

BAB II

TINJAUAN PUSTAKA

II.1. Perancangan

Perancangan mempunyai 2 maksud, yaitu untuk memenuhi kebutuhan kepada pemakai sistem dan untuk memberikan gambaran yang jelas kepada pemogram komputer dan ahli-ahli teknik lainnya yang terlibat. (Hanik Mujiati ; 2014 : 2)

II.2. Aplikasi

Aplikasi adalah suatu *sub* kelas perangkat lunak komputer yang memanfaatkan kemampuan komputer langsung untuk melakukan suatu tugas yang diinginkan pengguna. Contoh utama aplikasi adalah pengolah kata, lembar kerja, memanipulasi foto, merancang rumah dan pemutar media. Beberapa aplikasi yang digabung bersama menjadi suatu pake disebut sebagai suatu paket atau *suite* aplikasi (*application suite*). Contohnya adalah *Microsoft Office* dan *OpenOffice.org*, yang menggabungkan suatu aplikasi pengolah kata, lembar kerja dan beberapa aplikasi lainnya. Aplikasi-aplikasi dalam suatu paket biasanya memiliki atarmuka pengguna yang memiliki kesamaan sehingga memudahkan pengguna untuk mempelajari dan menggunakan tiap aplikasi. Sering kali, mereka memiliki kemampuan untuk saling berinteraksi satu sama lain sehingga menguntungkan pengguna. Contohnya, suatu lembar kerja dapat dibenamkan dalam suatu dokumen pengolah kata walaupun dibuat pada aplikasi lembar kerja yang terpisah. (Dahlan Abdullah ; 2013 : 152)

II.3. Keamanan

Masalah keamanan merupakan salah satu aspek penting dari sebuah sistem. Sayangnya sekali masalah keamanan ini sering kali kurang mendapat perhatian dari para pemilik dan pengelola sistem informasi. Seringkali masalah keamanan berada di urutan kedua, atau bahkan di urutan terakhir dalam daftar hal-hal yang dianggap penting. Apabila mengganggu performansi dari sistem, seringkali keamanan dikurangi atau ditiadakan (*Dowd & McHenry, 1998: 24-28*) yang dikutip oleh Rahardjo. Keamanan itu tidak dapat muncul demikian saja. Dia harus direncanakan. Ambil contoh berikut. Jika kita membangun sebuah rumah, maka pintu rumah kita harus dilengkapi dengan kunci pintu. Jika kita terlupa memasukkan kunci pintu pada budget perencanaan rumah, maka kita akan dikagetkan bahwa ternyata harus keluar dana untuk menjaga keamanan. (Ilham Alamsyah ; 2013 :2)

II.4. SMS (*Short Message Service*)

Pertukaran pesan melalui *Short Message Service* (SMS) merupakan suatu layanan yang populer dikalangan pemakai telepon bergerak di Indonesia. Pengiriman SMS dari satu perangkat ke perangkat lainnya melalui SMS Center untuk menyimpan dan menyampaikan SMS ke perangkat tujuan. (Hendra ; 2012 : 1)

II.4.1. Keamanan SMS

GSM menyediakan mekanisme keamanan untuk memastikan kerahasiaan dan integritas dari layanan, mekanisme ini ditempatkan antara perangkat bergerak

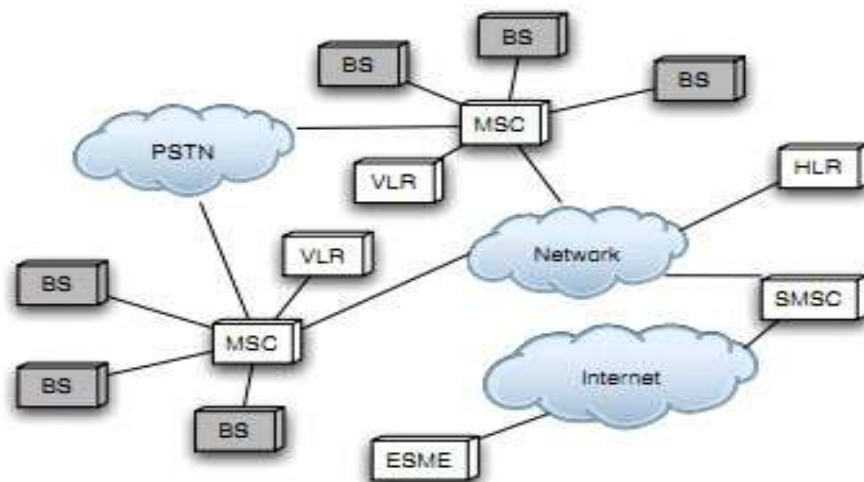
dengan jaringan operator dengan menggunakan algoritma A5, tetapi berdasarkan percobaan serangan kriptanalisis secara realtime terhadap keluaran algoritma A5/1 selama percakapan dua menit, kunci rahasia dapat dipecahkan dalam waktu satu detik, dan pada serangan kedua membutuhkan output dari algoritma A5/1 selama dua detik dari percakapan, dan kunci rahasia berhasil dipecahkan dalam waktu beberapa menit. (Hendra ; 2012 : 2)

II.4.2. Pengiriman SMS

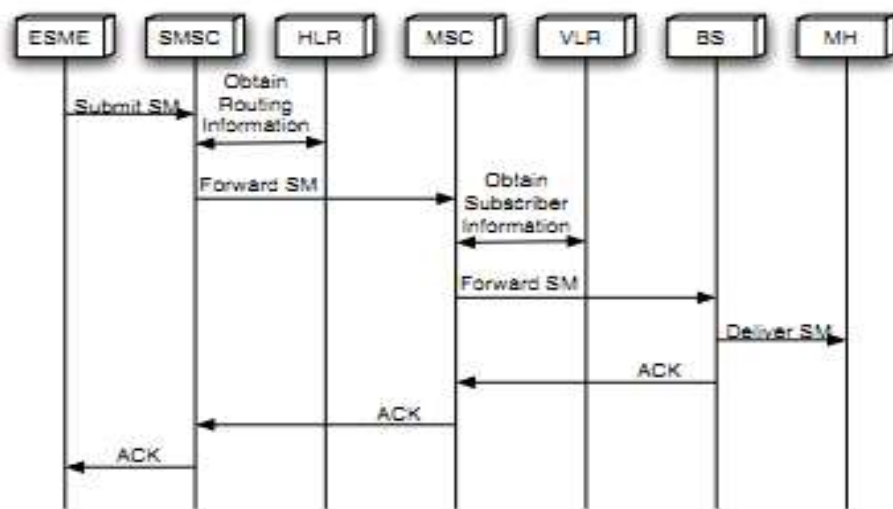
Ada dua metode untuk mengirim sebuah pesan *text* ke suatu perangkat bergerak, yaitu melalui suatu perangkat bergerak ataupun melalui suatu *External Messaging Entities* (ESMEs). ESMEs terdiri dari sejumlah besar perangkat berbeda dan memiliki berbagai antarmuka seperti email, portal messaging berbasis web yang terkoneksi pada jaringan telepon bergerak melalui internet maupun kanal dedicated tertentu. Pesan awalnya dikirim ke suatu server yang menangani trafik SMS yang dikenal sebagai *Short Messaging Service Center* (SMSC). Suatu *provider* yang mendukung pesan text harus memiliki paling sedikit satu SMSC pada jaringan mereka. SMSC perlu untuk menentukan bagaimana pesan disampaikan ke perangkat target. SMSC menanyakan kepada suatu basis data *Home Location Register* (HLR) yang menyimpan data pemakai dan informasi lokasi. Melalui interaksi dengan elemen lainnya, HLR menentukan routing informasi ke tujuan. Jika SMSC menerima balasan bahwa target tidak dapat dicapai, maka pesan akan disimpan untuk dikirim nantinya, jika sebaliknya maka akan dibalas dengan alamat *Mobile Switching Center* (MSC) yang tersedia untuk melayani. Ketika suatu pesan tiba dari SMSC ke MSC, MSC menanyakan

kepada suatu basis data *Visitor Location Register* (VLR) yang akan mengembalikan suatu duplikat informasi dari perangkat target ketika dia tidak berada pada HLR-nya. MSC kemudian mengirim pesan kepada Base Station (BS) untuk disampaikan ke target. (Hendra ; 2012 : 2)

Proses pengiriman SMS ditunjukkan oleh Gambar 1(a) dan 1(b) berikut :



(a). Jaringan SMS



(b). Aliran SMS

Gambar II.1. Penyederhanaan Jaringan SMS dan Aliran SMS

Sumber : Hendra : 2012 : 2

II.5. Kriptografi

Kriptografi berasal dari bahasa Yunani yaitu *cryptós* yang artinya “*secret*” (yang tersembunyi) dan *gráphein* yang artinya “*writting*” (tulisan). Jadi, kriptografi berarti “*secret writting*” (tulisan rahasia). Definisi yang dikemukakan oleh Bruce Schneier (1996), kriptografi adalah ilmu dan seni untuk menjaga keamanan pesan (*Cryptography is the art and science of keeping messages secure*). (Suriski Sitinjak ; 2010 : C-78)

II.6. Terminologi Kriptografi

Ada beberapa istilah-istilah yang penting dalam kriptografi, yaitu :

1. Pesan (*Plaintext* dan *Ciphertext*) : Pesan (*message*) adalah data atau informasi yang dapat dibaca dan dimengerti maknanya. Pesan asli disebut plainteks (*plaintext*) atau teks-jelas (*cleartext*). Sedangkan pesan yang sudah disandikan disebut cipherteks (*chipertext*)
2. 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.
3. Penyadap (*eavesdropper*) adalah orang yang mencoba menangkap pesan selama ditransmisikan.
4. Kriptanalisis dan Kriptologi : Kriptanalisis (*cryptanalysis*) adalah ilmu dan seni untuk memecahkan chiperteks menjadi plainteks tanpa mengetahui kunci yang digunakan. Pelakunya disebut kriptanalisis. Kriptologi (*cryptology*) adalah studi mengenai kriptografi dan kriptanalisis.

5. Enkripsi dan Dekripsi : Proses menyandikan plainteks menjadi cipherteks disebut enkripsi (*encryption*) atau enciphering. Sedangkan proses mengembalikan cipherteks menjadi plainteks semula dinamakan dekripsi (*decryption*) atau deciphering.
6. Cipher dan Kunci : Algoritma kriptografi disebut juga cipher yaitu aturan untuk enchipering dan dechipering, atau fungsi matematika yang digunakan untuk enkripsi dan dekripsi. Kunci (*key*) adalah parameter yang digunakan untuk transformasi *enciphering* dan *dechipering*. Kunci biasanya berupa string atau deretan bilangan. (Suriski Sitinjak ; 2010 : C-79)

II.7. Kriptografi Kunci Simetri (Kriptografi Kunci-Privat)

Pada sistem kriptografi kunci-simetri, kunci untuk enkripsi sama dengan kunci untuk dekripsi, oleh karena itulah dinamakan kriptografi simetri. Keamanan sistem kriptografi simetri terletak pada kerahasiaan kuncinya. Ada banyak algoritma kriptografi modern yang termasuk ke dalam sistem kriptografi simetri, diantaranya adalah DES (*Data Encryption Standard*), Blowfish, Twofish, Triple-DES, IDEA, Serpent, AES (*Advanced Encryption Standard*). (Suriski Sitinjak ; 2010 : C-79)

Algoritma kriptografi (*cipher*) simetri dapat dikelompokkan menjadi dua kategori, yaitu (Suriski Sitinjak ; 2010 : C-79) :

1. Cipher aliran (*stream cipher*)

Algoritma kriptografi beroperasi pada plainteks/cipherteks dalam bentuk bit tunggal, yang dalam hal ini rangkaian bit dienkrripsikan/didekrripsikan bit per bit.

2. Cipher blok (*block cipher*)

Algoritma kriptografi beroperasi pada plainteks/cipherteks dalam bentuk blok *bit*, yang dalam hal ini rangkaian *bit* dibagi menjadi blok-blok *bit* yang panjangnya sudah ditentukan sebelumnya. (Suriski Sitinjak ; 2010 : C-79)

II.8. Algoritma *Blowfish*

Blowfish diciptakan oleh seorang *Cryptanalyst* bernama *Bruce Schneier*, Presiden perusahaan *Counterpane Internet Security, Inc* (Perusahaan konsultan tentang kriptografi dan keamanan komputer) dan dipublikasikan tahun 1994. Dibuat untuk digunakan pada komputer yang mempunyai *microprosesor* besar (32-*bit* keatas dengan *cache* data yang besar). *Blowfish* merupakan algoritma yang tidak dipatenkan dan *licensefree*, dan tersedia secara gratis untuk berbagai macam kegunaan. (Suriski Sitinjak ; 2010 : C-79)

Pada saat *Blowfish* dirancang, diharapkan mempunyai kriteria perancangan sebagai berikut (Suriski Sitinjak ; 2010 : C-79) :

1. Cepat, *Blowfish* melakukan enkripsi data pada *microprocessors* 32-*bit* dengan *rate* 26 *clock cycles per byte*.
- b. *Compact* (ringan), *Blowfish* dapat dijalankan pada memori kurang dari 5K.
- c. Sederhana, *Blowfish* hanya menggunakan operasi-operasi sederhana: penambahan, XOR, dan *lookup* tabel pada operan 32-*bit*.
- d. Memiliki tingkat keamanan yang bervariasi, panjang kunci yang digunakan oleh *Blowfish* dapat bervariasi dan bisa sampai sepanjang 448 *bit*.

Dalam penerapannya sering kali algoritma ini menjadi tidak optimal. Karena strategi implementasi yang tidak tepat. Algoritma *Blowfish* akan lebih

optimal jika digunakan untuk aplikasi yang tidak sering berganti kunci, seperti jaringan komunikasi atau enkripsi *file* otomatis. Selain itu, karena algoritma ini membutuhkan memori yang besar, maka algoritma ini tidak dapat diterapkan untuk aplikasi yang memiliki memori kecil seperti *smart card*. Panjang kunci yang digunakan, juga mempengaruhi keamanan penerapan algoritma ini. (Suriski Sijinjak ; 2010 : C-79)

II.8.1. Bagian-Bagian Algoritma *Blowfish*

Algoritma *Blowfish* terdiri atas dua bagian, yaitu ekspansi kunci dan enkripsi data menurut (Suriski Sijinjak ; 2010 : C-79) berikut :

1. Ekspansi kunci (*Key-expansion*)

Berfungsi merubah kunci (minimum 32-bit, maksimum 448-bit) menjadi beberapa *array* subkunci (*subkey*) dengan total 4168 *byte* (18x32-bit untuk P-*array* dan 4x256x32-bit untuk S-*box* sehingga totalnya 33344 *bit* atau 4168 *byte*).

Kunci disimpan dalam K-*array*:

$$K_1, K_2, \dots, K_{j-1}, K_j, K_{j+1}, \dots, K_{14}$$

Kunci-kunci ini yang dibangkitkan (*generate*) dengan menggunakan subkunci yang harus dihitung terlebih dahulu sebelum enkripsi atau dekripsi data.

Sub-sub kunci yang digunakan terdiri dari :

P-*array* yang terdiri dari 18 buah 32-bit subkunci :

$$P_1, P_2, \dots, P_{18}$$

S-*box* yang terdiri dari 4 buah 32-bit, masing-masing memiliki 256 entri :

$$S_{1,0}, S_{1,1}, \dots, S_{1,255}$$

S2,0, S2,1, ..., S2,255

S3,0, S3,1, ..., S3,255

S4,0, S4,1, ..., S4,255

Langkah-langkah perhitungan atau pembangkitan subkunci tersebut adalah sebagai berikut:

- a. Inisialisasi *P-array* yang pertama dan juga empat *S-box*, berurutan, dengan string yang telah pasti. *String* tersebut terdiri dari digit-digit heksadesimal dari phi, tidak termasuk angka tiga di awal.

Contoh :

P1= 0x243f6a88

P2= 0x85a308d3

P3= 0x13198a2e

P4= 0x03707344

dan seterusnya sampai *S-box* yang terakhir (daftar heksadesimal digit dari phi untuk *P-array* dan *Sbox* bisa lihat Lampiran).

- b. XOR-kan P1 dengan 32-bit awal kunci, XOR-kan P2 dengan 32-bit berikutnya dari kunci, dan seterusnya untuk semua *bit* kunci. Ulangi siklus seluruh *bit* kunci secara berurutan sampai seluruh *P-array* ter-XOR-kan dengan *bit-bit* kunci. Atau jika disimbolkan : $P1 = P1 _ K1$, $P2 = P2 _ K2$, $P3 = P3 _ K3$, ... $P14 = P14 _ K14$, $P15 = P15 _ K1$, ... $P18 = P18 _ K4$.

Keterangan : $_$ adalah simbol untuk XOR.

- c. Enkripsikan *string* yang seluruhnya nol (*all-zero string*) dengan algoritma *Blowfish*, menggunakan subkunci yang telah dideskripsikan pada langkah 1 dan 2.
- d. Gantikan P1 dan P2 dengan keluaran dari langkah 3.
- e. Enkripsikan keluaran langkah 3 menggunakan algoritma *Blowfish* dengan subkunci yang telah dimodifikasi.
- f. Gantikan P3 dan P4 dengan keluaran dari langkah 5.
- g. Lanjutkan langkah-langkah di atas, gantikan seluruh elemen *P-array* dan kemudian keempat *S-box* secara berurutan, dengan hasil keluaran algoritma *Blowfish* yang terus-menerus berubah.

Total keseluruhan, terdapat 521 iterasi untuk menghasilkan subkunci-subkunci dan membutuhkan memori sebesar 4KB.

2. Enkripsi Data

Terdiri dari iterasi fungsi sederhana (*Feistel Network*) sebanyak 16 kali putaran (iterasi), masukannya adalah 64-bit elemen data X. Setiap putaran terdiri dari permutasi kunci-*dependent* dan substitusi kunci- dan data *dependent*. Semua operasi adalah penambahan (*addition*) dan XOR pada variabel 32-bit. Operasi tambahan lainnya hanyalah empat penelusuran tabel *array* berindeks untuk setiap putaran. (Suriski Sitinjak ; 2010 : C-80) Langkahnya adalah seperti berikut.

- a. Bagi X menjadi dua bagian yang masing-masing terdiri dari 32-bit: XL, XR.
- b. Lakukan langkah berikut

For i = 1 to 16:

XL = XL _ Pi

$$XR = F(XL) _ XR$$

Tukar XL dan XR

- c. Setelah iterasi ke-16, tukar XL dan XR lagi untuk melakukan membatalkan pertukaran terakhir.

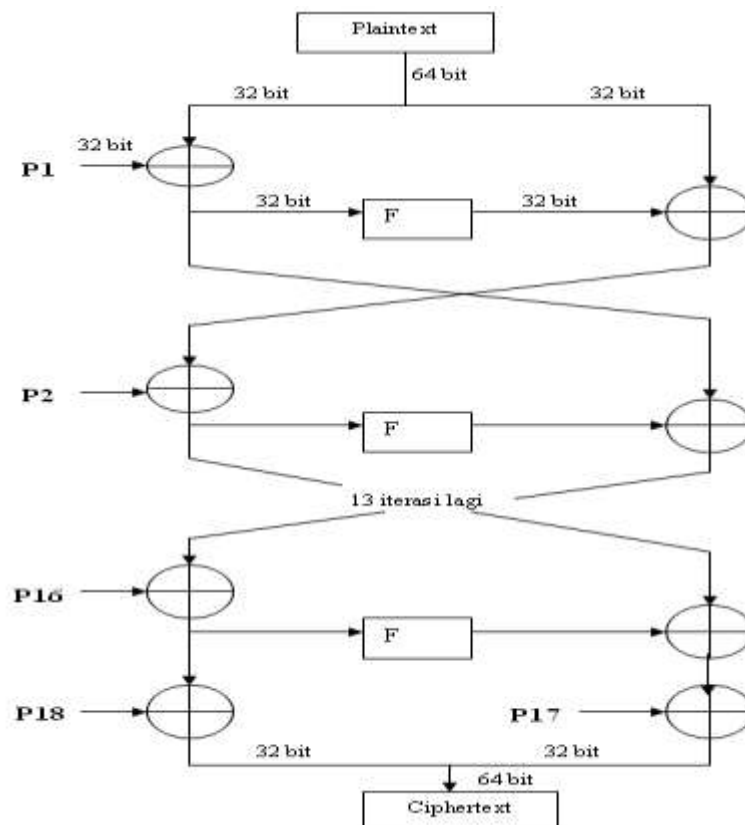
- d. Lalu lakukan

$$XR = XR _ P17$$

$$XL = XL _ P18$$

- e. Terakhir, gabungkan kembali XL dan XR untuk mendapatkan cipherteks.

Untuk lebih jelasnya, gambaran tahapan pada jaringan feistel yang digunakan Blowfish adalah seperti pada Gambar II.2.



Gambar II.2. Diagram Algoritma Enkripsi Blowfish

Sumber : Suriski Sitinjak ; 2010 : C-81

Pada langkah kedua, telah dituliskan mengenai penggunaan fungsi F.

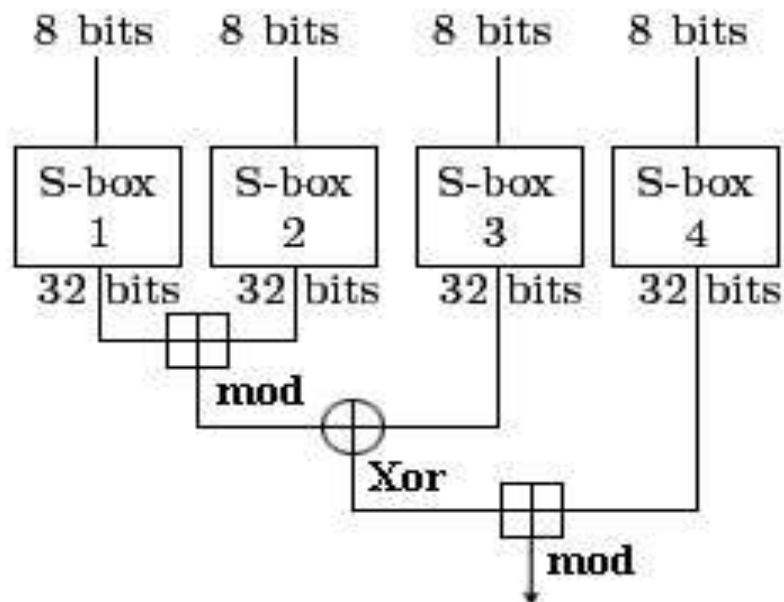
Fungsi F adalah: bagi XL menjadi empat bagian 8-bit: a,b,c dan d.

$$F(XL) = ((S1,a + S2,b \text{ mod } 232) \text{ XOR } S3,c) + S4,d \text{ mod } 232$$

.....(2.1)

Agar dapat lebih memahami fungsi F, tahapannya dapat dilihat pada

Gambar II.3.



Gambar II.3. Fungsi F dalam Blowfish

Sumber : Suriski Sitinjak ; 2010 : C-81

Dekripsi sama persis dengan enkripsi, kecuali bahwa P_1, P_2, \dots, P_{18} digunakan pada urutan yang berbalik (*reverse*). Algoritmanya dapat dinyatakan sebagai berikut :

for $i = 1$ to 16 do

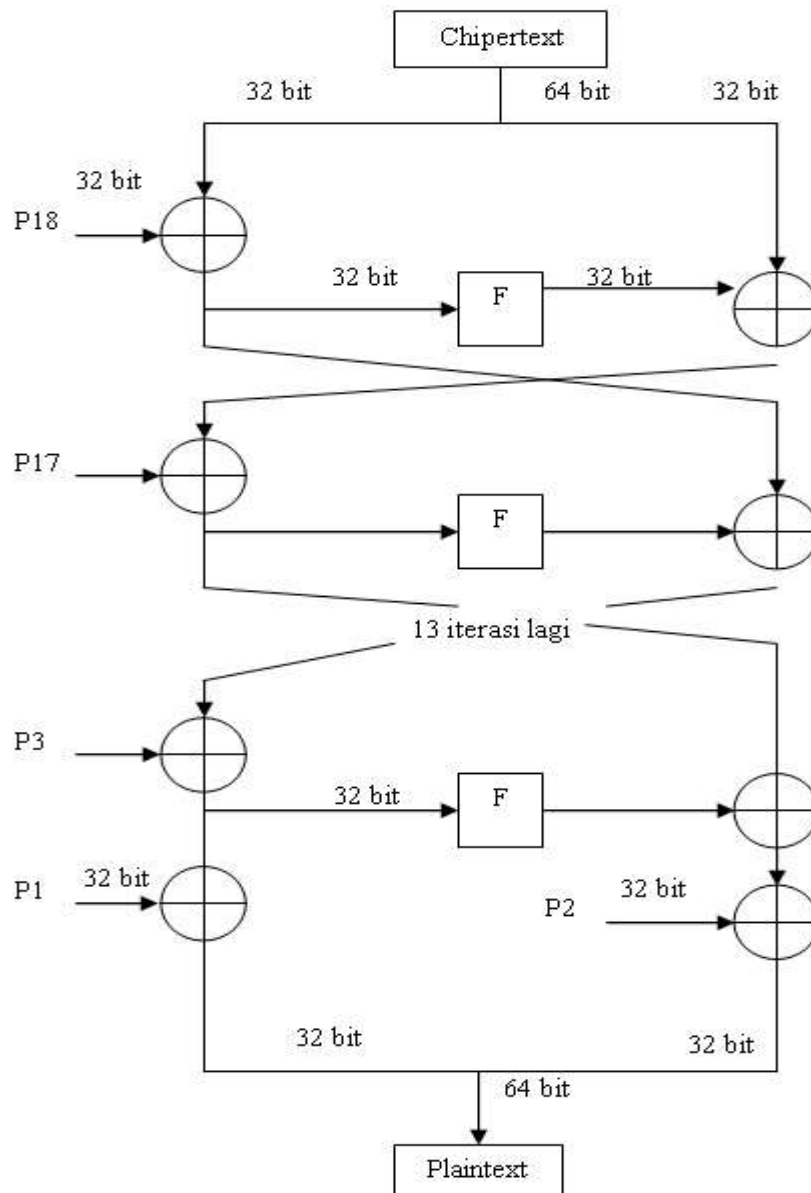
$XR_i = XL_{i-1} _ P_{19-i};$

$XL_i = F[XR_i] _ XR_{i-1};$

$$XL17 = XR16 \oplus P1;$$

$$XR17 = XL16 \oplus P2;$$

Blok diagram dekripsi seperti pada Gambar II.4 berikut :



Gambar II.4. Blok Diagram Dekripsi Blowfish

Sumber : Suriski Sitinjak ; 2010 : C-82

II.9. Java

Java adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer *standalone* ataupun pada lingkungan jaringan *Java 2* adalah generasi kedua dari *Java platform* (generasi awalnya adalah JDK atau *Java Development Kit*). *Java* inilah yang berdiri diatas mesin *interpreter* yang diberi nama *Java Virtual Machine*(JVM). JVM inilah yang akan membaca *bytecode* dalam *file .class* dari suatu program sebagai representasi langsung program yang berisi bahasa mesin”. Oleh karena itu bahasa java disebut juga sebagai bahasa pemrograman yang *portable* karena dapat dijalankan sebagai sistem operasi, asalkan pada sistem operasi tersebut terdapat JVM. (Utomo Budiyanto ; 2011 : 27)

Sun Microsystems telah mendefinisikan tiga *platform java* menurut Utomo Budiyanto 2011 yang masing – masing diarahkan untuk tujuan tertentu dan untuk lingkungan komputasi yang berbeda-beda:

1. *Java Standard Edition* (J2SE), adalah inti dari bahasa pemrograman *java*. JDK adalah salah satu *tool* dari J2SE untuk mengkompilasi program *java* pada JRE.
2. *Java Enterprise Edition* (J2EE), dengan *built-in* mendukung untuk *servlets*, JSP, dan XML, edisi ini ditujukan untuk aplikasi berbasis *server*.
3. *Java Micro Edition* (J2ME), didesain untuk meletakkan perangkat lunak *java* pada barang elektronik beserta perangkat pendukungnya.

Teknologi *Java* mencakup 2 elemen penting yaitu bahasa pemrograman (*programming language*) dan lingkungan aplikasi (*application environment*). *Java* sebagai bahasa pemrograman dapat diartikan bahwa *java* sebanding dengan

bahasa pemrograman seperti C++, *Pascal*, *Visual Basic*, dan lainnya, sedangkan *Java* sebagai lingkungan aplikasi berarti bahwa *java* dapat berjalan pada berbagai lingkungan seperti *browser(Applets)*, *server(servlets dan JSP)* dan pada *mobile device(midlet dan WAP)*. *Java* dalam hal ini mengungguli bahasa lainnya yang pernah ada jika dilihat dari sisi teknologi *mobile*. Hal ini dibuktikan dengan banyaknya jenis telepon genggam yang menggunakan *java* sebagai fitur utamanya. *Microsoft.NET mobile* pun kelihatannya belum dapat menyaingi keunggulan *Java* dalam bidang aplikasi *mobile*. Perlu diketahui bahwa *Microsoft* hanya mengandalkan solusi WAP yang mengembangkan ASP.NET untuk kebutuhan *mobile device*, sedangkan *Java* memiliki 2 solusi yaitu WAP dan MIDP (*Mobile Information Device Profile*). Solusi pertama adalah dengan mengandalkan J2EE (*Java 2 Enterprise Edition*) dengan produknya yang bernama JSP (*Java Server Pages*) dan *Java Servlets*. JSP dan *Servlets* ini digunakan untuk membentuk halaman WAP. Solusi kedua dengan menggunakan J2ME(*Java 2 Micro Edition*) MIDP dengan produknya yang bernama *Midlets*. *Midlets* inilah yang menjadi fitur andalan oleh beberapa jenis telepon genggam terbaru. (Utomo Budiyanto ; 2011 : 27)

II.9.1. J2ME (*Java 2 Micro Edition*)

Java Micro Editon atau yang biasa disebut J2ME adalah bagian dari J2SE, karena itu banyak pustaka yang ada pada J2SE dapat digunakan pada J2ME. Tetapi J2ME mempunyai beberapa pustaka khusus yang tidak dimiliki J2SE. Kelahiran *platform* J2ME timbul karena dibutuhkan adanya sebuah *platform* komputasi yang mengakomodasi piranti komputer elektronik dan *embedded*.

Piranti ini dikelompokkan menjadi dua kategori, menurut Utomo Budiyanto ; 2011 : 27 yaitu :

1. Personal, *piranti mobile* yang dapat digunakan untuk komunikasi melalui jaringan tertentu misalkan ponsel, Personal *Digital Assistant* (PDA), *Palm*, *Pocket PC* dan *organizer*.
2. Piranti informasi yang digunakan bersama dengan jaringan tetap, koneksi jaringan yang tidak putus-putus misalnya TV, internet dan sistem navigasi.

Kategori pertama mengarahkan piranti untuk tujuan khusus atau fungsi-fungsi tertentu yang terbatas dan tidak digunakan untuk mesin komputasi yang serba guna. Kategori kedua diarahkan untuk piranti yang mempunyai kapabilitas yang lebih besar dengan fasilitas *user interface* yang lebih baik, kemampuan komputasi yang lebih besar. (Utomo Budiyanto ; 2011 : 27)

1. Keunggulan J2ME

Salah satu kelebihan *Java* yang paling signifikan adalah *run everywhere*. Dengan kelebihan ini, para pengembang yang sudah terbiasa mengembangkan aplikasi dalam bingkai kerja J2ME dan J2EE akan mampu bermigrasi dengan mudah untuk mengembangkan aplikasi J2ME. Selain itu, *Java* juga merupakan *platform* yang memiliki banyak keunggulan lain, keunggulan *Java* secara umum adalah :

- a. *Multiplatform*, aplikasi J2ME bisa berjalan diatas banyak *platform* yang didalamnya terdapat JVM. Beberapa *platform* yang tersedia didalamnya terdapat JVM antara lain *Windows CR*, *Symbian*, *Embedded Linux* dan sebagainya.

- b. *Robust*, kode-kode *Java* adalah kode-kode *robust*, karena *virtual machine* mengatur keamanan proses eksekusi aplikasi. *Java virtual machine* menyediakan *garbage collector* yang berfungsi mencegah kebocoran memory.
- c. Terintegrasi dengan baik, J2ME bisa terhubung dengan *back-end J2EE server* dan *web services* dengan mudah karena menyediakan pustaka-pustaka API RMI dan *web services*.
- d. Berorientasi obyek, *Java* merupakan salah satu bahasa pemrograman yang murni berorientasi obyek. Hal ini mempermudah dan mempercepat pengembangan sistem yang dikembangkan dengan metode analisa dan desain berorientasi obyek. (Utomo Budiyanto ; 2011 : 27)

II.10. Sistem Operasi *Android*

II.10.1. *Android*

Android adalah sistem operasi untuk telepon seluler yang berbasis Linux, yang mencakup sistem operasi, *middleware* dan aplikasi. *Android* tidak terikat ke satu merek telepon seluler. *Android* menyediakan *platform* terbuka bagi para pengembang untuk menciptakan aplikasi mereka sendiri hingga dapat digunakan oleh berbagai peranti *mobile*. Beberapa fitur utama dari *Android* antara lain WiFi *hotspot*, *Multi-touch*, *Multitasking*, GPS, *support java*, mendukung banyak jaringan (GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, *Bluetooth*, Wi-Fi, LTE, and WiMAX) dan juga kemampuan dasar telepon seluler pada umumnya. (Alicia Sinsuw ; 2013 : 2)

II.10.2. *Android OS*

Android OS adalah sistem operasi yang berbasis *Linux*, sistem operasi *open source*. Selain *Android Software Development Kit (SDK)* untuk pengembangan aplikasi, *android* juga tersedia bebas dalam bentuk sistem operasi. Hal ini yang menyebabkan *vendor-vendor smartphone* begitu berminat untuk memproduksi *smartphone* dan komputer *tablet* berbasis *Android*. (Alicia Sinsuw ; 2013 : 2)

II.10.3. *Android SDK (Software Development Kit)*

Android SDK adalah *tools API (Application Programming Interface)* yang diperlukan untuk mengembangkan aplikasi pada *platform Android* menggunakan bahasa pemrograman *Java*. Beberapa fitur-fitur *Android* yang paling penting adalah mesin *Virtual Dalvik* yang dioptimalkan untuk perangkat *mobile*, *integrated browser* berdasarkan *engine open source WebKit*, Grafis yang dioptimalkan dan didukung oleh *libraries* grafis 2D, grafis 3D berdasarkan spesifikasi *opengl ES 1.0* (Opsional akselerasi perangkat keras), kemudian *SQLite* untuk penyimpanan data (*database*). Fitur-fitur *android* lainnya termasuk media yang mendukung *audio*, *video*, dan gambar, juga ada *fitur bluetooth*, EDGE, 3G dan WiFi, dengan fitur kamera, GPS, dan kompas. Selanjutnya fitur yang juga turut disediakan adalah lingkungan *Development* yang lengkap dan kaya termasuk perangkat emulator, *tools* untuk *debugging*, profil dan kinerja memori, dan *plugin* untuk IDE *Eclipse*. (Alicia Sinsuw ; 2013 : 2)

II.10.4. AVD (*Android Virtual Device*)

Android Virtual Device merupakan emulator untuk menjalankan aplikasi *android*, yang tampilannya dapat dilihat pada gambar 1. Setiap AVD terdiri dari sebuah profil perangkat keras yang dapat mengatur pilihan untuk menentukan *fitur hardware emulator*. Misalnya, menentukan apakah menggunakan perangkat kamera, apakah menggunakan *keyboard* QWERTY fisik atau tidak, berapa banyak memori internal, dan lain-lain. AVD juga memiliki sebuah pemetaan versi *Android*, maksudnya kita menentukan versi dari *platform Android* akan berjalan pada emulator. Pilihan lain dari AVD, misalnya menentukan *skin* yang kita ingin gunakan pada emulator, yang memungkinkan untuk menentukan dimensi layar, tampilan, dan sebagainya. Kita juga dapat menentukan *SD Card virtual* untuk digunakan dengan di emulator. (Alicia Sinsuw ; 2013 : 2)



Gambar II.5. Tampilan AVD
Sumber : (Alicia Sinsuw ; 2013 : 3)

II.11. UML (*Unified Modelling Language*)

Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. 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 OOD (Object-Oriented Design)*, Jim Rumbaugh *OMT (Object Modeling Technique)*, dan Ivar Jacobson *OOSE (Object-Oriented Software Engineering)*. 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

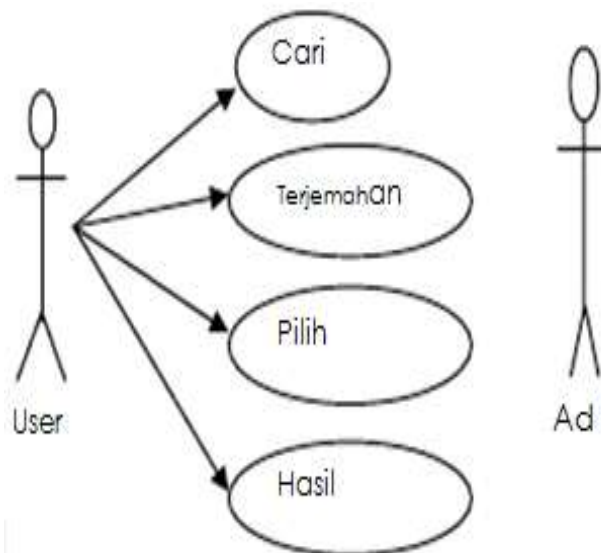
masa perang metodologi (*method war*) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan. Dimulai pada bulan Oktober 1994 *Booch, Rumbaugh dan Jacobson*, yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan memelopori 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. (Yuni Sugiarti ; 2013 : 33)

Dalam pembuatan skripsi ini penulis menggunakan diagram *Use Case* yang terdapat di dalam UML. Adapun maksud dari *Use Case* Diagram diterangkan dibawah ini.

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. (Yuni Sugiarti ; 2013 : 41)



Gambar II.6. Use Case Diagram
Sumber : (Junaedi Siregar ; 2013 : 76)

2. Class Diagram

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

Simbol	Deskripsi
 <p>Package</p>	Package merupakan sebuah bungkus dari satu atau lebih kelas
 <p>Operasi</p>	Kelas pada struktur sistem
 <p>Antarmuka / interface</p>	sama dengan konsep interface dalam pemrograman berorientasi objek
 <p>Asosiasi</p>	relasi antar kelas dengan makna umum, asosiasi biasanya juga disertai dengan multiplicity
 <p>Asosiasi berarah/directed asosiasi</p>	relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan multiplicity
 <p>Generalisasi</p>	relasi antar kelas dengan makna generalisasi-spesialisasi (umum-khusus)
 <p>Kebergantungan / defedency</p>	relasi antar kelas dengan makna kebergantungan antar kelas
 <p>Agregasi</p>	relasi antar kelas dengan makna semua-bagian (whole-part)

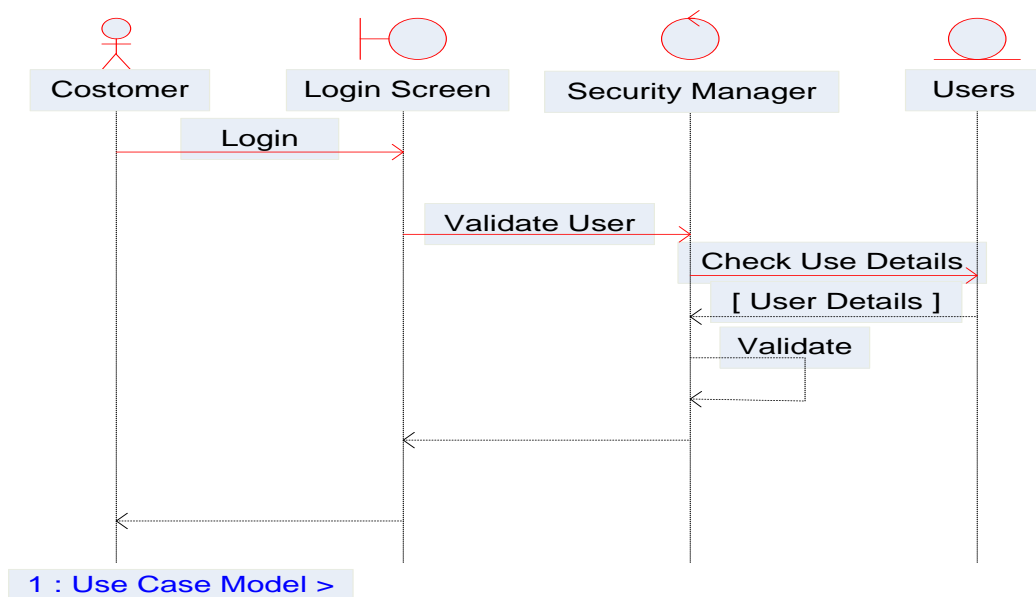
Gambar II.7. Class Diagram

Sumber : (Yuni Sugiarti ; 2013 : 59)

3. Sequence Diagram

Diagram *Sequence* menggambarkan kelakuan/prilaku objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram *sequence* maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

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



Gambar II.8. Contoh Sequence Diagram
 Sumber : (Yuni Sugiarti ; 2013 : 63)

4. Activity Diagram

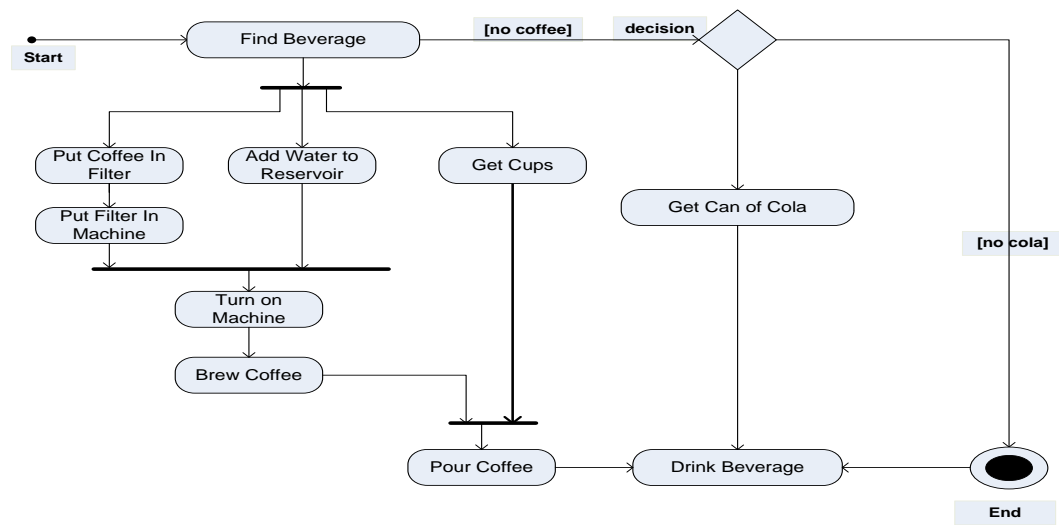
Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

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.



Gambar II.9. Activify Diagram
Sumber : (Yuni Sugiarti ; 2013 : 76)