**BAB II**

**LANDASAN TEORI**

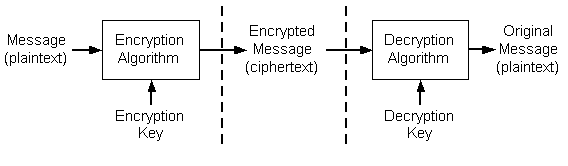
1. **Kriptografi**

Kriptografi berasal dari Bahasa Yunani dan terdiri dari dua kata, yaitu *kryptos* yang berarti tersembunyi dan *graphein* yang berarti menulis. Kriptografi adalah seni dan ilmu melindungi keamanan pesan (Bruce Schneier, 1996). Pesan dilindungi dengan cara merubah pesan (informasi) asli (yang disebut *plaintext*) menjadi teks sandi (*ciphertext*) menggunakan algoritma tertentu. Proses perubahan *plaintext* menjadi *ciphertext* ini disebut dengan enkripsi.

Terdapat istilah-istilah penting yang harus dipahami dalam kriptografi, antara lain:

1. *Plaintext* (P) adalah pesan (informasi) asli yang dapat dimengerti maknanya.
2. *Ciphertext* (C) adalah pesan tersandi yang merupakan hasil enkripsi.
3. Enkripsi (E) adalah proses perubahan *plaintext* menjadi *ciphertext*.
4. Dekripsi (D) adalah kebalikan dari enkripsi yakni proses perubahan *ciphertext* menjadi *plaintext*.
5. Kunci (K) adalah parameter yang mengatur proses enkripsi dan dekripsi. Kunci biasanya berupa *string* atau deretan bilangan.

Secara umum, ada dua proses yang terdapat dalam kriptografi, yaitu enkripsi dan dekripsi. Enkripsi yaitu proses pengubahan pesan (*plaintext*) menjadi teks sandi (*ciphertext*) dengan tujuan menyamarkan maknanya. Sedangkan dekripsi yaitu proses pengubahan kembali *ciphertext* menjadi *plaintext*. Selain itu, dalam kriptografi juga terdapat kunci (*key*) yang bersama-sama dengan algoritma kriptografinya akan melakukan proses enkripsi dan dekripsi (Bruce Schneier, 1996). Secara umum, proses kriptografi dapat dilihat pada gambar 2.1.



**Gambar 2.1 Proses enkripsi dan dekripsi**

* + 1. **Aspek-Aspek Keamanan Kriptografi**

Dalam perkembangannya, teknik kriptografi tidak hanya digunakan semata-mata untuk menjaga kerahasiaan saja. Aspek yang kemudian muncul seiring dengan semakin kompleksnya masalah yang dihadapi ketika melakukan pengiriman pesan adalah:

1. Autentikasi (*authentication*)

Aspek untuk menjamin keaslian suatu pesan, sehingga pihak yang menerima pesan dapat memastikan keaslian pesan tersebut datang dari orang yang dimintai informasi. Dengan kata lain informasi tersebut benar-benar datang dari orang yang dikehendaki.

1. Integritas (*integrity*)

Aspek untuk menjaga agar sebuah pesan tidak diubah-ubah sewaktu dikirim atau disimpan. Perubahan pesan saat dilakukan pengiriman tentu membawa dampak yang tidak kecil, terutama ketika pesan tersebut nantinya digunakan sebagai bahan pengambil keputusan. Oleh karena itu, kriptografi juga harus membuat pesan asli tidak dapat diubah saat dikirim atau disimpan.

1. Penghindaran penolakan (*non-repudiation*)

Aspek untuk menjaga bukti-bukti bahwa suatu pesan berasal dari seseorang, sehingga pihak yang mengirim tidak bisa menyangkal bahwa pesan tersebut berasal dari pihak tersebut.

1. Kerahasiaan (*confidentiality*)

Kerahasiaan adalah aspek penyembunyian pesan dari orang-orang yang tidak punya otoritas sehingga pesan penting hanya akan dibaca oleh orang yang dituju.

1. **Steganografi**

Steganografi berasal dari Bahasa Yunani dan terdiri dari dua kata, yaitu *steganos* yang berarti tertutup dan *graphein* yang berarti menulis. Steganografi berarti suatu cara untuk menyembunyikan keberadaan pesan (Simon Singh, 2001). Perbedaan kriptografi dengan steganografi yaitu kriptografi menyembunyikan makna pesan, sedangkan steganografi menyembunyikan keberadaan pesan. Pada kriptografi, makna pesan disembunyikan dengan cara mengacak susunan karakter-karakter pembentuk pesan dengan algoritma tertentu sehingga pesan tampak tidak bisa dibaca, tetapi pesan itu sendiri masih tampak secara visual. Sedangkan pada steganografi, keberadaan pesan tersebut disembunyikan di balik suatu penutup (*cover*) sehingga pesan tersebut menjadi tidak tampak secara visual. Pada steganografi modern, steganografi biasa dilakukan dengan cara menyisipkan suatu pesan pada sebuah *file* (*cover object*). Format *file* yang biasa digunakan antara lain adalah:

1. Format teks: .txt, .html, .pdf
2. Format citra: .bmp, .jpg, .png
3. Format audio: .mp3, .wav, .m4a
4. Format video: .mp4, .avi, .mkv

Kualitas citra yang telah disisipi pesan diukur berdasarkan nilai MSE (*Mean Square Error*) dan PSNR (*Peak Signal to Noise Ratio*). MSE adalah nilai *error* kuadrat rata-rata antara citra asli dan citra hasil manipulasi (dalam steganografi, MSE adalah nilai *error* kuadrat rata-rata antara citra asli dan citra hasil penyisipan). MSE dihitung menggunakan persamaan (2.1) sebagai berikut.

C:\Users\Lenovo\Downloads\2.PNG

(2.1)

dimana x dan y adalah kordinat dari gambar, M dan N adalah dimensi dari gambar, Sxy menyatakan citra hasil penyisipan, dan Cxy menyatakan citra asli. Sedangkan PSNR adalah perbandingan antara nilai maksimum dari sinyal yang diukur dengan besarnya derau yang berpengaruh pada sinyal tersebut. Nilai PSNR yang tinggi menyiratkan kemiripan yang lebih erat antara citra hasil penyisipan dengan citra asli. PSNR dihitung menggunakan persamaan (2.2) berikut.

C:\Users\Lenovo\Downloads\1.PNG

(2.2)

* + 1. **Kriteria Steganografi yang Baik**

Menurut Munir (2006), kriteria-kriteria yang harus diperhatikan dalam steganografi adalah sebagai berikut.

1. *Imperceptibility*

Keberadaan pesan rahasia tidak dapat dipersepsi oleh inderawi. Misalnya, jika *cover object* berupa citra, maka penyisipan pesan membuat citra stego sulit dibedakan oleh mata dengan citra aslinya.

1. *Fidelity*

Mutu *cover object* tidak berubah banyak akibat penyisipan pesan. Perubahan tersebut tidak dapat dipersepsi oleh inderawi.

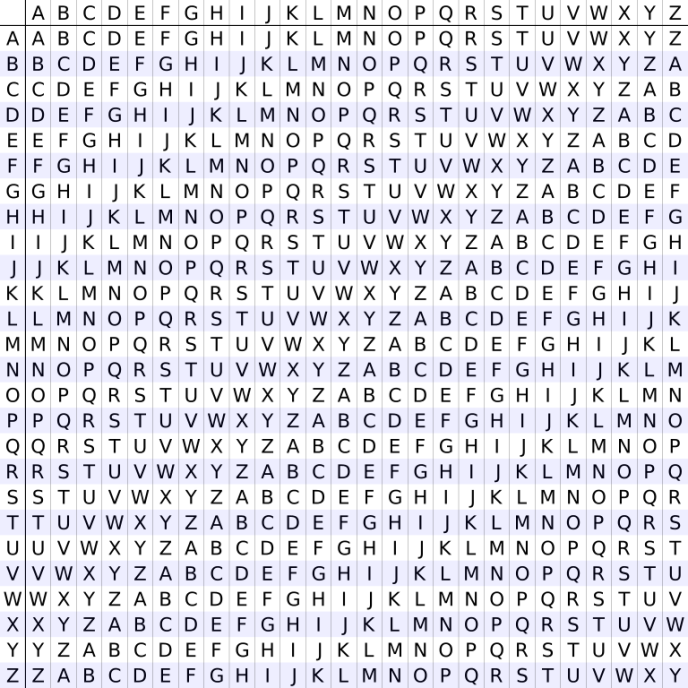
1. *Recovery*

Pesan yang disembunyikan harus dapat diungkapkan kembali. Karena tujuan steganografi adalah *data hiding*, maka sewaktu-waktu pesan rahasia di dalam *cover object* harus dapat diambil kembali untuk digunakan lebih lanjut.

1. **Metode Vigenere *Cipher* (Sandi Vigenere)**

Vigenere *cipher* merupakan sebuah algoritma kriptografi subtitusi klasik yang ditemukan oleh Giovan Battista Bellaso pada tahun 1553, namun disempurnakan oleh Blaise de Vigenere pada abad ke-19. Vigenere *cipher* ini mendapat sebutan *le chiffre indechiffrable* (Bahasa Perancis yang berarti sandi yang tak terpecahkan) karena selama tiga abad belum ada metode yang mampu memecahkan sandi ini.

Vigenere *cipher* menggunakan kunci yang panjangnya ditentukan oleh pengguna. Apabila panjang kunci tidak sama dengan *plaintext*, maka kunci akan diulang hingga *plaintext* dan kunci memiliki panjang yang sama. Proses enkripsi dapat dilakukan dengan menggunakan sebuah tabel alfabet yang disebut dengan *tabula recta* atau tabel Vigenere.



**Gambar 2.2 Tabel Vigenere**

Sebagai contoh, dengan *plaintext* FINALATTACK dan kunci RIDE, proses enkripsinya adalah sebagai berikut.

*Plaintext* : FINALATTACK

Kunci : RIDERIDERID

*Ciphertext* : WQQECIWXRKN

Pada contoh di atas, masing-masing karakter pada *plaintext* dipasangkan pada karakter-karakter yang terdapat pada kunci. Kemudian, dengan menggunakan tabel Vigenere, karakter F yang merupakan karakter pertama pada *plaintext* dienkripsi menggunakan kunci R yang merupakan karakter pertama pada kunci dengan cara menemukan elemen tabel pada baris F (dari *plaintext*) dan kolom R (dari kunci), yaitu W. Proses ini dilanjutkan pada karakter kedua dari *plaintext* dan kunci, masing-masing I dan I yang akan menghasilkan huruf Q, dan seterusnya.

Dekripsi dilakukan dengan cara menelusuri baris tabel dimulai dari karakter pertama pada kunci, kemudian temukan kolom tempat karakter *ciphertext* pertama berada. Label kolom tersebut akan menjadi karakter pertama *plaintext*. Sebagai contoh, untuk melakukan dekripsi karakter W yang merupakan karakter pertama pada *ciphertext*, temukan baris R yang merupakan karakter pertama pada kunci, kemudian temukan karakter W pada baris tersebut. Karakter W berada pada kolom F, maka karakter *plaintext* yang dihasilkan dari proses dekripsi karakter W dengan kunci R adalah F.

Selain menggunakan tabel seperti contoh di atas, proses enkripsi dan dekripsi juga dapat dilakukan dengan mengikuti persamaan (2.3) dan (2.4).

Persamaan enkripsi Vigenere *cipher*:

Persamaan dekripsi Vigenere *cipher*:

dimana:

Ci = nilai desimal karakter *ciphertext* ke-i

Pi = nilai desimal karakter *plaintext* ke-i

Ki = nilai desimal karakter kunci ke-i

Dengan menggunakan contoh di atas, proses enkripsi dapat dijabarkan sebagai berikut. Setiap karakter A-Z memiliki nilai seperti dapat dilihat pada gambar 2.3 (Patel, 2008).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |

**Gambar 2.3 Nilai masing-masing karakter pada Vigenere *cipher***

Dengan menggunakan persamaan 2.3, proses enkripsi dapat dijabarkan sebagai berikut.

Enkripsi *plaintext* F dengan kunci R

C1 = (P1 + K1)

= (5 + 17)

= 22

Dari operasi yang dilakukan di atas, didapatkan C1 yaitu 22. Karakter dengan nilai 22 yaitu W, sehingga dapat disimpulkan enkripsi *plaintext* F dengan kunci R menghasilkan *ciphertext* W.

Enkripsi *plaintext* I dengan kunci I

C2 = (P2 + K2)

= (8 + 8)

= 16

Dari operasi yang dilakukan di atas, didapatkan C2 yaitu 16. Karakter dengan nilai 16 yaitu Q, sehingga dapat disimpulkan enkripsi *plaintext* I dengan kunci I menghasilkan *ciphertext* Q.

Hal yang perlu diperhatikan adalah apabila hasil penjumlahan Pi dan Ki melebihi 25 atau hasil pengurangan Ci dan Ki kurang dari 0, maka perlu dilakukan operasi penjumlahan atau pengurangan dengan jumlah seluruh karakter untuk menghasilkan *ciphertext* atau *plaintext* sehingga persamaan (2.3) dan (2.4) akan berubah menjadi persamaan (2.5) dan (2.6).

Persamaan enkripsi jika Pi + Ki > 25:

Persamaan dekripsi jika Pi - Ki < 0:

Sebagai contoh akan dilakukan proses enkripsi *plaintext* T dengan kunci U.

Ci = (Pi + Ki)

= (19 + 20)

= 39 - 26

= 13

Dapat dilihat pada contoh di atas, hasil penjumlahan Pi dan Ki yaitu 39, melebihi nilai yang terdapat di dalam tabel Vigenere yaitu 25. Oleh karena itu, perlu dilakukan operasi pengurangan untuk mendapatkan nilai yang berada di dalam *range* tabel Vigenere. Pada contoh di atas, 39 - 26 menghasilkan 13, sehingga proses enkripsi *plaintext* T dengan kunci U akan menghasilkan *ciphertext* N.

Proses enkripsi dan dekripsi menggunakan algoritma Vigenere *cipher* dapat ditulis dalam notasi *pseudocode* yang ditunjukkan oleh gambar 2.4 dan 2.5.

*Pseudocode* proses enkripsi menggunakan algoritma Vigenere *cipher*

PROGRAM VigenereEncrypt

DEKLARASI

plain, key, cipher: string

plainlt, keylt, i, j, A, B, ..., Z: integer

ALGORITMA

read(plain, key, plainlt, keylt)

i=1

while keylt<plainlt do

key=key+key[i]

i=i+1

keylt=keylt+1

endwhile

A=0, B=1, ..., Z=25

for j=1 to plainlt do

cipher[j]=plain[j]+key[j]

if cipher[j]>25 then

cipher[j]=cipher[j] - 26

endif

cipher=cipher+cipher[j]

endfor

write(cipher)

**Gambar 2.4 *Pseudocode* proses enkripsi menggunakan algoritma Vigenere *cipher***

*Pseudocode* proses dekripsi menggunakan algoritma Vigenere *cipher*

PROGRAM VigenereDecrypt

DEKLARASI

plain, key, cipher: string

cipherlt, keylt, i, j, A, B, ..., Z: integer

ALGORITMA

read(plain, key, cipherlt, keylt)

i=1

while keylt<cipherlt do

key=key+key[i]

i=i+1

keylt=keylt+1

endwhile

A=0, B=1, ..., Z=25

for j=1 to cipherlt do

plain[j]=cipher[j]-key[j]

if plain[j]<0 then

plain[j]=plain[j] + 26

endif

plain=plain+plain[j]

endfor

write(plain)

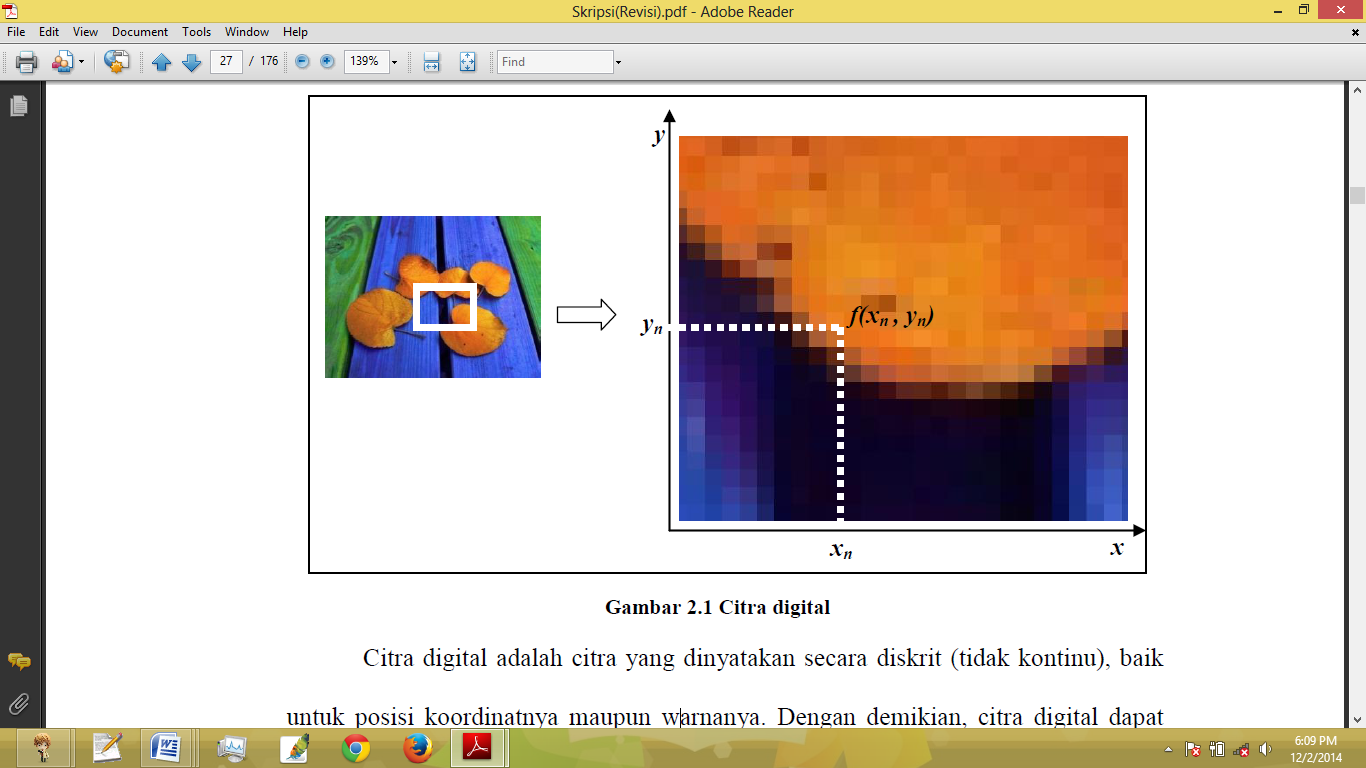
**Gambar 2.5 *Pseudocode* proses dekripsi menggunakan algoritma Vigenere *cipher***

1. **Citra**

Citra didefinisikan sebagai fungsi dua dimensi *f(x,y)* dimana *x* dan *y* adalah koordinat ruang dan besaran *f* adalah intensitas dari citra tersebut pada koordinat *(x,y)* (Gonzalez, 2008). Citra merupakan representasi dua dimensi untuk bentuk fisik objek tiga dimensi. Secara teoritis, citra dapat dikelompokkan menjadi dua macam, yaitu citra kontinu dan citra diskrit (digital). Citra kontinu dihasilkan dari sistem optik yang menerima sinyal analog, contohnya mata manusia dan kamera analog. Sedangkan citra digital dihasilkan melalui proses digitalisasi terhadap citra kontinu.

* + 1. **Citra Digital**

Citra digital adalah citra dengan *f(x,y)* yang nilainya didigitalisasikan baik dalam koordinat spasial maupun dalam *gray level*. Digitalisasi dari koordinat spasial citra disebut dengan *image sampling*, sedangkan digitalisasi dari *gray level* citra disebut dengan *gray level quantization*. Citra digital dapat direpresentasikan sebagai suatu matriks dimana baris dan kolomnya merepresentasikan suatu titik di dalam citra, dan nilai elemen matriks tersebut menunjukkan *gray level* di titik tersebut (Gonzalez, 2008). Citra digital juga dapat didefinisikan sebagai fungsi dua variabel *f(xn,yn)*, dimana *xn* dan *yn* adalah koordinat spasial sedangkan nilai *f(x,y)* adalah intensitas citra pada koordinat tersebut seperti yang ditunjukkan pada gambar 2.6.



**Gambar 2.6 Citra digital**

Citra digital adalah citra yang dinyatakan secara diskrit (tidak kontinu), baik untuk posisi koordinat maupun warnanya. Dengan demikian, citra digital dapat digambarkan sebagai suatu matriks, dimana indeks baris dan indeks kolom dari matriks menyatakan posisi suatu titik di dalam citra dan harga dari elemen matriks menyatakan warna citra pada titik tersebut. Dalam citra digital yang dinyatakan sebagai susunan matriks seperti ini, elemen-elemen matriks tadi disebut juga dengan istilah *pixel* yang berasal dari kata *picture element*.

Suatu *pixel* memiliki nilai dalam rentang tertentu dari nilai minimum sampai nilai maksimum. Jangkauan yang digunakan berbeda-beda tergantung jenis warnanya, namun secara umum jangkauannya adalah 0-255. Citra dengan penggambaran seperti ini digolongkan ke dalam citra *integer*. Berikut adalah jenis-jenis citra berdasarkan nilai *pixel*-nya (Putra, 2010).

1. Citra biner (monokrom)

Citra biner atau monokrom merupakan citra 1 *bit* yang terdiri dari dua warna yaitu hitam dan putih. Jadi, pada citra biner nilai dari setiap *pixel*-nya hanya memiliki dua kemungkinan yaitu hitam atau putih. Citra biner disebut juga citra satu kanal karena warnanya hanya ditentukan oleh satu fungsi intensitas saja. Citra biner seringkali muncul sebagai hasil dari proses pengolahan seperti segmentasi, *thresholding*, *morphology*, ataupun *dithering*.

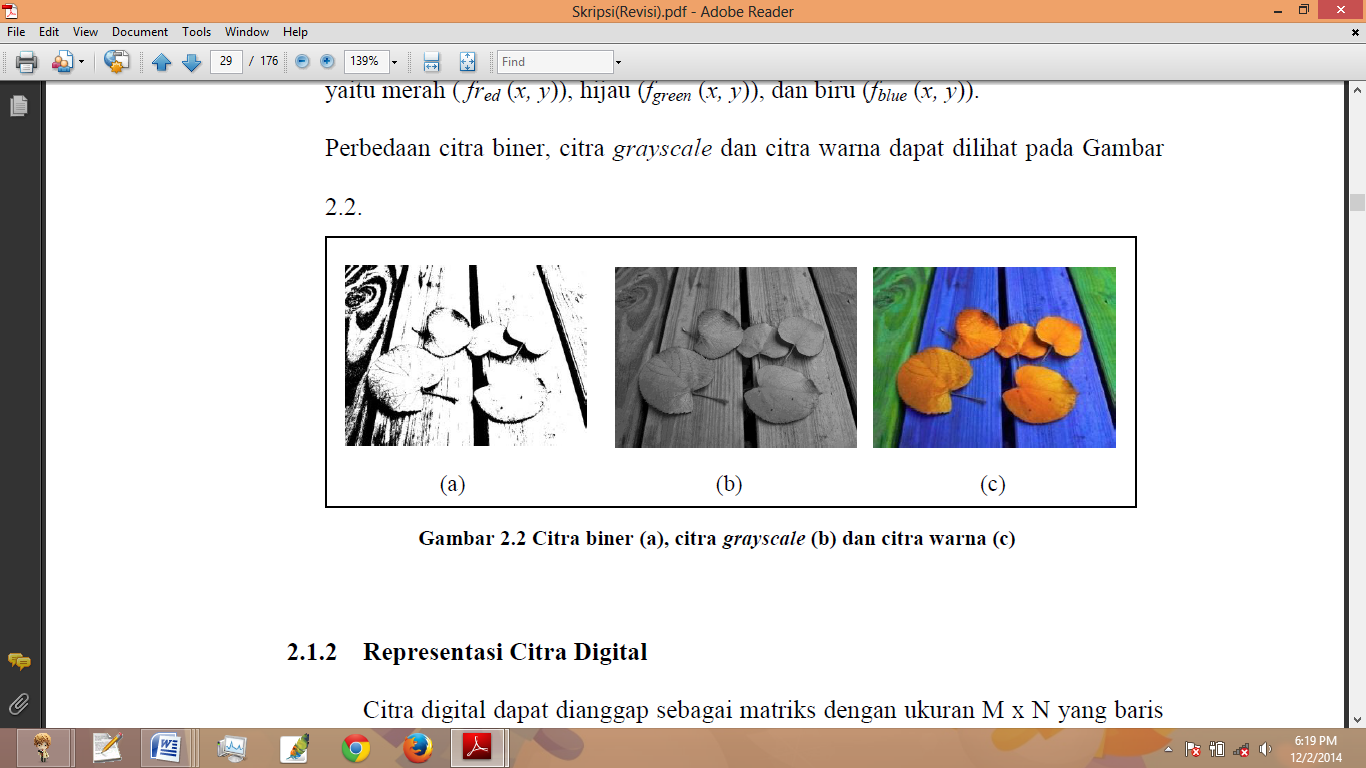
1. Citra skala keabuan (*grayscale*)

Dikatakan format citra skala keabuan karena pada umumnya warna yang dipakai adalah hitam sebagai warna minimum dan putih sebagai warna maksimumnya sehingga warna antara kedua warna tersebut adalah abu-abu. Citra warna keabuan disusun oleh warna dasar *red*, *green*, dan *blue* yang masing-masing memiliki nilai dasar sama, misalnya R=67, G=67, dan B=67. Nilai tersebut digunakan untuk menunjukkan tingkat intensitas. Untuk mendapatkan citra *grayscale* dari sebuah citra RGB digunakan persamaan

dimana *I(x,y)* adalah level keabuan pada suatu koordinat yang diperoleh dengan mengatur komposisi warna merah (R), hijau (G), dan biru (B) yang ditunjukkan oleh nilai parameter *α*, *β*, dan *γ*. Secara umum nilai ketiga parameter tersebut adalah 0,33. Nilai yang lain juga dapat diberikan untuk ketiga parameter tersebut asalkan total keseluruhan nilainya adalah 1.

1. Citra warna

Citra berwarna (*color image*) dikenal dengan nama citra spektral karena warna pada citra disusun oleh tiga komponen warna yang disebut komponen RGB, yaitu merah (*red*), hijau (*green*), dan biru (*blue*) (Gonzalez dan Woods, 2008). Intensitas suatu titik pada citra berwarna merupakan kombinasi dari tiga intensitas yaitu merah *(fred(x,y))*, hijau *(fgreen(x,y))*, dan biru *(fblue(x,y))*. Perbedaan citra biner, citra *grayscale*, dan citra warna ditunjukkan oleh gambar 2.7.



**Gambar 2.7 Citra biner (a), citra *grayscale* (b), dan citra warna (c)**

1. **Metode *Pixel Value Differencing* (PVD)**

*Pixel Value Differencing* merupakan salah satu metode yang dapat digunakan dalam steganografi. Metode ini beroperasi pada ranah spasial dari citra. Berdasarkan analisis terhadap sistem penglihatan manusia, mata manusia tidak sensitif terhadap perubahan pada *pixel* yang memiliki kontras tinggi melainkan sensitif terhadap perubahan pada *pixel* yang memiliki kontras rendah. Berdasarkan sifat tersebut, dapat disimpulkan bahwa lebih banyak *bit* data rahasia yang dapat disisipkan pada *pixel* yang memiliki kontras tinggi dan sedikit *bit* data rahasia yang dapat disisipkan pada *pixel* dengan kontras rendah. Hal tersebut yang menjadi dasar pemikiran metode *Pixel Value Differencing* pada steganografi (Wu dan Tsai, 2003).

Proses penyisipan pada metode ini dilakukan dengan cara membandingkan dua *pixel* yang bertetangga dengan menggunakan persamaan (2.7) (Wu dan Tsai, 2003).

Hasil dari perbandingan tersebut digunakan untuk mengetahui berapa banyak *bit* yang dapat disisipkan ke dalam dua *pixel* yang dibandingkan. Metode ini menggunakan skema Wu dan Tsai untuk mengetahui *range* dari perbandingan *pixel* sebelumnya. Skema Wu dan Tsai yang digunakan yaitu R = {[0,7], [8,15], [16,31], [32,63], [64,127], [128,255]}. Skema ini digunakan untuk mengetahui terdapat di *range* mana selisih dari dua *pixel* tersebut. Jika telah diketahui dimana letak *range*-nya, maka jumlah *bit* pesan yang dapat disisipkan dapat diketahui dengan persamaan (2.8) (Wu dan Tsai, 2003).

dimana:

t = jumlah *bit* yang dapat disisipkan

wi = lebar *range*

Penyisipan pesan dapat dilakukan dengan mengambil sebanyak t *bit* dari pesan yang akan disisipkan. Selanjutnya dihitung *difference value* yang baru untuk penyisipan ke dalam citra menggunakan persamaan (2.9) (Wu dan Tsai, 2003).

dimana:

b = nilai desimal *bit* yang disisipkan

li = nilai terkecil dari skema Wu dan Tsai, letak *range* selisih perbandingan dua *pixel*

Untuk menyisipkan pesan ada beberapa aturan yang harus dipenuhi, yaitu:

1. Jika Pi ≥ Pi+1 dan d’i > di, maka (Pi + ⌈m/2⌉, Pi+1 – ⌊m/2⌋).
2. Jika Pi < Pi+1 dan d’i > di, maka (Pi - ⌊m/2⌋, Pi+1 + ⌈m/2⌉).
3. Jika Pi ≥ Pi+1 dan d’i ≤ di, maka (Pi - ⌈m/2⌉, Pi+1 + ⌊m/2⌋).
4. Jika Pi < Pi+1 dan d’i ≤ di, maka (Pi + ⌊m/2⌋, Pi+1 – ⌈m/2⌉).

dimana m didapat dari selisih d’i dengan di dengan menggunakan persamaan (2.10) (Wu dan Tsai, 2003).

Proses-proses tersebut dilakukan terus hingga *bit* pesan tersisipi semuanya ke dalam citra.

Proses ekstraksi pesan dari citra stego menggunakan metode ini dimulai dengan menghitung *difference value* (di) antara dua *pixel* yang bertetangga. *Difference value* tersebut digunakan untuk mengetahui nilai *continuous range* (R) yang sudah didefinisikan menggunakan skema Wu dan Tsai.

Berdasarkan informasi tersebut dapat diketahui ukuran data rahasia yang disisipkan pada kedua *pixel* menggunakan persamaan 2.8, sehingga pesan rahasia yang telah disisipkan didapatkan kembali. Proses ekstraksi ini dilakukan sampai semua data rahasia yang telah disisipkan didapatkan kembali.

Proses *embedding* dan *retrieve* menggunakan algoritma *Pixel Value Differencing* dapat ditulis dalam notasi *pseudocode* yang ditunjukkan oleh gambar 2.8 dan 2.9.

*Pseudocode* proses *embedding* menggunakan algoritma *Pixel Value Differencing*

PROGRAM PVDEmbed

DEKLARASI

const maxr=100

const maxc=100

msg, b\_msg: string

img: array[1..maxr, 1..maxc] of integer

d1, d2, t, m, i, j, k, w, l, b, x, y: integer

ALGORITMA

read(msg, img)

b\_msg=msg

for i=1 to maxr do

for j=1 to maxc do

if j mod 2=0 then

d1=|img[i, j]-img[i, j-1]|

if d1>=0 and d1<=7

w=8

l=0

endif

if d1>=8 and d1<=15

w=8

l=8

endif

if d1>=16 and d1<=31

w=16

l=16

endif

if d1>=32 and d1<=63

w=32

l=32

endif

if d1>=64 and d1<=127

w=64

l=64

endif

if d1>128 and d1<=255

w=128

l=128

endif

t=|log 2 w|

b\_msg=t

b=b\_msg

d2=l+b

m=|d2-d1|

if img[i,j-1]>=img[i,j] and d2>d1 then

x=img[i, j-1]+⌈m/2⌉

y=img[i, j]-⌊m/2⌋

endif

if img[i,j-1]<img[i,j] and d2>d1 then

x=img[i, j-1]-⌊m/2⌋

y=img[i, j]+⌈m/2⌉

endif

if img[i,j-1]>=img[i,j] and d2<=d1 then

x=img[i, j-1]-⌈m/2⌉

y=img[i, j]+⌊m/2⌋

endif

if img[i,j-1]<img[i,j] and d2<=d1 then

x=img[i, j-1]+⌊m/2⌋

y=img[i, j]-⌈m/2⌉

endif

img[i, j-1]=x

img[i, j]=y

write(img[i, j])

endif

endfor

endfor

**Gambar 2.8 *Pseudocode* proses *embedding* menggunakan algoritma *Pixel Value Differencing***

*Pseudocode* proses *retrieve* menggunakan algoritma *Pixel Value Differencing*

PROGRAM PVDRetrieve

DEKLARASI

const maxr=100

const maxc=100

msg, b\_msg: string

img: array[1..maxr, 1..maxc] of integer

d1, d2, t, b, i, j: integer

ALGORITMA

read(img)

for i=1 to maxr do

for j=1 to maxc do

if j mod 2=0 then

d1=|img[i, j]-img[i, j-1]|

if d1>=0 and d1<=7

w=8

l=0

endif

if d1>=8 and d1<=15

w=8

l=8

endif

if d1>=16 and d1<=31

w=16

l=16

endif

if d1>=32 and d1<=63

w=32

l=32

endif

if d1>=64 and d1<=127

w=64

l=64

endif

if d1>128 and d1<=255

w=128

l=128

endif

t=|log 2 w|

b=d1-l

b\_msg=b

endif

endfor

endfor

b\_msg=msg

write(msg)

**Gambar 2.9 *Pseudocode* proses *retrieve* menggunakan algoritma *Pixel Value Differencing***

1. ***Rational Unified Process* (RUP)**

*Rational Unified Process* (RUP) adalah pendekatan pengembangan perangkat lunak yang dilakukan berulang-ulang (*iterative*), fokus pada arsitektur (*architecture-centric*), dan lebih diarahkan berdasarkan penggunaan kasus (*use case-driven*). RUP merupakan proses rekayasa perangkat lunak dengan pendefinisian yang baik (*well-defined*) dan penstrukturan yang baik (*well-structured*). RUP menyediakan pendefinisian struktur yang baik untuk proses alur hidup proyek perangkat lunak. RUP adalah sebuah produk proses perangkat lunak yang dikembangkan oleh *Rational Software* yang diakuisisi oleh IBM di bulan Februari 2003 (Rosa A. S. dan M. Shalahuddin, 2011).

RUP memiliki empat buah tahap atau fase yang dapat dilakukan secara iteratif, yaitu:

1. *Inception* (permulaan)

Tahap ini lebih pada memodelkan proses bisnis yang dibutuhkan (*bussiness modeling*) dan mendefinisikan kebutuhan (*requirements*) akan sistem yang dibuat. Berikut adalah tahap yang dibutuhkan pada fase ini.

1. Memahami ruang lingkup proyek (termasuk pada biaya, waktu, kebutuhan, resiko, dan lain sebagainya).
2. Membangun kasus bisnis yang dibutuhkan.

Hasil yang diharapkan pada tahap ini adalah memenuhi *lifecycle objective milestone* (batas/tonggak objektif dari siklus). Jika pada akhir tahap ini target yang diinginkan tidak dicapai maka dapat dibatalkan atau diulang kembali setelah dirancang ulang agar kriteria yang diinginkan dapat dicapai.

1. *Elaboration* (perluasan/perencanaan)

Tahap ini lebih difokuskan pada perencanaan arsitektur sistem. Tahap ini juga dapat mendeteksi apakah arsitektur sistem yang diinginkan dapat dibuat atau tidak dan mendeteksi resiko yang mungkin terjadi dari arsitektur yang dibuat. Tahap ini lebih pada analisis dan desain sistem serta implementasi sistem yang fokus pada purwarupa (*prototype*) sistem.

1. *Construction* (konstruksi)

Tahap ini fokus pada pengembangan komponen dan fitur-fitur sistem. Tahap ini lebih pada implementasi dan pengujian sistem yang fokus pada implementasi perangkat lunak pada kode program. Tahap ini menghasilkan produk perangkat lunak dimana menjadi syarat *initial operational capability milestone* atau batas/tonggak kemampuan operasional awal.

1. *Transition* (transisi)

Tahap ini lebih pada *deployment* atau instalasi sistem agar dapat dimengerti oleh *user*. Aktifitas pada tahap ini termasuk pada pelatihan *user* dan pemeliharaan dan pengujian sistem apakah sudah memenuhi harapan *user*. Produk perangkat lunak juga disesuaikan dengan kebutuhan yang didefinisikan pada tahap *inception*. Jika semua kriteria objektif sudah terpenuhi maka dianggap sudah memenuhi *product release milestone* (batas/tonggak peluncuran produk) dan pengembangan perangkat lunak selesai dilakukan.

Akhir dari keempat fase ini adalah produk perangkat lunak yang sudah lengkap. Keempat fase pada RUP dijalankan secara berurutan dan iteratif dimana setiap iterasi dapat digunakan untuk memperbaiki iterasi berikutnya.

1. **Perangkat Perancangan Sistem**
   * 1. ***Unified Modeling Language* (UML)**

Menurut Rosa dan Shalahuddin (2011), *Unified Modeling Language* (UML) merupakan bahasa visual untuk pemodelan dan komunikasi sebuah sistem dengan menggunakan diagram dan teks-teks pendukung. UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak. UML hanya berfungsi untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas pada metodologi tertentu, meskipun pada kenyataannya UML paling banyak digunakan pada metodologi berorientasi objek.

Pada UML 2.3 terdapat 13 macam diagram yang dikelompokkan dalam 3 kategori (Rosa A. S dan M. Shalahuddin, 2011), yaitu:

1. *Structure diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang dimodelkan seperti *class diagram*, *object diagram*, *component diagram*, *composite structure diagram*, *package diagram*, dan *deployment diagram*.
2. *Behavior diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahan yang terjadi pada sebuah sistem, seperti *use case diagram*, *activity diagram*, dan *state machine diagram*.
3. *Interaction diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antar subsistem pada suatu sistem, seperti *sequence diagram*, *communication diagram*, *timing diagram*, dan *interaction overview diagram*.

Diagram UML yang digunakan dalam penelitian ini berjumlah 4 buah karena telah cukup untuk menjelaskan tentang sistem yang akan dibangun. Diagram-diagram tersebut adalah (Rosa A. S dan M. Shalahuddin):

1. ***Use case diagram***

*Use case* atau diagram *use case* merupakan pemodelan untuk kelakuan (*behavior*) sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu.

Syarat penamaan pada *use case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *use case* yaitu pendefinisian apa yang disebut aktor dan *use case*.

1. Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang dibuat itu sendiri, jadi walaupun simbol aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.
2. *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antarunit atau aktor.

Simbol-simbol yang ada pada diagram *use case* dapat dilihat pada tabel 2.1.

**Tabel 2.1 Simbol *use case diagram***

|  |  |  |
| --- | --- | --- |
| **SIMBOL** | **NAMA** | **DESKRIPSI** |
|  | *Actor* | Orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri. Biasanya dinyatakan menggunakan kata benda di awal frase nama aktor |
|  | *Use case* | Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor. Biasanya dinyatakan dengan menggunakan kata kerja di awal frase nama *use case* |
|  | *Association* | Komunikasi antara aktor dan *use case* yang berpartisipasi pada *use case* atau *use case* memiliki interaksi dengan aktor |
| <<extend>> | *Extend* | Relasi *use case* tambahan ke sebuah *use case* dimana *use case* yang ditambahkan dapat berdiri sendiri walau tanpa *use case* tambahan itu. Biasanya *use case* tambahan memiliki nama depan yang sama denga *use case* yang ditambahkan |
| <<include>> | *Include* | Relasi *use case* atau tambahan ke sebuah *use case* dimana *use case* yang ditambahkan memerlukan *use case* ini untuk menjalankan fungsinya atau sebagai syarat dijalankan *use case* ini |
|  | *Generalization* | Hubungan generalisasi dan spesialisasi (umum-khusus) antara dua buah *use case* dimana fungsi yang satu adalah fungsi yang lebih umum dari lainnya |

1. ***Activity diagram***

Diagram aktifitas atau *activity diagram* menggambarkan *workflow* (aliran kerja) atau aktifitas dari sebuah sistem atau proses bisnis. Yang perlu diperhatikan disini bahwa diagram aktifitas menggambarkan aktifitas sistem bukan apa yang dilakukan oleh aktor, jadi aktifitas yang dapat dilakukan oleh sistem.

Diagram aktifitas juga banyak digunakan untuk mendefinisikan hal-hal berikut.

1. Rancangan proses bisnis dimana setiap urutan aktifitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan.
2. Urutan atau pengelompokan tampilan dari *system/user interface* dimana setiap aktifitas dianggap memiliki sebuah rancangan antarmuka tampilan.
3. Rancangan pengujian dimana setiap aktifitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.

Simbol-simbol yang ada pada diagram aktifitas dapat dilihat pada tabel 2.2.

**Tabel 2.2 Simbol *activity diagram***

|  |  |  |
| --- | --- | --- |
| **SIMBOL** | **NAMA** | **KETERANGAN** |
|  | Status awal | Status awal aktifitas sistem, sebuah diagram aktifitas memiliki sebuah status awal |
|  | Aktifitas | Aktitifitas yang dilakukan oleh sistem, aktifitas biasanya diawali dengan kata kerja |
|  | Penggabungan/ *join* | Asosiasi penggabungan dimana lebih dari satu aktifitas digabungkan menjadi satu. |
|  | Status akhir | Status akhir yang dilakukan sistem, sebuah diagram aktifitas memiliki sebuah status akhir |

1. ***Sequence diagram***

Diagram sekuen menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antarobjek. Oleh karena itu untuk menggambar diagram sekuen harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

Banyaknya diagram sekuen yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram sekuen sehingga semakin banyak *use case* yang didefinisikan maka diagram sekuen yang harus dibuat juga semakin banyak.

Simbol-simbol yang ada pada diagram sekuen dapat dilihat pada tabel 2.3.

**Tabel 2.3 Simbol *sequence diagram***

|  |  |  |
| --- | --- | --- |
| **SIMBOL** | **NAMA** | **KETERANGAN** |
| **Nama\_aktor**  **atau**  Nama\_aktor  **tanpa waktu aktif** | *Actor* | Orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri. Biasanya dinyatakan menggunakan kata benda di awal frase nama aktor |
|  | *Lifeline* | Menyatakan kehidupan suatu objek |
| Nama\_objek : nama\_kelas | *Object* | Menyatakan objek yang berinteraksi pesan |
|  | *Message* | Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktifitas yang terjadi |

Penomoran pesan berdasarkan urutan interaksi pesan. Penggambaran letak pesan harus berurutan, pesan yang lebih atas dari lainnya adalah yang berjalan terlebih dahulu.

1. ***Class diagram***

Diagram kelas atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi.

Kelas-kelas yang ada pada struktur sistem harus dapat melakukan fungsi-fungsi sesuai dengan kebutuhan sistem. Susunan struktur kelas yang baik pada diagram kelas sebaiknya memiliki jenis-jenis kelas berikut.

1. Kelas main

Kelas yang memiliki fungsi awal dieksekusi ketika sistem dijalankan.

1. Kelas yang menangani tampilan sistem

Kelas yang mendefinisikan dan mengatur tampilan ke pemakai.

1. Kelas yang diambil dari pendefinisian *use case*

Kelas yang menangani fungsi-fungsi yang harus diambil dari pendefinisian *use case*.

1. Kelas yang diambil dari pendefinisian data

Kelas yang digunakan untuk memegang atau membungkus data menjadi sebuah kesatuan yang diambil maupun akan disimpan ke basis data.

Jenis-jenis kelas di atas juga dapat digabungkan satu sama lain sesuai dengan pertimbangan yang dianggap baik asalkan fungsi-fungsi yang sebaiknya ada pada struktur kelas tetap ada. Struktur kelas juga dapat ditambahkan kelas utilitas seperti koneksi ke basis data, membaca *file* teks, dan lain sebagainya sesuai kebutuhan.

Dalam mendefinisikan metode yang ada di dalam kelas perlu memperhatikan apa yang disebut dengan *cohesion* dan *coupling*. *Cohesion* adalah ukuran seberapa dekat keterkaitan instruksi di dalam sebuah metode terkait satu sama lain sedangkan *coupling* adalah ukuran seberapa dekat keterkaitan instruksi antara metode yang satu dengan metode yang lain dalam sebuah kelas. Sebagai aturan secara umum maka sebuah metode yang dibuat harus memiliki kadar *cohesion* yang kuat dan kadar *coupling* yang lemah.

Simbol-simbol yang ada pada diagram kelas dapat dilihat pada tabel 2.4.

**Tabel 2.4 Simbol *class diagram***

|  |  |  |
| --- | --- | --- |
| **SIMBOL** | **NAMA** | **KETERANGAN** |
|  | Kelas | Kelas pada struktur sistem |
| **Nama\_interface** | Antarmuka | Sama dengan konsep *interface* dalam pemrograman berorientasi objek |
|  | Asosiasi | Relasi antar kelas dengan makna umum, asosiasi biasanya juga disertai dengan *multiplicity* |
|  | Asosiasi berarah | Relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan *multiplicity* |
|  | Generalisasi | Relasi antar kelas dengan makna generalisasi-spesialisasi (umum-khusus) |
|  | *Dependency* | Relasi antar kelas dengan makna kebergantungan antar kelas |
|  | *Aggregation* | Relasi antar kelas dengan makna semua bagian (*whole part*) |

1. **Bahasa Pemrograman Java**

Java adalah bahasa pemrograman yang dapat dijalankan di berbagai komputer termasuk telepon genggam. Java telah menunjukkan keberhasilannya dengan membuat halaman *web* statis menjadi interaktif dan dinamis. Sejak debutnya, Java telah mengambil pasar dan mengisi *browser* dimana-mana dengan animasi, audio, dan interaktivitas *real-time* (Sommerville, 2003).

Bahasa ini awalnya dibuat oleh James Gosling saat masih bergabung di Sun Microsystems yang saat ini merupakan bagian dari Oracle dan dirilis tahun 1995. Bahasa ini banyak mengadopsi sintaks yang terdapat pada C dan C++ namun dengan sintaks model objek yang lebih sederhana serta dukungan rutin-rutin aras bawah yang minimal. Aplikasi-aplikasi berbasis Java umumnya dikompilasi ke dalam *p-code* (*bytecode*) dan dapat dijalankan pada berbagai mesin virtual Java.

Java merupakan bahasa pemrograman yang bersifat umum/non-spesifik (*general purpose*) dan secara khusus didesain untuk memanfaatkan dependensi implementasi seminimal mungkin. Karena fungsionalitasnya memungkinkan aplikasi Java mampu berjalan di beberapa platform sistem informasi yang berbeda. Java dikenal pula dengan slogannya, “Tulis sekali, jalankan dimana pun”. Saat ini Java merupakan bahasa pemrograman yang paling populer digunakan dan secara luas dimanfaatkan dalam pengembangan berbagai jenis perangkat lunak maupun aplikasi berbasis *web*.

Java merupakan bahasa pemrograman yang banyak digunakan oleh para *programmer* dan *software developer* untuk mengembangkan berbagai tipe aplikasi, mulai dari aplikasi *console*, aplikasi *desktop*, *applet* (aplikasi yang berjalan di lingkungan *web browser*), sampai ke aplikasi-aplikasi yang berskala *enterprise*.

Sebagai sebuah *platform*, Java terdiri atas dua bagian utama yaitu:

1. *Java Virtual Machine* (JVM)

*Java Virtual Machine* adalah sebuah spesifikasi untuk sebuah komputer abstrak. JVM terdiri dari sebuah kelas pemanggil dan sebuah *interpreter* Java yang mengeksekusi kode arsitektur netral. Kelas pemanggil memanggil *file* API untuk dieksekusi oleh *interpreter* Java. Dengan kata lain JVM adalah perantara antara program yang akan dijalankan dan sistem operasi yang sedang digunakan.

1. Java *Application Programming Interface* (Java API)

Java API merupakan komponen-komponen dan kelas Java yang sudah jadi yang memiliki berbagai kemampuan, seperti kemampuan untuk menangani objek, *string*, angka, dsb. Sun membagi arsitektur Java menjadi tiga bagian.

1. J2EE (Java 2 *Enterprise Edition*)

J2EE adalah kelompok dari beberapa API dari Java dan teknologi lain selain Java. J2EE dirancang untuk membuat aplikasi yang rumit. J2EE sering dianggap sebagai *middleware* atau teknologi yang berjalan di *server*, namun sebenarnya J2EE tidak hanya terbatas untuk itu. Faktanya J2EE juga mencakup teknologi yang dapat digunakan di semua lapisan dari sebuah sistem informasi. Implementasi J2EE menyediakan kelas dasar dan API dari Java yang mendukung pengembangan dan rutin standar untuk aplikasi *client* maupun *server*, termasuk aplikasi yang berjalan di *web browser*. Untuk aplikasi berbasis *web*, aplikasi sistem tersebar dengan beraneka ragam klien dengan kompleksitas yang tinggi.

1. J2SE (Java 2 *Standard Edition*)

J2SE atau Java 2 *Standard Edition* merupakan bahasa pemrograman Java untuk aplikasi *desktop* yang menggunakan *object-oriented programming*. J2SE terdiri dari dua buah produk yang dikeluarkan untuk membantu dalam membuat aplikasi tanpa tergantung dari *platform* yang digunakan, yaitu:

* Java *Runtime Environment* (JRE)

Java *Runtime Environment* menyediakan perpustakaan Java *Virtual Machine* (JVM) dan komponen lain untuk menjalankan *applet* dan aplikasi yang ditulis dengan bahasa pemrograman Java. Selain itu, terdapat dua buah kunci teknologi yang merupakan bagian dari JRE, yaitu *Plug-in*, yang memungkinkan menjalankan *applet* di *browser* dan Java *Web Start*, yang menyebarkan aplikasi mandiri melalui jaringan. JRE tidak mengandung utilitas seperti *compiler* atau *debugger* untuk mengembangkan *applet* dan aplikasi.

* Java *Development Kit* (JDK)

Java *Development Kit* merupakan perangkat lunak yang digunakan untuk manajemen dan membangun berbagai aplikasi Java. JDK merupakan *superset* dari JRE, berisikan segala sesuatu yang ada di JRE ditambahkan *compiler* dan *debugger* yang diperlukan untuk mengembangkan *applet* dan aplikasi. Pada JDK, terdapat berbagai *tools* yang digunakan untuk membangun aplikasi Java.

1. J2ME (Java 2 *Micro Edition*)

J2ME adalah lingkungan pengembangan yang dirancang untuk meletakkan perangkat lunak Java pada barang elektronik beserta perangkat pendukungnya. J2ME biasa digunakan pada telepon selular, *pager*, *personal digital assistant* (PDA), dan sejenisnya. Ada dua tipe aplikasi pada Java yaitu aplikasi *standalone* dan *applet*. Aplikasi *standalone* dapat dijalankan menggunakan Java *interpreter* dan ideal untuk membuat sebuah aplikasi besar. Sedangkan *applet* adalah suatu aplikasi yang dijalankan melalui *web browser* ataupun *applet viewer*. *Applet* cocok digunakan untuk aplikasi *web*. Perbedaan *applet* dengan Java *standalone* adalah:

* *Applet* melakukan *extend* dari *class applet*.
* *Applet* tidak mempunyai *method* *main*().
* *Applet* mempunyai beberapa batasan keamanan seperti tidak diperbolehkan membaca atau menulis *file* pada sistem komputer.
* *Applet* tidak dapat menjalankan program lain pada komputer yang menjalankan *applet*.

*Source code* pada Java biasanya diakhiri dengan ekstensi .java (contohnya Selamat.java). Pada saat dikompilasi *file* Java akan menjadi berakhiran .class (contohnya Selamat.class). Untuk mengkompilasi program Java diperlukan javac.exe dari JDK dan untuk menjalankan aplikasi Java *standalone* dipergunakan java.exe.

Adapun kelebihan bahasa pemrograman Java adalah sebagai berikut.

1. *Multiplatform*

Kelebihan utama dari Java ialah dapat dijalankan di beberapa *platform*/sistem operasi komputer, sesuai dengan prinsip “Tulis sekali, jalankan dimana pun”. Dengan kelebihan ini pemrogram cukup menulis sebuah program Java dan dikompilasi (diubah dari bahasa yang dimengerti manusia menjadi bahasa mesin/*bytecode*) sekali lalu hasilnya dapat dijalankan pada beberapa *platform* tanpa perubahan.

1. OOP (*Object-Oriented Programming*)
2. Perpustakaan kelas yang lengkap

Java terkenal dengan kelengkapan *library*/perpustakaan (kumpulan program yang disertakan dalam pemrograman Java) yang sangat memudahkan para pemrogram untuk membangun aplikasinya.

1. Bergaya C++

Memiliki sintaks seperti bahasa pemrograman C++ sehingga menarik banyak pemrogram C++ untuk pindah ke Java.

Adapun kekurangan bahasa pemrograman Java adalah sebagai berikut.

1. Tulis sekali, jalankan dimana pun.

Masih ada beberapa hal yang tidak kompatibel antara *platform* satu dengan *platform* lain.

1. Mudah didekompilasi

Dekompilasi adalah proses membalikkan dari kode jadi menjadi kode sumber. Ini dimungkinkan karena kode jadi Java merupakan *bytecode* yang menyimpan banyak atribut bahasa tingkat tinggi seperti nama-nama kelas, metode, dan tipe data.

1. Penggunaan memori yang banyak

Penggunaan memori untuk program berbasis Java jauh lebih besar daripada bahasa tingkat tinggi generasi sebelumnya seperti C/C++ dan Pascal.

1. **NetBeans**

NetBeans dimulai pada tahun 1996 sebagai Xelfi (kata bermain pada Delphi), IDE Java proyek mahasiswa di bawah bimbingan Fakultas Matematika dan Fisika di Charles University di Praha. Pada tahun 1997, Stanek Romawi membentuk perusahaan di sekitar proyek tersebut dan menghasilkan versi komersial IDE NetBeans hingga kemudian dibeli oleh Sun Microsystems pada tahun 1999. Komunitas NetBeans terus tumbuh berkat individu dan perusahaan yang menggunakan dan berkontribusi dalam proyek ini.

NetBeans mengacu pada dua *platform* kerangka untuk aplikasi *desktop* Java dan sebuah lingkungan pengembangan terpadu (IDE) untuk pengembangan dengan Java, Javascript, PHP, Phyton, Ruby, Groovy, C, C++, Scala, Clojure, dan lain-lain.

IDE NetBeans ditulis dalam Java dan berjalan dimanapun JVM diinstal, termasuk Windows, Mac OS, Linux, dan Solaris. Sebuah JDK diperlukan untuk pengembangan fungsionalitas Java, tetapi tidak diperlukan untuk pembangunan di bahasa pemrograman lain.

*Platform* NetBeans memungkinkan aplikasi dikembangkan dari satu set modular komponen *software* yang disebut modul. Aplikasi berbasis *platform* NetBeans (termasuk IDE NetBeans) dapat diperpanjang oleh pengembang pihak ketiga.