BAB II
LANDASAN TEORI

Pada bab dua ini akan dibahas mengenai teori-teori yang digunakan penulis untuk membangun sistem yaitu mengenai pembangunan aplikasi kriptografi *file* menggunakan algoritma RSA, metode rekayasa perangkat lunak dengan *waterfall*, perancangan sistem dengan *UML* dan teori lainya yang mendukung mengenai pembangunan aplikasi kriptografi *file* menggunakan algoritma RSA.

2.1 Kriptografi

2.1.1 Definisi dan Terminologi

Kriptografi (*cryptography*) berasal dari Bahasa Yunani: "*cryptós*" artinya "*secret*"(rahasia), sedangkan "*gráphein*" artinya "*writing*" (tulisan). Jadi, kriptografi berarti "*secret writing*" (tulisan rahasia). Ada beberapa definisi kriptografi yang telah dikemukakan di dalam berbagai literatur. Definisi yang dipakai di dalam buku-buku yang lama (sebelum tahun 1980-an) menyatakan bahwa kriptografi adalah ilmu dan seni untuk menjaga kerahasian pesan dengan cara menyandikannya ke dalam bentuk yang tidak dapat dimengerti lagi maknanya. Definisi ini mungkin cocok pada masa lalu di mana kriptografi digunakan untuk keamanan komunikasi penting seperti komunikasi di kalangan militer, diplomat, dan mata-mata. Namun saat ini kriptografi lebih dari sekadar *privacy*, tetapi juga untuk tujuan *data integrity*, *authentication*, dan *non-repudiation*.[3]

Definisi kriptografi yang dikemukakan di dalam :

> **Kriptografi** adalah ilmu dan seni untuk menjaga keamanan pesan
> (*Cryptography is the art and science of keeping messages secure*)

Sebagai pembanding, selain definisi tersebut di atas, terdapat pula definisi yang dikemukakan di dalam :

Kriptografi adalah ilmu yang mempelajari teknik-teknik matematika yang berhubungan dengan aspek keamanan informasi seperti kerahasiaan, integritas data, serta otentikasi

Di dalam kriptografi kita akan sering menemukan berbagai istilah atau terminologi.
Beberapa istilah yang penting untuk diketahui diberikan di bawah ini :

a) ***Pesan, Plainteks, dan Cipherteks***

Pesan (*message*) adalah data atau informasi yang dapat dibaca dan dimengerti maknanya. Nama lain untuk pesan adalah plainteks (*plaintext*) atau teks-jelas (*cleartext*). Pesan dapat berupa data atau informasi yang dikirim (melalui kurir, saluran telekomunikasi, dsb) atau yang disimpan di dalam media perekaman (kertas, *storage*, dsb). Pesan yang tersimpan tidak hanya berupa teks, tetapi juga dapat berbentuk citra (*image*), suara/bunyi (audio), dan video, atau berkas biner lainnya. Agar pesan tidak dapat dimengerti maknanya oleh pihak lain, maka pesan perlu disandikan ke bentuk lain yang tidak dapat dipahami. Bentuk pesan yang tersandi disebut **cipherteks** (*ciphertext*) atau **kriptogram** (*cryptogram*). Cipherteks harus dapat ditransformasikan kembali menjadi plainteks semula agar pesan yang diterima bias dibaca.

b) **Pengirim dan penerima**

Komunikasi data melibatkan pertukaran pesan antara dua entitas. **Pengirim** (*sender*) adalah entitas yang mengirim pesan kepada entitas lainnya. **Penerima** (*receiver*) adalah entitas yang menerima pesan. Entitas di sini dapat berupa orang, mesin (komputer), kartu kredit, dan sebagainya.

c) **Enkripsi dan dekripsi**

Proses menyandikan plainteks menjadi cipherteks disebut **enkripsi** (*encryption*) atau *enciphering* (standard nama menurut ISO 7498-2).

Sedangkan proses mengembalikan cipherteks menjadi plainteks semula dinamakan **dekripsi** (*decryption*) atau *deciphering* (standard nama menurut ISO 7498-2). Enkripsi dan dekripsi dapat diterapkan baik pada pesan yang dikirim maupun pada pesan tersimpan. Istilah *encryption of data in motion* mengacu pada enkripsi pesan yang ditransmisikan melalui saluran komunikasi, sedangkan istilah *enrypton of data at-rest* mengacu pada enkripsi dokumen yang disimpan di dalam *storage*.

d) ***Cipher* dan kunci**

Algoritma kriptografi disebut juga *cipher* yaitu aturan untuk *enchipering* dan *dechipering*, atau fungsi matematika yang digunakan untuk enkripsi dan dekripsi. Beberapa *cipher* memerlukan algoritma yang berbeda untuk *enciphering* dan *deciphering*.

Konsep matematis yang mendasari algoritma kriptografi adalah relasi antara dua buah himpunan yaitu himpunan yang berisi elemen-elemen plainteks dan himpunan yang berisi cipherteks. Enkripsi dan dekripsi merupakan fungsi yang memetakan elemenelemen antara kedua himpunan tersebut. Misalkan P menyatakan plainteks dan dan C menyatakan chiperteks, maka fungsi enkripsi E memetakan P ke C,

$$E(P) = C$$

Dan fungsi dekripsi D memetakan C ke P,

$$D(C) = P$$

Karena proses enkripsi kemudian dekripsi mengembalikan pesan ke pesan asal, maka kesamaan berikut harus benar,

$$D(E(P)) = P$$

e) ***Sistem Kriptografi***

Kriptografi membentuk sebuah sistem yang dinamakan sistem kriptografi. **Sistem kriptografi** (*cryptosystem*) adalah kumpulan yang terdiri dari

algoritma kriptografi, semua plainteks dan cipherteks yang mungkin, dan kunci. Di dalam system kriptografi, *cipher* hanyalah salah satu komponen saja.

f) ***Penyadap***

Penyadap (*eavesdropper*) adalah orang yang mencoba menangkap pesan selama ditransmisikan. Tujuan penyadap adalah untuk mendapatkan informasi sebanyakbanyaknya mengenai sistem kriptografi yang digunakan untuk berkomunikasi dengan maksud untuk memecahkan cipherteks. Nama lain penyadap: *enemy*, *adversary*, *intruder*, *interceptor*, *bad guy*. Ron Rivest, seorang pakar kriptografi, menyatakan bahwa *cryptography is about communication in the presence of adversaries* (Kriptografi adalah perihal berkomunikasi dengan keberadaan pihak musuh).

g) **Kriptanalisis dan kriptologi**

Kriptografi berkembang sedemikian rupa sehingga melahirkan bidang yang berlawanan yaitu kriptanalisis. **Kriptanalisis** (*cryptanalysis*) adalah ilmu dan seni untuk memecahkan chiperteks menjadi plainteks tanpa mengetahui *kunci* yang digunakan. Pelakunya disebut **kriptanalis**. Jika seorang **kriptografer** (*cryptographer*) mentransformasikan plainteks menjadi cipherteks dengan suatu algoritma dan kunci maka sebaliknya seorang kriptanalis berusaha untuk memecahkan cipherteks tersebut untuk menemukan plainteks atau kunci. **Kriptologi** (*cryptology*) adalah studi mengenai kriptografi dan kriptanalisis.

2.1.2 Tujuan Kriptografi

kriptografi bertujuan untuk memberi layanan keamanan (yang juga dinamakan sebagai aspek-aspek keamanan) sebagai berikut :

1. **Kerahasiaan** (*confidentiality*), adalah layanan yang ditujukan untuk menjaga agar pesan tidak dapat dibaca oleh pihak-pihak yang tidak

berhak. Di dalam kriptografi, layanan ini direalisasikan dengan menyandikan pesan menjadi cipherteks

2. **Integritas data** (*data integrity*), adalah layanan yang menjamin bahwa pesan masih asli/utuh atau belum pernah dimanipulasi selama pengiriman. Dengan kata lain, aspek keamanan ini dapat diungkapkan sebagai pertanyaan: "Apakah pesan yang diterima masih asli atau tidak mengalami perubahan (modifikasi)?". Untuk menjaga integritas data, sistem harus memiliki kemampuan untuk mendeteksi manipulasi pesan oleh pihak-pihak yang tidak berhak, antara lain penyisipan, penghapusan, dan pensubsitusian data lain kedalam pesan yang sebenarnya. Di dalam kriptografi, layanan ini direalisasikan dengan menggunakan tanda-tangan digital (*digital signature*).

3. **Otentikasi** (*authentication*), adalah layanan yang berhubungan dengan identifikasi, baik mengidentifikasi kebenaran pihak-pihak yang berkomunikasi (*user authentication* atau *entity authentication*) maupun mengidentifikasi kebenaran sumber pesan (*data origin authentication*). Dua pihak yang saling berkomunikasi harus dapat mengotentikasi satu sama lain sehingga ia dapat memastikan sumber pesan.

4. **Nirpenyangkalan** (*non-repudiation*), adalah layanan untuk mencegah entitas yang berkomunikasi melakukan penyangkalan, yaitu pengirim pesan menyangkal melakukan pengiriman atau penerima pesan menyangkal telah menerima pesan.

2.1.3 Sejarah Kriptografi

Kriptografi mempunyai sejarah yang panjang. Informasi yang lengkap mengenai sejarah kriptografi dapat ditemukan di dalam buku David Kahn yang berjudul *The Codebreakers*. Buku yang tebalnya 1000 halaman ini menulis secara rinci sejarah kriptografi mulai dari penggunaan kriptografi oleh Bangsa Mesir 4000 tahun yang lalu (berupa *hieroglyph* yang tidak standard pada piramid)

hingga penggunaan kriptografi pada abad ke-20. Secara historis ada empat kelompok orang yang berkontribusi terhadap perkembangan kriptografi, dimana mereka menggunakan kriptografi untuk menjamin kerahasiaan dalam komunikasi pesan penting, yaitu kalangan militer (termasuk intelijen dan mata-mata), kalangan diplomatik, penulis buku harian, dan pencinta (*lovers*). Di antara keempat kelompok ini, kalangan militer yang memberikan kontribusi paling penting penting karena pengiriman pesan di dalam suasana perang membutuhkan teknik enkripsi dan dekripsi yang rumit.

Sejarah kriptografi sebagian besar merupakan sejarah kriptografi klasik, yaitu metode enkripsi yang menggunakan kertas dan pensil atau mungkin dengan bantuan alat mekanik sederhana. Secara umum algoritma kriptografi klasik dikelompokkan menjadi dua kategori, yaitu algoritma transposisi (*transposition cipher*) dan algoritma substitusi (*substitution cipher*). *Cipher* transposisi mengubah susunan huruf-huruf di dalam pesan, sedangkan cipher substitusi mengganti setiap huruf atau kelompok huruf dengan sebuah huruf atau kelompok huruf lain. Sejarah kriptografi klasik mencatat penggunaan *cipher* transposisi oleh tentara Sparta di Yunani pada permulaan tahun 400 SM. Mereka menggunakan alat yang namanya *scytale*. *Scytale* terdiri dari sebuah kertas panjang dari daun *papyrus* yang dililitkan pada sebuah silinder dari diameter tertentu (diameter silender menyatakan kunci penyandian). Pesan ditulis secara horizontal, baris per baris. Bila pita dilepaskan, maka huruf-huruf di dalamnya telah tersusun secara acak membentuk pesan rahasia. Untuk membaca pesan, penerima pesan harus melilitkan kembali melilitkan kembali kertas tersebut ke silinder yang diameternya sama dengan diameter silinder pengirim. Sedangkang algoritma substitusi paling awal dan paling sederhana adalah *Caesar cipher*, yang digunakan oleh raja Yunani kuno, Julius Caesar.

Caranya adalah dengan mengganti setiap karakter di dalam alfabet dengan karakter yang terletak pada tiga posisi berikutnya di dalam susunan alfabet.

Kriptografi juga digunakan untuk tujuan keamanan. Kalangan gereja pada masa awal agama Kristen menggunakan kriptografi untuk menjaga tulisan relijius dari gangguan otoritas politik atau budaya yang dominan saat itu. Mungkin yang sangat terkenal adalah “Angka si Buruk Rupa (*Number of the Beast*) di dalam

Kitab Perjanjian Baru. Angka "666" menyatakan cara kriptografik (yaitu dienkripsi) untuk menyembunyikan pesan berbahaya; para ahli percaya bahwa pesan tersebut mengacu pada Kerajaan Romawi.

Di India, kriptografi digunakan oleh pencinta (*lovers*) untuk berkomunikasi tanpa diketahui orang. Bukti ini ditemukan di dalam buku *Kama Sutra* yang merekomendasikan wanita seharusnya mempelajari seni memahami tulisan dengan *cipher*.

Pada Abad ke-17, sejarah kriptografi mencatat korban ketika ratu Skotlandia, Queen Mary, dipancung setelah surat rahasianya dari balik penjara (surat terenkrpsi yang isinya rencana membunuh Ratu Elizabeth I) berhasil dipecahkan oleh seorang pemecah kode.

Seperti yang telah disebutkan di atas bahwa kriptografi umum digunakan di kalangan militer. Pada Perang Dunia ke II, Pemerintah Nazi Jerman membuat mesin enkripsi yang dinamakan *Enigma*. Mesin yang menggunakan beberapa buah *rotor* (roda berputar) ini melakukan enkripsi dengan cara yang sangat rumit. Namun *Enigma cipher* berhasil dipecahkan oleh pihak Sekutu dan keberhasilan memecahkan *Enigma* sering dikatakan sebagai faktor yang memperpendek perang dunia ke-2.

Kriptografi modern dipicu oleh perkembangan peralatan komputer digital. Dengan komputer digital, *cipher* yang lebih kompleks menjadi sangat mungkin untuk dapat dihasilkan. Tidak seperti kriptografi klasik yang mengenkripsi karakter per karakter (dengan menggunakan alfabet tradisionil), kriptografi modern beroperasi pada *string biner*. *Cipher* yang kompleks seperti DES (*Data Encryption Standard*) dan penemuan algoritma RSA adalah algoritma kriptografi modern yang paling dikenal di dalam sejarah kriptografi modern. Kriptografi modern tidak hanya berkaitan dengan teknik menjaga kerahasiaan pesan, tetapi juga melahirkan konsep seperti tanda-tangan digital dan sertifikat digital. Dengan kata lain, kriptografi modern tidak hanya memberikan aspek keamanan *confidentiality*, tetapi juga aspek keamanan lain seperti otentikasi, integritas data, dan nirpenyangkalan.

2.1.4 Kriptanalisis

Sejarah kriptografi paralel dengan sejarah kriptanalisis (*cryptanalysis*) , yaitu bidang ilmu dan seni untuk memecahkan cipherteks. Kata "kriptanalisis" sendiri relatif masih baru (pertama kali diungkapkan oleh William Friedman pada tahun 1920), namun sebenarnya teknik kriptanalisis sudah ada sejak abad ke-9. Adalah seorang ilmuwan Arab pada Abad IX bernama Abu Yusuf Yaqub Ibnu Ishaq Ibnu As-Sabbah Ibnu 'Omran Ibnu Ismail Al- Kindi, atau yang lebih dikenal sebagai Al-Kindi yang menulis buku tentang seni memecahkan kode. Dalam buku yang berjudul '*Risalah fi Istikhraj al-Mu'amma* (*Manuscript for the Deciphering Cryptographic Messages*), ia menuliskan naskah untuk menguraikan kode-kode rahasia. Di dalam buku tersebut Al-Kindi memperkenalkan teknik penguraian kode atau atau sandi yang sulit dipecahkan. Ia juga mengklasifikasikan sandi rahasia itu serta menjelaskan ilmu fonetik Arab dan sintaksisnya. Yang paling penting lagi, dalam bukunya ini ia mengenalkan penggunaan beberapa teknik statistika untuk memecahkan kode-kode rahasia (dikutip dari Republika *online*, 16 Juni 2006).

Apa yang dilakukan oleh Al-Kindi didalam kriptanalisis dikenal dengan nama teknik **analisis frekuensi**, yaitu teknik untuk memecahkan cipherteks berdasarkan frekuensi kemunculan karakter di dalam pesan dan kaitannya dengan frekuensi kemunculan karakter di dalam alfabet. Analisis frekuensi dilatarbelakangi oleh fakta bahwa *cipher* gagal menyembunyikan statistik kemunculan karakter di dalam cipherteksnya. Misalnya, di dalam Bahasa Inggris huruf "E" adalah huruf paling sering muncul di dalam kalimat-kalimat berbahas Inggris.

2.1.5 Kriptografi Kunci-Simetri dan Nirsimetri

Berdasarkan kunci yang digunakan untuk enkripsi dan dekripsi, kriptografi dapat dibedakan lagi menjadi **kriptografi kunci-simetri** (*symmetric-key cryptography*) dan **kriptografi kunci-nirsimetri** (*asymmetric-key cryptography*).

Pada sistem kriptografi kunci-simetri, kunci untuk enkripsi sama dengan kunci untuk dekripsi, oleh karena itulah dinamakan kriptografi simetri . Istilah

lain untuk kriptografi kunci-simetri adalah kriptografi kunci privat (*private-key cryptography*), kriptografi kunci rahasia (*secret-key cryptography*), atau kriptografi konvensional (*conventional cryptography*).

Sistem kriptografi kunci-simetri (atau disingkat menjadi "kriptografi simetri" saja), mengasumsikan pengirim dan penerima pesan sudah berbagi kunci yang sama sebelum bertukar pesan. Keamanan sistem kriptografi simetri terletak pada kerahasiaan kuncinya. Kriptografi simetri merupakan satu-satunya jenis kriptografi yang dikenal dalam catatan sejarah hingga tahun 1976. Semua algoritma kriptografi klasik termasuk ke dalam sistem kriptografi simetri. Di sisi lain, ada puluhan algoritma kriptografi modern yang termasuk ke dalam sistem kriptografi simetri, diantaranya adalah *DES* (*Data Encryption Standard*), *Blowfish*, *Twofish*, *Triple-DES*, *IDEA*, *Serpent*, dan yang terbaru adalah *AES* (*Advanced Encryption Standard*).

Secara umum, *cipher* yang termasuk ke dalam kriptografi simetri beroperasi dalam mode blok (*block cipher*), yaitu setiap kali enkripsi/dekripsi dilakukan terhadap satu blok data (yang berukuran tertentu), atau beroperasi dalam mode aliran (*stream cipher*), yaitu setiap kali enkripsi/dekripsi dilakukan terhadap satu bit atau satu *byte* data. Aplikasi kriptografi simetri yang utama adalah adalah melindungi kerahasiaan data yang dikirim melalui saluran tidak aman dan melindungi kerahasiaan data yang disimpan pada media yang tidak aman. Kelemahan dari sistem ini adalah baik pengirim maupun penerima pesan harus memiliki kunci yang sama, sehingga pengirim pesan harus mencari cara yang aman untuk memberitahukan kunci kepada penerima pesan.

Jika kunci untuk enkripsi tidak sama dengan kunci untuk dekripsi, maka kriptografinya dinamakan sistem kriptografi nirsimetri. Nama lainnya adalah **kriptogarfi kunci-publik** (*public-key cryptography*), sebab kunci untuk enkripsi tidak rahasia dan dapat diketahui oleh siapapun (diumumkan ke publik), sementara kunci untuk dekripsi hanya diketahui oleh penerima pesan (karena itu rahasia). Pada kriptografi jenis ini, setiap orang yang berkomunikasi mempunyai sepasang kunci, yaitu kunci privat dan kunci publik. Pengirim mengenkripsi pesan dengan menggunakan kunci publik si penerima pesan (*receiver*).

Hanya penerima pesan yang dapat mendekripsi pesan karena hanya ia yang mengetahui kunci privatnya sendiri. Contoh algoritma kriptografi kunci-publik diantaranya *RSA*, *Elgamal*, *DSA*, dan sebagainya.

Kriptografi kunci-publik dapat dapat dianalogikan seperti kotak surat yang terkunci dan memiliki lubang untuk memasukkan surat. Setiap orang dapat memasukkan surat ke dalam kotak surat tersebut, tetapi hanya pemilik kotak yang dapat membuka kotak dan membaca surat di dalamnya karena ia yang memiliki kunci. Keuntungan sistem ini ada dua. Pertama, tidak ada kebutuhan untuk mendistribusikan kunci privat sebagaimana pada system kriptografi simetri. Kunci publik dapat dikirim ke penerima melalui saluran yang sama dengan saluran yang digunakan untuk mengirim pesan. Saluran untuk mengirim pesan umumnya tidak aman.

Kedua, jumlah kunci dapat ditekan. Untuk berkomunikasi secara rahasia dengan banyak orang tidak perlu kunci rahasia sebanyak jumlah orang tersebut, cukup membuat dua buah kunci, yaitu kunci publik bagi para koresponden untuk mengenkripsi pesan, dan kunci privat untuk mendekripsi pesan. Berbeda dengan kriptografi kunci-simetris dimana jumlah kunci yang dibuat adalah sebanyak jumlah pihak yang diajak berkorespondensi. Contoh penggunaan, misalkan jaringan komputer menghubungkan komputer karyawan di kantor cabang dengan komputer menejer di kantor pusat. Seluruh kepala cabang diberitahu bahwa kalau mereka mengirim laporan ke menejer di kantor pusat, mereka harus mengenkripsi laporan tersebut dengan kunci publik menejer (kunci publik menejer diumumkan kepada seluruh kepala cabang). Untuk mengembalikan laporan tersandi ke laporan semula, hanya menejer yang dapat melakukan dekripsi, karena hanya dialah yang memegang kunci privat. Selama proses transmisi cipherteks dari kantor cabang ke kantor pusat melalui saluran komunikasi mungkin saja data yang dikirim disadap oleh pihak ketiga, namun pihak ketiga ini tidak dapat mengembalikan cipherteks ke plainteksnya karena ia tidak mengetahui kunci untuk dekripsi.

Meski berusia relatif muda (sejak 1976), kriptografi kunci-publik mempunyai kontribusi yang luar biasa dibandingkan dengan sistem kriptografi simetri. Kontribusi yang paling penting adalah tanda-tangan digital pada pesan untuk memberikan aspek keamanan otentikasi, integritas data, dan

nirpenyangkalan. Tanda-tangan digital adalah nilai kriptografis yang bergantung pada isi pesan dan kunci yang digunakan. Pengirim pesan mengenkripsi pesan (yang sudah diringkas) dengan kunci privatnya, hasil enkripsi inilah yang dinamakan tanda-tangan digital. Tanda-tangan digital dilekatkan (*embed*) pada pesan asli. Penerima pesan memverifikasi tanda-tangan digital dengan menggunaklan kunci publik. memperlihatkan sebuah surat elektronik yang di bagia bawah sudah dibubuhi tanda-tangan digital (di antara *BEGIN* dan *END SIGNATURE*).

2.2 RSA

Dari sekian banyak algoritma kriptografi kunci-publik yang pernah dibuat, algoritma yang paling populer adalah algoritma RSA. Algoritma RSA dibuat oleh 3 orang peneliti dari MIT (*Massachussets Institute of Technology*) pada tahun 1976, yaitu: Ron (R)ivest, Adi (S)hamir, dan Leonard (A)dleman. Keamanan algoritma RSA terletak pada sulitnya memfaktorkan bilangan yang besar menjadi faktor-faktor prima. Pemfaktoran dilakukan untuk memperoleh kunci pribadi. Selama pemfaktoran bilangan besar menjadi faktor-faktor prima belum ditemukan algoritma yang mangkus, maka selama itu pula keamanan algoritma RSA tetap terjamin.[3]

Besaran-besaran yang digunakan pada algoritma RSA[3]:

1. p dan q bilangan prima (rahasia)
2. $r = p \cdot q$ (tidak rahasia)
3. $\phi(r) = (p-1)(q-1)$ (rahasia)
4. e (kunci enkripsi) (tidak rahasia)
5. d(kunci dekripsi) (rahasia)
6. m(plainteks) (rahasia)
7. c(cipherteks) (tidak rahasia)

2.2.1 Perumusan Algoritma RSA

Algoritma RSA didasarkan pada teorema Euler yang menyatakan bahwa:

$$a^{\phi(n)} \equiv 1 \pmod{n} \qquad (2.1)$$

dengan syarat

1. a harus relatif prima terhadap n
2. $\phi(n) = n(1 - 1/p_1)(1 - 1/p_2) \ldots (1 - 1/pr)$, yang dalam hal ini $p_1, p_2, \ldots, p_r$ adalah faktor prima dari n. $\phi(n)$ adalah fungsi yang menentukan berapa banyak dari bilangan-bilangan 1, 2, 3, …, n yang relatif prima terhadap n.

   Berdasarkan sifat $a^k \equiv b^k \pmod{n}$ untuk k bilangan bulat ≥ 1, maka persamaan (2.1) dapat ditulis menjadi

   $$a^{k\phi(n)} \equiv 1^k \pmod{n}$$

   atau

   $$a^{m\phi(n)} \equiv 1 \pmod{n} \qquad (2.2)$$

Bila a diganti dengan m, maka persamaan (2.2) menjadi

$$m^{m\phi(n)} \equiv 1 \pmod{n} \qquad (2.3)$$

Berdasarkan sifat $ac \equiv bc \pmod{n}$, maka bila persamaan (2.3) dikali dengan m menjadi:

$$m^{m\phi(n)+1} \equiv m \pmod{n} \qquad (2.4)$$

yang dalam hal ini m relatif prima terhadap n.

Misalkan e dan d dipilih sedemikian sehingga

$$e.d \equiv 1 \pmod{\phi(n)} \qquad (2.5)$$

atau

$$e.d = k\phi(n) + 1 \qquad (2.6)$$

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

$$m^{e.d} \equiv X \pmod{n} \qquad (2.7)$$

Persamaan (2.7) dapat ditulis kembali menjadi

$$(m^{e})^{d} \equiv m \pmod{n} \qquad (2.8)$$

yang artinya, perpangkatan m dengan e diikuti dengan perpangkatan dengan d menghasilkan kembali m semula. Berdasarkan persamaan (2.8), maka enkripsi dan dekripsi dirumuskan sebagai berikut:

$$E_e(m) = c \equiv m^e \bmod n \qquad (2.9)$$

$$D_d(c) = m \equiv c^d \bmod n \qquad (2.10)$$

Karena $e.d = d.e$, maka enkripsi diikuti dengan dekripsi ekivalen dengan dekripsi diikuti enkripsi:

$$D_d(E_e(m)) = E_e(D_d(m)) \equiv m^d \bmod n \qquad (2.11)$$

Oleh karena $m^d \bmod n \equiv (m + jn)^d \bmod n$ untuk sembarang bilangan bulat j, maka tiap plainteks m, $m + n$, $m + 2n$, …, menghasilkan cipherteks yang sama. Dengan kata lain, transformasinya dari banyak ke satu. Agar transformasinya satu-ke-satu, maka m harus dibatasi dalam himpunan {0, 1, 2, …, $n - 1$} sehingga enkripsi dan dekripsi tetap benar seperti pada persamaan (2.9) dan (2.10).

2.2.2 Algoritma Membangkitkan Pasangan Kunci

1. Pilih dua buah bilangan prima sembarang, p dan q.
2. Hitung $n = p \cdot q$ (sebaiknya $p \neq q$, sebab jika $p = q$ maka $n = p^2$ sehingga p dapat diperoleh dengan menarik akar pangkat dua dari n).
3. Hitung $\phi(n) = (p - 1)(q - 1)$.
4. Pilih kunci publik, e, yang relatif prima terhadap $\phi(n)$.
5. Bangkitkan kunci *privat* dengan menggunakan persamaan (5), yaitu $e.d \equiv 1 \pmod{\phi(n)}$.

   Perhatikan bahwa $e.d \equiv 1 \pmod{\phi(n)}$ ekivalen dengan $e.d = 1 + k\phi(n)$, sehingga secara sederhana d dapat dihitung dengan

$$d = \frac{1 + k\phi(n)}{e} \qquad (2.12)$$

Hasil dari algoritma diatas :

- Kunci public adalah pasangan (*e*,*n*)
- Kunci privat adalah pasangan (*d*,*n*)

Catatan: *n* tidak bersifat rahasia, sebab ia diperlukan pada perhitungan enkripsi/dekripsi.

Contoh :

Misalkan $p = 47$ dan $q = 71$ (keduanya prima). Selanjutnya, hitung nilai

$$n = p \cdot q = 3337$$

dan

$$\phi(n) = (p - 1)(q - 1) = 3220.$$

Pilih kunci publik $e = 79$, karena 79 relatif prima dengan 3220. *e* dan *n* dapat dipublikasikan ke umum.

Selanjutnya akan dihitung kunci dekripsi *d* seperti yang dituliskan pada langkah instruksi 5 dengan menggunakan persamaan (2.12),

$$d = \frac{1 + (k \times 3220)}{79}$$

Dengan mencoba nilai-nilai $k = 1, 2, 3, \ldots$, diperoleh nilai *d* yang bulat adalah 1019. Ini adalah kunci privat untuk mendeskripsi pesan dan harus dirahasiakan.
Jadi, perhitungan kunci ini menghasilkan pasangan kunci privat dan kunci public:
Kunci public : ($e = 79$, $n = 3337$)
Kunci privat : ($d = 1019$, $n = 3337$)

2.2.3 Algoritma Enkripsi dan Deskripsi

Enkripsi

1. Ambil kunci publik penerima pesan, *e*, dan modulus *n*.
2. Nyatakan plainteks m menjadi blok-blok *m*1, *m*2, …, sedemikian sehingga setiap blok merepresentasikan nilai di dalam selang [0, *n* – 1].
3. Setiap blok *mi* dienkripsi menjadi blok *ci* dengan rumus $c_i = m_i^{\ e} \bmod n$

Dekripsi

1. Setiap blok cipherteks *ci* didekripsi kembali menjadi blok *mi* dengan rumus $m_i = c_i^{\ d} \bmod n$

Contoh :

Misalkan *B* mengirim pesan kepada *A*. Pesan (plainteks) yang akan dikirim oleh *A* adalah

m = HARI INI

atau dalam sistem desimal (pengkodean ASCII) adalah

7265827332737873

B memecah *m* menjadi blok yang lebih kecil, misalnya *m* dipecah menjadi enam blok yang berukuran 3 digit:

m_1 = 726 m_4 = 273

m_2 = 582 m_5= 787

m_3 = 733 m_6 = 003

Nilai-nilai m_i ini masih terletak di dalam selang [0, 3337 – 1] agar transformasi menjadi satu-ke-satu.

B mengetahui kunci publik *A* adalah *e* = 79 dan *n* = 3337. *A* dapat mengenkripsisikan setiap blok *plainteks* sebagai berikut :

$c_1 = 726^{79} \bmod 3337 = 215$; $c_2 = 582^{79} \bmod 3337 = 776$;

$c_3 = 733^{79} \bmod 3337 = 1743$; $c_4 = 273^{79} \bmod 3337 = 933$;

$c_5 = 787^{79} \bmod 3337 = 1731$; $c_6 = 003^{79} \bmod 3337 = 158$;

Jadi, cipherteks yang dihasilkan adalah

c = 215 776 1743 933 1731 158.

Dekripsi dilakukan dengan menggunakan kunci privat

d = 1019

Blok-blok cipherteks didekripsikan sebagai berikut:

m_1 = 2151019 mod 3337 = 726

m_2 = 7761019 mod 3337 = 582

m_3 = 17431019 mod 3337 = 733

…

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

m = 7265827332737873

yang dalam sistem pengkodean ASCII adalah

m = HARI INI.

2.2.4 Keamanan RSA

Keamanan algoritma *RSA* didasarkan pada sulitanya memfaktorkan bilangan besar menjadi factor-faktor primanya.[3]

Masalah pemfaktoran: Faktorkan *n*, yang dalam hal ini *n* adalah hasil kali dari dua atau lebih bilangan prima.

Pada *RSA*, masalah pemfaktoran berbunyi: Faktorkan *n* menjadi dua faktor primanya, *p* dan *q*, sedemikian sehingga $n = p \cdot q$. Sekali *n* berhasil difaktorkan menjadi *p* dan *q*, maka $\phi(r) = (p - 1)(q - 1)$ dapat dihitung. Selanjutnya, karena kunci enkrispi *e* diumumkan (tidak rahasia), maka kunci dekripsi *d* dapat dihitung dari persamaan $e \cdot d \equiv 1 \pmod{\phi(n)}$.

Selama 300 tahun para matematikawan mencoba mencari factor bilangan yang besar namun tidak banyak membuahkan hasil. Semua bukti yang diketahui menunjukkan bahwa upaya pemfaktoran itu luar biasa sulit. Belum ditemukan algoritma pemfaktoran bilangan besar dalam waktu polynomial, tetapi juga tidak dapat dibuktikan algoritma tersebut ada.

Fakta inilah yang membuat algoritma *RSA* dianggap aman. Penemu algoritma *RSA* bahkan menyarankan nilai *p* dan *q* panjangnya lebih dari 100

angka. Dengan demikian hasil kali $n = p \times q$ akan berukuran lebih dari 200 angka. Menurut Rivest dan kawan-kawan, usaha untuk mencari faktor bilangan 200 angka membutuhkan waktu komputasi selama 4 milyar tahun, sedangkan untuk bilangan 500 angka membutuhkan waktu 10^{25} tahun! [MEY82] (dengan asumsi bahwa algoritma pemfaktoran yang digunakan adalah algoritma yang tercepat saat ini dan komputer yang dipakai mempunyai kecepatan 1 milidetik).

Secara umum dapat disimpulkan bahwa *RSA* hanya aman jika *n* cukup besar. Jika panjang *n* hanya 256 bit atau kurang, ia dapat difaktorkan dalam beberapa jam saja dengan sebuah computer *PC* dan program yang tersedia secara bebas. Jika panjang *n* 512 bit atau kurang, ia dapat difaktorkan dengan beberapa ratus computer[WIK06].

Tahun 1997, tiga orang penemu *RSA* membuat sayembara untuk memecahkan *chiperteks* dengan menggunakan *RSA* di majalah *Scientic American*. Hadiahnya $100. Tahun 1994, kelompok yang bekerja dengan kolaborasi internet berhasil memecahkan *chiperteks* hanya dalam waktu 8 bulan.

2.3 Sistem Keamanan AES (*Advanced Encryption System*)

Algoritma AES pertama kali didesain oleh Vincent Rijmen dan Joan Daemen yang diumumksn pada tahun 1998 dan tergolong block chiper. AES yang dikenal juga dengan nama Rijndael merupakan salah satu dari sekian banyak desain yang lolos seleksi dan akhirnya diadopsi oleh NIST (National Institute of Standards and Technology) menjadi standar untuk Amerika pada tahun 2001. AES sendiri direkomendasikan oleh NIST sebagai pengganti DES (Descryption Encryption System)[15]. AES dikenal sebagai algoritma enkripsi yang cukup cepat baik pada perangkat lunak maupun perangkat keras [15]. Ukuran block chiper pada AES adalah 128 bit dengan panjang key bervariasi yaitu 128, 192 atau 256. Sampai saat ini AES masih dianggap cukup aman untuk melindungi informasi atau data pemerintah Amerika yang sifatnya non-classified. Pada tahun 2003 pemerintah Amerika akhirnya mengumumkan bahwa AES bisa juga digunakan untuk melindungi informasi atau data ynag sifatnya classified. Walaupun berbasis algoritma Rijndael, AES tidak sama persis seperti Rijndael.

Rijndael mendukung semakin besar ukuran block chiper yang sudah pasti (fixed) sebesar 128 bit[3].

Bila dibandingkan dengan sistem keamanan RSA, sistem AES dianggap lebih efektif dipandang dari sisi konsumsi waktu enkripsi dan sumber daya (resource) memori. AES membutuhkan waktu yang lebih cepat dalam proses enkripsi[15] dan lebih besar dalam penggunaan *memory* resource[3].

Walaupun dipandang aman, banyak yang menguji keaman dari algoritma AES ini[16][17]. Persamaan logaritma yang tergolong sederhana dari sistem AES ini justru menyebabkan kemungkinan dapat dipecahkannya persamaan logaritma tersebut. Pada tahun 2002 melalui suatu proses pengujian yang sifatnya teoritis ditemukan bahwa AES mungkin untuk dijebol atau dipecahkan. Metode attack ini dinamakan "XSL attack" [17]. Namun, metode inipun juga masih spekulatif dan belum bisa dibuktikan saat ini [3].

2.4 Metode Rekayasa Perangkat Lunak dengan *Waterfall*

Waterfall mengusulkan sebuah pendekatan kepada perkembangan perangkat lunak yang sistematik dan sekuensial yang mulai pada tingkat dan kemajuan sistem pada seluruh analisis, desain, kode, pengujian, dan pemeliharaan. Dimodelkan setelah siklus rekayasa *konvensional*, model *waterfall* melingkupi aktifitas-aktifitas berikut :

1. Rekayasa dan pemodelan sistem/ informasi. Karena perangkat lunak selalu merupakan bagian dari sebuah sistem yang lebih besar, kerja dimulai dengan membangun syarat dari semua elemen dari semua sistem dan mengalokasikan beberapa subset dari kebutuhan perangkat lunak tersebut. Pandangan sistem ini penting ketika perangkat lunak harus berhubungan dengan elemen-elemen yang lain seperti perangkat lunak, manusia, dan *database*. Rekayasa dan analisis sistem menyangkut pengumpulan kebutuhan pada tingkat sistem dengan sejumlah kecil analisis serta desain tingkat puncak. Rekayasa informasi mencakup juga pengumpulan kebutuhan pada tingkat bisnis strategis dan tingkat area bisnis.
2. Analisis kebutuhan perangkat lunak. Proses pengumpulan kebutuhan diintensifkan dan difokuskan, khususnya pada perangkat lunak. Untuk

memahami sifat program yang yang dibangun, perekayasa perangkat lunak (analis) harus memahami domain informasi, tingkah laku, unjuk kerja, dan antar muka yang diperlukan. Kebutuhan baik untuk sistem maupun perangkat lunak didokumentasikan dan dilihat lagi dengan pelanggan.

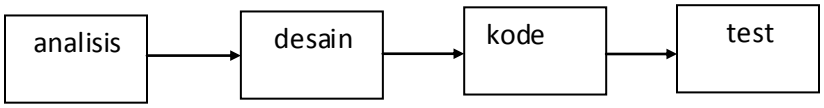
3. Desain. Desain perangkat lunak sebenarnya adalah proses multi langkah yang berfokus pada empat atribut sebuah program yang berbeda; struktur data arsitektur perangkat lunak, *refresentasi interface* dan detail (algoritma) prosedural.
4. Generasi kode. Desain harus diterjemahkan ke dalam bentuk mesin yang bisa dibaca. Langkah pembuatan kode melakukan tugas ini. Jika desain dilakukan dengan cara yang lengkap, pembuatan kode dapat diselesaikan secara mekanis.
5. Pengujian. Sekali kode dibuat, pengujian program dimulai. Program dimulai. Proses pengujian berfokus pada logika internal perangkat lunak, memastikan bahwa semua pernyataan sudah diuji, dan pada ekternal fungsional – yaitu mengarahkan pengujian untuk menemukan kesalahan-kesalahan dan memastikan bahwa input yang dibatasi akan memberikan hasil yang aktual yang sesuai dengan hasil yang dibutuhkan.
6. Pemeliharaan. Perangkat lunak akan mengalami perubahan setelah disampaikan kepada pelanggan (perkecualian yang mungkin adalah perangkat lunak yang dilekatkan). Perubahan akan terjadi karena kesalahan-kesalahan ditentukan, karena perangkat lunak harus disesuaikan untuk mengakomodasi perubahan-perubahan dilingkungan eksternalnya (contohnya perubahan yang dibutuhkan sebagai akibat dari perangkat *pheriperal* atau sistem operasi yang baru), atau karena pelanggan membutuhkan perkembangan fungsional atau unjuk kerja. Pemeliharaan perangkat lunak mengaplikasikan lagi setiap *fase* program sebelumnya dan tidak membuat yang baru lagi

Model *sekuensial linier* adalah paradigma rekayasa perangkat luas yang paling luas dipakai dan paling tua. Tetapi kritik dari paradigma tersebut telah menyebabkan dukungan aktif untuk mempertanyakan

kehandalannya[HAN95]. Masalah-masalah yang kadang-kadang terjadi ketika model ini diaplikasikan adalah :

1. Jarang sekali proyek nyata mengikuti aliran sekensial yang dianjurkan oleh model. Meskipun model linier bisa mengakomodasi iterasi, model itu melakukannya dengan cara tidak langsung. Sebagai hasilnya, perubahan-perubahan dapat menyebabkan keraguan pada saat tim proyek berjalan.
2. Kadang-kadang sulit bagi pelanggan untuk menyatakan semua kebutuhannya secara eksplisit. Model *linier sekuensial* memerlukan hal ini dan mengalami kesulitan untuk mengakomodasi ketidakpastian natural yang ada pada bagian awal proyek.
3. Pelanggan harus bersikap sabar. Sebuah versi kerja dari program-program itu tidak akan diperoleh sampai akhir proyek dilalui. Sebuah kesalahan besar, jika tidak terdeteksi samapi program yang bekerja tersebut dikaji ulang, bisa menjadi petaka.
4. Pengembang sering melakukan penundaan yang tidak perlu. Di dalam anlisis yang menarik tentang proyek aktual, Bradac [BRA94] mendapatkan bahwa sifat alami dari siklus kehidupan klasik membawa *kepada blocking state* dimana banyak anggota tim proyek harus menunggu tim yang lain untuk melengkapi tugas yang saling memiliki ketergantungan. Kenyataannya, waktu yang dipakai untuk menunggu bisa mengurangi waktu untuk usaha produktif.

Masing-masing dari masalah tersebut bersifat riil. Tetapi paradigma siklus kehidupan klasik memiliki tempat yang terbatas namun penting di dalam rekayasa perangkat lunak. Paradigma itu memberikan template dimana metode analisis, desain, pengkodean, pengujian, dan pemeliharaan bisa dilakukan. Siklus kehidupan klasik tetap menjadi model bagi rekayasa perangkat lunak yang paling luas dipakai. Sekalipun memiliki kelemahan, secara signifikan dia lebih baik daripada pendekatan yang sifatnya sembarang kepada pengembang perangkat lunak.



Gambar 2.1 Model *Waterfall*

2.5 *Unified Modeling Language* (*UML*)

Unified Modelling Language (*UML*) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem perangkat lunak. *UML* menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan *UML* kita dapat membuat model untuk semua jenis *aplikasi* perangkat lunak, dimana aplikasi tersebut dapat berjalan pada perangkat keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun.

UML mendefinisikan notasi dan *syntax*/semantik. Notasi *UML* merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram perangkat 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 *OOD* (*Object-Oriented Design*), Jim Rumbaugh OMT *(Object Modeling Technique*), dan Ivar Jacobson *OOSE* (*Object-Oriented Software Engineering*).

Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease draft pertama dari UML (versi 0.8).Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh **Object Management Group** (OMG – http://www.omg.org).

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 Konsepsi Dasar *UML*

Dari berbagai penjelasan rumit yang terdapat di dokumen dan buku-buku *UML*. Sebenarnya konsepsi dasar *UML* bisa kita rangkumkan dalam gambar dibawah :

Major Area	***Views***	***Diagrams***	***Main Concepts***
Structural	*Use Case View*	*Use Case Diagram*	*use case, actor, association, extend, Include, use case generalization*
Dynamic	*Interaction View*	*Sequence Diagram*	*interaction, object, message, activation*
Model Management	*Management View*	*Class Diagram*	*package, subsystem, model*

Tabel 2.1 Konsepsi *UML*

Seperti juga tercantum pada gambar diatas *UML* mendefinisikan diagram-diagram sebagai berikut:

- *use case diagram*
- *sequence diagram*
- *class diagram*

2.5.2 *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 kasus uji untuk semua bagianyang 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 *usecase* 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 sama. 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.

2.5.3 *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 :

1. Nama
2. Atribut
   - Nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek
   - Merupakan ciri dari sebuah objek
   - Dipunyai secara individual oleh sebuah objek
3. Metoda

Atribut dan metoda dapat memiliki salah satu sifat berikut :

1. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan.
2. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya.
3. *Public*, dapat dipanggil oleh siapa saja.

Hubungan Antar *Class*

1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukkan arah *query* antar *class*.
2. Agregasi, yaitu hubungan yang menyatakan bagian ("terdiri atas..").
3. Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.
4. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-*passing* dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence diagram* yang akan dijelaskan kemudian.

2.5.4 *Sequence Diagram*

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

Sequence diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons 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.

2.5.5 Pemodelan Penggunaan Sistem

Pemodelan pada penggunaan sistem terdapat beberapa tahap, diantaranya : Pemodelan *Use-Case*

- Identifikasi actor
- Identifikasi *use-case*
- Pembuatan diagram *use-case*
- Pembuatan diagram sekuen atau diagram kolaborasi untuk memperjelas masing-masing *use-case*
- Pembuatan *diagram* aktivitas

2.5.6 Pemodelan Perilaku Kelas Objek

Perilaku kelas pada objek terdapat beberapa poin, yaitu :

1. Evaluasi semua *use-case* agar dapat memahami sepenuhnya sekuen interaksi di dalam sistem.
2. Identifikasi kejadian-kejadian yang menuntun sekuen interaksi dan pahami bagaimana kejadian-kejadian ini berhubungan dengan objek-objek tertentu.
3. Pembuatan *diagram sekuen* untuk masing-masing *use-case*.
4. Pembuatan *diagram kolaborasi* untuk masing-masing kelas.
5. Pembuatan *diagram aktivitas* untuk memperjelas masing-masing kelas atau operasi.
6. Pembuatan *diagram statechart* untuk sistem.
7. Lakukan *review* model perilaku objek yang diperoleh untuk verifikasi akurasi dan konsistensi.

2.6 Perancangan Berorientasi Objek

Pada setiap disiplin rekayasa, perancangan merupakan pendekatan berdisiplin untuk menemukan solusi masalah. Perancangan merupakan penghubung antara spesifikasi kebutuhan dan implementasi. Perancangan menekankan pada solusi logik mengenai cara sistem memenuhi kebutuhan.

Terdapat banyak metode perancangan berorientasi objek. Perbedaan-perbedaan pada metode perancangan berorientasi objek bukan pada langkah-

langkah esensi, hanya rincian-rincian, sehingga dapat disimpulkan langkah-langkah perancangan berorientasi objek adalah sebagai berikut :

1. Perancangan sistem meliputi arsitektur sistem dan pendeskripsian subsistem-subsistem dan alokasinya di pemroses dan proses.
2. Pemilihan strategi perancangan untuk implementasi manajemen data, dukungan antarmuka dan manajemen proses/memori, penanganan kesalahan.
3. Perancangan mekanisme kendali yang cocok untuk sistem.
4. Perancangan rinci kelas objek dalam hal struktur data dan algoritmanya.
5. Perancangan pertukaran pesan menggunakan kolaborasi antar objek dan hubungan objek.
6. Penciptaan model pertukaran pesan.
7. Melakukan *review* atas model rancangan dan melakukan iterasi bila perlu untuk perbaikan model rancangan yang sebelumnya.

2.7 PHP

2.7.1 Sejarah PHP

Pada awalnya PHP merupakan kependekan dari *Personal Home Page* (Situs *personal*). PHP pertama kali dibuat oleh Rasmus Lerdorf pada tahun 1995. Pada waktu itu PHP masih bernama Form Interpreted (FI), yang wujudnya berupa sekumpulan skirp yang digunakan untuk mengolah data formulir dari web.
Selankutnya Rasmus merilis kode sumber tersebut untuk umum dan menamakannya PHP/FI.

Kemudian pada tahun 1997, sebuah perusahaan bernama Zend menulis ulang *interpreter* PHP menjadi lebih bersih, lebih baik, dan lebih cepat. Kemudian pada Juni 1998, perusahaan tersebut merilis *interpreter* baru untuk PHP dan meresmikan rilis tersebut sebagai PHP 3.0 dan singkatan PHP diubah menjadi akronim berulang ***PHP:hypertext Preprocessing.***
Pada pertengahan tahun 1990, Zend merilis *interpreter* PHP baru dan rilis tersebut dikenal dengan PHP 4.0. PHP 4.0 adalah versi PHP yang paling banyak dipakai disebabkan kemampuannya untuk membangun aplikasi *web* kompleks tetapi tetap memliki kecepatan dan stabilitas yang tinggi.

Pada Juni 2004, Zend merilis PHP 5.0. Dalam versi ini, inti dari *interpreter* PHP mengalami perubahan besar. Versi ini juga memasukkan model pemrograman berorientasi objek ke dalam PHP untuk menjawab perkembangan bahasa pemrograman kea rah paradigma berorientasi objek.

2.7.2 Kelebihan PHP

PHP berbasis *server side scripting*. PHP sendiri dapat melakukan tugas-tugas yang dilakukan dengan mekanisme CGI seperti mengambil, mengumpulkan data dari database, meng-gerate halaman dinamis, atau bahkan menerima dan mengirim *cookie*. Dan yang menjadi keutamaan PHP itu sendiri adalah PHP bisa digunakan di berbagai *operating system*, diantaranya Linux, Unix, Windows, mac OsX, RISC OS, dan operating system lainnya.

Kelebihan PHP yang harus di ketahui :

1. PHP Berbasis Server Side Scripting.
2. Command Line Scripting pada PHP.
3. PHP dapat membuat Aplikasi Desktop.
4. Digunakan untuk berbagai macam platform OS.
5. Mendukung berbagai macam Web Server.
6. Object Oriented Programming atau Procedural.
7. Output file PHP pada XHTML, HTML, dan XML.
8. Mendukung banyak RDMS (*database*).
9. Mendukung banyak komunikasi.
10. Pengolahan teks yang sangat baik.

2.8 MySQL

MySQL adalah sebuah *database server*, dapat juga berperan sebagai *client* sehingga sering disebut *database client*/ *server* yang *open source* dengan kemampuan dapat berjalan baik di OS (*operating System*) manapun, dengan *platform* Windows maupun Linux. Selain itu *database* ini memiliki beberapa kelebihan dibanding *database* lain, di antaranya adalah :

1. MySQL adalah sebuah software *database* yang open source, artinya program ini bersifat *free* atau bebas digunakan oleh siapa saja tanpa harus membeli dan membayar lisensi kepada pembuatnya.
2. MySQL merupakan sebuah *database client*. Selain menjadi *server* yang melayani permintaan, MySQL juga dapat melayani *query* yang mengakses *database* pada *server*. Jadi MySQL dapat juga berperan sebagai *client*.
3. MySQL mampu menerima *query* yang bertumpuk dalam satu permintaan atau yang disebut *multi-Threading*.
4. MySQL merupakan sebuah *database* yang mampu menyimpan data berkapasitas sangat besar hingga berukuran Gigabyte sekalipun.
5. MySQL memiliki kecepatan dalam pembuatan tabel maupun peng-*update*-an tabel.

2.9 XAMPP

Xampp merupakan tool yang menyediakan paket perangkat lunak ke dalam satu buah paket. Dengan menginstall XAMPP maka tidak perlu lagi melakukan instalasi dan konfigurasi *web server Apache*, PHP dan MySQL secara manual. XAMPP akan menginstalasi dan mengkonfigurasikannya secara otomatis untuk anda atau auto konfigurasi.

Software XAMPP versi ini terdiri atas :

a. APACHE

Apache sudah berkembang sejak versi pertamanya. Sampai saat ditulisnya artikel ini versi terakhirnya yang ada yaitu *Apache ver 2.0.54*. *Apache* bersifat open source, artinya setiap orang boleh menggunakannya, mengambil dan bahkan mengubah kode programnya.

Tugas utama *apache* adalah menghasilkan halaman web yang benar kepada peminta, berdasarkan kode PHP yang dituliskan oleh pembuat halaman web. Jika diperlukan juga berdasarkan kode PHP yang dituliskan, maka dapat saja suatu *database* diakses terlebih dahulu (misalnya dalam MySQL) untuk mendukung halaman web yang dihasilkan.

b. PhpMyAdmin

Pengelolaan *database* dengan *MySql* harus dilakukan dengan mengetikkan baris-baris perintah yang sesuai (*command line*) untuk setiap maksud tertentu. Jika ingin membuat *database*, ketikkan baris perintah yang sesuai untuk membuat database. Jika ingin menghapus tabel, ketikkan baris perintah yang sesuai untuk menghapus tabel. Hal tersebut tentu cukup menyulitkan karena kita harus hafal dan mengetikkan perintahnya satu persatu.

Banyak sekali perangkat lunak yang dapat dimanfaatkan untuk mengelola data base dalam *MySQL*, salah satunya adalah *phpMyAdmin*. Dengan *phpMyAdmin* kita dapat membuat tabel, mengisi data dan lain-lain dengan mudah tanpa harus hafal perintahnya. Untuk mengaktifkan phpMyAdmin langkah-langkahnya adalah : yang pertama setelah XAMPP kita terinstall, kita harus mengaktifkan *web server Apache* dan MySQL dari control panel XAMPP. Yang kedua, jalankan *browser* (IE, Mozilla Firefox atau Opera) lalu mengetikkan alamat web berikut : http://localhost/phpmyadmin/ pada address bar lalu tekan Enter. Langkah ketiga apabila telah nampak interface (tampilan antar muka) phpMyAdmin, kita bisa memulainya dengan mengetikkan nama database, nama tabel dan seterusnya.