UNIVERSITAS INDONESIA

IMPLEMENTASI DAN ANALISA PENGIRIMAN DATA MENGGUNAKAN ALGORITMA KRIPTOGRAFI RSA PADA SISTEM EUCALYPTUS PRIVATE CLOUD IAAS

SKRIPSI

Dyani Mustikarini

0806316783

FAKULTAS TEKNIK UNIVERSITAS INDONESIA

DEPARTEMEN TEKNIK ELEKTRO

TEKNIK KOMPUTER

DEPOK

JUNI 2012

UNIVERSITAS INDONESIA

IMPLEMENTASI DAN ANALISA PENGIRIMAN DATA MENGGUNAKAN ALGORITMA KRIPTOGRAFI RSA PADA SISTEM EUCALYPTUS PRIVATE CLOUD IAAS

SKRIPSI

Diajukan sebagai salah satu syarat untuk memperoleh gelar sarjana

Dyani Mustikarini

0806316783

FAKULTAS TEKNIK UNIVERSITAS INDONESIA

DEPARTEMEN TEKNIK ELEKTRO

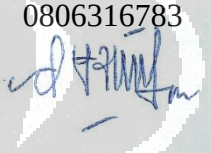
TEKNIK KOMPUTER

DEPOK

JUNI 2012

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri, dan semua sumber baik yang dikutip maupun dirujuk telah saya nyatakan dengan benar.

Nama	:	Dyani Mustikarini
NPM	:	0806316783
Tanda Tangan	:	
Tanggal	:	29 Juni 2012

HALAMAN PENGESAHAN

Seminar ini diajukan oleh :
Nama : Dyani Mustikarini
NPM : 0806316783
Program Studi : Teknik Komputer
Judul Seminar : Implementasi dan Analisa Pengiriman Data Menggunakan Algoritma Kriptografi RSA pada Sistem Eucalyptus Private Cloud IaaS

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang dilakukan untuk memperoleh gelar Sarjana Teknik pada program studi Teknik Komputer, Fakultas Teknik, Universitas Indonesia

DEWAN PENGUJI

Pembimbing : Dr. Ir. Anak Agung Putri Ratna, M.Eng (.....)

Penguji 1 : Ir. Endang Sriningsih, MT (.....)

Penguji 2 : Prima Dewi Purnamasari, ST., MT., MSc (.....)

Ditetapkan di : Depok
Tanggal : 29 Juni 2012

KATA PENGANTAR

Puji syukur saya panjatkan kehadirat Allah SWT karena atas segala rahmat dan hidayah-Nya saya dapat menyelesaikan skripsi ini. Saya menyadari bahwa skripsi ini tidak akan selesai tanpa bantuan dari berbagai pihak. Mulai dari pencarian tema, studi literatur hingga proses penyusunan dari buku skripsi ini, saya ingin mengucapkan terima kasih kepada:

1. Ibu Dr. Ir. Anak Agung Putri Ratna M.Eng selaku pembimbing skripsi saya. Terima kasih atas pengarahan, koreksi, dukungan, dan waktu yang telah diberikan selama saya mengerjakan skripsi ini.
2. Orang tua dan keluarga saya yang telah memberikan bantuan dukungan moral, do'a serta materil.
3. Henry Artajaya, Alifandi Yudistira, Ahmad Shaugi, Asep Sunandar, Noni Elysa dan Syamsudin Daniel, selaku teman bimbingan saya atas kerja samanya selama masa bimbingan.
4. Teman-teman di *program* studi Teknik Komputer dan Teknik Elektro angkatan 2008 yang tiada hentinya mendukung saya baik secara langsung maupun tidak langsung.
5. Seluruh keluarga besar Civitas Akademika Departemen Teknik Elektro Fakultas Teknik Universitas Indonesia.
6. Rendy Harya Putra yang telah memberikan dukungan dalam pengerjaan buku skripsi ini.

Akhir kata, semoga Allah SWT berkenan membalas kebaikan semua pihak yang telah membantu. Semoga skripsi ini bermanfaat bagi perkembangan ilmu pengetahuan.

Depok, 29 Juni 2012

Penulis,

Dyani Mustikarini

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPERLUAN AKADEMIS

Sebagai civitas akademik Universitas Indonesia, saya yang bertanda tangan dibawah ini :

Nama : Dyani Mustikarini

NPM : 0806316783

Program Studi : Teknik Komputer

Departemen : Teknik Elektro

Fakultas : Teknik

Jenis Karya : Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Non-eksklusif** (*Non-exclusive Royalti-Free Right*) atas karya ilmiah yang berjudul :

IMPLEMENTASI DAN ANALISA PENGIRIMAN DATA MENGGUNAKAN ALGORITMA KRIPTOGRAFI RSA PADA SISTEM EUCALYPTUS PRIVATE CLOUD IAAS

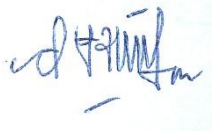
beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Non-Eksklusif ini, Universitas Indonesia berhak menyimpan, mengalihmediakan / mengalihformatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan mempublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok

Pada tanggal : 29 Juni 2012

Yang menyatakan,



(Dyani Mustikarini)

ABSTRAK

Nama : Dyani Mustikarini
Program Studi : Teknik Komputer
Judul : Implementasi dan Analisa Pengiriman Data Menggunakan Algoritma Kriptografi RSA pada Sistem Eucalyptus Private Cloud IaaS

Skripsi ini berisi mengenai konsep dasar, perancangan dan implementasi enkripsi data dengan RSA yang diterapkan pada private cloud Infrastucture as a Service (IaaS). Tujuan dari skripsi ini menganalisa keamanan pengiriman data dan waktu dari implementasi kriptografi RSA pada sistem Eucalyptus private cloud. Pengiriman data pada sistem virtualisasi private cloud membutuhkan enkripsi untuk mengantisipasi serangan dari *man-in-the-middle* sehingga penyerang tidak mengetahui isi data dengan mudah. Hasil penelitian menunjukkan bahwa waktu eksekusi program RSA dipengaruhi oleh ukuran data dan nilai kunci RSA yang dibangkitkan. Peningkatan ukuran data akan mempengaruhi peningkatan waktu eksekusi program RSA. Peningkatan waktu eksekusi untuk format .txt sebesar 31,44%, untuk format .doc sebesar 24,83% dan untuk format .pdf sebesar 24,85%. Nilai d untuk kunci privat RSA yang besar akan sangat mempengaruhi waktu eksekusi karena membutuhkan waktu dekripsi yang lebih lama. Sedangkan nilai e yang besar untuk kunci publik RSA tidak terlalu signifikan mempengaruhi waktu enkripsi menjadi lebih lama namun tetap berkontribusi terhadap waktu eksekusi RSA. Keamanan pengiriman data pada sistem private cloud dibutuhkan terutama dengan RSA 2048 bit dan sistem padding, namun pada skripsi ini hanya digunakan enkripsi plain RSA.

Kata kunci :
Cloud computing, Private Cloud, Cloud IaaS, RSA, Kriptografi, Eucalyptus

ABSTRACT

Name : Dyani Mustikarini
Study Program : Computer Engineering
Title : Implementation and Analysis Data Transfer Using RSA Cryptography Algorithm on Eucalyptus Private Cloud IaaS System

This thesis contains about fundamental concept, the design and the implementation of data encryption using RSA which is applied on private cloud Infrastucture as a Service (IaaS). The purposes of this thesis are to analyze the the data transfer security and the time of RSA cryptography appliance on Eucalyptus private cloud system. Secret data transfer on private cloud virtualization requires encryption in order to anticipated the attack from *man-in-the-middle* so that the attacker won't know the contents of data easily.The result of this research prove that RSA execution time influented by the size of data and the value of the generated RSA keys. Data size increment will influence the execution times of RSA. The increment time for .txt is 31,44%, increment time for .doc is 24,83%, and increment time for .pdf is 24,85%. Large values of d for RSA private key greatly affect the execution time because need a longer decryption time. However, the large value of e for public keys isn't influence the encryption time significantly but still contributes the execution time. The security of data transfer on private cloud system is needed especially using RSA 2048 bit and padding system appliance,however this thesis only implement plain RSA encryption.

Key words:
Cloud computing, Private Cloud, Cloud IaaS, RSA, Cryptography, Eucalyptus

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR ISTILAH

A

ATA over Ethernet : protocol jaringan pada lapisa ke 2 yang didesain untuk akses divais penyimpanan dengan performa tinggi melalui jaringan Ethernet yang digunakan untuk membangun Storage Area Network.

C

Cloud computing : sistem distribusi yang terdiri dari komponen yang terinterkoneksi dan virtualisasi komputer yang tersedia secara dinamis sebagai sumberdaya komputasi.

H

Hypervisor : virtualisasi hardware yang mengizinkan OS jamak berjalan pada satu komputer host secara bersamaan.

Hybrid cloud : private cloud yang terhubungkan oleh satu atau lebih layanan eksternal cloud dengan pengaturan yang bersifat sentral.

I

Instances : Mesin virtual yang berjalan pada hypervisor dan dikontrol dibawah UEC/Eucalyptus.

M

Multitenancy : kemampuan aplikasi untuk secara otomatis berada dalam kondisi terpartisi dan dapat menangani sejumlah pengguna berbeda.

P

Pay-as-you go : pembayaran untuk sumberdaya yang digunakan secara actual dalam periode tertentu.

Private Cloud : sistem cloud untuk pusat data organisasi internal.

Public Cloud : sistem cloud yang menggunakan akses Internet dan dikelola oleh suatu penyedia layanan cloud (cloud vendor).

S

Snapshots : volume yang dibuat pada blok penyimpanan yang disimpan pada Walrus dan digunakan sebagai titik awal untuk volume blok storage/penyimpanan data yang baru. Satu snapshot dapat digunakan untuk membuat banyak volume.

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Teknologi *cloud computing* memiliki kemampuan yang fleksibel, rendah biaya dan skalabilitas yang tinggi dalam berbagi sumber daya dan komputasi. Kemampuan ini yang membuat *cloud computing* saat ini menjadi isu teknologi yang hangat diperbincangkan, terutama masalah keamanan dari *cloud computing* itu sendiri. Keamanan *cloud computing* perlu diteliti karena semakin banyak *cracker* yang menyerang dan mencuri data-data privasi milik organisasi tertentu. Selain itu, *cloud computing* dapat memanfaatkan akses Internet sebagai koneksi klien ke server sehingga memungkinkan adanya *cyber attacks*. Masalah ini yang menjadi salah satu dasar penelitian skripsi ini mengenai keamanan dari *cloud computing*. Penelitian pada skripsi ini akan menggunakan keamanan enkripsi RSA (Rivest Shamir Adleman) pada virtualisasi *cloud computing* karena RSA merupakan salah satu jenis *key-exchange* yang paling sering digunakan karena sifatnya yang aman akibat sulitnya memecahkan pemfaktoran bilangan yang besar.

Menurut US NIST (National Institute of Standards and Technology), *cloud computing* didefinisikan sebagai suatu model untuk memberikan kenyamanan, akses jaringan *on-demand* untuk berbagi pakai sumberdaya komputasi seperti jaringan, server, storage, aplikasi dan service yang dapat disediakan dan dirilis dengan cepat dengan upaya manajemen minimal atau dengan interaksi penyedia layanan. Adanya interaksi dari user ke penyedia layanan yang mempercayakan data-datanya untuk disimpan pada cloud provider, memungkinkan adanya *man-in-the-middle* yang menyusup ke dalam jaringan dan dapat mengganggu aktivitas cloud.

Salah satu teknologi yang digunakan pada *cloud computing* adalah teknologi virtualisasi, misalnya menggunakan VM (*Virtual Machine*) pada server cloud. Masalah keamanan virtualisasi pada *cloud computing* perlu diteliti karena serangan *cracker* ke server dapat menyebabkan data pada mesin virtual akan terbaca dan dapat menyebabkan masalah lainnya. Jika masalah keamanan ini tidak

diantisipasi, maka akan menyebabkan banyak kerugian bagi sisi penyedia layanan maupun bagi konsumer atau user.

Penyedia cloud biasanya menggunakan teknologi virtualisasi yang dikombinasikan dengan kemampuan *self-service* untuk sumberdaya komputasi melalui infrastuktur jaringan seperti Internet atau jaringan privat. Pada cloud dengan lingkungan VM lebih dari satu memiliki server fisik yang sama sebagai infrastukturnya. Manajemen task seperti manajemen performasi pada lingkungan VM lebih berpotensi terkena serangan dibandingkan dengan server fisik tradisional seperti yang diungkapkan pada literatur jurnal yang mengulas mengenai keamanan level-virtualisasi pada cloud [1]. Potensi serangan yang cukup tinggi untuk cloud pada lingkungan VM menjadi suatu daya tarik sehingga dilakukan penelitian mengenai keamanan pengiriman data pada virtualisasi cloud.

Penelitian mengenai keamanan akses data pada *cloud computing* telah dibahas pada literatur jurnal yang meneliti keamanan akses data menggunakan *publickey-exchange* Diffie Hellman antar penyedia layanan, user dan pemilik data [2]. Menurut jurnal tersebut, algoritma Diffie Hellman memiliki logaritma diskrit yang dapat menghitung serangan *man in the middle* dan setiap sesi yang telah kadaluarsa, algoritma ini akan menukarkan kunci rahasia yang baru. Jurnal ini menjadi dasar penelitian pada skripsi ini untuk menggunakan teknik kriptografi yang lainnya, yaitu menggunakan pertukaran kunci publik RSA. Kemudian, diharapkan penelitian pada skripsi ini dapat menganalisa keamanan dari penggunaan enkripsi RSA untuk pengiriman data pada virtualisasi *cloud computing* dengan model private *IaaS.*

1.2 Tujuan Penelitian

Tujuan penelitian pada skripsi adalah untuk mengimplementasikan sistem virtualisasi pada model *cloud computing IaaS* (Infrastructure as a Service) menggunakan Eucalyptus dengan menerapkan enkripsi RSA serta menganalisa keamanan pengiriman paket dan waktu dari implementasi kriptografi RSA

1.3 Pembatasan Masalah

Sistem pada skripsi ini akan dibatasi pada virtualisasi pada model *cloud computing IaaS* menggunakan Eucalyptus dengan menerapkan sistem keamanan

pengiriman data dengan algoritma enkripsi RSA. Pada skripsi ini diasumsikan jaringan dalam kondisi ideal sehingga tidak memperhitungkan *delay* yang terjadi.

1.4 Metodologi Penelitian

Metodologi penelitian pada skripsi ini adalah

a. Studi literatur, yaitu mencari materi dari buku, jurnal, dan referensi lain seperti media internet yang berkaitan dengan tema yang diambil.
b. Konsultasi kepada pembimbing seminar dan para ahli di bidang networking security.
c. Membuat perancangan sistem.
d. Menerapkan rancangan sistem yang telah dibuat.
e. Uji coba sistem yang telah dibuat dan pengambilan data yang diperlukan.
f. Analisis hasil yang didapatkan dari implementasi terhadap sistem.

1.5 Sistematika Penulisan

Penulisan skripsi ini akan dibagi menjadi beberapa bab agar lebih sistematis, yaitu :

a. Bab 1 : Pendahuluan

   Bab ini terdiri dari latar belakang, batasan masalah, metodologi penelitian, dan sistematika penulisan.

b. Bab 2 : Konsep dasar keamanan pengiriman data pada *virtualisasi private cloud computing Iaas* dengan mengimplementasikan enkripsi RSA.

   Pada bab ini akan menjabarkan konsep dasar dari *cloud computing*, keamanan pada virtualisasi *cloud computing*, dan algoritma enkripsi RSA.

c. Bab 3 : Perancangan dan implementasi enkripsi RSA pada sistem virtualisasi *private cloud computing IaaS*.

   Bab ini menjelaskan rancangan dan implementasi dari sistem virtualisasi *private IaaS cloud computing* dengan menerapkan keamanan pengiriman data menggunakan enkripsi RSA berbasis C++. Kemudian pada bab ini juga akan dijabarkan aliran kerja sistem virtualisasi *cloud computing*

private IaaS dengan menggunakan enkripsi RSA serta flow chart dari program algoritma RSA itu sendiri.

d. Bab 4 : Uji Coba dan Analisa

Pada bab ini akan dideskripsikan mengenai hasil uji coba dari penelitian yang telah dilakukan dan akan dijabarkan analisis dari sistem RSA dan keamanan pengiriman data menggunakan enkripsi RSA pada sistem *private cloud IaaS.*

e. Bab 5 : Kesimpulan

Bab ini berisi penarikan kesimpulan dari skripsi ini.

BAB 2
KONSEP DASAR PENGIRIMAN DATA PADA VIRTUALISASI *PRIVATE CLOUD COMPUTING IAAS* DENGAN MENGIMPLEMENTASIKAN ENKRIPSI RSA

2.1 Definisi *Cloud Computing*

Menurut NIST (National Institute Standards and Technology), *cloud computing* merupakan suatu model yang memberikan kenyamanan dalam akses jaringan *on-demand* untuk berbagi sumberdaya komputasi yang dikonfigurasi (seperti jaringan, server, *storage*, aplikasi dan layanan) yang disediakan dan dirilis dengan cepat dengan upaya manajemen minimal atau dengan interaksi penyedia layanan.

Cloud computing juga dapat didefinisikan sebagai suatu tipe paralel atau sistem distribusi yang terdiri dari koleksi terinterkoneksi dan komputer virtualisasi yang tersedia secara dinamis sebagai satu atau lebih kesatuan sumberdaya komputasi berbasis pada pembangunan SLA (Service Level Agreements) melalui negosiasi antara provider dan konsumen [1].

Cloud computing pada model IT atau lingkungan komputasi dapat menciptakan komponen IT (hardware, software, jaringan dan layanan) maupun sebagai proses disekeliling elemen-elemen tersebut yang secara bersamaan dapat menyediakan pembangunan dan pengiriman layanan *cloud* kepada user melalui Internet atau jaringan privat. Untuk pengertian mengenani layanan *cloud* akan dibahas pada subbab model layanan *cloud computing*.

Cloud computing berdasarkan beberapa atribut berikut ini dapat didefinisikan sebagai suatu istilah dalam dunia IT yang memiliki banyak perbedaan dengan komputasi sebelum era *cloud*. Kelima atribut tersebut adalah [4]

- *Multitenancy* (Berbagi Sumberdaya)

  Pada model komputasi tradisional, sumberdaya terdedikasi seperti fasilitas komputasi yang didedikasikan untuk klien atau owner tunggal. Sedangkan pada model *cloud* dapat berbagi sumberdaya untuk beragam klien pada sumberdaya yang sama pada level jaringan, level *host* dan level aplikasi.

Jadi, *multitenancy* merupakan kemampuan aplikasi untuk secara otomatis berada dalam kondisi terpartisi dan dapat menangani sejumlah pengguna.

- Skalabilitas bersifat *massive*

  Suatu organisasi mungkin memiliki ratusan bahkan ribuan sistem, dengan sistem *cloud* menyediakan kemampuan untuk diperluas hingga 10 kali lipat dari sistem yang telah ada jadi *cloud* dapat memiliki skalabilitas mencapai 10 kali lipat dari ribuan sistem. Selain itu, sistem *cloud* memiliki kemampuan untuk meningkatkan *bandwidth* dan ruang penyimpanan secara besar-besaran.

- Elastis

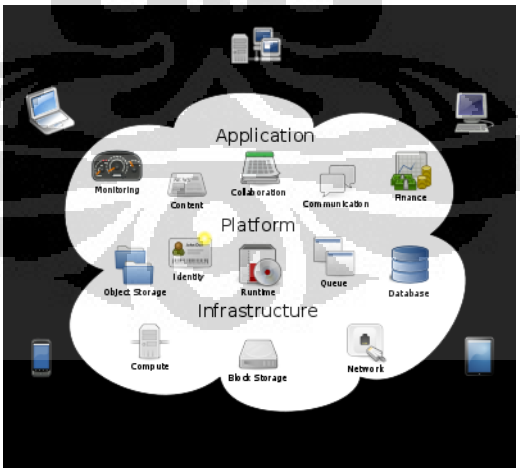
  User dapat meningkatkan dan mengurangi sumberdaya komputasi yang dibutuhkan secara cepat maupun memberhentikan sumberdaya untuk user yang lain saat sudah tidak diperlukan sehingga sistem *cloud* bersifat elastis.

- *Pay as you go*

  User hanya membayar untuk sumberdaya yang digunakan secara aktual dan hanya untuk jangka waktu tertentu saat sumberdaya tersebut dibutuhkan.

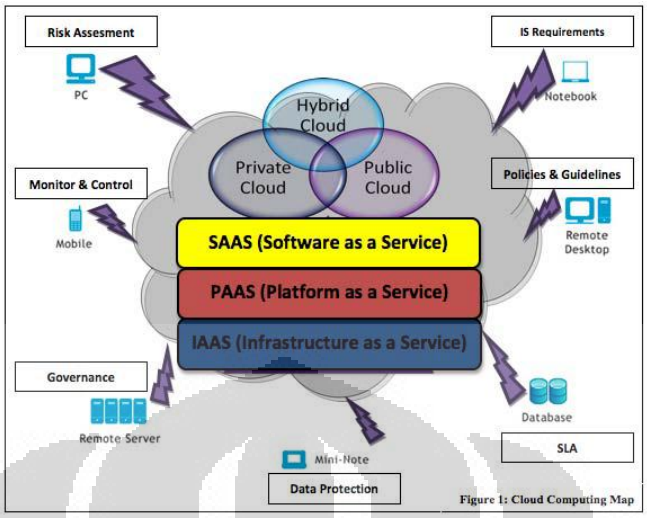
- *Self-provisioning of resources*

  User menentukan sumberdayanya sendiri seperti sistem tambahannya (kapabilitas proses, software, dan storage) dan sumberdaya jaringannya.



Gambar 2.1 Ilustrasi Cloud Computing [3]

2.2 Jenis-jenis *Cloud*



Gambar 2.2 Ketiga Jenis Cloud Computing [4]

Terdapat tiga tipe untuk model pembangunan *cloud* yaitu *public cloud*, *private cloud* dan *hybrid cloud*.

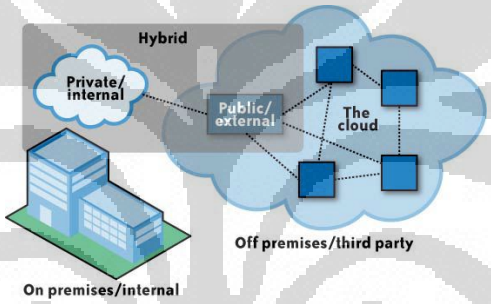
2.2.1 *Public Cloud*

Infrastuktur *public cloud* tersedia untuk publik atau sebuah grup industri yang besar dan tipe *cloud* ini dikelola oleh suatu penyedia layanan *cloud* (*cloud provider*). Pada tipe *cloud* ini user diberi akses ke *cloud* melalui antarmuka yang menggunakan *web browser*. *Public cloud* memiliki keamanan yang lebih rendah dibandingkan dengan model *cloud* yang lain karena menggunakan akses layanan publik seperti Internet yang membutuhkan sistem keamanan yang lebih tinggi untuk memastikan bahwa aplikasi dan akses data pada *public cloud* tidak terkena serangan yang berbahaya. Oleh sebab itu, kepercayaan dan privasi menjadi konsentrasi penelitian pada *public cloud* dengan *cloud* SLA (Service Level Agreement) sebagai pusat. SLA antara vendor dan klien harus ditetapkan dengan beberapa opsi seperti : (1) *cloud vendor* dan klien dapat saling memiliki kesepakatan dalam berbagi tanggung jawab bersama untuk mengecek *cloud* dan melakukan validasi diatas sistem mereka atau (2) masing-masing pihak menetapkan peran dan tanggung jawabnya masing-masing dalam menangani komputasi *cloud*.

2.2.2 *Private Cloud*

Private cloud merupakan sistem *cloud* untuk pusat data organisasi internal. Pada *cloud* jenis ini, sumberdaya yang bersifat *scalable* dan aplikasi virtualnya disediakan oleh vendor *cloud* yang dikumpulkan dan tersedia untuk di-*share* ke user. Perbedaannya dengan *public cloud* adalah private *cloud* memiliki sumberdaya dan aplikasi yang diatur oleh organisasi itu sendiri dengan fungsionalitas Intranet dan terdedikasi untuk organisasi itu serta tidak untuk di-*share* ke organisasi lain. Utilisasi pada *private cloud* lebih aman dibandingkan dengan *public cloud* karena sumberdaya *cloud* dan aplikasinya diatur oleh jaringan internal dimana hanya organisasi tersebut yang dapat mengakses *cloud*.

2.2.3 *Hybrid Cloud*



Gambar 2.3 Ilustrasi Hybrid Cloud [5]

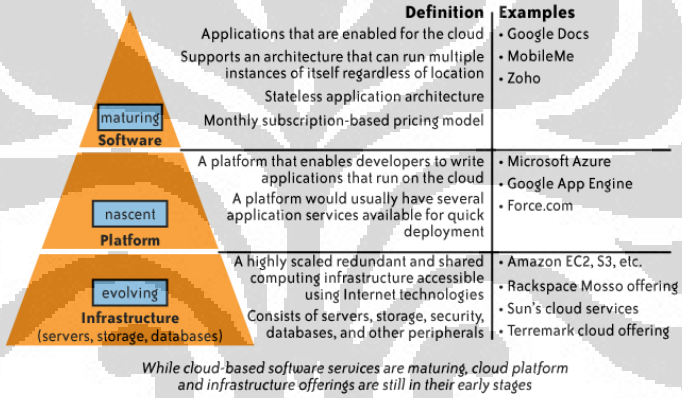
Hybrid cloud merupakan sebuah private *cloud* yang terhubungkan oleh satu atau lebih layanan eksternal *cloud* dengan pengaturan yang bersifat sentral, ditetapkan sebagai satu unit tunggal dan dibatasi oleh jaringan yang aman. *Hybrid cloud* memungkinkan organisasi menjalankan aplikasi yang tidak terpusat pada *public cloud*, dimana aplikasi inti dan data yang sensitif berada pada private *cloud*. *Hybrid cloud* mengizinkan berbagai macam pihak untuk mengakses informasi melalui Internet dan memiliki arsitektur yang terbuka yang mengizinkan antarmuka lain dengan sistem manajemen yang berlainan. *Hybrid cloud* juga mendukung solusi teknologi virtual melalui *public* dan *private cloud*.

2.3 Model Komputasi Tradisional

Model komputasi tradisional merupakan model komputasi dengan penggunaan hardware fisik seperti biasa yang didedikasikan untuk klien tunggal. Pada model komputasi tradisional, jumlah user yang meningkat akan sebanding

dengan biaya pembangunan untuk menunjang kebutuhan akan tambahan perangkat keras server dan komponen pendukung lainnya. Biaya lisensi bergantung pada *metric* (seperti tipe server, jumlah CPU dan karakteristik fisikal lainnya) yang tidak dihitung berdasarkan penggunaan secara langsung dan tidak bersifat virtual. Sedangkan pada model *cloud*, biaya tergantung pada penggunaan dan dapat menggunakan teknologi virtual. Jadi model komputasi tradisional ini membutuhkan penyesuaian yang tinggi antara penambahan klien dan hardware serta ekstensi jaringannya sehingga hal ini akan berakibat bertambahnya biaya dan sumberdaya manusia yang dibutuhkan.

2.4 Model Layanan Pengiriman *Cloud Computing*



Gambar 2.4 *Cloud Service Delivery Model* [5]

Arsitektur dari *cloud computing* dapat dikategorisasikan berdasarkan *cloud service delivery model*. Terdapat tiga variasi dari model layanan pengiriman dalam *cloud* yaitu [5]:

2.4.1 *Software as a Service (SaaS)*

Software as a Service dioperasikan pada lingkungan virtual dan menggunakan model biaya *pay-per-use*. Penyedia *SaaS* dapat menyimpan software pada pusat data milik mereka ataupun dengan menggunakan *outsource* penyedia *IaaS*. Ketersediaan dari layanan *IaaS* merupakan kunci utama tersedianya model *SaaS*. Aplikasi *SaaS* diakses menggunakan *web browser* melalui Internet sehingga keamanan pada browser menjadi hal yang vital. Jadi, model *SaaS* pada *cloud* memberikan ketersediaan layanan software untuk klien. Perbedaan *SaaS* dengan *Application Service Provider* (ASP) tradisional adalah

model *SaaS* merupakan aplikasi *multitenant* (memiliki banyak penyewa) yang di-*host* oleh vendor dan aplikasi *SaaS* didesain sebagai aplikasi net-Native sehingga dapat terbaharui secara berkelanjutan. Contoh vendor yang menerapkan model *SaaS* adalah DropBox.com yang memberikan layanan penyimpanan data yang mampu di-*share*. Beberapa keuntungan utama dari model *SaaS* adalah :

- Mengizinkan suatu organisasi untuk outsource hosting dan manajemen aplikasi untuk sebuah *third party* (vendor perangkat lunak dan penyedia layanan) dengan tujuan untuk mengurangi biaya dari lisensi aplikasi software, server dan infrastuktur lain yang dibutuhkan untuk host aplikasi secara internal.
- *SaaS* mengizinkan vendor perangkat lunak untuk mengontrol dan membatasi penggunaan, melarang penyalinan dan distribusi, dan memfasilitasi kontrol dari semua versi turunan dari suatu software.
- Pengiriman aplikasi menggunakan model *SaaS* biasanya menggunakan pendekatan pengiriman *one-to-many* dengan menggunakan infrastuktur Web sehingga memberikan kemudahan pada *end user* untuk mengakses aplikasi *SaaS* melalui *web browser*.
- Distribusi *SaaS* tipikal tidak membutuhkan hardware dan dapat dijalankan pada infrastuktur akses Internet yang telah ada sebelumnya sehingga memberikan kemudahan untuk pembangunan model *SaaS*, namun pengaturan ulang firewall diperlukan agar aplikasi *SaaS* dapat berjalan dengan baik.

Manajemen dari aplikasi *SaaS* digunakan oleh vendor dari perspektif *end user*, dimana aplikasi *SaaS* dapat dikonfigurasi menggunakan API (*Application Programming Interface*), namun aplikasi *SaaS* tidak dapat seluruhnya disesuaikan.

2.4.1 *Platform as a Service (PaaS)*

Model *PaaS* menawarkan *platform* yang dibutuhkan user untuk membuat aplikasi-aplikasi. Layanan utama PaaS biasanya berupa desain aplikasi, *development*, proses *testing* dan *deployment* serta *hosting*. Pada model *PaaS*, vendor menawarkan sebuah lingkungan untuk *developer* aplikasi yang mengembangkan aplikasi dan menawarkan layanan melalui platform provider. Provider membangun toolkit dan standar untuk pembangunan kanal untuk

distribusi dan pembayaran, kemudian provider akan menerima pembayaran untuk menyokong platform dan penjualan serta distribusi layanannya. Contoh *vendor* yang menyediakan layanan *PaaS* adalah Google App Engine.

PaaS menawarkan lingkungan pengembangan sebagai servis atau layanannya dimana pengembang menggunakan blok bangunan seperti *predefined blocks of code* untuk menciptakan aplikasi. Solusi *PaaS* merupakan pembangunan platform dimana alat pembangunannya di-*host* di *cloud* dan diakses melalui *browser*. Model *PaaS* memberikan kemudahan bagi pengembang untuk membangun aplikasi web tanpa menginstal tools pada komputer mereka dan mereka dapat menyebarkan aplikasi tanpa keahlian sistem administrasi tertentu. Manfaat dari *PaaS* tergantung pada peningkatan jumlah sumberdaya manusia yang dapat mengembangkan, mempertahankan dan mendistribusikan aplikasi web.

Kebutuhan minimum untuk pembangunan model *PaaS* adalah [5]

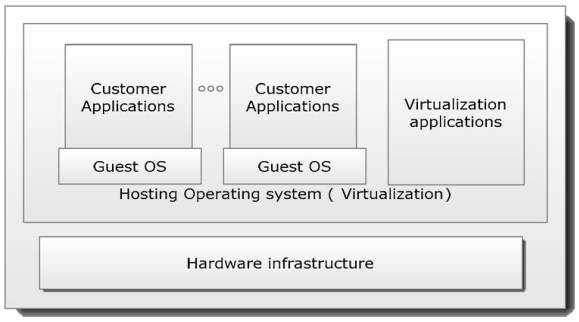
- Solusi pembangunan *PaaS* harus berbasis *browser*.
- *End-to-end PaaS* harus mendukung produktivitas yang tinggi dengan IDE (*Integrated Development Environment*) yang berjalan pada platform pengiriman target yang aktual sehingga *debugging* dan skenario tes platform berjalan pada lingkungan yang sama.
- *PaaS* harus mendukung integrasi dengan layanan *web* dan *database* eksternal.
- *PaaS* harus mendukung *monitoring* yang kompeherensif mengenai aplikasi dan aktivitas user untuk membantu pengembang mengontrol aplikasinya dan efek dari adanya perubahan.
- Skalabilitas, Reabilitas dan Keamanan harus dibangun pada solusi *PaaS* tanpa membutuhkan konfigurasi, pengembangan, dan biaya tambahan. *Multitenancy* harus diasumsukan tanpa adanya kerja tambahan pada sistem.
- *PaaS* harus mendukung kolaborasi formal dan *on-demand* selama keseluruhan daur hidup perangkat lunak untuk mempertahankan keamanan dari kode sumber dan properti intelektual yang bersangkutan.
- *PaaS* harus mendukung *pay-as-you-go billing*.

2.4.2 *Infrastucture as a Service (IaaS)*

Model *IaaS* mendukung infrastuktur untuk pelanggan menjalankan suatu aplikasi dimana pembayarannya menggunakan sistem *pay-peruse* dan perluasan layanannya tergantung pada permintaan pelanggan. Vendor *IaaS* dapat mencakup *hosting* aplikasi atau memperluasnya ke servis yang lain (seperti dukungan aplikasi, pengembangan dan peningkatan aplikasi) dan mendukung *outsourcing* IT yang komperehensif. Model *IaaS* mirip dengan komputasi utilitas dimana pelanggan dapat membayar untuk penggunaan *processing power*, *disk space* dan hal yang lainnya yang berhubungan dengan utilitas komputer. *IaaS* merupakan layanan yang berhubungan dengan *cloud* dan mengacu pada layanan online yang memberikan layanan seperti sumberdaya komputasi fisik, lokasi, partisi data, *storage*, keamanan, *backup*, dan yang lainnya. Pada *cloud computing*, provider merupakan kontrol utama dalam infrastukur sedangkan klien menggunakan layanan yang diberikan provider untuk mendistribusikan, mengatur dan memperluas layanan *online* menggunakan sumberdaya provider dan membayar apa yang dikonsumsi oleh pelanggan. Contoh *vendor* yang menyediakan layanan model *IaaS* adalah Amazon Compute Cloud.

2.5 Teknologi Virtualisasi

Teknologi virtualisasi merupakan suatu teknologi yang menyediakan mesin virtual yang dinamis dibawah suatu sistem operasi *host* (sistem operasi utama yang berjalan pada perangkat keras fisik). Virtualisasi merupakan teknologi yang dapat mengoptimalisasi performasi dari aplikasi serta teknologi ini memberikan efektifitas terhadap biaya. Namun, permasalahan dari teknologi ini adalah resiko keamanan terutama pada server virtual dimana penggunaan server virtual ini dapat meringankan beban biaya. Virtual Machine (VM) mengacu pada perangkat lunak komputer seperti komputer fisik yang menjalankan sistem operasi dan aplikasi. Sistem operasi di dalam VM disebut dengan *guest* OS. VM Monitor atau Manager (VMM) menciptakan dan mengontrol subsistem virtual dari VM yang lain.



Gambar 2.5 Sistem Operasi Berbasis Virtualisasi [1]

Virtualisasi merupakan platform teknologi yang menjadi dasar teknologi *cloud computing*. Virtualisasi mengacu pada sumberdaya komputasi (seperti CPU, *storage*, jaringan, memori, stack aplikasi dan *database*) dari aplikasi yang kemudian dimanfaatkan oleh user. Teknologi virtualisasi mengizinkan adanya model bisnis *cloud multitenancy* dengan mendukung platform sumberdaya yang dapat dibagi dan diperluas untuk semua penyewa. Berdasarkan perspektif penyedia *cloud* atau pihak pebisnis, teknologi ini mendukung sumberdaya terdedikasi untuk platform pelanggan dan menawarkan konsolidasi pusat data. Operasional Teknologi Informasi dengan virtualisasi juga menjadi lebih efisien. Teknologi virtulisasi yang dikembangkan dalam berbagai bentuk antara lain virtualisasi OS (VMWare, Xen), virtualisasi *storage* (SAN), dan virtualisasi aplikasi atau software (Apache Tomcat, JBoss, Oracle AppServer, dan WebSphere). Sedangkan berdasarkan perspektif *public cloud*, teknologi virtualisasi memberikan layanan berupa sumberdaya yang dapat di-*share* pada berbagai macam lapisan layanan virtual tergantung pada model layanan pengirimannya.

Salah satu teknik dalam virtualisasi disebut dengan *hypervisor*. *Hypervisor* akan mengizinkan OS (Operating System) jamak berjalan pada satu komputer host secara bersamaan sehingga fitur ini disebut dengan virtualisasi hardware. Secara konseptual, *hypervisor* memiliki level satu tingkat lebih tinggi dari supervisor. *Hypervisor* akan menyediakan platform virtual kepada guest OS dan *hypervisor* ini dapat memonitor eksekusi yang dilakukan oleh guest OS. *Hypervisor* diinstal pada hardware berupa server dimana berbagai *instance* dari beragam OS dapat berbagi sumberdaya.

2.6 *Cloud* Berbasis Virtualisasi

Virtualisasi digunakan untuk mempartisi suatu server fisik tunggal menjadi beberapa VM jamak. Untuk *cloud* computing, virtualisasi ini meningkatkan fleksibilitas dimana sumberdaya server dapat digunakan dan dimatikan atau dihapus secara cepat. Selain itu, virtualisasi mendukung sistem separasi untuk beberapa pelanggan atau penyewa pada hardware fisik tunggal sehingga dapat memberikan efektifitas biaya. Beberapa vendor yang menawarkan teknologi *cloud* berbasis virtualisasi antara lain adalah VMWare, Cytrix XenServer, Microsoft Hyper V, Ubuntu Enterprise Cloud, dan lainnya.

Disamping kelebihan-kelebihan teknologi *cloud* menggunakan virtualisasi seperti yang telah disebutkan sebelumnya, teknologi virtualisasi ini juga memberikan masalah potensial yang perlu diteliti dan ditangani, yaitu pada masalah keamanan *cloud*. Provider yang mengkombinasikan beragam VM pada sebuah server fisik menyebabkan adanya masalah performasi seperti masalah I/O *bottleneck*s sehingga apabila terdapat sumberdaya pada server yang terkena serangan maka seluruh VM akan terjangkit serangan. Oleh sebab itu, diperlukan manajemen tugas pada *cloud* berbasis virtualisasi dan diperlukan *monitoring real time* pada server fisik dan VM.

Serangan terhadap teknologi *cloud* berbasis virtualisasi terutama pada VM antara lain adalah serangan ke *hypervisor* yang digunakan oleh provider dimana *attacker*yang berhasil mengambil alih *hypervisor* akan memiliki akses teritori *hypervisor* terhadap semua VM, injeksi SQL pada lapisan layanan *cloud* karena lingkungan yang tidak aman, masalah autentikasi dan autorisasi dimana penyedia layanan harus mendistribusikan *security policy* dan metric milikinya karena pada *cloud* framework autentikasi dan autorisasi tidak menyebar secara otomatis, masalah komunikasi pada level virtualisasi, dan lainnya. Untuk masalah komunikasi pada level virtualisasi, jika terdapat parameter keamanan yang disepakati antara dua titik maka terdapat kemungkinan adanya potensi serangan terhadap suatu target.

2.6.1 Manajemen Keamanan Virtualisasi

Untuk menangani masalah keamanan virtualisasi, terdapat beberapa rekomendasi keamanan VM [5] antara lain adalah

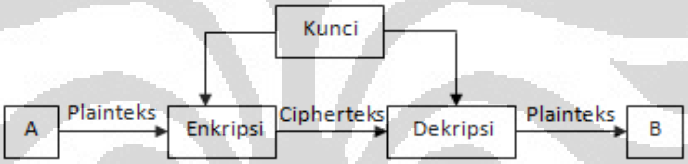
- Memperkuat kondisi host OS yaitu dengan menggunakan kata kunci yang kuat, men-*disable* program yang tidak dibutuhkan, menggunakan autentikasi penuh untuk kontrol akses, menggunakan firewall pada host, serta melakukan *update* pada host secara teratur.
- Menggunakan vendor yang dipercaya dan standar tertentu untuk konfigurasi guest dan host, seperti melakukan konfigurasi yang merujuk pada NIST Computer Resource Center dan Defence Information System Agency (DISA) Security Technical Implementation Guides (STIGA).
- Membatasi akses fisik ke host.
- Menggunakan komunikasi yang terenkripsi seperti *encrypted* VPN dan SSH.
- *Disabling* latar belakang tugas.
- Mengadakan garis pertahanan pada VM dengan *enabling* firewall atau sistem deteksi gangguan (intrusion detection system) pada VM.
- Mengimplementasikan pengecekan integritas file.
- Melakukan *backup* data.
- Selalu kontak dengan vendor virtualisasi selama proses *update* dan *patch*. Host OS dan VM juga harus memiliki jadwal *patch* yang sama.
- VM monitor harus *root secure* yang berarti tidak ada hak istimewa pada lingkungan guest virtual yang mengizinkan interferensi atau gangguan terhadap sistem host.

2.7 Kriptografi

Sebelum dibahas lebih lanjut mengenai pertukaran kunci, akan dijelaskan terlebih dahulu mengenai kriptografi. Kriptografi adalah seni atau teknik penyandian yang bertujuan untuk menyembunyikan suatu informasi sehingga tidak dimengerti oleh pihak lain kecuali pembuat sandi dan penerimanya. Kriptografi ini menyandikan suatu informasi berbasis teks. Istilah-istilah yang berhubungan dengan kriptografi antara lain adalah *cipher text*, *plain text*, enkripsi dan dekripsi. Plain text adalah pesan asli yang dikirimkan sedangkan *cipher tex*t

merupakan pesan yang telah disandikan atau dienkripsi. *Cipher text* berupa kode yang menggantikan kata-kata di dalam plain text dimana kode-kode ini tergantung pada jenis kriptografi yang digunakan. Enkripsi adalah suatu proses untuk membuat *plain text* menjadi *cipher text*, sedangkan dekripsi merupakan kebalikan dari enkripsi.

Pertukaran kunci terdiri dari dua macam yaitu pertukaran kunci simetris (*symmetric key-exchange*) dan pertukaran kunci asimetris (*asymmetric key-exchange*). Kunci simetris menggunakan kunci enkripsi dan dekripsi yang sama.Kunci asimetris sering disebut dengan kunci publik. Kunci ini disebut asimetris karena kunci yang digunakan untuk enkripsi dan dekripsi berbeda. Jenis kriptografi yang menggunakan kunci simetris antara lain adalah AES, DES, RC6, Blowfish, A5 dan lain-lain. Jenis kriptografi kunci asimetris adalah RSA, Digital Signature, dan Diffie-Hellman.Gambar 2.6 dan Gambar 2.7 menunjukkan blok diagram dari kunci simetris dan asimetris.



Gambar 2.6 Pertukaran Kunci Simetris [7]

Gambar 2.7 Pertukaran Kunci Asimetris [7]

Pada tahun 1976, peneliti dari Stanford University, yaitu Diffie dan Hellman mengajukan sebuah sistem kriptografi dimana kunci enkripsi dan dekripsinya berbeda. Hal ini berbeda dari era kriptosistem sebelumnya yang selalu menggunakan kunci enkripsi dan dekripsi yang sama. Oleh sebab itu, proposal Diffie Hellman ini menjadi suatu langkah awal terciptanya kriptografi asimetris lainnya, dimana kunci dekripsi tidak secara langsung diturunkan dari kunci enkripsinya. Berikut ini adalah tiga hal penting dalam algoritma kunci asimetris.

- D(E(P))=P. Jika kunci D diimplementasikan ke pesan yang terenkripsi yaitu E(P), maka akan didapatkan pesan plainteks P kembali. Tanpa properti ini maka penerima pesan akan sulit untuk mendekripsi cipherteks.
- D tidak boleh dengan mudah diperoleh atau disimpulkan dari kunci E, yang artinya kunci D bukan turunan dari kunci E dan berbeda dari kunci E.
- Kunci E tidak dapat dipecahkan oleh *plaintext attack* yang terpilih. Hal ini menjadi penting karena mungkin saja kunci E dapat didistribusikan ke publik, sehingga dalam algoritma kunci asimetris kunci E merupakan kunci publik sedangkan kunci D merupakan kunci rahasia yang digunakan untuk mendekripsikan cipherteks agar pesan asli diketahui oleh penerima.

Kriptografi kunci publik dapat dianalogikan seperti kotak surat yang terkunci dan memiliki lubang untuk memasukkan surat. Kotak surat yang diletakkan di depan rumah pemiliknya sehingga setiap orang dapat memasukkan surat ke dalam kotak tersebut, tetapi hanya pemilik kotak yang dapat membuka dan membaca surat yang ada di dalam kotak tersebut. Kriptografi kunci publik berkembang menjadi sebuah revolusi baru dalam sejarah kriptografi, tidak seperti pada kunci simetris yang hanya didasarkan pada subtitusi dan permutasi saja, akan tetapi kriptografi kunci publik didasarkan pada fungsi matematika seperti perpangkatan dan modulus.

2.8 Algoritma Kriptografi RSA

Pada tahun 1977, peneliti MIT yaitu Ron Rivest, Adi Shamir dan Len Adleman mengajukan suatu algoritma kriptografi kunci publik yang disebut dengan RSA. RSA merupakan singkatan dari nama belakang para penemu algoritma kunci publik ini.

2.8.1 Persamaan Matematis

RSA menggunakan fungsi matematis berupa modulus. Salah satu contoh sederhananya adalah menggunakan cipher sederhana yaitu Caesar cipher, dimana enkripsi dan dekripsi dapat direpresentasikan dengan fungsi matematis untuk memahami lebih dalam mengenai algoritma RSA.

2.8.1.1 Penambahan Modular

$$C = (P + K)\ mod\ n \ldots (2.1)$$

$$P = (C - K)\ mod\ n \ldots (2.2)$$

C merupakan fungsi cipherteks, P merupakan plainteks, dan K merupakan kunci rahasia. Persamaan (1) merupakan operasi enkripsi sedangkan persamaan (2) merupakan operasi dekripsi. Pada operasi dekripsi, C dikurangi K dapat digantikan dengan penambahan C dengan invers K sehingga persamaannya akan seperti persamaan berikut ini.

$$C = (P + K)\ mod\ n \ldots (2.3)$$

$$P = (C + K')\ mod\ n \ldots (2.4)$$

dimana $(K + K')\ mod\ n = 0 \ldots (2.5)$

Contoh untuk penambahan modular ini dapat digantikan dengan nominal modular yang kecil misalnya modulus 10. Perhitungan enkripsi dan dekripsi menggunakan modulus 10 ini dapat merujuk pada tabel 2.1. Misalnya pada sisi penerima ingin mengetahui plainteks, maka harus mengetahui kunci rahasia terlebih dahulu. Jika kunci rahasianya 4, maka kunci publiknya adalah 6 karena (4+6) mod 10 = 0. Untuk melakukan enkripsi, maka kunci K = 4 ditambahkan ke plainteks. Sedangkan untuk dekripsi atau untuk mengetahui plainteks, maka kunci K' = 6 ditambahkan ke cipherteks.

Tabel 2.1 Penambahan Modulus 10 [8]

K	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

2.8.1.2 Perkalian Modular

Persamaan kriptosistem untuk perkalian modular direpresentasikan oleh persamaan berikut ini.

$$C = (P \cdot K)\ mod\ n \ldots (2.5)$$

$$P = (C \cdot K^{-1})\ mod\ n \ldots (2.6)$$

dimana $(K \cdot K^{-1})\ mod\ n = 1 \ldots (2.7)$

Untuk operasi dekripsi pada persamaan (2.6) menggunakan invers multiplikatif yang dikalikan dengan cipherteks, dimana multiplikatif invers dinotasikan dengan K^{-1}.Saat nilai n sangat besar, maka akan sulit untuk mencari invers multiplikatif pada aritmetik modulus n, oleh sebab itu digunakan algoritma Euclid untuk mencari invers. Perkalian modular dapat digunakan sebagai cipher, jika nilai K yang dipilih sesuai dan K serta invers K dapat digunakan sebagai sepasang kunci enkripsi dan dekripsi yang dibutuhkan pada algoritma kunci publik. Namun, permasalahannya adalah walaupun terdapat algoritma Euclid yang dapat mengkalkulasikan K^{-1}, namun model enkripsi kunci publik menyatakan bahwa kunci rahasia tidak dapat diturunkan secara langsung dari kunci publiknya.

Pada algoritma RSA, digunakan fungsi totient $\varphi(n)$ yaitu fungsi aritmetik yang menghitung jumlah integer positif yang kurang dari sama dengan n dan

relatif prima terhadap n. Jika n merupakan integer positif, maka $\varphi(n)$ adalah jumlah integer k pada range $1 \leq k < n$ untuk $gcd(n,k) = 1$ (gcd = Greatest Common Division). Fungsi totient ini merupakan fungsi multiplikatif yang menyatakan bahwa dua angka saling relatif prima. Fungsi totient ini merupakan satu fungsi yang penting pada algoritma RSA, dimana terdapat ketentuan-ketentuan seperti berikut ini:

- Saat n adalah prima, maka $\varphi(n) = n - 1$
- Saat n merupakan dot product dari dua daerah prima p dan q dengan $n = p \cdot q$ dan $p \neq q$ adalah prime, maka $\varphi(n) = (p-1)(q-1)$

2.8.1.3 Perpangkatan Modular

Sistem kriptografi yang menggunakan perpangkatan modular direpresentasikan oleh persamaan 2.8 dan 2.9.

$$C = (P^K) \bmod n \quad (2.8)$$

$$P = \left(C^{K''}\right) \bmod n \quad (2.9)$$

dimana K'' merupakan inverse eksponensiatif dari kunci K.

2.8.2 Algoritma RSA

Skema RSA merupakan sebuah block cipher dengan setiap blok plainteks berupa bilangan integer antara 0 dan $n-1$ yang akan mengarahkan ke sebuah ukuran blok yang lebih kecil sama dengan $log_2(n)$. Ukuran untuk n biasanya 1024 bits atau kelipatannya yaitu 2048 dan selanjutnya. Pada algoritma RSA, terdapat tiga tahap operasi yang penting, yaitu pembangkitan kunci, enkripsi dan dekripsi.

2.8.2.1 Pembangkitan Kunci

Untuk pembangkitan kunci terdapat beberapa tahap berikut ini.

1. Pilih dua bilangan acak p dan q, dengan $p \neq q$.
2. Kalkulasikan $n = p \times q$
3. Hitung $\varphi(n) = (p-1)(q-1)$
4. Pilih sebuah bilangan e sehingga $gcd(e, \varphi(n)) = 1$ dengan $1 < e < (p-1)(q-1)$.

5. Kalkulasikan d sehingga $d \cdot e \, mod \, \varphi(n) = 1$, d merupakan invers multiplikatif dari e pada $mod \, \varphi(n)$.
6. Kunci publik diperoleh dalam format $K_U = \{n, e\}$.
7. Kunci rahasia diperoleh sebagai $K_R = \{d, n\}$.

2.8.2.2 Enkripsi

Setelah pembangkitan kunci berhasil, kemudian akan dilakukan enkripsi pada plainteks sehingga akan menghasilkan cipherteks. Untuk enkripsi pada RSA digunakan fungsi perpangkatan modular dengan blok plainteks $P < n$, dimana fungci cipherteks adalah

$$C = P^e \, mod \, n \ldots (2.10)$$

2.8.2.3 Dekripsi

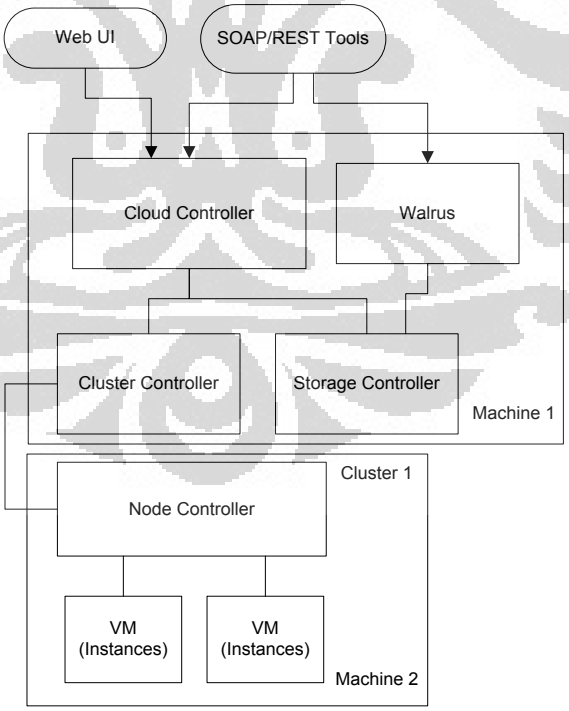
Untuk mendekripsi cipherteks dibutuhkan kunci privat atau kunci rahasia, dimana penerima harus memiliki kunci private yang tepat untuk dapat melakukan dekripsi terhadap cipherteks yang dikirimkan oleh pengirim. Untuk mendekripsikan cipherteks agar penerima mengetahui plainteks, maka digunakan fungsi berikut ini

$$P = C^d \, mod \, n \ldots (2.11)$$

BAB 3
PERANCANGAN DAN IMPLEMENTASI ENKRIPSI RSA PADA SISTEM VIRTUALISASI *PRIVATE CLOUD COMPUTING IAAS*

3.1 Topologi Sistem Virtualisasi *Private Cloud*

Sistem *cloud* yang akan digunakan adalah *private cloud* dimana *cloud* ini hanya dapat diakses oleh pihak yang diberikan izin untuk mengakses sumberdaya pada *cloud serve*r. Untuk menciptakan sistem *cloud* digunakan suatu tools Open Source yaitu *Ubuntu Enterprise Cloud*, tools ini untuk menciptakan sistem *cloud* dengan model layanan pengiriman *IaaS* (Infrastucture as a Service) yang menciptakan suatu sistem yang dapat saling berbagi utilitas komputer, seperti server storage atau penyimpanan data. Ubuntu Enterprise Cloud (UEC) menggunakan suatu platform perangkat lunak *cloud* yang dinamakan Eucalyptus dan menggunakan aplikasi virtualisasi server seperti KVM didalamnya. Eucalyptus ini diinstalasi pada computer yang menjadi server. Eucalyptus ini kompatibel dengan sistem operasi Open Source distribusi Linux seperti Ubuntu.



Gambar 3.1 Blok Diagram Eucalyptus Cloud [9]

Pada *cloud* berbasis Eucalyptus terdapat dua komponen penting, yaitu Cloud Controller (CLC) dan Node Controller (NC). Node Controller merupakan sebuah server yang menggunakan teknologi virtualisasi untuk menjalankan *hypervisor* seperti KVM. Pada *cloud* berbasis UEC, jika perancang sistem menginstalasi NC maka secara otomatis KVM akan terpasang. Mesin virtual yang dijalankan pada *hypervisor* dan dikontrol oleh UEC disebut dengan *instance*. Fungsi NC antara lain adalah mengatur siklus kehidupan dari *instance* yang berjalan pada node serta memonitor mengenai data ketersediaan dan penggunaan sumberdaya pada node dan melaporkan data tersebut ke CC. NC akan mengetahui OS yang digunakan node serta berbagai macam sumberdaya fisik seperti besar memori, ketersediaan disk dan status VM pada node.

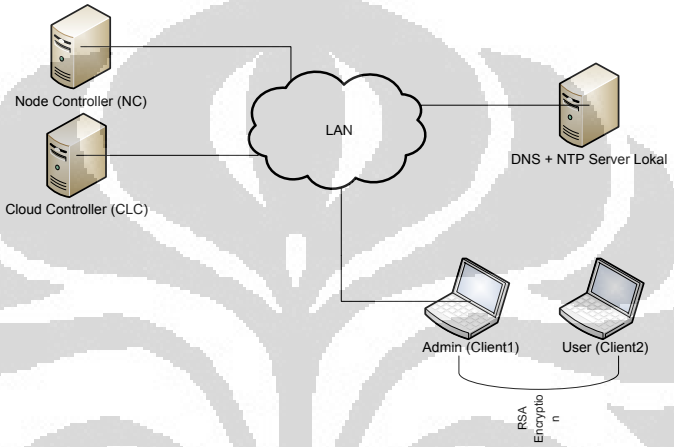
Cloud controller merupakan suatu perangkat controller yang berfungsi sebagai *front end* pada infrastuktur *cloud* yang berarti memberikan antarmuka layanan web kepada klien. Selain itu, CLC berinteraksi dengan komponen lain pada sistem *cloud* berbasis Eucalyptus. Fungsi CLC antara lain adalah memonitor ketersediaan sumberdaya pada *cloud*, memonitor instance yang sedang berjalan (penggunaan sumber daya), dan menentukan cluster yang akan digunakan untuk bertanggung jawab terhadap sebuah instance serta untuk mengetahui status dari *cloud*.

Pada topologi sistem virtualisasi *private cloud* ini, digunakan satu perangkat keras yang diinstal dengan CLC yang didalamnya terdapat berbagai macam controller, yaitu Cluster Controller (CC), Storage Controller (SC), dan Walrus Storage Controller. CC berfungsi untuk mengatur satu atau lebih NC (Node Controller) serta menjalankan *instance* pada NC. CC juga akan berkomunikasi dengan Cloud Controller (CLC). Secara singkat fungsi dari CC adalah menerima permintaan CLC untuk menjalankan *instance*, memutuskan NC yang digunakan untuk menjalankan *instance* tersebut, mengatur jaringan virtual untuk *instance*, dan mengumpulkan informasi mengenai NC yang terdaftar dan melaporkannya kepada CLC.

SC menyediakan blok penyimpanan yang kuat yang digunakan *instance*. SC berfungsi menciptakan divais layanan EBS (Elastic Block Store) yang kuat,

dan mendukung blok penyimpanan melalui protokol AoE (ATA over Ethernet) atau protokol iSCSI (Internet Small Computer Small Interface) untuk *instance*.

Walrus Storage Controller (WS3) ini mendukung layanan penyimpanan yang kuat menggunakan REST (Representational State Transfer) dan SOAP (Simple Object Access Protocol) APIs yang kompatibel dengan API S3 (antarmuka *get/put* untuk suatu objek). Fungsi dari WS3 antara lain adalah menyimpan *machine images* yaitu OS untuk *instance*, menyimpan *snapshot*, dan menyimpan serta memberikan layanan file menggunakan APIS3.



Gambar 3.2 Topologi Virtualisasi *Private Cloud* dengan UEC

Gambar 3.2 menunjukkan topologi dari sistem virtualisasi private *cloud* menggunakan UEC dengan menerapkan RSA untuk keamanan transfer datanya. Seperti yang terlihat pada Gambar 3.2, server *cloud* menggunakan UEC dengan platform perangkat lunak Eucalyptus. Topologi ini menggunakan jaringan LAN karena model pembangunan *cloud* yang akan diterapkan adalah model *private cloud*. Model *private cloud* dipilih karena lebih mudah diterapkan dibandingkan dengan model *public cloud*, dimana model *public cloud* tentunya lebih membutuhkan konsentrasi lebih pada keamanan sistemnya dan juga konfigurasi jaringannya karena model *public cloud* terbuka untuk pengguna umum. Eucalyptus untuk server *cloud* dipilih karena perangkat lunak ini stabil, baik untuk akademisi dalam mempelajari sistem *cloud IaaS* serta merupakan *software open source* sehingga tidak memerlukan biaya untuk menggunakannya.

Klien pertama merupakan administrator yang dapat mengatur konten-konten pada server serta dapat mengatur pengguna mana saja yang diperbolehkan

untuk mengatur *instances* pada server sehingga klien kedua atau user harus meminta kepada admin jika ingin memiliki akses untuk mengatur *instances*. Enkripsi RSA akan diterapkan pada administrator (*instances* yang diatur oleh admin) dan user sehingga file yang diminta diharapkan tidak akan jatuh ke pihak yang tidak diberikan izin karena akan terjadi pertukaran kunci antar pihak-pihak yang memiliki akses.

3.2 Spesifikasi Perangkat Keras

3.2.1 Perangkat Keras Untuk Server *Cloud*

Seperti yang telah diungkapkan pada subbab sebelumnya, server *cloud* yang menggunakan platform perangkat lunak Eucalyptus terdiri dari dua komponen penting yaitu CLC dan NC.Untuk melakukan instalasi CLC dan NC terhadap suatu hardware misalnya Laptop atau PC (Personal Computer) harus memenuhi kebutuhan minimum seperti pada Tabel 3.1 dan Tabel 3.2.

Tabel 3.1 Kebutuhan Minimum Hardware Server CLC [10]

Hardware	**Server Cluster Controller (CLC)**	
	Minimum	Disarankan
CPU	1 GHz	2 x 2GHz
Memory	1 GB	2 GB
Disk	5400rpm IDE	7200rpm SATA
Disk Space	40 GB	200 GB
Networking	100 Mbps	1000 Mbps

Tabel 3.2 Kebutuhan Minimum Hardware Server NC [10]

Hardware	**Server Node Controller (NC)**	
	Minimum	Disarankan
CPU	VT extensions	VT extensions, 64Bit, Multicore
Memory	1 GB	4 GB
Disk	5400rpm IDE	7200rpm SATA atau SCSI
Disk Space	40 GB	100 GB
Networking	100 Mbps	1000 Mbps

Pada penelitian skripsi ini, spesifikasi yang digunakan untuk Server NC dan server CLC tentunya memenuhi kebutuhan minimum seperti yang dideskripsikan pada Tabel 3.1 dan Tabel 3.2.

3.2.2 Perangkat Keras Klien

Untuk spesifikasi perangkat keras klien sebenarnya tidak ada kebutuhan minimum agar sistem *cloud* dapat berjalan dengan baik, hanya pada spesifikasi networking saja dimana kebutuhan minimum untuk klien adalah sekitar 100 Mbps. Tabel 3.3 ini mendeskripsikan spesifikasi perangkat keras yang digunakan oleh klien saat dilakukan penelitian skripsi ini.

Tabel 3.3 Spesifikasi Perangkat Keras Klien

Hardware	Client
Processor	1 CPU, Execution Cap 100%
Base Memory Motherboard	512 MB
Hardware Virtualization	Enable VT-x/AMD-V
Sata Controller	SATA 4.7 GB, Admin(Client1).vmdk

Perangkat keras untuk klien ini menggunakan hardware virtual pada Virtualbox begitu pula dengan perangkat keras untuk Cloud Server dengan Eucalyptus. Pada Tabel 3.3 terlihat bahwa Sata Controller menggunakan format vmdk. VMDK(Virtual Machine Disk) adalah format file hard disk drive virtual yang digunakan pada mesin virtual. VMDK ini merupakan open format sehingga terbuka untuk umum tanpa ada lisensi tertentu yang harus dimiliki.

Pada penelitian skripsi ini digunakan Virtualbox untuk mensimulasikan sistem yang akan dibuat sehingga terdapat spesifikasi minimum untuk membuat mesin virtualisasi. Spesifikasi perangkat keras yang digunakan untuk menginstal Virtualbox ditunjukkan pada Tabel 3.4.

Tabel 3.4 Spesifikasi PC Untuk Virtualbox

Hardware	**PC**	
	Minimum	Yang Digunakan
Processor	Pentium III	Intel Core i7
Kecepatan Processor	700 Mhz	2.2 GHz
RAM	512 MB	4GB

3.3 Spesifikasi Perangkat Lunak

Pada penelitian skripsi ini digunakan sistem operasi Ubuntu untuk server *cloud* yang terdiri dari CLC dan NC, serta perangkat keras klien. Ubuntu merupakan sistem operasi distribusi Linux yang tergolong dalam perangkat lunak open source sehingga tidak diperlukan biaya untuk mengunduh OS ubuntu. Versi Ubuntu yang digunakan untuk server CLC dan NC adalah versi Ubuntu khusus untuk server yang terbaru yaitu Ubuntu Server 11.10 (Oneiric Ocelot) 32 bit. Sedangkan, untuk perangkat keras klien digunakan versi Ubuntu Desktop 11.10 32 bit. Perbedaan paling mendasar dari Ubuntu Desktop dan Ubuntu Server adalah tampilan OS Ubuntu Desktop memiliki GUI (Graphical User Interface), sedangkan tampilan OS Ubuntu Server menggunakan command line atau terminal.

Perangkat lunak yang digunakan berikutnya adalah Eucalyptus untuk server *cloud IaaS*. Eucalyptus kompatibel dengan OS Ubuntu Server. Eucalyptus dapat kompatibel dengan berbagai teknologi virtual, salah satunya adalah KVM (Kernel based Virtual Machine) yaitu infrastuktur kernel untuk Linux. Versi Eucalyptus yang digunakan adalah Eucalyptus 2.0.2 Open Source. Tabel 3.5 merupakan fitur-fitur dari versi Eucalyptus 2.0.x yang digunakan saat penelitian.

Tabel 3.5 Fitur Eucalyptus 2.0.x

Fitur-Fitur	Eucalyptus 2
Kompatibel dengan antarmuka AWS Amazon	√
Kompatibel dengan AWS S3 (Bucked-Based Storage)	√
Kompatibel dengan AWS EBS	√
Local Block Storage Abstraction	√
Elastic Resource Orchestration and Management (Perpindahan, Penambahan, dan Penghapusan)	√
Manajemen Grup Pengguna dan Akses Role-Based	√
Multi-cluster	√
Manajemen image (AWS S3-Backed)	√
Mendukung hypervisor Open Source Xen dan KVM	√
Mendukung Linux Guest OS yang heterogen	√
Manajemen jaringan fleksibel, terdapat grup keamanan, dan trafik isolasi	√
Sumberdaya Laporan dan Akuntansi	√*
Mendukung Guest OS Ms. Windows	√*
Mendukung Hypervisor VMWare	√*
Konversi Image untuk VMWare	√*
Adapter SAN (Storage Area Network) untuk EBS	√*

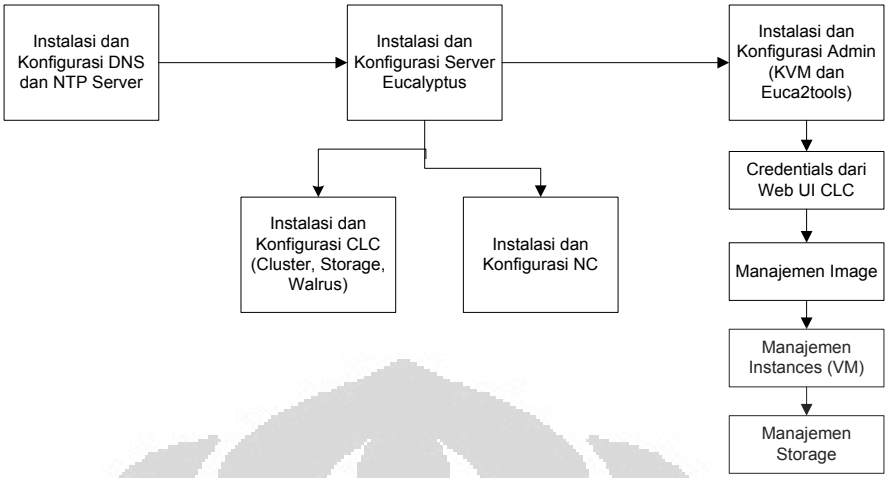
Keterangan : * Tidak tersedia pada Eucalyptus open source

Pada penelitian skripsi ini digunakan simulasi dengan Virtualbox untuk menerapkan sistem virtualisasi private *cloud*. Virtualbox merupakan suatu tools yang diinstal pada host OS sebagai aplikasi dan dapat menjalankan guest OS pada lingkungan virtualnya sendiri. Secara sederhananya, virtualbox merupakan emulator PC yang dapat melakukan emulasi komputer sehingga seakan-akan terdapat komputer-komputer di dalam suatu komputer. Setiap VM pada virtualbox dapat dikonfigurasikan secara mandiri dan dapat berjalan dibawah lingkungan virtualisasinya. Virtualbox yang digunakan saat penelitian adalah Virtualbox versi 4.1.2 Ubuntu r38459 Oracle.

Tabel 3.6 Spesifikasi Perangkat Lunak Virtualbox

Spesifikasi		Virtualbox 4.1.2 Oracle
Virtual Storage	**Hard disk controller**	IDE, SATA, SCSI
	Disk Image Files	VDI, VMDK, VHD
	Networking Hardware	PCNet-PCI II, PcNet-Fast III, Intel Pro 1000 MT Desktop, Intel Pro 1000 T Server, Intel Pro 1000 MT Server, Paravirtualized Network
	Networking Mode	NAT, Bridged Adapter, Internal Networking, Host Only Networking, General Networking

3.4 Konfigurasi Sistem Cloud



Gambar 3.3 Diagram Konfigurasi Sistem Cloud

Untuk melakukan konfigurasi sistem *cloud* menggunakan Eucalyptus terdapat beberapa tahap penting yaitu

3.4.1 Konfigurasi Sistem *Private Cloud*

Sistem *cloud* yang dibangun menggunakan UEC Eucalyptus pada LAN tanpa membutuhkan koneksi Internet harus menggunakan server NTP dan DNS lokal agar sistem dapat berjalan dengan baik. *Cloud* dengan Eucalyptus akan sensitif dengan sinkronisasi waktu sehingga membutuhkan server NTP lokal. Jika terdapat pesan error seperti berikut ini, maka waktu antar mesin *cloud* tidak sinkron. Hal ini terjadi karena sistem *cloud* yang dirancang adalah sistem private *cloud* dengan menggunakan LAN tanpa koneksi Internet.

```
Warning: failed to parse error message from AWS:
<unknown>:1:0: syntax error
EC2ResponseError: 403 Forbidden
Failure: 403 Forbidden
Message has expired (times in UTC):
Timestamp=20120413T03:28:54 Expires=null
Deadline=20120414'T'03:43:54
```

Server DNS digunakan agar host-host lokal yang ada saling dikenali dan untuk melakukan pengarahan alamat IP server NTP lokal.

3.4.1.1 Konfigurasi Server DNS Lokal

Tahapan untuk konfigurasi server DNS lokal antara lain adalah

1. Konfigurasi antarmuka jaringan pada file /etc/network/interfaces seperti berikut ini.

```
auto br0
iface br0 inet static
address 192.168.2.3
netmask 255.255.255.0
bridge_ports eth0
```

2. Instalasi DNS Server lokal. Pada saat penelitian digunakan bind9 Untuk membuat DNS server. Bind9 (Bekerley Internet Name Domain) merupakan perangkat lunak yang digunakan untuk server DNS pada sistem operasi Linux/Unix dan dirilis oleh Computer System Research Groups pada University of California, Bekerley.
3. Konfigurasi file /etc/hosts seperti yang ditunjukkan pada Gambar 3.4.

Gambar 3.4 Konfigurasi file hosts pada DNS Server

Gambar 3.4 di atas menunjukkan pemetaan server ntp ke alamat IP lokal 192.168.2.3. Hal ini bertujuan agar NTP server mengarah ke alamat IP lokal yang telah ditentukan.

3.4.1.2 Konfigurasi Server NTP Lokal

Tahapan untuk konfigurasi server NTP lokal antara lain adalah

- Konfigurasi antarmuka jaringan pada file /etc/network/interfaces seperti berikut ini.

```
auto eth1
iface eth1 inet static
address 127.127.1.0
netmask 255.255.0.0
```

- Konfigurasi pada file /etc/ntp.conf seperti berikut ini.

```
server 127.127.1.0
fudge 127.127.1.0 stratum 1 refid NIST
restrict default notrust nomodify
```

Perintah *server* menyatakan bahwa sistem waktu lokal menggunakan server waktu dengan alamat IP lokal **127.127.1.0**. Sedangkan, perintah *fudge* menyatakan stratum dan refid yang digunakan pada server NTP. Perintah *stratum* menunjukkan level kedekatan suatu server NTP dengan waktu yang digunakan sebagai referensi. Stratum 0 merupakan perangkat yang menjadi referensi utama untuk waktu dan stratum 1 merupakan perangkat yang terhubung dengan stratum 0 dimana perangkat ini biasanya juga menjadi server NTP. Stratum 2 ke atas merupakan perangkat yang menggunakan stratum dibawahnya sebagai referensi dan menyediakan layanan untuk server dengan stratum lebih rendah.

- Restart server NTP menggunakan perintah /etc/init.d/ntp restart.
- Melakukan pengecekkan server NTP dengan menggunakan perintah *ntpq-np* dan *watch ntpq –c lpee.*

  Hasil yang didapatkan setelah melakukan perintah ntpq-np akan memberikan list server waktu seperti yang ditunjukkan pada Gambar 3.5.

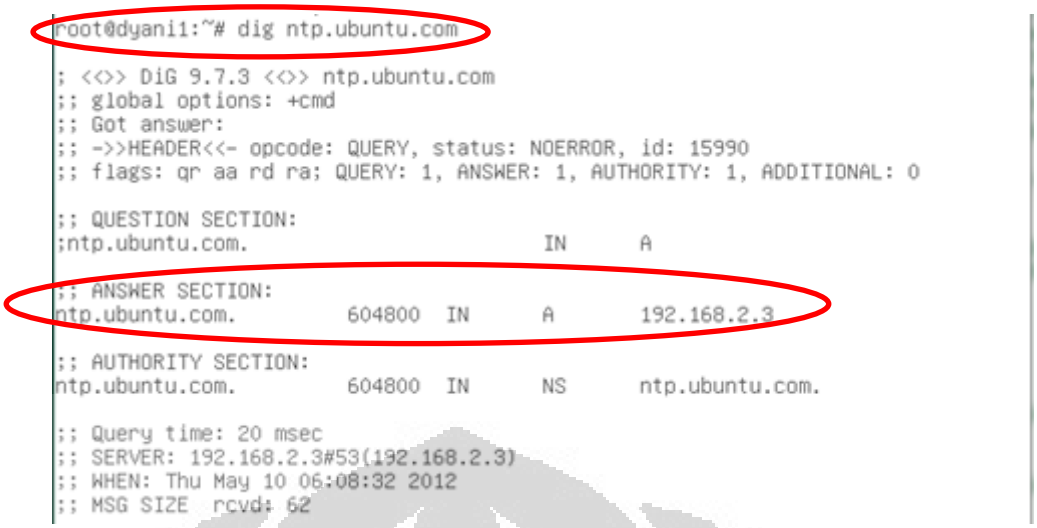
```
root@dyani4:~# ntpq -np
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*127.127.1.0     .NIST.           1 u    9   64  377    0.000    0.000   0.001
 192.168.2.50    .BCST.          16 u    -   64    0    0.000    0.000   0.001
```

Gambar 3.5 Pengecekkan NTP dengan ntpq –np

Pada Gambar 3.5 terlihat bahwa alamat IP **127.127.1.0** terdapat tanda **bintang** yang menunjukkan bahwa waktu pada sistem bersinkronisasi dengan waktu NTP. Jika tidak terdapat tanda bintang pada alamat IP yang dipilih menjadi server NTP, maka waktu server 127.127.1.0 tidak terjangkau atau memiliki jarak sinkronisasi yang jauh.

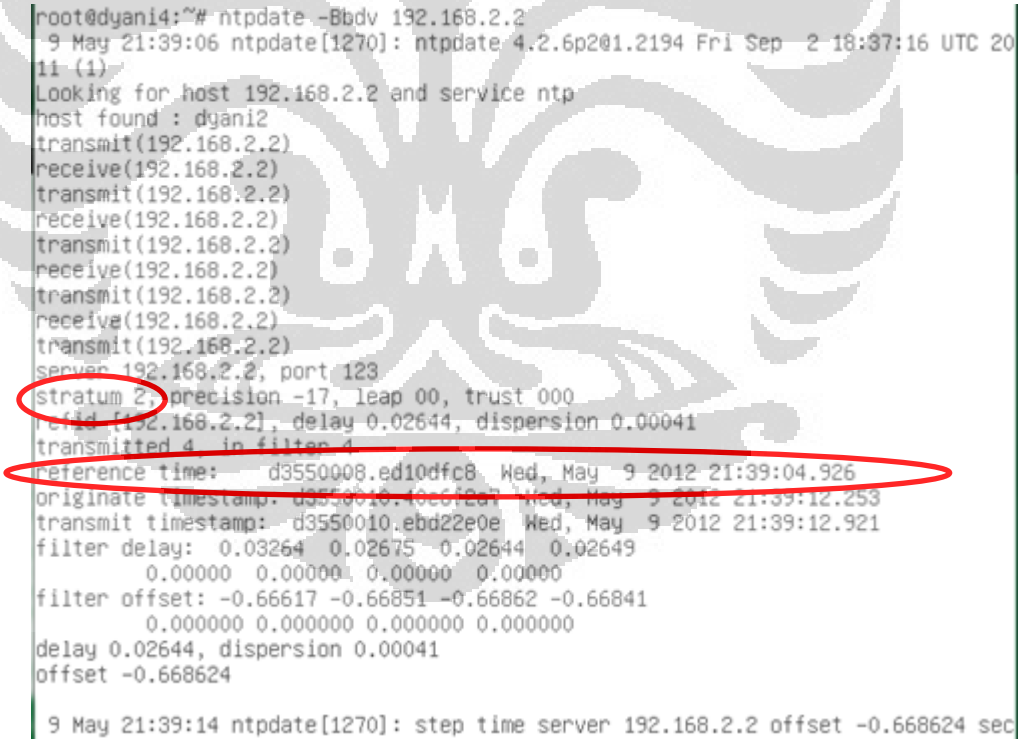
- Melakukan pengecekkan dari server *cloud* menggunakan perintah *dig* dan *ntpdate*.

  Hasil yang didapatkan setelah menggunakan perintah dig ditunjukkan pada Gambar 3.6 dimana alamat server NTP lokal akan mengarah pada alamat IP DNS lokal.



Gambar 3.6 Hasil Perintah dig DNS

Sedangkan, hasil yang didapatkan setelah menggunakan perintah *ntpdate* dideskripsikan pada Gambar 3.7 dimana pada tampilan hasil terlihat bahwa stratum lebih kecil dari 16 dan waktu komputer tersinkronisasi dengan baik.



Gambar 3.7 Hasil Ntpdate dari Server Cloud

3.4.2 Instalasi dan Konfigurasi Eucalyptus

3.4.2.1 Server Cloud Controller (CLC)

Untuk menginstal Eucalyptus CLC dapat menggunakan CD instalasi UEC atau dengan menggunakan perintah *apt-get install*. Pada penelitian skripsi ini, server CLC diinstal dengan menggunakan perintah *apt-get install*. APT (Advanced Packaging Tool) merupakan perintah pada terminal Linux yang digunakan untuk menginstal paket software, upgrade paket software yang telah terinstal sebelumnya, memperbaharui index paket, dan bahkan meningkatkan keseluruhan sistem Ubuntu. Jadi, *apt-get install* merupakan command line yang digunakan untuk menginstal paket software yang baru.

Setelah proses instalasi CLC yang terdiri dari CC, Walrus, dan SC, tahap penting yang harus dilakukan adalah melakukan pengaturan IP statik dengan menambahkan perintah pada /etc/network/interfaces menggunakan perintah *sudo vi*. Vi merupakan perintah teks editor sehingga administrator dapat mengubah atau membuat isi dari suatu file, dalam hal ini admin akan menambahkan isi pada file /etc/network/interfaces seperti Gambar 3.8.

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto br0
iface br0 inet static
address 192.168.2.1
netmask 255.255.255.0
broadcast 192.168.2.50
bridge_ports eth0

bridge_fd 9
bridge_hello 2
bridge_maxage 12
bridge_stp on
```

Gambar 3.8 Konfigurasi Jaringan Server CLC

Perintah pada Gambar 3.8 menunjukkan bahwa antarmuka untuk bridge digunakan sebagai antarmuka jaringan dengan menggunakan alamat IP statik untuk server CLC. Perintah *bridge_ports eth0* artinya menambahkan antarmuka ethernet ‘eth0’ ke port bridge atau dengan kata lain melakukan asumsi antarmuka eth0 sebagai port bridge. Kemudian, perintah *bridge_fd 9* merupakan perintah untuk waktu *forward delay* bagi antarmuka bridge. Waktu forward delay adalah

waktu tunda sebuah antarmuka dari keadaan terblokir menjadi keadaan forwarding, dengan kata lain merupakan waktu yang digunakan pada keadaan listening dan learning (mendengarkan dan mempelajari) suatu jaringan.

Perintah *bridge_hello 2* menunjukkan waktu hello yaitu waktu antara tiap BPDU (Bridge Protocol Data Unit) dikirimkan ke suatu port. BPDU merupakan frame data yang digunakan bridge untuk bertukar informasi mengenai identitas bridge, status bridge dan informasi penjaluran. Perintah *bridge_maxage 12* artinya memberikan waktu maxage yaitu waktu untuk mengkontrol waktu maksimum yang digunakan sebelum port bridge menyimpan informasi konfigurasi BPDU.

Ketiga waktu tersebut merupakan pewaktuan pada STP (Spanning Tree Protocol), dimana pewaktuan ketiganya menggunakan format detik. STP ini akan mengizinkan bridge ganda (multiple) untuk bekerja bersama-sama dimana setiap bridge akan berkomunikasi satu sama lain untuk mengetahui bagaimana caranya melakukan interkoneksi antar bridge. Jika stp dinyalakan dengan memberikan perintah 'on' berarti bridge akan mengirimkan dan menerima BPDU dan perintah tersebut menyatakan bahwa terdapat lebih dari satu bridge dalam LAN.

Tahap berikutnya adalah instalasi paket NTP. NTP (Network Time Protocol) merupakan protokol jaringan yang melakukan sinkronisasi pada sistem komputer melalui internet dan melakukan encode waktu menggunakan sistem UTC (Coordinated Universal Time). UTC merupakan standar waktu internasional yang digunakan oleh standar WWW dan Internet. Perangkat keras yang diinstal NTP akan bersikap sebagai NTP Server untuk nodes. Waktu pada semua komponen UEC akan disinkronisasi sehingga CLC yang terkoneksi dengan internet dapat menjalankan NTP server dan komponen lain yang dimiliki CLC akan melakukan sinkronisasi dengan NTP.

Setelah instalasi NTP pada server CLC selesai, tahap penting berikutnya adalah konfigurasi file /etc/ntp.conf.Hal ini untuk memastikan bahwa server NTP menggunakan waktunya sendiri sebagai sumber waktu sehingga saat Internet down, maka sistem CLC dan komponen-komponennya tetap dapat melakukan sinkronisasi waktu dengan server NTP. Kemudian, melakukan restart NTP perlu dilakukan untuk menyimpan konfigurasi yang telah dilakukan dan untuk mengetahui apakah pengaturan sebelumnya memberikan pengaruh pada sistem

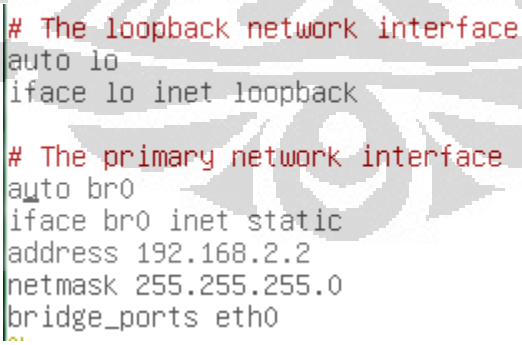
atau tidak. Melakukan restart tidak hanya dilakukan pada NTP, namun juga dilakukan pada CLC (Cloud Controller).

Gambar 3.9 Konfigurasi File Ntp.conf

Pada Gambar 3.9, konfigurasi yang ditambahkan saat penelitian adalah perintah yang ditunjukkan dengan lingkaran merah. Alamat IP untuk server NTP yang digunakan adalah 127.127.1.0.

3.4.2.2 Server Node Controller (NC)

Langkah pertama yang dilakukan untuk konfigurasi pada server NC adalah mengunduh paket software NC menggunakan perintah *apt-get install* sama seperti saat melakukan pengunduhan paket software untuk server CLC. Setelah proses instalasi server NC selesai, tahap berikutnya adalah melakukan konfigurasi jaringan sama seperti pada konfigurasi server CLC. Konfigurasi pada file /etc/network/interfaces untuk server NC yang dilakukan saat penelitian ditunjukkan pada Gambar 3.10.

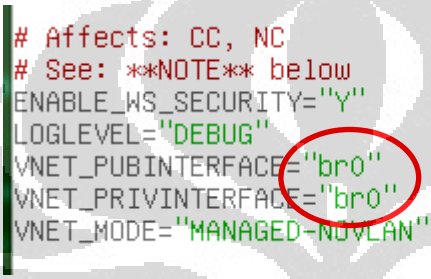


Gambar 3.10 Konfigurasi Jaringan Server NC

Kemudian, tahap selanjutnya adalah menambahkan alamat IP server DNS lokal pada file /etc/resolv.conf dengan menambahkan perintah '`nameserver (alamat_IP_DNS)`'. Pada server NC, paket software NTP juga harus diinstal

dengan konfigurasi file /etc/ntp.conf Pada File ntp.conf pada server NC ditambahkan alamat IP dari server NTP lokal sehingga NC dapat melakukan sinkronisasi dengan server NTP.

Tahap berikutnya adalah melakukan konfigurasi eucalyptus pada NC dengan melakukan perubahan pada file /etc/eucalyptus/eucalyptus.conf seperti yang ditunjukkan lingkaran merah pada Gambar 3.11. Pada server NC akan dilakukan konfigurasi interface ethernet sebagai bridge. NC akan melampirkan antarmuka jaringan untuk bridge tersebut bagi VM yang berjalan sebelum VM mengaktifkan konektivitas jaringannya.



Gambar 3.11 File Konfigurasi Eucalyptus

.

Pada Gambar 3.11 terlihat bahwa mode yang digunakan adalah mode MANAGED-NOVLAN. Mode ini mirip dengan mode MANAGED dalam hal fitur IP dinamik dan grup keamanannya, namun mode MANAGED-NOVLAN tidak mendukung isolasi jaringan VM. Perbandingan karakteristik mode jaringan Eucalyptus ditunjukkan pada Tabel 3.7.

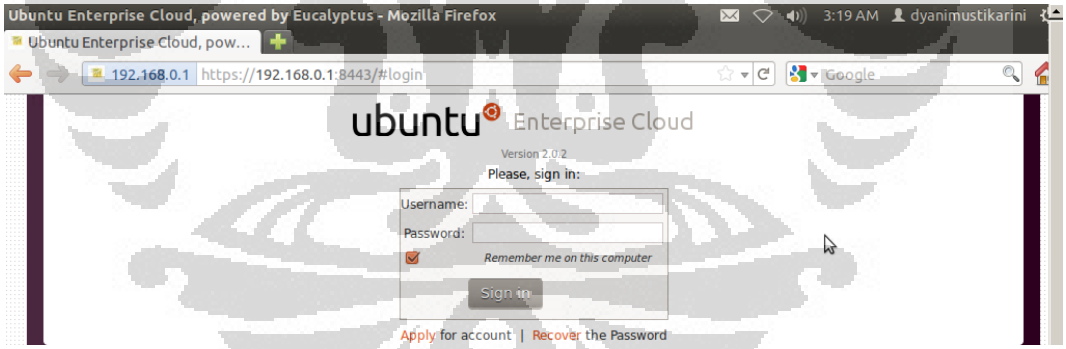
Tabel 3.7 Perbandingan Karakteristik Mode Jaringan Eucalyptus [10]

Mode Jaringan	DHCP server berjalan pada jaringan	CC menjalankan DHCP server sendiri	Isolasi Instance	Private IP	Ingress Filtering
System	Dibutuhkan	Tidak	Tidak	Tidak	Tidak
Static	Tidak dibutuhkan	Ya	Tidak	Tidak	Tidak
Managed	Tidak dibutuhkan	Ya	Ya	Ya	Ya
Managed-Novlan	Tidak dibutuhkan	Ya	Tidak	Ya	Ya

3.4.2.3 Konfigurasi pada Klien (Administrator)

Berikut ini adalah langkah-langkah untuk melakukan konfigurasi pada klien yang menjadi administrator setelah berhasil melakukan konfigurasi pada server cloud yang menggunakan UEC Eucalyptus.

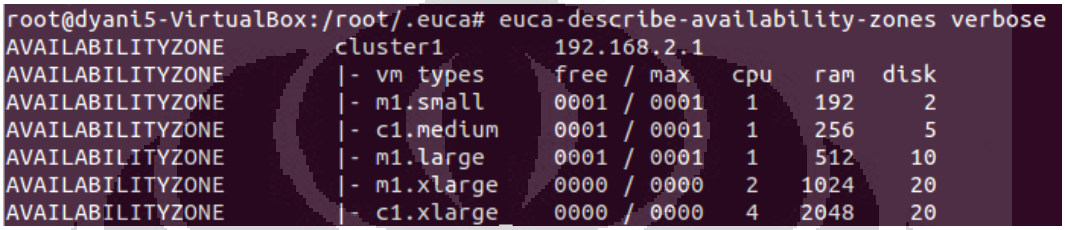
1. Instalasi QEMU-KVM (Quick Emulator - Kernel-based Virtual Machine) untuk membantu dalam melakukan instalasi image pada platform KVM. KVM merupakan infrastuktur virtual untuk kernel Linux. QEMU merupakan emulator prosessor menggunakan translator dinamik untuk memperoleh kecepatan yang baik. QEMU ini akan mendukung virtualisasi untuk model kernel KVM pada Linux.
2. Instalasi euca2ools untuk mengatur cloud dengan menggunakan tools ini.
3. Setelah melakukan instalasi tools euca2ools, terdapat beberapa tahapan yang harus dilakukan pada konfigurasi admin ini, yaitu :
   - Login ke antarmuka web dari CLC menggunakan browser dengan mengakses alamat IP lokal CLC. Setelah tahap ini berhasil maka akan didapatkan hasil seperti tampilan pada Gambar 3.11. Username dan password default adalah admin.



Gambar 3.12 Halaman Login UEC

   - Saat login pertama kali, antarmuka web UEC akan memberikan peringatan kepada admin untuk mengganti password dan memberikan informasi email admin. Setelah selesai melakukan tahap tersebut, arsip credentials harus diunduh dengan mengakses link http://192.168.2.1/#credentials dan menyimpan arsip tersebut kedalam direktori ~/.euca directory. Arsip credentials dapat diekstraksi dahulu dengan perintah *unzip* karena arsip tersebut dalam format .zip.

- Setelah menyimpan arsip credentials, langkah selanjutnya adalah melakukan verifikasi apakah euca2tools dapat berkomunikasi dengan UEC. Perintah . */root/.euca/eucarc* berfungsi menjalankan script eucarc untuk memastikan variabel lingkungan euca2tools dapat berjalan dengan baik. Selanjutnya, perintah *euca-describe-availability-zones verbose* berfungsi mengetahui detail dari ketersediaan cluster local yang telah terdaftar. Hasil dari perintah tersebut akan terlihat seperti pada Gambar 3.13.



Gambar 3.13 Ketersediaan Cluster Lokal

3.4.2.4 Melakukan Registrasi Walrus, Cluster, Storage dan Node Controller

Beberapa tahapan penting agar registrasi komponen-komponen sistem cloud berhasil antara lain adalah

- Pertukaran kunci SSH harus berjalan dengan baik.

  SSH (Secure Shell) merupakan protokol jaringan untuk keamanan komunikasi data menggunakan kriptografi kunci publik untuk autentikasi komputer remote. SSH merupakan protokol yang digunakan untuk mengakses suatu komputer dari komputer lain secara aman. Setelah pertukaran kunci SSH berhasil, maka SSH akan mengizinkan suatu komputer untuk menjalankan command line dan melakukan transfer file.

  Untuk melakukan pertukaran kunci SSH ini dilakukan beberapa tahap berikut ini.
- Pengaturan password untuk Node Controller (NC) dengan perintah *sudo passwd eucalyptus*.
- Melakukan konfigurasi seperti berikut ini pada server CLC:

  ```
  sudo -u eucalyptus ssh-copy-id -i
  /var/lib/eucalyptus/.ssh/id_rsa.pub
  eucalyptus@<IP_OF_NODE>
  ```

Pada penelitian yang telah dilakukan, konfigurasi yang dilakukan untuk pertukaran kunci SSH ini pada CLC adalah

```
sudo -u eucalyptus ssh-copy-id -i /var/lib/eucalyptus/.ssh/id_rsa.pub eucalyptus@192.168.2.2
```

- Penghapusan password pada NC jika diinginkan.
- Pengecekkan pertukaran kunci SSH.

  Pengecekkan kunci SSH dapat dilakukan dengan mengetikkan '`ssh eucalyptus@IP_ADDR_NODE`' pada terminal.

Gambar 3.14 Tampilan Pertama Kali Setelah Berhasil SSH

- Konfigurasi layanan dilakukan dengan baik.
- Setiap layanan dalam hal ini controller melakukan *publishing* atas eksistensinya.

  Publikasi dari setiap controller dilakukan dengan melakukan perintah *start eucalyptus-walrus-publication* untuk Walrus Controller, hal ini juga dilakukan untuk CC (Cloud), SC (Storage) dan NC (Node) dengan menggantikan kata walrus dengan masing-masing controller.
- Komponen UEC *listener* berjalan pada CLC.
- Melakukan registrasi untuk setiap controller menggunakan perintah *euca_conf register*
- Registrasi terverifikasi.

  Untuk melihat apakah registrasi telah terverifikasi dapat dilakukan dengan mengecek file /var/log/eucalyptus/registration.log.

3.4.3 Manajemen Image

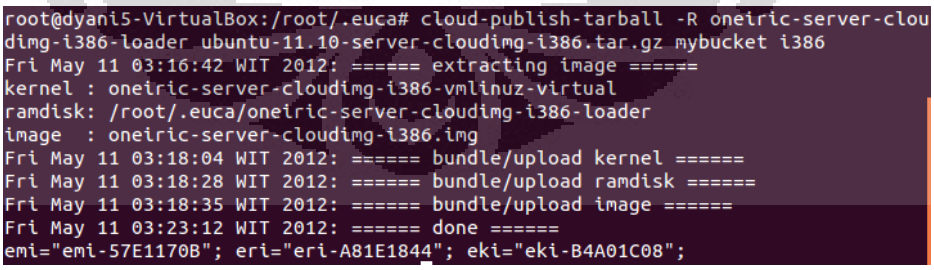
Eucalyptus Machine Image (EMI) merupakan kombinasi mesin yang terdiri dari *virtual disc image*, kernel dan *ramdisk image* serta file xml yang berisi meta data mengenai image. Image ini digunakan sebagai *template* untuk membuat *instances* pada UEC.

Pada penelitian ini digunakan image yang sudah siap pakai sehingga tidak diperlukan pembuatan image sendiri dengan melakukan proses untuk *bundling image*. Proses *bundling image* ini terdiri dari beberapa tahap seperti pembuatan virtual disk image, instalasi OS, instalasi aplikasi yang dibutuhkan, persiapan OS agar berjalan dibawah UEC, dan pendaftaran image dengan UEC. Untuk memperoleh image yang sudah siap pakai dapat diunduh melalui salah satu link yang tersedia di web yaitu http://help.ubuntu.com/community/UEC/Images. Terdapat dua tahap untuk melakukan manajemen image dengan menggunakan image yang telah siap pakai yaitu :

- Registrasi image ke cloud.

  Untuk mendaftarkan image ke cloud dapat menggunakan fasilitas UEC publish tarball. Pada penelitian skripsi ini, image yang digunakan adalah *oneiric-server-cloudimg* untuk prosesor 32 bit. Perintah yang digunakan untuk pendafataran image ini saat penelitian adalah

  ```
  #cloud-publish-tarball -R oneiric-server-cloudimg-i386-loader ubuntu-11.10-server-cloudimg-i386.tar.gz mybucket i386
  ```



Gambar 3.15 Hasil Registrasi Image

- Pengecekkan image yang telah terdaftar.

  Untuk mengecek image yang telah didaftarkan dapat menggunakan perintah *euca-describe-image*, namun sebelumnya perlu dilakukan environment

variable dengan perintah '. *eucarc* dan *source eucarc*'. Hasil dari pengecekkan image ditunjukkan seperti berikut ini.

```
/root/.euca# euca-describe-images
IMAGE eki-B4A01C08    mybucket/oneiric-server-
cloudimg-i386-vmlinuz-virtual.manifest.xml    admin
available public i386 kernel
IMAGE emi-57E1170B    mybucket/oneiric-server-
cloudimg-i386.img.manifest.xml    admin    available
public i386 machine eki- B4A01C08 eri-A81E1844
IMAGE eri-A81E1844    mybucket/oneiric-server-
cloudimg-i386.loader.manifest.xml  admin  available
public i386 ramdisk
```

3.4.4 Manajemen Instance

Untuk berinteraksi dengan instance diperlukan sepasang kunci dengan format .priv. Priv file ini merupakan kunci privat yang digunakan untuk berinteraksi dengan instance, seperti untuk menjalankan instance dan untuk ssh ke instance yang telah berjalan. Berikut ini merupakan beberapa tahapan untuk mengatur instance.

- Pembangkitan kunci privat

  Sebelum membuat kunci privat, diperlukan pengaturan *environment variable* eucalyptus dengan menggunakan tools euca. Perintah untuk membuat kunci privat seperti berikut ini.

  ```
  ~/root/.euca#euca-add-keypair mykey>mykey.priv
  ```

  Setelah hal tersebut dilakukan, maka kunci privat akan terbentuk dan dapat dicek dengan menggunakan perintah *euca-describe-keypairs*. Hasil dari penggunaan perintah tersebut akan seperti tampilan di bawah ini.

  ```
  KEYPAIR mykey 32:83:a7:1e:f0:8e:d9:ac:2f:ff:b8:96
  :b1:22:df:23:32:ec
  ```

- Menjalankan instance

Untuk menjalankan instance harus melakukan pengaturan environment variable, pembuatan kunci privat, dan dapat dilakukan pengecekkan image dan zona yang telah tersedia. Perintah yang digunakan untuk menjalankan instance adalah

```
#euca-run-instances -k mykey -t [model-virtual-machine] [id-image-emi]
```

Saat penelitian skripsi ini, perintah untuk menjalankan instance seperti berikut ini.

```
#euca-run-instances -k mykey -t c1.medium emi-57E1170B
```

- Pengecekkan instance yang telah berjalan

  Untuk melakukan pengecekkan instance dapat menggunakan perintah *euca-describe-instances*. Saat melakukan pengecekkan akan terdapat tampilan seperti berikut ini.

```
RESERVATION r-40920896 admin default
INSTANCE i-327F0645 emi-57E1170B 192.168.2.10 172.19.1.2 pending mykey 0 c1.medium 2012-05-10T20:32:45.03Z cluster 1 eki-B4A01C08 eri-A81E1844
```

  Pada tampilan di atas terlihat status dari instance masih ‘pending’, hal ini berarti instance tersebut belum berjalan sehingga harus dipastikan bahwa instance akan berubah status menjadi ‘running’. Setelah status dari instance berubah, maka tampilan dari pengecekkan instance akan berubah menjadi seperti di bawah ini.

```
RESERVATION r-40920896 admin default
INSTANCE i-327F0645 emi-57E1170B 192.168.2.10 172.19.1.2 running mykey 0 c1.medium 2012-05-10T20:48:35.484Z cluster 1 eki-B4A01C08 eri-A81E1844
```

- Pengecekkan console output untuk instance tertentu

Perintah untuk mengecek hasil pada console untuk instance yang telah berjalan dapat dilakukan dengan mengetikkan perintah berikut ini pada terminal.

```
#euca-get-console-output [ID_RUNNING_INSTANCE]
```

Pada saat penelitian pengecekkan console output dilakukan untuk instance dengan ID `i-327F0645`, sehingga perintah yang dilakukan akan seperti berikut ini.

```
~/root/.euca#euca-get-console-output i-327F0645
```

Hasil dari pengecekkan console output untuk instance tersebut akan terlihat seperti Gambar 3.16.

```
i-327F0645
2012-05-10-T20:55:35.48Z
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Linux version 3.0.0-12-generic-pae (
) (gcc version 4.61(Ubuntu/Linaro 4.6.1-9Ubuntu3))
#20-Ubuntu SMP Fri Oct 7 16:37:17 UTC 2011 (Ubuntu
3.0.0-12-generic-pae)
[    0.000000] KERNEL supported cpus :
[    0.000000] Intel GenuineIntel
[    0.000000] AMD AuthenticAMD
[    0.000000] NSC Geode by Geode
[    0.000000] Cyrix CyrixInstead
[    0.000000] Centaur CentaurHauls
[    0.000000] Transmeta GenuineTMx86
[    0.000000] Transmeta TransmetaCPU
[    0.000000] UMC UMC UMC UMC
...
...
...
ec2:
#############################################################
########
ec2: BEGIN
SSH HOST KEY FINGERPRINTSec2:
2048 f0:5f:fa:00:99:34:df:1f:6c:c9:de:30:ec:ef:15:85
/etc/ssh/ssh_host_rsa_key.pub (RSA)
ec2: 1024
e1:d1:ca:51:9f:b4:a0:ae:ea:b0:f2:fe:9c:64:84:a0
```

Gambar 3.16 Hasil Euca Consol

- SSH login ke instance tertentu

Sebelum melakukan SSH, admin melakukan autorisasi dengan menggunakan perintah berikut ini.

```
#euca-authorize default –P tcp –p 22 s 0.0.0.0/0
```

Perintah tersebut akan menghasilkan grup baru dimana protokol yang digunakan adalah tcp dengan port 22 untuk subnetmask 0.0.0.0 yang berarti tidak didefinisikan. Grup dari perintah tersebut bertujuan memberikan akses untuk trafik SSH. Untuk mengecek grup yang terdapat pada sistem dapat menggunakan perintah *euca-describe-groups*. Hasil dari perintah diatas dapat dicek menggunakan perintah *euca-describe-groups*sehingga akan memberikan tampilan seperti berikut ini.

```
PERMISSION admin default ALLOWS tcp 22 22 FROM CIDR
0.0.0.0/0
```

Proses selanjutnya adalah melakukan SSH ke instance. Untuk melakukan SSH login ke instance digunakan perintah seperti berikut.

```
# ssh –i [KEY_PAIRS]
[username@IP_Instances/IP_Public_Eucalyptus]
```

Pada saat penelitian perintah yang dilakukan untuk SSH login ke instance adalah

```
~/root/.euca# ssh –i mykey.priv ubuntu@192.168.2.10
```

Setelah SSH login berhasil, maka admin sudah login ke suatu instance yang berjalan sehingga admin dapat melakukan *sudo su* (berperan sebagai superuser), kemudian admin dapat menginstalasi suatu aplikasi yang dibutuhkan. Saat admin telah login ke instance, maka IP yang digunakan untuk antarmuka ethernet adalah IP privat dari Eucalyptus yaitu 172.19.1.2.

3.4.5 Managemen Storage (Volume)

Storage merupakan tempat menyimpan data dengan kapasitas tertentu. Pada sistem Eucalyptus Cloud, kapasitas dari storage ini dapat ditentukan untuk masing-masing cluster. Namun, pada penelitian ini hanya digunakan satu cluster. Untuk menciptakan volume yang diinginkan bagi suatu cluster dapat digunakan perintah seperti berikut.

```
~/root/.euca# euca-create-volume –s 10 –z cluster 1
```

Perintah tersebut berarti menciptakan storage sebesar 10 GB untuk zona cluster1. Hasil dari perintah tersebut akan terlihat seperti dibawah ini.

```
VOLUME vol-4E108021 10 creating 2012-05-19T20:03:02.827z
```

Jika ingin melakukan attach volume yang telah diciptakan dengan menggunakan perintah di atas pada suatu instance dapat dilakukan dengan mengetikkan perintah berikut ini.

```
~/root/.euca# euca-attach-volume –i i-327F0645 –d /dev/sda vol-4E108021
```

Perintah di atas berarti melakukan attach volume menggunakan volume yang telah dibuat pada instance dengan ID tertentu pada divais disk SCSI (divais yang digunakan untuk hard disk). Setelah menciptakan volume yang diinginkan, untuk mengecek volume tersebut dapat menggunakan perintah *euca describe volumes* dan status dari volume tersebut harus diperhatikan. Jika sudah terpakai, maka status dari volume tersebut akan 'in-use' sedangkan jika belum maka statusnya masih 'available'.

3.5 Algoritma RSA Pada Virtualisasi *Private Cloud*

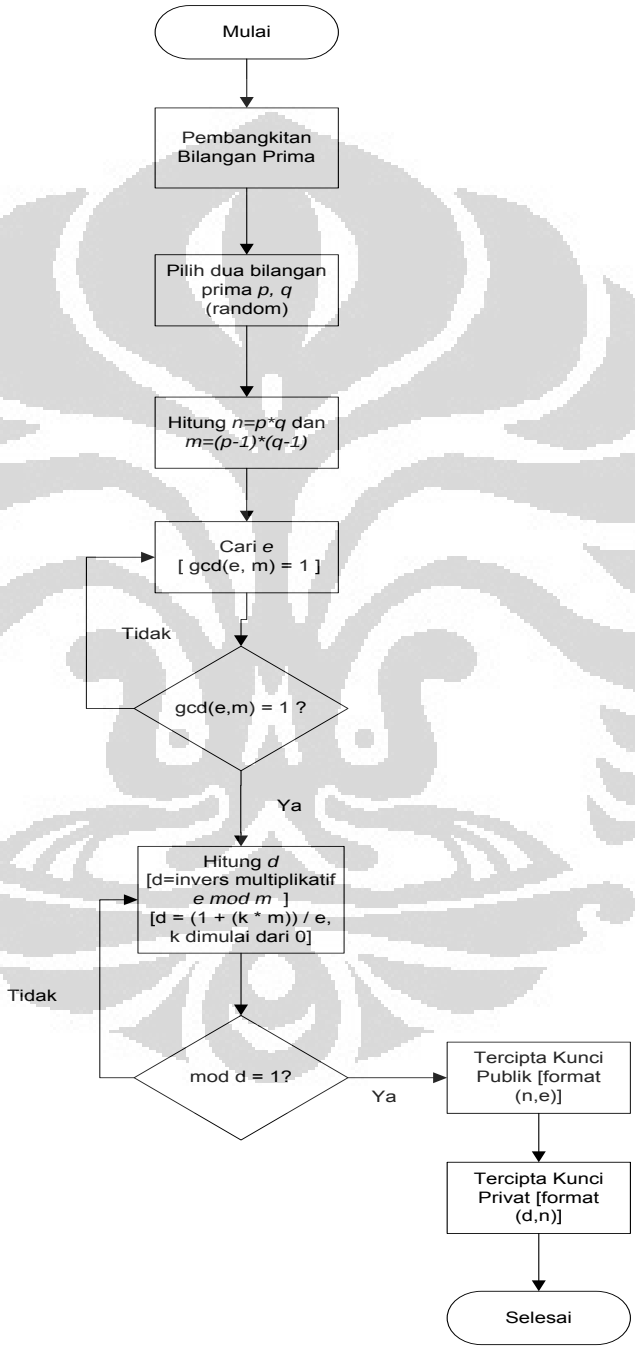
3.5.1 Algoritma RSA dengan C++

C++ merupakan bahasa pemrograman yang dikembangkan oleh Bjarne Strousop pada Bell Labs pada tahun 1979. C++ merupakan salah satu bahasa pemrograman yang terkenal dan cukup sering digunakan. Aplikasi domain yang dimiliki c++ antara lain adalah sistem software, driver divais, embedded sontfware, aplikasi client-server, dan software entertainment seperti video games. C++ adalah *structured programming* atau pemrograman terstruktur yang berarti bahasa ini mendukung perancangan *top down*, membagi program menjadi modul-modul dan menggunakan pengkodean terstruktur seperti struktur barisan dan struktur iterasi.

Pada implementasi sistem RSA ini menggunakan bahasa pemrograman C++. Pada sistem RSA tersebut terdiri dari program utama yaitu main.cpp yang menjadi program pemanggil modul-modul lainnya. Modul-modul yang lain antara

lain adalah modul untuk pembangkitan kunci RSA, modul untuk melakukan enkripsi dan dekripsi, dan modul utilitas untuk menjabarkan kebutuhan-kebutuhan yang diperlukan program lainnya seperti kebutuhan untuk konversi hexadecimal ke desimal.

3.5.2 Flowchart Pembangkitan Kunci RSA



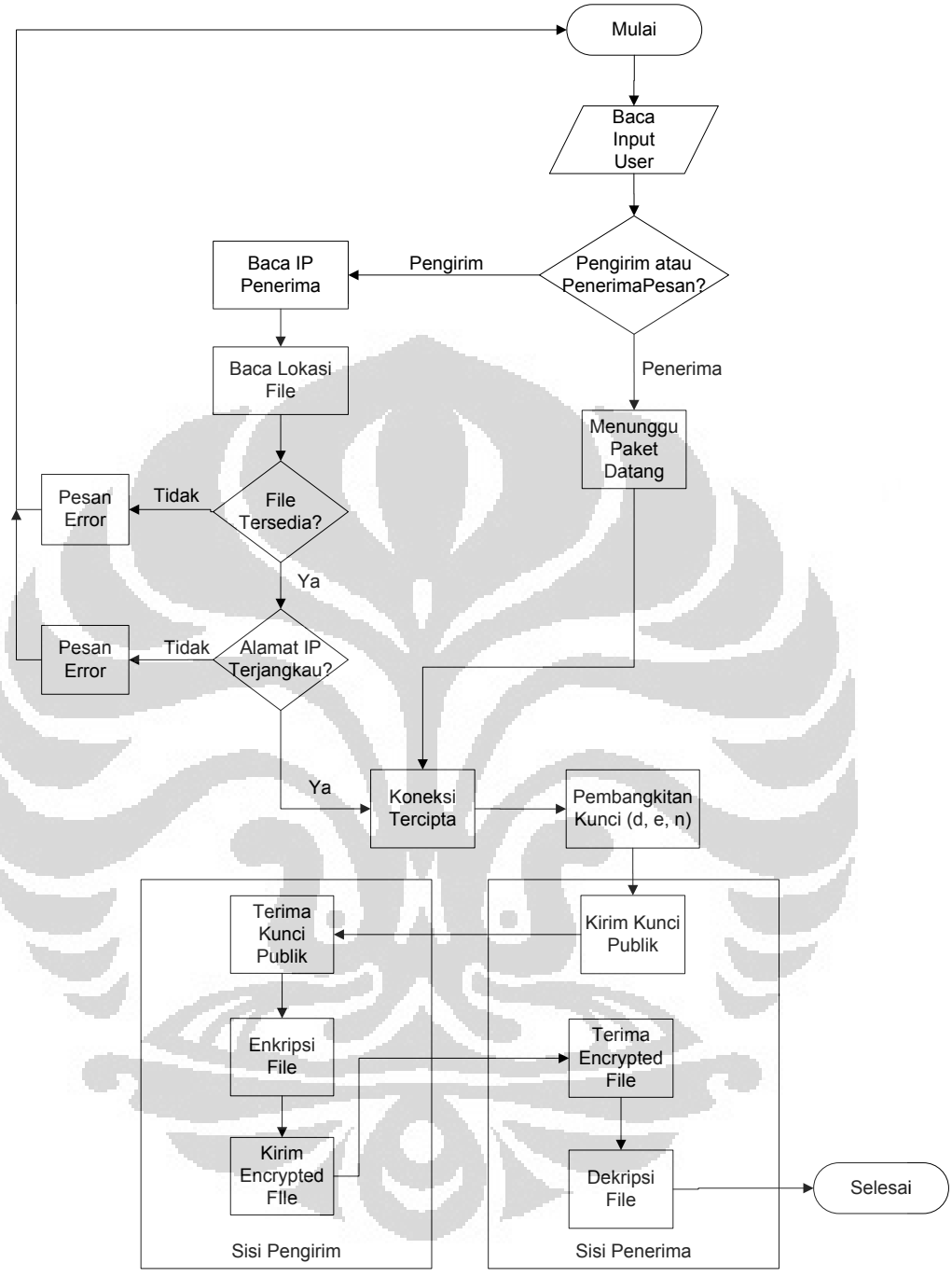
Gambar 3.17 Flowchart Pembangkitan Kunci e, d, dan n [11]

3.5.3 Pseudocode Pembangkitan Kunci

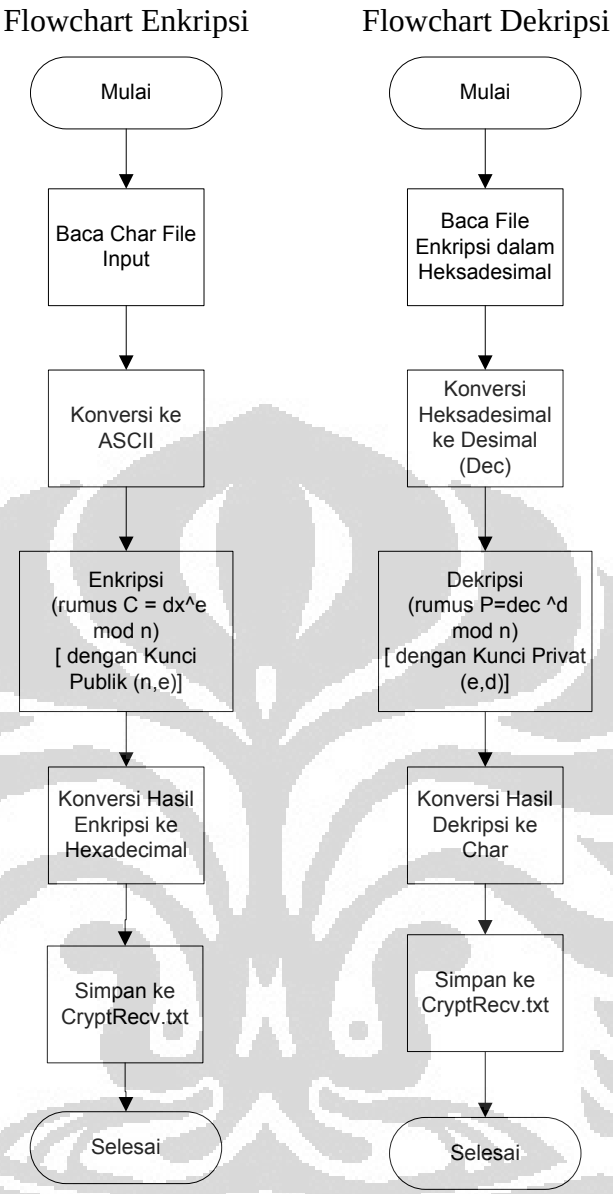
```
#generate bilangan prima secara random
void generatePrime(int prime, int idxPrime){
bool Prime [maximal_bilangan+1]
memset(Prime, true, sizeof(Prime))
for (i=2, i*i<=maximal_bilangan, i++)
        if (Prime[i]) {
                k = 3,  j=i*2
                while (j <= maximal bilangan){
                setting false bagi j untuk bilangan prima
                j = ik
                k++
}
for (i=2, i<=maximal bilangan, i++)
        If (Prime[i]) prime[++idxPrime] =i
}

#gcd/greatest common divisior
int gcd(a,b)
 while b tidak sama dengan 0 {
        c = a modulus b
        a = b
        b = c
} return a

#generate nilai e dan d
void generate_e_d(){
        #panggil fungsi untuk membangkitkan bilangan prima
        generatePrime(prime, idxPrime)
        a=prime[rand() mod idxPrime+1]
        do {b= prime[rand() mod idxPrime+1]
                }while (a=b)
        #tampilkan bilangan prima p dan q
        print ('p = ', a)
        print ('q = ', b)
        #menghitung nilai n dan m
        n = a b
        m = (a-1)(b-1)
        print n
        print m

        #generate e dan d
        gcd_e
        /*mencari nilai e coprime untuk m hingga mendapatkan bilangan yang*/
        /*terbesar yang dapat membagi e dan m untuk menghasilkan nilai 1*/
        do {
                e = prime[rand() mod idxPrime+1]
                gcd_e = gcd(e,m)
```

```
        print ('Trying e with ...', e)
        if e>=m OR gcd_e tidak sama dengan 1 print 'no'}
        while (e>=m OR gcd_e tidak sama dengan 1) print 'yes'

        k = 0, mod_d
        do {
                ++k
                mod_d = (1 + (k * m)) % e  /*rumus modulus d*/
                d = (1 + (k * m)) / e  /*rumus nilai d*/
                print ('Trying d with ...',d)
                if mod_d !=0 print 'no'
                } while mod_d!=0 print 'yes'
        print ('e = ', e)
print('d = ', d)
```

3.5.4 Flowchart Sistem Enkripsi dan Dekripsi RSA



Gambar 3.18 Flowchart Keseluruhan Sistem RSA [11]



Gambar 3.19 Flowchart Enkripsi dan Dekripsi RSA [11]

3.5.5 Pseudocode Sistem Keseluruhan RSA

```
#opsi pendek
const char short_options = "rdvVh"

#opsi panjang
const struct option long_options[] = {
  { receive , 'r'},
  { no-delete,  'd'},
  { version, 'v'},
  { verbose,   'V'},
  { help,      'h'},
```

```
  { NULL,  0 }
}

string cryptSendTemp = cryptSend.txt /*cryptSend bersifat sementara dan
                                      /*hanya terlihat saat pengiriman
                                     /*data*/
string cryptRecvTemp = cryptRecv.txt /*file hasil enkripsi disimpan dalam
                                      /*file txt*/

main(int argc, char argv)
bool isSender = true
int e, d, n
string filename
string ipaddr

pos =1
int next_option
#membaca opsi yang dimasukkan oleh pengguna
do{
next_option = getopt_long(argc, argv, short_options, long_options, NULL)
switch (next_option) {
     case r: isSender    = false
     case d: delRecvTemp = false
     case V: verboseMode = true
     case v: print_version(stdout, 0)
     case h: print_usage(stdout, 0)
     }
++pos
}while next_option != -1

#jika pengguna sebagai pengirim
if(isSender){
/*mengecek alamat IP*/
if (pos < argc) {
ipaddr = argv[pos++];
     if (strcmp(ipaddr.c_str() = 0)
   } print_usage(stderr, 1) /*print halaman help*/

/*mengecek ketersediaan file*/
if(pos < argc){
filename = argv[pos]
jika file yang ingin dikirimkan tidak tersedia, print('File not found')
exit}}
/*mengecek apakah alamat ip terjangkau*/
if (isSender)
print('Checking connection to ...', ipaddr.c_str())
     sleep
jika socket bermasalah print('socket error')
```

```
exit
jika alamat ip tidak terjangkau print (‘failed. Cannot connect’)
exit
else print(‘Ok’)

print(‘Receiving public keys...’) /*menerima kunci publik dari sisi penerima*/
jika ada error print(‘failed. cannot receive public keys’) print terminasi
program
exit
jika berhasil print(‘Ok’)

/*melakukan proses enkripsi file*/
print(‘Encrypting message’)
if (encrypt(filename, cryptSendTemp, e, n)) printf(‘success’)
   else {
     print(‘failed’)
     print terminasi program
     exit
   }

#jika pengguna sebagai penerima
listen()
print(‘Waiting for packets’)
jika koneksi tercipta dengan sisi pengirim, print(‘Connection Received’)
print(‘Generating random keys’) /*sisi penerima membangkitkan kunci
                                  /*publik*/
generate_e_d(e,d,n, verboseMode) /*memanggil fungsi pembangkitan nilai e
                                  /*dan d*/

print(‘Receiving encypted packets’)
FILE *fout
bool isFirst = true
     fout = fopen(cryptRecvTemp.c_str(), ‘w’)
     tmp = recv(client_sockfd, buf, BUFSIZ, 0)
     else {
       while (strcmp(buf, ‘fin’) != 0) {
         int send_tmp = send(client_sockfd, ‘ok’, 2, 0)
         if (!isFirst) fprintf(fout, " ")
         isFirst = false
         fprintf(fout, buf)
         tmp = recv(client_sockfd, buf, BUFSIZ, 0)
         buf[tmp] = '\0'
       }
       fprintf(fout, ‘\n’)
       fclose(fout)
       recv(client_sockfd, buf, BUFSIZ, 0)
       filename = buf
       printf(‘ok’) /*paket enkripsi berhasil diterima*/
```

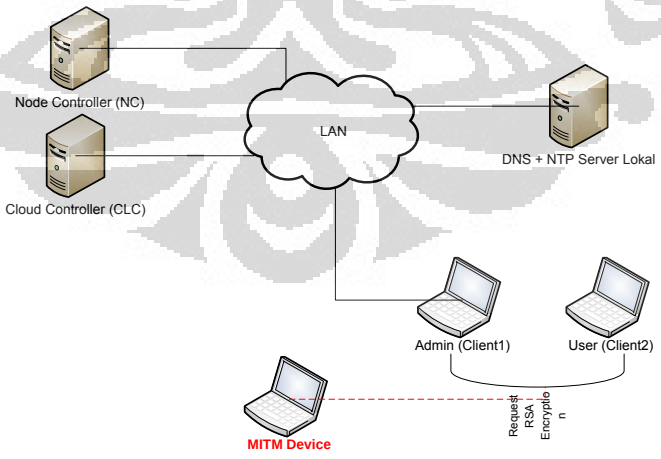
```
print('Decrypting packets')
        string newfilename
        parse_filename(filename, newfilename)
        newfilename = 'received/'+ newfilename
        if (decrypt(cryptRecvTemp, newfilename, d, n)) print('success')
        else {
          print('failed')
          print terminasi program
          exit
    printf('Packets received successfully!') /*paket berhasil didekripsi*/
    exit
```

3.5.6 Pseudocode Sistem Enkripsi dan Dekripsi RSA

```
#proses enkripsi file
int encrypt(string f1, string f2,  e,  n) {
  FILE *fin, *fout
  fin = fopen(f1.c_str(), 'r') /*membaca file atau read*/
  fout = fopen(f2.c_str(), 'w') /*melakukan write file*/
  if (fin = NULL OR fout = NULL)
return 0

/*membaca tiap karakter dalam file*/
char ch
  int counter = 0
 jika pembacaan karakter belum berakhir atau !=End of File{
    dx = ch
    bgmod = bigmod(dx, e, n) /*karakter dx dienkripsi menggunakan*/
                             /*rumus dx^e modulus n dengan memanggil*/
                             /*fungsi bigmodulus*/
    if (counter != 0) fprintf(fout, " ")
          ++counter
    fprintf(fout, "0x%X", bgmod) /*kode enkripsi  dalam format heksadesimal*/
  }
  fprintf(fout)
  fclose(fin)
  fclose(fout)
  return 1
}

#proses dekripsi file
int decrypt(string f1, string f2,  d,  n) {
  FILE *fin, *fout
  fin = fopen(f1.c_str(), 'r')
  fout = fopen(f2.c_str(), 'w')
```

```
 if (fin = NULL OR fout = NULL) return 0;
 char st[nilai array st], hex[nilai array hex]
 jika pembacaan string belum berakhir atau !=End of File {
  int idx = 0, p = 0
  for (int i=0; i<strlen(st); i++) {
   if (p) {
    hex[idx] = st[i]
    hex[idx+1] = null
    ++idx
   }
   if (st[i] == 'x') {
    hex[idx] = null
    p = 1
   }
  }
  int dec = hexToDec(hex) /*file enkripsi heksadesimal dikonversi ke desimal*/
                          /*sebelum didekripsi*/
  int bgmod = bigmod(dec, d, n) /*desimal didekripsi menggunakan
                                /*rumus dec^d modulus n dengan memanggil
                                /*fungsi bigmodulus*/

  fprintf(fout, %c, (char)bgmod)
 }
 fclose(fin)
 fclose(fout)
 return 1
}
```

3.6 Skenario Pengujian Pada Virtualisasi Private Cloud



Gambar 3.20 Topologi Pengujian Pada Sistem Eucalyptus Cloud

Skenario yang digunakan pada penelitian ini yaitu pengujian keamanan pengiriman data menggunakan tools security tertentu untuk sistem enkripsi RSA

yang diterapkan pada Eucalyptus *cloud*. Selain pengujian keamanan juga akan dilakukan pengujian terhadap waktu eksekusi, enkripsi dan dekripsi dari implementasi kriptografi RSA pada sistem Eucalyptus *cloud*. Skenario tersebut bertujuan untuk melakukan analisis dan mengamati pengiriman data menggunakan program enkripsi RSA serta menganalisa keamanan dari hasil uji coba. Tools yang digunakan untuk melakukan sniffing adalah Wireshark dimana tools ini akan menangkap transfer data yang terjadi pada jaringan *cloud* sehingga akan terlihat hasil dari penerapan program RSA untuk pengiriman data pada sistem *cloud*.

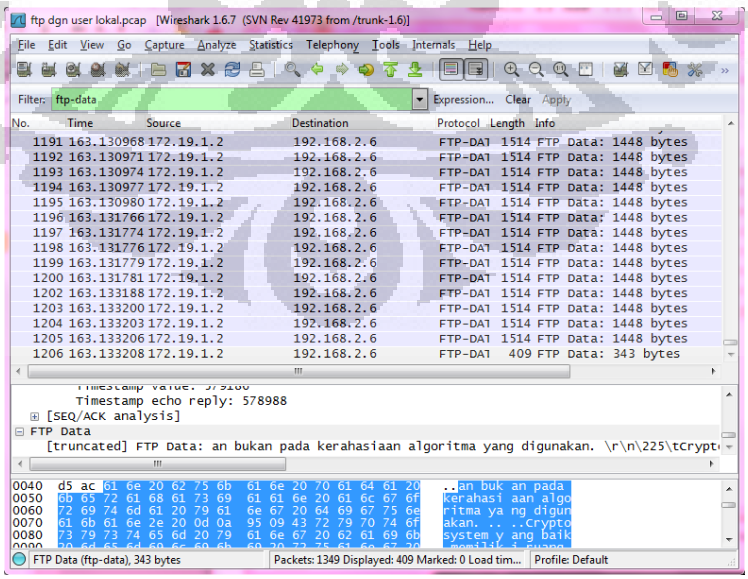
BAB 4
UJI COBA DAN ANALISA

4.1 Uji Coba dan Analisa Pengiriman Data pada Sistem Eucalyptus *Private Cloud*

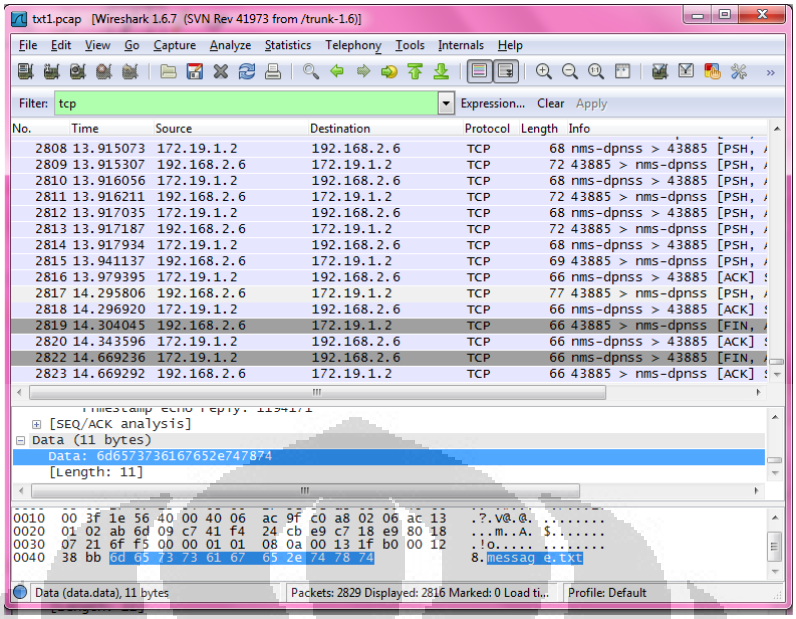
4.1.1 Uji Coba Program Enkripsi RSA

Uji coba untuk keamanan pengiriman data dilakukan dengan satu skenario yaitu melakukan uji coba untuk transfer data antara VM (atau *instances*) sistem Eucalyptus dengan user lokal. Program enkripsi RSA ini harus diinstalasi pada sisi pengirim dan penerima. Setelah kedua mesin mengaktifkan program enkripsi ini, maka kedua mesin dapat berkomunikasi dan melakukan pengiriman data sesuai dengan aliran kerja sistem enkripsi RSA yang telah dijelaskan pada Bab 3.

Tahap pertama adalah melakukan uji coba dengan menggunakan capture Wireshark. Paket yang akan dilakukan capture adalah paket TCP. Gambar 4.1 merupakan salah satu contoh hasil capture program enkripsi RSA menggunakan Wireshark yang telah terenkripsi tidak seperti pengiriman FTP dimana isi file akan terlihat saat transfer data. Kemudian tahap yang berikutnya adalah memperhitungkan waktu program RSA dari hasil penangkapan Wireshark dan merekam nilai-nilai dari kunci RSA.



Gambar 4.1 Capture Ftp-data Menggunakan Wireshark



Gambar 4.2 Capture Program RSA Menggunakan Wireshark

Hasil pertama untuk uji coba skripsi ini adalah tabel dan grafik antara ukuran data yang akan dikirimkan dengan waktu eksekusi untuk pembangkitan kunci, enkripsi, pengiriman paket enkripsi, dekripsi, hingga paket yang telah didekripsi diterima pada sisi penerima. Hasil pertama ini yang akan menjadi dasar untuk analisa waktu enkripsi dan dekripsi. File yang digunakan untuk uji coba adalah file dengan format .txt (file teks), .doc (file Ms. Word), dan .pdf (file Portable Document Format). Untuk setiap format file yang dikirimkan dengan ukuran data yang berbeda-beda dilakukan uji coba sebanyak 10 kali. Tabel 4.1 merupakan tabel waktu eksekusi program dengan banyaknya data dalam kB (kilo Bytes) dimana waktu ekseskusi tersebut merupakan waktu rata-rata dari 10 kali pengambilan data.

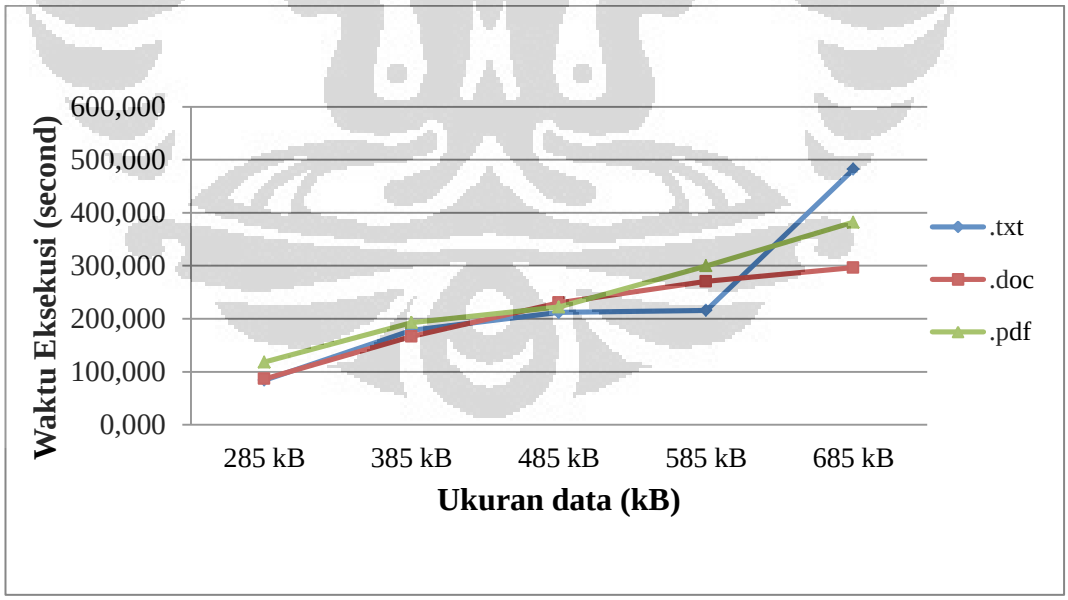
Tabel 4.1 ini diperoleh dari perhitungan selisih waktu yang ditangkap oleh Wireshark. Untuk menghitung waktu eksekusi dilakukan dengan menghitung selisih waktu *acknowledgement* dari paket yang telah selesai didekripsi di sisi penerima dikurangi dengan waktu pertama kali inisialisasi koneksi sisi pengirim dan penerima. Setelah melakukan perhitungan selisih waktu ini kemudian dilakukan perhitungan rata-rata untuk uji coba yang dijalankan 10 kali untuk setiap format dan ukuran file.

Tabel 4.1 Data untuk Format File .txt

Percobaan	txt 285 kB	txt 385 kB	txt 485 kB
1	117.275 detik	110.960 detik	479.317 detik
2	73.270 detik	248.504 detik	201.221 detik
3	73.274 detik	206.480 detik	161.256 detik
4	80.552 detik	173.608 detik	121.336 detik
5	77.899 detik	153.533 detik	144.284 detik
6	93.817 detik	148.720 detik	258.071 detik
7	74.478 detik	126.930 detik	210.984 detik
8	72.314 detik	153.533 detik	161.482 detik
9	90.826 detik	273.120 detik	209.452 detik
10	87.667 detik	185.668 detik	172.085 detik

Tabel 4.2 Rata-Rata Waktu Eksekusi Program Enkripsi RSA

Format File	Data 1 285 kB	Data 2 385 kB	Data 3 485 kB	Data 4 585 kB	Data 5 685 kB
.txt	84.137 detik	178.106 detik	211.949 detik	215.759 detik	482.254 detik
.doc	86.532 detik	166.697 detik	230.508 detik	270.492 detik	296.433 detik
.pdf	118.268 detik	192.788 detik	222.316 detik	299.880 detik	382.399 detik



Gambar 4.3 Grafik Waktu Eksekusi Versus Ukuran Data

4.1.2 Analisa RSA

Pada subbab analisa RSA ini akan dibahas menjadi beberapa analisa, yaitu, analisa dari grafik waktu eksekusi program enkripsi RSA versus banyak data, kemudian analisa hasil yang ditunjukkan dengan gambar capture Wireshark. Lalu, yang ketiga adalah analisa perhitungan program untuk pembangkitan kunci RSA dibandingkan dengan perhitungan manual, analisa dari grafik waktu enkripsi dan dekripsi RSA, serta yang terakhir adalah analisa keamanan dari program RSA pada skripsi ini. Namun, secara garis besar analisa pada skripsi ini terdiri dari dua poin penting yaitu analisa waktu dari implementasi RSA dan analisa keamanan pengiriman data dari implementasi RSA pada sistem Eucalyptus Cloud.

4.1.2.1 Analisa Waktu dari Implementasi Kriptografi RSA

Waktu yang akan dianalisa dari implementasi kriptografi RSA adalah waktu eksekusi program RSA, waktu enkripsi, dan waktu dekripsi. Sebelum dibahas lebih lanjut mengenai analisa waktu, maka akan dibahas terlebih dahulu mengenai capture atau penangkapan trafik menggunakan Wireshark untuk menunjukkan pengaruh dari implementasi kriptografi RSA untuk pengiriman data antar mesin virtual Eucalyptus dengan pengguna lokal.

Pada Gambar 4.1, terlihat saat pengguna dan *instances* Eucalyptus melakukan transfer data, maka isi file akan terlihat karena tidak dilakukan enkripsi. Sedangkan, pada Gambar 4.2 terlihat bahwa isi file tidak dapat terbaca antara pengguna dengan *instances* saat terjadi transfer data. Pada Gambar 4.2, saat file sampai pada alamat IP tujuan maka Wireshark hanya dapat menangkap nama file saja yaitu message.txt saat file sampai tujuan sedangkan isi dari file tidak dapat terlihat karena pengiriman data antara user dengan *instances* menggunakan program enkripsi RSA. Selain itu, Wireshark juga dapat menangkap file yang telah terenkripsi dalam format heksadesimal.

Gambar 4.3 menunjukkan grafik waktu eksekusi program enkripsi RSA dengan banyaknya data yang dienkripsi. Pada grafik tersebut terlihat bahwa waktu eksekusi mengalami kenaikan sebanding dengan banyaknya data yang dienkripsi sehingga ukuran data yang semakin besar akan membutuhkan waktu yang lama. Pada format file .txt, untuk setiap penambahan ukuran data terjadi peningkatan waktu eksekusi rata-rata sebesar 31,44%. Sedangkan untuk format file .doc

peningkatan rata-rata waktu eksekusiprogram adalah 24,83% dan untuk format file .pdf peningkatan rata-rata waktueksekusi adalah 24,85%.

Peningkatan waktu eksekusi untuk setiap format file disebabkan program enkripsi RSA ini membaca setiap karakter yang ada dalam file sehingga membutuhkan proses untuk melakukan enkripsi pada setiap karakter dalam file begitu pula untuk proses dekripsinya. Jadi, semakin banyak karakter dalam file yang dibaca oleh program akan menyebabkan waktu eksekusi lebih lama.

Pada program enkripsi RSA ini masih sederhana dengan bilangan prima acak yang digunakan masih sedikit yaitu dengan batas maksimum sebesar 256 (yang berarti paling besar bilangan prima 251) sehingga enkripsinya kurang kuat. Pada saat uji coba, misalnya hasil bilangan prima yang dibangkitkan oleh program adalah 137 dan 181, maka perhitungan manual pembangkitan kunci program enkripsi RSA akan seperti berikut ini.

1. Bilangan prima p dan q.

   Semakin besar bilangan prima yang dibangkitkan, maka akan semakin banyak faktorialnya sehingga data akan sulit dipecahkan dalam waktu singkat oleh *cracker*.

   p = 137

   q = 181

2. Nilai n.

   Seperti yang telah diungkapkan pada Bab 2, bahwa nilai n merupakan hasil perkalian dari bilangan prima p dan q.

   n = pq = 24797

3. Nilai m.

   Nilai m merupakan hasil perkalian dari bilangan p dikurangi 1 dengan bilangan q dikurangi 1.

   m = (p-1)(q-1) = 136*180 = 24480

4. Angka e *coprime* untuk m.

   Nilai e *coprime* untuk m berarti bilangan terbesar yang dapat membagi e dan m untuk menghasilkan nilai 1. Pembagi tersebut disebut dengan gcd (gratest common divisor).

   e = 107 → gcd (e, 24480) = 1 (ya)

pembagi untuk 107 → 1, 107

pembagi untuk 24480 → 1,2,3,4,5, ...

gcd (107, 24480) =1

Hasil nilai e yang dibangkitkan oleh program adalah 10. Hal ini menunjukkan bahwa nilai e coprime untuk m sesuai dengan teori algoritma RSA yaitu ma dengan 107 karena common divison terbesar untuk angka 107 dan 24480 sama dengan 1.

5. Nilai d $(d \cdot e\%m = 1)$

Nilai d dengan rumus $d \cdot e\%m = 1$ sama halnya dengan mencari nilai d yang memenuhi untuk $d \cdot e = 1 + m \cdot n$ yaitu $d = (1 + m \cdot n)/e$ dengan n adalah bilangan integer.

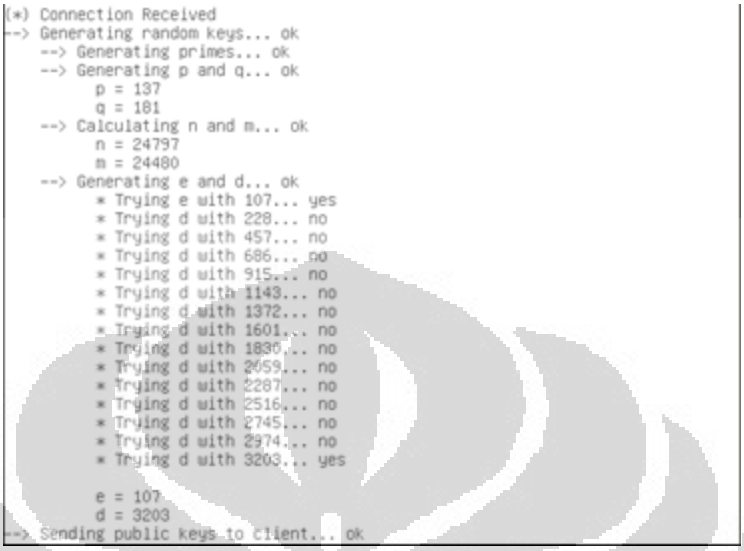
Tabel 4.3 Perhitungan Nilai d Secara Manual

integer n	d	Kebenaran nilai d
1	24481/107 = 228	Tidak
2	48961 / 107 = 457	Tidak
3	73441 / 107 = 686	Tidak
4	97921 / 107 = 915	Tidak
5	122401 / 107 = 1143	Tidak
6	146880 / 107 = 1372	Tidak
7	171361 / 107 = 1601	Tidak
8	195841 / 107 = 1830	Tidak
9	220321 / 107 = 2059	Tidak
10	244801 / 107 = 228	Tidak
11	269280 / 107 = 2516	Tidak
12	293761 / 107 = 2745	Tidak
13	318241 / 107 = 2974	Tidak
14	342721 / 107 = 3203	Ya

Pada Tabel 4.2, kebenaran nilai d **'Tidak'** berarti $(1 + m \cdot n)$ **tidak habis dibagi** e sehingga nilai d **tidak digunakan**, contohnya untuk integer n = 1 dengan nilai d = 228 tidak digunakan karena pembagian nilai 24481 tidak habis dibagi 107. Sedangkan, kebenaran nilai d **'Ya'** berarti nilai $(1 + m \cdot n)$ **habis dibagi** e sehingga nilai d **digunakan**.

Berdasarkan Tabel 4.2 saat n = 14 maka sisa hasil bagi akan sama dengan 0 yang akan memenuhi persamaan $d \cdot e = 1 + m \cdot n \rightarrow 3202 \cdot 107 = 1 + 24480 \cdot 14$. Oleh sebab itu, nilai d adalah 3203. Perhitungan di atas sama

seperti hasil pembangkitan kunci program RSA seperti Gambar 4.4 sehingga nilai d = 3203 pada program sesuai dengan teori algoritma RSA.



Gambar 4.4 Pembangkitan Kunci dari Program RSA

Tabel 4.4 Contoh Perhitungan Program dan Perhitungan Manual

p	q	Perhitungan program				Perhitungan manual			
		n	m	e	d	n	m	e	d
137	181	24797	24480	107	323	24797	24480	107	323
7	67	469	396	181	361	469	396	181	361
167	13	2171	1992	223	1063	2171	1992	223	1063

Berdasarkan Tabel 4.3, terlihat bahwa perhitungan pembangkitan kunci secara manual seperti yang telah dijabarkan sebelumnya akan memiliki nilai yang sama dengan perhitungan program sehingga pembangkitan kunci pada program sudah sesuai dengan teori yang dijabarkan pada Bab 2. Kesuksesan pembangkitan kunci akan berpengaruh pada proses enkripsi dan dekripsi, apabila pembangkitan kunci RSA ini salah atau error maka proses enkripsi dan dekripsi tidak akan berhasil.

Setelah membangkitkan kunci RSA, hal yang penting berikutnya adalah enkripsi dan dekripsi file input yang dipilih. Berikut ini merupakan analisa perhitungan enkripsi dan dekripsi secara sederhana.

1. Enkripsi

Misalnya karakter yang akan dienkripsi adalah ‘A’, jika dikonversi menggunakan tabel ASCII, maka nilai desimal dari ‘A’ adalah **65**. Proses enkripsi dilakukan dengan rumus berikut.

$$C = P^e \% n$$

$$C = 65107 \% 24797 = 4745$$

2. Dekripsi

Setelah file enkripsi diterima, proses dekripsi dilakukan dengan menggunakan kunci privat hasil file enkripsi dalam heksadesimal dikonversikan ke desimal.

$$P = C^d \% n$$

$$P = 47453203 \% 24797 = 47453 \times 47453200 \% 24797$$

$$P = 47453 \times (474532)100 \% 24797$$

$$P = 106833793625 \times (4, \ldots\ldots \times 10117)100 \% 24797$$

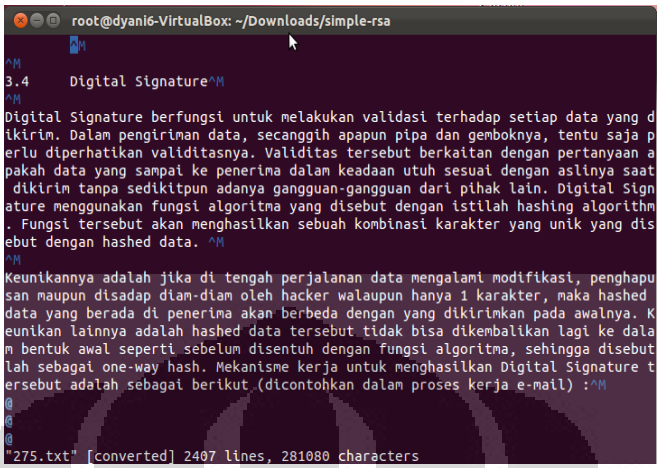
$$P = 106833793625 \times (4, \ldots\ldots \times 10117 \% 24797)100 \% 24797$$

$$P = \mathbf{65}$$

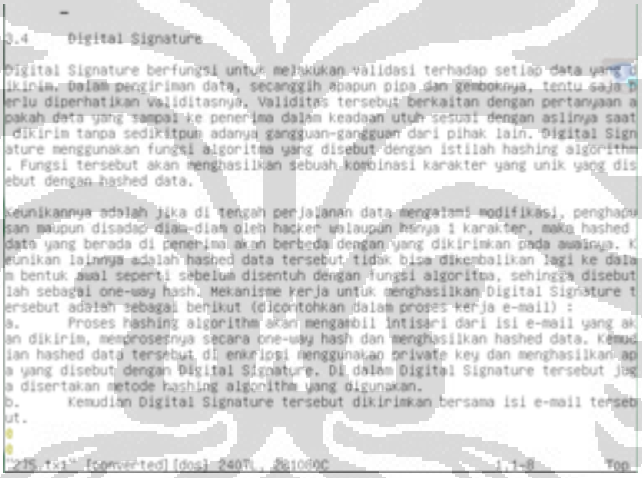
Pada program enkripsi RSA untuk perhitungan enkripsi dan dekripsi di atas akan menghasilkan file enkripsi dalam cryptRecv.txt yaitu 0x1289. Jika *attacker* ingin memecahkan isi file, maka MITM perlu membangkitkan kunci publik dan memiliki kunci privat yang sesuai untuk melakukan dekripsi file sehingga jika MITM tidak memiliki keduanya, maka file tidak akan dapat diketahui isinya.

File input yang akan dikirimkan oleh pengirim ke penerima, dimana pada uji coba alamat IP pengirim adalah alamat IP pengguna lokal sedangkan alamat IP penerima adalah alamat IP *instances*, akan dienkripsi sesuai dengan flowchart algoritma RSA. Karakter file akan dibaca setiap barisnya oleh program kemudian akan dikonversikan ke ASCII dan dienkripsi menggunakan rumus yang tertera pada Bab 2. Hasil file yang telah terenkripsi akan disimpan pada file cryptRecv.txt, dimana file ini dapat disimpan atau dihapus secara otomatis. File crypRecv akan berisi kode-kode enkripsi RSA dalam bentuk heksadesimal. Pada saat uji coba, file yang belum dienkripsi dengan file yang diterima setelah

didekripsi akan sama isinya, seperti yang ditunjukkan Gambar 4.5 dan Gambar 4.6.



Gambar 4.5 Isi File Pada Pengirim



Gambar 4.6 Isi File Pada Penerima

Pembangkitan kunci RSA dengan pasangan bilangan prima p dan q yang dipilih secara acak akan mempengaruhi hasil grafik pada Gambar 4.3 sehingga untuk format file .txt, .doc, dan .pdf pun akan membutuhkan waktu eksekusi yang berbeda-beda untuk setiap ukuran data yang sama. Pasangan bilangan prima p dan q akan mempengaruhi nilai n dan m yang kemudian akan mempengaruhi kunci publik untuk enkripsi dan kunci privat untuk dekripsi. Pengaruh pembangkitan besarnya pasangan bilangan prima secara acak yang akan mempengaruhi besarnya nilai n, m, e, dan d akan direpresentasikan pada Tabel 4.5. Pada Tabel 4.5 diambil sampel data .txt dan .doc dengan ukuran yang sama yaitu 285 kB untuk membuktikan bahwa waktu eksekusi program tidak hanya dipengaruhi oleh

ukuran data tetapi juga dipengaruhi oleh ukuran kunci RSA. Waktu eksekusi ini mayoritas dipengaruhi oleh kunci publik (yang terdiri dari nilai n dan e) untuk enkripsi dan kunci privat (yang terdiri dari nilai d dan n) untuk dekripsi. Untuk kasus kunci privat dengan nilai d yang besar, maka besarnya nilai d ini akan sangat mempengaruhi waktu eksekusi karena waktu dekripsi yang dibutuhkan akan sangat lama.

Pada Tabel 4.5 waktu dekripsi dihitung dengan menggunakan selisih waktu saat paket yang diterima telah selesai didekripsi dengan paket *acknowledgement* dari encrypted file yang telah berhasil diterima. Perhitungan waktu ini dihitung dengan menggunakan penangkapan dari Wireshark.

Tabel 4.5 Kunci RSA dengan Nilai d Terurut Beserta Waktu

p	q	n	m	e	d	t dekripsi (detik)	t eksekusi (detik)
7	67	469	396	181	361	2.569	87.663
19	59	1121	1044	79	859	5.504	120.264
167	13	2171	1992	223	1063	7.866	88.801
227	11	2497	2260	3	1507	9.999	77.596
43	197	8471	8232	157	1573	10.143	169.177
193	13	2509	2304	113	1937	12.360	90.717
43	241	10363	10080	191	2111	16.012	94.826
23	241	5543	5280	19	2779	18.800	81.471
127	31	3937	3780	23	3287	21.893	88.367
239	31	7409	7140	67	3943	24.780	99.819
151	181	27331	27000	227	4163	46.965	124.122
67	109	7303	7128	59	5195	57.430	145.535
47	127	5969	5796	97	5557	55.942	130.268
139	59	8201	8004	191	5783	61.571	156.879
193	113	21809	21504	157	8629	103.774	173.818
139	119	15707	15456	79	9391	152.291	241.206
139	127	17653	17388	239	11495	213.470	289.264
109	149	16241	15984	29	12677	132.258	211.560
227	163	37001	36612	131	22079	472.425	546.838
239	181	43259	42840	199	26479	469.895	545.927

Keterangan:

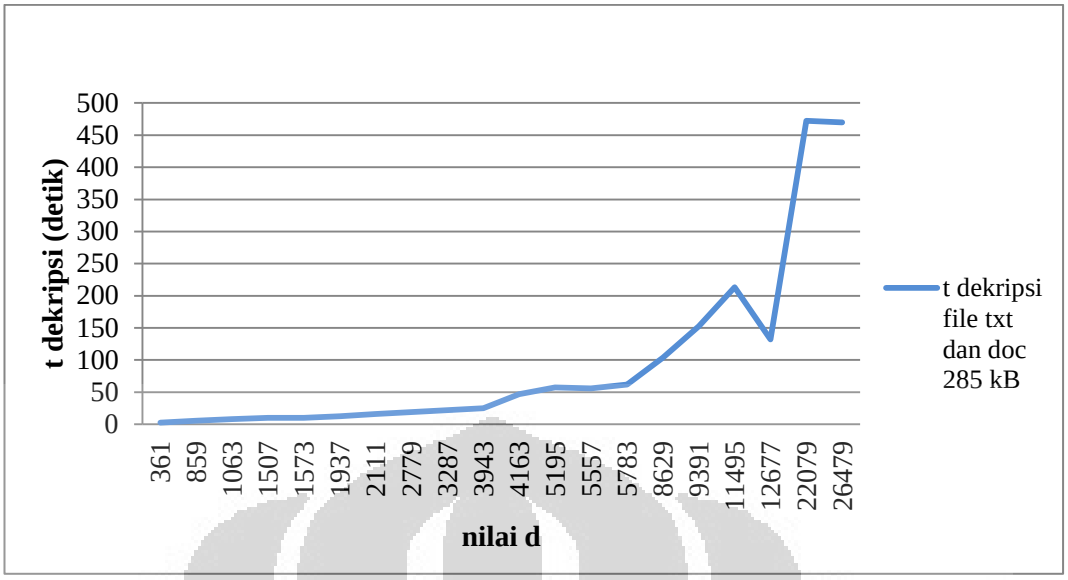
p dan q : bilangan prima

n : perkalian p dan q

m : perkalian $(p-1)$ dan $(q-1)$

e : nilai e dengan syarat $\gcd(e, m) = 1$

d : nilai d dengan rumus $d = (1 + m \cdot n)/e$ dan n = integer



Gambar 4.7 Grafik Waktu Dekripsi

Berdasarkan Tabel 4.5 dan Gambar 4.7 di atas terlihat bahwa waktu dekripsi dipengaruhi oleh nilai d, dimana nilai d yang semakin besar akan memberikan kecenderungan pada waktu dekripsi yang semakin lama. Hal ini sesuai dengan rumus untuk dekripsi (2.11) yang menunjukkan bahwa pangkat d yang semakin besar akan membutuhkan waktu yang lama untuk melakukan dekripsi pada cipher teks sehingga menyebabkan cipher teks lebih sulit untuk dipecahkan.

Pada beberapa kasus, waktu nilai d yang lebih besar memiliki waktu untuk dekripsi yang lebih kecil hal ini kemungkinan disebabkan oleh adanya faktor parameter QoS yang terjadi pada jaringan seperti adanya delay (waktu tunda kedatangan paket) karena adanya paket yang hilang atau adanya kemacetan sehingga terjadi retransmisi paket. Hal ini disebabkan karena paket TCP yang bersifat connection oriented sehingga protokol ini menjamin dan memastikan apakah paket sampai ke tujuan atau tidak. Kemungkinan adanya faktor QoS ini hanya mempengaruhi waktu eksekusi program RSA namun tidak akan mempengaruhi rusaknya file enkripsi.

Pada Tabel 4.6, waktu enkripsi dihitung dengan menggunakan hasil penangkapan Wireshark dimana waktu enkripsi ini merupakan selisih waktu saat paket encrypted file diterima oleh sisi penerima dengan waktu saat pertama kali inisialisasi koneksi antara sisi pengirim dan sisi penerima.

Tabel 4.6 Kunci RSA dengan Nilai e Terurut Beserta Waktu

p	q	n	m	e	d	t enkripsi (detik)	t eksekusi (detik)
227	11	2497	2260	3	1507	67.091	77.596
23	241	5543	5280	19	2779	61.975	81.471
127	31	3937	3780	23	3287	65.836	88.367
109	149	16241	15984	29	12677	78.454	211.560
67	109	7303	7128	59	5195	86.680	145.535
239	31	7409	7140	67	3943	71.733	99.819
139	119	15707	15456	79	9391	86.903	241.206
19	59	1121	1044	79	859	112.634	120.264
47	127	5969	5796	97	5557	71.804	130.268
193	13	2509	2304	113	1937	75.847	90.717
227	163	37001	36612	131	22079	71.741	546.838
43	197	8471	8232	157	1573	158.764	169.177
193	113	21809	21504	157	8629	66.016	173.818
7	67	469	396	181	361	80.401	87.663
43	241	10363	10080	191	2111	74.680	94.983
139	59	8201	8004	191	5783	90.419	156.879
239	181	43259	42840	199	26479	71.457	545.927
167	13	2171	1992	223	1063	75.523	88.801
151	181	27331	27000	227	4163	71.733	124.122
139	127	17653	17388	239	11495	71.886	289.264

Keterangan:

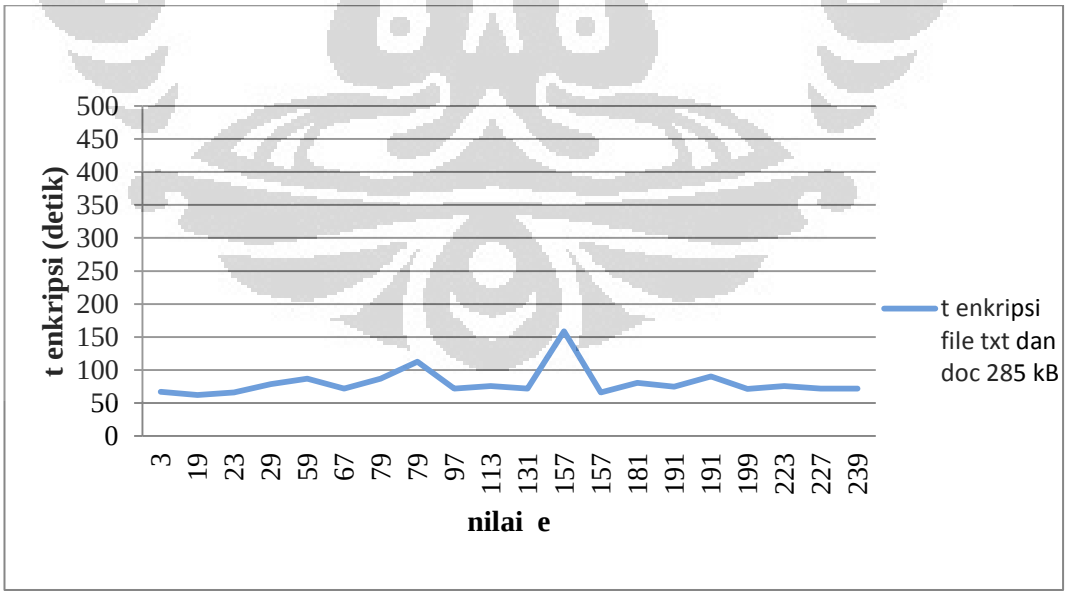
p dan q : bilangan prima

n : perkalian p dan q

m : perkalian $(p-1)$ dan $(q-1)$

e : nilai e dengan syarat $\gcd(e, m) = 1$

d : nilai d dengan rumus $d = (1 + m \cdot n)/e$ dan n = integer



Gambar 4.8 Grafik Waktu Enkripsi

Berdasarkan Tabel 4.6 dan Grafik 4.8 terlihat bahwa nilai e yang semakin besar tidak mempengaruhi peningkatan waktu enkripsi secara signifikan karena waktu enkripsi yang tidak meningkat seiring peningkatan nilai e. Nilai e yang semakin besar memberikan pengaruh terhadap kekuatan enkripsi data atau file. Namun, waktu enkripsi tetap memberikan kontribusi terhadap waktu eksekusi program yang apabila waktu enkripsi lama maka waktu eksekusi program pun akan bertambah. Jadi, waktu enkripsi ini dipengaruhi secara signifikan oleh ukuran data yang akan dienkripsi dimana jika ukuran data yang akan dienkripsi semakin tinggi maka akan meningkatkan waktu enkripsi yang akan memberikan efek pada waktu eksekusi program yang meningkat pula.

4.1.2.2 Analisa Keamanan dari Implementasi Kriptografi RSA

Program enkripsi RSA sederhana ini merupakan plain RSA yang tidak memiliki sistem *padding* sehingga keamanannya pun masih kurang karena pemecahan faktorisasi yang dapat dilakukan dalam waktu singkat. *Padding* merupakan penambahan bit-bit dummies untuk menggenapi menjadi panjang blok yang sesuai dengan standar yang ada, biasanya *padding* ditambahkan pada blok terakhir plain teks. Sistem *padding* untuk kriptografi RSA yang baik didefinisikan oleh PKCS1 (Public Key Cryptography Standard) untuk standar kriptografi RSA yang aman.

Menurut rekomendasi NIST saat ini, kunci RSA yang merupakan algoritma asimetris yang aman setidaknya memiliki panjang 2048 bit sedangkan pada program enkripsi RSA ini nilai n tidak mencapai 256 bit sehingga kemanannya pun masih kurang. Berdasarkan beberapa kali percobaan yang dilakukan, nilai n maksimal yang dapat dihasilkan oleh program hanya 16 bit sehingga program plain RSA ini belum memenehui rekomendasi NIST. Hasil bit *n* pada percobaan maksimal hanya 16 bit karena tidak menggunakan sistem *padding* dimana nilai n pun akan berbeda-beda. Sedangkan RSA dengan sistem *padding* akan memiliki bit n yang sama karena adanya penambahan bit-bit dummies. Selain itu, bit *n* yang besar hingga 256 bit bahkan 2048 bit membutuhkan keahlian khusus dalam membuat sisem dengan bit *n* yang sesuai dengan standar dan rekomendasi internasional karena bit *n* yang besar dapat

membuat error atau kerusakan pada enkripsi dan dekripsi file berbasis teks jika keahlian khusus tersebut tidak dimiliki.

Tabel 4.7 Rekomendasi NIST

Date	Minimum of Strength	Symmetric Algorithms	Asymmetric	Discrete Logarithm Key	Discrete Logarithm Group	Elliptique Curve	Hash (A)	Hash (B)
2010	80	2TDEA*	1024	160	1024	160	SHA-1** SHA-224 SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
2011 - 2030	112	3TDEA	2048	224	2048	224	SHA-224 SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
> 2030	128	AES-128	3072	256	3072	256	SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
>> 2030	192	AES-192	7680	384	7680	384	SHA-384 SHA-512	SHA-224 SHA-256 SHA-384 SHA-512
>>> 2030	256	AES-256	15360	512	15360	512	SHA-512	SHA-256 SHA-384 SHA-512

Untuk melakukan cracking pada plain RSA dapat dilakukan salah satu metode cracking RSA yaitu *choosen-ciphertext attack*. Seorang attacker ataupun cryptanalist dapat menggunkan metode ini untuk memasuki sistem kemudian memilih ciphertext dan melakukan dekripsi menggunakan kunci privat yang sebelumnya tidak diketahui oleh penyerang. Sistem plain RSA pada skripsi ini memiliki peluang terkena serangan *choosen-ciphertext* sehingga penyerang dapat mengetahui plainteks dari cipherteks yang dipilih. Hal ini disebabkan nilai e yang cenderung masih rendah sehingga kekuatan enkripsi pun masih kurang.

Seperti yang telah diungkapkan bahwa nilai d yang sangat besar akan membutuhkan waktu untuk dekripsi yang cukup lama sehingga hal ini berpengaruh pada keamanan file dimana pemecahan file yang telah enkripsi akan lebih sulit untuk dilakukan. Menurut Wiener pada tahun 1990, jika p di antara q dan $2q$ (yang sangat mirip) dan asumsi nilai d lebih kecil daripada $n^{1/4}/3$, maka d akan dapat dihitung secara effisien dari n dan e selain itu kunci e = 2 tidak aman untuk digunakan. Pada program enkripsi terdapat nilai e = 3 yang cenderung tidak aman walaupun nilai e yang sangat kecil ini jarang muncul dalam beberapa kali percobaan. Selain nilai n yang berpengaruh untuk enkripsi, nilai e juga

berpengaruh (sesuai dengan rumus enkripsi). Semakin besar nilai e, maka enkripsi terhadap file juga akan semakin kuat. Namun, pada program RSA ini nilai e tidak cukup besar sehingga enkripsinya pun kurang kuat. Selain itu, pada sample percobaan misalnya p = 239,dan q = 181, maka nilai p akan berada di antara q dan 2q sehingga kunci d dapat dihitung dari n dan e menurut teorema Wiener. Hal-hal mengenai keamanan ini yang akan menjadi evaluasi untuk mengembangkan program enkripsi RSA dengan panjang kunci 2048 bit dan menggunakan *padding*. Parameter keamanan ini yang menjadi pertimbangan untuk menerapkan program RSA sederhana ini pada jaringan lokal, walaupun sebaiknya pada jaringan lokal juga dibutuhkan enkripsi RSA dengan panjang bit minimal 2048 dengan sistem *padding*.

BAB 5
KESIMPULAN

1. Sistem Eucalyptus merupakan salah satu teknologi virtualisasi *cloud IaaS* yang dapat mendukung sumberdaya komputasi fisik dengan *instances* (VM) yang tercipta. Eucalyptus memiliki komponen penting yang dapat mengatur dan menciptakan VM tersebut yaitu CLC (Cloud Controller) dan NC (Node Controller) yang memiliki fungsinya masing-masing.
2. Sistem pengiriman data dengan enkripsi RSA diterapkan pada *instances* Eucalyptus dengan pengguna lokal. Pengiriman file dengan format .txt, .pdf, dan .doc dilakukan antara user lokal dan *instances* yang diatur oleh admin.
3. Pengiriman data dengan enkripsi RSA untuk masing-masing format file membutuhkan waktu eksekusi dimana peningkatan waktu eksekusi ini dipengaruhi oleh penambahan ukuran file. Peningkatan waktu eksekusi untuk format file .txt sebesar 31,44%, format file .doc sebesar 24,83%, dan format file .pdf sebesar 24,85%.
4. Peningkatan waktu eksekusi juga dipengaruhi oleh nilai dari kunci yang dibangkitkan oleh program RSA, dimana mayoritas waktu eksekusi dipengaruhi oleh kunci publik (yang terdapat nilai e) untuk enkripsi dan kunci privat (yang terdapat nilai d) untuk dekripsi. Pada kasus nilai d yang besar akan sangat mempengaruhi waktu eksekusi karena dibutuhkan waktu dekripsi yang lama. Sedangkan, nilai e yang besar untuk kunci publik RSA tidak terlalu signifikan mempengaruhi waktu enkripsi menjadi lebih lama namun tetap berkontribusi terhadap waktu eksekusi RSA.
5. Sistem pengiriman data dengan enkripsi RSA pada penelitian menggunakan plain RSA tanpa sistem *padding* sehingga kurang aman dan tidak memenuhi standar PKCS1 sehingga plain RSA memiliki peluang terkena *choosen-ciphertext attack*.
6. Program enkripsi RSA pada penelitian menghasilkan nilai n sekitar 16 bit sehingga tidak memenuhi rekomendasi NIST dimana untuk RSA yang aman dibutuhkan nilai n sebesar 2048 bit.

DAFTAR ACUAN

[1] Sabahi, Farhad. (2011). *Virtualization-Level Security in Cloud Computing.* Jurnal IEEE, 250-254.

[2] Sanka, Sunil, Chittaranjan Hotta, Muttukhrisnan Rajarajan. (2010). *Secure Data Access in Cloud Computing.* Jurnal IEEE.

[3] *Cloud computing*. Maret 5, 2012. http://en.wikipedia.org/Cloud_computing/

[4] S, Ramgovind, Eloff MM, Smith E. (2010). *The Management of Security in Cloud Computing.*Jurnal IEEE.

[5] Mather, Tim, Subra Kumaraswamy, Shahed Latif. *Cloud Security and Privacy : An Enterprise Perspective on Risk and Compliance*. United States of America : O'reilly Media, 2009.

[6] Krutz, Ronald L., Russel Dean Vines. *Cloud Security : A Compherensive Guide to Secure Cloud Computing.* Indianapolis : Wiley Publishing, 2010.

[7] *Kriptografi Simetris dan Hybrid*. Maret 13, 2012. http://nazernasrisamaaja.wordpress.com/2011/03/30/kriptografi-simetrisasimetris-dan-hybrid/

[8] Xue, Yuan. *RSA Algorithm*. Maret 6, 2012.

[9] *Eucalyptus (computing)* . Maret 2, 2012. http://en.wikipedia.org/wiki/Eucalyptus_(computing)

[10] D, Johnson, et al., ed. *Eucalyptus Beginner's Guide – UEC Edition*. May 2010

[11] Hendryli, Janson, et al., *Kriptografi RSA*. Maret 6, 2012.

[12] Sulistyanto, Hernawan. (2004). *Autentikasi dalam Basis Data Jaringan Menggunakan Kriptosistem Kunci Publik RSA*. Jurnal Teknik Elektro dan Komputer Emitor Vol 4.

[13] *UEC/Package Install*. Maret 2, 2012. http://help.ubuntu.com/ community/UEC/

[14] S. Tanenbaum, Andrew. *Computer Networks, Fourth Edition*. Prentice Hall, 2003.

LAMPIRAN

1. Tabel ASCII

ascii table

Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(nul)	0	0000	0x00	(sp)	32	0040	0x20	@	64	0100	0x40	`	96	0140	0x60
(soh)	1	0001	0x01	!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
(stx)	2	0002	0x02	"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
(etx)	3	0003	0x03	#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
(eot)	4	0004	0x04	$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
(enq)	5	0005	0x05	%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
(ack)	6	0006	0x06	&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
(bel)	7	0007	0x07	'	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(bs)	8	0010	0x08	(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
(ht)	9	0011	0x09	)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
(nl)	10	0012	0x0a	*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
(vt)	11	0013	0x0b	+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
(np)	12	0014	0x0c	,	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
(cr)	13	0015	0x0d	-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
(so)	14	0016	0x0e	.	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
(si)	15	0017	0x0f	/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
(dle)	16	0020	0x10	0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
(dc1)	17	0021	0x11	1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
(dc2)	18	0022	0x12	2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
(dc3)	19	0023	0x13	3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
(dc4)	20	0024	0x14	4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
(nak)	21	0025	0x15	5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
(syn)	22	0026	0x16	6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
(etb)	23	0027	0x17	7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
(can)	24	0030	0x18	8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
(em)	25	0031	0x19	9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
(sub)	26	0032	0x1a	:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
(esc)	27	0033	0x1b	;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
(fs)	28	0034	0x1c	<	60	0074	0x3c	\	92	0134	0x5c	|	124	0174	0x7c
(gs)	29	0035	0x1d	=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
(rs)	30	0036	0x1e	>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
(us)	31	0037	0x1f	?	63	0077	0x3f	_	95	0137	0x5f	(del)	127	0177	0x7f

(lanjutan)

2. Data Percobaan .txt

Percobaan	txt 285 kB	txt 385 kB	txt 485 kB	txt 585	txt 685
1	117.275 detik	110.960 detik	479.317 detik	184.225 detik	197.822 detik
2	73.270 detik	248.504 detik	201.221 detik	216.940 detik	178.509 detik
3	73.274 detik	206.480 detik	161.256 detik	163.581 detik	575.822 detik
4	80.552 detik	173.608 detik	121.336 detik	390.839 detik	537.729 detik
5	77.899 detik	153.533 detik	144.284 detik	176.620 detik	975.323 detik
6	93.817 detik	148.720 detik	258.071 detik	225.216 detik	472.296 detik
7	74.478 detik	126.930 detik	210.984 detik	196.493 detik	209.372 detik
8	72.314 detik	153.533 detik	161.482 detik	167.582 detik	878.279 detik
9	90.826 detik	273.120 detik	209.452 detik	201.717 detik	257.970 detik
10	87.667 detik	185.668 detik	172.085 detik	234.379 detik	539.414 detik

3. Data Percobaan .doc

Percobaan	doc 285	doc 385	doc 485	doc 585	doc 685
1	79.314 detik	244.188 detik	328.419 detik	175.952 detik	361.534 detik
2	113.577 detik	280.062 detik	121.605 detik	164.742 detik	308.338 detik
3	72.290 detik	274.402 detik	293.206 detik	414.967 detik	176.692 detik
4	67.882 detik	192.834 detik	123.492 detik	341.906 detik	186.570 detik
5	73.538 detik	113.133 detik	247.841 detik	218.611 detik	213.032 detik
6	86.576 detik	121.769 detik	159.921 detik	322.028 detik	208.782 detik
7	93.301 detik	142.599 detik	455.329 detik	159.838 detik	228.612 detik
8	69.696 detik	103.722 detik	154.454 detik	309.419 detik	546.116 detik
9	131.552 detik	82.889 detik	133.545 detik	415.986 detik	346.819 detik
10	77.595 detik	111.371 detik	287.263 detik	181.473 detik	387.839 detik

4. Data Percobaan .pdf

Percobaan	pdf 285	pdf 385	pdf 485	pdf 585	pdf 685
1	77.274 detik	467.730 detik	562.056 detik	150.772 detik	205.601 detik
2	81.025 detik	125.475 detik	478.576 detik	284.510 detik	150.404 detik
3	113.91 detik	104.741 detik	120.612 detik	205.337 detik	540.496 detik
4	195.908 detik	152.377 detik	272.308 detik	612.140 detik	305.408 detik
5	95.510 detik	271.108 detik	115.526 detik	166.477 detik	213.058 detik
6	151.873 detik	193.800 detik	113.281 detik	487.851 detik	531.800 detik
7	206.711 detik	161.806 detik	134.488 detik	256.173 detik	292.962 detik
8	83.579 detik	145.749 detik	121.452 detik	237.834 detik	1141.932 detik
9	82.952 detik	159.492 detik	138.351 detik	250.208 detik	254.510 detik
10	93.942 detik	145.600 detik	166.509 detik	347.498 detik	187.823 detik

(lanjutan)

5. Sample Data File .txt

No	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.19.1.2	192.168.2.6	TCP	74	56260 > nms-dpnss [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=862511 TSecr=0 WS=2
2	0.000371	192.168.2.6	172.19.1.2	TCP	74	nms-dpnss > 56260 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=629699 TSecr=862511 WS=8
3	0.000851	172.19.1.2	192.168.2.6	TCP	66	56260 > nms-dpnss [ACK] Seq=1 Ack=1 Win=14600 Len=0 TSval=862512 TSecr=629699
4	0.002602	192.168.2.6	172.19.1.2	TCP	69	nms-dpnss > 56260 [PSH, ACK] Seq=1 Ack=1 Win=14480 Len=3 TSval=629700 TSecr=862512
5	0.003005	172.19.1.2	192.168.2.6	TCP	66	56260 > nms-dpnss [ACK] Seq=1 Ack=4 Win=14600 Len=0 TSval=862512 TSecr=629700
6	0.006842	172.19.1.2	192.168.2.6	TCP	68	56260 > nms-dpnss [PSH, ACK] Seq=1 Ack=4 Win=14600 Len=2 TSval=862513 TSecr=629700
7	0.007360	192.168.2.6	172.19.1.2	TCP	66	nms-dpnss > 56260 [ACK] Seq=4 Ack=3 Win=14480 Len=0 TSval=629701 TSecr=862513
8	0.007632	192.168.2.6	172.19.1.2	TCP	69	nms-dpnss > 56260 [PSH, ACK] Seq=4 Ack=3 Win=14480 Len=3 TSval=629701 TSecr=862513
9	0.050415	172.19.1.2	192.168.2.6	TCP	66	56260 > nms-dpnss [ACK] Seq=3 Ack=7 Win=14600 Len=0 TSval=862524 TSecr=629701
10	4.682.548	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=3 Ack=7 Win=14600 Len=4 TSval=863682 TSecr=629701
11	4.682.566	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=7 Ack=7 Win=14480 Len=2 TSval=630870 TSecr=863682
12	4.683.183	172.19.1.2	192.168.2.6	TCP	66	56260 > nms-dpnss [ACK] Seq=7 Ack=9 Win=14600 Len=0 TSval=863682 TSecr=630870
13	4.683.193	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=7 Ack=9 Win=14600 Len=4 TSval=863682 TSecr=630870
14	4.683.636	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=9 Ack=11 Win=14480 Len=2 TSval=630870 TSecr=863682
15	4.683.973	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=11 Ack=11 Win=14600 Len=4 TSval=863682 TSecr=630870
16	4.684.340	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=11 Ack=15 Win=14480 Len=2 TSval=630870 TSecr=863682
17	4.684.800	172.19.1.2	192.168.2.6	TCP	71	56260 > nms-dpnss [PSH, ACK] Seq=15 Ack=13 Win=14600 Len=5 TSval=863682 TSecr=630870
18	4.685.498	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=13 Ack=20 Win=14480 Len=2 TSval=630870 TSecr=863682
19	4.685.683	172.19.1.2	192.168.2.6	TCP	71	56260 > nms-dpnss [PSH, ACK] Seq=20 Ack=15 Win=14600 Len=5 TSval=863683 TSecr=630870
20	4.686.313	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=15 Ack=25 Win=14480 Len=2 TSval=630871 TSecr=863683
21	4.686.495	172.19.1.2	192.168.2.6	TCP	69	56260 > nms-dpnss [PSH, ACK] Seq=25 Ack=17 Win=14600 Len=3 TSval=863683 TSecr=630871
22	4.687.002	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=17 Ack=28 Win=14480 Len=2 TSval=630871 TSecr=863683
23	4.687.226	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=28 Ack=19 Win=14600 Len=4 TSval=863683 TSecr=630871
24	4.687.523	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=19 Ack=32 Win=14480 Len=2 TSval=630871 TSecr=863683
25	4.687.940	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=32 Ack=21 Win=14600 Len=4 TSval=863683 TSecr=630871
26	4.688.262	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=21 Ack=36 Win=14480 Len=2 TSval=630871 TSecr=863683
27	4.688.580	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=36 Ack=23 Win=14600 Len=4 TSval=863683 TSecr=630871
28	4.688.945	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=23 Ack=40 Win=14480 Len=2 TSval=630871 TSecr=863683
29	4.689.151	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=40 Ack=25 Win=14600 Len=4 TSval=863683 TSecr=630871
30	4.689.547	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=25 Ack=44 Win=14480 Len=2 TSval=630872 TSecr=863683
31	4.689.922	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=44 Ack=27 Win=14600 Len=4 TSval=863684 TSecr=630872
32	4.690.235	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=27 Ack=48 Win=14480 Len=2 TSval=630872 TSecr=863684
33	4.690.639	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=48 Ack=29 Win=14600 Len=4 TSval=863684 TSecr=630872

(lanjutan)

No	Time	Source	Destination	Protocol	Length	Info
34	4.690.968	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=29 Ack=52 Win=14480 Len=2 TSval=630872 TSecr=863684
35	4.691.494	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=52 Ack=31 Win=14600 Len=4 TSval=863684 TSecr=630872
36	4.691.926	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=31 Ack=56 Win=14480 Len=2 TSval=630872 TSecr=863684
37	4.692.167	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=56 Ack=33 Win=14600 Len=4 TSval=863684 TSecr=630872
38	4.692.486	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=33 Ack=60 Win=14480 Len=2 TSval=630872 TSecr=863684
39	4.692.836	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=60 Ack=35 Win=14600 Len=4 TSval=863684 TSecr=630872
40	4.693.248	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=35 Ack=64 Win=14480 Len=2 TSval=630872 TSecr=863684
41	4.696.102	172.19.1.2	192.168.2.6	TCP	71	56260 > nms-dpnss [PSH, ACK] Seq=64 Ack=37 Win=14600 Len=5 TSval=863685 TSecr=630872
42	4.696.416	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=37 Ack=69 Win=14480 Len=2 TSval=630873 TSecr=863685
43	4.696.778	172.19.1.2	192.168.2.6	TCP	71	56260 > nms-dpnss [PSH, ACK] Seq=69 Ack=39 Win=14600 Len=5 TSval=863685 TSecr=630873
44	4.697.065	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=39 Ack=74 Win=14480 Len=2 TSval=630873 TSecr=863685
45	4.697.712	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=74 Ack=41 Win=14600 Len=4 TSval=863685 TSecr=630873
46	4.698.099	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=41 Ack=78 Win=14480 Len=2 TSval=630873 TSecr=863685
47	4.698.584	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=78 Ack=43 Win=14600 Len=4 TSval=863686 TSecr=630873
48	4.699.060	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=43 Ack=82 Win=14480 Len=2 TSval=630874 TSecr=863686
49	4.699.398	172.19.1.2	192.168.2.6	TCP	71	56260 > nms-dpnss [PSH, ACK] Seq=82 Ack=45 Win=14600 Len=5 TSval=863686 TSecr=630874
50	4.699.698	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=45 Ack=87 Win=14480 Len=2 TSval=630874 TSecr=863686
51	4.700.140	172.19.1.2	192.168.2.6	TCP	71	56260 > nms-dpnss [PSH, ACK] Seq=87 Ack=47 Win=14600 Len=5 TSval=863686 TSecr=630874
52	4.700.703	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=47 Ack=92 Win=14480 Len=2 TSval=630874 TSecr=863686
53	4.701.160	172.19.1.2	192.168.2.6	TCP	69	56260 > nms-dpnss [PSH, ACK] Seq=92 Ack=49 Win=14600 Len=3 TSval=863686 TSecr=630874
54	4.701.624	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=49 Ack=95 Win=14480 Len=2 TSval=630874 TSecr=863686
55	4.702.135	172.19.1.2	192.168.2.6	TCP	71	56260 > nms-dpnss [PSH, ACK] Seq=95 Ack=51 Win=14600 Len=5 TSval=863687 TSecr=630874
56	4.702.604	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=51 Ack=100 Win=14480 Len=2 TSval=630875 TSecr=863687
57	4.703.005	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=100 Ack=53 Win=14600 Len=4 TSval=863687 TSecr=630875
58	4.703.454	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=53 Ack=104 Win=14480 Len=2 TSval=630875 TSecr=863687
59	4.703.939	172.19.1.2	192.168.2.6	TCP	71	56260 > nms-dpnss [PSH, ACK] Seq=104 Ack=55 Win=14600 Len=5 TSval=863687 TSecr=630875
60	4.704.405	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=55 Ack=109 Win=14480 Len=2 TSval=630875 TSecr=863687
61	4.704.831	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=109 Ack=57 Win=14600 Len=4 TSval=863687 TSecr=630875
62	4.705.268	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=57 Ack=113 Win=14480 Len=2 TSval=630875 TSecr=863687
63	4.705.827	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=113 Ack=59 Win=14600 Len=4 TSval=863687 TSecr=630875
64	4.706.125	192.168.2.6	172.19.1.2	TCP	68	nms-dpnss > 56260 [PSH, ACK] Seq=59 Ack=117 Win=14480 Len=2 TSval=630875 TSecr=863687
65	4.706.569	172.19.1.2	192.168.2.6	TCP	70	56260 > nms-dpnss [PSH, ACK] Seq=117 Ack=61 Win=14600 Len=4 TSval=863688 TSecr=630875