *bit*, dan sebaliknya.

II.6.4.1. Algoritma Asimetris

Menurut Rina Chandra Noer Santi (2010), dalam Algoritma Asimetris *(Asymmetric Algorithms)* ini digunakan dua buah kunci. Satu kunci yang disebut kunci publik (*public key*) dapat dipublikasikan, sedang kunci yang lain yang disebut kunci pribadi (*private key*) harus dirahasiakan. Proses menggunakan sistem ini dapat diterangkan secara sederhana sebagai berikut : bila A ingin mengirimkan pesan kepada B, A dapat menyandikan pesannya dengan menggunakan kunci publik B, dan bila B ingin membaca surat tersebut, ia perlu

mendekripsikan surat itu dengan kunci pribadinya. Dengan demikian kedua belah pihak dapat menjamin asal surat serta keaslian surat tersebut, karena adanya mekanisme ini.

II.6.4.2 Algoritma Simetris

Menurut Dafid (2006), Algoritma simetris dapat pula disebut sebagai algoritma konvensional, dimana kunci dekripsi dapat ditentukan dari kunci enkripsinya, begitu pula sebaliknya. Pada algoritma simetris, kunci enkripsi dan kunci dekripsinya sama.

Dan menurut Rina Chandra Noer Santi (2010), dalam *Symmetric Algorithms* ini, kunci yang digunakan untuk proses enkripsi dan dekripsi pada prinsipnya identik K1 = K2 = K, tetapi satu buah kunci dapat pula diturunkan dari kunci yang lainnya. Kunci-kunci ini harus dirahasiakan. Oleh karena itulah sistem ini sering disebut sebagai *secret-key cipher system*.

II.6.4.2.1 Algoritma *Blowfish*

Menurut Suriski Sitinjak, et al (2010), *Blowfish* diciptakan oleh seorang *Cryptanalyst* bernama Bruce Schneier, Presiden perusahaan *Counterpane Internet Security, Inc* (Perusahaan konsultan tentang kriptografi dan keamanan komputer) dan dipublikasikan tahun 1994. Dibuat untuk digunakan pada komputer yang mempunyai *microprocessor* besar (32-*bit* keatas dengan *cache* data yang besar). *Blowfish* merupakan algoritma yang tidak dipatenkan dan *license free*, dan tersedia secara gratis untuk berbagai macam kegunaan.

Pada saat *Blowfish* dirancang, diharapkan mempunyai kriteria perancangan sebagai berikut (Schneier, 1996):

1. Cepat, *Blowfish* melakukan enkripsi data pada *microprocessors* 32-*bit* dengan *rate* 26 *clock cycles per byte*.
2. *Compact* (ringan), *Blowfish* dapat dijalankan pada memori kurang dari 5K.
3. Sederhana, *Blowfish* hanya menggunakan operasi-operasi sederhana: penambahan, *XOR*, dan *lookup* tabel pada *operan* 32-*bit*.
4. Memiliki tingkat keamanan yang bervariasi, panjang kunci yang digunakan oleh *Blowfish* dapat bervariasi dan bisa sampai sepanjang 448 *bit*.

Dalam penerapannya sering kali Algortima ini menjadi tidak optimal. Karena strategi implementasi yang tidak tepat. Algoritma *Blowfish* akan lebih optimal jika digunakan untuk aplikasi yang tidak sering berganti kunci, seperti jaringan komunikasi atau enkripsi *file* otomatis. Selain itu, karena algoritma ini membutuhkan memori yang besar, maka algoritma ini tidak dapat diterapkan untuk aplikasi yang memiliki memori kecil seperti *smart card*. Panjang kunci yang digunakan, juga mempengaruhi keamanan penerapan algoritma ini.

Algoritma *Blowfish* terdiri atas dua bagian, yaitu ekspansi kunci dan enkripsi data (Schneier, 1996).

1. Ekspansi kunci (*Key-expansion*)

   Berfungsi merubah kunci (minimum 32-*bit*, maksimum 448-*bit*) menjadi beberapa *array* subkunci (*subkey*) dengan total 4168 *byte* (18x32-*bit* untuk P-*array* dan 4x256x32-*bit* untuk S-box sehingga totalnya 33344 *bit* atau 4168 *byte*). Kunci disimpan dalam K-*array*:

   K1, K2, ... Kj 1 _ j _ 14

Kunci-kunci ini yang dibangkitkan (*generate*) dengan menggunakan subkunci yang harus dihitung terlebih

dahulu sebelum enkripsi atau dekripsi data. Sub-sub kunci yang digunakan terdiri dari : P-*array* yang terdiri dari 18 buah 32-*bit* subkunci,

P1, P2, ..., P18

S-*box* yang terdiri dari 4 buah 32-*bit*, masing-masing memiliki 256 entri :

S1,0, S1,1, ..., S1,255

S2,0, S2,1, ..., S2,255

S3,0, S3,1, ..., S3,255

S4,0, S4,1, ..., S4,255

Langkah-langkah perhitungan atau pembangkitan subkunci tersebut adalah sebagai berikut:

a. Inisialisasi P-*array* yang pertama dan juga empat S-*box*, berututan, dengan *string* yang telah pasti. *String* tersebut terdiri dari *digit-digit heksadesimal* dari *phi*, tidak termasuk angka tiga di awal.

   Contoh :

   P1= 0x243f6a88

   P2= 0x85a308d3

   P3= 0x13198a2e

   P4= 0x03707344

   dan seterusnya sampai S-*box* yang terakhir.

b. *XOR*-kan P1 dengan 32-*bit* awal kunci, *XOR*-kan P2 dengan 32-*bit* berikutnya dari kunci, dan seterusnya untuk semua bit kunci. Ulangi siklus

seluruh *bit* kunci secara berurutan sampai seluruh P-*array* ter-*XOR*-kan dengan *bit-bit* kunci. Atau jika disimbolkan : P1 = P1 _ K1, P2 = P2 _K2, P3 = P3 _ K3, . . . P14 = P14 _ K14, P15 = P15 _ K1, . . . P18 = P18 _ K4. Keterangan : _ adalah simbol untuk *XOR*.

c. Enkripsikan string yang seluruhnya *nol* (*all-zero string*) dengan Algoritma *Blowfish*, menggunakan subkunci yang telah didekripsikan pada langkah a dan b.
d. Gantikan P1 dan P2 dengan keluaran dari langkah c.
e. Enkripsikan keluaran langkah c menggunakan algoritma *Blowfish* dengan subkunci yang telah dimodifikasi.
f. Gantikan P3 dan P4 dengan keluaran dari langkah e.
g. Lanjutkan langkah-langkah di atas, gantikan seluruh elemen P-*array* dan kemudian keempat S-*box* secara berurutan, dengan hasil keluaran Algoritma *Blowfish* yang terus-menerus berubah. Total keseluruhan, terdapat 521 iterasi untuk menghasilkan subkunci-subkunci dan membutuhkan memori sebesar 4*KB*.

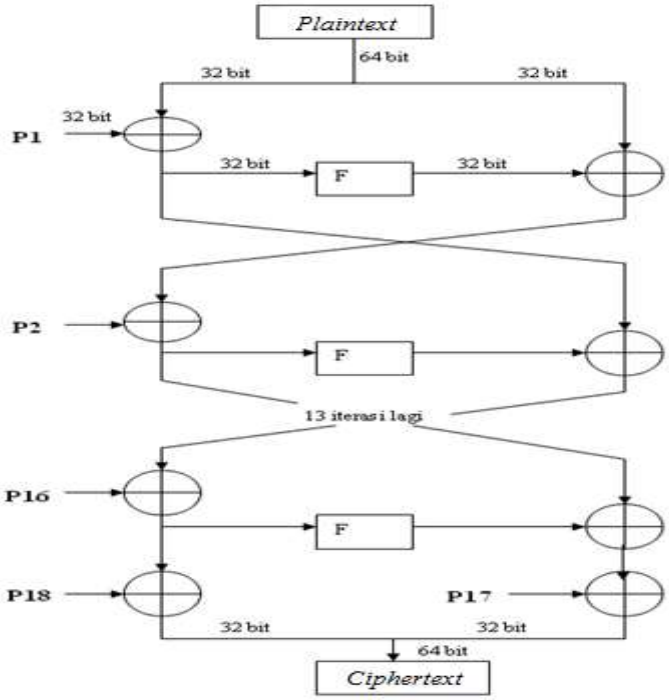
2. Enkripsi Data

Terdiri dari iterasi fungsi sederhana (*Feistel Network*) sebanyak 16 kali putaran (iterasi), masukannya adalah 64-*bit* elemen data X. Setiap putaran terdiri dari permutasi kunci-dependent dan substitusi kunci serta data-dependent. Semua operasi adalah penambahan (*addition*) dan *XOR* pada variabel 32-*bit*. Operasi tambahan lainnya hanyalah empat penelusuran tabel *array* berindeks untuk setiap putaran. Langkahnya adalah seperti berikut.

a. Bagi X menjadi dua bagian yang masing-masing terdiri dari 32-bit: XL, XR.
b. Lakukan langkah berikut

   For i = 1 to 16, XL = XL _ Pi, XR = F(XL) _ XR, Tukar XL dan XR.
c. Setelah iterasi ke-16, tukar XL dan XR lagi untuk melakukan membatalkan pertukaran terakhir.
d. Lalu lakukan XR = XR _ P17, XL = XL _ P18.
e. Terakhir, gabungkan kembali XL dan XR untuk mendapatkan *cipherteks*.

Untuk lebih jelasnya, gambaran tahapan pada jaringan *feistel* yang digunakan *Blowfish* adalah seperti pada Gambar II.1.

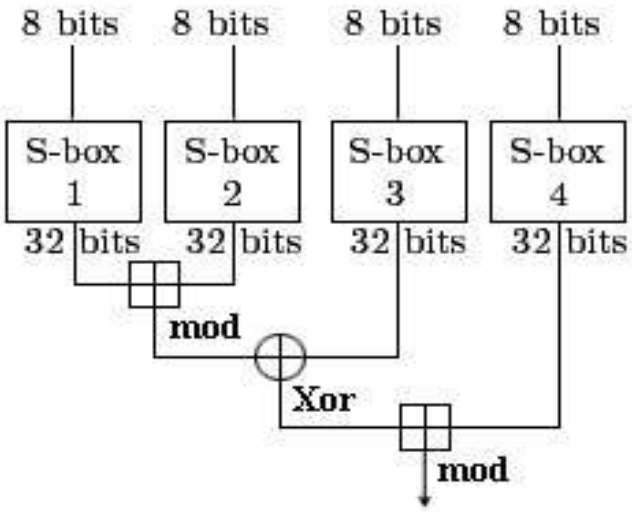


Gambar II.1. Blok Diagram Enkripsi *Blowfish*
(Sumber : Suriski Sitinjak, et al; 2010)

Pada langkah kedua, telah dituliskan mengenai penggunaan fungsi F. Fungsi F adalah: bagi XL menjadi empat bagian 8-*bit*: a,b,c dan d.

F(XL) = ((S1,a + S2,b *mod* 2^{32}) *XOR* S3,c) + S4,d *mod* 2^{32} .....(2.1)

Agar dapat lebih memahami fungsi F, tahapannya dapat dilihat pada Gambar II.2.



Gambar II.2. Fungsi F dalam *Blowfish*
(Sumber : Suriski Sitinjak, et al; 2010)

Dekripsi sama persis dengan enkripsi, kecuali bahwa P1, P2,…, P18 digunakan pada urutan yang berbalik (*reverse*). Algoritmanya dapat dinyatakan sebagai berikut (Schneier, 1996) :

for i = 1 to 16 do

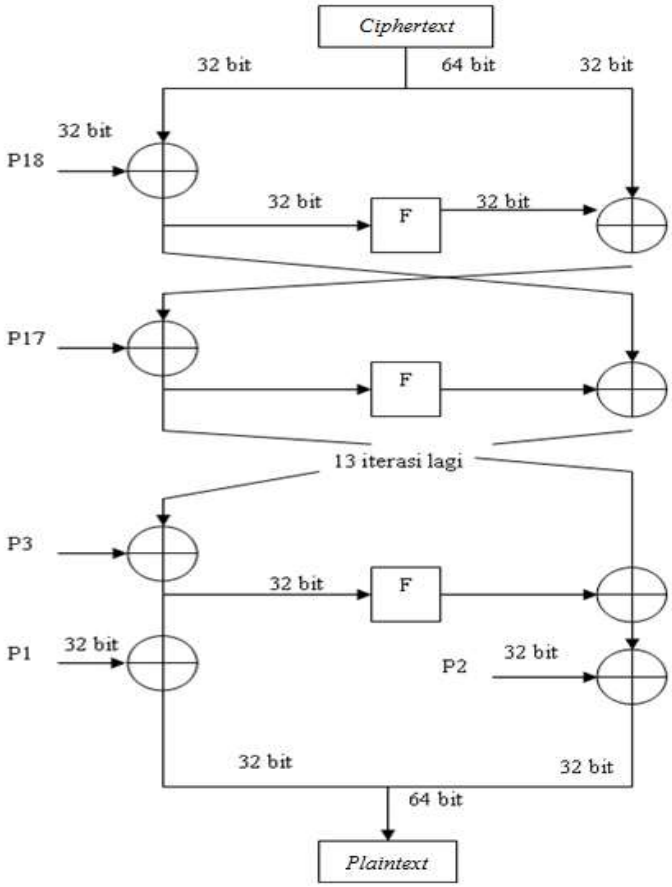
XRi = XLi-1 _ P19-i;

XLi = F[XRi] _ XRi-1;

XL17 = XR16 _ P1;

XR17 = XL16 _ P2;

Blok diagram dekripsi seperti pada Gambar II.3.



Gambar II.3. Blok Diagram Dekripsi *Blowfish*
(Sumber : Suriski Sitinjak, et al ; 2010)

II.7. Pengertian *Client Server*

Definisi *Client Server* menurut R Achmad Margoyuwono (2013), yaitu: *server* adalah komputer *database* yang berada di pusat, dimana informasinya dapat digunakan bersama-sama oleh beberapa *user* yang menjalankan aplikasi di dalam komputer lokalnya yang disebut dengan "*Client"*. Sedangkan Definisi *Client Server* menurut Arief Ramadhan (2005) yaitu: *"client* dan *Server* pada dasarnya tidaklah berarti dua buah komputer yang berbeda. *Client* dan *Server* adalah dua buah aplikasi yang berjalan dan saling berinteraksi satu sama lain

sehingga aplikasi *Client* dan *Server* bisa saja berada bersama dalam satu buah komputer secara sekaligus".

II.8. *Android*

Menurut Busran dan Fitriyah (2015), *Android* adalah sistem operasi berbasis *Linux* yang dirancang untuk perangkat seluler layar sentuh seperti telepon pintar dan komputer *tablet. Android* awalnya dikembangkan oleh *Android, Inc*., dengan dukungan finansial dari *Google*, yang kemudian membelinya pada tahun 2005. Sistem operasi ini dirilis secara resmi pada tahun 2007, bersamaan dengan didirikannya *Open Handset Alliance*, konsorsium dari perusahaan-perusahaan perangkat keras, perangkat lunak, dan telekomunikasi yang bertujuan untuk memajukan standar terbuka perangkat seluler.

Tabel II.9 Versi *Android*

Android	***Code Name***	**Tanggal Peluncuran**
Android versi 1.1	-	09 Maret 2009
Android versi 1.5	*Cupcake*	30 April 2009
Android versi 1.6	*Donut*	15 September 2009
Android versi 2.0/2.1	*Eclair*	26 Oktober 2009
Android versi 2.2	*Frozen Yoghurt (Froyo)*	20 Mei 2010
Android versi 2.3	*Gingerbread*	06 Desember 2010
Android versi 3.0/3.1	*Honeycomb*	22 Februari 2011
Android versi 4.0	*Ice Cream Sandwich (ICS)*	19 Oktober 2011
Android versi 4.1/4.2	*Jellybean*	09 Juli 2012
Android versi 4.4	*Kitkat*	31 Oktober 2013

(Sumber : Dhanar Intan Surya Saputra; 2013)

II.9. *Android SDK*

Menurut Tavipia Rumambi, et al (2014), *Android SDK* adalah *tools API* (*Application Programming Interface*) yang diperlukan untuk memulai

mengembangkan aplikasi pada platform *Android* menggunakan bahasa pemrograman *Java*.

II.10. *Eclipse*

Menurut Gusti Ngurah Wira Satryawan, et al (2014), *Eclipse* adalah sebuah *IDE* (*Integrated Development Environment*) untuk mengembangkan perangkat lunak dan dapat dijalankan di semua *platform* (*platform independent*). *Eclipse* pada saat ini merupakan salah satu *IDE* favorit dikarenakan gratis dan *open source*. Kelebihan dari *Eclipse* yang membuatnya populer fasilitas *plug-in* yang dimilikinya, dengan menggunakan *plug-in* membuat *Eclipse* dapat digunakan untuk mengembangkan pemrograman selain *Java* untuk berbagai macam keperluan. Pengembangan aplikasi *Android* menggunakan *Eclipse*, menggunakan bahasa *Java* dan *plug-in Android Development Tools* (*ADT*). Aplikasi Android yang telah dibuat di *Eclipse* dapat dijalankan menggunakan *AVD* (*Android Virtual Device*), sehingga kita tidak harus memerlukan perangkat Android asli.

II.11. *XAMPP*

Menurut Satriawaty Mallu (2015), *XAMPP* dari *Apache, MYSQL, PHP* dan *Perl* adalah perangkat lunak bebas, yang mendukung banyak sistem operasi, merupakan kompilasi dari beberapa program. XAMPP memiliki arti sebagai berikut:

X: Program ini dapat dijalankan dibanyak sistem operasi seperti *Windows, Linux*, *Mac OS* dan juga *Solaris*.

A: *Apache*, merupakan aplikasi *web server*. Tugas utama Apache adalah menghasilkan halaman *web* kepada user berdasarkan kode *PHP* yang dituliskan oleh pembuat *web*.

M : *MySQL* merupakan aplikasi *database server*, bahasa terstruktur yang digunakan untuk membuat dan mengelola *database* beserta isinya pengguna dapat memanfaatkan *MySQL* untuk menambahkan, mengubah, dan menghapus data yang berada dalam *database*.

P : *PHP*, bahasa pemrograman *web.* Bahasa pemprograman PHP merupakan bahasa pemprograman untuk membuat *web* yang bersifat *server-side scripting*.

P : *Perl* adalah bahasa pemprograman untuk segala keperluan, dikembangkan, pertama kali oleh Larry Wall di mesin *Unix*.

II.12. *PHP*

Menurut Muhammad Taufiq Muslih dan Bambang Eka Purnama (2013) *PHP* adalah singkatan dari "*PHP*: *Hypertext Preprocessor*", yang merupakan sebuah bahasa *scripting* yang terpasang pada *HyperText Markup Language* (*HTML*). Sebagian besar *sintaks* mirip dengan bahasa C, *Java* dan *Perl*, ditambah beberapa fungsi *PHP* yang spesifik. Tujuan utama penggunaan bahasa ini adalah untuk memungkinkan perancang *web* menulis halaman *web* dinamik dengan cepat.