BAB II

LANDASAN TEORI

Bab ini akan memberikan penjelasan tentang teori yang mendukung dalam pembuatan Penerapan Enkripsi Rivest Code 6 (RC6) dan Triple DES (3DES) Pada Data Inventory. Teori-teori ini dimaksudkan untuk memudahkan seseorang dalam mempelajari aplikasi ini. Sehingga dalam penyampaian informasinya akan tercapai.

Dalam pembuatan aplikasi ini dibutuhkan beberapa perangkat pendukung diantaranya dapat dibedakan menjadi dua, yaitu pendukung kebutuhan perangkat lunak dan pendukung kebutuhan perangkat keras.

Perangkat lunak yang dibutuhkan yaitu terdiri dari Microsoft Visual Studio .NET 2005 sebagai IDE dan MySQL sebagai databasenya.

2.1 Konsep Dasar Kriptografi

Kriptografi berasal dari bahasa Yunani yaitu *kryptos* yang artinya "yang tersembunyi" dan *graphein* yang artinya "tulisan", jadi kriptografi adalah seni dan ilmu untuk menjaga keamanan data. Dan ahlinya disebut sebagai *cryptographer*. *Cryptanalst* merupakan orang yang melakukan *cryptanalysis*, yaitu seni dan ilmu untuk membuka *ciphertext* menjadi *plaintext* tanpa melalui cara yang seharusnya. Data yang dapat dibaca disebut *plaintext* dan teknik untuk membuat data tersebut menjadi tidak dapat dibaca disebut *enkripsi*. Data hasil dari enkripsi disebut *ciphertext*, dan proses untuk mengembalikan *ciphertext* menjadi *plaintext* disebut *dekripsi*. Cabang matematika yang mencakup kriptografi dan *cryptanalysis* disebut *cryptology* dan pelakunya disebut *cryptologist*. Sistem kriptografi atau *cryptosystem* adalah sebuah algoritma ditambah semua kemungkinan *plaintext*, *ciphertext* dan kunci[1]. Dalam sistem ini, seperangkat parameter yang menentukan transformasi pengkodean tertentu disebut suatu set kunci. Proses enkripsi dan dekripsi diatur oleh satu atau beberapa kunci kriptografi. Secara umum, kunci-kunci yang digunakan untuk proses enkripsi dan dekripsi tidak perlu identik,

[1] Scheier, Bruce. *Applied Cryptography, Second Edition*. New York: John Wiley & Son, 1996.

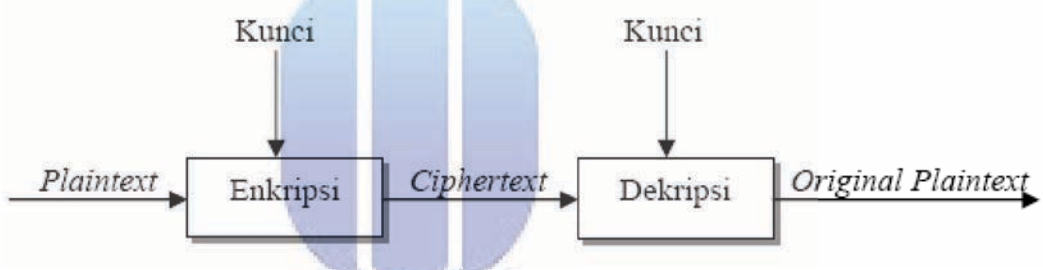
tergantung pada sistem yang digunakan. Setiap algoritma kriptografi terdiri algoritma enkripsi (E) dan algoritma dekripsi (D). Dasar matematis yang mendasari proses enkripsi dan dekripsi adalah relasi antara dua himpunan yaitu himpunan yang berisi elemen *plaintext* dan himpunan yang berisi elemen *ciphertext*. Enkripsi dan dekripsi merupakan fungsi tranformasi antara dua himpunan tersebut. Secara umum dapat digambarkan secara matematis sebagai berikut:

Ek(P) = C (Proses Enkripsi)

Dk(C) = P (Proses Dekripsi)

Dk(E(P)) = P (Proses Dekripsi)

Dalam proses tersebut, *plaintext* disandikan dengan P dengan suatu kunci K lalu dihasilkan pesan C. Pada proses dekripsi, C diuraikan dengan menggunakan kunci K sehingga menghasilkan M yang sama dengan sebelumnya.



Gambar 2.1 Cryptosystem

Setiap *cryptosystem* yang baik memiliki karakteristik sebagai berikut:

a. Keamanan sistem terletak pada kerahasiaan kunci dan bukan pada kerahasian algoritma yang digunakan.
b. *Cryptosystem* yang baik memiliki ruang kunci (*keyspace)* yang besar.
c. *Cryptosystem* yang baik akan menghasilkan *ciphertext* yang terlihat acak dalam seluruh test statistik yang dilakukan.
d. *Cryptosystem* yang baik mampu menahan seluruh serangan yang telah dikenal sebelumnya.

Namun demikian, perlu diperhatikan bahwa bila suatu *cryptosystem* berhasil memenuhi seluruh karakteristik di atas, belum tentu merupakan sistem yang baik. Banyak *cryptosystem* lemah yang terlihat baik pada awalnya. Kadang kala untuk

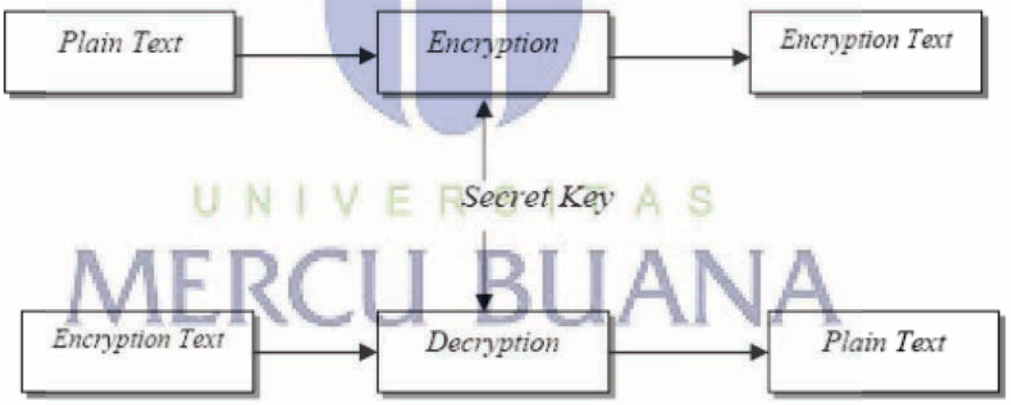
menunjukkan bahwa suatu *cryptosystem* kuat atau baik dapat dilakukan dengan menggunakan pembuktian matematika[2].

2.2 Algoritma Kriptografi

Perkembangan algoritma kriptografi dapat kita bagi menjadi dua, yaitu:[3]

- Kriptografi Klasik
- Kriptografi Modern

Algoritma modern selain memfokuskan diri pada tingkat kesulitan algoritma juga pada kunci yang digunakan. Macam-macam algoritma menurut kuncinya adalah algoritma simetris dan algoritma asimetris. Algoritma simetris disebut juga sebagai algoritma konvensional, yaitu algoritma yang menggunakan kunci yang sama untuk proses enkripsi dan dekripsinya. Keamanan algoritma simetris tergantung pada kuncinya. Algoritma simetris sering juga disebut algoritma kunci rahasia, algoritma kunci tunggal atau algoritma satu kunci. Dua kategori yang termasuk pada algoritma simetris ini adalah algoritma *block cipher* dan *stream cipher.*



Gambar 2.2 Algoritma kriptografi Simetris

Algoritma *block cipher* adalah algoritma yang masukan dan keluarannya berupa satu *block*, dan setiap *block*nya terdiri dari banyak bit. 3DES merupakan salah satu modeoperasi enkripsi *block cipher*[4].

[2] Wahana. *Memahami Model Enkripsi dan Security Data*. Yogyakarta, Andi Offset, 2003.
[3] Kurniawan, Yusuf, Ir. *Kriptografi Keamanan Internet dan Jaringan Komunikasi.* Bandung: Informatika Bandung, 2004.

Algoritma *Stream cipher* adalah *cipher* yang berasal dari hasil XOR antara bit *plaintext* dengan setiap bit kuncinya. *Stream cipher* sangat rawan terhadap *attack* pembalikan bit. RC6 merupakan salah satu mode operasi enkripsi *stream cipher.*

2.3 Algoritma RC 6

Algoritma RC6 merupakan salah satu kandidat *Advanced Encryption Standard* (AES) yang diajukan oleh *RSA Laboratoriest* kepada NIST. Dirancang oleh Ronald L Rivest, M.J.B. Robshaw, R. Sidney dan Y.L. Yin, algoritma ini merupakan pengembangan dari algoritma sebelumnya yaitu RC5 dan telah memenuhi semua kriteria yang diajukan oleh NIST. Algoritma RC6 adalah versi yang dilengkapi dengan beberapa parameter, sehingga dituliskan sebagai RC6-w/r/b, dimana parameter w merupakan ukuran kata dalam satuan bit, r adalah bilangan bulat bukan negatif yang menunjukan banyaknya iterasi selam proses enkripsi, dan b menunjukan ukuran kunci enkripsi dalam *byte*. Ketika algoritma ini masuk sebagai kandidat AES, maka ditetapkan nilai parameter w = 32, r = 20 dan b bervariasi antara 16, 24, dan 32 *byte*[5]. RC6-w/r/b memecah *block* 128 bit menjadi 4 buah *block* 32 bit, dan mengikuti enam aturan operasi dasar sebagai berikut :

A + B Operasi penjumlahan bilangan integer.

A – B Operasi pengurangan bilangan integer.

A B Operasi *exclusive*-OR (XOR)

A x B Operasi perkalian bilangan integer.

A<<<B A dirotasikan ke kiri sebanyak variabel kedua (B)

A>>>B A dirotasikan ke kanan sebanyak variabel kedua (B)

2.3.1. Proses Enkripsi

Karena RC6 memecah *block* 128 bit menjadi 4 buah *block* 32 bit, maka algoritma ini bekerja dengan 4 buah register 32-bit A, B, C, D. *Byte* yang pertama dari

---

[4] Kurniawan, Yusuf, Ir. *Kriptografi Keamanan Internet dan Jaringan Komunikasi.* Bandung: Informatika Bandung, 2004.

[5] Abdurohman, Maman. *Analisis Performansi Algoritma Kriptografi RC6.* Fakultas Teknologi Informasi, T. Elektro, ITB. Bandung, 2002.

plaintext atau *ciphertext* ditempatkan pada *byte* A, sedangkan *byte* yang terakhirnya ditempatkan pada *byte* D. Dalam prosesnya akan didapatkan (A, B, C, D) = (B, C, D, A) yang diartikan bahwa nilai yang terletak pada sisi kanan berasal dari register disisi kiri[6]. Berikut ini adalah algoritma enkripsi RC6:

```
B = B + S[ 0 ]
D = D + S[ 1 ]
for i = 1 to 20 do
{
t = ( B x ( 2B + 1 ) )
<<< 5
u = ( D x ( 2D + 1 ) )
<<< 5
A = ( ( A XOR t ) <<< u ) + S[ 2i ]
C = ( ( C XOR u ) <<< t )+ S[ 2i + 1 ]
(A, B, C, D) = (B, C, D, A)
}
A = A + S[ 42 ]
C = C + S[ 43 ]
```

Algoritma RC6 menggunakan 44 buah sub kunci yang dibangkitkan dari kunci dan dinamakan dengan S[0] hingga S[43]. Masing-masing sub kunci panjangnya 32 bit. Proses enkripsi pada algoritma RC6 dimulai dan diakhiri dengan proses *whitening* yang bertujuan untuk menyamarkan iterasi yang pertama dan yang terakhir dari proses enkripsi dan dekripsi. Pada proses *whitening* awal, nilai B akan dijumlahkan dengan S[0], dan nilai D dijumlahkan dengan S[i]. Pada masing-masing iterasi pada RC6 menggunakan 2 buah sub kunci. Sub kunci pada iterasi yang pertama menggunakan S[2] dan S[3], sedangkan iterasi-iterasi berikutnya menggunakan sub-sub kunci lanjutannya. Setelah iterasi ke-20 selesai, dilakukan proses *whitening* akhir dimana nilai A dijumlahkan dengan S[42], dan nilai C dijumlahkan dengan S[43]. Setiap iterasi pada algoritma RC6 mengikuti aturan sebagai berikut, nilai B dimasukan ke dalam fungsi f, yang didefinisikan sebagai f(x) =x(2x+1) , kemudian diputar kekiri sejauh lg-w atau 5 bit. Hasil yang didapat pada proses ini dimisalkan sebagai u. Nilai u kemudian di XOR dengan C dan hasilnya menjadi nilai C. Nilai t juga digunakan sebagai acuan bagi C untuk memutar nilainya kekiri. Begitu pula dengan nilai u, juga digunakan sebagai acuan bagi nilai A untuk melakukan proses pemutaran kekiri. Kemudian sub kunci S[2i] pada iterasi dijumlahkan dengan A, dan sub kunci S[2i+1] dijumlahkan dengan C. keempat bagian dari *block* kemudian akan dipertukarkan

[6] Abdurohman, Maman. *Analisis Performansi Algoritma Kriptografi RC6.* Fakultas Teknologi Informasi, T. Elektro, ITB. Bandung, 2002.

dengan mengikuti aturan, bahwa nilai A ditempakan pada D, nilai B ditempatkan pada A, nilai C ditempatkan pada B, dan nilai (asli) D ditempatkan pad C. demikian iterasi tersebut akan terus berlangsung hingga 20 kali.

2.3.2. Proses Deskripsi

Proses dekripsi *ciphertext* pada algoritma RC6 merupakan pembalikan dari proses enkripsi. Pada proses *whitening*, bila proses enkripsi menggunakan operasi penjumlahan, maka pada proses dekripsi menggunakan operasi pengurangan. Sub kunci yang digunakan pada proses *whitening* setelah iterasi terakhir diterapkan sebelum iterasi pertama, begitu juga sebaliknya sub kunci yang diterapkan pada proses *whitening* sebelum iterasi pertama digunakan pada *whitening* setelah iterasi terkahir. Akibatnya, untuk melakukan dekripsi, hal yang harus dilakukan semata-mata hanyalah menerapkan algoritma yang sama dengan enkripsi, dengan tiap iterasi menggunakan sub kunci yang sama dengan yang digunakan pada saat enkripsi, hanya saja urutan sub kunci yang digunakan terbalik[i]. Berikut ini adalah algoritma deskripsi RC6:

```
C – C - S[ 43 ]
A – A - S[ 42 ]
for i – 20 downto 1 do
{
(A, B, C, D) – (D, A, B, C)
u – ( D x ( 2D + 1 ) )
<<< 5
t – ( B x ( 2B + 1 ) )
<<< 5
C – ( ( C - S[ 2i + 1 ] )
>>> t) Å u
A – ( ( A - S[ 2i ] ) >>> u
) Å t
}
D – D - S[ 1 ]
B = B - S[ 0 ]
```

2.4 Algoritma 3DES (Triple Data Encrytion Standard)

Algoritma enkripsi yang paling banyak digunakan di dunia adalah DES yang telah diadopsi oleh NIST (*Nasional Institude of Standard and Tecnology*) sebagai standard pengolahan informasi Federal AS. Data dienkrip dalam *block-block* 64 bit menggunakan kunci 56 bit. Algoritma DES berasal dari algoritma Lucifer buatan IBM. Algoritma ini ditawarkan kepada NIST dan menjadi DES tahun 1977. Akan tetapi terdapat dua masalah besar pada algoritma ini. Pertama,

kunci yang hanya 56 bit, sehingga sangat rawan terhadap serangan *brute force*. Kedua, desain struktur internal DES dimana bagian subsitusinya (S-box) masih dirahasiakan

2.4.1. Data Encryption Standard

DES beroperasi pada ukuran blok 64-bit. DES mengenkripsikan 64-bit plainteks menjadi 64-bit cipherteks dengan menggunakan 56-bit kunci internal yang dibangkitkan dari kunci eksternal yang panjangnya 64-bit.

2.4.2. Proses Kunci

Kunci eksternal yang diinputkan akan diproses untuk mendapatkan 16 kunci internal. Pertama, Kunci eksternal yang panjangnya 64-bit disubstitusikan pada matriks permutasi kompresi PC-1. Dalam permutasi ini, setiap bit kedelapan (*parity bit*) dari delapan byte diabaikan. Hasil permutasi panjangnya menjadi 56-bit, yang kemudian dibagi menjadi dua bagian, yaitu kiri (*C0*) dan kanan (*D0*) masing-masing panjangnya 28-bit. Kemudian, bagian kiri dan kanan melakukan pergeseran bit pada setiap putaran sebanyak satu atau dua bit tergantung pada tiap putaran. Pada proses enkripsi, bit bergeser kesebelah kiri (*left shift*). Sedangkan untuk proses dekripsi, bit bergeser kesebelah kanan (*right shift*). Setelah mengalami pegeseran bit, *Ci* dan *Di* digabungkan dan disubstitusikan pada matriks permutasi kompresi dengan menggunakan matriks PC-2, sehingga panjangnya menjadi 48-bit. Proses tersebut dilakukan sebanyak 16 kali secara berulang-ulang.

2.4.3. Proses Enkripsi

Plainteks yang diinputkan pertama akan disubatitusikan pada matriks permutasi awal (*initial permutation*) atau IP panjangnya 64-bit. Kemudian dibagi menjadi dua bagian, yaitu kiri (*L*) dan kanan (*R*) masing-masing panjangnya menjadi 32-bit. Kedua bagian ini masuk ke dalam 16 putaran DES. Satu putaran DES merupakan model jaringan Feistel, secara matematis jaringan Feistel dinyatakan sebagai berikut:

$$Li = Ri\text{-}1 \quad ; 1 \leq i \leq 16 \qquad (4.1)$$

$$Ri = Li\text{-}1 + f(Ri\text{-}1, ki) \qquad (4.2)$$

Bagian *R* disubstituaikan pada fungsi ekspansi panjangnya menjadi 48-bit kemudian di-XOR-kan dengan kunci internal yang sudah diproses sebelumnya pada proses pembangkitan kunci (pada putaran pertama menggunakan kunci internal pertama, dan seterusnya). Hasil XOR kemudian disubstitusikan pada *S-box* yang dikelompokkan menjadi 8 kelompok, masing-masing 6-bit hasilnya menjadi 4-bit. Kelompok 6-bit pertama menggunakan *S1*, kelompok 6-bit kedua menggunakan *S2*, dan seterusnya. Setelah proses *S-box* tersebut panjangnya menjadi 32-bit. Kemudian disubstitusikan lagi pada matriks permutasi *P-box*, kemudian di-XOR-kan dengan bagian *L*. Hasil dari XOR tersebut disimpan untuk bagian *R* selanjutnya. Sedangkan untuk bagian *L* diperoleh dari bagian *R* yang sebelumnya. Proses tersebut dilakukan 16 kali. Setelah 16 putaran selesai, bagian *L* dan *R* digabungkan dan disubstitusikan pada matriks permutasi awal balikan (*invers initial permutation*) atau IP-1, hasilnya merupakan cipherteks 64-bit.

2.4.4. Proses Dekripsi

Proses dekripsi terhadap cipherteks merupakan kebalikan dari proses enkripsi. DES menggunakan algoritma yang sama untuk proses enkripsi dan dekripsi. Jika pada proses enkripsi urutan kunci internal yang digunakan adalah *k1, k2, ..., k16* maka pada proses dekripsi urutan kunci internal yang digunakan adalah *k16, k15, ..., k1*.

2.5 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 bahasabahasa

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 [1], metodologi coad [2], metodologi OOSE [3], metodologi OMT [4], metodologi shlaer-mellor [5], metodologi wirfs-brock [6], 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.

3.5.1. Konseps Dasar UML

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

Tabel 2.1 Konsep dasar UML

Major Area	View	Diagrams	Main concepts
structural	Static view	Class diagram	Class,association, generalization,dependency ,realization,interface
	Use case view	Use case diagram	Use case, actor,association,extend, include,use case generalization

	Implementation view	Component diagram	Component,interface, depedency, realization
	Deployment view	Deployment diagram	Node,component, dependency,location
dynamic	State machine view	Statechart diagram	State,event,transition, Action
	Activity view	Activity diagram	State,activity,completion transition,fork,join
	Interaction view	Sequence diagram	Interaction,object,message ,activation
		Collaboration diagram	Collaboration,interaction, collaboration role,message
Model management	Model management view	Class diagram	Package,subsystem,model
extensibility	all	All	Constraint,stereotype, tagged values

Abstraksi konsep dasar UML yang terdiri dari *structural classification*, *dynamic behavior*, dan *model management*, bisa kita pahami dengan mudah apabila kita melihat gambar diatas dari *Diagrams*. *Main concepts* bisa kita pandang sebagai term yang akan muncul pada saat kita membuat diagram. Dan view adalah kategori dari diagaram tersebut.

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

- *Use case diagram*
- *Class diagram*
- *State chart diagram*
- *Activity diagram*
- *Sequence diagram*
- *Collaboration diagram*
- *Component diagram*

- *Deployment diagram*

3.5.2. 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 jurnal, dan sebagainya. Seorang/ sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

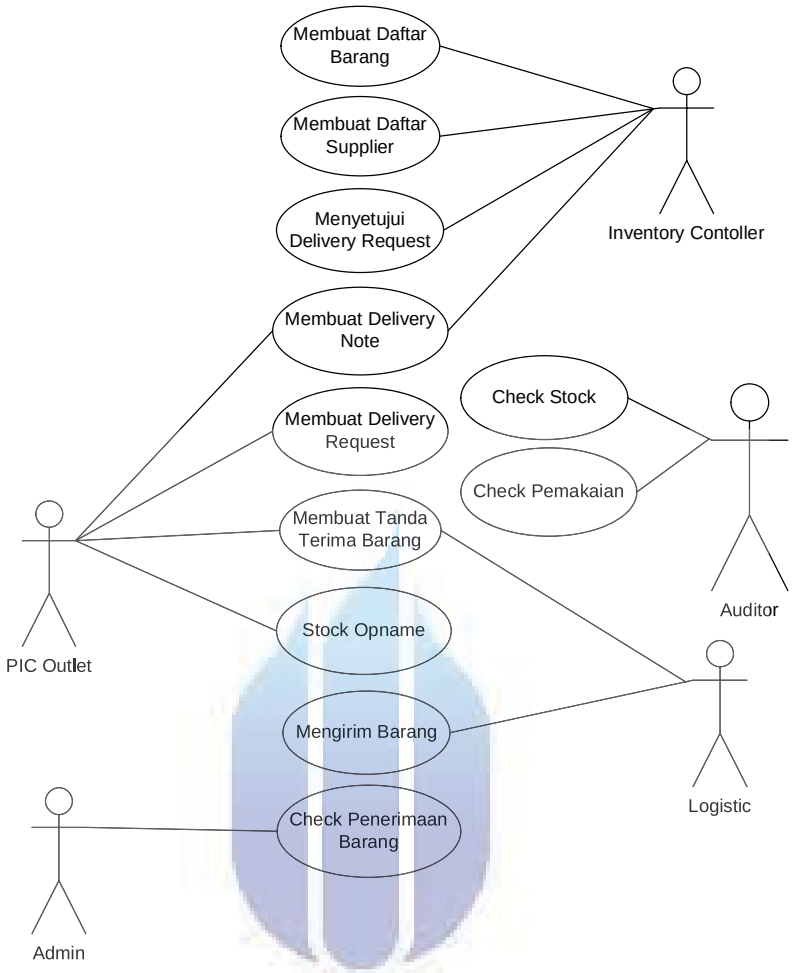
Use case diagram dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem.

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* :



Gambar 2.3 ***Use case diagram***

3.5.3. Class Diagram

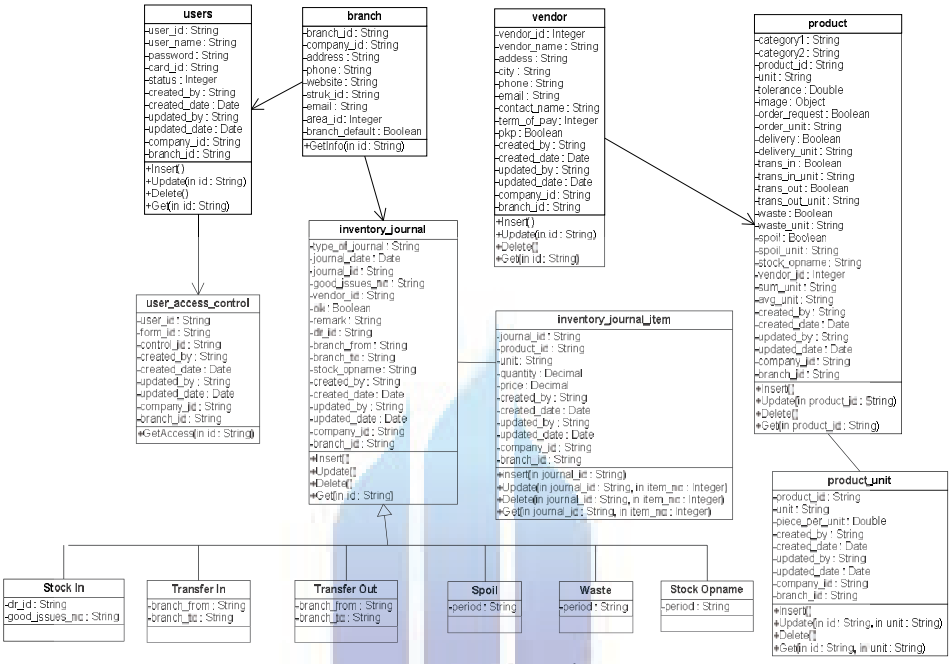
Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *Class diagram* menggambarkan struktur dan dekripsi *class, package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.

Class memiliki tiga area pokok, yaitu Nama (dan stereotype), Atribut dan Metoda. Atribut dan metoda dapat memiliki salah satu sifat berikut :

• *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan

• *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya

• *Public*, dapat dipanggil oleh siapa saja



Gambar 2.4 *Class Diagram*

3.5.4. Activity Diagram

Activity diagrams 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.

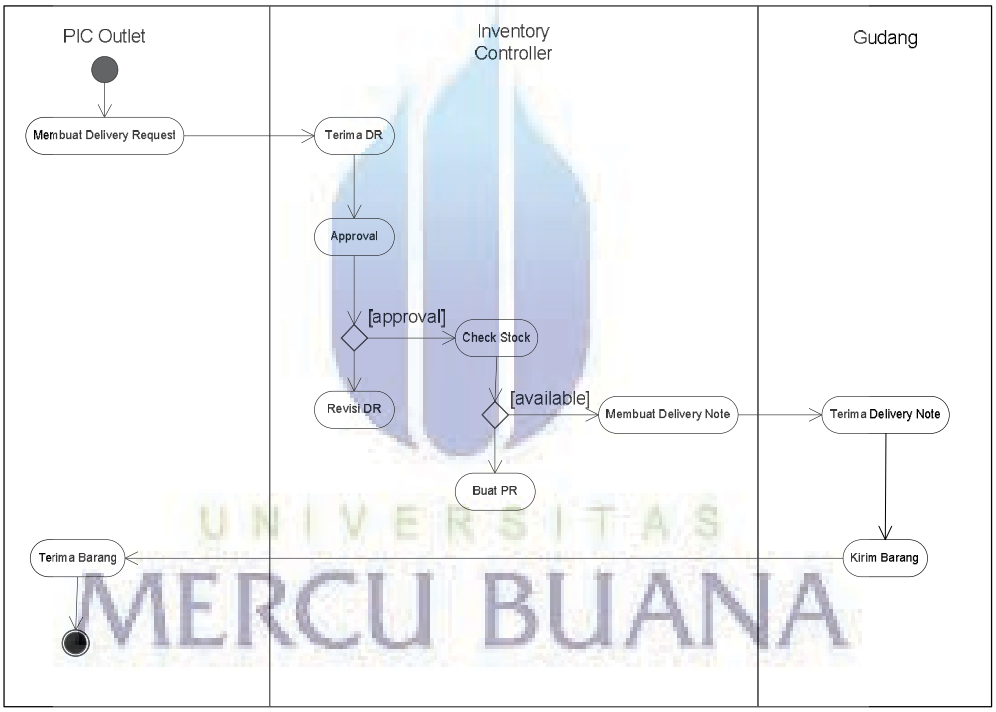
Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan behaviour internal sebuah sistem (dan interaksi 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* :



Gambar 2.5 *Activity Diagram*

3.5.5. Sequence Diagram

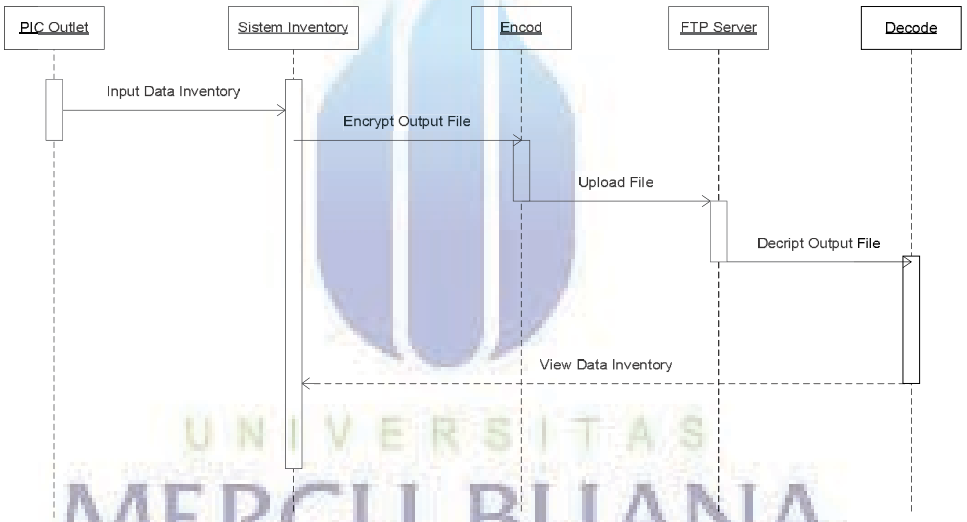
Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

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

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

Contoh *sequence diagram* :



Gambar 2.6 *Sequence Diagram*

[i] Abdurohman, Maman. *Analisis Performansi Algoritma Kriptografi RC6.* Fakultas Teknologi Informasi, T. Elektro, ITB. Bandung, 2002.