BAB II
TINJAUAN PUSTAKA

2.1 Konsep Dasar data

2.1.1 *File*

File atau Berkas adalah sekumpulan data (informasi) yang berhubungan yang diberi nama dan tersimpan di dalam media penyimpanan sekunder (*secondary storage*). *File* memiliki ekstensi. Ekstensi berkas merupakan penandaan jenis berkas lewat nama berkas. Ekstensi biasanya ditulis setelah nama berkas dipisahkan dengan sebuah tanda titik. Pada sistem yang lama (MS-DOS) ekstensi hanya diperbolehkan maksimal 3 huruf, contohnya : *exe, bat, com, txt*. Batasan itu dihilangkan pada sistem yang lebih baru (Windows), contohnya : *mpeg*, java. Pada UNIX bahkan dikenal ada *file* yang memiliki lebih dari satu ekstensi, contohnya : *tar. z, tar. gz* (Sukrisno & Utami, 2007).

Struktur pada *file* terdiri dari 2 bagian yaitu *header file* dan isi *file*. Pada *header file* terdapat kode *biner* maupun kode ASCII yang berisikan tentang fungsi utama pada *file*. Pada isi *file* terdapat isi dari *file* yang telah terbentuk baik berupa *text*, lagu, *video*, dll. Di bawah ini merupakan gambar struktur *file* yang ada pada semua *file*.

Gambar 1 : Struktur *File*

2.1.2 Operator Logika

Operator logika adalah simbol-simbol yang digunakan untuk melakukan ekspresi terhadap data-data logika. Proses operasi tersebut akan menghasilkan salah satu dari dua jenis nilai kebenaran yaitu *TRUE* dan *FALSE* atau 1 dan 0.

Simbol-simbol operator logika tersebut dapat dilihat pada tabel yang ada di bawah ini :

Tabel 1 : Tabel Operator Logika

Operator	Keterangan
Not	Tidak
And	Dan
Or	Atau
Xor	*Exclusive Or*

2.1.3 XOR

Operator biner XOR banyak digunakan dalam perhitungan biner untuk algoritma kriptografi tertentu. Notasi matematis untuk operator XOR adalah $\oplus$. Operator XOR merupakan operator yang digunakan untuk dua buah ekspresi. Operator XOR akan menghasilkan nilai *true* atau 1 jika kedua ekpresi memiliki nilai yang berbeda. Operator xor akan menghasilkan nilai 0 atau 0 jika kedua espresi bernilai sama.

Tabel 2 : Aturan Nilai Kebenaran pada Operator XOR (Nathasia & Wicaksono, 2011)

Ekspresi1	Ekspresi2	Ekspresi 1 XOr Ekspresi2
0	0	0
0	1	1
1	0	1
1	1	0

2.2 Kriptografi

2.2.1 Sejarah Kriptografi

Kriptografi merupakan sebuah ilmu yang digunakan untuk penyandian data. Kriptografi telah dikenal dan dipakai cukup lama sejak kurang lebih tahun 1900 sebelum masehi pada prasasti-prasasti kuburan.

Ilmu Kriptografi sebenarnya sudah mulai dipelajari manusia sejak tahun 400 SM, yaitu pada zaman Yunani kuno. Dari catatan bahwa "Penyandian Transposisi" merupakan sistem kriptografi pertama yang digunakan atau dimanfaatkan. Bidang ilmu ini terus berkembang seiring dengan kemajuan peradaban manusia, dan memegang peranan penting dalam strategi peperangan yang terjadi dalam sejarah manusia, mulai dari sistem kriptografi "*Caesar Chiper*" yang terkenal pada zaman Romawi kuno, "*Playfair Cipher*" yang digunakan Inggris dan "*ADFVGX Cipher*" yang digunakan Jerman pada Perang Dunia I hingga algoritma-algoritma kriptografi rotor yang populer pada Perang Dunia II , seperti *Sigaba* / M-134 (Amerika Serikat), *Typex* (Inggris), *Purple* (Jepang), dan mesin kriptografi legendaris *Enigma* (Jerman). Sejarah telah dipenuhi oleh contoh-contoh orang yang berusaha merahasiakan informasi rahasia mereka dari orang lain.

Seiring dengan perkembangan zaman, kebutuhan akan metode yang lebih canggih tidak dapat dihindari. Sekarang, dengan adanya era informasi, kebutuhan itu menjadi lebih penting lagi. Dengan adanya fasilitas internet, maka permintaan akan pelayanan informasi semakin meningkat dengan seiringnya perkembangan teknologi. Pertukaran data yang sensitif seperti nomor *account* kartu kredit, sudah sering dilakukan dan menjadi hal yang biasa di dalam dunia internet. Karena itu, melindungi data sudah menjadi hal penting yang sangat kruisal di dalam hidup.

Ada tiga istilah yang berkaitan dengan proteksi data yaitu kriptografi, kriptologi, dan kriptanalisis. Arti ketiganya kurang lebih sama. Secara teknis, kriptologi adalah ilmu yang mempelajari tentang komunikasi pada jalur yang tidak aman beserta masalah-masalah yang berhubungan dengan itu.

Kriptografi berasal dari kata "*crypto*" yang berarti rahasia dan "*graphy*" yang berarti tulisan. Jadi, dapat dikatakan bahwa kriptografi adalah tulisan yang tersembunyi. Dengan adanya tulisan yang tersembunyi ini, orang-orang tidak

mengetahui bagaimana tulisan tersebut disembunyikan dan tidak mengetahui bagaimana cara membaca maupun menerjemahkan tulisan tersebut. William Stallings mendefinisikan kriptografi sebagai "*the art and science of keeping messages secure*".

Kriptografi berbasis pada algoritma pengkodean data informasi yang mendukung kebutuhan dari dua aspek keamanan informasi, yaitu *secrecy* (perlindungan terhadap kerahasiaan data informasi) dan *authenticity* (perlindungan terhadap pemalsuan dan pengubahan informasi yang tidak diinginkan.

Kriptografi menjadi dasar bagi keamanan komputer dan jaringan karena merupakan sarana bagi distribusi data dan informasi. Sehingga data dan informasi tersebut harus diamankan agar hanya orang-orang yang berhak mengaksesnya yang dapat mengetahui maupun menggunakan data tersebut. Salah satu cara yang paling banyak digunakan dalam mengamankan data adalah dengan kriptografi. Data-data tersebut diamankan dengan sedemikian rupa oleh pengirim sehingga orang lain tidak dapat mengenali data tersebut.

Pembakuan penulisan pada kriptografi dapat ditulis dalam bahasa matematika. Fungsi-fungsi yang mendasar dalam kriptografi adalah enkripsi dan dekripsi. Enkripsi adalah proses mengubah suatu pesan asli (*plaintext*) menjadi suatu pesan dalam bahasa sandi (*ciphertext*).

C = E (M), dimana :

M = pesan asli

E = proses enkripsi

C = pesan dalam bahasa sandi (untuk ringkasnya disebut sandi)

Sedangkan dekripsi adalah proses mengubah pesan dalam suatu bahasa sandi menjadi pesan asli kembali.

M = D (C)

D = proses dekripsi

Umumnya, selain menggunakan fungsi tertentu dalam melakukan enkripsi dan dekripsi, seringkali fungsi itu diberi parameter tambahan yang disebut dengan istilah kunci.

2.2.2 Aspek-aspek dalam Kriptografi

Kriptografi yang baik tidak ditentukan oleh kerumitan dalam mengolah data atau pesan yang akan disampaikan. Ada 4 syarat yang perlu dipenuhi, yaitu:

1. **Kerahasiaan**. Pesan (*plaintext*) hanya dapat dibaca oleh pihak yang memliki kewenangan.
2. ***Autentikasi***. Pengirim pesan harus dapat diidentifikasi dengan pasti, penyusup harus dipastikan tidak bisa berpura-pura menjadi orang lain.
3. **Integritas**. Penerima pesan harus dapat memastikan bahwa pesan yang dia terima tidak dimodifikasi saat dalam proses transmisi data.
4. ***Non-Repudiation***. Pengirim pesan harus tidak bisa menyangkal pesan yang dia kirimkan.

2.2.3 Proses dalam Kriptografi

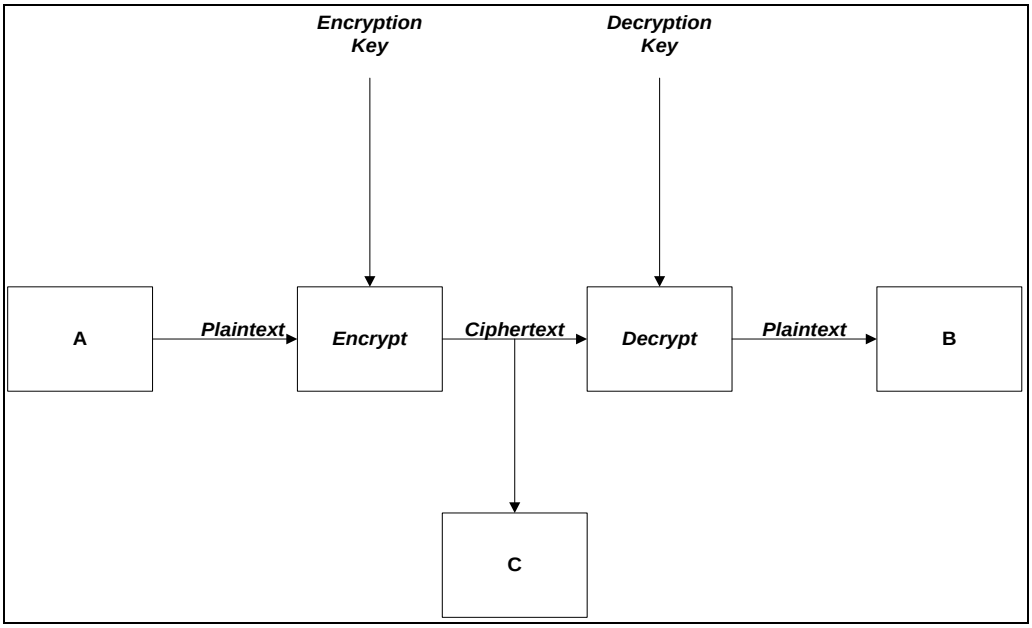
a. Enkripsi

Proses menyandikan *plainteks* menjadi *chiperteks* disebut enkripsi (*encryption*) atau *enciphering* (standard nama menurut ISO 7498-2) sedangkan proses mengembalikan *chiperteks* mejadi *plainteks* disebut dekripsi (*decryption)* atau *dechiphering* (standard ISO 7498-2).

Enkripsi adalah istilah dalam kriptografi yang berarti proses menyandikan suatu data atau informasi berbentuk teks menjadi bentuk lain yang tidak dapat dipahami (Nathasia & Wicaksono, 2011). Tujuannya adalah untuk meyakinkan *privasi* dengan menyembunyikan informasi dari orang-orang yang tidak ditujukan, bahkan mereka yang memiliki akses ke data terenkripsi. Sedangkan dekripsi merupakan kebalikan dari enkripsi, yaitu transformasi data terenkripsi kembali ke bentuknya semula.

Enkripsi dilakukan pada saat pengiriman dengan cara mengubah data asli menjadi data rahasia, sedangkan dekripsi dilakukan pada saat penerimaan dengan cara mengubah data rahasia menjadi data asli. Jadi data yang dikirimkan selama proses pengiriman adalah data rahasia, sehingga data asli tidak dapat diketahui oleh pihak yang tidak berkepentingan. Data asli hanya dapat diketahui oleh penerima dengan menggunakan kunci rahasia.

Enkripsi dan dekripsi pada umumnya membutuhkan penggunaan sejumlah informasi rahasia, disebut sebagai kunci. Untuk beberapa mekanisme enkripsi, kunci yang sama digunakan baik untuk enkripsi dan dekripsi berbeda.



Gambar 2 : Skenario Komunikasi Dasar Kriptografi

b. Dekripsi

Di dalam skenario komunikasi dasar, seperti yang diperlihatkan pada Gambar 2, terdapat dua belah pihak, sebut saja A dan B, yang ingin berkomunikasi satu sama lain.

Kemudian pihak ketiga, C, adalah seorang *eavesdropper* (orang yang mengakses informasi rahasia tanpa izin). Ketika A ingin mengirimkan informasi, yang disebut *plaintext*, kepada B, dia mengenkripsi *plaintext* tersebut dengan menggunakan metode yang telah dirancang oleh B. Biasanya, metode enkripsi diketahui oleh si *eavesdropper*, dalam hal ini adalah C. Yang membuat pesan tersebut tetap bersifat rahasia adalah *key*-nya. Ketika B menerima pesan yang telah dienkripsi, yang disebut dengan *ciphertext*, dia mengubahnya kembali menjadi *plaintext* dengan menggunakan *key* dekripsi.

Dekripsi merupakan proses kebalikan dari enkripsi dimana proses ini akan mengubah *ciphertext* menjadi *plaintext* dengan menggunakan algortima ‘pembalik’ dan *key* yang sama.

Contoh:

Ciphertext	*Plaintext*
Khoos	hello
Ebh	bye
iuhhobqa	freelynx

Untuk melakukan dekripsi ini, algoritma yang digunakan tentu saja berbeda dengan algortima enkripsi, namun pada dasarnya adalah "membalik" algoritma enkripsi. Perhatikan contoh algortima dekripsi berikut yang diambil dari algoritma enkripsi diatas:

Konversikan huruf menjadi angka, masing-masing angka yang diperoleh kurangkan dengan *$n*, kemudian konversikan angka yang diperoleh kembali menjadi huruf dengan *$n* adalah key yang sama dengan yang digunakan dalam proses enkripsi.

Jika diperhatikan, baik algoritma enkripsi maupun dekripsi tidak jauh berbeda, yang berbeda hanyalah ketika memasukkan unsur *key* kedalam algoritma tersebut, dimana enkripsi menggunakan proses penjumlahan sedangkan dekripsi menggunakan pengurangan. Masih menggunakan contoh yang sama dengan enkripsi, apabila kita gunakan *key* sama dengan 1, algoritma dekripsi diatas akan mengubah huruf:

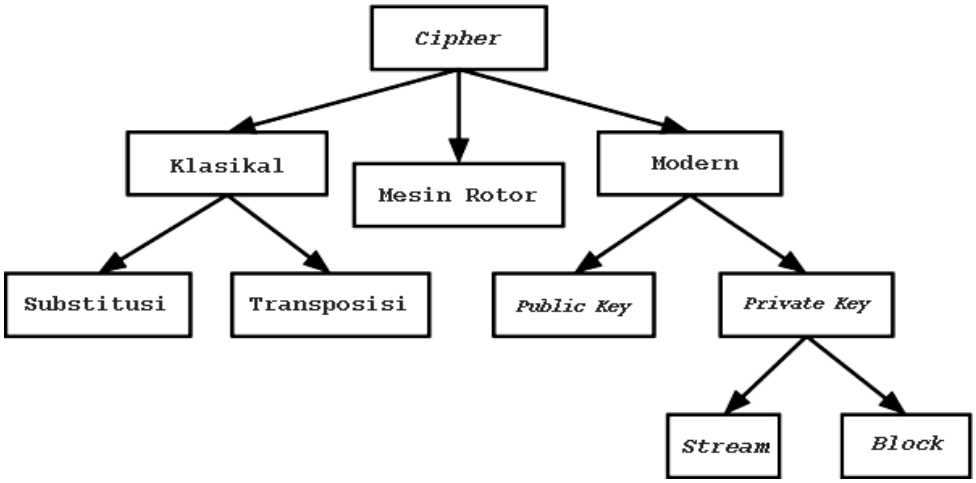
B menjadi A
C menjadi B
Z menjadi Y
A menjadi Z

2.2.4 Algoritma Kriptografi Modern

Kriptografi dibagi dalam dua bagian, yaitu *cipher* modern dan klasikal. Mode klasikal biasanya menggunakan mode karakter, sehingga kemampuan dari mode ini terbatas karena keterbatasan jumlah karakter.

Sedang pada kriptografi modern, pada proses enkripsi dan dekripsi menggunakan rangkaian bit dalam bentuk biner, 0 dan 1. Rangkaian bit yang

menyatakan *plainteks* dienkripsi menjadi *ciphertext* dalam bentuk rangkaian bit, begitu pula sebaliknya.



Gambar 3 : Klasifikasi Kriptografi

Perkembangan algoritma kriptografi modern berbasis bit didorong oleh penggunaan komputer digital yang merepresentasikan data dalam bentuk biner. Algoritma kriptografi yang beroperasi dalam mode bit dapat dikelompokkan menjadi dua kategori yaitu *block cipher* dan *stream cipher*.

2.2.5 *Block Cipher*

Algoritma kriptografi beroperasi pada *plainteks/cipherteks* dalam bentuk blok bit, yang dalam hal ini rangkaian bit dibagi menjadi blok-blok bit yang panjangnya sudah ditentukan sebelumnya. Misalnya panjang blok adalah 64 bit, maka itu berarti algoritma enkripsi memerlukan 8 karakter setiap kali enkripsi (1 karakter = 8 bit dalam pengkodean ASCII). *Cipher* blok mengenkripsi satu blok bit setiap kali.

Pada *cipher* blok, rangkaian bit-bit *plainteks* dibagi menjadi blok-blok bit dengan panjang sama, biasanya 64 bit (tapi adakalanya lebih). Algoritma enkripsi menghasilkan blok *cipherteks*, pada kebanyakan sistem kriptografi simetri yang berukuran sama dengan blok *plainteks*.

Dengan blok *cipher*, blok *plainteks* yang sama akan dienkripsi menjadi blok *cipherteks* yang sama bila digunakan kunci yang sama pula. Ini berbeda

dengan *cipher* aliran dimana bit-bit *plainteks* yang sama akan dienkripsi menjadi bit-bit *cipherteks* yang berbeda setiap kali dienkripsi.

Misalkan blok plainteks (P) yang berukuran m bit dinyatakan sebagai vector P = (p1, p2, …, pm) yang dalam hal ini pi adalah 0 atau 1 untuk i = 1, 2, …, m, dan blok *cipherteks* (C) adalah C = (c1, c2, …, cm) yang dalam hal ini ci adalah 0 atau 1 untuk i = 1, 2, …, m.

Bila *plainteks* dibagi menjadi n buah blok, barisan blok-blok *plainteks* dinyatakan sebagai (P1, P2, …, Pn). Untuk setiap blok *plainteks* Pi, bit-bit penyusunnya dapat dinyatakan sebagai vector Pi = (pi1, pi2, …, pim). Enkripsi dan dekripsi dengan kunci K dinyatakan berturut-turut dengan persamaan

$$\mathbf{EK(P) = C} \qquad \mathbf{(1)}$$

untuk enkripsi, dan

$$\mathbf{DK(C) = P} \qquad \mathbf{(2)}$$

Fungsi E haruslah fungsi yang berkoresponden satu-ke-satu, sehingga *E-1 = D*.

2.2.6 *Stream Cipher*

Chiper aliran *(stream cipher)* merupakan salah satu tipe algoritma kriptografi kriptografi simetri. *Chiper* aliran mengenkripsikan *plainteks* menjadi *chiperteks* bit per bit (1 bit setiap kali transformasi).

a. Gambaran Umum dalam *Stream Cipher*

Stream cipher merupakan algoritma kriptografi yang ditemukan oleh Mayor J. Maugborne dan G. Vernam. Algoritma ini merupakan algoritma berjenis *symetric key* yang artinya bahwa kunci yang digunakan untuk melakukan enkripsi dan dekripsi merupakan kunci yang sama. Dalam proses enkripsi, algoritma ini menggunakan cara *stream cipher* dimana *cipher* berasal dari hasil XOR antara bit *plaintext* dan bit key (Nathasia & Wicaksono, 2011). Algoritma *stream cipher* diadopsi dari one-time pad *cipher*, dimana dalam hal ini karakter

diganti dengan bit (0 atau 1). Dengan kata lain, *stream cipher* merupakan versi lain dari *one-time pad cipher*.

Dalam proses enkripsi, *cipherteks* diperoleh dengan melakukan penjumlahan *modulo 2* satu bit *plainteks* dengan satu bit kunci, seperti terlihat pada rumus di bawah ini :

$$\mathbf{c_1 = (p_1 + k_1) \bmod 2} \qquad \mathbf{(3)}$$

Dimana :

c_1 = *cipherteks*

p1 = *plainteks*

k1 = kunci

Sedangkan dalam proses dekripsi, untuk mendapatkan kembali *plainteks*, diperoleh dengan melakukan penjumlahan *modulo* 2 satu bit *cipherteks* dengan satu bit kunci :

$$\mathbf{p_1 = (\, c_1 - k_1\,) \bmod 2} \qquad \mathbf{(4)}$$

Pada *cipher* aliran, bit hanya mempunya dua buah nilai, sehingga proses enkripsi hanya menyebabkan dua keadaan pada bit tersebut, yaitu berubah atau tidak berubah. Dua keadaan tersebut ditentukan oleh kunci enkripsi yang disebut dengan aliran-bit-kunci (*keystream*). Oleh karena operasi penjumlahan *modulo* 2 identik dengan operasi bit dengan operator XOR, maka persamaan 4 dapat ditulis secara sederhana sebagai berikut (Nathasia & Wicaksono, 2011):

$$\mathbf{c_1 = p_1 \text{ XOR } k_1} \qquad \mathbf{(5)}$$

Sedangkan pada proses pendekripsian dituliskan:

$$\mathbf{_{p1} = c_1 \text{ XOR } k_1} \qquad \mathbf{(6)}$$

Dalam operator logika XOR, hasil akan T (benar) apabila salah satu dari kedua operand (tetapi tidak keduanya) bernilai T atau 1. Atau dengan kata lain, apabila diaplikasikan dalam bit maka operator XOR akan menghasilkan 1 jika dan hanya jika salah satu operand bernilai 1.

Contoh :

X : 00111010 10101011

Y : 10100100 01010101

Hasil : 10011110 11111110

Sedangkan suatu bilangan dalam biner apabila di- XOR-kan dengan dirinya sendiri akan menghasilkan 0.

Contoh :

X : 01010101 10101010

Y : 01010101 10101010

Hasil : 00000000 00000000

b. Proses Enkripsi dan Deskripsi dengan *Stream Cipher*

Perhitungan di atas merupakan salah satu dasar dalam penerapan algoritma Stream dalam kriptografi, yaitu suatu *string* yang diterjemahkan ke dalam biner dapat dienkripsikan dengan suatu kunci tertentu dan dapat pula diperoleh kembali dari pesan sandi dengan menggunakan operator XOR pada kunci yang sama (Irfianti, 2007).

Dalam algoritma ini, terdapat beberapa langkah untuk proses enkripsi dan dekripsi. Misalnya kita akan mengenkripsi *plainteks* "*Stream*" dengan kunci "*Cipher*". Maka langkah-langkahnya adalah :

1. Karakter-karakter yang terdapat pada *plainteks* dan kunci merupakan karakter ASCII. Maka, ubah *plainteks* dan kunci menjadi bilangan biner :

ASCII	Biner
Stream	01010110 01100101 01110010 01101110 01100001 01101101
Cipher	01000011 01101000 01101001 01110000 01100101 01110010

2. Lalu, kedua bilangan biner itu kita XOR-kan menurut persamaan **5 :**

Plainteks : 01010110 01100101 01110010 01101110 01100001 01101101

Kunci : 01000011 01101001 01110000 01101000 01100101 01110010

Cipherteks : 00010101 00001100 00000010 00000110 00000100 00011111

Hasil dari XOR tersebut adalah : "00010101 00001100 00000010 00000110 00000100 00011111". Rangkaian bilangan bit ini merupakan bit *cipherteks* dalam bentuk biner. Untuk mengetahui nilai ASCII dari *cipherteks* tersebut, maka kita perlu konversikan rangkaian biner tersebut ke bilangan ASCII sesuai dengan tabel 2. 1. Dengan berdasar pada tabel tersebut, maka dapat kita lihat bahwa rangkaian bit dari *cipherteks* tersebut mempunyai nilai ASCII :

§ ♀ ☻ ♠ ♦ ▼

Proses dekripsi dalam algoritma *Stream Cipher* merupakan kebalikan dari proses enkripsi (Iswahyudi et al. , 2012). *Cipherteks* dari hasil enkripsi di-XOR-kan dengan kunci yang sama. Misalnya dengan mengambil contoh sebelumnya, dimana *cipherteks* : "§ ♀ ☻ ♠ ♦ ▼" dan kunci : "*Cipher*". Maka langkah pendekripsiannya adalah sebagai berikut :

1. Ubah karakter ASCII dari cipherteks dan kunci ke dalam rangkaian biner :

ASCII	Biner
§ ♀ ☻ ♠ ♦ ▼	00010101 00001100 00000010 00000110 00000100 00011111
Cipher	01000011 01101000 01101001 01110000 01100101 01110010

2. Lalu, kedua rangkaian biner itu kita XOR-kan dengan berdasar pada persamaan **6** :

Cipherteks : 00010101 00001100 00000010 00000110 00000100 00011111
Kunci : 01000011 01101001 01110000 01101000 01100101 01110010
Plainteks : 01010110 01100101 01110010 01101110 01100001 01101101

Maka didapat rangkaian bit plainteks "01010110 01100101 01110010 01101110 01100001 01101101". Dengan berdasar pada tabel 2. 1, maka kita dapat mengubah rangkaian bit menjadi bilangan ASCII, yaitu menjadi : "*Stream*". Terbukti *plainteks* pada hasil dekripsi adalah sama dengan *plainteks* pada proses enkripsi.

2.2.7 Penelitian Terkait terhadap *Stream Chiper* XOR

Algoritma *Stream Cipher* telah diterapkan oleh Natashia dkk (Nathasia & Wicaksono, 2011), dalam papernya yang berjudul "Penerapan Teknik Kriptografi *Stream Cipher* untuk Pengaman Basis Data". *Stream Ciper* yang merupakan salah satu teknik yang terkenal dalam teknik Kriptografi, dinilai dapat menghasilkan tingkat keamanan data yang tinggi. Hal ini terbukti dari hasil penelitian yang telah dilakukan. *Stream Cipher* sendiri merupakan salah satu bentuk dari algoritma kriptografi modern yang memproses *plainteks* per bit pada saat proses enkripsi/dekripsi, dimana algoritma *Stream Cipher* tidak rumit dan mudah untuk diterapkan. Kunci yang digunakan kemudian XOR-kan untuk membentuk aliran bit *cipherteks*, dan di XOR-kan untuk yang kedua kali untuk menghasilkan *plainteks*. Dengan adanya pembangkit aliran kunci *(keystream generator)* yang secara maksimal membentuk aliran kunci yang acak maka enkripsi data berada pada tingkat kemanan yang tinggi. Keuntunggan cipher aliran yang lain yaitu dapat digunakan pada sistem keamanan misalnya protokol jaringan. Pada penelitian ini, *Stream cipher* diterapkan untuk mengamankan data yang berada pada sebuah database. Penggunaan bahasa pemrograman *Delphi* dan pembuatan database melalui *Ms. Access* telah berhasil diterapkan. *Stream Cipher* diterapkan pada enkripsi nomor pin nasabah suatu sistem perbankan. Dari penelitian ini dapat disimpulkan bahwa algoritma *Stream Cipher* mempunyai keunggulan untuk digunakan sebagai algoritma pengamanan data.

2.3 Steganografi

2.3.1 Sejarah Steganografi

Steganografi berasal dari bahasa Yunani yaitu "*steganós*" yang berarti menyembunyikan dan "*graptos*" yang artinya tulisan sehingga secara keseluruhan artinya adalah tulisan yang disebunyikan. Secara umum steganografi merupakan seni atau ilmu yang digunakan untuk menyembunyikan pesan rahasia dengan segala cara sehingga selain orang yang dituju, orang lain tidak akan menyadari keberadaan dari pesan rahasia tersebut (Nani, 2011).

Kini, istilah steganografi termasuk penyembunyian data digital dalam *file-file* komputer. Contohnya, si pengirim mulai dengan *file* gambar biasa, lalu mengatur warna setiap *pixel* ke-100 untuk menyesuaikan suatu huruf dalam

alphabet (perubahannya begitu halus sehingga tidak ada seorangpun yang menyadarinya jika ia tidak benar-benar memperhatikannya).

Pada umumnya, pesan steganografi muncul dengan rupa lain seperti gambar, artikel, daftar belanjaan, atau pesan-pesan lainnya. Pesan yang tertulis ini merupakan tulisan yang menyelubungi atau menutupi. Contohnya, suatu pesan bisa disembunyikan dengan menggunakan tinta yang tidak terlihat diantara garis-garis yang kelihatan.

Teknik steganografi meliputi banyak sekali metode komunikasi untuk menyembunyikan pesan rahasia (teks atau gambar) di dalam *file-file* lain yang mengandung teks, *image*, bahkan *audio* tanpa menunjukkan ciri-ciri perubahan yang nyata atau terlihat dalam kualitas dan struktur dari *file* semula (Aditya et al. , 2010). Metode ini termasuk tinta yang tidak tampak, *microdots*, pengaturan kata, tanda tangan digital, jalur tersembunyi dan komunikasi spektrum lebar.

Tujuan dari steganografi adalah merahasiakan atau menyembunyikan keberadaan dari sebuah pesan tersembunyi atau sebuah informasi. Dalam prakteknya kebanyakan diselesaikan dengan membuat perubahan tipis terhadap data digital lain yang isinya tidak akan menarik perhatian dari penyerang potensial, sebagai contoh sebuah gambar yang terlihat tidak berbahaya. Perubahan ini bergantung pada kunci (sama pada kriptografi) dan pesan untuk disembunyikan. Orang yang menerima gambar kemudian dapat menyimpulkan informasi terselubung dengan cara mengganti kunci yang benar ke dalam algoritma yang digunakan.

Pada metode steganografi cara ini sangat berguna jika digunakan pada cara steganografi komputer karena banyak format *file* digital yang dapat dijadikan media untuk menyembunyikan pesan. Format yang biasa digunakan diantaranya:

Format *image* : *bitmap (bmp), gif, pcx, jpeg,* dll.

Format *audio* : *wav, voc, mp3,* dll.

Format lain : *teks file, html, pdf,* dll.

Kelebihan steganografi daripada kriptografi adalah pesan-pesannya tidak menarik perhatian orang lain. Pesan-pesan berkode dalam kriptografi yang tidak disembunyikan, walaupun tidak dapat dipecahkan, akan menimbulkan kecurigaan.

Seringkali, steganografi dan kriptografi digunakan secara bersamaan untuk menjamin keamanan pesan rahasianya.

Sebuah pesan steganografi (*plaintext*), biasanya pertama-tama dienkripsikan dengan beberapa arti tradisional, yang menghasilkan *ciphertext*. Kemudian, *covertext* dimodifikasi dalam beberapa cara sehingga berisi *ciphertext*, yang menghasilkan *stegotext*. Contohnya, ukuran huruf, ukuran spasi, jenis huruf, atau karakteristik *covertext* lainnya dapat dimanipulasi untuk membawa pesan tersembunyi; hanya penerima (yang harus mengetahui teknik yang digunakan) dapat membuka pesan dan mendekripsikannya.

2.3.2 Metode steganografi

Terdapat banyak metode yang digunakan dalam melakukan penyembunyian data kedalam data lainnya. Berikut adalah penjelasan mengenai beberapa metode yang banyak digunakan dalam steganografi.

Tabel 3 : Klasifikasi Metode Steganografi (Aditya et al. , 2010; Nani, 2011; Wandani et al. , 2012)

Media	Metode
Teks	Metode Spasi Terbuka
	Metode *Syntactic*
	Metode Semantic
Gambar	LSB (*Least Significant Bit*)
	Algorithms and Transformation
	Redundant Pattern Encoding
	Spread Spectrum Method
Suara	*Low Bit Encoding*
	Phase Encoding
	Spread Spectrum
	Echo Hiding
EOF (*End Of File*)	*Redundant bits*

2.3.3 Metode *End of File* (EOF)

Pada pembahasan ini, penulis hanya memberikan penjelasan tentang algoritma *End Of File* (EOF) pada steganografi. Metode *End Of File* (EOF)

merupakan salah satu metode yang digunakan dalam steganografi. Metode ini meyembunyikan pesan rahasia dengan cara manambahkan bit-bit pesan yang akan disembunyikan ke akhir file citra penampung.

Teknik ini menggunakan cara dengan menyisipkan data pada akhir file. Sehingga, tidak akan mengganggu kualitas data awal yang akan disisipkan pesan. Namun, ukuran file setelah disisipkan pesan rahasia akan bertambah (Sukrisno & Utami, 2007; Wandani et al. , 2012). Sebab, ukuran file yang telah disisipkan pesan rahasia sama dengan ukuran file sebelum disisipkan pesan rahasia ditambah dengan ukuran pesan rahasia yang disisipkan. Untuk mengenal data yang disisipkan pada akhir file, diperlukan suatu tanda pengenal atau simbol pada awal dan akhir data yang akan disisipkan.

Proses penyisipan pesan dengan metode End Of File dapat dituliskan dalam algoritma sebagai berikut (Nani, 2011) :

1. Inputkan pesan yang akan disisipkan
2. Ubah pesan menjadi kode-kode desimal.
3. Inputkan citra grayscale yang akan disisipi pesan.
4. Dapatkan nilai derajat keabuan masing- masing piksel.
5. Tambahkan kode desimal pesan sebagai nilai derajat keabuan diakhir citra.
6. Petakan menjadi citra baru.

2.3.4 Penelitian Terkait pada Algoritma *End Of File* (EOF)

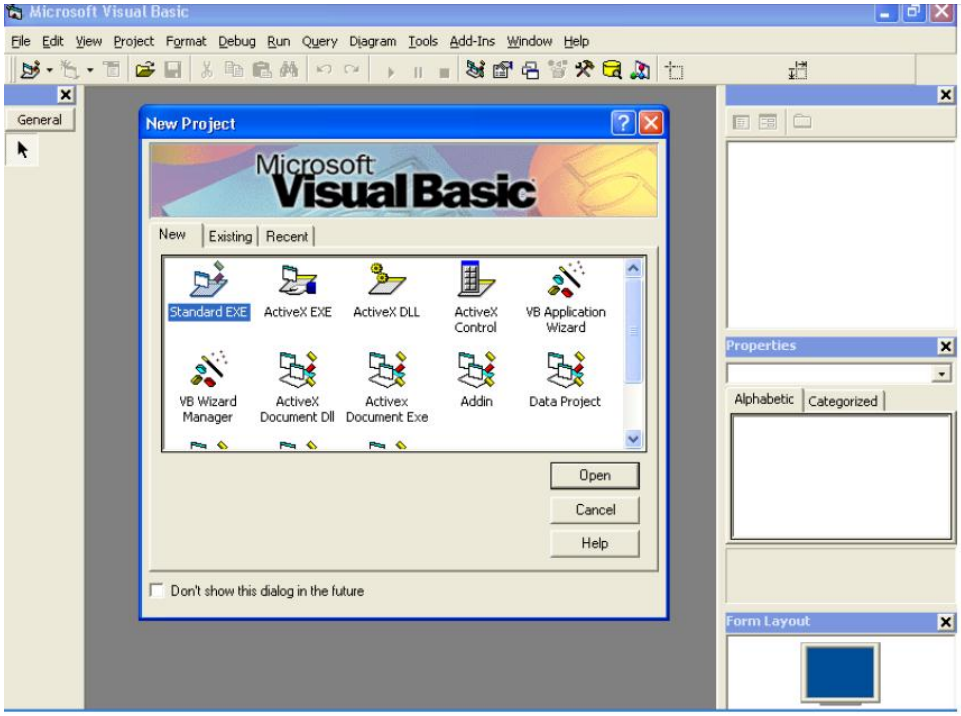
Penelitian lain yang dilakukan oleh Paskalis Andrianus Nani (Nani, 2011) yaitu "Penerapan algoritma *Blowfish* pada Proses Steganografi EOF", juga menggunakan teknik Kriptografi dengan salah satu algoritma yaitu *Blowfish*. Algoritma *Blowfish* dipilih karena dinilai sebagai algoritma yang cepat, sederhana dan mempunyai keamanan yang variabel, sedangkan EOF digunakan untuk menyisipkan data. Pada penelitian ini, data yang digunakan sebagai induk yaitu citra berformat . *bmp grayscale* dengan resolusi 1000 x 936 piksel. dengan file pesanannya yaitu 3 buah file berformat . *txt*. *File* induk dalam penelitian ini *unwira. bmp* disisipi *file* . *txt* satu per satu, dimana sisipan tersebut dikendalikan oleh metode EOF. Masing-masing menghasilkan ukuran file yang tidak terlalu berbeda dengan ukuran file aslinya. Dalam penelitian ini dijelaskan bahwa

penentuan kunci membutuhkan banyak iterasi sehingga membutuhkan lebih banyak waktu, kemudian kunci yang digunakan harus dihitung sebelum proses enkripsi/dekripsi dan kunci yang digunakan harus sub kunci yang besar, sehingga menurut penulis, algoritma *blowfish* kurang efektif untuk diterapkan. Sedangkan metode *End Of File* (EOF) dinilai efektif untuk mengamankan data yang disisipkan.

2.4 Visual Basic

Microsoft Visual Basic adalah merupakan sebuah bahasa *pemrograman* komputer yang menjadi sarana (*tools*) untuk menghasilkan *program – program* aplikasi yang berbasis windows, seperti pascal dan C. Visual Basic 6. 0 hampir sama dengan bahasa *pemrograman* basic (*quick basic*), dikarenakan visual basic 6. 0 adalah pengembangan dari basic. *Program* aplikasi dapat berupa *program database*, *program grafis* dan lain sebagainya. Didalam Visual Basic 6. 0 terdapat komponen - komponen yang sangat membantu dalam pembuatan *program* aplikasi. Dalam pembuatan *program* aplikasi pada Visual Basic 6. 0 dapat didukung oleh *software* seperti Microsoft Access, Microsoft Exel, Seagate Crystal Report, dan lain sebagainya.

Untuk dapat menyusun dan membuat suatu *program* aplikasi dari VB 6. 0, tentunya harus mengetahui fasilitas – fasilitas yang disediakan agar proses penyusunan dan pembuatan *program* tersebut berjalan dengan baik. Visual Basic 6. 0 terdapat dalam 3 versi perangkat lunak yang diproduksi oleh Microsoft yaitu Microsoft Visual Basic 6. 0 Professional, Microsoft Visual Basic 6. 0 Enterprise Edition, dan Microsoft Visual Studio 6. 0. Di dalam perancangan kali ini yang digunakan adalah Microsoft Visual Basic 6. 0 Enterprise Edition.



Gambar 4 : Tampilan Awal Microsoft Visual Basic

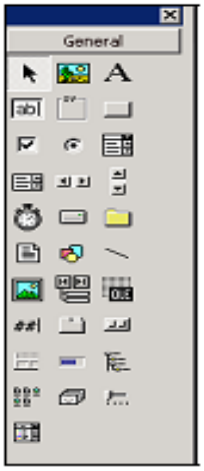
Beberapa keunggulan yang dapat menjadikan visual basic 6. 0 lebih istimewa dibanding bahasa *pemrograman* lain adalah sebagai berikut:

1. Menggunakan platform pembuatan *program* yang diberi nama *developer studio* yang memilik tampilan dan sarana yang sama dengan visual c++ dan visual j++.
2. Memiliki *compiler* yang handal sehingga lebih efisien.
3. Memiliki beberapa tambahan *wizard* baru.
4. Kemampuan membuat A*ctiveX* dan fasilitas internet yang lebih banyak.
5. Memiliki *autolist*.
6. Dapat digunakan sebagai sarana membuat aplikasi *database*, biasanya untuk aplikasi *client/server*.

Adapun beberapa fitur yang dimiliki oleh visual basic 6. 0, antara lain:

a. Toolbox

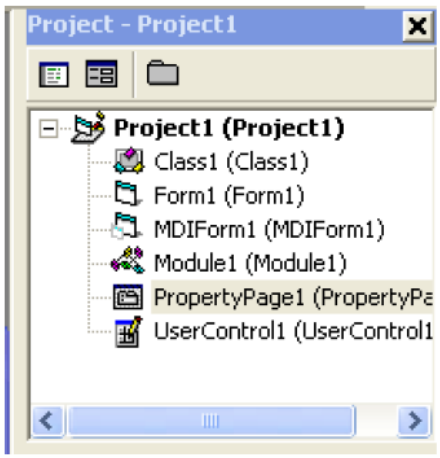
Toolbox merupakan tempat *icon – icon* untuk objek yang akan dimasukan dalam *form* pada pembuatan *program* aplikasi. Secara *default* pada *toolbox* hanya terdapat objek - objek seperti Gambar 5.



Gambar 5 : Tampilan *Toolbox*

b. Project Explorer

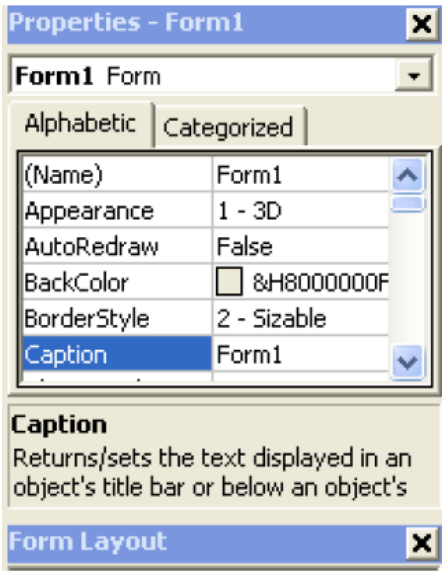
Project Explorer merupakan tempat untuk melihat daftar *form, modules*, dan *design* dengan mengklik kanan pada bagian *project explorer* dan pilih *add*, lalu pilih yang akan ditambah. Tampilan *project explorer* seperti pada gambar 6.



Gambar 6 : *Project Explorer*

c. Properties Window

Properties Windows merupakan tempat yang digunakan untuk mengatur *properti* dari setiap objek kontrol. Pada *properti windows* ini semua objek kontrol dapat diatur sesuai dengan *program* aplikasi yang akan dibuat. Setelah kontrol yang terdapat pada *toolbox* diletakkan pada *form*, kontrol tersebut dapat diberikan nilai sesuai dengan fungsi yang akan di jalankan, nilai tersebut diletakkan di *windows properties*. Tampilan *properties* tampak seperti gambar 7.



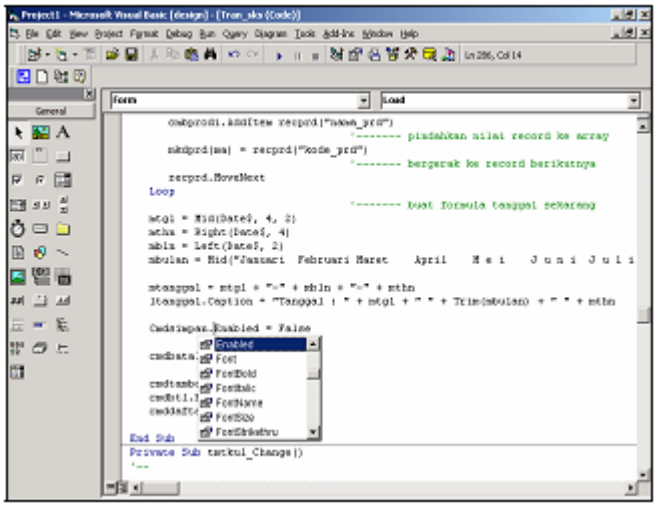
Gambar 7 : *Tool Combobox*

d. Button

Tool button digunakan sebagai *trigger* untuk melaksanakan suatu aksi tertentu, biasanya berbentuk perintah. Dalam perancangan ini, *tool button* digunakan sebagai *trigger* dalam melakukan proses enkripsi atau enkripsi pesan.

Gambar 8 : *Tool Command*

e. Code Editor



Gambar 9 : Tampilan *Window Code Editor*

Window yang dipergunakan untuk menuliskan *program*, setiap kontrol dalam *form* dapat memiliki fungsi tertentu. Fungsi tersebut diwujudkan dalam deret perintah, perintah perintah ini yang di tuliskan kedalam *window code editor*. VB menyediakan *autolist* untuk memberikan nilai yang terdapat pada suatu kontol, sehingga mengurangi kesalahan penulisan program. Cara menggunakannya adalah dengan mengklik ganda kontrol yang ada.

2.5 *Unified Modelling Language* (UML)

Unified Modelling Language adalah bahasa standar yang digunakan untuk menjelaskan dan memvisualisasikan artifak dari proses analisis dan dasain berorientasi objek. UML menyediakan standar pada notasi dan diagram yang bisa digunakan untuk memodelkan suatu sistem. UML dikembangkan oleh 3 orang pendekar 'berorientasi objek', yaitu Grady Booch, Jim Rumbaugh dan Ivar Jacobson.

Unified Modelling Language merupakan suatu kumpulan teknik terbaik yang telah terbukti sukses dalam memodelkan sistem yang besar dan kompleks. UML tidak hanya digunakan dalam proses pemodelan perangkat lunak, namun hampir dalam semua bidang yang membutuhkan pemodelan.

2.5.1 Bagian UML

Bagian-bagian utama dari UML adalah *view, diagram, model element*, dan *general mechanism*.

a. View

View digunakan untuk melihat sistem yang dimodelkan dari beberapa aspek yang berbeda. *View* bukan melihat grafik, tapi merupakan suatu abstraksi yang berisi sejumlah diagram. Beberapa jenis *view* dalam UML antara lain: *use case view, logical view, component view, concurrency view*, dan *deployment view*.

b. *Use case view*

Mendeskripsikan fungsionalitas sistem yang seharusnya dilakukan sesuai yang diinginkan *external actors*. *Actor* yang berinteraksi dengan sistem dapat berupa *user* atau sistem lainnya. *View* ini digambarkan dalam *use case diagrams* dan kadang-kadang dengan *activity diagrams*. *View* ini digunakan terutama untuk pelanggan, perancang (*designer*), pengembang (*developer*), dan penguji sistem (*tester*).

c. *Logical view*

Mendeskripsikan bagaimana fungsionalitas dari sistem, struktur statis (*class, object,danrelationship*) dan kolaborasi dinamis yang terjadi ketika objek mengirim pesan ke objek lain dalam suatu fungsi tertentu. *View* ini digambarkan dalam *class diagrams* untuk struktur statis dan dalam *state, sequence, collaboration,* dan *activity diagram* untuk model diamisnya. *View* ini digunakan untuk perancang (*designer*) dan pengembang (*developer*).

d. *Component view*

Mendeskripsikan implementasi dan ketergantungan modul. Komponen yang merupakan tipe lainnya dari *code module* diperlihatkan dengan struktur dan ketergantungannya juga alokasi sumber daya komponen dan informasi *administrative* lainnya. *View* ini digambarkan dalam *component view* dan digunakan untuk pengembang (*developer*).

e. *Concurrency view*

embagi sistem ke dalam proses dan prosesor. *View* ini digambarkan dalam diagram dinamis (*state, sequence, collaboration,* dan *activity diagrams*) dan diagram implementasi (*component* dan *deployment diagrams*) serta digunakan

untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

f. *Deployment view*

Mendeskripsikan fisik dari sistem seperti komputer dan perangkat (*nodes*) dan bagaimana hubungannya dengan lainnya. *View* ini digambarkan dalam *deployment diagrams* dan digunakan untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

g. Diagram

Diagram berbentuk grafik yang menunjukkan simbol elemen model yang disusun untuk mengilustrasikan bagian atau aspek tertentu dari sistem. Sebuah diagram merupakan bagian dari suatu *view* tertentu dan ketika digambarkan biasanya dialokasikan untuk *view* tertentu.

2.5.2 Diagram dalam UML

1. *Use Case Diagram*

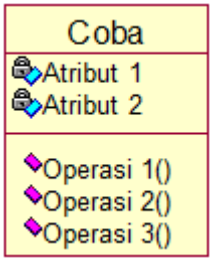
Use case adalah abstraksi dari interaksi antara sistem dan *actor*. *Use case* bekerja dengan cara mendeskripsikan tipe interaksi antara *user* sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai.

Gambar 10 : Notasi *Use Case*

Use case merupakan konstruksi untuk mendeskripsikan bagaimana sistem akan terlihat di mata *user*. Sedangkan *use case* diagram memfasilitasi komunikasi diantara analis dan pengguna serta antara analis dan *client*.

2. *Class Diagram*

Class adalah dekripsi kelompok obyek-obyek dengan properti, perilaku (operasi) dan relasi yang sama. Sehingga dengan adanya *class diagram* dapat memberikan pandangan global atas sebuah sistem. Hal tersebut tercermin dari *class- class* yang ada dan relasinya satu dengan yang lainnya.



Gambar 11 : Notasi *Class*

Sebuah sistem biasanya mempunyai beberapa *class diagram*. *Class diagram* sangat membantu dalam visualisasi struktur kelas dari suatu sistem.

3. *Component Diagram*

   Component software merupakan bagian fisik dari sebuah sistem, karena menetap di komputer tidak berada di benak para analis. *Component* merupakan implementasi *software* dari sebuah atau lebih class. *Component* dapat berupa *source code*, komponent biner, atau *executable component*. Sebuah komponent berisi informasi tentang *logic class* atau *class* yang diimplementasikan sehingga membuat pemetaan dari *logical view* ke *component view*. Sehingga *component diagram* merepresentasikan dunia riil yaitu *component software* yang mengandung *component, interface* dan *relationship*.

4. *Deployment Diagram*

   Menggambarkan tata letak sebuah sistem secara fisik, menampakkan bagian-bagian *software* yang berjalan pada bagian-bagian *hardware*, menunjukkan hubungan komputer dengan perangkat (*nodes*) satu sama lain dan jenis hubungannya. Di dalam *nodes, executeable component* dan objek yang dialokasikan untuk memperlihatkan unit perangkat lunak yang dieksekusi oleh *node* tertentu dan ketergantungan komponen.

5. *State Diagram*

   Menggambarkan semua *state* (kondisi) yang dimiliki oleh suatu objek dari suatu *class* dan keadaan yang menyebabkan *state* berubah. Kejadian dapat berupa objek lain yang mengirim pesan. *State class* tidak digambarkan untuk semua *class*, hanya yang mempunyai sejumlah *state* yang terdefinisi dengan baik dan kondisi *class* berubah oleh *state* yang berbeda.

6. *Sequence Diagram*

   Sequence Diagram digunakan untuk menggambarkan perilaku pada sebuah skenario. Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara objek juga interaksi antara objek, sesuatu yang terjadi pada titik tertentu dalam eksekusi sistem.

7. *Collaboration Diagram*

   Menggambarkan kolaborasi dinamis seperti *sequence diagrams*. Dalam menunjukkan pertukaran pesan, *collaboration diagram* menggambarkan objek dan hubungannya (mengacu ke konteks). Jika penekannya pada waktu atau urutan gunakan *sequence diagrams*, tapi jika penekanannya pada konteks gunakan *collaboration diagram*.

8. *Activity Diagram*

   Menggambarkan rangkaian aliran dari aktivitas, digunakan untuk mendeskripsikan aktifitas yang dibentuk dalam suatu operasi sehingga dapat juga digunakan untuk aktifitas lainnya seperti *use case* atau interaksi.

2.5.3 Tujuan Penggunaan UML

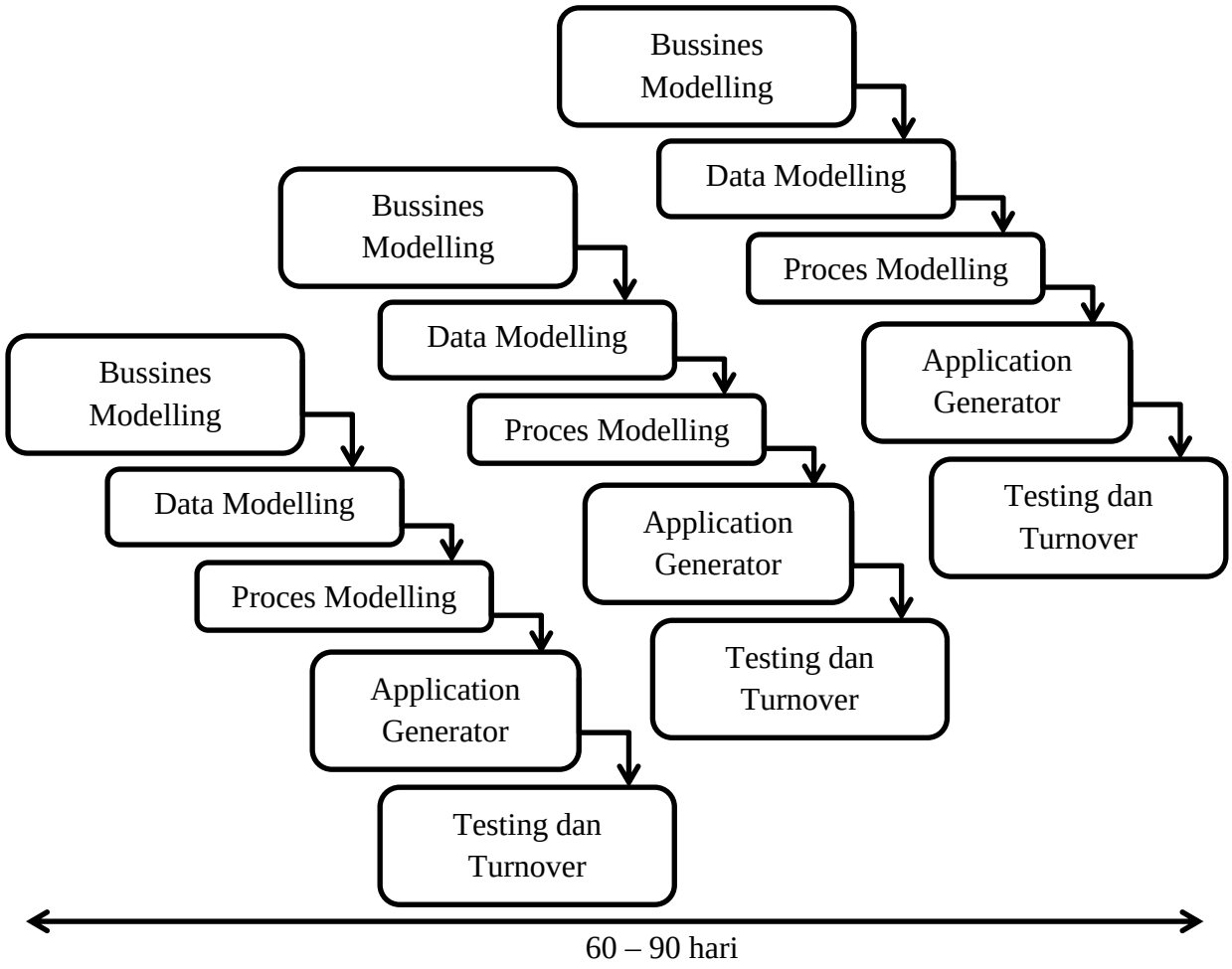
- Memberikan bahasa pemodelan yang bebas dari berbagai bahas pemrograman dan proses rekayasa.
- Menyatukan praktek-praktek terbaik yang terdapat dalam pemodelan.
- Memberikan model yang siap pakai, bahasa pemodelan visual yang ekspresif untuk mengembangkan dan saling menukar model dengan mudah dan dimengerti secara umum.
- UML bisa juga berfungsi sebagai sebuah (*blue print*) cetak biru karena sangat lengkap dan detail. Dengan cetak biru ini maka akan bisa diketahui informasi secara detail tentang *coding program* atau bahkan membaca *program* dan menginterpretasikan kembali ke dalam bentuk diagram (*reserve enginering*).

2.6 Pengembangan Sistem

Agar mempermudah dalam pengembangan sistem, maka penulis membangun sebuah sistem yang akan membantu dalam menggambarkan proses penyelesaian masalah. Metode yang sesuai dalam pengembangan sistem ini adalah metode *Rapid Aplication Development* (RAD).

RAD adalah sebuah model proses perkembangan *software* sekuensial linier yang menekankan siklus perkembangan yang sangat pendek. Model ini merupakan sebuah adaptasi "kecepatan tinggi" dari model sekuensial linier di mana perkembangan cepat dicapai dengan menggunakan pendekatan kontruksi berbasis komponen.

Selain itu, RAD mengadopsi model waterfall dan pembangunan dalam waktu singkat. Jika keutuhan yang diinginkan pada tahap analisa kebutuhan telah lengkap dan jelas, maka waktu yang dibutuhkan untuk menyelesaikan secara lengkap perangkat lunak yang dibuat adalah berkisar 60 sampai 90 hari (Setiawan, Endrawan, Fathoni, & P, 2011).



Gambar 12 : Alur Model RAD

3.6.1 Bussines Modelling

Aliran informasi di antara fungsi – fungsi bisnis dimodelkan dengan menjawab pertanyaan – pertanyaan berikut :

a. Informasi apa yang mengendalikan proses bisnis?
b. Informasi apa yang di munculkan?
c. Siapa yang memunculkanya?
d. Ke mana informasi itu pergi?
e. Siapa yang memprosesnya?

3.6.2 Data Modelling

a. Aliran informasi yang didefinisikan sebagai bagian dari fase *business modeling* disaring ke dalam serangkaian objek data yang dibutuhkan untuk menopang bisnis tersebut.

b. Karakteristik (disebut atribut) masing – masing objek diidentifikasi dan hubungan antara objek – objek tersebut didefinisikan.
c. Bagian dari pemodelan bisnis yang didefinisikan ke dalam sekumpulan objek data.
d. Karakteristik (atribut) dari setiap objek diidentifikasikan dan hubungannya.

3.6.3 Proces Modelling

a. Aliran informasi yang didefinisikan di dalam fase *data modeling* ditransformasikan untuk mencapai aliran informasi yang perlu bagi implementasi sebuah fungsi bisnis.
b. Gambaran pemrosesan diciptakan untuk menambah, memodifikasi, menghapus, atau mendapatkan kembali sebuah objek data.
c. Objek data akan diimplementasikan pada fungsi bisnis.
d. Deskripsi proses dibangun untuk penambahan modifikasi, penghapusan, atau pengambilan kembali objek data.

3.6.4 Application Generation

a. RAD mengasumsikan pemakaian teknik generasi ke empat.
   Selain menciptakan perangkat lunak dengan menggunakan bahasa pemrograman generasi ketiga yang konvensional
b. Pada semua kasus, alat – alat bantu otomatis dipakai untuk memfasilitasi konstruksi perangkat lunak.
c. Melakukan penggunaan kembali komponen yang ada (jika mungkin).
d. Atau membuat kembali penggunaan kembali komponen jika dibutuhkan.

3.6.5 Testing dan Turnover

a. Karena proses RAD menekankan pada pemakaian kembali, banyak komponen program telah diuji. Hal ini mengurangi keseluruhan waktu pengujian.
b. Tetapi komponen baru harus di uji dan semua interface harus dilatih secara penuh.

Ada beberapa alasan di dalam memilih metode RAD ini antara lain adalah sebagai berikut:

a. Alasan yang Buruk
   - Apabila menggunakan RAD, hanya untuk menghemat biaya pengembangan suatu sistem. Hal ini disebabkan karena dengan menggunakan metode RAD membutuhkan suatu tim yang mengerti betul mengenai manajemen biaya. Apabila tidak, maka biaya yang dikeluarkan akan semakin besar.
   - Apabila menggunakan RAD, hanya untuk menghemat waktu pengembangan suatu sistem. Hal ini disebabkan karena dengan menggunakan metode RAD membutuhkan suatu tim yang mengerti betul mengenai manajemen waktu. Apabila tidak, maka waktu yang dibutuhkan akan semakin lama.

b. Alasan yang Baik
   - Apabila menggunakan RAD untuk mendapatkan suatu desain yang dapat diterima oleh konsumen dan dapat dikembangkan dengan mudah.
   - Apabila menggunakan RAD untuk memberikan batasan-batasan pada suatu sistem supaya tidak mengalami perubahan.
   - Apabila menggunakan RAD untuk menghemat waktu dan jika mungkin dapat menghemat biaya serta menghasilkan produk berkualitas.

Dengan menggunakan RAD maka ada beberapa tujuan yang tidak mungkin akan tercapai secara bersama yaitu :

- Kemungkinan terjadi kesalahan sangat kecil karena pihak pengembang tidak mempunyai hak untuk mengubah komponen-komponen yang digunakan dalam mengembangkan suatu sistem.
- Tingkat kepuasan konsumen yang tinggi karena kebutuhan-kebutuhan sekunder dari konsumuen harus dikorbankan supaya suatu sistem dapat diselesaikan sesuai jadwal.
- Biaya pengembangan yang termurah karena dengan menggunakan komponen yang sudah ada dapat menyebabkan biaya yang lebih besar apabila dibandingkan dengan mengembangkan komponen sendiri.

2.6.6 Kelebihan Pengembangan Sistem

- Hasil Pengembangan bisa lebih cepat dibandingkan SDLC lainnya.
- Memerlukan biaya yang lebih sedikit.
- Mementingkan dari segi bisnis dan teknik.
- Berkonsentrasi pada sudut pandang *user*.
- Menyediakan kemungkinan perubahan secara cepat sesuai permintaan *user*.
- Menghasilkan jarak kesesuaian yang kecil antara kebutuhan *user* dan spesifikasi sistem.
- Waktu, biaya dan *effort* (usaha) minimal.