Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR – KI141502

IMPLEMENTASI METODE ENKRIPSI MENGGUNAKAN ALGORITMA RIVEST CIPHER (RC4) PADA APLIKASI SURAT ELEKTRONIK *E-SURAT* BERBASIS WEB

NINDYASARI DEWI UTARI
NRP 5113100039

Dosen Pembimbing I
Henning Ciptaningtyas, S.Kom., M.Kom.

Dosen Pembimbing II
Ir. Muchammad Husni, M.Kom.

Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

TUGAS AKHIR – KI141502

IMPLEMENTASI METODE ENKRIPSI MENGGUNAKAN ALGORITMA RIVEST CIPHER (RC4) PADA APLIKASI SURAT ELEKTRONIK *E-SURAT* BERBASIS WEB

NINDYASARI DEWI UTARI
NRP 5113100039

Dosen Pembimbing I
Henning Titi Ciptaningtyas, S.Kom., M.Kom.

Dosen Pembimbing II
Ir. Muchammad Husni, M.Kom.

Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – KI141502

IMPLEMENTATION OF ENCRYPTION METHOD WITH RIVEST CIPHER (RC4) ALGORITHM IN EMAIL *E-SURAT* WEB BASE APPLICATION

NINDYASARI DEWI UTARI
NRP 5113100039

First Advisor
Henning Titi Ciptaningtyas, S.Kom., M.Kom.

Second Advisor
Ir. Muchammad Husni, M.Kom.

Department of Informatics
Faculty of Information Technology
Sepuluh Nopember Institute of Technology
Surabaya 2017

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

IMPLEMENTASI METODE ENKRIPSI MENGGUNAKAN ALGORITMA RIVEST CIPHER (RC4) PADA APLIKASI SURAT ELEKTRONIK *E-SURAT* BERBASIS WEB

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:
NINDYASARI DEWI UTARI
NRP: 5113100039

Disetujui oleh Pembimbing Tugas Akhir:

1. Henning Titi Ciptaningtyas, S.Kom., M.Kom. .....
   (NIP. *194908231976032001*) (Pembimbing 1)

2. Ir. Muchammad Husni, M.Kom. .....
   (NIP. *051100121*) (Pembimbing 2)

SURABAYA
JUNI, 2017

(Halaman ini sengaja dikosongkan)

IMPLEMENTASI METODE ENKRIPSI MENGGUNAKAN ALGORITMA RIVEST CIPHER (RC4) PADA APLIKASI SURAT ELEKTRONIK *E-SURAT* BERBASIS WEB

Nama Mahasiswa	**: NINDYASARI DEWI UTARI**
NRP	**: 5113100039**
Jurusan	**: Teknik Informatika FTIF-ITS**
Dosen Pembimbing 1	**: Henning Titi Ciptaningtyas, S.Kom., M.Kom.**
Dosen Pembimbing 2	**: Ir. Muchammad Husni, M.Kom.**

Abstrak

Enkripsi merupakan sebuah proses penyandian yang melakukan perubahan sebuah kode atau pesan dari yang dapat dimengerti (plaintext) menjadi sebuah kode yang tidak dapat dimengerti (ciphertext). Sedangkan proses sebaliknya disebut dekripsi. Proses tersebut memerlukan suatu mekanisme dan kunci tertentu untuk menyembunyikan maknanya dan mencegah penerima yang tidak sah dari mengambil data asli.

Pengamanan data pada sebuah aplikasi sangat penting, khususnya pada aplikasi yang membutuhkan koneksi internet. Karena lewat internet, persebaran informasi sangat mudah. Dengan adanya kemudahan seperti ini, maka dibutuhkan pengamanan terhadap sebuah data. Maka, untuk mengamankan sebuah data, dibutuhkan metode pengamanan berupa enkripsi yang tepat agar tidak mudah diserang. Oleh karena itu pada Tugas Akhir ini akan mengimplementasikan metode Rivest Cipher (RC4) untuk mengamankan sebuah data pada aplikasi berbasis web. Pemakaian metode enkripsi menggunakan algorittma Rivest Cipher (RC4) ini memiliki karakteristik algoritma yang simetrik karena menggunakan kunci yang sama untuk mengenkripsi suatu pesan, data, ataupun informasi. Secara umum, algoritma Rivest Cipher (RC4) terbagi menjadi dua, inisialisasi state-array dan penghasilan kunci enkripsi serta pengenkripsiannya.

Dari hasil uji coba yang telah dilakukan, dihasilkan bahwa algoritma Rivest Cipher (RC4) memiliki kecepatan enkripsi yaitu sekitar 8 ms dan kecepatan dekripsi yaitu sekitar 2 ms. Untuk hasil coba performa yang telah dilakukan, dihasilkan bahwa komputer web server mampu menangani 1024 akses pengguna dengan request per secondnya mencapai 34 request.

Kata kunci: Keamanan Data, Enkripsi, Algoritma Rivest Cipher (RC4)

IMPLEMENTASI METODE ENKRIPSI MENGGUNAKAN ALGORITMA RIVEST CIPHER (RC4) PADA APLIKASI SURAT ELEKTRONIK *E-SURAT* BERBASIS WEB

Student's Name	**: NINDYASARI DEWI UTARI**
Student's ID	**: 5113100039**
Department	**: Teknik Informatika FTIF-ITS**
First Advisor	**: Henning Titi Ciptaningtyas, S.Kom., M.Kom.**
Second Advisor	**: Ir. Muchammad Husni, M.Kom.**

Abstract

Encryption is an encoding process that changes a code or message from a plaintext into an unintelligible code (ciphertext). While the reverse process is called decryption. The process requires a certain mechanism and key to capitalize its meaning and not the unauthorized recipient from retrieving the original data.

Data security on an application is very important, especially in applications that require internet connection. Because over the internet, the distribution of information is very easy. With the ease like this, then the security needed to a data. So, to secure a data, required security methods in the form of proper encryption so as not easily attacked. Therefore, in this undergraduate thesis implements Rivest Cipher (RC4) method to secure data on web based application. The use of encryption method using Rivest Cipher (RC4) algorithm has symmetric symmetric characteristics because it uses the same key to encrypt a message, data, or information. In general, the Rivest Cipher (RC4) algorithm is divided into two, state-array initialization and encryption key revenue and its encryption.

From the results of trials that have been done, the Rivest Cipher (RC4) algorithm has an encryption speed of about 8 ms and decryption speed is about 2 ms. For the results of the performance try that has been done, resulting in a web server computer capable

of installing 1024 access users with demand per second is 34 requests.

Keywords : Data Security, Encryption, Rivest Cipher (RC4) Algorithm

KATA PENGANTAR

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT yang telah melimpahkan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul:

"**Implementasi Metode Enkripsi Menggunakan Algoritma *Rivest Cipher* (RC4) pada Aplikasi Surat Elektronik *E-Surat* Berbasis Web**"

Terselesaikannya Tugas Akhir ini tidak terlepas dari bantuan dan dukungan banyak pihak, Oleh karena itu melalui lembar ini penulis ingin mengucapkan terima kasih dan penghormatan kepada:

1. Allah SWT serta junjungan Nabi Muhammad SAW, karena limpahan rahmat dan karunia-Nya penulis dapat menyelesaikan Tugas Akhir dan juga perkuliahan di Teknik Informatika ITS.
2. Ayah dan Ibu penulis, Nindyo Pramono dan Ewi Setyaki yang tiada hentinya memberikan dukungan doa, moral, dan material kepada penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini.
3. Adik penulis, Widya Yudha Patria serta keluarga tercinta yang telah memberikan dukungan dan semangatnya kepada penulis.
4. Ibu Henning Titi Ciptaningtyas, S.Kom., M.Kom. dan Bapak Ir. Muchammad Husni, M.Kom. selaku pembimbing I dan II yang telah membimbing dan memberikan motivasi, nasihat dan bimbingan dalam menyelesaikan Tugas Akhir ini.
5. Bapak Darlis Herumurti, S.Kom., M.Kom. selaku kepala jurusan Teknin Informatika ITS dan segenap dosen dan

karyawan Teknik Informatika ITS yang telah memberikan ilmu dan pengalaman kepada penulis selama menjalani masa studi di Teknik Informatika ITS.

6. Rekan diskusi Yusuf Nugroho yang selalu membantu ketika penulis menemukan kesusahan dan kesalahan dalam setiap pengerjaan Tugas Akhir.
7. Rekan penulis dalam pengerjaan Tugas Akhir yaitu Kinasih Nur Azizah yang selalu mengingatkan dan saling menguatkan.
8. Sahabat penulis, Dhita, Harry, Rifqi, Lino, Budi yang selalu ada dan saling memberi semangat selama pengerjaan Tugas Akhir.
9. Teman-teman Laboratorium AJK, Daniel, Fathoni, Mas Thiar, Syukron, Wicak, Uul, Asbun, Risma, Awan, Ambon, Vivi, Oing, Didin, Fuad, Bebet, Fatih yang senantiasa menghibur dan mendukung penulis dalam mengerjakan tugas akhir ini serta menemani penulis di laboratorium.
10. Serta semua pihak yang yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa laporan Tugas Akhir ini masih memiliki banyak kekurangan. Oleh karena itu dengan segala kerendahan hati penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan penulis kedapannya. Selain itu, penulis berharap laporan Tugas Akhir ini dapat berguna bagi pembaca secara umum.

Surabaya, Juni 2017

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pada era teknologi informasi yang semakin berkembang, pegiriman data dan informasi merupakan suatu hal yang penting. Apalagi dengan adanya fasilitas internet yang semakin memudahkan dalam bertukar informasi. Namun dengan adanya kemudahan seperti ini, keamanan data pun akan menjadi hal yang tidak kalah pentingnya. Karena pada saat pengiriman data tersebut, tidak jarang terdapat informasi yang bersifat rahasia.

Oleh karena itu, dibutuhkan suatu cara agar data atau informasi tersebut aman. Salah satu cara yang digunakan adalah penyandian data atau enkripsi data. Enkripsi merupakan sebuah proses penyandian yang melakukan perubahan sebuah kode atau pesan dari yang dapat dimengerti (*plainteks*) menjadi sebuah kode yang tidak dapat dimengerti (*cipherteks*). Sedangkan proses sebaliknya disebut dekripsi. Proses tersebut memerlukan suatu mekanisme dan kunci tertentu untuk menyembunyikan maknanya dan mencegah penerima yang tidak sah dari mengambil data asli [1].

Proses enkripsi memiliki berbagai macam metode. Pada tugas akhir ini diterapkan metode enkripsi algoritma *Rivest Cipher* (RC4) untuk melakukan pengamanan data yang mungkin terjadi pada proses aplikasi surat elektronik berbasis web aplikasi *E-Surat* Institut Teknologi Sepuluh Nopember Surabaya (ITS). Selama ini pada aplikasi *E-Surat* Institut Teknologi Sepuluh Nopember Surabaya (ITS) masih belum memiliki keamanan pada sistemnya. Sehingga aplikasi ini kemungkinan dapat diretas oleh orang yang tidak bertanggungjawab.

Pemakaian metode enkripsi menggunakan algorittma *Rivest Cipher* (RC4) ini memiliki karakteristik algoritma yang

simetrik karena menggunakan kunci yang sama untuk mengenkripsi suatu pesan, data, ataupun informasi. Kunci enkripsi didapat dari sebuah 256 bit *state-array* yang diinisialisasi dengan sebuah *key* tersendiri dengan panjang 1-256 bit. Setelah itu, *state-array* tersebut akan diacak kembali dan diproses untuk menghasilkan sebuah kunci enkripsi yang akan di-XOR-kan dengan *plainteks* atau *cipherteks*. Secara umum, algoritma *Rivest Cipher* (RC4) terbagi menjadi dua, inisialisasi *state-array* dan penghasilan kunci enkripsi serta pengenkripsiannya. Dari karakteristik yang demikian, membuat algoritma *Rivest Cipher* (RC4) menjadi metode yang dapat diimplementasikan pada aplikasi surat elektronik *E-Surat* Institut Teknologi Sepuluh Nopember Surabaya (ITS).

1.2 Rumusan Masalah

Tugas Akhir ini mengangkat beberapa rumusan masalah sebagai berikut:

1. Bagaimana cara menerapkan metode enkripsi menggunakan algoritma *Rivest Cipher* (RC4) pada aplikasi surat elektronik?
2. Bagaimana hasil dari penerapan metode enkripsi yang menggunakan algoritma *Rivest Cipher* (RC4) pada aplikasi surat elektronik?

1.3 Batasan Permasalahan

Permasalahan yang dibahas pada Tugas Akhir ini memiliki batasan sebagai berikut:

1. Metode Enkripsi yang digunakan untuk mengamankan data adalah algoritma *Rivest Cipher* (RC4).
2. Model arsitektur yang digunakan adalah *client-server*.
3. Bahasa pemrograman yang digunakan adalah Node JS.
4. Kerangka kerja yang digunakan adalah *Express*.

5. Library tampilan antarmuka yang digunakan adalah Pug JS.
6. Database yang digunakan adalah *MySQL*.
7. Aplikasi yang dibuat adalah aplikasi surat elektronik yang sederhana, menggunakan *full service email*. Jadi alamat *email* yang digunakan tidak asli.
8. Pesan yang dienkripsi hanya isi pesan teks, lampiran tidak termasuk.

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Mengimplementasikan metode enkripsi menggunakan algoritma *Rivest Cipher* (RC4) dalam mengamankan aplikasi surat elektronik.
2. Mengetahui performa algoritma *Rivest Cipher* (RC4) dalam aplikasi surat elektronik berbasis web.
3. Mengetahui performa dari aplikasi surat elektronik berbasis web.

1.5 Manfaat

Manfaat yang diperoleh dari pembuatan Tugas Akhir ini adalah pengiriman data Aplikasi Surat Elektronik menjadi aman berdasarkan hasil implementasi metode enkripsi algoritma *Rivest Cipher* (RC4).

1.6 Metodologi

Pembuatan Tugas Akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

Tahapan awal dari Tugas Akhir ini adalah penyusunan Proposal Tugas Akhir. Proposal Tugas Akhir berisi pendahuluan, deskripsi dan gagasan metode – metode yang dibuat dalam Tugas Akhir ini. Pendahuluan ini terdiri atas hal yang menjadi latar belakang diajukannya Tugas Akhir, rumusan masalah yang diangkat, batasan masalah untuk Tugas Akhir, dan manfaat dari hasil pembuatan Tugas Akhir ini. Selain itu dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan Tugas Akhir. Terdapat pula sub bab jadwal kegitan yang menjelaskan jadwal pengerjaan Tugas Akhir.

1.6.2 Studi Literatur

Tugas Akhir ini menggunakan literatur *paper* yang berasal dari jurnal internasional bereputasi yaitu IEEE dan *Sciencedirect* untuk mencari informasi yang dapat dijadikan referensi dalam pengerjaan Tugas Akhir ini.

1.6.3 Implementasi Perangkat Lunak

Implementasi merupakan tahap untuk membangun metode-metode yang sudah diajukan pada proposal Tugas Akhir. Untuk membangun algoritma yang telah dirancang sebelumnya, maka dilakukan implementasi dengan menggunakan suatu perangkat lunak.

1.6.4 Pengujian dan Evaluasi

Pada tahap ini dilakukan pengujian terhadap hasil implementasi untuk mendapatkan performa dari metode enkripsi algoritma *Rivest Cipher* (RC4) dan juga performa aplikasi surat elektronik berbasis web.

1.6.5 Penyusunan Buku

Pada tahap ini dilakukan penyusunan buku yang menjelaskan seluruh konsep, teori dasar dari metode yang digunakan, implementasi, serta hasil yang telah dikerjakan sebagai dokumentasi dari pelaksanaan Tugas Akhir.

1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan Tugas Akhir adalah sebagai berikut:

1. Bab I. Pendahuluan
   Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan Tugas Akhir.
2. Bab II. Tinjauan Pustaka
   Bab ini berisi kajian teori dari metode dan algoritma yang digunakan dalam penyusunan Tugas Akhir ini. Secara garis besar, bab ini berisi tentang metode enkripsi dengan metode *Rivest Cipher* (RC4), Web Service REST API, bahasa pemrograman NodeJS, *Node Package Manage* dalam Node JS, kerangka kerja Express, MySQL Workbench sebagai aplikasi basis data, Pug JS yang digunakan sebagai antarmuka aplikasi.
3. Bab III. Perancangan Perangkat Lunak
   Bab ini berisi pembahasan mengenai kasus penggunaan aplikasi surat elektronik, arsitektur sistem yang berisi desain umum sistem, desain *backend*, desain *frontend*, dan desain dari algoritma *Rivest Cipher* (RC4)
4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi dari pengerjaan penelitian yang berisi implementasi Node JS, basis data yang digunakan, implementasi *backend*, *frontend*, dan implementasi dari algoritma *Rivest Cipher* (RC4).

5. Bab V. Hasil Uji Coba dan Evaluasi
   Bab ini berisikan hasil uji coba dari metode enkripsi menggunakan algoritma *Rivest Cipher* (RC4) berupa uji fungsionalitas dan uji performa aplikasi dan algoritma *Rivest Cipher* (RC4).
6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan Tugas Akhir, dan saran untuk pengembangan solusi ke depannya.
7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam Tugas Akhir.
8. Lampiran
   Dalam lampiran terdapat tabel-tabel data hasil uji coba.

BAB II
TINJAUAN PUSTAKA

Bab ini berisi tentang metode enkripsi dengan metode *Rivest Cipher* (RC4), Web Service REST API, bahasa pemrograman Node js kerangka kerja Express, aplikasi basis data MySQL, dan Pug JS sebagai library antarmuka aplikasi.

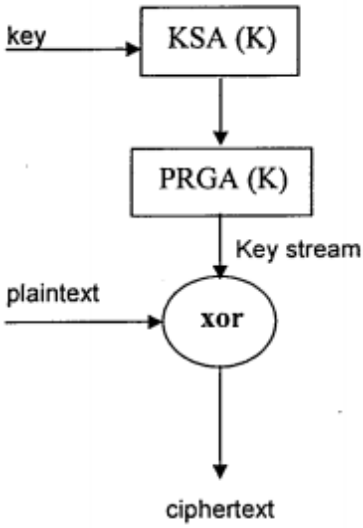
2.1 *Rivest Cipher* (RC4)

Rivest Cipher (RC4) adalah suatu algoritma enkripsi yang dirancang oleh Ron Rivest pada tahun 1987 untuk RSA Data Security dan baru di publikasikan untuk umum pada tahun 1997. Algoritma ini merupakan salah satu algoritma yang dapat digunakan untuk melakukan enkripsi data sehingga data asli hanya dapat dibaca oleh seseorang yang memiliki kunci enkripsi tersebut [2].

Awalnya *Rivest Cipher* (RC4) adalah sebuah rahasia dagang, akan tapi pada September 1994, kode tersebut dikirim oleh seseorang yang tidak diketahui ke milist Chypermunks dan menyebar ke banyak situs internet. Kode yang bocor tersebut akhirnya dikonfirmasi sebagai *Rivest Cipher* RC4 karena memiliki output yang sama dengan software dengan license *Rivest Cipher* RC4 di dalamnya. Karena algoritma sudah diketahui, RC4 tidak lagi menjadi rahasia dagang. Nama "RC4" sekarang adalah sebuah merek dagang, namun sering disebut sebagai "ARCFOUR" atau "ARC4" (artinya diduga RC4, karena algoritma ini tidak pernah dirilis secara resmi oleh RSA), untuk menghindari kemungkinan masalah tentang merek dagang [3].

Algoritma ini merupakan pengembangan dari *Rivest Cipher* (RC2) dan dikembangkan oleh Ronald Rivest. Perbedaan dengan algoritma sebelumnya adalah pada ukuran blok, ukuran kunci, dan jumlah *round* yang dilakukan. Bedanya, pada *Rivest Cipher* (RC2) ini menggunkan *block*

cipher dan *Rivest Cipher* (RC4) menggunakan *stream cipher* [4]. Pada *Rivest Cipher* (RC4) melakukan proses enkripsi dan dekripsi *one byte at time* dan disebut sebagai algoritma kriptografi yang simetrik karena menggunakan kunci yang sama untuk mengenkipsi ataupun mendekripsi suatu pesan, data, ataupun informasi [5]. Algoritma *Rivest Cipher* (RC4) terdiri atas 2 bagian yaitu *Key Scheduling* (KSA) dan *Pseudo Random Generation Algorithm* (PRGA) dan dapat dilihat arsitekturnya pada Gambar 2.1.



Gambar 2.1 Arsitektur *Rivest Cipher* (RC4)

Rivest Cipher (RC4) termasuk algoritma *stream cipher* yang paling cepat saat diimplementasikan dan lebih efisien penggunaannya sehingga banyak orang yang menggunakan algoritma ini. Misalnya dibandingkan dengan algoritma *stream cipher* SEAL. Pada algoritma SEAL proses enkripsinya perlu membangkitkan terlebih dahulu tiga buah tabel(S-box) berukuran besar yang isinya ditentukan oleh kunci yang

diterima. Hal tersebut membuat waktu inisialisasi dalam enkripsi plainteks menjadi sedikit lebih lama dibandingkan dengan algoritma *Rivest Cipher* (RC4) [6]. Maka dari itu, pada penilitian ini digunakan algoritma *Rivest Cipher* (RC4) untuk mengenkripsi data yang akan dikirmkan ke server.

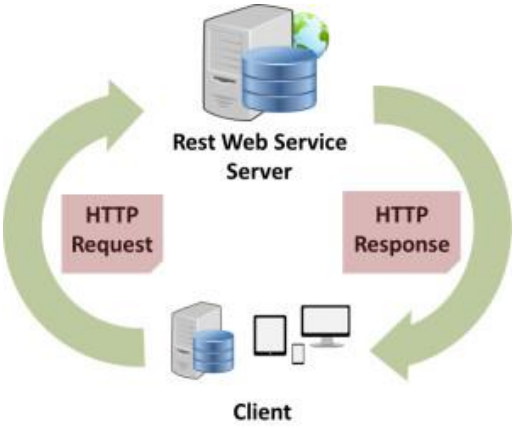
2.2 *Web Service* REST API

Representational State Transfer yang disingkat REST merupakan salah satu jenis arsitektur untuk penerapan web service yang menerapkan konsep perpindahan antar *state* [7]. REST menggunakan protokol HTTP yang bersifat *stateless*. Perintah HTTP yang bias digunakan adalah fungsi GET, POST, PUT atau DELETE. Hasil yang dikirimkan dari server biasanya dalam bentuk format XML atau JSON sederhana tanpa ada protokol pemaketan data, sehingga informasi yang diterima lebih mudah dibaca dan diparsing disisi *client* [8].

REST merupakan salah satu model *design* rancang bangun web service selain SOAP (*Simple Object Access Protocol*). REST mulai berkembang pesat karena sistemnya yang lebih sederhana daripada SOAP. Selain itu untuk melakukan testing terhadap REST service dapat dilakukan secara sederhana pada web browser tanpa harus melakukan simulasi *client – server* [9]. Tentunya REST berbeda dengan web service SOAP. Perbedaan REST dengan SOAP adalah terletak pada protokol yang digunakan, REST tidak terbatas pada protokol tertentu, web service ini dapat menggunakan protokol SMTP, FTP, HTTP, dsb. Di sisi lain, SOAP hanya mengandalkan pada XML saja yang bersifat *verbose* sehingga meningkatkan beban pada server dan meningkatkan trafik jaringan. Sebaliknya, web service REST bersifat ringan digunakan [10].

Pada penilitian ini memilih menggunakan web service REST karena dalam membangun layanan web membutuhkan protokol HTTP, sehingga web service ini mendukung dalam

pembuatan layanan web. Pemilihan teknologi yang tepat dalam pembuatan layanan web sangat penting karena akan mempengaruhi saat melakukan implementasi. Selain itu karena web service ini ringan dan mudah untuk diimplementasikan. Arsitektur web service REST dapat dilihat pada Gambar 2.2.



Gambar 2.2 Arsitektur REST API Web Service

Dari gambar diatas dapat diketahui cara kerja dari REST Web Service, yaitu pertama sebuah *client* mengirimkan sebuah data atau request melalui HTTP *Request* dan kemudian server merespon melalui HTTP *Response*.

2.3 Node JS

Node JS adalah sebuah platform perangkat lunak pada sisi server dan aplikasi jaringan. Ditulis menggunakan bahasa JavaScript yang dapat dijalankan pada Windows, Linux, dan Mac OS X. Node JS memiliki pustaka server sendiri sehingga memungkinkan untuk menjalankan server web tanpa

menggunakan web server seperti Apache sehingga Node JS ini ringan dan lebih efisien untuk digunakan.

Node JS memiliki *package* yaitu NPM *(*Node Packages Manager). NPM merupakan *package* terbesar dari *open source library* di dunia. Pada Node JS, bersifat asinkron, proses yang jalan tidak berdasar pada waktunya. Jadi setiap *request* yang masuk akan selalu di proses tanpa melihat urutan *request*nya.

Teknologi yang tidak memanfaatkan multi-thread ini memudahkan pengembang yang terkadang kesulitan mengatur sumber daya yang digunakan thread. Akhirnya banyak yang memanfaatkan kemampuan dasar NodeJS sebagai web server. Contoh penggunaan Node JS dapat dilihat pada Kode Sumber 2.1.

```
1   var express   = require('express');
2   var app       = express();
3   var fs        = require("fs");
4
5   app.get('/', function (req, res) {
6      fs.readFile( __dirname + "/" +
7      "users.json", 'utf8', function (err,
8      data) {
9        console.log( data );
10       res.end( data );
11     });
12  })
13
14  var server = app.listen(8081, function () {
15    var host = server.address().address
16    var port = server.address().port
17    console.log("Example app listening at
18                 http://%s:%s", host, port)
19  })
```

Kode Sumber 2.1 Contoh Penggunaan NodeJS

Pada penilitian ini, Node JS digunakan sebagai bahasa pemrograman dalam membuat aplikasi web. Karena bahasa pemrograman ini memiliki banyak kelebihan dibandingkan bahasa pemrograman yang lain. Diantaranya adalah, Node JS

dapat menangani beberapa permintaan *client* dan *server* secara asinkron. Hal ini berarti bahwa server Node JS dapat menangani beberapa permintaan paralel tanpa banyak kerumitan sehingga aplikasi web menjadi cepat [11]. Selain itu Node JS memiliki banyak *library* yang *powerful*. Node JS juga memiliki arsutektur *event driven* yang membuatnya cocok untuk *mobile apps*, *chatting apps*, atau aplikasi apapun yang menggunakan rest services sebagai *backend* [12] seperti pada penilitian ini.

2.4 NPM (Node Package Manager)

NPM adalah kependekan dari *Node Package Manager*. NPM adalah *package manager* untuk JavaScript. Jadi NPM adalah sebuah *tool*/aplikasi kecil untuk mengatur *package* JavaScript yang mengunakan Node JS.

Secara default data paket npm disimpan di registry *npmjs.org*. Sehingga untuk menginstall paket npm tertentu anda bisa mencari paket ini melalui *command* npm atau langsung melalui website. Sejak versi Node JS 0.6.3 *command* npm sudah ter-bundle dengan *installer* Node JS.

2.5 Kerangka Kerja Express JS

Express JS adalah sebuah kerangka kerja untuk pengembangan web yang muncul di awal kehadiran Node JS, merupakan yang tertua dan tetap terpopuler untuk digunakan. Express JS memiliki dokumentasi yang lengkap dan penggunaannya yang cukup mudah, dapat membuat kita mengembangkan berbagai produk seperti aplikasi web ataupun RESTful API. Dalam Express ini mengabstraksi seputar HTTP *request* dan *response*.

Pada penelitian ini, Express JS digunakan sebagai kerangka kerjanya karena Expess JS memiliki beberapa keunggulan, diantaranya adalah dukungan pembuatan *middleware*, dukungan terhadap HTTP Verb seperti POST,

GET, PUT, DELETE, OPTION, HEAD, dan sebagainya. Express JS juga sudah terpasang *template engine* Jade/Pug, memiliki manajemen *file* statik seperti CSS dan Javascript, serta bebas untuk di kostumisasi [13]. Selain itu karena kerangka kerja ini cukup popular digunakan oleh *developer* saat ini karena mudah untuk diimplementasikan. Sehingga kerangka kerja ini cocok digunakan untuk penelitian kali ini. Contoh penggunaan kerangka kerja Express JS dapat dilihat pada Kode Sumber 2.2.

Cara instalasi Express JS pada ubuntu adalah sebagai berikut:

```
$ npm install express
```

```
1  var express = require('express')
2  var app = express()
3
4  app.get('/', function (req, res) {
5    res.send('Hello World')
6  })
7
8  app.listen(3000)
```

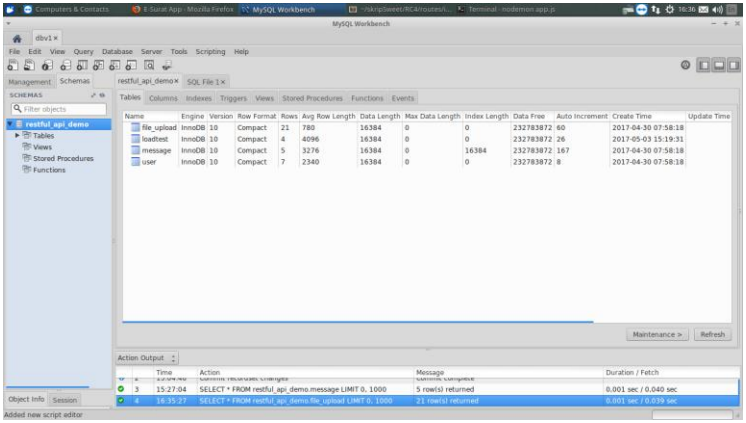
Kode Sumber 2.2 Contoh Penggunaan Express JS

2.6 MySQL

MySQL adalah salah satu jenis database server yang sangat terkenal dan banyak digunakan untuk membangun aplikasi web yang menggunakan database sebagai sumber dan pengolahan datanya. MySQL dikembangkan oleh perusahaan swedia bernama MySQL AB yang pada saat ini bernama Tcx DataKonsult AB sekitar tahun 1994-1995, namun cikal bakal kodenya sudah ada sejak tahun 1979. Awalnya Tcx merupakan perusahaan pengembang software dan konsultan database, dan saat ini MySQL sudah diambil alih oleh Oracle Corp [14].

Pada penilitian ini memilih MySQL sebagai database server karena memiliki banyak kelebihan, diantaranya adalah handal, cepat digunakan, *open-source*, memiliki kecepatan

yang tinggi dalam menangani *query*, mudah diimplementasikan, dan masih banyak lagi [15]. Pada penelitian kali ini, MySQL Workbench dipilih sebagai perangkat lunak untuk menangani database. MySQL Workbench merupakan salah satu perangkat lunak basis data produk dari MySQL. Perangkat aplikasi ini biasa digunakan oleh seorang arsitek basis data, pengembang basis data, serta administator basis data. MySQL Workbench menyediakan model data, pengembangan SQL, dan peralatan administrasi yang komperhensif untuk konfigurasi server basis data, administrasi pengguna, dan masih banyak lagi. MySQL Workbench tersedia pada platform Windows, Linux dan Mac OS.



Gambar 2.3 Antarmuka MySQL Workbench

2.7 Pug JS

Pug JS adalah sebuah library untuk melakukan *templating* HTML berperforma tinggi yang diimplementasikan dengan JavaScript untuk Node JS dan browser. Pug JS membantu dalam pembangunan halaman antarmuka menjadi lebih dinamis. Pug JS merupakan template yang popular yang bekerja menggunakan kerangka kerja Express. Perbedaan Pug

JS dengan HTML adalah pada Pug JS tidak memerlukan tag pembuka dan tag penutup. Pug JS dapat juga digunakan dalam dalam bahasa selain javascript yaitu PHP, Rubi, Scala, Python, dan Java.

Pada penelitian ini, memilih Pug JS sebagai *templating* HTML karena pad Pug JS terdapat *rendering* untuk mengubah HTML ke dalam Pug JS dengan mudah. Selain itu, Pug JS dapat langsung sinkron dengan bahasa pemrograman Node JS secara langsung.

Cara instalasi Pug JS pada ubuntu adalah sebagai berikut:

```
$ npm install pug
```

```
1  doctype html
2  html(lang='en-US')
3    head
4      title Pug JS
5    body
6      div(ng-app='')
7        p "Hello World"
```

Kode Sumber 2.3 Contoh Penggunaan Pug JS

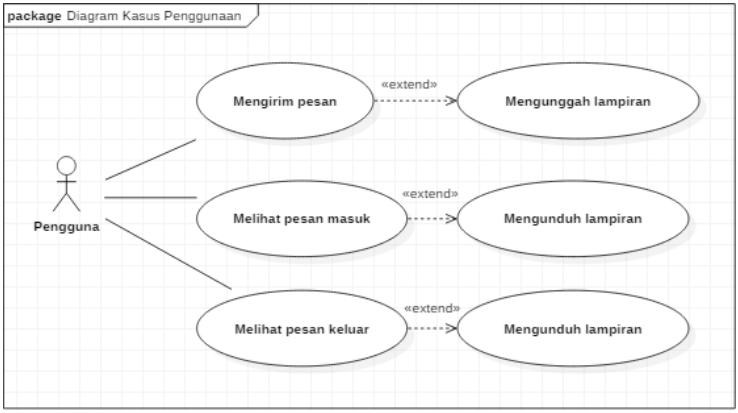
(Halaman ini sengaja dikosongkan)

BAB III
PERANCANGAN SISTEM

Bab ini menjelaskan tentang analisis dan perancangan sistem perangkat lunak. Sistem perangkat lunak yang dibuat pada Tugas Akhir ini adalah sistem surat elektronik sederhana yang isi pesannya dienkripsi menggunakan algoritma *Rivest Cipher* (RC4). Pada bab ini pula akan dijelaskan gambaran umum sistem dalam bentuk *flowchart.*

3.1 Kasus Penggunaan

Pada sub bab ini akan dijelaskan mengenai diagram kasus penggunaan sistem.



Gambar 3.1 Diagram Kasus Penggunaan

Penjelasan dari diagram kasus penggunaan pada Gambar 3.1 dideskripsikan pada Table 3.1.

Tabel 3.1 Daftar Kode Kasus Penggunaan

No	Nama	Aktor	Deskripsi
UC01	Mengirim pesan	Pengguna	Pengguna dapat melakukan pengiriman pesan yang berisi pesan teks atau lampiran.
UC02	Melihat pesan masuk	Pengguna	Pengguna dapat melihat pesan masuk pada akunnya.
UC03	Melihat pesan keluar	Pengguna	Pengguna dapat melihat pesan keluar pada akunnya.
UC04	Mengunggah lampiran	Pengguna	Pengguna dapat mengunggah lampiran yang ada pada saat mengirim pesan.
UC05	Mengunduh lampiran	Pengguna	Pengguna dapat mengunduh lampiran yang ada di pesan masuk.

3.2 Arsitektur Sistem

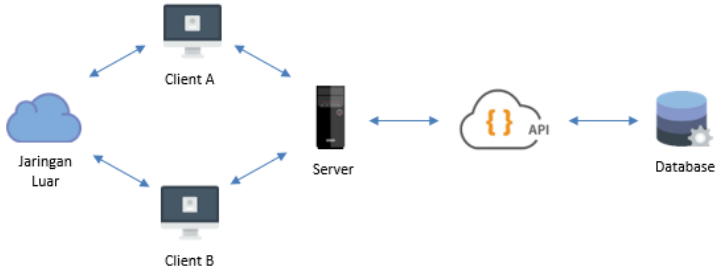
Pada sub bab ini membahas arsitektur sistem yang meliputi desain umum dari sistem yang akan dibangun, desain *backend*, desain *frontend*, dan desain algoritma yang dipakai.

3.2.1 Desain Umum Sistem

Pada Tugas Akhir ini adalah membuat sistem untuk mengamankan data pada surat elektronik dengan mengamankan

algoritma RC4. Sistem ini menggunakan Node JS sebagai Bahasa pemrogramannya. Sistem ini akan bekerja pada port 3030 dengan kronologi *Client* A akan mengirimkan sebuah pesan kepada *Client* B.

Client A akan mengirimkan pesan kepada *Client* B. *Client* A mengirimkan pesan berupa HTTP *Request* ke REST API. Pada REST API ini akan terjadi proses enkripsi data yang akan dikirim ke database. Kemudian REST API akan merespon dengan cara mengembalikan pesan yang telah terenkripsi. Setelah *Client* A mengirim pesan, *Client* B akan menerima pesan tersebut. *Client* B akan melakukan *request* ke REST API untuk melakukan dekripsi pesan. Kemudian REST API akan merespon dengan cara memberikan *plaintext* kepada Client B. Arsitektur dari proses tersebut dapat dilihat pada Gambar 3.2.



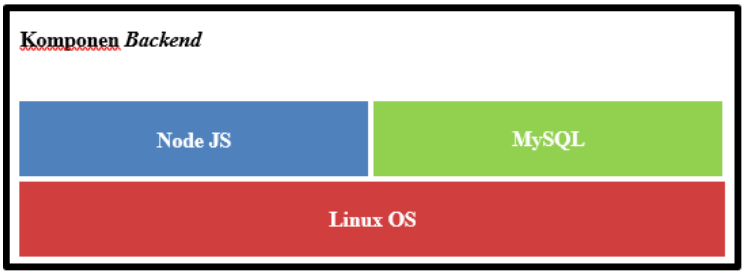
Gambar 3.2 Desain Sistem Secara Umum

3.2.2 Desain *Backend*

Backend dibuat dengan kerangka kerja dari bahasa Node JS yaitu Express JS. *Backend* akan menangani segala bentuk operasi basis data yang dikirimkan dari *frontend*, memberikan data yang dibutuhkan *frontend*. Terdapat dua jenis rute yang ditangani oleh *backend*, yaitu rute yang diakses tanpa autentikasi dan rute yang hanya bisa diakses oleh pengguna yang sudah melakukan login. Tabel 3.2 merupakan daftar rute pada *backend*. Rute yang dibuat dalam *backend*:

Tabel 3.2 Daftar Rute pada Backend

No	Rute	Metode	Hak akses	Aksi
1	*/login*	POST	Tidak ada	Melakukan autentikasi *username* dan *password*.
2	*/compose*	POST	Pengguna	Mengirim pesan berupa teks atau lampiran.
3	*/inbox*	POST	Pengguna	Menampilkan keseluruhan pesan masuk.
4	*/viewInbox*	POST	Pengguna	Membalas pesan berupa teks atau lampiran.



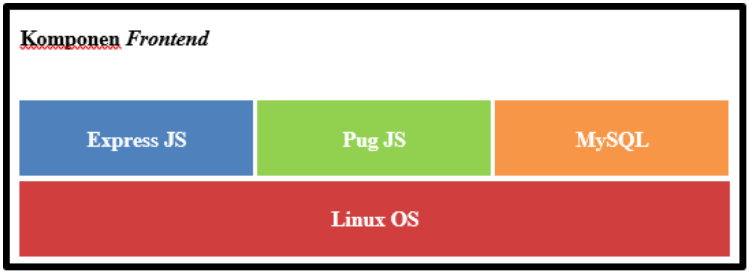
Gambar 3.3 Desain Arsitektur pada Backend

3.2.3 Desain *Frontend*

Bagian *frontend* menggunakan Pug JS. Pada *frontent* juga terdapat 2 rute yang ditangani, yaitu rute tanpa autentikasi dan ruten yang hanya bias diakses oleh pengguna yang sudah login. Tabel 3.3 merupakan daftar rute yang ditangani oleh *frontend*.

Tabel 3.3 Daftar Rute pada Frontend

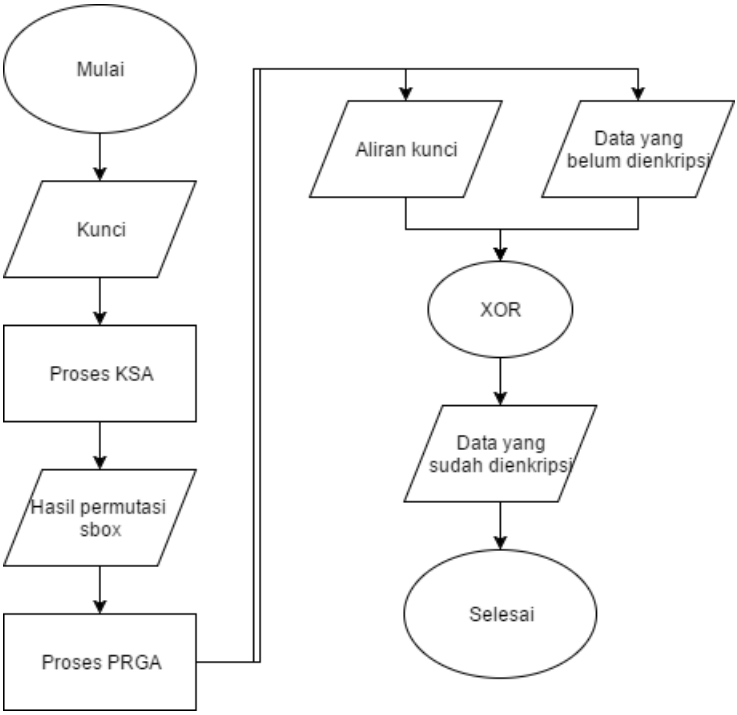
No	Rute	Me-tode	Hak akses	Aksi
1	/	GET	Tidak ada	Menampilkan halaman login.
2	*/compose*	GET	Pengguna	Menampilkan halaman kirim pesan atau lampiran.
3	*/inbox*	GET	Pengguna	Menampilkan halaman seluruh pesan masuk.
4	*/viewInbox/:msg_id*	GET	Pengguna	Menampilkan halaman detail dari pesan masuk.
5	*/outbox*	GET	Pengguna	Menampilkan halaman seluruh pesan keluar.
6	*/viewOutbox/:msg_id*	GET	Pengguna	Menampilkan halaman detail dari pesan keluar.
7	*/download/:id_file*	GET	Pengguna	Menampilkan tautan untuk mengunduh lampiran.
8	*/logout*	GET	Pengguna	Menampilkan tombol untuk logout.



Gambar 3.4 Desain Arsitektur pada Frontend

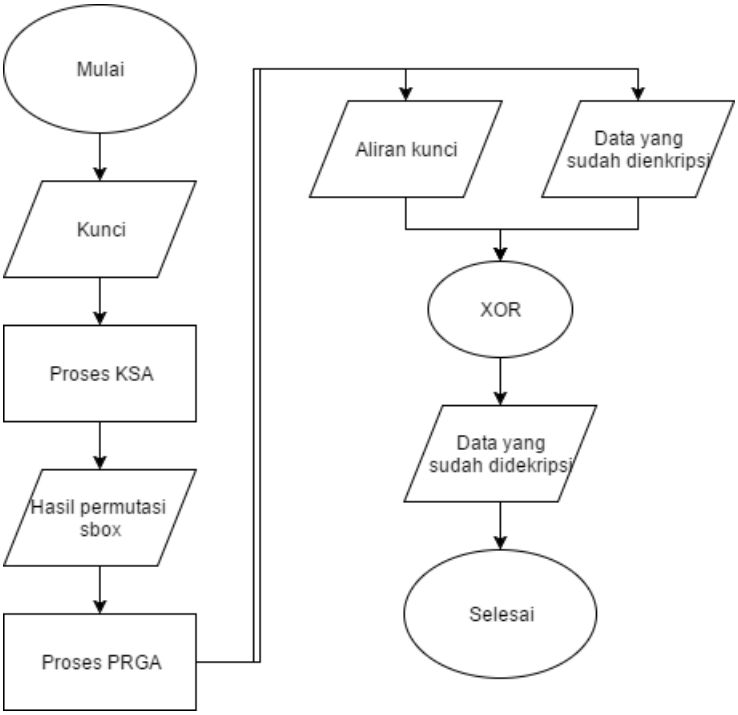
3.2.4 Desain Algoritma *Rivest Cipher* (RC4)

Rivest Cipher (RC4) merupakan suatu algoritma strem cipher dimana memiliki kehebatan untuk memeriksa tingkat kerahasiaan dan kinerja dengan melakukan perubahan tertentu dan melihat hasilnya. *Rivest Cipher* (RC4) dikenal cepat dan efisien, untuk itu dapat ditulis hanya menggunakan beberapa baris kode dan hanya membutuhkan 256 bit. Selain itu algoritma *Rivest Cipher* (RC4) termasuk algoritma kriptografi simetrik. Disebut algoritma kriptografi simetrik karena menggunakan kunci yang sama untuk mengenkripsi ataupun mendekripsi suatu pesan, data, ataupun informasi. Kunci enkripsi didapat dari sebuat 256 bit *state-array* yang diinisialisasi dengan sebuah *key* tersendiri dengan panjang 1-256 bit. Setelah itu, state-array tersebut akan diacak kembali dan diproses untuk menghasilkan sebuah kunci enkripsi yang akan di-XOR-kan dengan *plainteks* ataupun cipherteks Algoritma *Rivest Cipher* (RC4) terbagi menjadi dua bagian yaitu *Key Scheduling Algorithm* (KSA) dan *Pseudo Random Generation Algorithm* (PRGA). Alur dari proses enkripsi dan dekripsi ini dapat dilihat pada Gambar 3.5 dan Gambar 3.6.



Gambar 3.5 Diagram Alur Proses Enkripsi

Alur proses enkripsi menggunakan algoritma *Rivest Cipher* (RC4) ini yang pertama adalah menginisialisasi kunci untuk melakukan proses enkripsi. Proses ini bernama *Key Scheduling Algorithm* (KSA). Dari proses KSA ini akan dihasilkan permutasi *sbox.* Hasil dari permutasi ini didapatkan dari pertukaran nilai dari *sbox[i]* dan *sbox[j]* sebanyak 256 kali. Setelah itu akan dilakukan proses *Pseudo Random Generation Algorithm* (PRGA). Pada proses ini akan dihasilkan sebuah *keystream* yang didapatkan dari *sbox* baru yang digunakan untuk proses XOR dengan *plainteks* yang akan menghasilkan *ciphertext* atau pesan yang sudah terenkripsi.



Gambar 3.6 Diagram Alur Proses Dekripsi

Alur dari proses dekripsi sama dengan proses enkripsi. Pertama adalah menginisialisasi kunci. Setelah itu melakukan proses *Key Scheduling Algorithm* (KSA). Dari proses ini akan menghasilkan hasil permutasi dari *sbox*. Kemudian setelah itu melakukan proses *Pseudo Random Generation Algorithm* (PRGA) yang akan mengasilkan sebuah *keystream* yang digunakan untuk melakukan XOR dengan *ciphertext* agar mendapatkan pesan yang terdekripsi.

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi dari perancangan yang sudah dilakukan pada bab sebelumnya. Implementasi berupa kode sumber untuk membangun program. Sebelum masuk ke penjelasan implementasi, akan ditunjukkan terlebih dahulu lingkungan untuk melakukan implementasi.

4.1 Lingkungan Implementasi

Lingkungan implementasi dan pengembangan dilakukan menggunakan komputer Laboratorium Arsitektur dan Jaringan Komputer dengan spesifikasi Intel *Core* I3 2120 dan memori 8GB. Perangkat lunak yang digunakan dalam proses pengembangan antara lain:

- Sistem operasi Ubuntu Linux 14.04.3.
- Editor teks Sublime Text 3.
- Node JS sebagai bahasa pemrograman.
- Express JS sebagai kerangka kerja pemrograman.
- Pug JS sebagai tampilan antarmuka.
- MySQL untuk basis data.
- Peramban *web* Mozilla Firefox.

4.2 Implementasi Node JS

Seperti yang telah dijelaskan pada bab sebelumnya, instalasi paket yang dibutuhkan adalah menggunakan npm. Cara kerja np mini hanya mengunduh paket untuk kebutuhan pada sistem, hanya saja beberapa paket bias di install pada sistem operasi dan dapat digunakan pada *command line*. Untuk mengunduh paket yang dibutuhkan secara bersamaan digunakan perintah `npm install`. Perintah ini akan membaca

file `package.json` yang berisi daftar paket yang akan digunakan. Isi `package.json` untuk Tugas Akhir ini seperti berikut:

```
1   {
2     "name": "emailta",
3     "version": "0.0.0",
4     "private": true,
5     "scripts": {
6       "start": "node ./bin/www"
7     },
8
9     "dependencies": {
10      "arc4": "^3.3.7",
11      "body-parser": "~1.15.1",
12      "debug": "~2.2.0",
13      "express": "~4.13.4",
14      "jade": "~1.11.0",
15      "morgan": "~1.7.0",
16      "mysql": "^2.13.0",
17      "pug": "^2.0.0-beta11",
18      "serve-favicon": "~2.3.0"
19    }
20  }
```

Kode Sumber 4.1 Isi package.json

4.3 Implementasi Basis Data

Pada Tugas Akhir ini digunakan aplikasi basis data yang menyambungkan *backend* dengan database yaitu MySQL Workbench. Untuk dapat terhubung dengan basis data, dibutuhkan file `database.js` yang berisi seperti berikut:

```
1   var express    = require("express");
2   var mysql      = require('mysql');
3
4   var connection =
5   mysql.createConnection({
6    connectionLimit : 100000,
7    host            : '10.151.36.30',
8    user            : 'pisang',
9    password        : 'pisanggoreng',
10   database        : 'restful_api_demo'
11  });
12
13  connection.connect();
14  module.exports = connection;
```

Kode Sumber 4.2 Isi database.js

Terdapat 3 tabel yang telah dibuat dalam database, diantaranya adalah tabel *message*, *user*, dan *file_upload*. Tabel 4.1 merupakan detail implementasi dari basis data pada sistem. Detail meliputi nama atribut, tipe data dan deskripsi masing-masing kolom.

Tabel 4.1 Implementasi Basis Data

No	Nama Tabel	Nama Kolom	Tipe Data	Deskripsi
1	message	msg_id	integer(70)	*Primary key* tabel.
2	message	msg_source	varchar(45)	Alamat email pengirim.
3	message	msg_target	varchar(45)	Alamat email penerima.
4	message	msg_plain	text	Pesan teks.
7	message	msg_time	datetime	Waktu pesan

No	Nama Tabel	Nama Kolom	Tipe Data	Deskripsi
				ketika dikirim.
8	user	user_id	integer	*Primary key* tabel.
9	user	user_email	varchar(45)	Alamat email pengguna.
10	user	user_password	varchar(45)	Password pengguna.
11	file_upload	id_file	integer(70)	*Primary key* tabel.
12	file_upload	name_file	varchar(100)	Nama file yang di unggah.
13	file_upload	path_file	varchar(100)	Alamat folder file yang diunggah.
14	file_upload	size_file	varchar(45)	Ukuran file yang diunggah.
15	file_upload	time_file	timestamp	Waktu file diunggah.

4.4 Implementasi *Backend*

Implementasi *backend* pada Tugas Akhir ini adalah menggunakan kerangka kerja Express JS dan menggunakan Web Service REST API. Rute *backend* diimplementasikan dalam sebuah file javascript. Tabel implementasi rute *backend* dapat dilihat pada Tabel 4.1.

Tabel 4.2 Implementasi Rute pada Backend

No	Rute	Me-tode	Hak akses	Aksi	Langkah Proses
1	*/login*	POST	Tidak ada	Melakukan autentikasi *username* dan *password.*	1. Menerima data dari *frontend* berisi *username* dan *password*. 2. Jika pengguna terdaftar, maka pengguna akan diarahkan ke halaman pesan masuk. 3. Jika pengguna tidak terdaftar, maka pengguna akan tetap diarahkan pada halaman login.
2	*/compose*	POST	Pengguna	Mengirim pesan berupa teks atau lampiran.	1. Menerima data dari halaman *frontend* berisi *email* yang dituju, lampiran jika perlu, dan isi pesan. 2. Mengirim data ke database ketika tombol *send* diklik.

No	Rute	Me-tode	Hak akses	Aksi	Langkah Proses
					3. Ketika proses pengiriman data dilakukan, akan terjadi proses enkripsi data untuk melakukan penyandian data.
3	*/viewInbox*	POST	Pengguna	Membalas pesan berupa teks atau lampiran.	1. Menerima data dari *frontend* berisi pesan teks. 2. Mengirim data ke database ketika tombol *send* diklik.

4.4.1 Inisialisasi *Session*

Pada aplikasi web, kegunaan *session* adalah untuk melakukan aktivitas yang berhubungan dengan interaksi pengguna pada sebuah web server. Selain itu, kegunaan *session* adalah untuk memulai eksekusi *session* pada server dan kemudian menyimpannya pada browser. Pada Tugas Akhir ini, *script session* dsimpan pada folder terpisah, yaitu pada folder /*middleware*. *Script* ini akan dipanggil pada fungsi lain yang membutuhkan *session*. Jika halaman diberi *script session*, dan pengguna belum melakukan login atau salah dalam melakukan login, maka halaman akan diarahkan pada halaman login. Kode sumber *session* ditunjukkan pada Kode Sumber 4.

```
1  'use strict';
2  module.exports = function(req,res,next){
3   if(req.session.pisang) next();
4   else{
5     var response = {code:403,msg:"Forbidden
6                     Access!"};
7     req.session.response = response;
8     res.redirect("/login");
9   }
10 };
```

Kode Sumber 4.3 Isi auth.js

4.4.2 Halaman Login

Hampir semua aplikasi web menggunakan login untuk melakukan verifikasi pengguna. Pada umumnya, form login berisi *username* atau alamat *email* dan *password*. Pada halaman login ini memanggil fungsi *session* dan mengambil data dari database. *Query* yang dilakukan pada halaman ini adalah dengan cara melakukan *select* pada tabel *user* dengan atribut *user_email* dan *user_password.* Jika alamat *email* dan *password* tidak sesuai atau tidak ada di database, maka akan mengarah tetap pada halaman *login*, jika sesuai maka akan mengarah pada halaman *inbox*. Fungsi ini dapat dilihat pada Kode Sumber 4.4 dalam rute /*login* metode *post*.

```
1   router.get("/login", function(req, res){
2     res.render('login');
3     req.session.pisang = "login";
4
5   }).post("/login", function(req, res){
6     var query = "SELECT * FROM ?? WHERE ?? = ? AND
7                  ?? = ?";
8     var table = ["user", "user_email",
9                  req.body.email, "user_password",
10                 req.body.password];
11
12    query = mysql.format(query,table);
13    connection.query(query,function(err,rows){
14      if(err){
15        res.json({"Error" : true, "Message" :
16                    "Error executing MySQL query"});
17      }
18      else if(rows.length){
19        req.session.pisang = rows[0];
20        res.redirect("/inbox");
21      }
22      else {
23        res.redirect("/login");
24      }
25    });
26  });
```

Kode Sumber 4.4 Isi index.js

4.4.3 Halaman Mengirim Pesan

Pada halaman mengirim pesan, pengguna akan diberikan form dari halaman *frontend* yang berisi alamat *email* yang dituju, isi pesan teks, dan sebuah lampiran jika membutuhkan. Fungsi ini dapat dilihat pada Kode Sumber 4.5.

```
1   router.get('/compose',
2   require('../middleware/auth.js'), function(req,
3   res){
4       console.log("MASUK FUNGSI GET /COMPOSE");
5        res.render('compose', {
6          'login': req.session.pisang.user_email
7       });
8
9   }).post("/compose", multer({ dest:
10  './img/'}).single('fileUploaded'),function(req,
11  res){
12
13      var query = "INSERT INTO ??(??,??,??,??)
14                   VALUES (?,?,?,?)";
15      var table = ["message", "msg_source",
16                    "msg_target", "msg_plain",
17                    "msg_time",
18                    req.session.pisang.user_email,
19                    req.body.msg_target,
20                    ciphertext.toString(),myDate];
21
22      query = mysql.format(query,table);
23  });
```

Kode Sumber 4.5 Isi email.js

Dalam kode sumber pengiriman pesan terdapat satu rute /*compose* yang memiliki dua metode, yaitu *get* dan *post*. Pada metode *get* adalah melakukan *request session*. Kemudian pada metode *post* adalah *backend* dari pengiriman pesan. Pada rute ini terdapat fungsi *callback* Node JS. *Callback* adalah *asynchronous* yang ekivalen dengan sebuah fungsi. Fungsi *callback* akan dipanggil saat penyelesaian tugas yang diberikan dan fungsi ini dibuat untuk dieksekusi dalam fungsi lain. Pada kode sumber ini, fungsi pertama yang dilakukan adalah pengiriman pesan. Sebuah pesan yang akan dikirim akan dienkripsi terlebih dahulu pada *client-side* kemudian dilakukan *query* pengiriman data ke database. Setelah melakukan enkripsi pada *client-side* dan mengirim ke database, akan dilakukan fungsi *callback* setelahnya. Fungsi *callback* ini dilakukan untuk menambahkan pengiriman data lampiran ke database. Karena

tabel yang digunakan pada saat pengiriman data pesan teks dengan lampiran berbeda, yaitu tabel *message* dan *file_upload*, namun masih dalam satu form yang sama. Sehingga setelah melakukan pengiriman data ke tabel *message*, selanjutnya akan melakukan *query* pengiriman data ke tabel *file_upload*. Setelah *query* pengiriman data ke tabel *message* dilakukan, maka *id* yang dikirim akan disimpan pada *rows.insertId* yang selanjutnya akan digunakan pada *query* pengiriman data ke tabel *file_upload*.

Pada implementasi *backend* ini, proses enkripsi pesan disimpan pada sebuah *script* yang berisi rumus dari algoritma *Rivest Cipher* (RC4). Kemudian terdapat sebuah *script* lagi untuk menyimpan fungsi enkripsi dan dekripsi dan melakukan *required script* yang berisi rumus dari algoritma *Rivest Cipher* (RC4) dengan nama *main.js*. Setelah itu dengan menggunakan *tool browserify* untuk membungkus semua modul yang ada pada *script* yang berisi fungsi enkripsi dan dekripsi dengan melakukan perintah `browserify main.js -o bundle.js` pada terminal. Setelah itu akan dihasilkan *file bundle.js* yang berisi modul-modul proses enkripsi yang telah dibuat. Fungsi yang ada pada *script main.js* dapat dilihat pada Kode Sumber 4.6.

```
1   var rc4         = require("./RC4Cipher.js");
2   var CryptoJS    = require("crypto-js");
3
4   window.encrypt = function(message, key){
5     var ciphertext = CryptoJS.RC4.encrypt
6                       (message, key).toString();
7     return ciphertext;
8   }
9
10  window.decrypt = function(ciphertext, key){
11    var ciphertext = CryptoJS.RC4.decrypt
12                     (ciphertext, key).toString();
13    return ciphertext;
14  }
```

Kode Sumber 4.6 Isi main.js

File *bundle.js* akan tersimpan pada folder *public*. Setelah akan dibuat *script javascript* pada folder *public* yang berisi operasi enkripsi untuk *client-side* yang akan dipanggill pada *frontend. Script* tersebut dapat dilihat pada Kode Sumber 4.7.

```
1   $(document).on('click','#submit', function(e){
2     var text_ori = $('#message_ori').val();
3     var date     = $('#date_open').val();
4     var to       = $('#message_to').val();
5     var from     = $('#message_from').val();
6
7     var key      = from+to+date+'TA2017';
8
9     var sortAlphabets = function(stringConcat){
10      return stringConcat.split('')
11             .sort().join('');
12      };
13
14    key = sortAlphabets(key);)
15
16    text_ori   = window.encrypt(text_ori, key);
17    $('#message_c').val(text_ori);
18
19    $('#submitFormBtn').click();
20  });
```

Kode Sumber 4.7 Isi encrypt.js

Pada *script* diatas akan menampung *id* yang terdapat pada *form* dari *frontend*. *Message_ori* adalah *id* dari isi pesan teks yang akan dikirim, *date_open* adalah *id* untuk mendapatkan waktu pengiriman pesan, *message_to* adalah *id* untuk alamay *email* penerima, dan *message_from* adalah *id* untuk mendapatkan alamat *email* pengirim. Kemudian keempat variabel tersebut ditampung pada variabel *key* ditambah *string* TA2017 yang kemudian akan diurutkan berdasarkan alfabet yang akan digunakan sebagai kunci pengenkripsian pesan. Setelah itu dilakukan proses enkripsi dengan mengambil nilai dari *id* pesan teks yang akan dikirim dari *form*.

Dari hasil enkripsi yang sudah didapatkan pada *client-side*, kemudian dilakukan *query insert* ke *database*, yaitu *insert* alamat email pengguna sebagai *msg_source*, alamat email tujuan sebagai *msg_target*, isi pesan sebagai *msg_plain*, dan waktu pengiriman sebagai *msg_time*. Atribut-atribut tersebut ditampung pada tabel *message*.

Setelah melakukan *insert* isi pesan ke tabel *message*, berikutnya adalah melakukan *insert* lampiran ke tabel *file_upload*. Pada proses mengunggah lampiran ini, digunakan *multer* sebagai operasi pengunggahan *file*. Pada fungsi ini terdapat variabel yang akan menampung nama *file*, *path file*, dan ukuran *file*. Fungsi ini dapat dilihat pada Kode Sumber 4.8.

```
fs.rename(req.file.path,
'./img/'+req.file.filename+ '-' +
req.file.originalname, function(err) {
  var nameFile = req.file.filename+ '-' +
                 req.file.originalname;
  var pathFile = '/path/'+req.file.filename+ '-' +
                 req.file.originalname;
  var sizeFile = req.file.size;

  var query2 = "INSERT INTO ??(??,??,??,??) VALUES
               (?,?,?,?)";
  var table2 = ["file_upload", "name_file",
               "path_file", "size_file", "msg_id",
                nameFile, pathFile, sizeFile,
                rows.insertId];
  query2     = mysql.format(query2,table2);
  connection.query(query2,function(err,rows){
    if(err) {
      res.json({"Error" : true, "Message" : "Error
                executing MySQL query"});
    } else {
      console.log(rows);
    }
  });
});
```

Kode Sumber 4.8 Isi email.js

4.4.4 Halaman Pesan Masuk

Halaman pesan masuk termasuk halaman utama setelah pengguna berhasil melakukan *login*. Pada halaman pesan masuk, pengguna dapat melihat keseluruhan pesan masuk yang diterima. Fungsi ini dapat dilihat pada Kode Sumber 4.9.

```
router.get("/", require('../middleware
/auth.js'), function(req,res){
  var loginList    = [];
  global.inboxList = [];
  var query = "SELECT * FROM ?? WHERE ?? = ?
              ORDER BY ?? DESC";
  var table = ["message", "msg_target",
              req.session.pisang.user_email,
              "msg_time"];
  query     = mysql.format(query,table);
  connection.query(query,function  (err,rows){
    if(err) {
      return res.json({"Error" : true,
      "Message" 1 : "Error executing MySQL
                    query"});
      } else {
        for (var i = 0; i < rows.length; i++) {
          var login = req.session.pisang
                      .user_email;
          var getTime = '' + rows[i].msg_time;
          var time = getTime.substr(0,24);
          global.inbox = {
           'msg_time'     : time,
           'msg_target'   : rows[i].msg_target
        }
        inboxList.push(inbox);
        }
      }
      res.render('inbox', {
       'inboxList': inboxList,
       'login' : req.session.pisang.user_email
      });
   });
});
```

Kode Sumber 4.9 Isi inbox.js

Pada kode sumber halaman pesan masuk, terdapat *query* untuk mengambil seluruh atribut yang ada pada tabel *message* kemudian diurutkan berdasarkan waktu yang paling terakhir. Kemudian cara untuk menampilkan urutan pesan masuk yang ada pada *database* dilakukan perulangan *variable i*. Untuk *i* kurang dari jumlah atribut, maka akan ditampilkan alamat *email* yang masuk berupa *msg_source* dan waktu pengiriman berupa *msg_time*.

Pada rute ini juga menginisialisasi dalam variabel *inbox* pada yang bertipe *global* agar dapat dipanggil pada rute yang lain. Inisialisasi ini dilakukan dengan cara menyimpan atribut yang ada dalam *database* dalam sebuah variabel yang kemudian memasukkan objek ber*variabel inboxList* ke dalam variabel *global inbox*. Setelah dilakukan inisialisasi, maka juga dilakukan proses *render* agar dapat ditampilkan pada *frontend*.

4.4.5 Halaman Detail Pesan Masuk

Setelah pengguna melihat seluruh isi pesan masuk, pengguna dapat melihat detail dari masing-masing pesan masuk yang ada. Pada halaman ini, pengguna dapat melihat alamat *email* yang masuk, isi pesan, waktu kirim, dan lampiran jika ada. Selain itu pengguna juga dapat membalas pesan masuk secara langsung karena terdapat *form* balas pesan dibawah isi pesan yang ada. Fungsi ini dapat dilihat pada Kode Sumber 4.10.

```
router.get('/viewInbox/:msg_id/:key_sender',
require('../middleware/auth.js'), function(req,
res, next) {
  var query = "SELECT * FROM ?? WHERE ?? = ?";
  var table = ["message", "msg_id",
               req.params.msg_id];
  query     = mysql.format(query,table);
  connection.query(query,function(err, rows,
  fields){
    if(err) {
      return res.json({"Error" : true, "Message" :
                  "Error executing MySQL query"});
    } else {
      const db_message_plain   = rows[0].msg_plain;

      /*Modul For Decrypt By Key*/
      const key_from_sender_db= rows[0].key_sender;
      var cipher_config       = key_from_sender_db

      str1 = rows[0].msg_source;
      str2 = rows[0].msg_target;
      str3 = moment(rows[0].msg_time).format("YYYY-
             MM-DD HH:mm:ss");
      str4 = "TA2017";

      var stringConcat  = str1.concat
                          (str2,str3,str4);
      var sortAlphabets = function(stringConcat) {
      return stringConcat.split('')
             .sort().join('');
    }

    var keySort    = sortAlphabets(stringConcat);

    var plaintext = CryptoJS.RC4.decrypt
                   (db_message_plain.toString(),
                    md5keySort.toString())
                    .toString(CryptoJS.enc.Utf8);
  });
});
```

Kode Sumber 4.10 Isi inbox.js

Kode sumber diatas menjelaskan tentang *query* untuk menampilkan detail isi pesan yang diterima oleh pengguna. Untuk membuka detail pesan masuk akan dilakukan proses dekripsi pesan dengan cara melaukan penggabungan dari alamat *email* penerima, alamat *email* pengirim, waktu pesan terkirim, dan sebuah *string* yaitu TA2017. Kemudian dilakukan pengurutan *string* berdasarkan alfabet untuk dijadikan kunci yang digunakan untuk melakukan proses dekripsi. Setelah proses pendekripsian pesan dilakukan, berikutnya pada halaman ini juga terdapat form untuk membalas pesan secara langsung. Proses ini seperti *insert* ke *database* seperti biasanya. Pengguna memasukkan pesan teks kemudian klik tombol kirim. Fungsi ini dapat dilihat pada Kode Sumber 4.11.

```
1   router.post("/viewInbox",
2   require('../middleware/auth.js'),
3   function(req, res, next){
4     var source = req.session.pisang.user_email;
5     var target = viewInbox2.msg_source;
6     var pesan  = req.body.msg_plain;
7
8     var query  = "INSERT INTO ??(??,??,??)
9                   VALUES (?,?,?)";
10    var table  = ["message", "msg_source",
11                  "msg_target", "msg_plain",
12                  source, target, pesan];
13    query      = mysql.format(query,table);
14
15    connection.query(query,function
16    (err,rows,fields){
17      if(err) {
18        res.json({"Error" : true, "Message" :
19        "Error executing MySQL query"});
20      } else {
21         res.redirect("/inbox");
22      }
23    });
24  });
```

Kode Sumber 4.11 Isi inbox.js

Pada halaman detail pesan masuk ini juga terdapat *query* untuk menampilkan lampiran yang ada. Berbeda dengan *query* menampilkan pesan masuk yang berisi pesan teks, jika menampilkan pesan masuk yang berisi teks diambil dari tabel *message*, sedangkan untuk menampilkan lampiran diambil dari tabel *file_upload*. Proses tersebut dapat dilihat pada Kode Sumber 4.10.

```
1  router.get("/",
2  require('../middleware/auth.js'),
3  function(req,res){
4    var query = "SELECT * FROM ?? WHERE ?? = ?
5    ORDER BY ?? DESC";
6    var table = ["message", "msg_target",
7                 req.session.pisang.user_email,
8                 "msg_time"];
9    query     = mysql.format(query,table);
10 });
```

Kode Sumber 4.12 Isi inbox.js

Selain terdapat *form* untuk melakukan pembalasan pesan, pada halaman detail pesan masuk ini juga akan menampilkan sebuah tautan jika pengirim melampirkan *file* saat mengirim pesan. Fungsi pengunduhan lampiran pada detail pesan masuk dapat dilihat pada Kode Sumber 4.11.

```
1  router.get("/download/:id_file",
2  require('../middleware/auth.js'),
3  function(req,res){
4
5    var path = require('path');
6    var mime = require('mime');
7    var file = __dirname + '/..'  +
8               viewAttachment2.path_file;
9    res.download(file);
10 });
```

Kode Sumber 4.13 Isi inbox.js

4.4.6 Halaman Pesan Keluar

Pada halaman pesan keluar, pengguna dapat melihat keseluruhan pesan keluar yang diterima. Pada halaman ini, pengguna hanya dapat melihat alamat *email* yang keluar dan waktunya. Pada kode sumber halaman pesan keluar, terdapat *query* untuk mengambil atribut alamat *email* keluar dan waktu yang ada pada tabel *message* kemudian diurutkan berdasarkan waktu yang paling terakhir. Kemudian cara untuk menampilkan urutan pesan keluar yang ada pada *database* dilakukan perulangan *variable i*. Untuk *i* kurang dari panjang *rows*, maka akan ditampilkan alamat *email* yang masuk berupa *msg_source* dan waktu pengiriman berupa *msg_time*.

Pada rute ini juga menginisialisasi dalam variabel *outbox* pada yang bertipe *global* agar dapat dipanggil pada rute yang lain. Inisialisasi ini dilakukan dengan cara menyimpan atribut yang ada dalam *database* dalam sebuah variabel yang kemudian memasukkan objek bervariabel *outboxList* ke dalam variabel *global outbox*. Setelah dilakukan inisialisasi, maka juga dilakukan proses *render* agar dapat ditampilkan pada *frontend*. Fungsi ini dapat dilihat pada Kode Sumber 4.12.

```
router.get("/",
require('../middleware/auth.js'),
function(req,res){
  var outboxList = [];
  var query = "SELECT * FROM ?? WHERE ?? = ?
              ORDER BY ?? DESC";
  var table = ["message", "msg_source",
              req.session.pisang.user_email,
              "msg_time"];
  query     = mysql.format(query,table);

  connection.query(query,function
  (err,rows,fields){

    if(err) {
      return res.json({"Error" : true,
      "Message" : "Error executing MySQL
      query"});

    } else {
      for (var i = 0; i < rows.length; i++) {
        var login   = req.session.
                      pisang.user_email;
        var getTime = '' + rows[0].msg_time;
        var time    = getTime.substr(0,24);
        var outbox  = {
         'msg_id'     : rows[i].msg_id,
         'msg_time'   : time,
         'msg_target' : rows[i].msg_target,
         'login'      :login
        }
        outboxList.push(outbox);
      }
    }
    res.render('outbox', {
     'outboxList': outboxList,
     'login': req.session.pisang.user_email
        });
    });
});
```

Kode Sumber 4.14 Isi outbox.js

4.4.7 Halaman Detail Pesan Keluar

Setelah pengguna melihat seluruh isi pesan keluar, pengguna dapat melihat detail dari masing-masing pesan keluar yang ada. Pada halaman ini, pengguna dapat melihat alamat *email* yang keluar, isi pesan, waktu kirim, dan lampiran jika ada. Hampir sama dengan detail pesan masuk, pada kode sumber detail pesan keluar juga dilakukan proses dekripsi pesan. Bedanya, pada detail pesan keluar, tidak terdapat *form* membalas pesan. Pada halaman detail pesan keluar pengguna juga dapat melakukan unggah lampiran seperti pada halaman detail pesan masuk. Fungsi ini dapat dilihat pada Kode Sumber 4.13.

```
router.get('/viewOutbox/:msg_id/:key_sender',
require('../middleware/auth.js'), function(req, res,
next) {
  var viewOutbox = [];
  var query = "SELECT * FROM ?? WHERE ?? = ? AND ?? =
              ?";
  var table = ["message", "msg_id",
req.params.msg_id,
               "key_sender", req.params.key_sender];
  query     = mysql.format(query,table);

  connection.query(query,function(err,rows,fields){
    if(err) {
      return res.json({"Error" : true, "Message" :
                       "Error executing MySQL
query"});
    } else {
      /*Modul For Decrypt By Key*/
      const db_message_plain   = rows[0].msg_plain;
      const key_from_sender_db = rows[0].key_sender;

      str1 = rows[0].msg_source;
      str2 = rows[0].msg_target;
      str3 = moment(rows[0].msg_time).format("YYYY-
             MM-DD HH:mm:ss");
      str4 = "TA2017";

      var stringConcat  =
str1.concat(str2,str3,str4);
      var sortAlphabets = function(stringConcat) {
      return stringConcat.split('').sort().join('');

      var keySort    = sortAlphabets(stringConcat);
      var md5keySort = md5(keySort);

      var plaintext  = CryptoJS.RC4.decrypt
                       (db_message_plain.toString(),
                        md5keySort.toString())
                        .toString(CryptoJS.enc.Utf8);
    }
  });
});
```

Kode Sumber 4.15 Isi outbox.js

4.5 Implemetasi *Frontend*

Frontend adalah antarmuka untuk pengguna sistem. *Frontend* dibangun menggunakan pustaka *javascript* Pug JS. Dalam antarmuka pengguna, terdapat 6 kasus penggunaan, yaitu:

- Pengguna dapat melakukan login.
- Pengguna dapat melakukan pengiriman pesan berupa teks dan lampiran.
- Pengguna dapat melihat keseluruhan pesan masuk yang ada menggunakan sebuah kunci berupa alamat *email* yang masuk.
- Pengguna dapat melihat detail pesan masuk berupa isi pesan masuk dan lampiran jika ada, serta pengguna dapat membalas pesan ketika pengguna sudah membuka isi pesan masuk.
- Pengguna dapat melihat keseluruhan pesan keluar berupa alamat *email* yang dikirim oleh pengguna.
- Pengguna dapat melihat detail pesan keluar berupa isi pesan keluar dan lampiran jika ada.

Tabel 4.4 merupakan implementasi dari rute pada *frontend*.

Tabel 4.3 Implementasi Rute pada Frontend

No	Rute	Me-tode	Hak akses	Aksi	Langkah Proses
1	/	GET	Tidak ada	Menampilkan halaman login.	1. Pengguna mengisi *username* dan *password* pada halaman login. 2. Pengguna mengeklik tombol login. 3. Jika pengguna benar memasukkan *username* dan *password*, maka akan diarahkan ke halaman *inbox*. Jika salah, maka akan tetap berada pada halaman login.
2	*/compose*	GET	Pengguna	Menampilkan halaman kirim pesan atau lampiran.	1. Pengguna memilih menu *compose*. 2. Pengguna mengisi *form* berisi alamat email yang dituju, kunci, lampiran jika perlu, dan pesan teks.

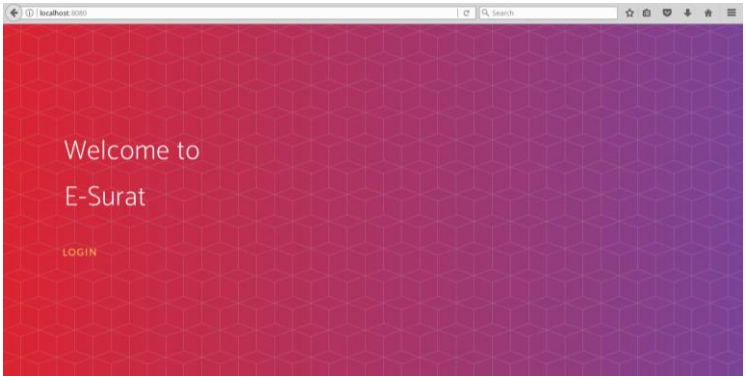
No	Rute	Me-tode	Hak akses	Aksi	Langkah Proses
					3. Pengguna mengeklik tombol *send*.
3	*/inbox*	GET	Pengguna	Menampilkan halaman seluruh pesan masuk.	1. Pengguna memilih menu *inbox*. 2. Pengguna melihat keseuruhan pesan masuk pada akunnya dari alamat email siapa saja.
4	*/viewInbox/:msg_id*	GET	Pengguna	Menampilkan halaman masing-masing pesan masuk.	1. Pengguna mengisi form *key* pada halaman *inbox* untuk memasukkan kunci agar dapat membuka pesan masuk. 2. Pengguna mengeklik tombol *view detail* untuk melihat detail isi pesan. 3. Jika pengguna benar memasukkan kunci, maka akan ditampilkan pesan

No	Rute	Me-tode	Hak akses	Aksi	Langkah Proses
					sesungguhnya, jika salah, maka akan ditampilkan pesan yang tidak dapat dibaca. 4. Pengguna akan melihat detail isi pesan masing-masing alamat email sesuai kunci yang benar setelah pengguna memasukkan kunci pada halaman *inbox*.
5	*/outbox*	GET	Pengguna	Menampilkan halaman seluruh pesan keluar.	1. Pengguna memilih menu *outbox*. 2. Pengguna akan melihat keseluruhan pesan keluar pada akunnya mengirim ke alamat email siapa saja.
6	*/viewOutbox/:msg_id*	GET	Pengguna	Menampilkan halaman masing-masing pesan keluar.	1. Pengguna mengeklik salah satu alamat email pada halaman *outbox*.

No	Rute	Me-tode	Hak akses	Aksi	Langkah Proses
					2. Pengguna akan melihat detail isi pesan keluar.
7	*/download/:id_file*	GET	Pengguna	Menampilkan tautan untuk mengunduh lampiran.	1. Pengguna dapat melakukan pengunduhan file lampiran pada saat pengguna membuka detail isi pesan pada halaman *inbox*.
8	*/logout*	GET	Pengguna	Menampilkan tombol untuk logout.	1. Pengguna mengeklik tombol *logout*. 2. Pengguna akan keluar dari akun dan akan diarahkan pada halaman login.

4.5.1 Halaman Utama

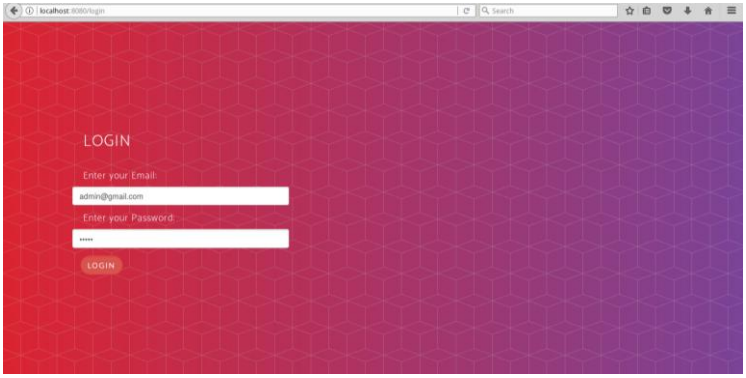
Untuk membuka aplikasi pada Tugas Akhir ini, dapat diakses di IP *private* 10.151.36.30 dengan *port* 8080. Setelah itu akan muncul halaman utama dari aplikasi ini. Tampilan halaman utama ini dapat dilihat pada Gambar 4.1



Gambar 4.1 Halaman Utama

4.5.2 Halaman Login

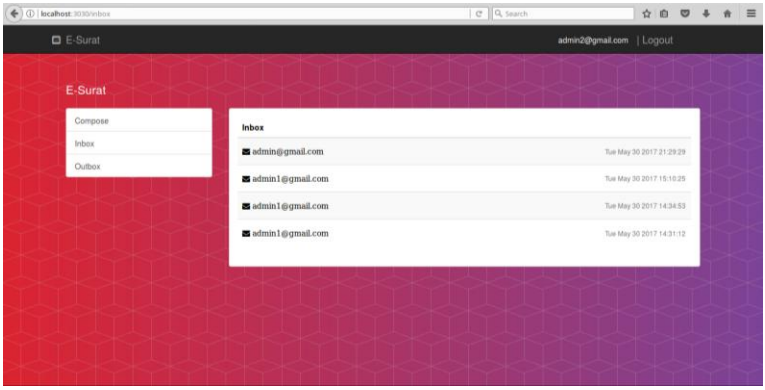
Setelah pengguna mengakses aplikasi *E-Surat* ini, pengguna harus melakukan login agar dapat masuk ke dalam aplikasi ini dengan cara memasukkan alamat *email* dan *password*. Tampilan halaman login dapat dilihat pada Gambar 4.2.



Gambar 4.2 Halaman Login

4.5.3 Halaman Pesan Masuk

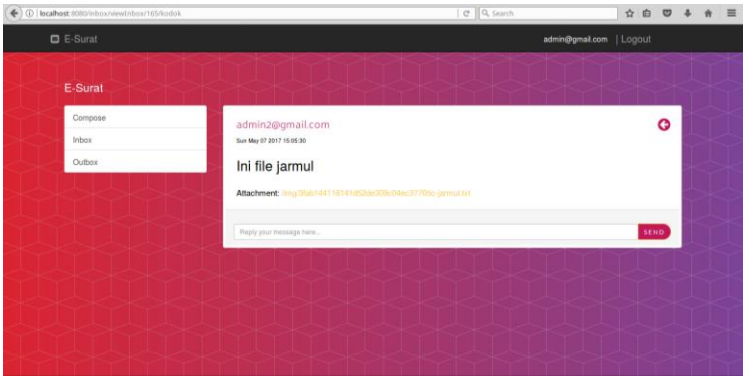
Setelah pengguna berhasil melakukan login, maka akan menuju ke halaman *inbox*, pada halaman ini pengguna dapat melihat keseluruhan pesan masuk yang ada berupa alamat *email* pesan masuk dan waktunya. Tampilan halaman pesan masuk dapat dilihat pada Gambar 4.3.



Gambar 4.3 Halaman Pesan Masuk

4.5.4 Halaman Detail Pesan Masuk

Setelah pengguna memasukkan kunci yang benar, maka akan terlihat detail dari isi pesan masuk. Jika kunci yang dimasukkan salah, maka pesan yang terlihat adalah pesan yang tidak dapat dibaca. Dapat dilihat pada halaman detail pesan masuk juga terdapat kolom untuk membalas pesan dari alamat *email* yang masuk. Tampilan halaman detail pesan masuk dapat dilihat pada Gambar 4.4.

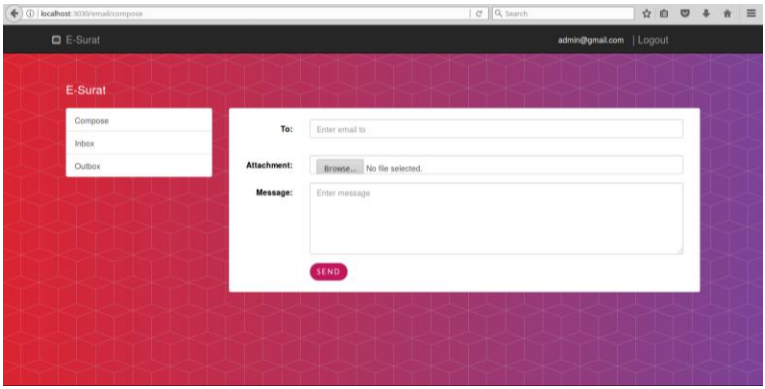


Gambar 4.4 Halaman Detail Pesan Masuk

Pada halaman pesan masuk ini pengguna juga dapat melakukan pengunduhan lampiran yang ada kemudian menyimpanannya pada komputer pengguna. Jika kunci yang dimasukkan salah maka lampiran tersebut tidak akan muncul dan pesan yang masuk adalah pesan teks yang tidak dapat dibaca.

4.5.5 Halaman Mengirim Pesan

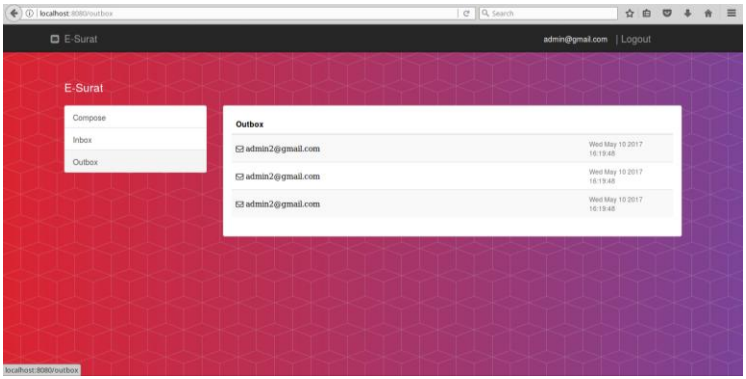
Pengguna juga dapat melakukan pengiriman pesan. Pada halaman pengiriman pesan ini terdapat 3 kolom yang harus diisi oleh pengguna yaitu kolom *to* yang berisi alamat *email* yang dituju, kolom *attachment* jika pengirim akan menambahkan lampiran, yang terakhir adalah kolom *message* yang berupa isi pesan teks yang akan dikirim pengirim untuk penerima. Kemudian pengguna dapat mengeklik tombol *send* untuk mengirim pesan tersebut. Tampilan halaman detail pesan masuk dapat dilihat pada Gambar 4.5.



Gambar 4.5 Halaman Mengirim Pesan

4.5.6 Halaman Pesan Keluar

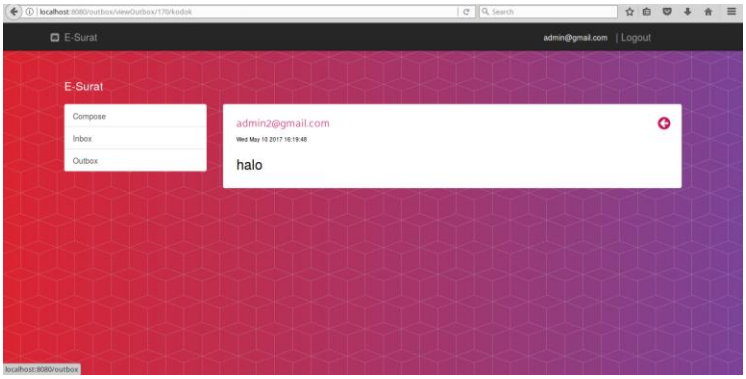
Setelah pengirim mengirim pesannya, maka pengirim juga dapat melihat kembali pesan keluar yang telah dikirim pada halaman pesan keluar. Pada halaman mirip dengan halaman pesan masuk. Bedanya adalah pada halaman pesan keluar tidak terdapat kolom kunci untuk membuka detail pesan keluar karena kolom kunci tersebut disembunyikan pada kode sumber di *frontend*. Tampilan halaman pesan keluar dapat dilihat pada Gambar 4.6.



Gambar 4.6 Halaman Pesan Keluar

4.5.7 Halaman Detail Pesan Keluar

Hampir sama juga dengan halaman detail pesan masuk, pada halaman detail pesan keluar juga akan terlihat isi dari pesan keluar yang berupa alamat *email* pengirim, pesan teks, dan lampiran jika ada. Tampilan detail pesan keluar dapat dilihat pada Gambar 4.7.



Gambar 4.7 Halaman Detail Pesan Keluar

4.6 Implementasi Algoritma *Rivest Cipher* (RC4)

Pada Tugas Akhir ini, proses pertama yang dilakukan dalam melakukan enkripsi dan dekripsi menggunakan algoritma *Rivest Cipher* (RC4) ini adalah proses *Key Scheduling Algorithm* (KSA), dimana proses ini melakukan pemberian nilai inisialisasi pada table *sbox, key* dan selanjutnya melakukan proses permutasi sebanyak 256 iterasi.

Pada algoritma ini, inisialisasi *sbox* dan *key* adalah langkah pertama dalam proses enkripsi dan dekripsi pada algoritma *Rivest Cipher* (RC4). Ketika proses inisialisasi *sbox* dan *key* terlihat tabel *sbox* dalam *array* 0 sampai 255 mempunyai nilai 0 sampai 255, sedangkan *key* dalam *array* 0 sampai 255 mempunyai kode asciii dari setiap karakter *string key* yang diberikan. Kemudian pada proses permutasi *sbox*, dilakukan sebanyak 256 iterasi dengan mempertukarkan nilai *sbox* dalam *array* i dengan nilai 0 sampai 255 dengan nilai *sbox* dalam *array* j yaitu pada rumus j = (j+sbox[i[+key[i]) mod 256.

Proses kedua adalah PRGA. Setelah memiliki *state array* yang telah diacak, maka selanjutnya akan menginisialisasi kembali i dan j dengan 0. Setelah itu, akan melakukan pseudorandom atau *Pseudo Random Generation Algorithm* (PRGA) untuk menghasilkan kunci enkripsi yang kemudian akan di XOR-kan dengan *plaintext*. Proses iterasi terjadi sebanyak panjang dari *plaintext*. *Key stream* diperoleh dari proses permutasi tabel sbox, dengan rumusan sebagai berikut:

i = (i+1) mod 256

j = (j+sbox[i]) mod 256

Selanjutnya dipermutasikan dengan menukar nilai dari S[i] dan S[j] dengan rumus *Swap (sbox[i], sbox[j])*. Setelah mendapatkan *keystream* maka akan di XOR dengan *plainteks* untuk mendapatkan *cipherteks*nya. Pseudocode dari algoritma ini dapat dilihat pada Kode Sumber 3.1.

```
1   // Inisialisasi sbox dan key
2   len_key = length(key_str)
3   for(i=0; i<=255; i++) {
4         key_chr = substr(key_str, (i mod
5   len_key), 1)
6         key[i]  = ord(key_str)
7         sbox[i] = i
8   }

9   //Permutasi sbox
10  for(i=0; i<=255; i++){
11    j = (j + sbox[i] + key[i] mod 256)
12    swap(sbox[i], sbox[j])
13  }
14
15  i=0
16  j=0
17  for(n=1; n<=strlen(plaintext); n++){
18    i = (i + i) mod 256;
19    j = (j + sbox[i]) mod 256;
20  swap (sbox[i], sbox[j])
21
22  keystream = sbox[(sbox[i] + sbox[j]) mod
23  256]
24  chrteks = substr(plaintext, n-1, 1);
25  ordchr = ord(chrteks);
26  cipher = ordchr xor keystream;
27  }
```

Kode Sumber 4.16 Pseudocode Algoritma Rivest Cipher (RC4)

(Halaman ini sengaja dikosongkan)

BAB V
UJI COBA DAN EVALUASI

Bab ini akan dijelaskan mengenai skenario uji coba pada perangkat lunak yang telah dibangun. Setelah itu, hasil uji coba akan dievaluasi kinerjanya sehingga dapat diputuskan apakah perangkat lunak ini mampu menyelesaikan permasalahan yang telah dirumuskan diawal. Secara garis besar, bab ini berisikan pembahasan mengenai lingkungan pengujian, data pengujian, dan uji kinerja.

5.1 Lingkungan Pengujian

Lingkungan untuk pengujian menggunakan dua buah komputer yang terdiri dari satu *web server* dan satu laptop penguji. Pada komputer *web server* ini hanya untuk menyalakan server dari aplikasi web. Sedangkan komputer kedua digunakan untuk mengakses aplikasi web tersebut.

Proses pengujian dilakukan di Laboratorium Arsitektur dan Jaringan Komputer gedung Teknik Informatika ITS. Spesifikasi perangkat keras dan perangkat lunak yang digunakan pada masing-masing komputer adalah sebagai berikut:

- *Web server* (IP 10.151.36.30)
  - Perangkat Keras
    - ✓ Komputer fisik
    - ✓ Prosesor Inter Core i3
    - ✓ RAM 4 GB
    - ✓ *Harddisk* 500 GB
  - Perangkat Lunak
    - ✓ Ubuntu Server Linux 14.02.03 LTS
    - ✓ Wireshark
    - ✓ MySQL Workbench
- Pengujian Performa

- Perangkat Keras
  - ✓ Laptop
  - ✓ Prosesor AMD FX Quad Core x4
  - ✓ RAM 4 GB
  - ✓ *Harddisk* 500 GB
- Perangkat Lunak
  - Windows 10
  - Google Chrome

5.2 Skenario Uji Coba

Skenario uji coba dilakukan dalam beberapa tahap uji coba:

- Uji Fungsionalitas
- Uji Performa

5.2.1 Uji Fungsionalitas

Uji coba fungsionalitas dilakukan dengan cara melakukan *sniffing* menggunakan aplikasi perangkat lunak Wireshark. Pengguna akan melakukan pengiriman pesan menggunakan peramban web. Permaban web yang digunakan adalah *Google Chrome*. Uji coba ini berguna untuk mengecek saat pengguna melakukan pengiriman pesan, apakah pesan yang terkirim terdeteksi pada Wireshark. Pada uji coba kali ini digunakan dua protokol untuk mengujinya, yaitu protokol HTTP (*Hypertext Transfer Protocol*) pada port 3030 dan protokol HTTPS (*Hypertext Transfer Protocol Secure*) pada port 3000. HTTP merupakan sebuah protokol yang mengatur komunikasi antar *client* dan *server*. Sedangkan HTTPS merupakan versi *secure* daripada HTTP. Perbedaannya adalah pada keamanan yang dikirimkan, yaitu autentikasi server, kerahasisan data, dan integritas data. Selain untuk melakukan *sniffing* pesan yang dikirim, juga akan melakukan ujicoba apakah operasi-operasi selain pengiriman pesan dapat

memberikan hasil yang diharapkan. Desain arsitektur dari uji fungsionalitas dapat dilihat pada Gambar 5.1.

Gambar 5.1 Desain Arsitektur Uji Fungsionalitas

Tabel 5.1 Implementasi Uji Fungsionalitas

No	Nama Kegiatan	Uji Coba	Hasil Harapan
1	*Login* sebagai pengguna	Dapat *login* ke dalam sistem dengan *username* dan *password* yang sudah terdaftar.	Pengguna dapat memasuki halaman sesuai *role*.
2	Membuka halaman pesan masuk	Dapat mengakses halaman pesan masuk setelah berhasil melakukan login.	Pengguna dapat membuka seluruh isi pesan masuk.

No	Nama Kegiatan	Uji Coba	Hasil Harapan
3	Membuka halaman pesan keluar	Dapat mengakses halaman pesan keluar.	Pengguna dapat membuka seluruh isi pesan keluar.
4	Mengirim pesan	Dapat mengirim pesan yang berisi alamat email yang akan dituju, kunci, isi pesan, dan lampiran jika ada.	Pengguna berhasil mengirim ke alamat email tujuan dan ketika dilakukan *sniffing* menggunakan Wireshark, pesan yang terkirim tidak akan terekam jika menggunakan protokol HTTPS.
5	Membuka halaman detail pesan masuk.	Dapat membuka halaman detail pesan masuk menggunakan sebuah kunci.	Pengguna dapat berhasil melihat isi detail pesan masuk menggunakan kunci yang benar.
6	Membuka halaman detail pesan keluar.	Dapat membuka halaman detail pesan keluar.	Pengguna dapat berhasil melihat isi detail pesan keluar.
7	Mengunggah lampiran.	Dapat mengunggah lampiran berekstensi	Pengguna dapat berhasil mengunggah lampiran ketika mengirim pesan.

No	Nama Kegiatan	Uji Coba	Hasil Harapan
		txt, word, jpg, pdf, zip.	
8	Mengunduh lampiran.	Dapat mengunduh lampiran pada saat menerima pesan.	Pengguna dapat berhasil mengunduh lampiran dan menyimpannya pada komputer.

5.2.2 Uji Performa

Uji performa pada penelitian kali ini dibagi menjadi dua kasus, yaitu uji performa aplikasi dan uji performa algoritma. Uji performa aplikasi yaitu menguji performa aplikasi pada komputer *web server*. Uji coba pada komputer *web server* dilakukan untuk menguji kemampuan *web server* dalam menangani *request* dari pengguna yang masuk. Pada uji performa kali ini kita akan mendapatkan *total request* yang ada, *request per second*, dan *time request*. Kemudian, uji coba algoritma adalah uji coba menghitung total waktu yang dihasilkan ketika melakukan proses enkripsi dan dekripsi pesan.

5.2.2.1 Uji Performa Aplikasi

Uji coba ini adalah pengujian tentang performa aplikasi surat elektronik. Yang akan diuji adalah jika dibatasi maksimal permintaan pengguna, maksimal permitaan perdetiknya, dan jumlah pengguna yang mengakses sistem secara bersamaan, maka akan diketahui total waktu yang dihasilkan berdasarkan studi kasus yang terjadi. Selain itu, juga akan diketahui permintaan setiap detiknya berdasarkan waktu yang dicapai.

Uji performa pada penelitian kali ini menggunakan *loadtest*. *Loadtest* atau uji beban merupakan salah satu metode uji pembebanan untuk menguji performa *web server* yang dijalankan. Uji beban ini bertujuan untuk mengukur respon dari sistem pada penelitian ini. Selain itu pengujian ini dilakukan untuk mengetahui kondisi puncak normal dari sistem yang diuji. Hal ini membantu untuk mengidentifikasi kapasitas operasi maksimum dari sistem tersebut. Pada uji performa yang menggunakan uji beban *loadtest* ini, akan dihasilkan beberapa informasi tentang respon yang didapat. Diantaranya adalah, *concurrent, maximal request, total eror, request per second, latency,* dan waktu(??)

Concurrent merupakan total *request* yang dilakukan oleh pengguna secara bersamaan. *Maximal request* merupakan maksimal dari total *request* yang diinginkan. Kemudian total eror merupakan jumlah eror yang dihasilkan pada uji beban tersebut. *Request per second* adalah permintaan pernggunga setiap detiknya pada sistem aplikasi yang diuji. Sedangkan *latency* adalah jumlah waktu yang dibutuhkan sebuah *request* untuk melintasi sebuah sistem.

5.2.2.2 Uji Performa Algoritma

Uji coba perfoma algoritma adalah menghitung kecepatan waktu untuk melakukan enkripsi dan dekripsi pesan. Uji performa algoritma pada penelitian ini menggunakan *performance.now()*. *Performance.now()* merupakan salah satu *library* dari Node JS menggunakan bahasa Javascript. Penggunaan fungsi ini dilakukan pada sebelum dan sesudah fungsi enkripsi dan dekripsi dengan cara menampung fungsi *performance.now()* pada sebuah variabel kemudian dihitung selisihnya untuk mendapatkan kecepatan waktu enkrips dan dekripsi. Hasil yang diharapkan pada uji performa algoritma ini adalah mengetahui seberapa lama proses untuk enkripsi dan

dekripsi pesan. *Syntax* dari *library* ini dapat dilihat pada Kode Sumber 5.1.

```
1 var t0 = performance.now();
2 doSomething();
3 var t1 = performance.now();
4 console.log("Call to doSomething took " + (t1
5 - t0) + " milliseconds.")
```

Kode Sumber 5.1 Penggunaan performance-now

5.3 Hasil Uji Coba

Pada subbab ini akan dijelaskan mengenai hasil pengujian yang dilakukan pada skenario pengujian yang telah ditentukan. Pengujian yang dilakukan adalah uji fungsionalitas dan uji dan performa.

5.3.1 Hasil Uji Fungsionalitas

Tabel 5.2 merupakan hasil uji fungsionalitas. Uji fungsionalitas menggunakan peramban Google Chrome pada komputer penguji dan mengguankan perangkat lunak Wireshark untuk melakukan *sniffing* data yang terkirim pada komputer *web server*.

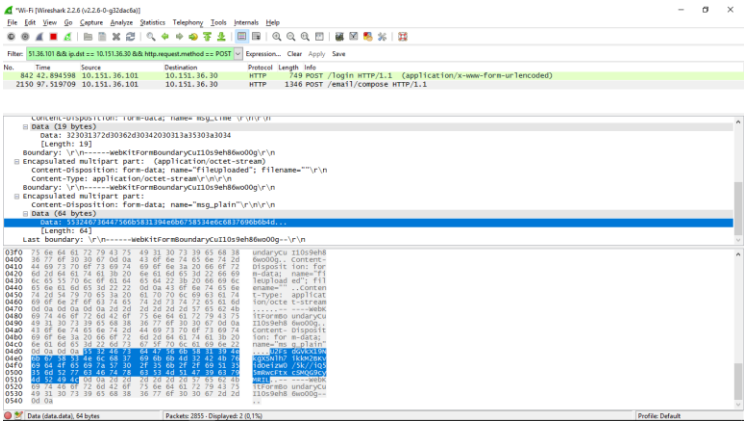
Tabel 5.2 Hasil Eksekusi Uji Fungsionalitas

No	Nama Kegiatan	Uji Coba	Hasil Harapan
1	*Login* sebagai pengguna	Dapat *login* ke dalam sistem dengan *username* dan *password* yang sudah terdaftar.	Sukses.

No	Nama Kegiatan	Uji Coba	Hasil Harapan
2	Membuka halaman pesan masuk	Dapat mengakses halaman pesan masuk setelah berhasil melakukan login.	Sukses dan muncul seluruh pesan masuk.
3	Membuka halaman pesan keluar	Dapat mengakses halaman pesan keluar.	Sukses dan muncul seluruh pesan keluar.
4	Mengirim pesan	Dapat mengirim pesan yang berisi alamat email yang akan dituju, isi pesan, dan lampiran jika ada.	Sukses.
5	Membuka halaman detail pesan masuk.	Dapat membuka halaman detail pesan masuk.	Sukses dan muncul isi detail pesan masuk.
6	Membuka halaman detail pesan keluar.	Dapat membuka halaman detail pesan keluar.	Sukses dan muncul isi detail pesan keluar.
7	Mengunggah lampiran.	Dapat mengunggah lampiran berekstensi txt, word, jpg, pdf, zip.	Sukses.

No	Nama Kegiatan	Uji Coba	Hasil Harapan
8	Mengunduh lampiran.	Dapat mengunduh lampiran pada saat menerima pesan.	Sukses dan berhasil menyimpan lampiran pada komputer.

Pada pengujian fungsionalitas ini juga dilakukan *sniffing* menggunakan Wireshark saat melakukan uji pengiriman pesan untuk menguji keamanan pesan yang terkirim sudah terenkripsi atau belum. Cara mengujinya adalah dengan cara menyalakan server pada komputer *web server*. Komputer *web server* memiliki IP 10.151.36.30 menggunakan salah satu komputer yang ada di Laboratorium Arsitektur dan Jaringan Komputer. Setelah itu pada komputer penguji mengakses halaman 10.151.36.30 dengan port 3030. Untuk mendapatkan apakah pesan yang terkirim sudah terenkripsi atau belum adalah dengan melakukan *filter* pada Wireshark yaitu ip.src == 10.151.36.101 && ip.dst == 10.151.36.30 && http.request.method == POST. Alamat IP 10.151.36.101 adalahh alamat IP komputer penguji dengan menggunakan WIFI Mikrotik yang ada pada Laboratorium Arsitektur dan Jaringan Komputer. Hasil *sniffing* pada HTTP dapat dilihat pada Gambar 5.2. Pesan yang ditampilkan pada Wireshark adalah pesan yang sudah terenkripsi.



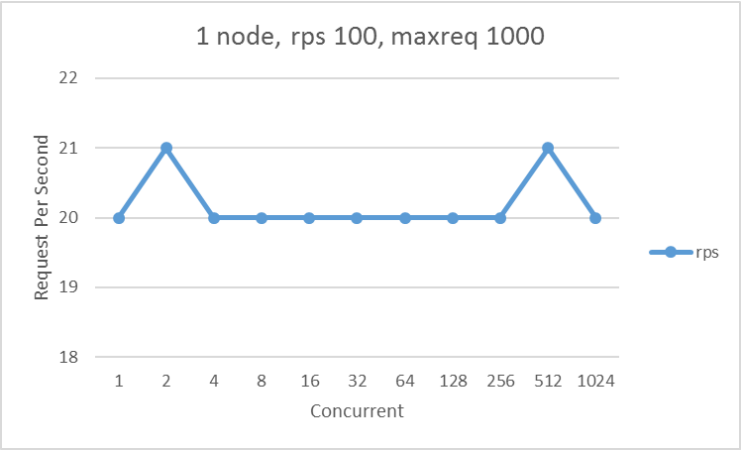
Gambar 5.2 Hasil Sniffing HTTP

5.3.2 Hasil Uji Performa

Pada uji performa ini terdapat dua subbab yang dihasilkan yaitu hasil uji coba performa aplikasi dan hasil uji coba algoritma.

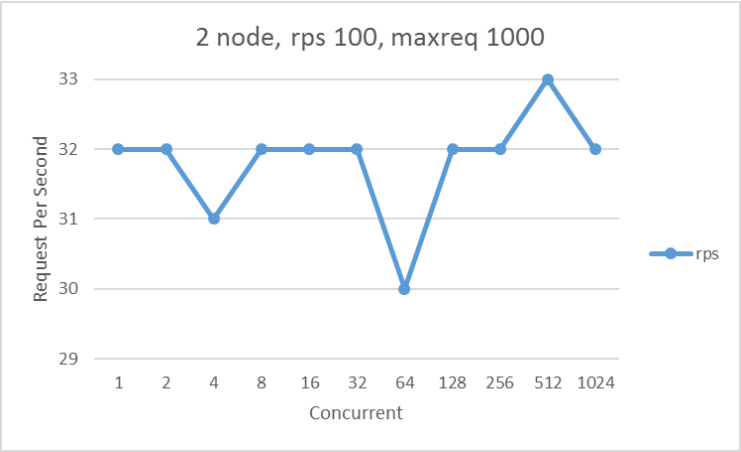
5.3.2.1 Hasil Uji Performa Aplikasi

Pada uji performa ini dilakukan dengan cara menjalankan fungsi *loadtest* dan menjalankan server dari sistem aplikasi. Saat menjalankan server dari sistem aplikasi, sistem ini akan diuji menjadi 8 node. Node yang dimaksud adalah pembagian CPU. Sehingga akan didapatkan performa dari masing-masing pembagian CPU dari menggunakan 1 sampai 8 node CPU. Sebelumnya akan diinisialisasi terlebih dahulu *rps* dan *maximal request* yang akan diuji, yaitu *rps* 100 dan *maximal request* 1000. Selain itu juga mengeset *concurrent* dari 1 *cncurrent* hingga 1024 concurrent. Hasil uji performa ini akan digambarkan dalam grafik. Grafik yang dihasilkan adalah hasil dari uji performa menggunakan 1 hingga 8 node CPU. Grafik tersebut dapat dilihat pada Gambar 5.4 hingga Gambar 5.11.



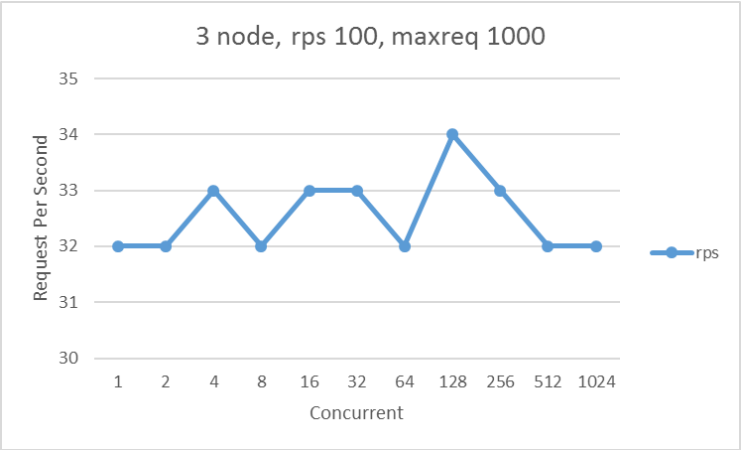
Gambar 5.3 Grafik Performa 1 node CPU

Dapat dilihat pada grafik diatas bahwa pada penggunaan 1 node CPU ini dihasilkan *rps* yang ada adalah 20 dan 21 *request* saat mengakses sistem. 20 *request* terjadi ketika *concurrent*nya 2 dan 512. Selain itu stabil pada 20 *request*.



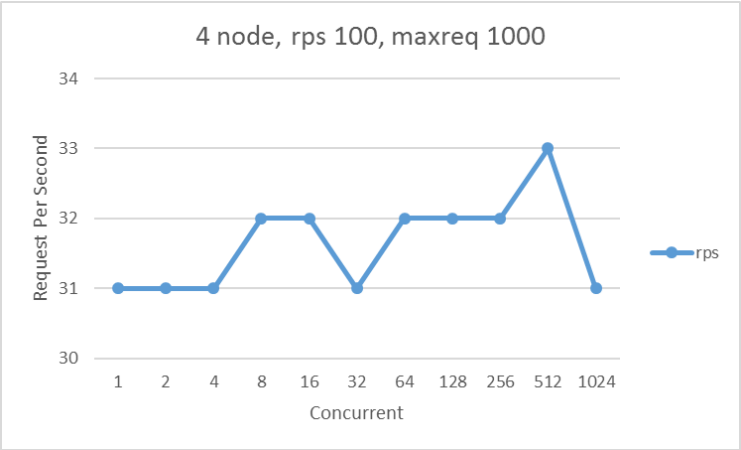
Gambar 5.4 Grafik Performa 2 node CPU

Pada Gambar 5.5 terjadi kenaikan *request per second*nya, dimana saat 1 node CPU rata-rata *request* adalah 20 dan pada pembagian 2 node CPU rata-rata *request* adalah 32.



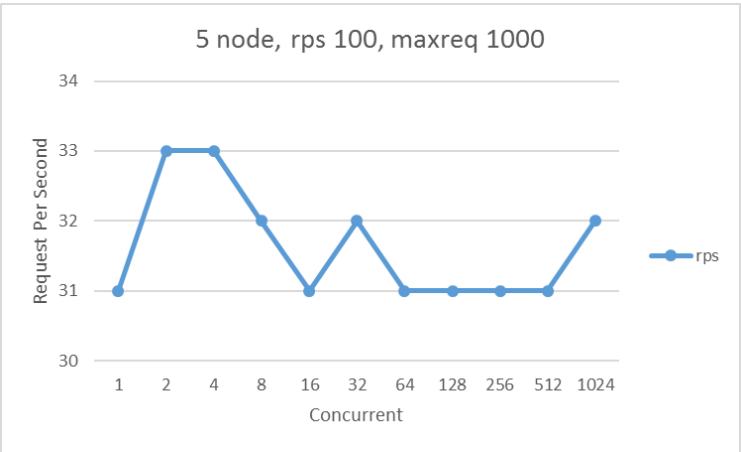
Gambar 5.5 Grafik Performa 3 node CPU

Sama seperti pembagian 2 node CPU, pada pembagian 3 node CPU rata-rata *request* yang dihasilkan adalah 32. Dari 11 contoh *concurrent* yang dibuat, terdapat 6 contoh *concurrent* yang memiliki *request per second*nya 32, selain itu 4 *concurrent* memiliki 33 *request*, dan 1 *concurrent* memiliki 34 *request*. Hasil tersebut dapat dilihat pada Gambar 5.6.



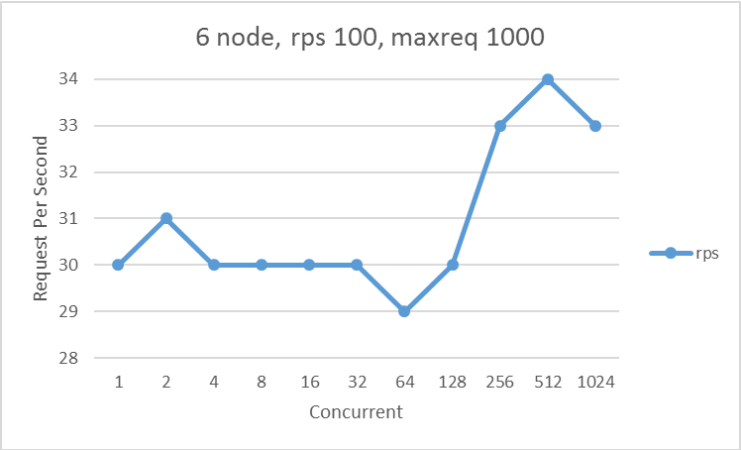
Gambar 5.6 Grafik Performa 4 node CPU

Pada Gambar 5.7 adalah hasil pembagian dari 4 node CPU. Rata-rata *request per second* yang dihasilkan berada pada 31 dan 32 *request* yang sama-sama berjumlah 5 *concurrent*. Sisanya 1 *concurrent* memiliki 33 *request per second*nya.



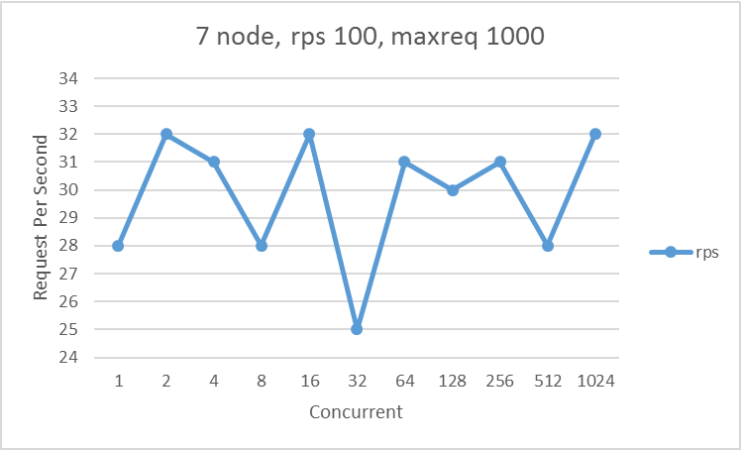
Gambar 5.7 Grafik Performa 5 node CPU

Pada pembagian 5 node CPU masih menghasilkan grafik yang stabil, dimana *request per second*nya diantara 31 hingga 33.



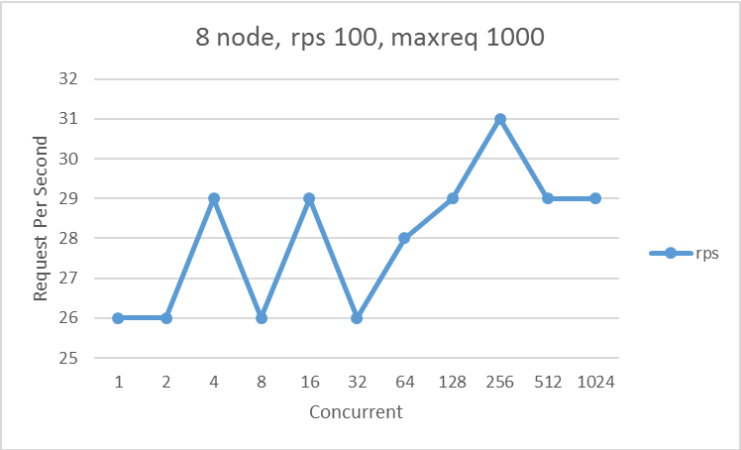
Gambar 5.8 Grafik Performa 6 node CPU

Pada pembagian 6 node CPU terjadi kenaikan *request per second* pada *concurrent* 256, 512, dan 1024 yaitu 33 dan 34 *request*. Dimana sebelumnya pada *concurrent* 128 memiliki 30 *request* dan pada *concurrent* 64 hanya memilik 29 *request*.



Gambar 5.9 Grafik Performa 7 node CPU

Terjadi *request per second* yang tidak stabil pada pembagian 7 node CPU karena mengalami naik turun *request* disetiap *concurrent*. Pada pembagian ini juga mengalami penurunan *request* yang cukup jauh yaitu 25 *request* pada *concurrent* 32. *Request* tertinggi pada pembagian ini adalah 32 *request* ketika berada pada 2, 16, dan 1024 *concurrent*. Hasil tersebut dapat dilihat pada Gambar 5.10.



Gambar 5.10 Grafik Performa 8 node CPU

Pada pembagian 8 node CPU ini juga mengalami ketidakstabilan *request* yang dihasilkan. *Request* yang dihasilkan adalah bervariasi dari 29 *request* hingga 31 *request*.

5.3.2.2 Hasil Uji Performa Algoritma

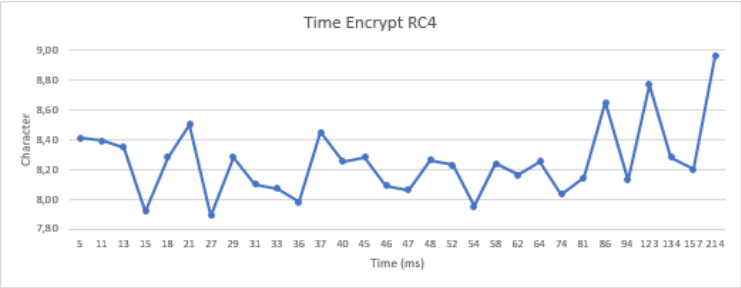
Hasil uji coba performa algoritma adalah menggunakan *library performance.now()* untuk menghitung waktu proses enkripsi dan dekripsi pesan. Dari uji coba yang dilakukan, menghasilkan total waktu proses enkripsi dan dekripsi dalam *millisecond*. Uji coba ini akan menguji 30 contoh teks dengan jumlah karakter yang berbeda-beda namun memiliki kunci yang sama. Dihasilkan rata-rata waktu untuk mengenkripsi adalah 8 ms dan rata-rata waktu untuk proses dekripsi adalah 2 ms. Setelah mendapatkan waktu kecepatan enkripsi dari algoritma *Rivest Cipher* (RC4), maka akan dibandingkan dengan algoritma *Corrected Block Tiny Encryption Algorithm* (XXTEA). Hasil dari uji performa algoritma ini dapat dilihat pada Tabel 5.3.

Tabel 5.3 Hasil Uji Performa Algoritma

No	Karakter	Time Encrypt RC4	Time Decrypt RC4
1	5	8.40747 ms	2.51197 ms
2	11	8.38147 ms	2.67606 ms
3	13	8.35109 ms	2.58926 ms
4	15	7.92218 ms	2.63236 ms
5	18	8.27742 ms	2.61536 ms
6	21	8.50298 ms	2.76569 ms
7	27	7.88963 ms	2.60538 ms
8	29	8.28200 ms	2.66430 ms
9	31	8.10235 ms	2.61995 ms
10	33	8.06621 ms	2.65722 ms
11	36	7.98189 ms	2.63265 ms
12	37	8.45471 ms	2.65027 ms
13	40	8.24966 ms	3.08412 ms
14	45	8.28208 ms	2.64809 ms
15	46	8.09472 ms	2.61874 ms
16	47	8.06404 ms	2.77421 ms
17	48	8.25855 ms	2.62471 ms
18	52	8.23498 ms	2.63671 ms
19	54	7.95399 ms	2.63892 ms
20	58	8.23871 ms	2.62371 ms
21	62	8.15805 ms	2.63001 ms

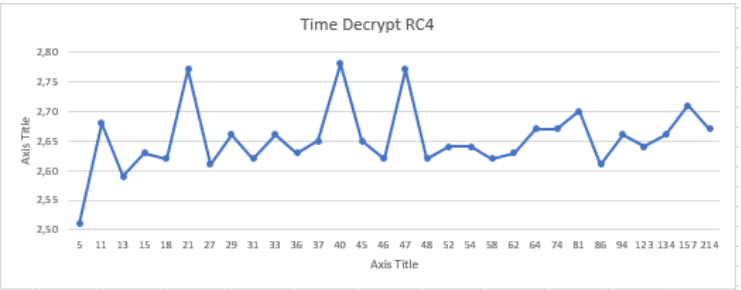
22	64	8.24924 ms	2.66788 ms
23	74	8.02625 ms	2.66944 ms
24	81	8.13507 ms	2.69848 ms
25	86	8.65260 ms	2.61490 ms
26	94	8.12652 ms	2.66405 ms
27	123	8.77163 ms	2.64304 ms
28	134	8.28191 ms	2.66011 ms
29	157	8.20139 ms	2.71184 ms
30	214	8.95801 ms	2.67346 ms

Dari hasil uji performa dua algoritma tersebut, dapat disimpulkan bahwa waktu enkripsi pada algoritma *Corrected Block Tiny Encryption Algorithm* (XXTEA) lebih cepat daripada waktu enkripsi pada algoritma *Rivest Cipher* (RC4). Rata-rata waktu yang dihasilkan algoritma *Rivest Cipher* (RC4) adalah sekitar 7 sampai 8 ms, sedangkan rata-rata waktu yang digunakan *Corrected Block Tiny Encryption Algorithm* (XXTEA) untuk mengenkripsi adalah hanya sekitar 1 sampai 2 ms. Selain itu, rata-rata waktu yang digunakan *Rivest Cipher* (RC4) untuk mendekripsi adalah sekitar 2 sampai 3 ms, sedangkan untuk algoritma *Corrected Block Tiny Encryption Algorithm* (XXTEA) sekitar 1 samapi 2 ms. Hasil grafik dari tabel 5.3 dapat dilihat pada Gambar 5.12 sampai 5.15.



Gambar 5.11 Grafik Hasil Waktu Enkripsi RC4

Pada grafik hasil waktu untuk mengenkripsi menggunakan algoritma *Rivest Cipher* (RC4), didapatkan rata-rata waktu yang digunakan untuk mengenkripsi pesan adalah antara 7.8 hingga 9 ms. Waktu terlama adalah ketika melakukan enkripsi pada karakter paling banyak yaitu 214 karakter.



Gambar 5.12 Grafik Hasil Waktu Dekripsi RC4

Dari grafik perhitungan waktu mendekripsi menggunakan algoritma *Rivest Cipher* (RC4), didapatkan rata-rata waktu yang digunakan untuk mendekripsi pesan lebih cepat daripada proses enkripsinya, yaitu antara 2,5 hingga 2,8 ms. Pada karakter paling sedikit didapatkan waktu dekripsi paling cepat.

5.3.3 Evaluasi

Berdasarkan uji fungsionalitas yang dilakukan, sistem dapat berjalan sesuai dengan diagram kasus penggunaan yang dirumuskan pada bab 3. Paket pesan yang terkirim tidak dapat di deteksi oleh Wireshark yang ditangani dengan protokol HTTP, sehingga paket data yang terkirim dan terdeteksi oleh Wireshark adalah pesan yang sudah terenkripsi. Kemdian pada uji performa, sistem yang berjalan rata-rata mampu menangani 34 *request* jika terdapat banyak pengguna yang secara bersamaan ketika sedang mengakses sistem dan dapat menampung hingga 1024 pengguna yang mengakses secara bersamaan. Dari hasil uji algoritma *Rivest Cipher* (RC4) juga dapat disimpulkan bahwa rata-rata waktu untuk melakukan proses enkripsi adalah 8 ms dan rata-rata waktu untuk melakukan dekripsi adalah 2 ms.

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas tentang kesimpulan yang didasari oleh hasil uji coba yang telah dilakukan pada bab sebelumnya. Kesimpulan nantinya sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut di masa depan.

6.1 Kesimpulan

Dalam pengerjaan Tugas Akhir ini melalui tahap perancangan aplikasi, implementasi algoritma, serta uji coba. Sehingga mendapatkan kesimpulan sebagai berikut:

1. Komputer *web server* dapat menangani aplikasi surat elektronik hingga 1024 pengguna yang mengakses secara bersamaan, dengan *request per second*nya mencapai 34 *request*.
2. Rata-rata waktu yang dibutuhkan algoritma *Rivest Cipher* (RC4) untuk mengenkripsi pesan adalah sekitar 8 ms dan untuk mendekripsi pesan adalah sekitar 2 ms.
3. Hasil uji coba fungsionalitas pada kegiatan yang dilakukan dalam aplikasi surat elektronik berjalan sesuai harapan.

6.2 Saran

Saran yang diberikan untuk pengembangan aplikasi ini adalah:

1. Menambahkan modul *email* agar alamat email yang digunakan adalah alamat email sebenarnya.

2. Membandingkan aplikasi surat elektronik yang sudah dibuat dengan *platform* email lainnya, misal dengan protokol SMTP, PGP, dsb.
3. Menambahkan fitur notifikasi pada aplikasi surat elektronik agar pengguna dapat mengetahui jika ada pesan yang masuk.

DAFTAR PUSTAKA

[1] Mousa, Allam and Hamad Ahmad, "Evaluation of the RC4 Algorithm for Data Encryption," *International Journal of Computer Science and Applications,* Vol. 3, No. 2, June 2006.

[2] Mooduto, Hanriyawan A., "Enkripsi Data Menggunakan RC4," *Jurnal Ilmiah R & B,* Volume 4, No. 2, Oktober 2004.

[3] Suryani, Karina Novita, "Algoritma RC4 Sebagai Metode Enkripsi," *Makalah IF2091, Struktur Diskrit,* 2009.

[4] Safrina, Rika, "Studi dan perbandingan Sistem Penyandian Pesan dengan Algoritma RC2, RC4, RC5, RC6," *Studi dan Perbandingan Sistem Penyandian Pesan dengan Algoritma RC2, RC4, RC5 dan RC6*, 2006.

[5] Mulyani, "Analisa Enkripsi dan Dekripsi dengan Metode RC4 Menggunakan Bahasa Pemrograman PHP," [Online]. Available: https://www.academia.edu/22961376/ANALISA_ENKRIPSI_DAN_DESKRIPSI_DENGAN_METODE_RC4_MENGGUNAKAN_BAHASA_PEMROGRAMAN_PHP, [Accessed: 22 Mei 2017].

[6] Bastari, A. Thoriq Abrowi, "Analisis Perbandingan *Stream Cipher* RC4 dan SEAL," *Makalah IF3058*, 2010/2011.

[7] Rahman, Muhamad Aminudin, Imam Kuswardayan, Ridho Rahman Hariadi, "Perancangan dan Implementasi RESTful Web untuk Game Sosial Food Merchant Saga pada Perangkat Android," *Jurnal Teknik Pomits*, Vol. 2, No. 1, 2013.

[8] Kurniawan, Erick, "Implementasi REST Web Service untuk Sales Order dan Sales Tracking Berbasis Mobile," *Jurnal Eksis*, Vol. 7, No. 1, Mei 2014.

[9] A, Rizky Dewa, "Penjelasan Tentang Web API, Web Services, SOAP, REST," [Online]. Available:

https://www.academia.edu/23707779/Pengertian_and_penjelasan_Singkat_WEB_servicers_Web_API_, [Accessed: 23 Mei 2017].

[10] Kumari, Smita, " Performance comparison of SOAP and REST based Web Services for Enterprise Application Integration," *IEEE*, 2015.

[11] Mincey, Maria, "Node.js Advantages for Real-time Web Apps," [Online]. Available: https://artdriver.com/blog/node-js-advantages-for-building-real-time-web-apps , [Accessed: 23 Mei 2017].

[12] Sutrisno, Nugroho Adi, "Perbandingan Node JS vs Ruby on Rails," [Online], Available: https://www.codepolitan.com/perbandingan-nodejs-vs-ruby-on-rails-58b7dfbc8c83a, [Accessed: 23 Mei 2017].

[13] Arslan, Muhammad, "Memulai Pembuatan Aplikasi Web dengan Express.js," [Online]. Available: https://www.codepolitan.com/memulai-pembuatan-aplikasi-web-dengan-express-js-1-instalasi-dan-pengenalan, [Accessed: 5 Mei 2017].

[14] Ritonga, Pahmi, "Pengertian MySQL Menurut Para Pakar," [Online]. Available: http://www.bangpahmi.com/2015/03/pengertian-mysql-menurut-para-fakar.html, [Accessed: 23 Mei 2017].

[15] Supriyadi, Arif, "Makalah SQL," [Online]. Available: https://www.academia.edu/12377987/Makalah_SQL, [Accessed: 23 Mei 2017].

LAMPIRAN

A. Data yang Digunakan Untuk Skenario Uji Coba

1 node, rps 100, maxreq 1000									
conc	totalReq	totalErr	totalTime	rps	meanLatency	maxLatency	minLatency	percy	error
1	1000	0	50.20	20	20355.3	39698	630	0.263	error null
2	1000	0	48.64	21	19606.6	38327	648	0.269	error null
4	1000	0	50.00	20	20110.9	39111	628	0.274	error null
8	1000	0	49.75	20	20057.6	39212	642	0.299	error null
16	1000	0	49.53	20	19821.7	38897	620	0.272	error null
32	1000	0	48.78	20	19745.1	38504	624	0.294	error null
64	1000	0	48.89	20	19230.1	38689	111	0.265	error null
128	1000	0	49.36	20	19753.2	38617	345	0.267	error null
256	1000	0	51.06	20	20657.9	40302	643	0.272	error null
512	1000	0	48.57	21	19622.8	38428	626	0.277	error null
1024	1000	0	50.27	20	20248	39764	625	0.280	error null

2 node, rps 100, maxreq 1000									
conc	**totalReq**	**totalErr**	**totalTime**	**rps**	**meanLatency**	**maxLatency**	**minLatency**	**percy**	**error**
1	1000	0	31.28	32	10586.1	20317	760	0.410	error null
2	1000	0	31.37	32	10765.2	21367	685	0.382	error null
4	1000	0	31.93	31	10787.4	20896	767	0.408	error null
8	1000	0	31.28	32	10947.8	21018	745	0.380	error null
16	1000	0	31.26	32	10608.3	20655	709	0.768	error null
32	1000	0	31.18	32	10647.7	20783	676	0.458	error null
64	1000	0	33.66	30	11119.1	23184	711	0.357	error null
128	1000	0	31.36	32	10644.8	20778	696	0.360	error null
256	1000	0	31.19	32	10649.4	20520	700	1.232	error null
512	1000	0	30.50	33	10598.4	20544	705	0.470	error null
1024	1000	0	31.45	32	10731.3	20672	695	0.344	error null

3 node, rps 100, maxreq 1000									
conc	**totalReq**	**totalErr**	**totalTime**	**rps**	**meanLatency**	**maxLatency**	**minLatency**	**percy**	**error**
1	1000	0	30.78	32	10588.2	20139	930	0.465	error null
2	1000	0	30.85	32	10695.6	20304	754	0.386	error null
4	1000	0	30.31	33	10612.1	20420	826	0.452	error null
8	1000	0	30.79	32	10739.6	20776	847	0.489	error null
16	1000	0	30.25	33	10683.4	20183	800	0.474	error null
32	1000	0	30.64	33	10741.2	20122	898	0.447	error null
64	1000	0	30.92	32	10604.8	20379	938	0.641	error null
128	1000	0	29.46	34	10010.9	19280	817	0.488	error null
256	1000	0	30.50	33	10549.1	20729	783	0.511	error null
512	1000	0	31.05	32	10702.1	20567	883	0.511	error null
1024	1000	0	30.94	32	10652.7	20527	830	1.457	error null

4 node, rps 100, maxreq 1000									
conc	**totalReq**	**totalErr**	**totalTime**	**rps**	**meanLatency**	**maxLatency**	**minLatency**	**percy**	**error**
1	1000	0	31.89	31	11556.8	21345	1094	0.354	error null
2	1000	0	32.22	31	11412	21918	1123	3.420	error null
4	1000	0	32.19	31	11530.4	21869	1092	0.383	error null
8	1000	0	30.99	32	11246	21017	1181	0.376	error null
16	1000	0	30.82	32	10958.5	20841	1079	1.068	error null
32	1000	0	32.01	31	11359.6	21333	1152	0.399	error null
64	1000	0	30.81	32	10957.8	21216	1066	4.935	error null
128	1000	0	31.56	32	11252.8	21678	1068	4.454	error null
256	1000	0	31.64	32	11164.8	20860	1082	0.384	error null
512	1000	0	30.36	33	10503.6	19933	1074	0.368	error null
1024	1000	0	31.90	31	11401.6	21071	1047	0.491	error null

5 node, rps 100, maxreq 1000									
conc	**totalReq**	**totalErr**	**totalTime**	**rps**	**meanLatency**	**maxLatency**	**minLatency**	**percy**	**error**
1	1000	0	32.40	31	11812.2	22751	1320	0.391	error null
2	1000	0	30.62	33	9634.9	21147	156	0.506	error null
4	1000	0	30.37	33	9933.5	20213	165	0.357	error null
8	1000	0	31.34	32	11646.2	21632	1239	0.388	error null
16	1000	0	32.04	31	11720.7	22365	1338	5.305	error null
32	1000	0	31.13	32	10838.1	20452	1342	0.368	error null
64	1000	0	32.50	31	11894.4	22332	1314	0.366	error null
128	1000	0	32.33	31	11898.5	22457	1429	1.162	error null
256	1000	0	31.78	31	11843.5	22125	1331	0.559	error null
512	1000	0	32.39	31	11688.2	21953	1181	0.452	error null
1024	1000	0	31.28	32	11142.4	20972	1224	0.392	error null

6 node, rps 100, maxreq 1000									
conc	**totalReq**	**totalErr**	**totalTime**	**rps**	**meanLatency**	**maxLatency**	**minLatency**	**percy**	**error**
1	1000	0	33.88	30	12502.9	23323	1486	0.377	error null
2	1000	0	32.63	31	11836.5	22782	1577	1.488	error null
4	1000	0	33.44	30	12464.7	23043	1478	0.396	error null
8	1000	0	33.24	30	12358.2	23241	1391	3.462	error null
16	1000	0	33.64	30	12432.4	23304	1381	4.217	error null
32	1000	0	33.43	30	12404.6	23421	1567	0.522	error null
64	1000	0	33.98	29	12457.2	23660	1469	2.548	error null
128	1000	0	32.80	30	12022.9	22899	1317	0.481	error null
256	1000	0	29.90	33	9304.6	19610	138	5.035	error null
512	1000	0	29.03	34	9265.3	19002	139	0.493	error null
1024	1000	0	30.51	33	9656.2	20226	163	0.405	error null

7 node, rps 100, maxreq 1000									
conc	**totalReq**	**totalErr**	**totalTime**	**rps**	**meanLatency**	**maxLatency**	**minLatency**	**percy**	**error**
1	1000	0	35.27	28	13242.5	24703	1799	10.765	error null
2	1000	0	31.23	32	9466.8	21436	153	0.458	error null
4	1000	0	32.08	31	9428	22274	149	10.797	error null
8	1000	0	36.19	28	13090.3	26060	1700	0.510	error null
16	1000	0	31.01	32	9699.7	20912	260	0.890	error null
32	1000	0	39.88	25	14743.3	29667	895	10.099	error null
64	1000	0	32.15	31	9968.4	22079	155	2.426	error null
128	1000	0	33.13	30	10214.8	23104	166	3.422	error null
256	1000	0	32.43	31	9900.6	22619	170	0.469	error null
512	1000	0	35.34	28	13290.4	25014	1875	3.391	error null
1024	1000	0	31.15	32	9483.4	22188	105	2.435	error null

8 node, rps 100, maxreq 1000									
conc	**totalReq**	**totalErr**	**totalTime**	**rps**	**meanLatency**	**maxLatency**	**minLatency**	**percy**	**error**
1	1000	0	39.10	26	14551.9	28787	2119	2.200	error null
2	1000	0	39.20	26	14738.2	28491	2138	5.627	error null
4	1000	0	34.38	29	10101..4	23825	158	0.505	error null
8	1000	0	39.20	26	14226.7	28515	2174	0.481	error null
16	1000	0	33.95	29	10175.8	23224	128	10.381	error null
32	1000	0	37.89	26	13650.7	27838	949	7.045	error null
64	1000	0	35.26	28	10696.9	25211	152	0.451	error null
128	1000	0	34.28	29	10436.9	24823	169	7.599	error null
256	1000	0	32.70	31	9911.1	23653	131	0.481	error null
512	1000	0	33.98	29	11034.4	24427	118	13.771	error null
1024	1000	0	34.85	29	10801.9	24470	147	3.138	error null

BIODATA PENULIS

Nindyasari Dewi Utari, lahir di Surakarta, 20 November 1995. Penulis merupakan anak pertama dari dua bersaudara pasangan Bapak Nindyo Pramono dan Ewi Setyaki. Penulis menempuh pendidikan formal dimulai dari Pembina Karanganyar (2000-2002), SD 02 Bejen Karanganyar (2002-2008), SMP Negeri 01 Karanganyar (2008-2010), SMA Negeri 01 Karanganyar (2010-2013) dan S1 Teknik Informatika ITS (2013-2017). Bidang studi yang diambil oleh penulis pada saat berkuliah di Teknik Informatika ITS adalah Arsitektur dan Jaringan Komputer (AJK). Selama masa kuliah, penulis aktif dalam organisasi kemahasiswaan yaitu Himpunan Mahasiswa Teknik Computer-Informatika (2014-2016). Penulis juga aktif dalam berbagai kegiatan kepanitiaan yaitu SCHEMATICS 2014-2015, GERIGI ITS (2014-2015), anggota Paduan Suara Mahasiswa (PSM) ITS (2013), FTIf Journey (2014) serta mengikuti berbagai lomba dalam bidang teknologi maupun diluar teknologi seperti, Lomba *Mobile Developer* di Universitas Binus Jakarta, Lomba *Mobile Developer* di Universitas Gadjah Mada, dan Lomba *Enterpreneurship* di Universitas Bakrie Jakarta. Penulis sempat menjadi asisten dosen selama menjalani perkuliahan di Teknik Informatika yaitu asisten dosen mata kuliah Sistem Operasi, Jaringan Komputer, Jaringan Multimedia, dan Grafika Komputer. Penulis memiliki ketertarikan dalam hal *travelling* dan kuliner. Komunikasi dengan penulis dapat melalui email: **nindyasaridewiutari@gmail.com**