BAB II

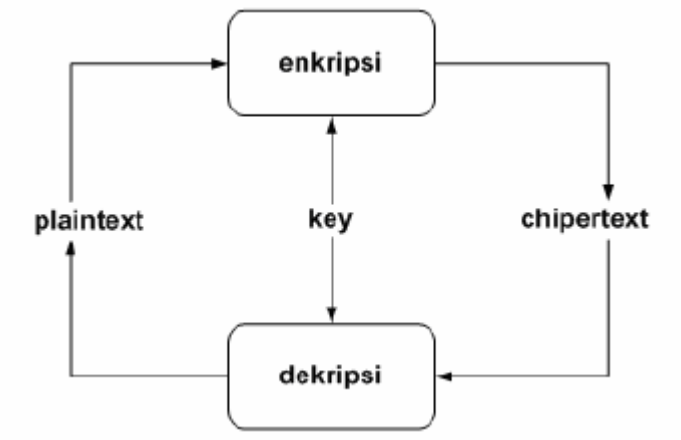
LANDASAN TEORI

2.1 Penyandian Data (Kriptografi)

Kata Kriptografi berasal dari bahasa Yunani *kryptos* (tersembunyi) dan *graphien* (menulis). Kriptografi atau penyandian data merupakan seni dan ilmu untuk menjaga berita[1]. Selain pengertian tersebut terdapat pula pengertian ilmu yang mempelajari teknik-teknik matematika yang berhubungan dengan aspek keamanan informasi seperti kerahasiaan data, keabsahan data, integritas data, serta autentikasi data [1]. Ilmu atau seni ini banyak diterapkan disemua bidang, khususnya bidang militer dan agen rahasia. Sebenarnya ilmu ini sudah diterapkan sejak dahulu dan sampai kini terus dikembangkan dan diteliti penggunaannya.

Kriptografi mempunyai 2 (dua) bagian yang penting, yaitu *enkripsi* dan *dekripsi*. *Enkripsi* adalah proses penyandian dari pesan asli menjadi pesan yang tidak dapat diartikan seperti pesan aslinya. *Dekripsi* sendiri berarti merubah pesan yang sudah disandikan menjadi pesan aslinya. Pesan asli biasanya disebut *plaintext*, sedangkan pesan yang sudah disandikan disebut *ciphertext.* Adapun algoritma matematis yang digunakan pada proses *enkripsi* yakni disebut c*hiper* dan sistem yang memanfaatkan Kriptografi untuk mengamankan sitem informasi disebut *kriptosistem*.

Dalam Gambar 2.1 berikut ini memperlihatkan aliran proses *enkripsi* dan deskripsi :



Gambar 2.1: Aliran *Enkripsi* dan *Deskripsi* pada *Kriptografi*

Adapun tujuan dari sistem Kriptografi adalah sebagai berikut :

1. **Kerahasiaan**, adalah layanan yang digunakan untuk menjaga isi dari informasi dari siapapun kecuali yang memiliki otoritas atau kunci rahasia untuk membuka/mengupas informasi yang telah disandi.
2. **Integritas data,** adalah berhubungan dengan penjagaan dari perubahan data secara tidak sah. Untuk menjaga integritas data, sistem harus memiliki kemampuan untuk mendeteksi manipulasi data oleh pihak-pihak yang tidak berhak, antara lain penyisipan, penghapusan, dan pensubsitusian data lain kedalam data yang sebenarnya.
3. **Autentikasi,** adalah berhubungan dengan identifikasi/pengenalan, baik secara kesatuan sistem maupun informasi itu sendiri. Dua pihak yang saling berkomunikasi harus saling memperkenalkan diri. Informasi yang dikirimkan

melalui kanal harus diautentikasi keaslian, isi datanya, waktu pengiriman, dan lain-lain.

4. **Non-repudiasi,** adalah usaha untuk mencegah terjadinya penyangkalan terhadap pengiriman atau terciptanya suatu informasi oleh yang mengirimkan atau membuat.

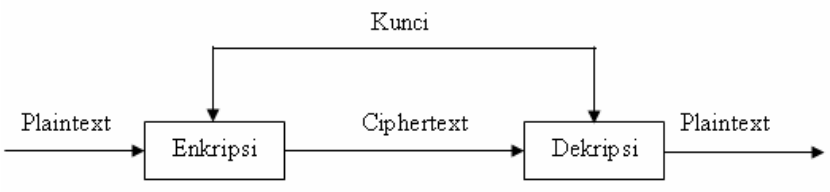
2.1.1 Chiper

Chiper merupakan teknologi untuk *enkripsi* dan *dekripsi* data. Teknik Kriptografi untuk *enkripsi* data ada 2 macam :

2.1.1.1 Kriptografi Simetri

Algoritma simetris atau disebut juga algoritma Kriptografi konvensional adalah algoritma yang menggunakan kunci yang sama untuk proses *enkripsi* dan proses *dekripsi*.

Algoritma Kriptografi simetris dibagi menjadi 2 kategori yaitu algoritma aliran (*Stream Ciphers*) dan algoritma blok (*Block Ciphers*). Pada algoritma aliran, proses penyandiannya berorientasi pada satu bit atau satu byte data. Sedang pada algoritma blok, proses penyandiannya berorientasi pada sekumpulan bit atau byte data (per blok). Contoh algoritma kunci simetris adalah DES (*Data Encryption Standard*), blowfish, twofish, MARS, IDEA, 3DES (DES diaplikasikan 3 kali), AES (*Advanced Encryption Standard*) yang bernama asli Rijndael.

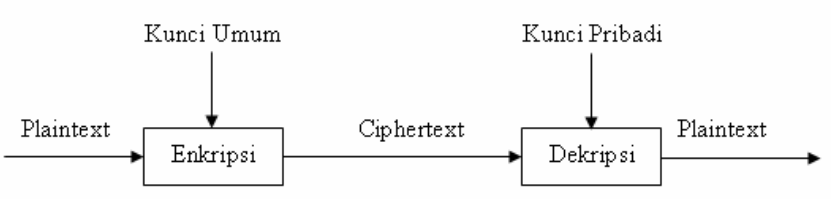


Gambar 2.2 : Proses *Enkripsi*/*Deskripsi* Algoritma Simetrik

2.1.1.2 Kriptografi Asimetri

Kriptografi asimetrik (*asymmetric cryptography*) adalah algoritma yang menggunakan kunci yang berbeda untuk proses *enkripsi* dan *dekripsi*. Kunci *enkripsi* dapat disebarkan kepada umum dan dinamakan sebagai kunci publik (*public key*) sedangkan kunci *dekripsi* disimpan untuk digunakan sendiri dan dinamakan sebagai kunci pribadi (*private key*). Oleh karena itulah, Kriptografi ini dikenal pula dengan nama Kriptografi kunci publik (*public key cryptography*). Contoh algoritma terkenal yang menggunakan kunci asimetris adalah RSA (*Riverst Shamir Adleman*) dan ECC (*Elliptic Curve Cryptography*).

Pada kriptosistem asimetrik, setiap pelaku sistem informasi memiliki sepasang kunci, yaitu kunci publik dan kunci pribadi. Kunci publik didistribusikan kepada umum, sedangkan kunci pribadi disimpan untuk diri sendiri.



Gambar 2.3 : Proses Enkripsi / Deskripsi Algoritma Asimetrik

2.1.2 Message Digest / Digital Fingerprint

Ide ini berawal dari pembuatan suatu ringkasan matematik atas sebuah data, dimana perubahan atas data tersebut akan menghasilkan ringkasan matematik yang berbeda. Sebagai contoh ada sebuah data X memiliki nilai *digital fingerprint* : d07182257bce13d49a5c183864e4a277, maka jika anda merubah barang 1 byte data tersebut, maka nilai digital fingerprint akan berbeda. Teknologi ini menyediakan solusi bahwa data selalu terjamin integritasnya karena perubahan atas data yang ditransmisikan akan terdeteksi dengan mudah. Contoh algoritma *message digest* adalah : MD2, MD4, MD5, SHA1, RIPEM160, dan RIPEM128.

Sebuah *message digest* berfungsi sebagai sidik jari bagi sebuah data. Dengan sidik jari yang berbeda, maka data yang diperoleh juga berbeda sekalipun perubahan tersebut hanya 1 bit saja. Jadi secara umum *message digest* digunakan untuk memeriksa integritas sebuah data. Hal tersebut bisa anda lakukan ketika anda mendownload sebuah file dari server, maka anda dapat menghitung nilai digest-nya dan membandingkannya dengan nilai hasil komputasi di server. Jika hasilnya sama, maka anda boleh yakin bahwa selama proses pengiriman, data tidak mengalami modifikasi oleh pihak lain.

2.1.3 Digital Signature

Idenya adalah dari meng*enkripsi message digest* dari suatu data dengan kunci rahasia (*private key*) seseorang. Pihak lain kemudian dapat mendapatkan *message digest* aslinya dengan men*dekripsi* data dengan kunci publik dari orang tersebut. Pengembangan dari model ini adalah adanya *certificate* yakni setiap orang memiliki

semacam *certificate* yang disahkan *Kajian dan Implementasi Sistem* oleh pihak ketiga yang ditunjuk, misalnya pengadilan, yang bisa digunakan untuk identitas digital seseorang. Jadi fungsi *certificate* ini seperti kartu tanda penduduk atau kartu SIM. Contoh algoritma signature adalah RSA dan DSA.

Pada algoritma tersebut, data yang dikirimkan ke pihak penerima nantinya tidak sekedar data saja, tapi juga data ditambah dengan *digital signature*-nya.

Untuk meyakinkan bahwa data yang kita terima betul-betul terjamin keamanannya dan dikirimkan oleh pihak yang benar. Penjelasannya adalah sebagai berikut, sebagaimana diketahui bahwa data dikirim bersama dengan *digital signature*-nya. Dari *digital signature* dilakukan *dekripsi* dengan menggunakan kunci public dari pihak yang bersangkutan. Jika gagal di*dekripsi*, maka kunci publik tentunya tidak bersesuain dengan kunci privat (*private key*) dari pihak yang benar, sehingga dari sini kita sudah bisa menentukan bahwa pengirim data adalah pihak yang sah. Dari *dekripsi* yang dilakukan dihasilkan digest dari data, jika nilai digest ini sama dengan hasil nilai digest yang kita lakukan terhadap data maka data terjamin keamanannya.

2.2 Algoritma RSA

Algoritma RSA ditemukan oleh tiga orang yang kemudian nama-nama mereka disingkat menjadi RSA. Ketiga penemu itu adalah Ron Riverst, Adi Shamir, dan Leonard Adleman. RSA dibuat di MIT pada tahun 1977 dan dipatenkan oleh MIT pada tahun 1983[2]. Sejak 21 September tahun 2000, paten tersebut berakhir, sehingga saat ini semua orang dapat menggunakannya dengan bebas. Lebih jauh, RSA adalah algoritma yang mudah untuk diimplementasikan dan dimengerti.

Algoritma RSA adalah sebuah aplikasi dari sekian banyak teori seperti extended euclid algorithm, euler's function sampai fermat theorem. Tidak pernah dibuktikan aman tidaknya RSA, hanya karena sulitnya pemfaktoran bilangan yang sangat besar, maka RSA dianggap aman .

Algoritma RSA merupakan standar *de facto* Kriptografi asimetrik saat ini. Sama halnya dengan Kriptografi asimetrik lainnya, RSA juga menggunakan dua buah kunci yaitu kunci publik dan kunci pribadi. Untuk menghasilkan kunci publik dan kunci pribadi dibuat algoritma yang berfungsi untuk membangkitkan kunci. Kemudian kunci publik digunakan untuk *enkripsi plaintext*, dan kunci pribadi digunakan untuk *dekripsi* sehingga *plaintext* diperoleh kembali. Sedikit keistimewaan pada algoritma RSA, *plaintext* dapat pula di*enkripsi* dengan kunci pribadi dan di*dekripsi* dengan kunci publik. Jadi, proses *enkripsi* dan *dekripsi* dengan kuncinya berlaku secara bolak-balik.

Algoritma RSA pada dasarnya dapat dibedakan menjadi tiga bagian besar, yakni algoritma pembangkitan kunci (*key generation algorithm*), algoritma *enkripsi* (*encryption algorithm*), dan algoritma *dekripsi* (*decryption algorithm*).

Algorima RSA mempunyai beberapa besaran-besaran sebagai berikut[2]:

1. *p* dan *q* merupakan bilangan prima yang besar (rahasia)
2. $n = p.q$ (modulus) (tidak rahasia)
3. $\varphi(n) = (p-1)(q-1)$(fungsi euler) (rahasia)
4. e (kunci publik) (tidak rahasia)
5. d (kunci privat) (rahasia)
6. m (*plainteks*) (rahasia)

7. c (cipherteks) (tidak rahasia)

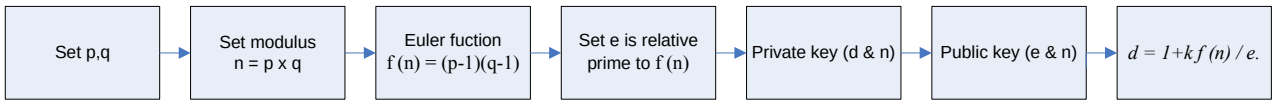
2.2.1 Fungsi *Totient Euler φ*

Fungsi *totient euler* atau biasa disebut dengan fungsi euler merupakan salah satu fungsi yang dipakai dalam perhitungan matematis pada algoritma RSA. Fungsi euler mendefinisikan $\varphi(n)$ untuk $n \geq 1$ yang menyatakan jumlah bilangan bulat positif $< n$ yang relatif prima dengan n. Dua bilangan bulat a dan b dikatakan relatif prima jika gcd(a,b) =1(pembagi bersama terbesar dari a dan b adalah 1)

- Jika $n = pq$ (p dan q bilangan prima), maka $\varphi(n) = \varphi(p)\ \varphi(q) = (p-1)(q-1)$.
- Contoh: $\varphi(15) = \varphi(3)\ \varphi(5) = 2x4 = 8$ buah bilangan bulat yang relatif prima terhadap 15, yaitu 1,2,4,7,8,11,13,14.

2.2.2 Algoritma Pembangkitan Kunci

Dalam proses pembangkitan kunci baik kunci publik maupun kunci privat pada algoritma RSA, dapat dilakukan dengan langkah-langkah sebagai berikut [2] :



Gambar 2.4 : Proses Pembangkitan Kunci Algoritma RSA

Berikut ini akan djelaskan lebih lanjut tentang proses pembangkitan kunci pada algoritma RSA :

1. Pilih dua buah bilangan prima secara random yakni p dan q akan tetapi nilai $p \neq q$.

2	3	5	7	11	13
17	19	23	29	31	37
41	43	47	53	59	61
67	71	73	79	83	.....

Misal nilai p adalah 47 dan q adalah 71, p dan q adalah bilangan prima, akan tetapi nilai $p \neq q$.

2. Hitung n = $p.q$, sehingga nilai n = 47 x 71 adalah 3337
3. Hitung φ(n) = (p-1)(q-1), sehingga nilai φ(n) = (53-1)(71-1) adalah 3220
4. Pilih kunci publik (*e*), sehingga nilai *e* relatif prima terhadap φ(n)

2	3	5	7	11	13
17	19	23	29	31	37
41	43	47	53	59	61
67	71	73	....	3233	

Misalkan ditentukan nilai *e* secara acak yang relatif prima terhadap nilai φ(n) adalah 17. 17 relatif prima terhadap 3220. maka nilai *e* adalah 17.

5. Hitung kunci privat (*d*) dengan menggunakan persamaan $d\ e \equiv 1\ (mod\ \varphi(n))$. Perhatikan bahwa $d\ e \equiv 1\ (mod\ \varphi(n))$ ekivalen dengan $e.d = 1 + k\ \varphi(n)$, sehingga secara sederhana *d* dapat dihitung dengan $d = 1+k\ \varphi(n) / e$. Dengan rumus tersebut maka di dapat nilai d = (1+*k* 3220) / 17. *k*=1,2,3,4,.... Dengan mencoba nilai-nilai k = 1,2,3,..... sehingga diperoleh d yang bulat adalah 1019.

Pada kunci publik terdiri atas:

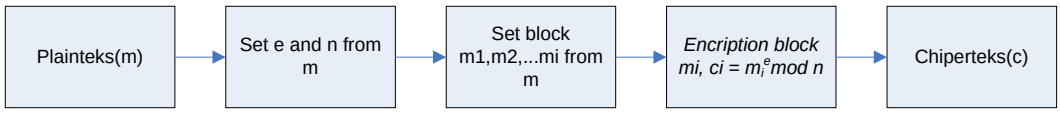
- *n*, modulus yang digunakan.
- *e*, kunci publik, kunci untuk *enkripsi*

Pada kunci privat terdiri atas:

- *n*, modulus yang digunakan.

- *d*, kunci privat, kunci untuk *dekripsi*, yang harus dijaga kerahasiaannya.

2.2.3 Proses enkripsi pesan



Gambar 2.5 : Proses *Enkripsi* Algoritma RSA

Dalam peng*enkripsi*an pesan dalam RSA dapat dimisalkan Yance ingin mengirim sebuah pesan (*m)* ke Helmi. Maka Yance harus melakukan beberapa langkah sebagai berikut:

1. Yance menentukan kunci publik (*e*) dan modulus (*n*) dari pesan terlebih dahulu yakni nilai *e* = 17 dan nilai *n* =3337.
2. Kemudian Yance memecah *plainteks* atau pesan (*m)* menjadi blok-blok *m1,m2,....* sedemikian sehingga setiap blok merepresentasikan nilai di dalam selang [0, n-1]. Misal pesan yang akan dikirimkan adalah HARI INI. Sebelum memecah *plainteks* menjadi blok, Yance mengubah *plainteks* dalam kode ASCII yakni 7265827332737873. Kemudian, Yance memecah *plainteks* menjadi beberapa blok yang lebih kecil dan nilai-nilai m_i harus terletak dalam selang [0,3337-1] agar tranformasi menjadi satu-ke-satu,misal *m* dipecah menjadi 2 blok yang berukuran 3 digit :

   $m_1 = 726$ $m_4 = 273$

   $m_2 = 582$ $m_5 = 787$

   $m_3 = 733$ $m_6 = 003$

3. Setiap blok m_i di*enkripsi* menjadi blok c_i dengan rumus $c_i = m_i^e \ mod \ n$ yakni Helmi memberikan kunci publik ke pada Yance, e = 79 dan n = 3337. Yance melakukan *enkripsi* setiap blok pesan sebagai berikut:

   $726^{79} \bmod 3337 = 215 = c_1$

   $582^{79} \bmod 3337 = 776 = c_2$

   $733^{79} \bmod 3337 = 1743 = c_3$

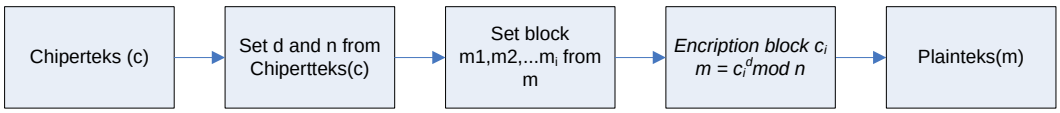
   $273^{79} \bmod 3337 = 933 = c_4$

   $787^{79} \bmod 3337 = 1731 = c_{5c}$

   $003^{79} \bmod 3337 = 158 = c_6$

   Jadi, hasil dari *enkripsi* atau cipherteks yang akan dikirimkan ke Helmi adalah 21577617439331731158. Setelah ter*enkripsi* maka Yance dapat mengirimkan Chiperteks pada Helmi.

2.2.3 Proses dekripsi pesan



Gambar 2.6 : Proses *Enkripsi* Algoritma RSA

Helmi menerima cipherteks (c) dari Yance. Kemudian Helmi melakukan *dekripsi* pesan dari Yance yang masih berupa Chiperteks. Setiap blok cipherteks (c_i) di*dekripsi* kembali menjadi blok m_i dengan rumus $m_i = c_i^d \ mod \ n$[2]. *Dekripsi* dilakukan dengan

menggunakan kunci private d = 1019, kemudian blok–blok chiperteks di*dekripsi*kan sebagai berikut :

$215^{1019} \bmod 3337 = 726 = m_1$

$776^{1019} \bmod 3337 = 582 = m_2$

$1743^{1019} \bmod 3337 = 733 = m_3$

$993^{1019} \bmod 3337 = 273 = m_4$

$1731^{1019} \bmod 3337 = 787 = m_5$

$158^{1019} \bmod 3337 = 003 = m_6$

Blok *plainteks* yang lain dikembalikan dengan cara yang serupa. Akhirnya kita memperoleh kembali *plainteks* semula.

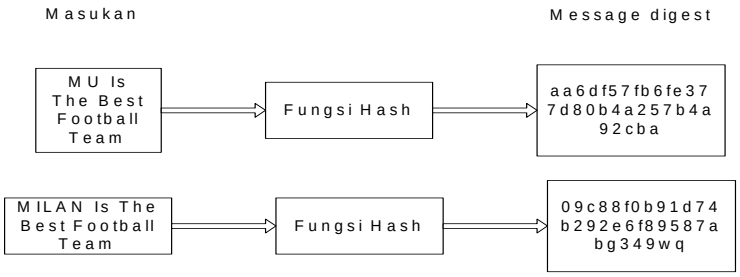
m = 7265827332737873

yang dalam karakter ASCII adalah

m= HARI INI.

2.3 Fungsi Hash Satu-Arah

Didalam sistem Kriptografi, selain ada algortima *enkripsi*/*dekripsi* yang mempunyai tujuan untuk kerahasiaan data, juga ada suatu fungsi yang mempunyai tujuan untuk menjaga integritas dan otentikasi data. Fungsi itu disebut dengan fungsi *hash*. Fungsi *hash* merupakan fungsi yang menerima masukan untuk diubah menjadi keluaran yang mempunyai panjang yang tetap (biasanya lebih pendek dari masukannya)[3]. Keluaran dari fungsi *hash* disebut dengan nilai *hash* atau *message digest*.



Gambar 2.7 : Contoh pesan yang diubah menjadi *message digest* menggunakan fungsi hash

Dari gambar 2.7 tersebut terdapat dua pesan dengan panjang berbeda. Kemudian diproses dengan fungsi hash yang menghasilkan *message digest* dengan panjang yang sama. Fungsi hash "satu arah" adalah jika pesan atau data yang sudah diubah menjadi *message digest* tidak dapat dikembalikan lagi menjadi menjadi pesan semula. Fungsi hast bersifat tidak dirahasiakan, dan keamanannya terletak pada sifat satu arahnya itu[3]. Contoh dari fungsi hash yang telah ada, antara lain : MD2, MD4, MD5, WHIRLPOOL, SHA (Secure hash Function), RIPMEND, dan lain-lain.

2.3.1 Secure Hash Function(SHA)

SHA adalah salah satu dari algoritma dari fungsi *hash* satu-arah(*one-way function*) yang dibuat oleh NSA dan dipublikasikan oleh NIST[4]. SHA merupakan kelanjutan dari pendahulunya MD5, yang sudah tidak aman lagi, karena pada tahun 2005 telah ditemukan kecacatan pada algoritma tersebut yaitu dengan kunci-publik yang berbeda akan menghasilkan *message digest* yang sama. Algoritma SHA ini secara komputasi tidak mungkin menemukan pesan yang berkoresponden dengan *message digest* yang diberikan. Sehingga algoritma ini aman. Ada 6 algortima dari

SHA yang dipublikasikan yaitu SHA-0, SHA-1, SHA-224, SHA-256, SHA-384, dan SHA-512. Berikut adalah tabel dari beberapa algoritma dari SHA[4] :

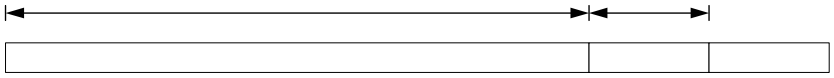
Tabel 2.1 : Beberapa algoritma SHA

Algoritma	**Ukuran *message digest* (bit)**	**Ukuran blok pesan (bit)**	**Ukuran maksimum pesan (bit)**
SHA-0	160	512	$2^{64} - 1$
SHA-1	160	512	$2^{64} - 1$
SHA-256/224	256/224	512	$2^{64} - 1$
SHA-512/384	512/384	1024	$2^{128} - 1$

2.3.2 SHA-1

SHA-1 dipublikasikan oleh NIST pada tahun 1995. Ukuran maksimum masukan pesan 2^{64} bit (2.147.483.648 *gigabyte*) dan menghasilkan *message digest* dengan panjang 160 bit , lebih panjang dari *message digest* MD5 yaitu : 128 bit[8]. Berikut adalah langkah-langkah pembuatan *message digest* dengan SHA-1[3] :

1. Pembuatan *padding bit* (bit-bit pengganjal) dari pesan.

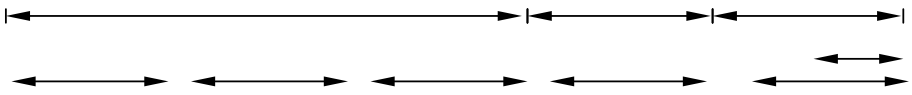


Gambar 2.8 : Pesan yang ditambah dengan Padding Bit

Penambahan *padding bit* pada pesan, mengakibatkan panjang pesan dalam satuan bit kongruen dengan 448 modulo 512. Panjang *padding bit* adalah antara 1 sampai 512. Dan *padding bit* terdiri dari sebuah bit 1 dengan sisanya bit 0.Pesan yang telah diberi *padding bit* selanjutnya ditambah lagi dengan 64

bit yang menyatakan panjang pesan semula, sehingga panjang pesan sekarang menjadi 512.

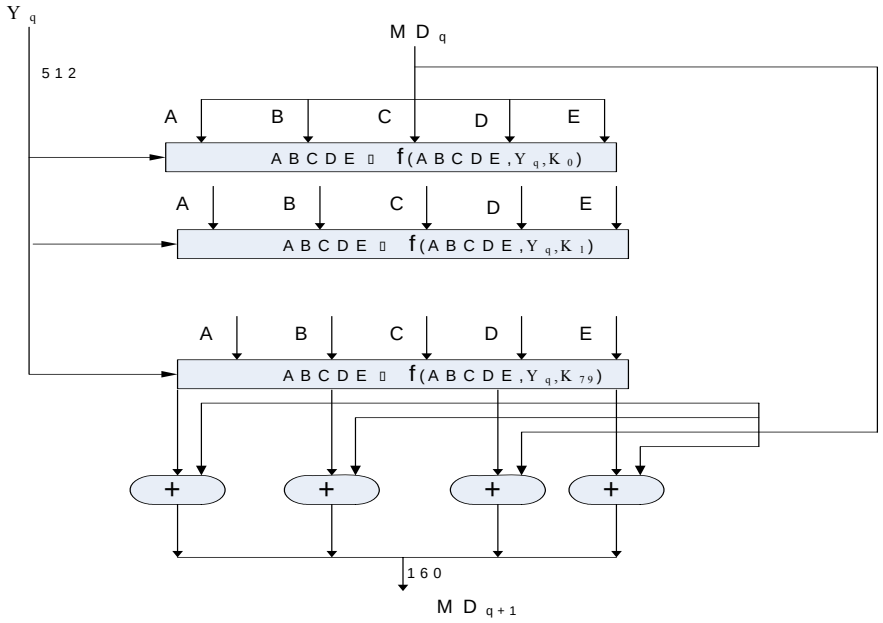
Sebagai contohnya, misalkan ada pesan **abc.** Panjang pesan *L*, 8 x 3 = 24 bit. Padding bits (*K*) dapat dicari dengan $L+1+K \equiv 448 \bmod 512$. K = 448-(24+1) = 423 bit



Gambar 2.9 : Contoh pesan yang ditambah dengan padding bits

2. Menginisialisasi *buffer* (penyangga) MD SHA-1 membutuhkan 5 buah *buffer (penyangga) MD* yang masing-masing panjangnya 32 bit. Sehingga total panjang *buffer (penyangga) MD* adalah 5 x 32=160 bit. Kelima penyangga dalam notasi HEX. A= 67452301, *B*= EFCDAB89, *C* = 98BADCFE, D = 10325476, dan *E* = C3D2E1F0[3].
3. Pengolahan pesan dalam blok

   Pesan dibagi menjadi L buah blok yang masing-masing berukuran 512 bit(Y_0 sampai Y_{l-1}). Setiap blok 512 bit diproses bersama dengan *buffer MD* menjadi keluaran 128 bit. Proses ini disebut debagai proses $H_{sha.}$ Berikut adalah gambar proses H_{sha} [3] :



Gambar 2.10 : Proses pengolahan blok 512 bit (proses H sha)

Pada gambar 2.10, Yq merupakan blok 512 bit ke-q dari pesan yang telah ditambah bit-bit pengganjal. MDq adalah nilai *message digest* 160 bit dari proses H_{sha} ke-q. Pada awal proses, MDq berisi nilai inisialisasi penyangga (*buffer*)MD[3]. Proses H_{sha} terdiri dari 80 putaran, dan masing-masing putaran menggunakan bilangan penambah K_t, yaitu:

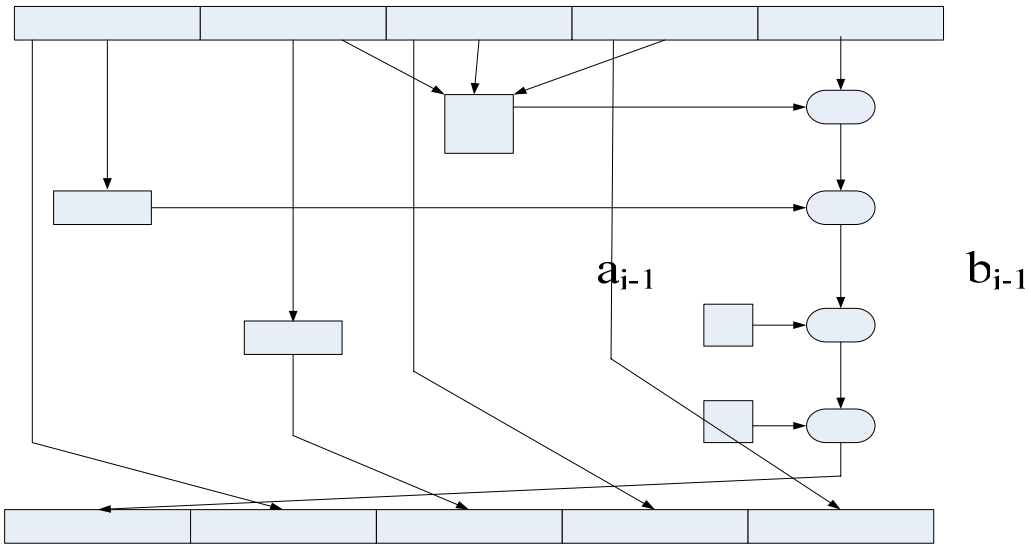
Putaran $0 \leq t \leq 19$ K_t = 5A827999

Putaran $20 \leq t \leq 39$ K_t = 6ED9EBA1

Putaran $40 \leq t \leq 59$ K_t = 8F1BBCDC

Putaran $60 \leq t \leq 79$ K_t = CA62C1D6

Dibawah ini merupakan gambar setiap putaran dari SHA-1.



Gambar 2.11 : Proses satu putaran dari SHA-1

Pada gambar 2.11, operasi dasar dari SHA-1 dapat ditulis sebagai persamaan sebagai berikut:

$a,b,c,d,e \leftarrow (CLS_5(a) + f_t(b,c,d) + e + W_t + K_t),a,CLS_{30}(b),c,d$ ……(1)

keterangan

a,b,c,d,e	=	5 buah penyangga(*buffer*) berukuran 32 bit(A,B,C,D,E)
t	=	putaran,$0 \leq t \leq 79$
f_t	=	fungsi logika
CLS_s	=	*circular left shift* sebanyak *s* bit
W_t	=	*word* 32 bit yang diturunkan dari blok 512 bit yang sedang diproses
K_t	=	konstanta penambah
+	=	operasi penjumlahan dalam modulo 2^{32}

Berikut adalah tabel dari fungsi f_t sebagai fungsi logika yang melakukan operasi *bitwise* untuk setiap putaran[8].

Tabel 2.2 : Fungsi logika f_t pada setiap putaran

Putaran	$f_t(b, c, d)$
0 .. 19	$(b \wedge c) \vee (\sim b \wedge d)$
20 .. 39	$b \oplus c \oplus d$
40 .. 59	$(b \wedge c) \vee (b \wedge d) \vee (c \wedge d)$
60 .. 79	$b \oplus c \oplus d$

Nilai W_1 dan W_{16} berasal dari 16 word pada blok yang sedang diproses, sedangkan nilai W_t berikutnya diperoleh dari persamaan [8]:

$$W_t = W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3} \quad \text{.....(2)}$$

- Setelah putaran ke 79, a, b, c, d, dan e ditambahkan ke A,B,C,D,danE dan selanjutnya algoritma memproses untuk blok data berikutnya (Y_{q+1}). Keluaran akhir dari algoritma SHA adalah hasil penyambungan bit-bit di A, B, C, dan E[4]. Catatan operator logika AND,OR,NOT,XOR masing-masing dilambangkan dengan $\wedge, \vee, \sim, \oplus$.

2.4 Tanda-tangan Digital (*Digital Signature*)

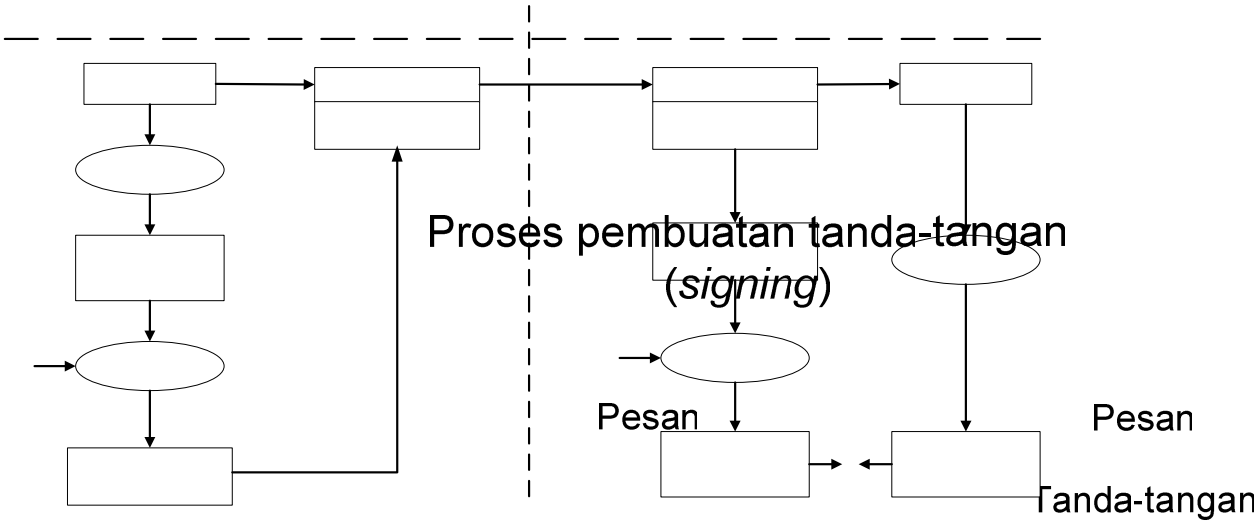
Tanda-tangan digital (*digital signature*) ialah tanda-tangan yang dibubuhkan ke dalam suatu data digital. Tujuan pemberian tanda-tangan tersebut ialah untuk menjaga integritas data, untuk otentikasi data, dan mencegah pihak-pihak yang melakukan komunikasi melakukan penyangkalan terhadap data yang dikirimkan[3].

Salah satu cara untuk menandatangani pesan ialah dengan menggunakan fungsi *hash*. Tanda-tangan diambil dari nilai *hash* atau *message digest* dari pesan. Kemudian nilai *hash* di*enkripsi* menggunakan sistem Kriptografi kunci-asimetri.

Hasil *enkripsi* itulah yang digunakan sebagai tanda-tangan digital. Kemudian tanda-tangan digital digabungkan ke pesan, lalu keduanya dikirimkan lewat saluran komunikasi.

Jika pada sistem Kriptografi kunci-asimetri, untuk meng*enkripsi* pesan menggunakan kunci-publik, sedangkan untuk meng*dekripsi* pesan menggunakan kunci-privat. Maka setiap orang memungkinkan untuk mengetahui kunci-publik sehingga tidak dapat digunakan untuk proses otentikasi pesan[3].

Pada sistem *digital signature* untuk meng*enkripsi* nilai *hash* dari pesan menggunakan kunci-privat, sedangkan untuk mendapatkan nilai *hash* dari pesan yang dikirimkan dengan melakukan *dekripsi* menggunakan kunci-publik. Dengan melakukan pembalikan kunci tersebut maka kerahasiaan dan otentikasi pesan dapat tercapai. Hal ini yang membedakan antara proses *digital signature* dengan Kriptografi kunci-asimetri.



Gambar 2.12 :Tanda-tangan digital yang menggunakan fungsi hash

Berdasarkan gambar 2.12, proses tanda-tangan digital terbagi menjadi dua proses, yaitu: proses pembentukan tanda-tangan digital (*Signing*) dan proses verifikasi tanda-tangan digital (*Verifying*).

2.4.1 Proses pembentukan tanda-tangan digital (*Signing*)

Langkah-langkah dalam pembentukan tanda-tangan digital sebagai berikut:

1. Menghitung *message digest* dari pesan yang akan dikirimkan dengan menggunakan fungsi *hash* satu-arah. Dalam hal ini menggunakan algoritma SHA-1.
2. Meng*enkripsi message digest* dengan menggunakan algoritma Kriptografi kunci-asimetri. Dalam hal ini menggunakan algoritma RSA. Peng*enkripsi*an *message digest* menggunakan kunci-privat(*private key*) si pengirim. Hasil *enkripsi* inilah yang disebut sebagai tanda-tangan digital (*digital signature*).

3. Menggabungkan *digital signature* ke pesan dengan cara menyambungkannya ke bagian belakang pesan, lalu keduanya dikirim lewat saluran komunikasi.

2.4.2 Proses verifikasi tanda-tangan digital (*Verifying*)

Setelah pesan beserta tanda-tangan digitalnya diterima oleh penerima pesan, tanda-tangan digital diverifikasi untuk dibuktikan keasliannya dengan cara berikut:

1. Men*dekripsi* tanda-tangan digital dengan menggunakan kunci-publik(*public key*) si pengirim pesan. Hasil *dekripsi*nya berupa *message digest* (MD) dari pesan semula.
2. Mengubah pesan menjadi menjadi *message digest* (MD') dengan menggunakan fungsi *hash* satu-arah yang sama dengan si pengirim pesan.
3. Mencocokan *message digest* dari pesan. Jika MD' = MD, maka tanda-tangan digital yang diterima terbukti keasliannya/keotentikannya dan benar-benar dikirim oleh pengirim pesan yang benar.

2.5 Serangan Terhadap Kriptografi

Serangan (*attack*) merupakan suatu usaha yang dilakukan oleh seorang kriptoanalis untuk menemukan kunci atau *plainteks* dari chiperteks dari sistem Kriptografi. Kriptanalis akan berusaha untuk menemukan *plainteks* dari chiperteks tanpa memiliki akses kunci yang digunakan dari sistem Kriptografi. Berdasarkan teknik yang digunakan dalam menemukan kunci, serangan dibagi menjadi *exhaustive* atau *brute force attack* dan *analytical attack*[3].

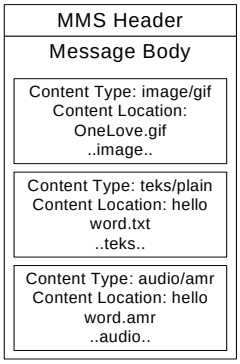
exhaustive atau *brute force attack* merupakan serangan untuk mendapatkan *plainteks* dari *chiperteks* dengan mencoba satu persatu seluruh kombinasi atau kemungkinan kunci yang digunakan. Dalam hal ini kriptanalis harus tahu jenis algoritma Kriptografi yang digunakan oleh pengirim pesan dan juga memiliki sejumlah *plainteks* dan *chiperteks* yang bersesuaian yang digunakan untuk mencari pola dari proses *enkripsi* maupun *dekripsi* dari algoritma Kriptografi yang digunakan.

Analytical attack merupakan serangan untuk mendapatkan *plainteks* dari *chiperteks* dengan menganalisa kelemahan algoritma Kriptografi untuk mengurangi kemungkinan kunci yang tidak mengkin digunakan, sehingga kriptanalisi tidak mencoba semua kemugkinan kunci. Dalam hal ini kriptanalis juga harus tahu jenis algoritma Kriptografi yang digunakan oleh pengirim pesan. Analisis yang digunakan dengan menggunakan pendekatan matematik dan statistic untuk mendapatkan kunci yang digunakan dalam sistem Kriptografi tersebut. Ada beberapa teknik analisis yang digunakan oleh kriptanalis, yaitu *linier cryptanalysis*, *differential cryptanalysis*, dan *integral cryptanalysis* [3].

2.6 MMS *(Multimedia Messaging Service)*

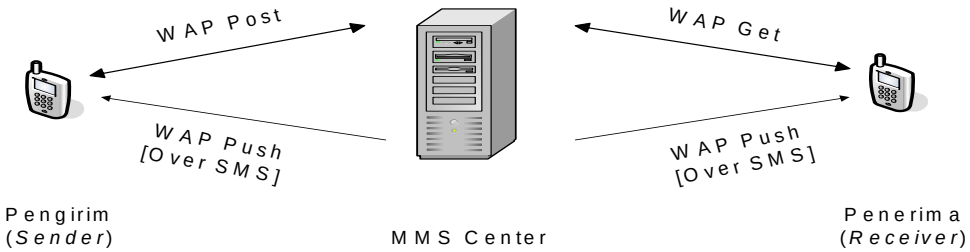
MMS atau *Multimedia Messaging Service* merupakan layanan pengiriman pesan dari satu peralatan ke peralatan lain komunikasi bergerak [1]. Layanan pesan MMS hampir sama dengan SMS yang menyediakan layanan pesan berbentuk teks. Perbedaan mendasar dengan MMS terletak pada jenis atau format pesan yang akan

dikirimkan yaitu berupa gambar dan video, dan chanel yang digunakan yaitu protocol IP [1]. Format pesan MMS yang dikirimkan adalah sebagai berikut [6]:



Gambar 2.13 : Format Data MMS

Teknologi pengiriman pesan MMS merupakan gabungan dari teknologi WAP dan SMS [6]. SMS disini berfungsi sebagai media pemberitahuan informasi tentang terkirimnya sebuah pesan MMS yang berisi data multimedia yang berada pada URL tertentu. Alamat URL harus diakses untuk mendapatkan data multimedia yang telah dikirimkan oleh si pengirim pesan. Data yang dikirim tersebut disimpan didalam MMS Center. Secara umum arsitektur dari MMS dapat dilihat pada gambar dibawah ini [7]:



Gambar 2.14 : Proses Pengiriman Pesan MMS

2.7 Pemrograman Berorientasi Objek

Java adalah sebuah bahasa pemrograman berorientasi objek yang diciptakan oleh Sun Microsystem, sebuah perusahaan besar yang berada di negara Amerika Serikat. Bahasa ini berkembang sangat pesat terutama untuk web-programming. Dalam beberapa tahun terakhir, Java telah merambah dunia mobile dengan J2ME (Micro Edition : MIDlet, dipakai dalam mobile-phone, PDA, smart-phone dan sebagainya).

2.7.1 Kelas dan Objek

Dalam paradigma pemrograman berorientasi objek dikenal kelas dan objek. Kelas merupakan *blue print* dari objek-objek yang akan dibuat. Analogi kelas dan objek seperti *rancangan* model rumah dan *pembangunan* rumah-rumah, adapun proses pembuatan objek dari kelas dikenal dengan *instantiasi.* Sedangkan objek merupakan representasi nyata dari sebuah objek. Dalam *object-oriented programming* (OOP) dengan Java, tidak mendefenisikan objek secara langsung melainkan bekerja dengan kelas dari sebuah objek.

2.7.2 Pewarisan (*Inheritance*)

Salah satu kelebihan pemrograman berorientasi objek adalah penggunaan ulang kode-kode yang telah dibuat. Pewarisan adalah salah satu cara untuk menggunakan kode-kode yang telah dibuat sebelumnya.

Untuk membuat suatu kelas dapat kita turunkan dengan pewarisan field-field dan metode pada kelas dasarnya. Dalam Java tidak dikenal pewarisan ganda, sehingga digunakan interface.

2.7.3 Polimorfisme (*Polymorphism*)

Salah satu pilar Pemrograman Berorientasi Objek yang lain adalah polymorfisme yaitu kemampuan beberapa objek bertipe sama bereaksi secara berbeda terhadap *"*pesan*"* yang sama. Dalam pemrograman berorientasi obek Java, karakteristik ini berlaku untuk suatu tingkah laku (metode).

2.7.4 Pengkapsulan (*Encapsulation*)

Pilar lain dari Pemrograman Berorientasi Objek adalah pengkapsulan, dimana pengembang software dapat menyembunyikan detail suatu objek. Hak akses publik memungkinkan semua kelas mengaksesnya, hak akses protected hanya diberikan kepada kelasnya sendiri dan turunannya, serta kelas-kelas dalam satu paket. sedangkan private hanya boleh diakses oleh kelasnya sendiri.

2.8 J2ME

J2ME merupakan salah satu bagian dari teknologi java yang dikembangkan oleh Sun Microsystem untuk dapat menjalankan program java pada perangkat-perangkat bergerak atau mobile device semacam handphone, Palm, PDA, dan PocketPC, yang memiliki karakteristik berbeda dengan sebuah komputer biasa,

misalnya kecilnya jumlah memori pada handphone dan PDA [3]. J2ME terdiri atas komponen-komponen sebagai berikut :

a. Java Virtual Machine (JVM)

   Komponen ini untuk menjalankan aplikasiaplikasi Java pada emulator atau mobile device.

b. Java API (Application Programming Interface)

   Komponen ini merupakan kumpulan library untuk menjalankan dan mengembangkan program Java pada mobile device.

c. Tools lain untuk pengembangan aplikasi Java, semacam emulator Java phone, emulator Motorolla, Siemens, dan Nokia.

2.8.1 J2ME Configuration

J2ME cofiguration mendefinisikan lingkungan kerja J2ME runtime. Ada dua kategori J2ME Configuration, yakni :

a. CLDC (Connected Limited Device Configuration)

   Kategori ini umumnya digunakan untuk aplikasi Java pada handphone semacam Nokia, Samsung Java phone, Motorola, PDA semacam PALM, Pocket PC, dan two way pager.. Umumnya, perangkat-perangkat tersebut hanya memiliki memori berukuran 160-1024 KiloBytes.

b. CDC (Connected Device Configuration)

   Kategori ini umumnya digunakan untuk aplikasi Java pada perangkat-perangkat dengan ukuran memori paling tidak 2 Megabytes.

   Berikut adalah perbandingan antara CLDC dan CDC :

CLDC	CDC
Mengimplementasikan sebagian dari J2SE	Mengimplementasikan seluruh fitur J2SE
JVM yang di gunakan adalah KVM	JVM yang digunakan adalah CVM
Digunakan pada perangkat genggam (handphone, PDA, twoway pager) dengan memori terbatas (160-512KB)	Digunakan pada perangkat genggam(internet tv, Nokia Communikator, car TV)dengan memori minimal 2MB
Processor : 16/32 bit	Processor : 32 bit

2.8.2 J2ME Profile

Jika J2ME *Configuration* menyediakan library library Java untuk implementasi fitur-fitur standar (kemampuan menelpon ke suatu nomor *handphone* lain) dari sebuah *Handheld device*, J2ME *Profile* menyediakan implementasi-implementasi tambahan yang sangat spesifik dari sebuah *handheld device.* Ada lima kategori J2ME *Profile*, yakni :

a. Mobile Information Device Profile (MIDP)

   MIDP menyediakan librari-librari Java untuk implementasi dasar antar muka (GUI), implementasi jaringan (networking), *database*, dan *timer.* MIDP dirancang khusus untuk *wireless phone* dan *pager*.

b. Foundation Profile

   Profil yang digunakan untuk konfigurasi CDC. Profil ini menamabahkan beberapa kelas dari J2SE kedalam konfigurasi CDC, dan berperan juga sebagai pondasi untuk membentuk profile yang baru lainnya.

c. Personal Profile

Profile yang mendefinisikan ulang personal java sebagai profil yang dapat digunakan sebagai profil dalam J2ME. Profil ini merupakan hasil perluasan dari *Foundation profile.*

d. Personal Digital Assistance Profile

Profil untuk PDA yang memperluas fungsi-fungsi pada konfigurasi CDLC dan digunakan untuk khusus untuk menambahkan kemampuan-kemampuan lebih apabila dibandingkan dengan penggunaan profile MIDP.

e. RMI Profile

Profil yang menambahkan dukungan RMI(*Remote method Invocation*) ke dalam konfigurasi CDC.

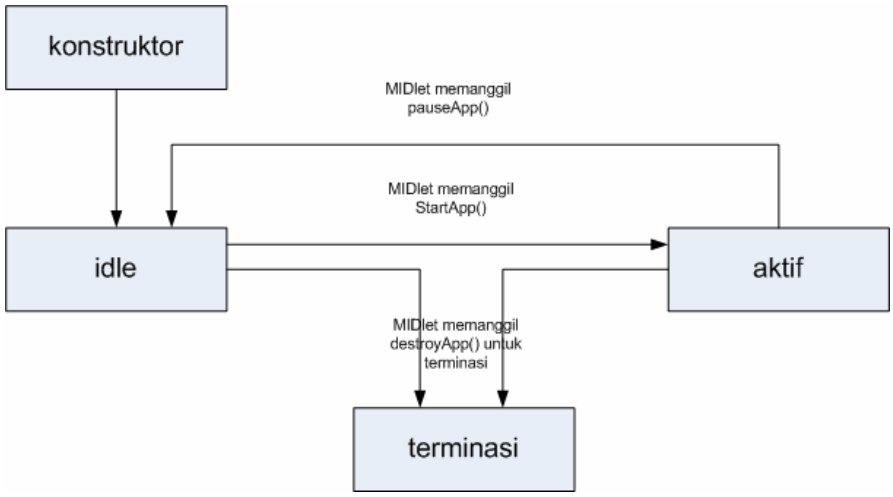
2.8.3 Midlet

Aplikasi yang berjalan di atas alat yang mendukung MIDP (Mobile Information Device Apllet) disebut MIDlet. Pengaplikasian API ditentukan oleh interaksi antara aplikasi dan MIDlet serta bagaimana MIDlet ini dikontrol. Penanganan eksekusi aplikasi dilakukan oleh Java Application Manager (JAM). JAM mengatur instalasi, eksekusi, manajemen versi, dan penghilangan MIDlet. Kendali yang diperbolehkan antara lain adalah :

- *User interface* (input/output)
- *Persistent storage* (simple record-oriented database)
- *Networking* (HTTP,WAP)

- *Timers* (PersonalInformation Management).

MIDlet berupa sebuah kelas abstrak yang merupakan sub kelas dari bentuk dasar aplikasi sehingga antarmuka antara aplikasi J2ME dan aplikasi manajemen pada perangkat dapat terbentuk.



Gambar 2.15 : Alur Hidup MIDlet

Status Idle terjadi ketika MIDlet selesai diinisialisasi dan tidak melakukan aksi apapun. Status Idle juga bisa terjadi ketika dari status aktif kemudian berhasil memanggil fungsi MIDlet.pauseApp() atau MIDlet.notifyPaused() serta ketika akan start() terjadi kesalahan berupa exception MIDlet State Change Exception. Status aktif terjadi ketika MIDlet sedang aktif atau berjalan dengan normal yakni setelah MIDlet memanggil fungis MIDlet.startApp(). Sedangkan status terminasi terjadi ketika MIDLet berhenti berjalan atau MIDlet memanggil fungsi MIDlet.destroyApp().

JAD (Java Application Descriptor) digunakan untuk mendeskripsikan isi aplikasi untuk keperluan pemetaan. File JAD berisi deskripsi file JAR (*Java Archive*) dan pemetaan atribut MIDlet, sedangkan file JAR berisi kumpulan kelas dan *resource*. File JAR ini yang nantinya akan diinstalkan pada handphone yang berbasis Java untuk menjalankan aplikasi.

2.9 *Unified Modeling Language* (UML)

UML adalah bahasa grafis untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak. UML berorientasi objek, menerapkan banyak level abstraksi, tidak bergantung proses pengembangan, tidak bergantung bahasa dan teknologi, pemaduan beberapa notasi di beragam metodelogi, usaha bersama dari banyak pihak, didukung oleh kakas – kakas yang diintegrasikan lewat XML. Standar UML dikelola oleh OMG (*Object Management Group*).

UML adalah bahasa pemodelan untuk menspesifikasikan, memvisualisasikan, membangun dan mendokumentasikan artifak – artifak dari sistem.

UML bukanlah :

1. Bahasa pemrograman visual, tapi bahasa pemodelan visual.
2. Spesifikasi kakas, tetapi spesifikasi bahasa pemodelan.
3. Proses, tetapi yang memungkinkan proses – proses.

2.9.1 Tujuan UML

Tujuan utama perancangan menggunakan UML adalah :

1. Menyediakan bahasa pemodelan visual yang ekspresif dan siap pakai untuk mengembangkan dan pertukaran model – model yang berarti.
2. Menyediakan mekanisme perluasan dan spesifikasi untuk memperluas konsep – konsep inti.
3. Mendukung spesifikasi independen bahasa pemrograman dan proses pengembangan tertentu.
4. Menyediakan basis formal untuk pemahaman bahasa pemodelan.
5. Mendorong pertumbuhan pasar kakas berorientasi objek.
6. Mendukung konsep–konsep pengembangan level lebih tinggi seperti komponen, kolaborasi, *framework* dan *pattern*.

2.9.2 Diagram dan Teknik Pemodelan UML

Diagram mengemukakan banyak hal, penggunaan notasi yang terdefenisi baik dan ekspresif adalah penting pada proses pengembangan perangkat lunak, yaitu :

1. Notasi standar memungkinkan pengembang mendeskripsikan skenario atau rumusan arsitektur dan kemudian mengkomunikasikan secara tidak ambigu.
2. Notasi yang bagus membebaskan otak untuk berkosentrasi pada masalah – masalah yang lebih lanjut.
3. Notasi yang baik memungkinkan mengeleminasi keperluan pemeriksaan konsistensi dan kebenaran keputusan dengan menggunakan *tool* terotomatisasi.

2.9.2.1 Diagram Struktur

Diagram ini untuk memvisualisasi, menspesifikasikan, membangun dan mendokumentasikan aspek statik dari sistem. Diagram struktur pada UML terdiri dari:

1. Diagram Kelas (*Class diagram*)

   Diagram ini menunjukkan sekumpulan kelas, *interface* dan kolaborasi serta keterhubungannya. Diagram kelas ditujukan untuk pandangan statik terhadap sistem.

2. Diagram Objek (*Objek diagram*)

   Diagram ini menunjukkan sekumpulan objek dan keterhubungannya. Diagram ini menunjukkan potongan statik dari instan – instan yang ada pada diagram kelas. Diagram ini memperlihatkan satu prototipe atau kasus tertentu yang mungkin terjadi. Diagram objek menyediakan notasi grafis formal guna memodelkan objek, kelas dan saling keterhubungan. Diagram objek berguna untuk *abstract modelling* dan perancangan program – program sesungguhnya.

3. Diagram Komponen (*Component diagram*)

   Diagram ini menunjukkan organisasi dan kebergantungan diantara sekumpulan komponen. Diagram ini merupakan pandangan statik terhadap implementasi sistem.

4. Diagram Pengembangan (*Deployment diagram*)

   Diagram ini menunjukkan konfigurasi pemrosesan saat jalan dan komponen–komponen yang terdapat didallamnya. Diagram ini merupakan pandangan

statik dari arsitektur. Pilihan model dan diagram yang digunakan dipengaruhi oleh bagaimana persoalan ditangani dan bagaimana solusi dibentuk.

2.9.2.2 Diagram Perilaku

Diagram ini untuk memvisualiasi, menspesifikasi, membangun dan mendokumentasikan aspek dinamis dari sistem. Diagram perilaku pada UML terdiri dari :

1. Diagram Use case (*Use case diagram*)

   Diagram ini menunjukkan sekumpulan kasus fungsional dan aktor (jenis kelas khusus) dan keterhubungannya.

2. Diagram Sekuen (*Sequen diagram*)

   Diagram ini menunjukkan interaksi yang terjadi antar objek. Diagram ini merupakan pandangan dinamis terhadap sistem. Diagram ini menekankan pada basis keberurutan waktu dari pesan – pesan yang terjadi.

3. Diagram Kolaborasi (*Collaboration diagram*)

   Diagram ini merupakan diagram interaksi. Diagram ini menekankan pada organisasi struktur dari objek – objek yang mengirim dan menerima pesan.

4. Diagram *Statechart* (*Statechart diagram*)

   Diagram ini berisi *state*, transisi, kejadian dan aktivitas. *Statechart* merupakan pandangan dinamis dari sistem. Diagram ini penting dalam memodelkan perilaku antarmuka, kelas, kolaborasi dan menekankan pada urutan kejadian. Penting untuk sistem reaktif yang dipicu kejadian di dunia nyata.

5. Diagram Aktivitas (*Activity diagram*)

Diagram ini untuk menunjukkan aliran aktivitas di sistem. Diagram ini adalah pandangan dinamis terhadap sistem. Diagram ini penting untuk memodelkan fungsi sistem dan menekankan pada aliran kendali diantara objek–objek.

2.9.3 Notasi dalam UML

UML menyediakan beberapa notasi dan artifak standar yang bisa digunakan sebagai alat komunikasi bagi para pelaku dalam proses analisis dan desain. Artifak di dalam UML didefenisikan sebagai informsi dalam berbagai bentuk yang digunakan atau dihasilkan dalam proses pengembangan perangkat lunak. Yang perlu diperhatikan untuk menjaga konsisten antar artifak, selama proses analisis dan desain adalah bahwa setiap perubahan yang terjadi pada satu artifak harus juga dilakukan pada artifak sebelumnya.

1. Aktor

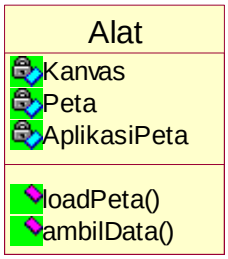
   Aktor adalah segala sesuatu yang berinteraksi dengan sistem aplikasi komputer. Jadi aktor dapat berupa manusia, perangkat keras, atau mungkin objek lain dalam sistem yang sama. Biasanya yang dilakukan aktor adalah memberikan informasi pada sistem dan atau memerintahkan sistem untuk melakukan sesuatu. Notasi Aktor dapat dilihat pada gambar 2.16 di bawah ini.



Gambar 2.16: Notasi Aktor

2. Kelas

Kelas merupakan pembentuk utama dari sistem berorientasi objek karena kelas menunjukkan kumpulan objek yang memiliki atribut dan operasi yang sama. Kelas digunakan untuk mengabstraksikan elemen – elemen dari sistem yang sedang dibangun. Kelas bisa untuk merepresentasikan baik perangkat lunak maupun perangkat keras, baik konsep maupun benda nyata.



Gambar 2.17: Notasi Kelas

3. *Interface*

Interface merupakan kumpulan operasi tanpa implementasi dari suatu kelas. Implementasi operasi dalam *interface* dijabarkan oleh operasi dalam kelas. Oleh karena itu keberadaan *interface* selalu disertai oleh kelas yang mengimplementasikan operasinya. *Interface* ini merupakan salah satu cara mewujudkan prinsip enkapsulasi dalam objek. Notasi *Interface* dapat dilihat pada gambar 2.18 di bawah ini.



Gambar 2.18: Notasi *Interface*

4. *Use case*

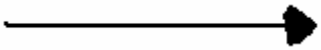
Use – case menjelaskan urutan kegiatan yang dilakukan aktor dan sistem untuk mencapai suatu tujuan tertentu walaupun menjelaskan kegiatan. Namun hanya menjelaskan "apa" yang dilakukan oleh aktor dan sistem, bukan "bagaimana" aktor dan sistem melakukan kegiatan tersebut. Notasi *Use – case* dapat dilihat pada gambar 2.19 di bawah ini.



Gambar 2.19: Notasi *Use case*

5. Interaksi

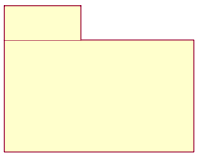
Interaksi digunakan untuk menunjukkan baik aliran pesan atau informasi antar objek maupun hubungan antar objek. Biasanya interaksi ini dilengkapi juga dengan teks bernama *operation signature* yang tersusun dari nama operasi, parameter yang dikirim dan tipe parameter yang dikembalikan. Notasi Interkasi dapat dilihat pada gambar 2.20 di bawah ini.



Gambar 2.20: Notasi Interaksi

6. Paket

Paket adalah kontainer atau wadah konseptual yang digunakan untuk mengelompokkan elemen – elemen dari sistem yang sedang dibangun, sehingga bisa dibuat model yang lebih sederhana. Tujuannya adalah untuk mempermudah penglihatan (*visibility*) dari model yang sedang dibangun. Notasi Paket dapat dilihat pada gambar 2.21 di bawah ini.



Gambar 2.21 : Notasi Paket

7. Catatan (*Note*)

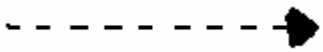
   Note digunakan untuk memberikan keterangan dan komentar tambahan dari satu elemen sehingga bisa langsung terlampir dalam model. Catatan ini bisa ditampilkan ke semua elemen notasi yang lain. Notasi *Note* dapat dilihat pada gambar 2.22 di bawah ini.



Gambar 2.22 : Notasi *Note*

8. Ketergantungan (*Depedency*)

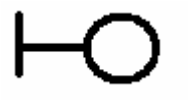
   Ketergantungan merupakan relasi yang menunjukkan bahwa perubahan pada salah satu elemen memberi pengaruh pada elemen lain. Elemen yang ada di bagian tanda panah adalah elemen yang tergantung pada elemen yang ada di bagian tanpa tanda panah.



Gambar 2.23 : Notasi *Depedency*

9. *Boundry Class*

   Boundry Class adalah kelas yang menghubngkan *user* dengan sistem. Oleh karena itu sering disebut juga sebagai *user interface class*. Notasi *Boundry Class* dapat dilihat pada gambar 2.24 di bawah ini.



Gambar 2.24 : Notasi *Boundry Class*

10. *Control Class*

Control class adalah kelas yang mengkoordinasi aktivitas dalam sistem. Kelas ini menghubungkan *boundry class* dengan entitas kelas. Notasi *Control Class* dapat dilihat pada gambar 2.25 di bawah ini.



Gambar 2.25 : Notasi *Control Class*

11. *Entity Class*

Entity Class adalah kelas yang menghubungkan dengan data atau infrmasi yang digunakan oleh sistem. *Entity Class* ini adalah kelas yang menyimpan dan mengelola data. Notasi *Entity Class* dapat dilihat pada gambar 2.26 di bawah ini.



Gambar 2.26 : Notasi *Entity Class*