# BAB I

# PENDAHULUAN

## 1.1 Latar Belakang

Penggunaan *Multimedia Message Service* (MMS) pada telepon seluler menjadi salah satu pilihan dalam melakukan pertukaran data. MMS merupakan salah satu aplikasi pada telepon seluler yang dapat mengirimkan *file* berupa gambar, *audio*, *video*, bahkan *Microsoft Office file*. *Microsoft Office* adalah aplikasi perkantoran yang dirilis oleh *Microsoft Corporation* dalam membantu pengoperasian dan penyelesaian pekerjaan sehari-hari serta memberikan hasil yang optimal. *Microsoft Office file* diantaranya terdiri dari *Microsoft Office Word, Microsoft Office Excel*, *Microsoft Office Power Point, Microsoft Office Access, Microsoft Office Publisher, Microsoft Office Visio*.

Permasalahannya, celah keamanan terbesar pada pertukaran data melalui MMS yaitu pesan yang dikirimkan akan disimpan di *Multimedia Messaging Service Center* (MMSC) dan *operator* atau orang yang mempunyai hak akses penuh dapat melihat *file* yang dikirimkan (Hius, 2013). Hal ini sangat rentan sekali terhadap kebocoran data. Masalah tersebut menjadi sebuah kekhawatiran bagi pengguna MMS ketika *file* yang dikirimkan adalah *file* yang sangat penting yang hanya boleh diketahui oleh orang-orang tertentu saja.



Dengan alasan tersebut butuh metode yang dapat mengamankan data MMS agar tidak diketahui oleh orang yang tidak berhak yaitu salah satunya dengan kriptografi. Kriptografi merupakan salah satu solusi untuk membatasi wewenang tersebut, dengan aplikasi kriptografi yang ditanamkan pada telepon seluler, maka pengguna dapat memastikan MMS yang ingin dikirim tak dapat dibaca oleh orang lain kecuali pengguna yang diijinkan.

Kriptografi yang dipakai menggunakan algoritma *hybrid* yaitu dalam penelitian ini menggabungkan antara algoritma kunci simetris *Blowfish* yang memakai satu kunci

mempunyai kelebihan dalam kecepatan operasi, dan algoritma kunci asimetris *Rivest-Shamir Adleman* (RSA) yang memakai dua kunci yaitu kunci publik dan kunci privat mempunyai tingkat keamanan yang lebih tinggi. Jadi dengan algoritma *Hybrid*, pengguna MMS dapat mengamankan data lebih cepat dan lebih tinggi keamanannya dibandingkan dengan hanya satu algoritma saja.

Berdasarkan latar belakang tersebut yang menunjukkan bahwa algoritma RSA dan *Blowfish* dapat mengamankan data MMS pada *Microsoft Office File* maka disusunlah penelitian ini dengan judul "**Aplikasi Keamanan Data *Multimedia Message Service* (MMS) pada *Microsoft Office File* Memanfaatkan Algoritma *Rivest-Shamir Adleman* (RSA) dan *Blowfish* Berbasis Android**". Secara gambarannya, penelitian ini membuat aplikasi yang dapat digunakan untuk mengamankan data *Microsoft Office File* yang akan dikirimkan melalui MMS. Algoritma RSA digunakan untuk untuk tujuan pertukaran kunci. Sedangkan algoritma *Blowfish* digunakan untuk mengenkripsi dan mendekripsi *file*. *File* yang dikirim melalui MMS menjadi lebih aman setelah diubah ke dalam data terkunci. Karena *file* yang dikirimkan bukan *file* asli melainkan *file* yang sudah didekripsi dan hanya dapat dibaca oleh pihak yang berhak.



## 1.2 Identifikasi Masalah

Berdasarkan situasi dan kondisi yang tercermin pada latar belakang, menjelaskan sebuah pokok permasalahan yaitu celah keamanan pada proses pertukaran data melalui MMS sangat besar. Hal ini dikarenakan pesan yang kita kirimkan akan terlebih dahulu disimpan pada MMS *Center* yang menyebabkan duplikasi data yang tidak dapat diketahui tingkat keamanan data tersebut.

## 1.3 Perumusan Masalah

Perumusan masalah dalam penelitian ini adalah:

1. Bagaimana membuat aplikasi yang dapat mengimplementasikan kriptografi algoritma RSA dan *Blowfish* untuk keamanan data MMS pada *Microsoft Office File*?
2. Bagaimana *file* hasil enkripsi dapat didekripsi sehingga *file* akan kembali seperti semula?

## 1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah:

1. Membuat aplikasi yang dapat digunakan untuk keamanan data MMS pada *Microsoft Office File* dengan mengimplementasikan kriptografi algoritma RSA dan *Blowfish*.
2. Dapat mengenkripsi *file* menjadi *chipertext file* dan mendekripsinya sehingga menjadi *file* semula.

## 1.5 Batasan Masalah

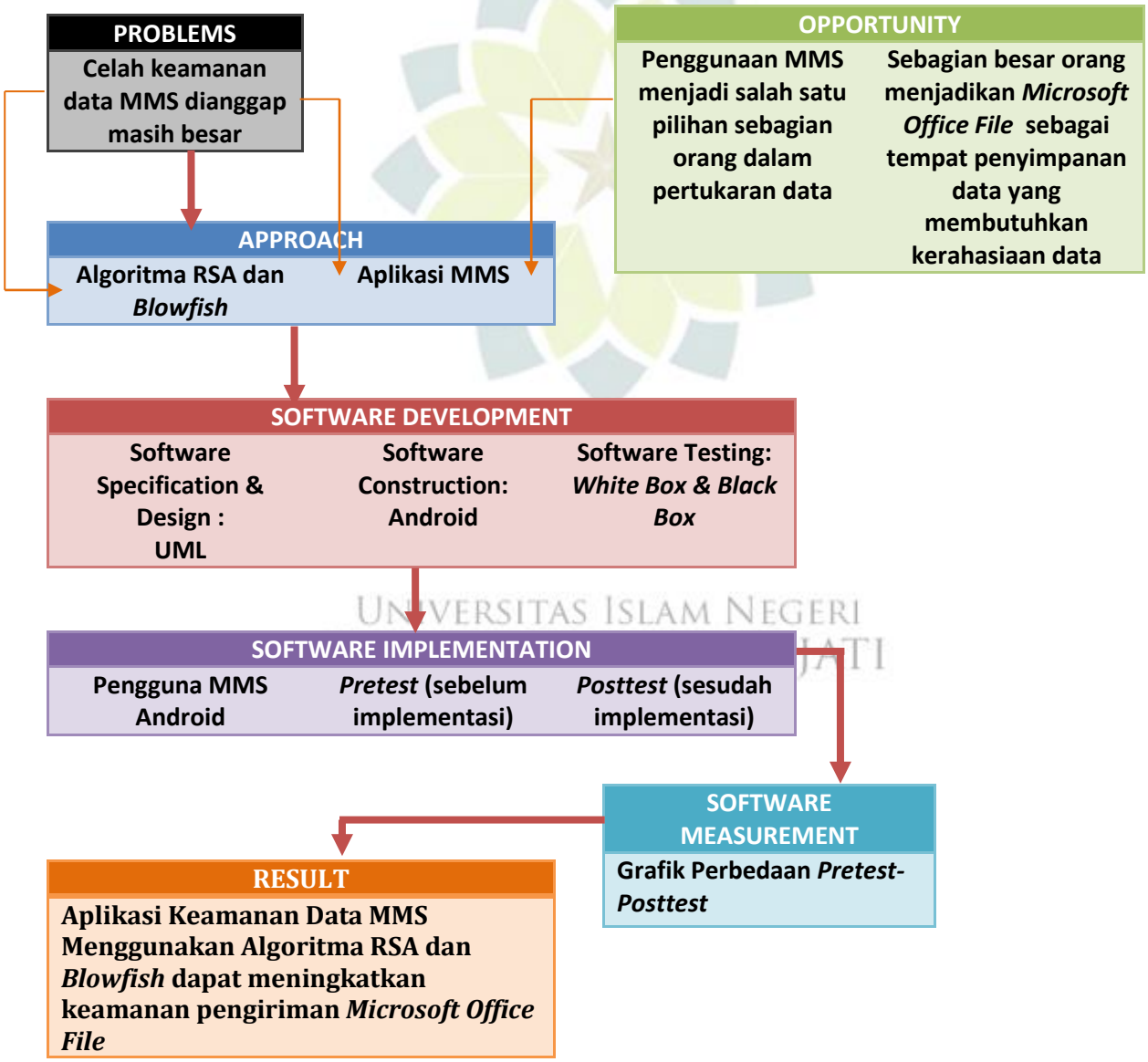
Seperti yang telah dijelaskan pada rumusan masalah merupakan masalah yang masih cukup luas. Oleh sebab itu, penelitian ini dibatasi sebagai berikut:

1. Algoritma yang digunakan adalah algoritma *hybrid* yang merupakan kombinasi dari algoritma RSA dan *Blowfish*.
2. Data (*plaintext*) yang akan diproses oleh algoritma *Blowfish* berbentuk *Microsoft Office Word*, *Microsoft Office Excel*, *Microsoft Office Access*, dan *Microsoft Power Point* .
3. Aplikasi digunakan minimal pada sistem operasi Android 4.1.2.
4. Pengiriman dan penerimaan *file* dilakukan oleh aplikasi *Go SMS Pro* yang sudah ter-*install* pada sistem *smartphone* tersebut.
5. Aplikasi hanya mengirimkan *chipertext* hasil dari enkripsi kunci yang digunakan untuk mengenkripsi *file*.

## 1.6 Kerangka Pemikiran

Kerangka pemikiran adalah dasar pemikiran dari penelitian yang disintesiskan dari fakta-fakta, observasi dan telaah kepustakaan. Oleh karena itu kerangka pemikiran memuat teori, dalil atau konsep-konsep yang akan dijadikan dasar penelitian. Uraian dalam kerangka pemikiran menjelaskan hubungan dan keterkaitan antar variabel.

Kerangka yang akan dijadikan sebuah pemikiran untuk proses penyusunan kajian terlihat pada Gambar 1.1.




Gambar 1.1 Kerangka Pemikiran

## 1.7 *State of The Art*

*State of The Art* dimaksudkan untuk menganalisis penelitian sebelumnya yang pernah ada, yang sejalan dan mempunyai konsep yang hampir sama dengan penelitian saat ini. Lalu melihat sejauh mana perbedaan masing–masing penelitian, sehingga masing–masing penelitian mempunyai tema yang original.

Penelitian Hius (2013) bertujuan untuk menganalisa algoritma berdasarkan kecepatan proses enkripsi dan dekripsi, pemakaian memori dan integritas data yang dienkripsi dan mengimplementasikan algoritma ECKBA pada aplikasi MMS menggunakan J2ME.

Selanjutnya, Sitinjak, dkk. (2010) bertujuan untuk mengamankan data ataupun informasi yang tersimpan dalam bentuk *file* dengan menggunakan algoritma *Blowfish* dan mengimplementasikan aplikasi pada *Visual Basic 6.0*.

Selanjutnya, penelitian yang dilakukan oleh Tanto dan Ricco bertujuan untuk merancang perangkat lunak yang dapat mengenkripsi dan mendekripsi *file* dengan menggunakan algoritma *hybrid* kombinasi dari algoritma RSA dan RC4 pada *Visual Basic*.

Selanjutnya, penelitian yang dilakukan oleh Tambunan (2010) bertujuan untuk mengatasi masalah keamanan dokumen pada *Microsoft Office* dengan menggunakan algoritma *Blowfish* yang diimplementasikan ke dalam *Microsoft Visual Basic 6.0* menjadi sebuah model kriptosistem berbasis *desktop*.



Hasil penelitian dari para peneliti yang sudah disebutkan tersebut sudah terlihat gambaran aplikasi yang dibuat dan akan sangat membantu dalam menentukan tema yang original. Dapat diambil ikhtisar dari penelitian-penelitian tersebut dan membandingkannya dengan penelitian yang akan dilakukan, ditampilkan pada Tabel 1.1.

Tabel 1.1 *State of The Art*

| No | Nama | Metode | Teknologi | Hasil |
|---|---|---|---|---|
| 1 | Hius (2013) | Algoritma ECKBA | Aplikasi MMS Mobile pada J2ME | Aplikasi yang dapat mengamankan *image*, *audio*, dan *video* |
| 2 | Sitinjak, dkk (2010) | Algoritma *Blowfish* | Aplikasi *Desktop* pada *Visual Basic 6.0* | Aplikasi desktop yang dapat mengamankan dokumen-dokumen multimedia, *Microsoft Office*, dan arsip |
| 3 | Tanto dan Ricco | Algoritma RSA dan RC4 | Aplikasi *Desktop* pada *Visual Basic* | Aplikasi *desktop* yang dapat mengamankan dokumen *Microsoft Office*, *audio*, dan *image* |
| 4 | Tambunan (2010) | Algoritma *Blowfish* | Aplikasi *Desktop* pada *Visual Basic 6.0* | Aplikasi *desktop* yang dapat mengamankan dokumen-dokumen *Microsoft Office* |
| 5 | Abdul Aziz (2014) | Algoritma RSA dan *Blowfish* | Aplikasi SMS pada Sistem Operasi Android | Aplikasi android yang dapat mengamankan dan mempertukarkan dokumen *Microsoft Office* |

## 1.8 Metodologi Penelitian

Guna mendapatkan data yang diperlukan untuk membantu dalam penelitian yang akan dilakukan, maka digunakan metodologi sebagai berikut:

1. Pengumpulan Data

Pengumpulan data ini dimaksudkan untuk mencari bahan dalam melakukan penelitian dengan beberapa metode pengumpulan data berikut.



a) Metode Kepustakaan

Metode ini digunakan untuk mendapatkan konsep-konsep teoritis dengan cara menganalisa data pada literatur (pustaka) dan media lain yang dapat membantu dalam pemecahan masalah.

b) Metode Survey

Metode ini diperlukan untuk mengetahui sampai sejauh mana tingkat kebutuhan pengguna pada aplikasi yang akan dibuat ini. Dengan metode ini juga akan diketahui fitur-fitur apa saja yang dibutuhkan dan diinginkan oleh pengguna.

2. Metode Pengembangan Perangkat Lunak

Dalam mengembangkan aplikasi keamanan data MMS pada *Microsoft Office File* dengan algoritma *hybrid* ini menggunakan metode *prototype.* Dengan metode *prototype* ini pengembang dan pelanggan dapat saling berinteraksi selama proses pembuatan sistem. Kunci agar metode *prototype* ini berhasil dengan baik adalah dengan mendefinisikan aturan-aturan main pada saat awal, yaitu pelanggan dan pengembang harus setuju bahwa *prototype* dibangun untuk mendefinisikan kebutuhan.

Tahapan-tahapan dalam metode *Prototype* adalah sebagai berikut:

a) Pengumpulan kebutuhan

Pelanggan dan pengembang bersama-sama mendefinisikan *format* seluruh perangkat lunak, mengidentifikasikan semua kebutuhan, dan garis besar sistem yang akan dibuat.

b) Membangun *prototyping*

Membangun *prototyping* dengan membuat perancangan sementara yang berfokus pada penyajian kepada pelanggan. Perancangan sementara berupa rancangan perangkat lunak dengan menggunakan *Unified Modelling Language* (UML).

c) Evaluasi *prototyping*

Evaluasi ini dilakukan oleh pelanggan apakah *prototyping* yang sudah dibangun sudah sesuai dengan keinginann pelanggan. Jika sudah sesuai maka langkah d akan diambil. Jika tidak *prototyping* direvisi dengan mengulangu langkah a, b, dan c.

d) Mengkodekan sistem

Dalam tahap ini, *prototyping* yang sudah di sepakati diterjemahkan ke dalam bahasa pemrograman java.

e) Menguji sistem

Setelah sistem sudah menjadi suatu perangkat lunak yang siap pakai, harus dites dahulu sebelum digunakan. Pengujian ini dilakukan dengan *White Box* dan *Black Box Testing*.

f) Evaluasi Sistem

Pelanggan mengevaluasi apakah sistem yang sudah jadi sesuai dengan yang diharapkan. Jika ya, langkah g dilakukan; jika tidak, ulangi langkah c dan e.

g) Menggunakan sistem

Perangkat lunak yang telah diuji dan diterima pelanggan siap untuk digunakan.

**1.9 Sistematika Penulisan**

BAB I PENDAHULUAN

Bab I menguraikan latar belakang, perumusan masalah yang merumuskan berbagai masalah yang diteliti secara lebih jelas, tujuan penelitian yang berisi tentang tujuan dilakukannya penelitian, manfaat penelitian, batasan masalah untuk memberikan batasan yang tegas dan jelas serta sistematika penulisan yang menguraikan urutan penyajian yang digunakan dalam penyusunan skripsi ini.

BAB II LANDASAN TEORI

Bab II membahas tentang landasan teori dari topik penulisan skripsi secara mendalam beserta dengan referensinya.

BAB III ANALISIS DAN PERANCANGAN

Bab III akan menguraikan hasil analisis dan perancangan aplikasi yang akan dibangun.

BAB IV IMPLEMENTASI

Bab IV akan menguraikan implementasi aplikasi yang telah dianalisa dan dirancang sebelumnya.

BAB V PENUTUP

Bab V berisi uraian tentang kesimpulan dan saran terhadap aplikasi yang hendak dibangun dan dikembangkan lebih lanjut.

DAFTAR PUSTAKA

Daftar Pustaka berisi semua sumber tertulis (buku, artikel jurnal, dokumen resmi, atau sumber-sumber lain dari internet) atau tercetak (CD, *video*, *film* atau kaset) yang pernah dikutip dan digunakan dalam proses penyusunan.

LAMPIRAN-LAMPIRAN

Berisi semua dokumen yang digunakan dalam proses penyusunan dan perancangan seperti *source code*, kelengkapan dokumen dan lain sebagainya.

# BAB II

# LANDASAN TEORI

## 2.1 Aplikasi

Menurut Ali Zaki dan SmitDev Community (2007:11) "Program aplikasi adalah komponen yang berguna melakukan pengolahan data maupun kegiatan-kegiatan seperti pembuatan dokumen atau pengolahan data".

Program aplikasi berjalan diatas sistem operasi, sehingga agar program aplikasi bisa diaktifkan, perlu melakukan instalasi sistem operasi terlebih dahulu.

Contoh program aplikasi yang lazim terdapat di perkantoran adalah *Office Suite*. Misalnya *Microsoft Office* dan *Open Office*, yang terdiri dari paket pengolah kata, (*MS Word, Open Office Writer*); angka (*MS Excel, Open Office Calc*); presentasi (*MS PowerPoint, Open Office Impress*); dan database (*MS Access, Open Office Base*).

Selain aplikasi perkantoran, ada beberapa aplikasi lain yang bisa digunakan di komputer. Perangkat lunak grafis: untuk melakukan penglahan gambar. Perangkat lunak jaringan: membuat komputer bisa digunakan untuk keperluan jaringan. Perangkat lunak desain web dan *browser*: untuk membuat dan melihat halaman web. Perangkat lunak internet: untuk memanfaatkan berbagai layanan internet. Perangkat lunak komunikasi: memungkinkan memanfaatkan komputer untuk menghubungi orang lain. Perangkat lunak *utility*: untuk meperbaiki atau merawat komputer, terutama sistem operasinya. Perangkat lunak hiburan dan pendidikan: untuk bersenang-senang dan belajar. Perangkat lunak pembuat multimedia: untuk membuat atau mengolah *video* dan/atau *film*. Perangkat lunak pemrograman: untuk membuat perangkat lunak di komputer.

## 2.2 Keamanan Data

Menurut Dony Ariyus (2008:10),

> Keamanan data pada lalu-lintas jaringan adalah suatu hal yang diinginkan semua orang untuk menjaga privasi, supaya data yang dikirim aman dari gangguan orang yang tidak bertanggung-jawab, yang disembunyikan menggunakan algoritma kriptografi.

Pada dasarnya komponen kriptografi terdiri dari beberapa komponen, seperti (Dony Ariyus, 2008):

1. *Enkripsi*: merupakan hal yang sangat penting dalam kriptografi, merupakan cara pengamanan data yang dikirimkan sehingga terjaga kerahasiaannya. Pesan asli disebut *plaintext* (teks-biasa), yang diubah menjadi kode-kode yang tidak dimengerti. Enkripsi bisa diartikan dengan *cipher* atau kode. Sama halnya dengan tidak mengerti sebuah sebuah kata maka kita akan melihatnya di dalam kamus atau daftar istilah. Beda halnya dengan enkripsi, untuk mengubah teks-biasa ke bentuk teks-kode kita gunakan algoritma yang dapat mengkodekan data yang kita ingini.
2. *Dekripsi*: merupakan kebalikan dari enkripsi. Pesan yang telah dienkripsi dikembalikan ke bentuk asalnya. Algoritma yang digunakan untuk dekripsi tentu berbeda dengan yang digunakan untuk enkripsi.
3. *Kunci*: adalah kunci yang dipakai untuk melakukan enkripsi dan dekripsi. Kunci terbagi menjadi dua bagian, yaitu kunci rahasia (*private key*) dan kunci umum (*public key*).
4. *Chipertext*: merupakan suatu pesan yang telah melalui proses enkripsi. Pesan yang ada pada teks-kode ini tidak bisa dibaca karena berupa karakter-karakter yang tidak mempunyai makna (arti).

5. *Plaintext*: sering disebut *cleartext*. Teks-asli atau teks-biasa ini merupakan pesan yang ditulis atau diketik yang memiliki makna. Teks-asli inilah yang diproses menggunakan algoritma kriptografi untuk menjadi *chipertext* (teks-kode).
6. *Pesan*: dapat berupa data atau informasi yang dikirim (melalui kurir, saluran komunikasi data) atau yang disimpan di dalam media perekaman (kertas, *storage*).
7. *Cryptanalysis*: kriptanalisis bisa diartikan sebagai analisis kode atau suatu ilmu untuk mendapatkan teks-asli tanpa harus mengetahui kunci yang sah secara wajar. Jika suatu teks-kode berhasil diubah menjadi teks-asli tanpa menggunkan kunci yang sah, proses tersebut dinamakan *breaking code*. Hal ini dilakukan oleh para kriptanalis. Analisis kode juga dapat menemukan kelemahan dari suatu algoritma kriptografi dan akhirnya dapat menemukan kunci atau teks-asli dari teks-kode yang dienkripsi dengan algoritma tertentu.

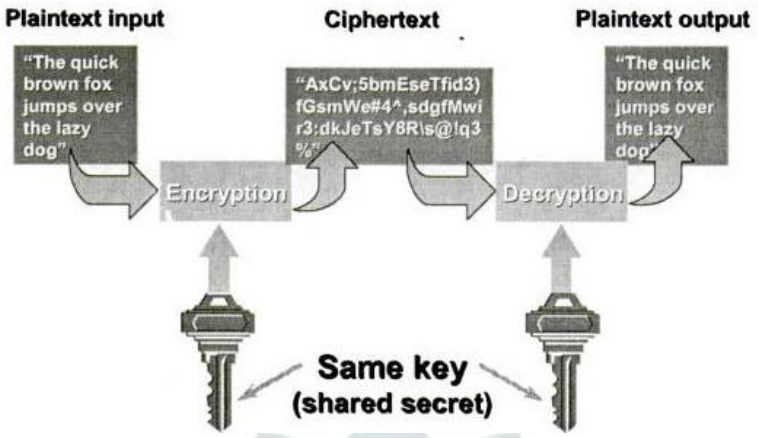
## 2.3 Algoritma Kriptografi Modern

Kriptografi modern merupakan suatu perbaikan yang mengacu pada kriptografi klasik. Pada kriptografi modern terdapat berbagai macam algoritma yang dimaksudkan untuk mengamankan informasi yang dikirim melalui jaringan komputer. Algoritma kriptografi modern terdiri dari dua bagian (Dony Ariyus, 2008):



### 2.3.1 Algoritma Simetris

Algoritma simetris adalah algoritma yang menggunakan kunci yang sama untuk enkripsi dan dekripsinya. Contoh: Alice ingin mengirim pesan x dengan aman menggunakan saluran umum kepada Bob. Alice menggunakan kunci $x\sigma$ yang sebelumnya telah disepakati antara Alice dan Bob. Untuk mengirim pesan e $x\sigma$ (x) kepada Bob, dia akan mendekripsi teks-kode yang diterima dengan kunci yang sama dengan yang digunkan untuk memperoleh akses ke pesan yang diterima, begitu sebaliknya. Secara gambaran algoritma simetris terlihat pada Gambar 2.1.




Gambar 2.1 Algoritma Simetris
Sumber: Buku Pengantar Ilmu Kriptografi, 2008

Aplikasi dari algoritma simetris digunakan oleh beberapa algoritma di bawah ini:

a) *Data Encryption Standard* (DES),

b) *Advance Encryption Standard* (AES),

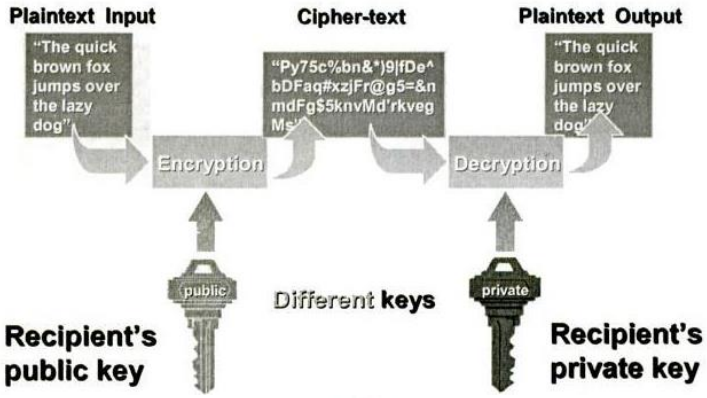
c) *International Data Encryption Algorithm* (IDEA),

d) A5,

e) RC4.

### 2.3.2 Algoritma Asimetris

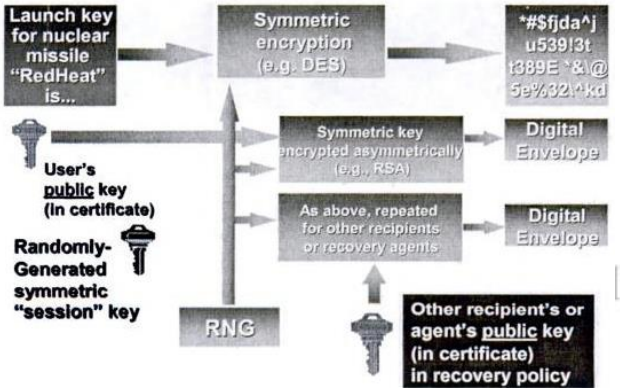
Algoritma asimetris adalah pasangan kunci kriptografi yang salah satunya digunakan untuk proses enkripsi dan yang satu lagi untuk dekripsi. Semua orang yang mendapatkan kunci publik dapat menggunakannya untuk mengenkripsi suatu pesan, sedangkan hanya satu orang saja yang memiliki rahasia itu, yang dalam hal ini kunci rahasia, untuk melakukan pembongkaran terhadap kode yang dikirim untuknya. Contoh algoritma terkenal yang menggunakan kunci asimetris adalah RSA (merupakan singatan dari nama penemunya, yakni Rivest, Shamir dan Adleman). Secara gambarannya algoritma asimetris terlihat pada Gambar 2.2.



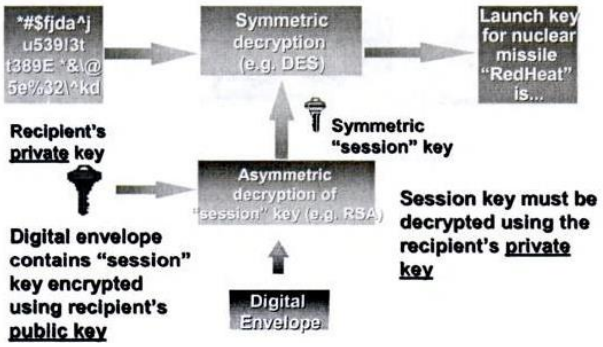

Gambar 2.2 Algoritma Asimetris
Sumber: Buku Pengantar Ilmu Kriptografi, 2008

### 2.3.3 Algoritma Hibrida

Algoritma hibrida adalah algoritma yang memanfaatkan dua tingkatan kunci, yaitu kunci rahasia (simetri) – yang disebut juga *session key* (kunci sesi) – untuk enkripsi data dan pasangan kuncirahasia-kuncipublik untuk pemberian tanda tangan digital serta melindungi kunci simetri. Secara gambarannya enkripsi dan dekripsi hibrida terlihat pada Gambar 2.3 dan Gambar 2.4.




Gambar 2.3 Enkripsi Hibrida
Sumber: Buku Pengantar Ilmu Kriptografi, 2008

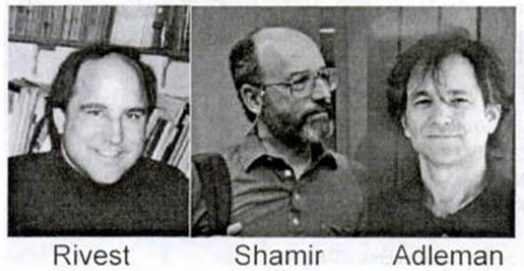



Gambar 2.4 Dekripsi Hibrida
Sumber: Buku Pengantar Ilmu Kriptografi, 2008

## 2.4 *Rivest-Shamir Adleman* (RSA)

Dari sekian banyak algoritma kriptografi kunci-publik yang pernah dibuat, algoritma yang paling populer adalah algoritma RSA. Algoritma ini melakukan pemfaktoran bilangan yang sangat besar. Oleh karena alasan tersebut RSA dianggap aman. Untuk membangkitkan dua kunci, dipilih dua bilangan prima acak yang besar. Algoritma RSA dibuat oleh 3 orang peneliti dari *Massachussets Institute of Technology* (MIT) pada tahun 1976, yaitu: Ron (R)ivest, Adi (S)hamir, dan Leonard (A)dleman. RSA mengekripsikan teks-asli yang dienkripsi menjadi blok-blok yang mana setiap blok memiliki nilai bilangan biner yang diberi symbol "n", blok teks-asli "M" dan blok teks-kode "C", numeric yang lebih kecil daripada "n" (data biner dengan pangkat terbesar). Jika bilangan prima yang panjangnya 200 digit, dapat ditambah beberapa bit 0 di kiri bilangan untuk menjaga agar pesan tetap kurang dari nilai "n". Pembuat algoritma RSA ini terlihat pada Gambar 2.5.






Gambar 2.5 Pembuat Algoritma RSA

Sumber: Buku Pengantar Ilmu Kriptografi, 2008

Besaran yang digunakan pada algoritma RSA:

1. $p$ dan $q$ bilangan prima (rahasia)
2. $r = p \cdot q$ (tidak rahasia)
3. $\phi(r) = (p\text{-}1)(q\text{-}1)$ (rahasia)
4. $PK$ (kunci enkripsi) (tidak rahasia)
5. $SK$ (kunci dekripsi) (rahasia)
6. $X$ (teks-asli) (rahasia)
7. $Y$ (teks-kode) (tidak rahasia)

Algoritma RSA didasarkan pada teorema Euler yang menyatakan bahwa:

$a^{\phi(r)} = 1 \pmod r$ ……(2.1)

yang dalam hal ini:

1. $a$ harus relatif prima terhadap $r$.
2. $\phi(r) = r(1 - 1/p_1)(1 - 1/p_2) \ldots (1 - 1/p_n)$, yang dalam hal ini $p_1, p_2, \ldots, p_n$ adalah faktor prima dari $r$.

   $\phi(r)$ adalah fungsi yang menentukan berapa banyak bilangan 1, 2, 3, …, $r$ yang relatif prima terhadap $r$.

Berdasarkan sifat $a^m \equiv b^m \pmod r$ untuk $m$ bilangan bulat $\geq 1$ maka persamaan (2.1) dapat ditulis menjadi:

$a^{m\phi(r)} \equiv 1^m \pmod r$ ..……...……...……(2.2)

Bila $a$ diganti dengan $X$ maka persamaan (2.2) menjadi:

$X^{m\phi(r)} \equiv 1 \pmod r$ ...…...……...……...…...……(2.3)

Berdasarkan sifat $ac \equiv bc \pmod r$ maka bila persamaan (2.3) dikali dengan $X$ menjadi:

$X^{m\phi(r)-1} \equiv X \pmod r$ ...…...……...……....……(2.4)

yang dalam hal ini $X$ relatif prima terhadap $r$.

Misalkan $SK$ dan $PK$ dipilih sedemikian rupa sehingga

$$SK \cdot PK \equiv (\text{mod } \phi(r)) \quad \ldots\ldots(2.5)$$

atau

$$SK \cdot PK \equiv m\phi(r) + 1 \quad \ldots\ldots(2.6)$$

Sulihkan (2.6) ke dalam persamaan (2.4) menjadi:

$$X^{SK \cdot PK} \equiv X \text{ (mod } r) \quad \ldots\ldots(2.7)$$

Persamaan (2.7) dapat ditulis kembali menjadi:

$$(X^{PK)SK} \equiv X \text{ (mod } r) \quad \ldots\ldots(2.8)$$

yang artinya perpangkatan $X$ dengan $PK$ diikuti dengan perpangkatan dengan $SK$ menghasilkan $X$ semula.

Berdasarkan persamaan (2.8) maka enkripsi dan dekripsi dirumuskan sebagai berikut:

$$E_{PK}(X) = Y \equiv X^{PK} \text{ mod } r \quad \ldots\ldots(2.9)$$

$$D_{SK}(Y) = X = Y^{SK} \text{ mod } r \quad \ldots\ldots(2.10)$$

karena $SK \cdot PK = PK \cdot SK$, maka enkripsi diikuti dengan dekripsi ekuivalen dengan dekripsi diikuti enkripsi:

$$E_{SK}(D_{SK}(X)) = D_{SK}(E_{PK}(X)) \equiv X^{PK} \text{ mod } r \quad \ldots\ldots(2.11)$$

oleh karena $X^{PK}$ mod $r \equiv (X + mr)^{PK}$ mod $r$ untuk sembarang bilangan bulat m, maka tiap teks-asli $X$, $X + r$, $X + 2r$, … menghasilkan teks-kode yang sama. Dengan kata lain, transformasinya adalah dari banyak ke satu. Agar transformasinya satu ke satu maka $X$ harus dibatasi dalam himpunan $\{0, 1, 2, \ldots, r - 1\}$ sehingga enkripsi dan dekripsi tetap benar seperti pada persamaan (2.9) dan (2.10).

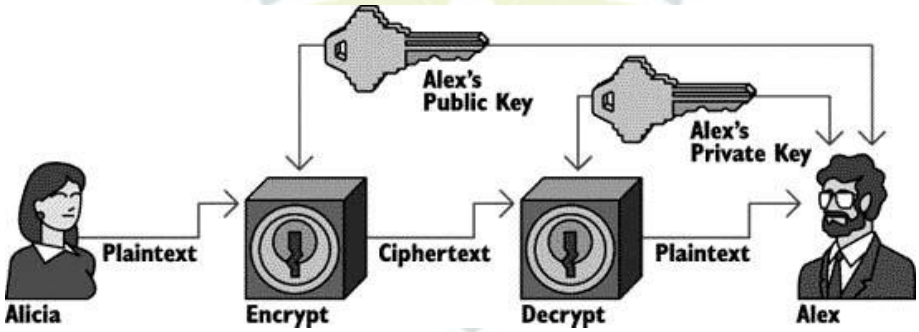
Untuk melakukan enkripsi RSA, teks-asli disusun menjadi blok *x1*, *x2*, …, sedemikian sehingga setiap blok merepresentasikan nilai di dalam rentang 0 sampai *r* – 1. Setiap blok *xi* dienkripsi menjadi blok *yi* dengan rumus:

$C = M^e \bmod n$……....(2.12)

untuk melakukan dekripsi terhadap teks-kode yang menggunakan algoritma RSA, setiap blok teks-kode *yi* didekripsi kembali menjadi blok *xi* dengan rumus:

$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$…….(2.13)

Proses enkripsi dan dekripsi RSA terlihat pada Gambar 2.6.




Gambar 2.6 Proses Enkripsi dan Dekripsi RSA
Sumber: http://labsky2012.blogspot.com, 2012

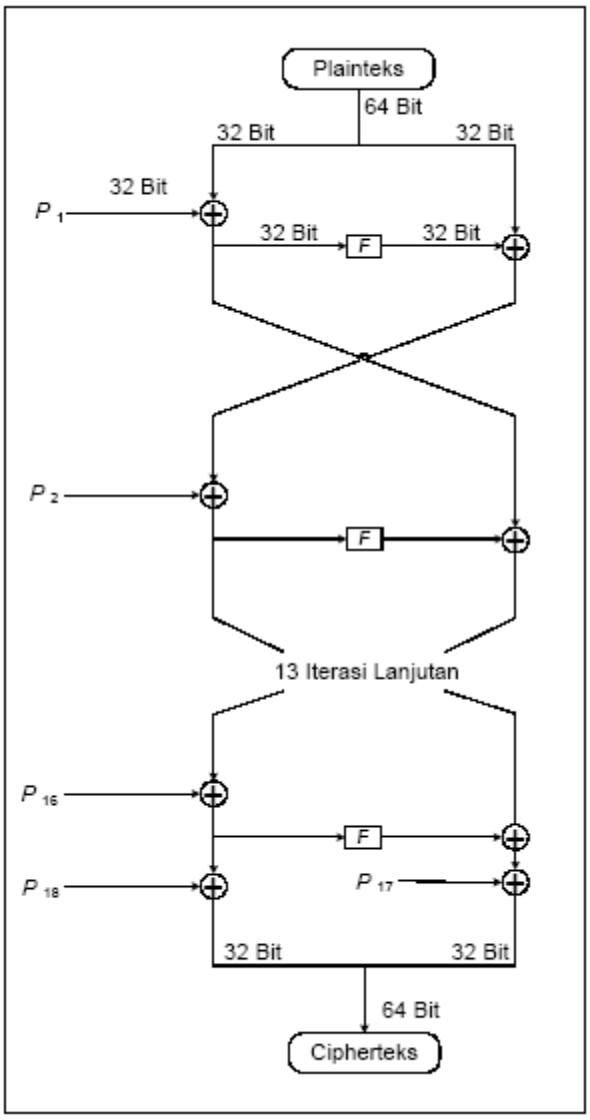
## 2.5 *Blowfish*

*Blowfish* merupakan sebuah algoritma kunci simetri blok kode yang dirancang pada tahun 1993 oleh Bruce Chneier untuk mengganti DES. Pada saat itu banyak sekali rancangan algoritma yang ditawarkan, namun hampir semua terhalang oleh paten atau kerahasiaan pemerintah Amerika. Schneier menyatakan bahwa *Blowfish* bebas paten dan akan diletakkan pada domain publik. Dengan pernyataan Schneier tersebut *Blowfish* telah mendapatkan tempat di dunia kriptografi, khususnya bagi masyarakat yang membutuhkan lgoritma kriptografi yang cepat, kuat, dan tidak terhalang oleh lisensi.

Keberhasilan *Blowfish* dalam menembus pasar terbukti dengan diadopsinya Blowfish sebagai *Open Cryptography Interface* (OCI) pada kernel Linux versi 2.5 ke atas. Dengan diadopsinya *Blowfish* berarti dunia *open source* menganggap *Blowfish* adalah salah satu algoritma terbaik. Kesuksesan *Blowfish* mulai memudar setelah kehadiran algoritma dengan ukuran blok yang lebih besar seperti AES. AES sendiri memang dirancang untuk menggatikan DES, sehingga secara keseluruhan AES lebih unggul dari DES dan juga *Blowfish*.

*Blowfish* adalah algoritma kriptografi kunci simetri blok kode dengan panjang blok tetap 64 bit. *Blowfish* menerapkan teknik kunci berukuran sembarang. Ukuran kunci yang dapat diterima oleh *Blowfish* adalah antara 32 bit hingga 448 bit, dengan ukuran *default* sebesar 128 bit. *Blowfish* memanfaatkan teknik pemanipulasiaan bit dan teknik pemutaran ulang dan pergiliran kunci yang dilakukan sebanyak 16 kali. Algoritma utama terbagi menjadi dua subalgoritma utama, yaitu bagian ekspansi kunci dan bagian enkripsi-dekripsi data.

Pengekspansian kunci dilakukan pada saat awal dengan masukan sebuah kunci dengan panjang 32 bit hingga 448 bit, dan keluaran adalah sebuah larik upa-kunci dengan total 4168 byte. Bagian enkripsi-dekripsi data terjadi dengan memanfaatkan perulangan 16 kali terhadap jaringan *Feistel*. Setiap perulangan terdiri dari permutasi dengan masukan kunci dan substitusi data. Semua operasi dilakukan dengan memanfaatkan *operator* XOR dan *operator* penambahan. Penambahan dilakukan terhadap empat larik *lookup* yang dilakukan setiap putarannya. Proses enkripsi pada *Blowfish* ini terlihat pada Gambar 2.7.






Gambar 2.7 Proses Enkripsi pada *Blowfish*
Sumber: Buku Pengantar Ilmu Kriptografi, 2008

Kunci *blowfish* dibangkitkan menggunakan upa-kunci berukuran besar. Kunsi tersebut harus dikomputasikan pada saat awal, sebelum pengkomputasian enkripsi dan dekripsi data. Langkah-langkahnya adalah sebagai berikut:

1. Terdapat kotak permutasi (P-box) yang teridiri dari 18 buah 32 bit upa-kunci : P1, P2, P3, …, P18. P-box ini telah ditetapkan sejak awal, 4 buah P-box awal sebagai berikut:

   P1 = 0x243f6a88

P2 = 0x85a308d3

P3 = 0x13198a2e

P4 = 0x03707344

2. XOR-kan P1 dengan 32 bit awal kunci, XOR-kan P2 dengan 32 bit berikutnya dari kunci, dan teruskan hingga seluruh panjang kunci telah ter-XOR-kan (kemungkinan sampai P14, 14x32 = 448, panjang maksimal kunci).
3. Terdapat 64 bit dengan isi kosong. Bit-bit tersebut dimasukkan ke langkah 2.
4. Gantikan P1 dan P2 dengan keluaran dari langkah 3.
5. Enkripsikan kembali keluaran langkah 3 dengan langkah 2, namun kali ini dengan upa-kunci yang berbeda (sebab langkah 2 menghasilkan upa-kunci baru).
6. Gantikan P3 dan P4 dengan keluaran dari langkah 5.
7. Lakukan seterusnya hingga seluruh P-box teracak sempurna.
8. Total keseluruhan terdapat 521 iterasi untuk menghasilkan seuruh upa-kunci yang dibutuhkan. Aplikasi hendaknya menyimpannya daripada harus membuat ulang seluruh upa-kunci tersebut.

Adapun proses enkripsi-dekripsi data pada algoritma *Blowfish* adalah sebagai berikut:

1. Masukan dari proses ini adalah data 64 bit yang diinisialkan "x".
2. Bagi x menjadi 2 buah bagian sama besar, xL (x kiri) sepanjang 32 bit, dan xR (x kanan) sepanjang 32 bit.
3. Lakukan iterasi sebanyak i=1 hingga i=16:

   xL = xL XOR P[i];

   xL = F(xL) XOR xR;

   Swap(xL, xR);

4. Fungsi F yaitu sebagai berikut: Bagi xL menjadi 4 buah 8 bit a, b, c, dan d.

   $F(xL) = ((S1,a + S2,b \bmod 2^{32}) \text{ XOR } S3,c) + S4,d \bmod 2^{32}$.

5. Langkah terakhir adalah:

   Swap(xL, xR);

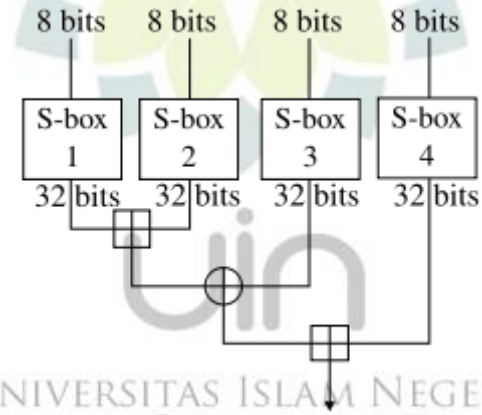
   xR = xR XOR P[17];

   xL = xL XOR P[18];

   Gabungkan xL dan xR menjadi 64 bit return hasil gabungan.

6. Pada proses dekripsi langkah-langkahnya sama persis dengan proses enkripsi, hanya saja P-box digunakan dengan urutan terbalik.

   Untuk fungsi jaringan *feistel* terlihat pada Gambar 2.8.



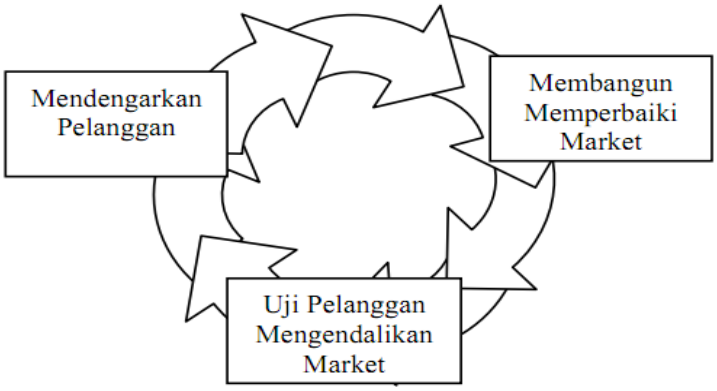



Gambar 2.8 Fungsi Jaringan *Feistel*
Sumber: Buku Pengantar Ilmu Kriptografi, 2008

Walaupun *Blowfish* dengan ukuran bloknya sepanjang 64 bit terasa sudah cukup kecil, namun terdapat kebutuhan untuk sebuah algoritma kriptografi yang lebih sederhana. Kebutuhan ini dirasakan oleh masyarakat yang hanya membutuhkan kriptografi untuk kepentingan sederhana, hanya untuk merahasiakan sesuatu yang tidak cukup penting. Untuk itulah dirancang sebuah *Blowfish* mini. *Blowfish* mini ini mempunyai blok sepanjang 32 bit, hanya saja algoritma utamanya sama persis dengan *blowfish*.

## 2.6 *Prototype*

Metode pengembangan sistem yang digunakan yaitu dengan menggunakan metode *prototype*. *Prototype* paradigma dimulai dengan mengumpulkan kebutuhan. Pengembangan dan pelanggan bertemu dan mendefinisikan keseluruhan sistem yang akan dibuat, mengidentifikasi segala kebutuhan yang diketahui, kemudian melakukan perancangan (Pressman, 2002). Secara gambarannya *prototype* paradigma terlihat pada Gambar 2.9.




Gambar 2.9 *Prototype* Paradigma
Sumber: Buku Rekaya Perangkat Lunak, 2002

*Prototype* bisa berfungsi sebagai sistem yang pertama, memang benar bahwa baik pelanggan maupun pengembang menyukai paradigma *prototype*. Para pemakai merasa nyaman dengan sistem aktual, sedangkan pengembang sistem membangunnya dengan segera. Tetapi *prototiping* bisa juga terjadi masalah karena alasan-alasan sebagai berikut.

1. Pelanggan melihat apa yang tampak tanpa melihat bahwa prototipe itu dijalin bersama-sama "*dengan permen karet dan baling ware*", tanpa melihat bahwa di dalam permintaan untuk membuatnya bekerja, kita belum mencantumkan kualitas perangkat lunak secara keseluruhan atau kemampuan pemeliharaan untuk jangka panjang.
2. Pengembang sering membuat kompromi-kompromi implementasi untuk membuat prototipe bekerja dengan cepat. sistem operasi atau bahasa pemrograman yang tidak sesuai bisa

dipakai secara sederhanakarena mungkin diperoleh dan dikenal; algoritma yang tidak efisien dan sederhana bisa diimplementasikan untuk mendemonstrasikan kemampuan. Setelah selang waktu tertentu, pengembang mungkin mengenali pilihan-pilihan tersebut dan melupakan semua alasan mengapa mereka tidak cocok. Meskipun berbagai masalah bisa terjadi, *prototype* bisa menjadi paradigma yang efektif. Kuncinya adalah mendefinisikan aturan-aturan main sejak awal; yaitu pelanggan dan pengembang keduanya harus setuju bahwa *prototype* dibangun untuk berfungsi sebagai mekanisme pendefinisian kebutuhan.

## 2.7 *Unified Modelling Language* (UML)

*Unified Modelling Language* (UML) adalah sebuah "bahasa" yang telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem (Grady Booch dkk, 1999).

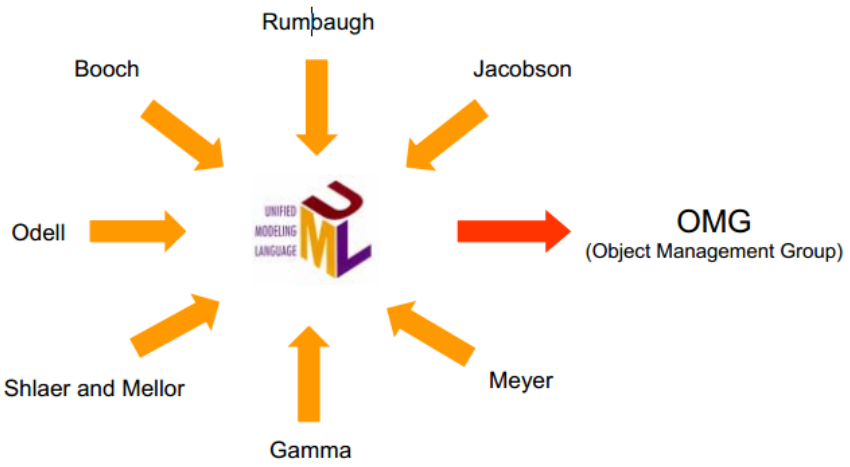
Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan operation dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa-bahasa berorientasi objek seperti C++, Java, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk *modeling* aplikasi prosedural dalam VB atau C.



Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax*/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML syntax mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch dengan *Object-Oriented Design* (OOD), Jim

Rumbaugh dengan *Object Modeling Technique* (OMT), dan Ivar Jacobson dengan *Object-Oriented Software Engineering* (OOSE).

Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi wirfs-brock, dsb. Masa itu terkenal dengan masa perang metodologi (*method war*) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan. *Object Management Group* terlihat pada Gambar 2.10.




Gambar 2.10 *Object Management Group*
Sumber: Buku *The Unified Modeling Language User Guide*, 1999

Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang boleh dikatan metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 dirilis *draft* pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh *Object Management Group*. Tahun 1997 UML versi 1.1 muncul, dan saat ini versi terbaru adalah versi 1.5 yang dirilis

bulan Maret 2003. Booch, Rumbaugh dan Jacobson menyusun tiga buku serial tentang UML pada tahun 1999. Sejak saat itulah UML telah menjelma menjadi standar bahasa pemodelan untuk aplikasi berorientasi objek.

### 2.5.1 *Use Case Diagram*

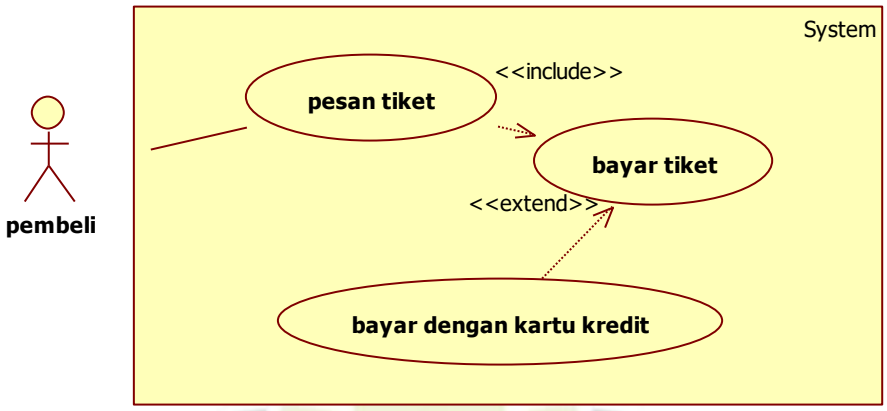
*Use case diagram* menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah use case merepresentasikan sebuah interaksi antara aktor dengan sistem. Use case merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-create sebuah daftar belanja, dan sebagainya.

Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. *Use case diagram* dapat sangat membantu bila kita sedang menyusun requirement sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem.

Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa use case yang di-*include* akan dipanggil setiap kali *use case* yang meng-include dieksekusi secara normal.

Sebuah *use case* dapat di-*include* oleh lebih dari satu use case lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. Sebuah *use case* juga dapat meng-*extend use case* lain dengan behaviour-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

Contoh *use case diagram* terlihat pada Gambar 2.11.




Gambar 2.11 Contoh *Use Case Diagram*

### 2.5.2 *Activity Diagram*

*Activity diagram* menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

*Activity diagram* merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-trigger oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan behaviour *internal* sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari *level* atas secara umum

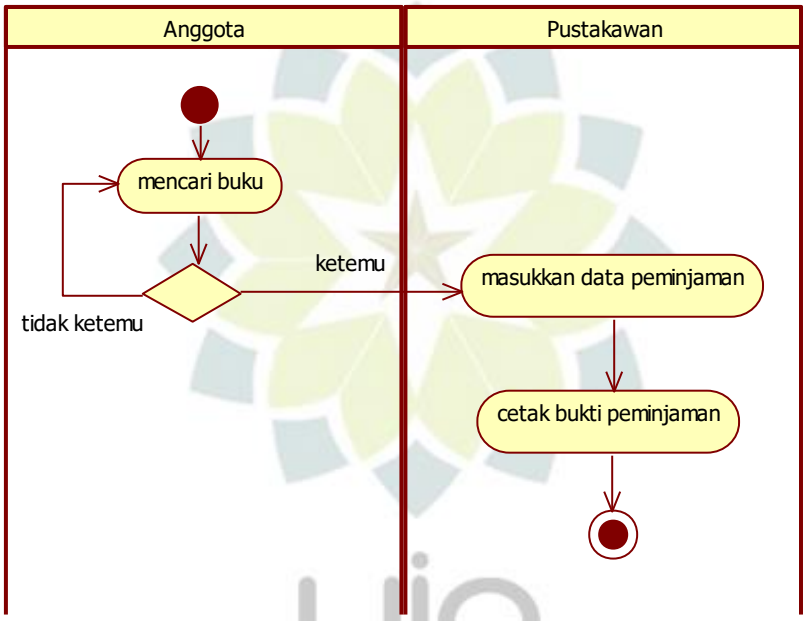
Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara use case menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi

tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal.

*Activity diagram* dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.

Contoh *activity diagram* tanpa *swimlane* terlihat pada Gambar 2.12.




Gambar 2.12 Contoh *Activity Diagram*



### 2.5.3 *Sequence Diagram*

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

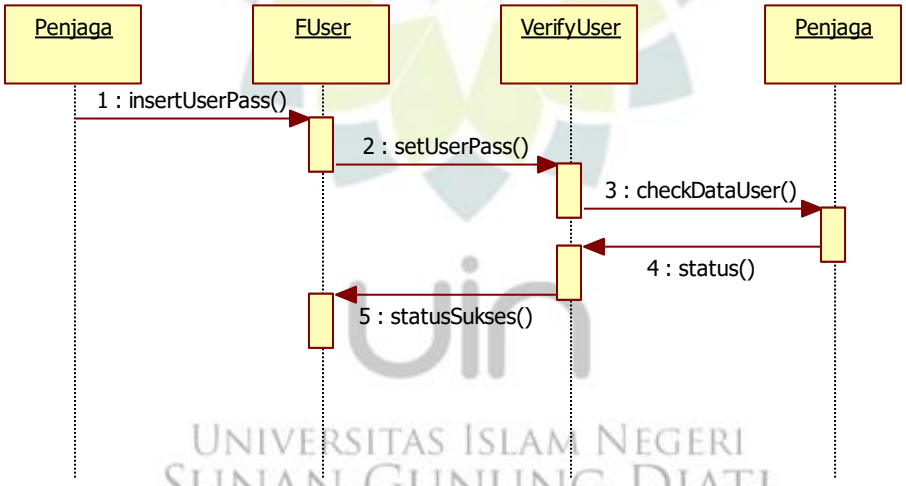
*Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respon dari sebuah *event* untuk menghasilkan output tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi

secara *internal* dan *output* apa yang dihasilkan. Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal.

*Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, message akan dipetakan menjadi operasi/metoda dari *class*. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message*.

Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk *objek boundary*, *controller* dan *persistent entity*.

Contoh *sequence diagram* terlihat pada Gambar 2.13.






Gambar 2.13 Contoh *Sequence Diagram*

### 2.5.4 *Class Diagram*

*Class* adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).

*Class diagram* menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.

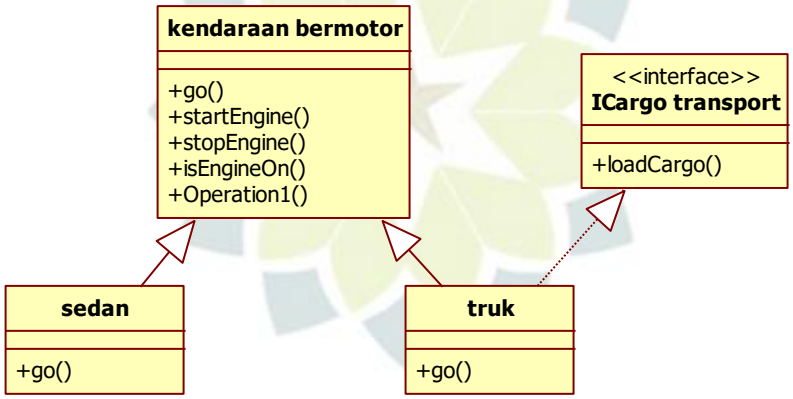
*Class* memiliki tiga area pokok :

a) Nama (dan *stereotype*).

b) Atribut.

c) Metoda.

Contoh *class diagram* terlihat pada Gambar 2.14.




Gambar 2.14 Contoh *class diagram*
Sumber: Buku *The Unified Modeling Language User Guide*, 1999