IMPLEMENTASI ALGORITMA KRIPTOGRAFI ELGAMAL DAN ALGORITMA KOMPRESI SHANON-FANO UNTUK KEAMANAN PENGIRIMAN *EMAIL*

SKRIPSI

oleh:
PRIYO CAHYADI
0410963041-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2011

IMPLEMENTASI ALGORITMA KRIPTOGRAFI ELGAMAL DAN ALGORITMA KOMPRESI SHANON-FANO UNTUK KEAMANAN PENGIRIMAN *EMAIL*

SKRIPSI

SEBAGAI SALAH SATU SYARAT UNTUK MEMPEROLEH GELAR SARJANA ILMU KOMPUTER

oleh:
PRIYO CAHYADI
0410963041-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2011

LEMBAR PENGESAHAN SKRIPSI

IMPLEMENTASI ALGORITMA KRIPTOGRAFI ELGAMAL DAN ALGORITMA KOMPRESI SHANNON FANO UNTUK KEAMANAN PENGIRIMAN *EMAIL*

oleh :

PRIYO CAHYADI
0410963041 – 96

Telah dipertahankan di depan Majelis Penguji
pada tanggal 10 Januari 2011
dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana dalam bidang Ilmu Komputer

Pembimbing I	Pembimbing II
Edy Santoso, S.Si., M.Kom	**Dewi Yanti L, S.Kom., M.Kom**
NIP. 197404142003121004	**NIP. 198111162005012004**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Agus Suryanto, M.Sc
NIP. 196908071994121001

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	**: Priyo Cahyadi**
NIM	**: 0410963041-96**
Program Studi	**: Ilmu Komputer**
Penulis Skripsi Berjudul	**: Implementasi Algoritma Kriptografi Elgamal Dan Algoritma Kompresi Shannon Fano Untuk Keamanan Pengiriman *Email***

Dengan ini menyatakan bahwa :

1. **Isi dari Skripsi yang saya buat adalah benar – benar karya sendiri dan tidak menjiplak karya orang lain, selain nama – nama yang termaktub di isi dan tertulis di daftar pustaka dalam Skripsi ini.**
2. **Apabila dikemudian hari ternyata Skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 10 Januari 2011
Yang menyatakan,

(Priyo Cahyadi)
NIM. 0410963041

IMPLEMENTASI ALGORITMA KRIPTOGRAFI ELGAMAL DAN ALGORITMA KOMPRESI SHANON-FANO UNTUK KEAMANAN PENGIRIMAN *EMAIL*

ABSTRAK

Seiring dengan bertambahnya pengguna teknologi informasi juga memunculkan tindakan kejahatan dalam dunia teknologi informasi, salah satunya penyadapan *email.* Penyadapan *email* dapat menyebabkan terbongkarnya suatu rahasia atau penyalahgunaan isi dari *email*. Perlindungan terhadap *email* dapat dilakukan dengan menggunakan kriptografi.

Salah satu algoritma kriptografi yang cukup kuat adalah algoritma kriptografi kunci publik Elgamal yang mana enkripsi pesan dilakukan dengan menggunakan metode modulo. Algoritma ini menghasilkan *chipertext* yang ukurannya dua kali lipat dari *plaintext* karena untuk setiap karakter *plaintext* dienkripsi menjadi dua karakter. Untuk itu diperlukan kompresi memperkecil ukuran *file*. Salah satu algoritma kompresi yang populer adalah algoritma kompresi Shannon Fano. Algoritma kompresi Shannon Fano mengkodekan karakter dengan frekuensi terbanyak dengan rangkaian bit yang pendek, dan karakter yang frekuensi sedikit dengan rangkaian bit yang lebih panjang.

Berdasarkan pengujian terhadap algoritma kriptografi Elgamal dengan menggunakan sepuluh *text* uji dihasilkan waktu rata-rata dekripsi dan dekompresi satu kunci untuk serangan *bruteforce* terhadap kunci *private* dengan kunci publik belun diketahui adalah 6.2 detik. Waktu rata-rata dekripsi dan dekompresi dua kunci untuk serangan *bruteforce* dengan kunci publik diketahui adalah 12.39 detik. Berdasarkan pengujian yang telah dilakukan pada algoritma kompresi Shannon Fano terhadap 14 *text* uji didapatkan rata - rata rasio kompresi 13% dengan rentang ukuran file uji 606 bytes – 4210 bytes.

IMPLEMENTATION OF ELGAMAL CRYPTOGRAPHIC ALGORITHM AND SAHANNON FANO COMPRESSION ALGORITHM FOR DELIVERY OF EMAIL SECURITY

ABSTRACT

Along with the increase of users in information technology also raises the crime in information technology, one of that is intercepts email. Interception of email can break a secret or misuse of the contents of the email that caused harm to a particular party. Protection against email can be done by using cryptography.

One of the cryptographic algorithm which strong enough is ElGamal cryptographic algorithm. Elgamal cryptographic algorithm is public key algorithm which encrypts the message character using modulo method. This algorithm produce ciphertext whose size doubled from plaintext because for each character will be encrypted into two characters. For that, compression is needed to reduce the file size. One of the popular compression algorithm is Shannon Fano compression algorithm. Shannon Fano algorithm encode character with the highest frequency by a series of short bits, and characters that have slightly frequency encode by longer sequence of bits.

Based on tests of ElGamal cryptography algorithm by using ten test generated text produce result the average time decryption and decompression of the keys to bruteforce attack against the private key with a public key known belun is 6.2 seconds. The average time decryption and decompression are two keys to bruteforce attacks with a public key known is 12:39 seconds. Based on the testing that has been done on Shannon Fano compression algorithm on 14 test text obtained average compression ratio of 13% with a range of test file size 606 bytes - 4210 bytes.

Kata Pengantar

Puji syukur penulis panjatkan kehadirat *Allah SWT* atas segala limpahan kasih sayang dan rahmat-Nya sehingga penulis dapat menyelesaikan laporan penelitian berjudul "Implementasi Algoritma Kriptografi Elgamal Dan Algoritma Kompresi Shannon Fano Untuk Keamanan Pengiriman *Email*" ini dapat diselesaikan. Skripsi ini disusun dan diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana Strata satu (S-1) Program Studi Ilmu Komputer, jurusan Matematika, Fakultas Matematika dan Ilmu pengetahuan Alam, Unversitas Brawijaya, Malang.

Penulis menyadari bahwa laporan penelitian ini tidak dapat terealisasikan tanpa bantuan dari berbagai pihak, untuk itu penulis menyampaikan ucapan terima kasih kepada:

1. Edy Santoso, S.Si., M.Kom., dan Dewi Yanti Liliana S.Kom., M.Kom selaku selaku pembimbing. Terima kasih atas semua saran, bantuan, waktu dan bimbingannya.
2. Mardji, Dr., MT., selaku Ketua Program Studi Ilmu Komputer.
3. Bondan Sapta Prakoso, ST., selaku Penasihat Akademik.
4. Drs. Agus Suryanto, MSc., selaku Ketua Jurusan Matematika.
5. Segenap bapak dan ibu dosen yang telah memberikan ilmunya kepada penulis.
6. Segenap staf dan karyawan di Jurusan Matematika FMIPA Universitas Brawijaya.
7. Bapak, Ibu dan adik ku, terima kasih atas doa, dukungan dan semangat yang tiada henti.
8. Muhammad Syaiful Rizal yang telah membantu proses pembuatan program.
9. Mas Alfa yang telah membantu proses pembuatan program.
10. Chusnul Chotimah yang telah banyak membantu dalam penulisan skripsi.
11. Teman-teman Embong Anyar 2 L1, terimakasih atas segala dukungannya.
12. Sahabat-sahabat ilkomers `04, terima kasih atas semua dukungan yang telah diberikan.
13. Pihak lain yang telah membantu terselesaikannnya Tugas Akhir ini yang tidak bisa penulis sebutkan satu-persatu.

Semoga penulisan laporan tugas akhir ini bermanfaat bagi pembaca sekalian. Penulis menyadari bahwa Skripsi ini masih jauh dari kesempurnaan, dan mengandung banyak kekurangan, sehingga dengan segala kerendahan hati penulis mengharapkan kritik dan saran yang membangun dari pembaca.

Malang,10 Januari 2011

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Halaman

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi yang sangat pesat dewasa ini membawa budaya baru dalam kehidupan. Media fisik dalam sistem informasi mulai berganti ke media digital. Seperti kita ketahui budaya berkirim surat telah berganti menjadi surat elektronik (*email*) yang menggunakan jaringan internet ataupun SMS (*Short Message Service*) yang mengunakan jaringan telepon seluler. Pemanfaatan teknologi semakin mengefisienkan pengiriman data dari segi waktu dan biaya.

Seiring dengan bertambahnya pengguna teknologi informasi juga memunculkan tindakan kejahatan dalam dunia teknologi informasi, salah satunya adalah penyadapan media teknologi informasi (Budi Raharjo, 2005). Penyadapan media teknologi informasi sangat meresahkan bagi penggunanya, salah satunya adalah penyadapan *email*. Penyadapan *email* dapat menyebabkan terbongkarnya suatu rahasia ataupun penyalah gunaan isi dari *email* tersebut yang menyebabkan kerugian bagi pihak tertentu.

Seiring bertambahnya tindakan kejahatan dalam teknologi informasi diperlukan tindakan untuk melindungi teknologi informasi itu sendiri agar tidak sampai di tangan pihak yang tidak berhak, salah satunya adalah kriptografi. Dengan menggunakan kriptografi data akan menjadi lebih aman karena isi data akan disembunyikan dengan menggunakan sebuah kunci dan isi data dapat dibaca kembali dengan menggunakan kunci tersebut, metode ini disebut kriptografi klasik atau kriptografi simetri.

Masalah akan kembali muncul apabila penyadap berhasil mendapatkan kunci kriptografi tersebut dan isi data dapat diketahui karena penyadap memiliki kunci untuk mengembalikan isi data ke bentuk semula.

Berdasarkan kelemahan yang dimiliki oleh kriptografi simetri maka dikembangkan kriptografi asimetri atau disebut juga kriptografi kunci *public*. Keunggulan kriptografi kunci *public* adalah memungkinkan kunci enkripsi pesan diketahui oleh siapapun karena kunci enkripsi tidak digunakan dalam proses dekripsi pesan (Rinaldi

Munir, 2006). Kunci dekripsi pesan hanya diketahui oleh penerima *email*. Apabila penyadap berhasil mendapatkan enkripsi *email* dan kunci enkripsinya, penyadap tak dapat membaca *email* tersebut karena kunci enkripsi tidak dapat digunakan untuk mendekripsi pesan.

Salah satu algoritma kriptografi kunci *public* adalah algoritma kriptografi Elgamal. Kekuatan algoritma ini adalah logaritma diskrit pada proses enkripsi kunci dekripsi menjadi kunci enkripsi dan juga kriptografi isi datanya karena menggnakan proses modulo. Akan tetapi algoritma Elgamal ini memiliki kelemahan karena ukuran data akan menjadi dua kali lipat (Rinaldi Munir, 2006). Kelemahan ini dapat diminimalkan dengan menggunakan proses kompresi *text*. Salah satu algoritma kompresi *text* yang cukup baik dan populer saat ini adalah algoritma kompresi *text* Shannon-Fano (Romi Wiryadinata, 2007).

Berdasarkan latar belakang yang telah dikemukakan, judul yang diangkat dalam skripsi ini adalah "Implementasi Algoritma Kriptografi Elgamal dan Algoritma Kompresi Shannon-Fano Untuk Keamanan Pengiriman *Email*".

1.2 Rumusan Masalah

Berdasarkan uraian pada latar belakang, maka dapat dirumuskan beberapa permasalahan, antara lain:

1. Bagaimana mengimplementasikan algoritma kompresi Shanon Fano pada *plaintext* sehingga menghasilkan *file* yang memilki ukuran lebih kecil dan *file* yang dihasilkan dapat dilakukan proses dekompresi.
2. Bagaimana mengimplementasikan algoritma kriptografi Elgamal pada pesan (*plaintext* terkompresi) dengan kunci *public* sehingga dihasilkan *ciphertext* yang dapat dilakukan proses dekripsi.
3. Bagaimana membuat aplikasi *email client* yang dapat mengirim *email* yang sudah dikompresi,dienkripsi dan dapat melakukan proses dekripsi dan dekompresi .

1.3 Batasan Masalah

Batasan masalah pada penelitian ini :

1. Data yang dikompresi, dienkripsi, dekripsi dan didekompresi berupa *text*
2. *Email client* hanya dapat mengirim dan membaca pesan.
3. Menggunakan karakter ASCII 7 bit (127 karakter).

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah sebagai berikut :

1. Mengimplementasikan algoritma kompresi Shanon Fano pada *plaintext* sehingga menghasilkan *file* yang memilki ukuran lebih kecil dan *file* yang dihasilkan dapat dilakukan proses dekompresi.

2. Mengimplementasikan algoritma kriptografi Elgamal pada pesan (*plaintext* terkompresi) dengan kunci *public* sehingga dihasilkan *ciphertext* yang dapat dilakukan proses dekripsi.

3. Membuat aplikasi *email client* yang dapat mengirim *email* yang sudah dikompresi,dienkripsi dan dapat melakukan proses dekripsi dan dekompresi .

1.5 Manfaat Penelitian

Manfaat yang dapat diambil dari pelaksanaan penelitian ini adalah:

1. Memanfaatkan perkembangan ilmu pengetahuan dan teknologi pada bidang kriptografi yang bertajuk pada pembangunan sebuah *email client* berbasis *desktop* untuk mengirimkan pesan (*text*) yang sudah dikompresi dan dikriptografi, sehingga dengan hadirnya tugas akhir ini akan menambah referensi tentang algoritma kompresi khususnya algoritma kompresi Shanon-Fano dan algoritma kriptografi khususnya algoritma kriptografi Elgamal.

2. Akan memberikan sebuah konstribusi dengan mengamankan informasi dari *text* yang dikirim dan pengurangan ukuran *text* yang dikirim. Dengan kriptografi, *text* yang didapatkan *attacker* menjadi tidak berarti dan memakan waktu sangat lama untuk memecahkannya. Dan dengan kompresi maka ukuran *text* yang dikirimkan akan lebih sedikit, sehingga memudahkan untuk mendistribusikannya.

1.6 Sistematika Penulisan

Untuk memberikan gambaran tentang tugas akhir, berikut disajikan garis besar pembahasan dari keseluruhan isi laporan tugas akhir untuk setiap bab.

BAB I : PENDAHULUAN

Bab ini berisi latar belakang penulisan, permasalahan yang ada, batasan masalah, tujuan dan manfaat serta sistematika penulisan skripsi.

BAB II : TINJAUAN PUSTAKA

Bab ini berisi penjelasan singkat tentang teori yang melandasi algoritma kriptografi Elgamal dan algoritma kompresi Shanon-Fano.

BAB III : METODELOGI DAN PERANCANGAN

Bab ini berisi metode-metode yang digunakan dalam menyelesaikan masalah perancangan sistem untuk algoritma kriptografi Elgamal dan algoritma kompresi Shanon-Fano.

BAB IV : HASIL DAN PEMBAHASAN

Bab ini berisi tentang penjelasan implementasi sistem dan hasil pengujian yang dilakukan.

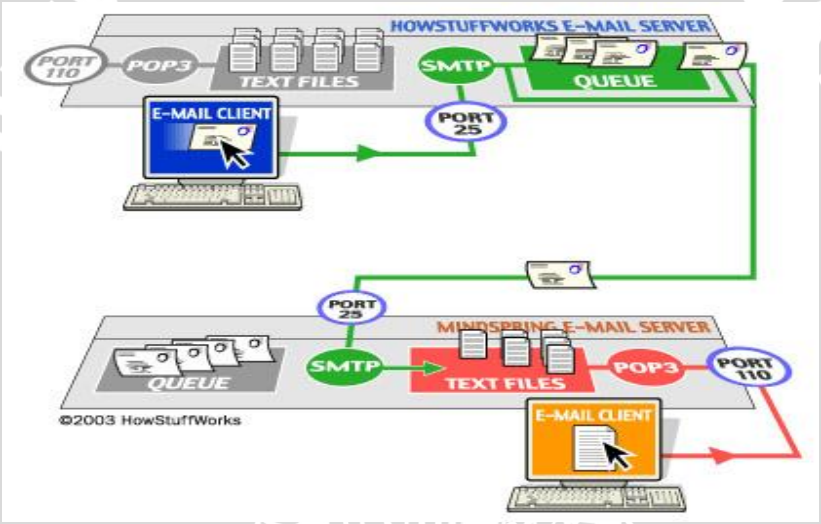
BAB V : PENUTUP

Bab ini berisi kesimpulan yang diperoleh dari hasil pengujian dan saran-saran untuk pengembangan lebih lanjut.

BAB II
TINJAUAN PUSTAKA

2.1. *Email*

Elektronik mail atau biasa disingkat sebagai *email*, merupakan sebuah metode untuk mengirimkan pesan dalam bentuk digital. Pesan ini biasanya dikirimkan melalui medium internet. Sebuah pesan elektronis terdiri dari isi, alamat pengirim, dan alamat-alamat yang dituju.Skema pengiriman *email* dapat dilihat pada **Gambar 2.1** Skema Pengiriman *Email*.



Gambar 2.1 Skema Pengiriman *Email*

SMTP (*Simple Mail Transfer Protocol*). Protokol ini digunakan untuk pengiriman *email* dengan menggunakan aplikasi *email client* (*outlook express, eudora, thunderbird*), dengan identitas **port 25**.

POP (*Post Office Protocol*) lebih populer dengan sebutan ***POP3***. Protokol ini digunakan oleh aplikasi *email client* untuk men '*download*' *email* dari *mail server*. Cara kerja protokol ini adalah men'*download' email*, menghapus *email* dari *mail server* jadi proses

pembacaan *email* pada aplikasi *email client* dilakukan secara *offline*. identitas protokol ini adalah **port 110**.(Brain Marshall , 2009).

2.2 ASCII

American Standard Code for Information Interchange (ASCII) adalah format yang banyak digunakan untuk *file text* di dalam dunia komputer dan internet. Di dalam *file* ASCII, masing-masing *alphabetic, numeric*, atau karakter khusus seperti *Return, Tab Control* dan sebagainya. Dengan adanya standart ini, membuatkan anda mudah melakukan pertukaran informasi antar berbagai peralatan yang berbeda, antar *operating system* yang berlainan, bahkan komputer yang berbeda.

Ada dua jenis kode yang sering digunakan, yaitu ASCII- 8 bit dan ASCII- 7 bit. Kode yang terdapat pada ASCII-8 bit jauh lebih lengkap dari ASCII-7 bit. Menurut Tino ASCII-7 bit mempunyai kombinasi kode $2^7 = 128$, dengan ketentuan sebagai berikut:

1. 26 kode untuk huruf kapital *(upper case)* dari A –Z.
2. 26 kode untuk huruf kecil *(lower case)* dari a –z.
3. 10 digit desimal dari 0 –9.
4. 34 karakter kontrol untuk informasi status operasi komputer.
5. 32 karakter khusus *(special characters).*

Kode ASCII dengan nilai kode 0 sampai dengan 31 dan 127 termasuk dalam status karakter-karakter kontrol yang tidak dapat dicetak (*non-printable characters*). Contohnya, tabel ASCII karakter 28 mewakili fungsi "*file separator*", atau karakter 8 yang mewakili *backspace*.

Kode ke 32 adalah karakter *spasi*, menandakan pemisah antara kata, yang dihasilkan oleh *space-bar* dari *keyboard*. Kode 33 sampai 126 dikenal sebagai karakter-karakter yang dapat di cetak (*printable characters*), terdiri dari beberapa huruf, angka, tanda baca, dan beberapa simbol. (Jack Febrian, 2008).

2.3. Kriptografi

Kriptografi berasal dari bahasa Yunani yaitu *crypto* (menyembunyikan) dan *graphy* (tulisan). Kriptografi adalah sebuah ilmu yang mempelajari teknik-teknik matematika yang berhubungan

dengan aspek keamanan informasi, seperti kerahasiaan data, keabsahan data, dan autentifikasi data (Menezes, Oorschot and Vanstone, 1996).

Tujuan Kriptografi yaitu : (Menezes, Oorschot and Vanstone, 1996).

1. Kerahasiaan, menjaga agar pesan tidak dapat dibaca oleh pihak-pihak yang tidak berhak. Di dalam kriptografi, layanan ini direalisasikan dengan menyandikan pesan menjadi *chipertext*.
2. Integritas Data, menjamin agar pesan masih asli/utuh atau belum pernah dimanipulasi selama pengiriman. Di dalam kriptografi, layanan ini direalisasikan dengan menggunakan tanda tangan digital.
3. Autentikasi, yaitu identifikasi, baik mengidentifikasi kebenaran pihak-pihak yang berkomunikasi (*user authentication atau entity authentication*) maupun mengidentifikasi kebenaran sumber pesan. Di dalam kriptografi, layanan ini direalisasikan dengan menggunakan tanda tangan digital (*digital signature*). Tanda tangan digital menyatakan sumber pesan.
4. *Non repudation* (penyangkalan) adalah layanan untuk mencegah entitas yang berkomunikasi melakukan penyangkalan, yaitu pengirim pesan menyangkal telah mengirimkan pesan. Di dalam kriptografi, layanan ini direalisasikan dengan menggunakan tanda tangan digital (*digital signature*). Tanda tangan digital menyatakan sumber pesan.

Plaintext adalah pesan asli yang dapat dibaca atau pesan yang merupakan masukan dalam algoritma kriptografi. Sedangkan *Ciphertext* adalah pesan yang sudah diacak bardasarkan dari *plaintext* dan kuncinya (William Stallings, 2003)

Enkripsi adalah sebuah penyandian dari *text* yang dapat dibaca (*plaintext*) menjadi kode (*ciphertext*). Sedangkan dekripsi adalah proses merubah kode (*chipertext*) menjadi *text* yang dapat dibaca (*plaintext*). (Rinaldi munir, 2006)

Kunci *public* adalah kunci yang digunakan untuk enkripsi. Sedangkan kunci rahasia adalah kunci yang digunakan untuk dekripsi. (William Stallings, 2003)

Kriptoanalisis (*cryptanalysis*) adalah kebalikan dari kriptografi, yaitu suatu ilmu untuk memecahkan mekanisme kriptografi dengan cara mendapatkan kunci dari *ciphertext* yang digunakan untuk mendapatkan *plaintext*. Kriptologi *(cryptology)* adalah ilmu yang mencakup kriptografi dan kriptoanalisis.

Berdasarkan dari waktu pemakaiannya, kriptografi dibagi menjadi dua, yaitu: (Rinaldi Munir, 2006)

1. Kriptografi klasik
   Kriptografi klasik dibagi menjadi dua, yaitu:
   1. *Cipher* Substitusi (*Subtitutions Chiper*)
      Contoh dari *Chiper* Substitusi adalah *Caesar Cipher*.Pada *Caesar Cipher*, tiap huruf disubstitusi dengan huruf ketiga berikutnya dari susunan alfabet yang sama, dalam hal ini kuncinya adalah jumlah pergeseran huruf (yaitu 3). Susunan alfabet setelah digeser sejauh 3 huruf membentuk sebuah tabel substitusi. Tabel substitusi dapat dilihat pada **Tabel 2.1** Tabel Substitusi.

Tabel 2.1 Substitusi

Plain Teks	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cipher teks	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Dengan menggunakan *Caesar Cipher*, maka pesan:
UNIVERSITAS BRAWIJAYA
Disandikan dengan *Caesar Cipher* menjadi:
XQLYHUVLWDV EUDZLMDBD
Dalam praktek orang biasanya mengelompokkan *ciphertext* menjadi kelompok-kelompok yang terdiri dari beberapa huruf. Tujuannya agar kriptanalis menjadi lebih sukar. Dengan pengelompokan menjadi 4 huruf maka contoh di atas menjadi:
XQLY HUVL WDVE UDZL MDBD

   2. *Cipher* Transposisi (*Transposition Cipher*)
      Pada *cipher* transposisi, huruf-huruf di dalam *plaintext* tetap sama, hanya saja urutannya diubah. Dengan kata lain

algoritma ini melakukan *transpose* terhadap rangkaian karakter di dalam *text*.

Misalkan *plaintext*-nya adalah:
UNIVERSITAS BRAWIJAYA MALANG
Untuk mengenkripsi pesan, *plaintext* ditulis secara horizontal dengan lebar kolom tetap, misal selebar 6 karakter (kunci k=6):

UNIVER
SITASB
RAWIJA
YAMALA
NG
Maka *chipertext*-nya dibaca secara vertikal menjadi :
USRYN NITAAG ITWM VAIA ESJL RBAA

3. Kriptografi Moderen

Berdasarkan dari penggunaan kuncinya, kriptografi dibagi menjadi dua,yaitu: (Rinaldi Munir,2006)

1. Algoritma Simetris

Algoritma simetris adalah algoritma kriptografi yang mana kunci enkripsinya sama dengan kunci dekripsinya. Algoritma ini sangat tergantung pada satu kunci, Pengirim dan penerima pesan harus menyetujui satu kunci terlebih dahulu. Apabila kunci itu bocor kepada pihak yang tak berwenang, maka pesan dapat dekripsi dan di enkripsi oleh pihak yang tak berwenang.

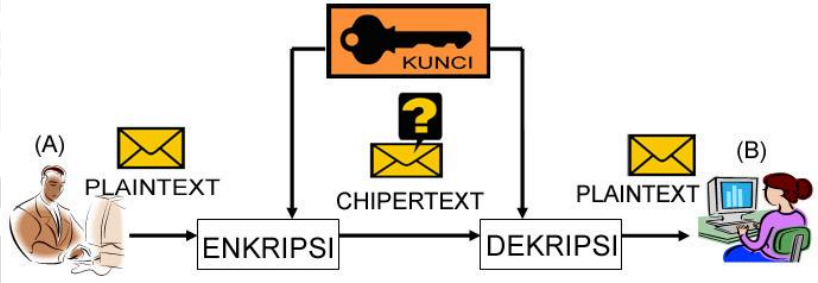
Permasalahan akan muncul apabila pihak yang akan menerima pesan lebih dari satu orang, maka jumlah kunci yang harus disetujui sebanyak kombinasi dari orang yang memberikan kunci. Rumus kombinasi dapat dilihat pada rumus 2.1.

$$C_2^n = \frac{n!}{(n-2)!.2!} = \frac{n.(n-1)}{2} \qquad (2.1)$$

Keterangan :

1. C = Jumlah kombinasi kunci
2. n = Banyak kunci yang diberikan oleh tiap orang

Algoritma simetris dapat digambarkan seperti pada Gambar 2.2

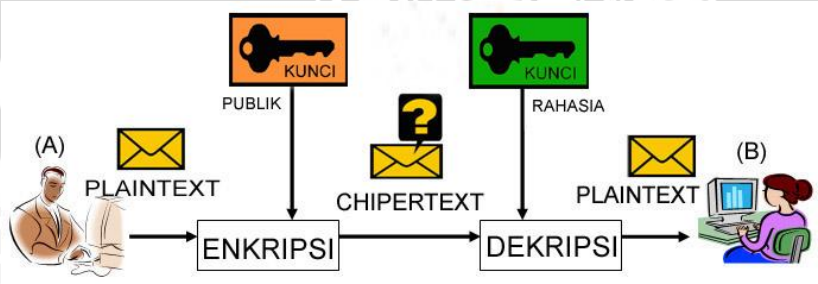


Gambar 2.2 Skema Algoritma Simetris

2. Algoritma Asimetris

Algoritma asimetris biasa disebut algoritma kunci *public*. Pada algoritma ini terdapat dua buah kunci yaitu kunci *public* dan kunci *private*. Kunci *public* digunakan untuk mengenkripsi pesan, sedangkan kunci *private* digunakan untuk mendekripsi pesan. Kunci *public* bersifat umum dan bisa diketahui oleh siapa saja sedangkan kunci *private* hanya diketahui oleh sang penerima pesan dan sistem.

Keuntungan dari algoritma ini adalah menjamin kerahasiaan pesan tanpa melakukan kesepakatan kunci antara pengirim dan penerima. Algoritma asimetris dapat digambarkan seperti pada Gambar 2.3



Gambar 2.3 Skema Algoritma Asimetris

Keterangan:
A = komputer pengirim pesan
B = komputer penerima pesan

Algoritma Asimetris:
1. Penerima B membuat pasangan kunci , yaitu kunci *private* x dan kunci *public* y.
2. Pengirim A dengan kunci *public* y dan pesan Ps, pesan dienkripsi sehingga diperoleh *ciphertext* c = ey (Ps)
3. Penerima B untuk mendekripsi *ciphertext* menggunakan kunci *private* x untuk mendekripsi pesan Ps, sehingga menjadi pesan aslinya. dx [ey(Ps)]=dx(c)=Ps
4. Dengan mengetahui kunci *public* y dan *ciphertext* c, bagi *attacker* akan kesulitan dalam mendapatkan kunci *private*.
5. Dengan mengetahui kunci *public* y dan *ciphertext* c, bagi *attacker* akan kesulitan dalam mengetahui pesan Ps.

2.3.1. Algoritma Kriptografi Elgamal

Algoritma Kriptografi Elgamal merupakan salah satu algoritma kriptografi kunci *public* yang dibuat oleh Taher Elgamal pada tahun 1984. Algoritma in pada umumnya digunakan untuk digital signature, namun kemudian dimodifikasi sehingga juga bisa digunakan untuk enkripsi dan deskripsi. Kekuatan algoritma ini terletak pada sulitnya menghitung logaritma diskrit. (Rinaldi Munir, 2006)

Secara umum algoritma *Elgamal* diterangkan sebagai berikut:
1. Pembentukan kunci *public* dan kunci *private* oleh komputer penerima.
   Kunci *public* : tripel (y, g, p), rumus penghitungan nilai y dapatdilihat pada rumus 2.2.

   $y = g^x \bmod p$ (2.2)

   Kunci *private* : (x)

Keterangan : 1. g (input bilangan acak, dengan syarat g < p) tidak rahasia, sebab diperlukan dalam proses enkripsi
2. p adalah jumlah karakter ASCII yang dipakai dalam enkripsi.
3. x adalah bilangan acak, dengan syarat $1 \leq x \leq p-2$.

2. Konversi tiap karakter *plaintext* ke kode nomor ASCII.
   Karakter *text* yang telah dikonversi ke nomor ASCII-nya dikodekan dengan $m_{(i)}$.

3. Enkripsi Karakter *Text*
   Setiap karakter *text* (m) di enkripsi. Rumus enkripsi dapat dilihat pada rumus 2.3 dan rumus 2.4

$$a = g^k \bmod p \quad (2.3)$$
$$b = y^k \, m \bmod p \quad (2.4)$$

Keterangan : k adalah bilangan acak, dengan syarat $1 \leq k \leq p-2$.
m adalah nomor ASCII dari tiap karakter *plaintext*.

4. Dekrpsi *Ciphertext.*
   Rumus dekripsi *ciphertext* dapat dilihat pada rumus 2.5 :

$$m = b.a^{p-1-x} \bmod p \quad (2.5)$$

Berikut adalah contoh penerapan algoritma kriptografi Elgamal pada *text* : BRAWIJAYA

1. Pembentukan kunci *public* dan kunci *private* oleh komputer penerima. Kunci *private*, x= 7 (bilangan acak) g=13, p=127,
   Kunci *public* , $y = g^x \bmod p$
   $= 13^7 \bmod 127$
   $= 103$

2. Konversi Karakter Pesan ke kode ASCII
   Mengkonfersi karakter *text* ke dalam nomor yang sesuai dalam kode ASCII. Konversi karakter pesan ke kode ASCII dapat dilihat pada **Tabel 2.2** Konversi Karakter Pesan ke kode ASCII.

Tabel 2.2 Konversi Karakter Pesan ke Kode ASCII

No	Karakter	Plainteks (m)	ASCII
1	B	m1	66
2	R	m2	82
3	A	m3	65
4	W	m4	87
5	I	m5	73
6	J	m6	74
7	A	m7	65
8	Y	m8	89
9	A	m9	65

3. Enkripsi Karakter *Text*

Enkripsi Karakter *Text* dapat dilihat pada **Tabel 2.3** Enkripsi Karakter *Text*.

g=13 , y=103 p=127

Tabel 2.3 Enkripsi Karakter *Text*

No.	**"m"**	**"k"**	**$a=g^k \bmod p$**	**$b=y^k m \bmod p$**
1	66	73	104	67
2	82	13	15	73
3	65	5	72	33
4	87	71	69	44
5	73	33	61	64
6	74	7	103	87
7	65	5	72	33
8	89	5	72	53
9	65	31	71	78

4. Dekrpsi Karakter

Dekrpsi Karakter dapat dilihat pada **Tabel 2.4** Dekrpsi Karakter.

p =127, p-1-x = 119

Tabel 2.4 Dekrpsi Karakter

No	(a)	(b)	a^{p-1-x} mod p	b. a^{p-1-x} mod p	Karakter
1	104	67	37	66	B
2	15	73	22	82	R
3	72	33	52	65	A
4	69	44	103	87	W
5	61	64	19	73	I
6	103	87	68	74	J
7	72	33	52	65	A
8	72	53	52	89	Y
9	71	78	22	65	A

2.4 Serangan *Brute Force*

Brute Force adalah teknik yang paling banyak digunakan untuk memecahkan *password*, kunci, kode atau kombinasi. Cara kerja metode ini sangat sederhana yaitu mencoba semua kombinasi kunci yang mungkin. Metode ini akan menemukan solusi tetapi waktu yang dibutuhkan cukup lama dan membutuhkan memory yang besar sampai solusi ditemukan. (Saiful Huda, 2009)

2.5 Landasan Matematika

Teori matematika yang digunakan adalah sebagai berikut : (Rinaldi Munir,2006)

1. Relatif prima.
2. Aritmatika modulo.
3. Kekongruenan.
4. Inversi modulo.
5. Algoritma perpangkatan modulo.

2.5.1 Relatif Prima

Dua buah bilangan bulat a dan b dikatakan relatif prima jika PBB (a,b) = 1. jika a dan b relatif prima, maka terdapat bilangan m dan n sedemikian sehingga

ma + nb = 1

Sebagai contoh, bilangan 20 dan 3, PBB (20,3) = 1. begitu juga 11 dan 7,PBB (7,11) = 1. tapi 20 dan 5 tidak relatif prima karena PBB (5,20) = 5.

2.5.2 Aritmatika Modulo

Misalkan a adalah bilangan bulat dan m adalah bulat > 0. operasi a mod m memberikan sisa jika a dibagi dengan m. Bilangan m disebut modulus atau modulo, dan hasil aritmatika modulo m terletak didalam himpunan {0,1,2,3,m-1}.

Notasi jika a mod m = r, maka
a = mq + r, dengan 0 <= r <=m

2.5.3 Kekongruenan

Misalnya 38 mod 5 = 3 dan 13 mod 5 = 3, maka dapat dikatakan 38 ≡ 13 (mod 5) (38 kongkruen dengan 13 dalam modulo 5). Jika a tidak kongkruen dengan b maka ditulis a ≡/ b (mod m).

Kekongkruenan a ≡ b (mod m) dapat juga ditulis dengan

a = b + km

yang dalam hal ini k adalah bulangan bulat. Berdasarkan definisi arimatika modulo, kita juga dapat menuliskan a mod m = r sebagai

a ≡ r (mod m)

Berikut adalah beberapa teorema yang digunakan dalam algoritma ElGamal :

Misal m adalah bilangan bulat positif, maka

1) Jika a ≡ b (mod m) dan c adalah sembarang bilangan bulat maka :
   a. (a+c) ≡ (b+c) (mod m)
   b. ac = bc (mod m)
   c. $a^p \equiv b^p$ (mod m) , p > 0

2) Jika a $\equiv$ b (mod m) dan c $\equiv$ d (mod m), maka
   a. (a+c) $\equiv$ (b+d) (mod m)
   b. ac = bd (mod m)

2.5.4 Inversi Modulo

Jika a dan m relatif prima dan m > 1, maka kita dapat menemukan inversi dari a modulo m. Inversi dari (mod m), disebut juga inversi perkalian, adalah bilangan bulat sedemikian sehingga

$$\mathbf{aa^{-1} \equiv 1 \pmod m}$$
$$\mathbf{pa + qm \equiv 1 \pmod m}$$

karena qm = 0 (mod m), maka

$$\mathbf{pa \equiv 1 \pmod m}$$

kekongruenan yang terkahir ini berarti bahwa p adalah inversi dari a (mod m)

2.5.5 Algoritma perpangkatan modulo

Algoritma kunci *public* seperti ElGamal, RSA, Diffle –Hellman tampak sederhana perhitungannya namun sulit implementasinya dalam perangkat lunak. Hal ini karena algoritma tersebut melakukan operasi perpangkatan dengan bilangan yang besar.

Sebagai contoh pada perhitungan y :
Jika p = 127, g =13 dan x = 113

$y = g^x \bmod p = 13^{113} \bmod 127$
$= \mathbf{7.509288860117345095518488302733e+125} \bmod 127$
$= 17$

Permasalahan di atas bisa dilakukan jika menggunakan penghitungan manual dengan kalkulator, namun jika dihitung dalam kode program akan menghasilkan nilai 0. Hal ini karena tidak ada tipe data dalam *resource* pemrograman yang dapat menampung data sebelum dimodulus tersebut. Masalah tersebut dapat diselesaikan dengan proses penghitungan sebagai berikut :

$$\mathbf{vw \bmod p = [(v \bmod p)(w \bmod p)] \bmod p}$$

contoh :

$13^{113} \bmod 127 = ((13^2 \bmod 127) * (13^4 \bmod 127) * (13^8 \bmod 127) * (13^{16} \bmod 127) * (13^{32} \bmod 127) * (13^{32} \bmod 127) * (13^{16} \bmod 127) (13^2 \bmod 127) (13 \bmod 127)) \bmod 127$

Penyelesaian:

$13^2 \bmod 127 = 169 \bmod 127$

$= 42$

$13^4 \bmod 127 = 13^2 * 13^2 \bmod 127$

$= [(13^2 \bmod 127) (13^2 \bmod 127)] \bmod 127$

$= 42*42 \bmod 127$

$= 1764 \bmod 127$

$= 113$

$13^8 \bmod 127 = 13^4 * 13^4 \bmod 127$

$= [(13^4 \bmod 127) (13^4 \bmod 127)] \bmod 127$

$= 113*113 \bmod 127$

$= 12769 \bmod 127$

$= 69$

$13^{16} \bmod 127 = 13^8 * 13^8 \bmod 127$

$= [(13^8 \bmod 127) (13^8 \bmod 127)] \bmod 127$

$= 69*69 \bmod 127$

$= 4761 \bmod 127$

$= 62$

$13^{32} \bmod 127 = 13^{16} * 13^{16} \bmod 127$

$= [(13^{16} \bmod 127) (13^{16} \bmod 127)] \bmod 127$

$= 62*62 \bmod 127$

$= 3844 \bmod 127$

$= 34$

$13^{64} \bmod 127 = 13^{32} * 13^{32} \bmod 127$

$= [(13^{32} \bmod 127) (13^{32} \bmod 127)] \bmod 127$

$= 34*34 \bmod 127$

$= 1156 \bmod 127$

$= 13$

$13^{113} \bmod 127 = (13^2 * 13^4 * 13^8 * 13^{16} * 13^{32} * 13^{32} * 13^{16} * 13^2 * 13 \bmod 127) \bmod 127$

$= [(13^{64} \bmod 127)(13^{32} \bmod 127)(13^{16} \bmod 127 (13 \bmod 127)] \bmod 127$

$= (42*34*62*13) \bmod 127$

$= 356252 \bmod 127 = 17$

Contoh lain :

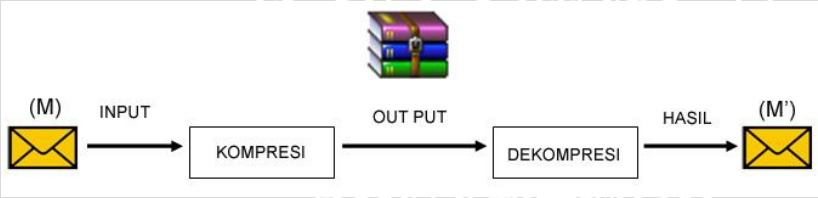
$a^8 \bmod m = (a^4 \cdot a^4) \bmod m$
$= ((a^4 \bmod m)(a^4 \bmod m)) \bmod m$
$= (a^4 \bmod m)^2 \bmod m$
$= (a^2 . a^2 \bmod m)^2 \bmod m$
$= ((a^2 \bmod m)^2 \bmod m)^2 \bmod m$

Atau dengan kata lain:

$a^8 \bmod m = ((a^2)^2)^2 \bmod m$
$= ((a^2 \bmod m)^2 \bmod m)^2 \bmod m$

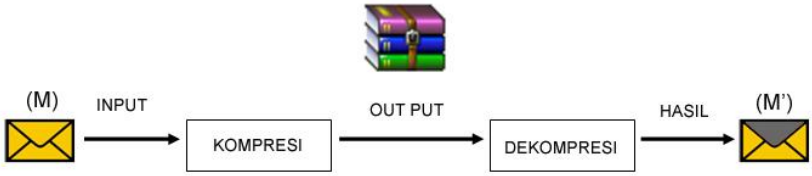
2.6 Kompresi Data

Kompresi data adalah cara untuk memperkecil ukuran data (pemampatan ukuran data), sehingga data akan lebih mudah dan lebih cepat apabila dikirimkan. Algoitma kompresi data loseless dapat digambarkan seperti pada **Gambar 2.4** Kompresi Data *Loseless*.



Gambar 2.4 Kompresi Data *Loseless*

Secara umum kompresi dibedakan menjadi *loseless compression* dan *lossy compression. Loseless compression* tidak terjadi perubahan antara stream data masukan dan stream data keluaran, proses kompresi secara *loseless* ini merupakan salah satu klasifikasi yang sering ditemukan pada kompresi jenis *text, executable file,* dan beberapa data citra dan audio (Richardson,I.,2007). Sedangkan *lossy compression* memiliki perubahan antara stream data masukan dan stream data keluaran. Kompresi data *lossy* dapat digambarkan seperti **Gambar 2.5** *Kompresi Data Lossy*.



Gambar 2.5 Kompresi Data *Lossy*

Ada beberapa metode kompresi yang dapat digunakan, metode-metode tersebut dikelompokkan dalam empat pembagian berdasarkan metode atau langkah kerjanya, berikut ini adalah klasifikasi beberapa metode kompresi yaitu:

a. Algoritma kompresi dasar
   - *Run Length Encoding*
   - *Differential Encoding*

b. Statistical compresion
   - Huffman (*static and dynamic*)
   - Shannon-Fano (*static and dynamic*)

c. Dictionary based compresion
   - LZ77
   - LZ78
   - LZW

d. Kompresi dengan transformasi
   - DCT (*Discrete Cosine Transform*)
   - *Wavelet* (Gelombang singkat)

2.6.1. Kompresi Shannon-Fano

Algoritma Shannon-Fano ditemukan oleh Claude Shannon (bapak teori informasi) dan Robert Fano pada tahun 1949. Pada dasarnya metode ini menggantikan setiap simbol dengan sebuah alternatif kode biner yang panjangnya ditentukan berdasarkan probabilitas dari simbol tersebut (Rhee, M.Y., 2003).

Secara umum algoritma Shannon-Fano diterangkan sebagai berikut:

1. Hitung masing-masing frekuensi kemunculan simbol.

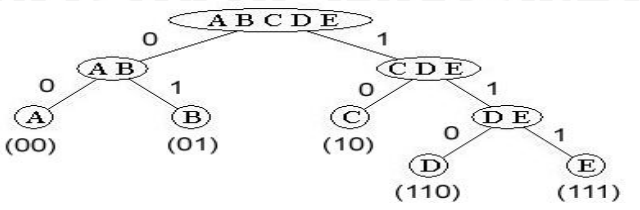
2. Urutkan frekuensi kemunculan simbol dari yang terbesar ke yang terkecil dan jumlahkan seluruh frekuensi kemunculan simbol, kemudian masukan ke sebuah *node*.
3. Bagi menjadi dua buah *node* dengan jumlah frekuensi kemunculan simbol yang sama besar atau hampir sama besar, dan berikan kode '0' untuk bagian kiri dan kode '1' untuk bagian kanan.
4. Ulangi langkah kedua, hingga tidak ada *node* yang tidak dapat dibagi lagi.
5. Subtitusi simbol-simbol yang ada dengan kode yang telah dihasilkan.

Contoh penerapan algoritma kompresi Shannon-Fano, misalkan dalam sebuah kumpulan huruf ACABECBADEDAB. Dari kumpulan huruf di atas dihitung frekuensi kemunculan tiap masing – masing huruf. Kemudian posisi huruf diurutkan dari jumlah frekuensi terbesar ke jumlah frekuensi terkecil. Jumlah frekuensi huruf dapat dilihat pada Tabel 2.5 frekuensi huruf.

Tabel 2.5 Frekuensi Huruf

Huruf	Frekuensi
A	4
B	3
C	2
D	2
E	2

Sebelum membuat pohon biner, huruf – huruf yang muncul akan dibagi menjadi dua bagian, dimana dua bagian tersebut diambil dari selisih yang paling sedikit dari jumlah frekuensi yang terdapat pada tabel. Untuk kaki sebelah kanan diberikan kode 1 dan untuk kaki sebelah kiri diberikan kode 0. Pohon biner Shannon-Fano dapat dilihat pada Gambar 2.6.



Gambar 2.6 Pohon Biner Shannon-Fano

Tabel biner Shannon Fano dapat dilihat pada **Tabel 2.6** Tabel Biner Shanon-Fano.

Tabel 2.6 Tabel Biner Shanon-Fano

Huruf	Frekuensi	Kode Biner	Panjang Biner	Total (bit)
A	4	00	2	8
B	3	01	2	6
C	2	10	2	4
D	2	110	3	6
E	2	111	3	6
			Total	30 bit

Jumlah bit sebelum kompresi : 13 (huruf) x 8(bit) = 104 bit

Setelah dilakukan kompresi, terjadi penurunan ukuran bit yaitu dari 104 bit menjadi 30 bit. Sehingga terjadi penurunan sebesar 74 bit setelah dikompresi menggunakan algoritma Shannon-Fano.
Rumus rasio kompresi dapat dilihat pada rumus 2.6 :

R= (file hasil kompresi : file asli)*100% (2.6)

Berdasarkan contoh perhitungan di atas maka rasio kompresinya dapat dihitung sebagai berikut:

R = (file hasil kompresi : file asli)*100%
= (30:104)*100%
= 28.8 %

Keterangan : Rasio kompresi (R) adalah persentase selisih dari jumlah ukuran *file* asli dengan *file* hasil kompresi.

Langkah selanjutnya adalah mensubstitusi huruf dengan kode biner yang dihasilkan dalam proses kompresi.

Text : ACABECBADEDAB

Menjadi :0010000111110010 01101111100001

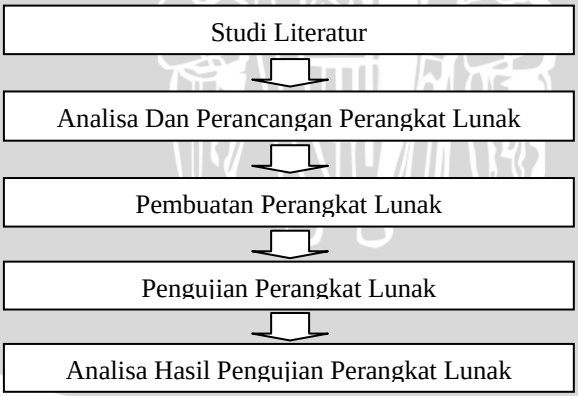
BAB III
METODOLOGI DAN PERANCANGAN SISTEM

Pada bab ini akan dibahas metode, rancangan yang digunakan dan langkah-langkah yang dilakukan dalam penelitian Implementasi Algoritma Kriptografi Elgamal Dan Algoritma Kompresi Shannon-Fano Untuk Keamanan Pengiriman *Email*.

Langkah-langkah yang digunakan dalam penelitian ini meliputi :

1. Melakukan studi literatur mengenai algoritma kompresi Shanon-Fano dan algoritma kriptografi Elgamal.
2. Menganalisa dan merancang proses kompresi pesan menggunakan algoritma kompresi Shanon Fano dan keamanan *email* dengan mengunakan algoritma kriptografi Elgamal.
3. Membuat perangkat lunak berdasarkan analisa dan perancangan yang telah dilakukan yaitu menggunakan algoritma kriptografi Elgamal dan algoritma kompresi Shannon-Fano.
4. Pengujian perangkat lunak berdasarkan hasil kompresi dan keamanan email.
5. Analisis hasil dari pengujian perangkat lunak tersebut.

Langkah-langkah penelitian ini dapat digambarkan seperti pada Gambar 3.1.



Gambar 3.1 Langkah - Langkah Penelitian

3.1 Analisis Sistem

Pada subbab analisis sistem akan dibahas berbagai hasil analisis terhadap sistem dan semua yang diperlukan dalam proses kriptografi dan kompresi.

3.1.1 Dekripsi Umum Sistem

Sistem ini digunakan untuk mengkriptografi (melakukan proses enkripsi), mengkompresi *text email*, melakukan proses dekompresi dan melakukan proses dekripsi. Sistem ini diharapkan mampu mengirim *email* dan menjaga keamanan *email*. Algoritma yang digunakan dalam penelitian ini adalah algoritma kriptografi Elgamal dan algoritma kompresi Shannon-Fano.

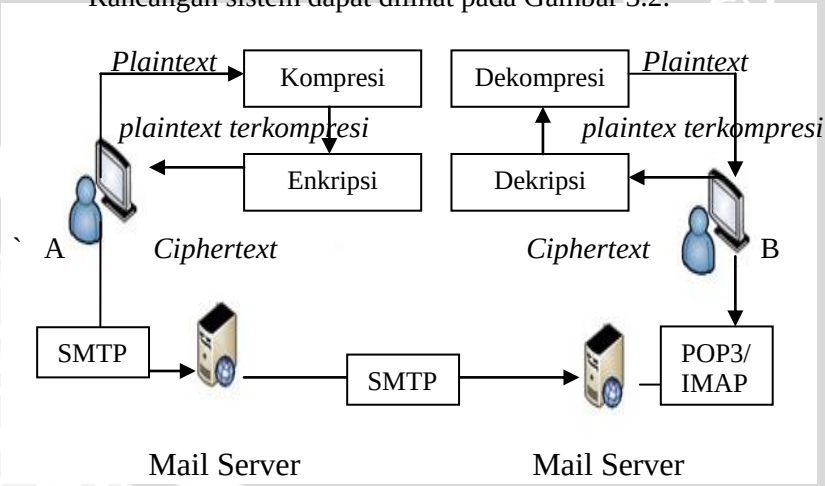
3.1.2 Batasan Sistem

Batasan dari sistem yang akan dikembangkan adalah :

1. Program yang akan dibuat menggunakan 7 bit karakter ASCII.
2. Dekompresi yang dapat dilakukan hanya untuk ekstensi *file* .txt hasil kompresi program ini.

3.1.3 Rancangan Sistem

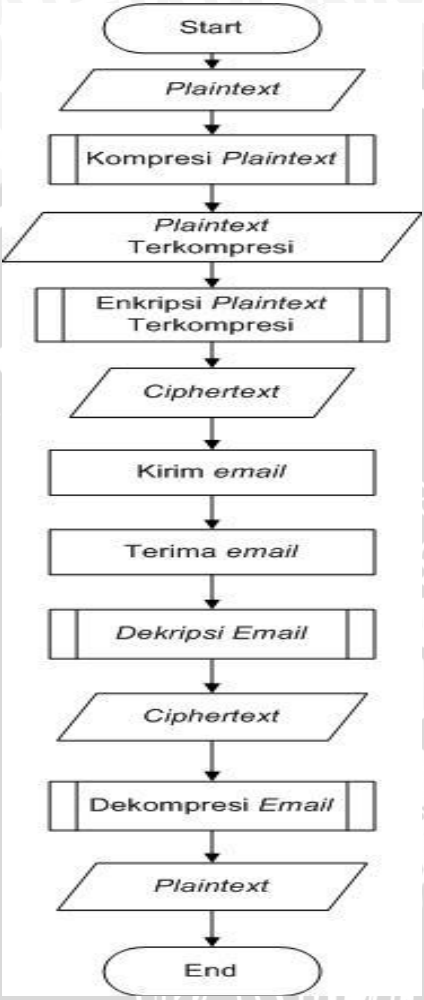
Rancangan sistem dapat dilihat pada Gambar 3.2.



Gambar 3.2 Rancangan Sistem

Dari gambar 3.2 rancangan sistem dapat dilihat bahwa *mail client* pengirim pesan (A) melakukan kompresi terhadap *plaintext* menjadi *plaintext* terkompresi, *file* terkompresi tersebut dienkripsi dengan kunci *public* menjadi *ciphertext*. Dengan port dan protokol SMTP maka pesan berupa *ciphertext* dikirim ke *mail server*. *Mail server* mengirim ke *mail server* tujuan jika si penerima memiliki *mail server* berbeda. Dengan protokol POP3 atau Imap maka *mail client* penerima pesan (B) dapat menerima *email* tersebut (*email* dalam bentuk *ciphertext*).

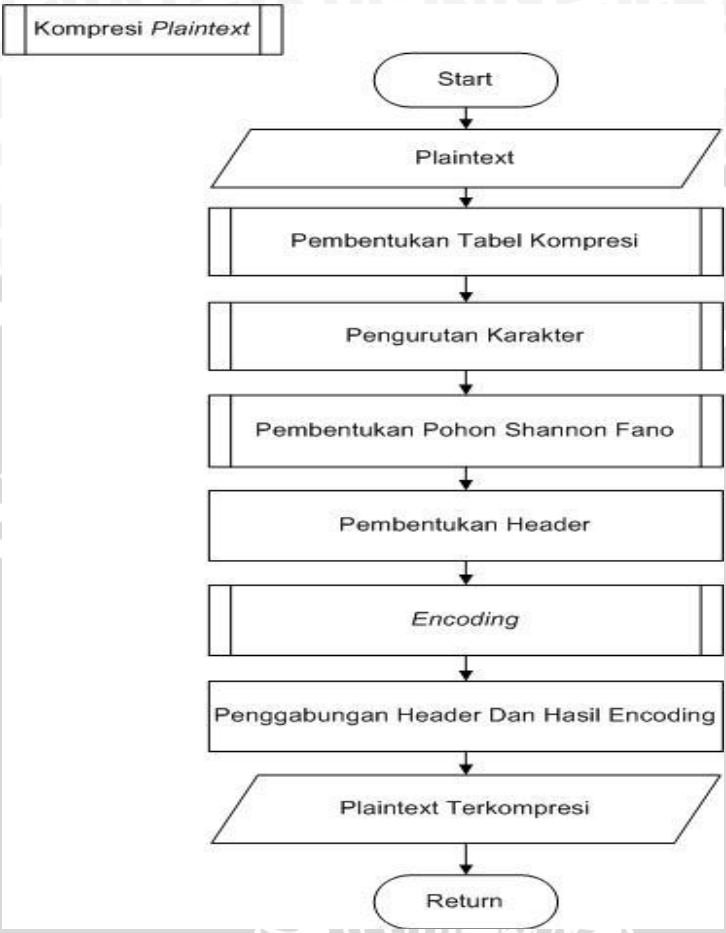
Penerima *email* dapat melakukan dekripsi dengan menggunakan kunci *private* terhadap *email* yang diterima untuk dirubah menjadi *plaintext* terkompresi dan kemudian melakukan dekompresi terhadap *plaintext* terkompresi. Fungsi dekripsi akan melakukan pengecekan kunci *private* terlebih dahulu. Inti dari program diatas adalah *mail client* dapat mengirim *email* ke *mail server* setelah melakukan kompresi dan enkripsi. Dan *mail client* penerima dapat men-*download* kiriman *email* di *mail server*, lalu melakukan dekripsi dan dekompresi. *Flowchart* umum sistem dapat dilihat pada Gambar 3.3.



Gambar 3.3 *Flowchart* Umum Sistem

3.1.4 Proses Kompresi *Plaintext*

Proses kompresi *plaintext* diawali dengan pembacaan *file input* yaitu berupa *file plaintext* kemudian dilakukan proses kompresi sehingga dihasilkan *plaintext* terkompresi. *Flowchart* proses Dekompresi dapat dilihat pada **Gambar 3.4** Proses kompresi.

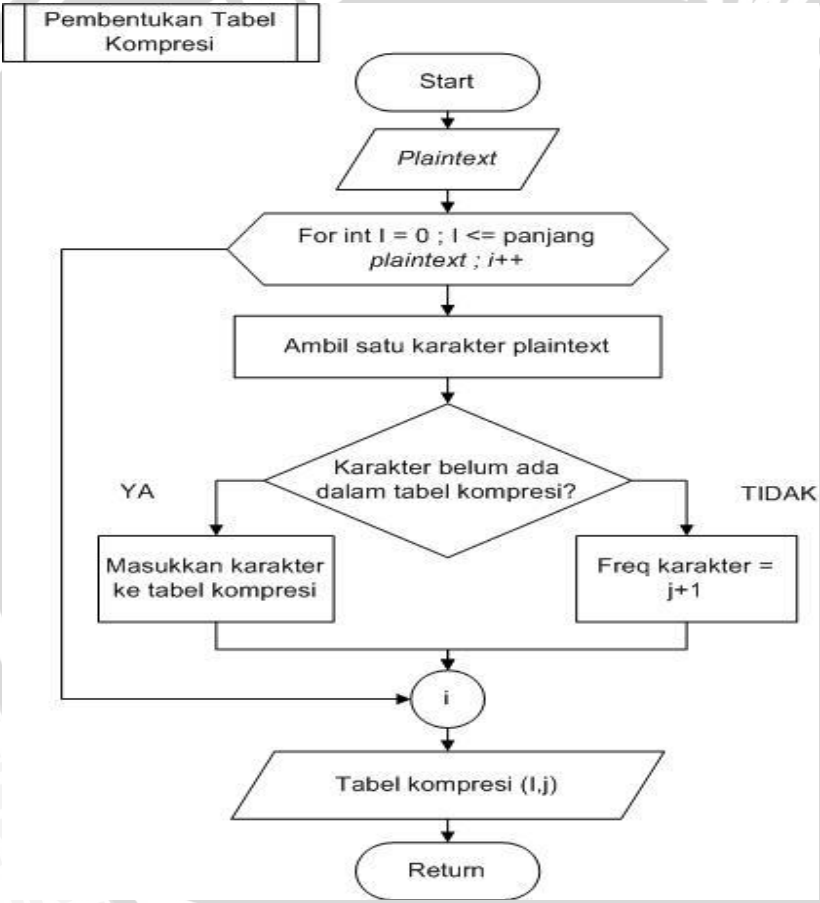


Gambar 3.4 Proses kompresi

Langkah proses kompresi adalah sebagai berikut :
1. Pembentukan tabel kompresi.
2. Pengurutan karakter (*sorting*).
3. Pembentukan pohon Shannon-Fano.
4. Pembentukan *header*.
5. Proses *encoding*.
6. Pengabungan *header* dan hasil *encoding* .
7. Proses dekompresi

3.1.4.1 Pembentukan Tabel Kompresi

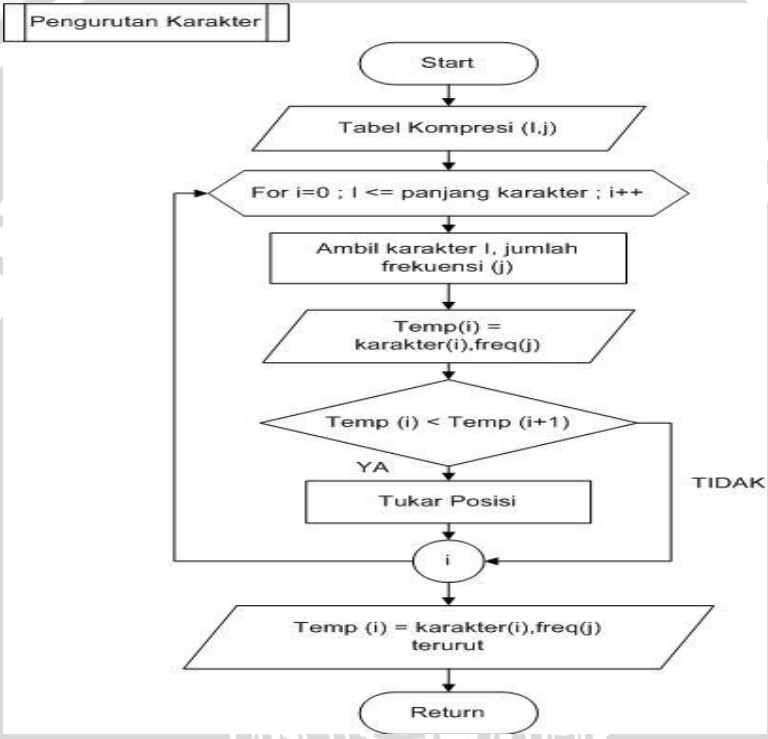
Pembentukan tabel kompresi ini digunakan untuk menghitung peluang kemunculan setiap karakter pada *ciphertext*. Tabel kompresi ini selanjutnya akan digunakan dalam proses pembuatan pohon Shannon-Fano. Tabel kompresi ini bersifat dinamis, jadi apabila terjadi perubahan data maka tabel kompresi akan berubah sehingga hasil kompresi menjadi efisien. *Flowchart* pembuatan tabel kompresi dapat dilihat pada **Gambar 3.5** *Flowchart* Pembuatan Tabel Kompresi.



Gambar 3.5 *Flowchart* Pembuatan Tabel Kompresi

3.1.4.2. Pengurutan Karakter (*Sorting*)

Semua karakter yang ada dalam *file* akan dibaca untuk dihitung frekuensi kemunculannya. Proses pengurutan yang dilakukan adalah pengurutan dari frekuensi besar ke frekuensi kecil. *Flowchart* pengurutan karakter dapat dilihat pada **Gambar 3.6** *Flowchart* Pengurutan Karakter.

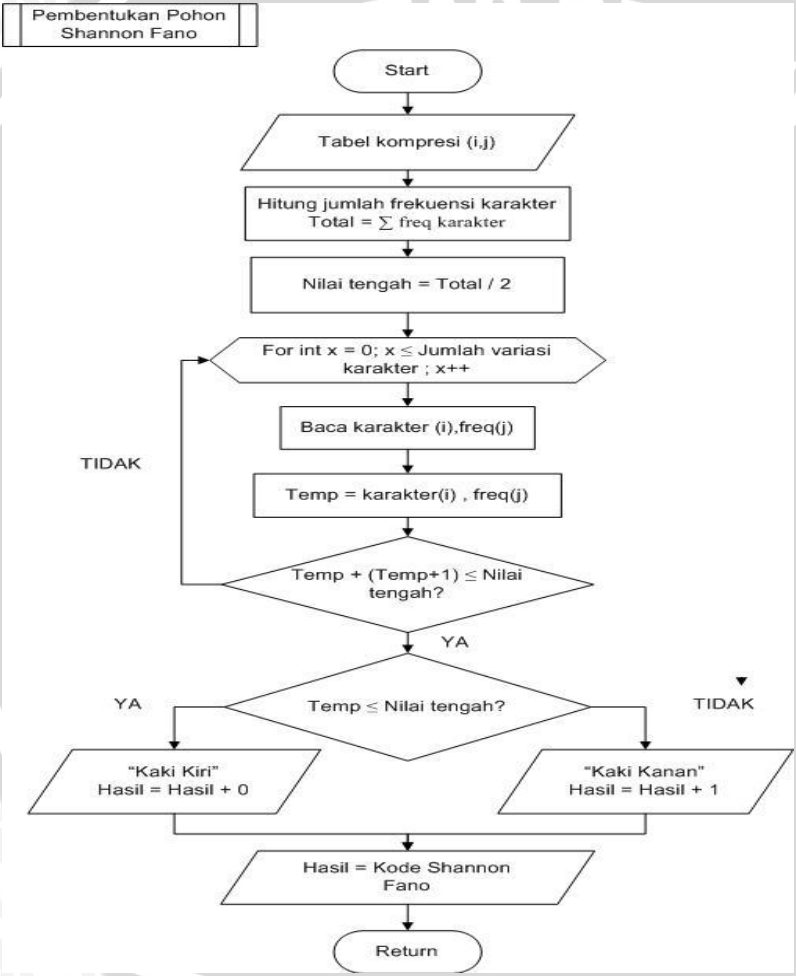


Gambar 3.6 *Flowchart* Pengurutan Karakter

3.1.4.3 Pembentukan Pohon Shannon-Fano

Berdasarkan pada tabel kompresi, setiap karakter yang ada akan direpresentasikan oleh kode Shannon-Fano yang dihasilkan oleh pembentukan pohon Shannon-Fano. Adapun aturan yang ada dalam pembentukan pohon ini adalah apabila karakter berada pada kaki kiri maka akan diinisialisasi dengan kode 0, apabila berada pada kaki kanan akan diinisialisasi dengan kode 1. Pemisahan kaki kanan dan

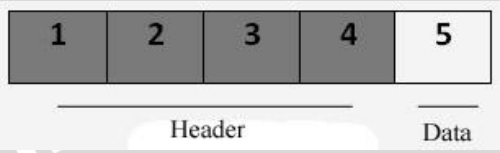
kaki kiri ditentukan oleh nilai tengah dari jumlah frekuensi karakter. Penjumlahan frekuensi karakter dari atas ke bawah sampai dengan nilai maksimum yang mendekati nilai tengah frekuensi karakter akan menjadi kaki kiri dan diinisialisasi dengan 0, sisanya akan menjadi kaki kanan dan diinisialisasi dengan 1. *Flowchart* pembentukan pohon Shannon-Fano dapat dilihat pada **Gambar 3.7** *Flowchart* Pembentukan Pohon Shannon-Fano.



Gambar 3.7 *Flowchart* Pembentukan Pohon Shannon-Fano

3.1.4.4 Pembentukan *Header*

Header merupakan bagian dari data yang akan disimpan ke dalam *file* terkompresi. *Header* berisi bagian-bagian yang berfungsi sebagai pedoman atau kamus dalam proses dekompresi. Adapun susunan *header* dapat dilihat pada **Gambar 3.8** Susunan *File* Terkompresi.



Gambar 3.8 Susunan *File* Terkompresi

Keterangan untuk bagian Gambar 3.11 adalah :

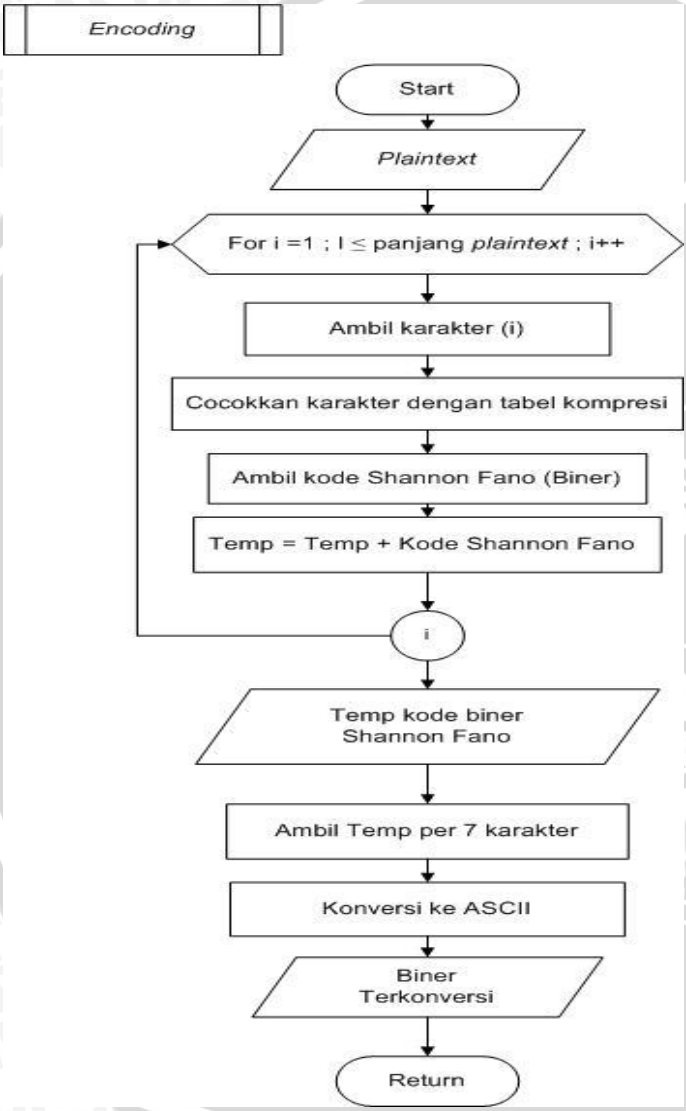
1. *Header* 1 berisi jumlah karakter pada *header* 2.
2. *Header* 2 berisi kumpulan karakter yang terdapat pada tabel kompresi.
3. *Header* 3 merupakan kumpulan frekuensi dari masing-masing karakter pada *header* 2.
4. *Header* 4 merupakan jumlah tambahan bilangan biner 0 yang diperlukan untuk pengkonversian string biner ke karakter ASCII.
5. Data hasil kompresi

Keterangan:

1. Setiap bagian dari *header file* tersebut dipisahkan oleh karakter titik (.).
2. Setiap frekuensi untuk masing-masing karakter pada *header* 3 diberi pembatas berupa karakter strip (-). Hal ini dilakukan agar tidak terjadi ambiguitas dalam proses pembacaan *file* untuk proses selanjutnya.

3.1.4.5 Proses *Encoding*

Flowchart proses *encodin*g dapat dilihat pada **Gambar 3.9** *Flowchart* Proses *Encodin*g.



Gambar 3.9 *Flowchart* Proses *Encodin*g.

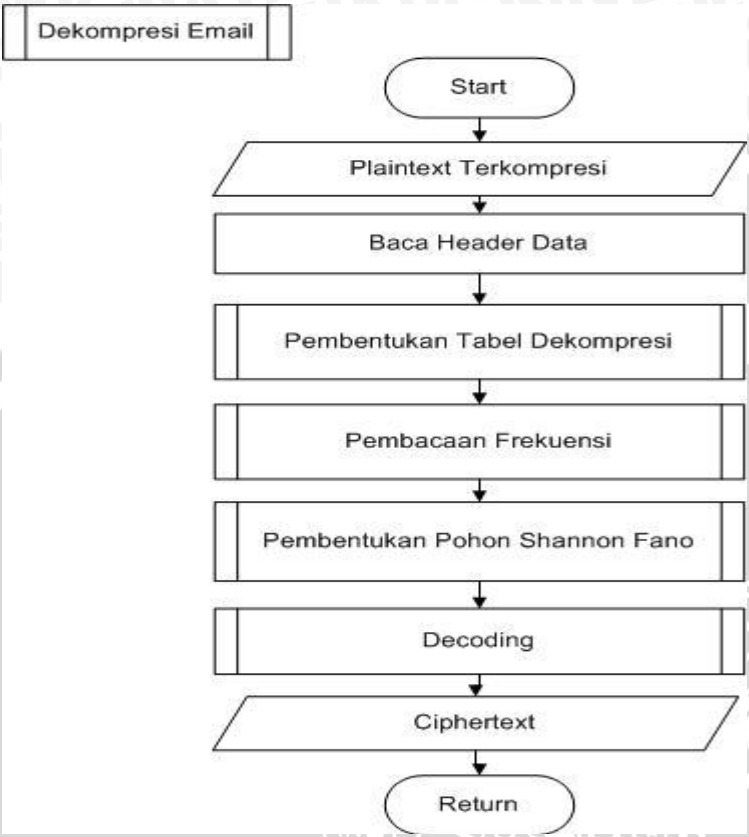
3.1.4.6 Pengabungan *Header* Dan Hasil *Encoding*

Dalam proses kompresi, hasil dari proses *encoding* adalah berupa *string* biner. Sebelum disimpan ke dalam *file* terkompresi perdelapan bit dari *string* biner ini terlebih dahulu dikonversi ke dalam karakter ASCII, karena apabila *string* biner hasil *encoding* tidak di konversi ke karakter ASCII maka ukuran *file*-nya akan menjadi besar. *File* hasil kompresi dari aplikasi ini akan disimpan dengan ekstensi .Txt.

File.Txt hasil kompresi terdiri atas *header* dan data. Seperti telah dijelaskan sebelumnya bahwa *header* ini digunakan sebagai kamus untuk menerjemahkan *string* biner menjadi karakter kembali dalam proses dekompresi. Sedangkan data dari *file*.Txt hasil kompresi berisi karakter ASCII yang merupakan hasil konversi dari per tujuh bit *string* biner hasil *encoding*. Jika panjang dari data ini tidak habis dibagi tujuh, maka akan ditambahkan nol (0) dibelakangnya. Sehingga *string* dapat habis dibagi delapan dan selanjutnya akan dikonversi ke dalam karakter ASCII.

3.1.5 Proses Dekompresi

Proses dekompresi adalah pengembalian sebuah *file* yang terkompres menjadi seperti *file* aslinya. Proses dekompresi ini dilakukan oleh penerima *email*. Proses ini diawali dengan pembacaan *file input* yaitu berupa *file* terkompresi. Aplikasi yang akan dibuat pada penelitian ini hanya dapat mendekompresi *file*.Txt hasil kompresi *chipertext* . *Flowchart* proses Dekompresi dapat dilihat pada **Gambar 3.10** Proses Dekompresi.



Gambar 3.10 Proses Pembangkitan Kode Shannon-Fano.

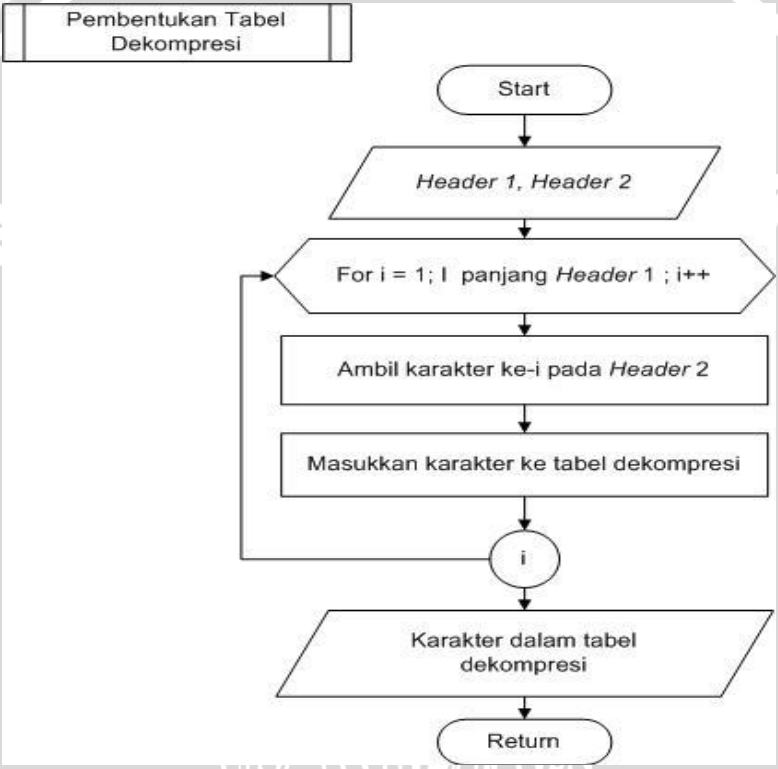
Proses dekompresi ini dilakukan oleh beberapa langkah, yaitu:
1. Pembacaan *Header* Data.
2. Pembentukan Tabel Dekompresi.
3. Pembacaan Frekuensi.
4. Pembentukan Pohon Shannon-Fano.
5. Proses *Decoding.*

3.1.5.1 Pembacaan *Header* Data

Pembacaan awal dari *header* data adalah pada header satu sehingga didapatkan banyaknya karakter pada header dua. Kemudian akan dilakukan pembacaan pada *header* dua yaitu berisi kumpulan karakter penyusun *file*.

3.1.5.2. Pembentukan Tabel Dekompresi

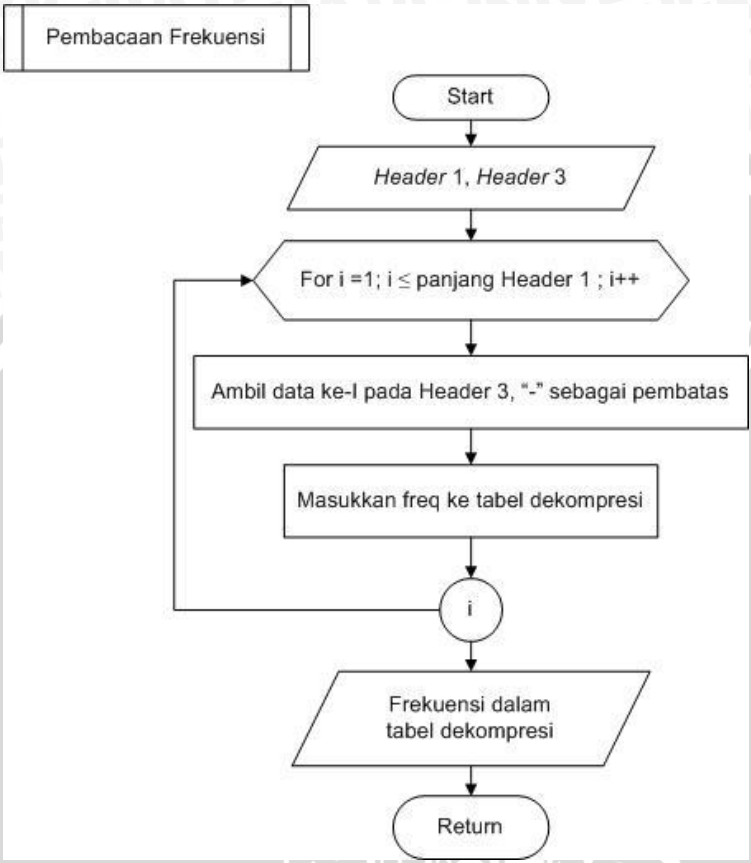
Tabel dekompresi ini berisi kumpulan karakter penyusun *file* yang didapatkan pada pembacaan *header* dua. Jumlah karakter yang diambil adalah sesuai yang terisi pada *header* satu *file* terkompresi. *Flowchart* pembuatan tabel dekompresi dapat dilihat pada Gambar **3.11** Pembuatan Tabel Dekompresi.



Gambar 3.11 Pembuatan Tabel Dekompresi

3.1.5.3 Pembacaan Frekuensi

Frekuensi dari masing-masing karakter didapat dari pembacaan header tiga. Frekuensi kemunculan karakter ini dimasukkan ke tabel dekompresi untuk membentuk kembali pohon Shannon-Fano. *Flowchart* pembacaan frekuensi dapat dilihat pada **Gambar 3.12** *Flowchart* Pembacaan Frekuensi.



Gambar 3.12 *Flowchart* Pembacaan Frekuensi

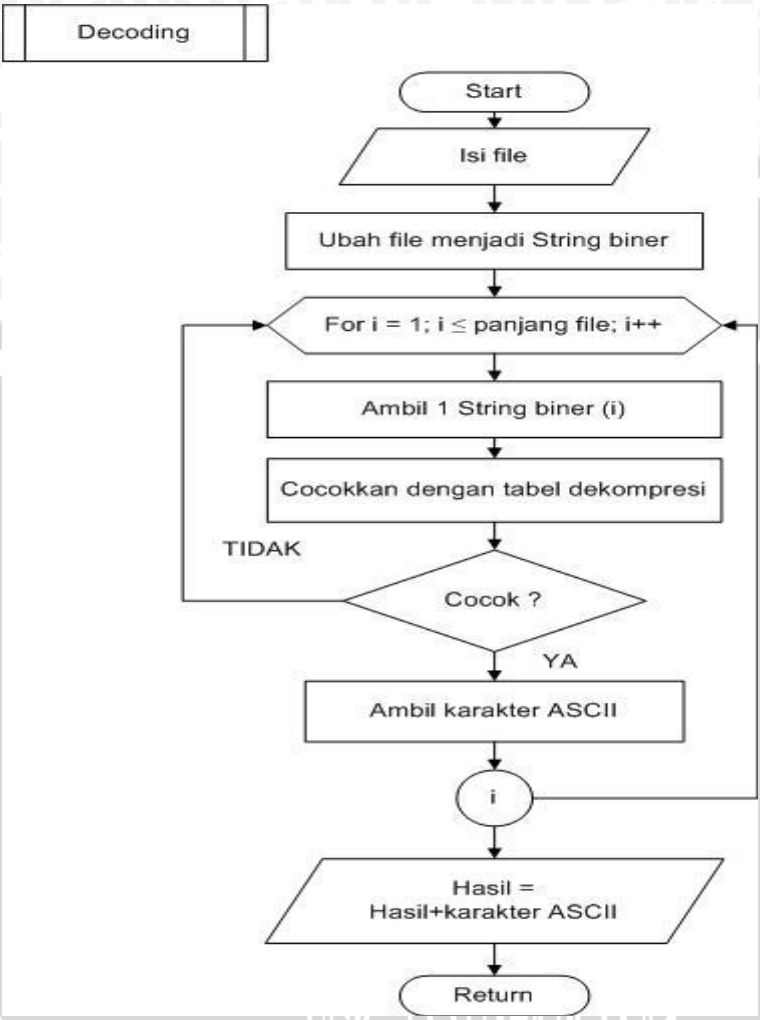
3.1.5.4. Pembentukan Pohon Shannon-Fano

Aturan yang diterapkan dalam pembentukan pohon Shannon-Fano dalam proses dekompresi ini adalah sama dengan aturan yang diterapkan dalam proses kompresi. Dari pohon Shannon-Fano ini akan terbentuk kode Shannon-Fano untuk masing-masing karakter yang kemudian disimpan dalam tabel dekompresi. Kode Shannon-Fano pada proses dekompresi ini sama dengan kode Shannon-Fano pada proses kompresi. Aturan ini diterapkan agar proses dekompresi dapat mengembalikan *file* terkompresi sama seperti *file* aslinya.

3.1.5.5. Proses *Decoding*

Decoding merupakan kebalikan dari *encoding*. Proses *decoding* bertujuan untuk mengembalikan data *string* biner menjadi sebuah karakter kembali. Proses *decoding* dilakukan dengan membaca data pada bagian tujuh yang berupa kumpulan karakter ASCII. Kemudian kumpulan karakter ASCII ini akan dikonversi kembali ke dalam *string* biner. Setelah itu *string* biner hasil konversi tersebut akan dipotong dari belakang sebanyak angka yang dibaca pada bagian enam. Bagian yang dipotong ini merupakan bilangan biner ”0” yang ditambahkan dalam proses pengkonversian ke dalam karakter ASCII. Sehingga bilangan biner yang dipotong tidak menjadi bagian dari isi *file*.

Proses *decoding* selanjutnya adalah membandingkan *string* biner hasil pemotongan di atas dengan tabel dekompresi. Pembacaan dilakukan perbilangan biner. Jika tidak terdapat kecocokan dalam tabel dekompresi maka dibaca satu bilangan biner berikutnya. Proses berlanjut hingga ditemukan kecocokan *string* biner yang dibaca dengan yang terdapat pada tabel dekompresi. Jika terdapat kecocokan pada kode Shannon-Fano maka karakter yang bersesuaian akan diambil dari tabel dekompresi untuk menggantikan kode Shannon-fano tersebut. *Flowchart* proses *decoding* dapat dilihat pada **Gambar 3.13** *Flowchart* Proses *Decoding*.

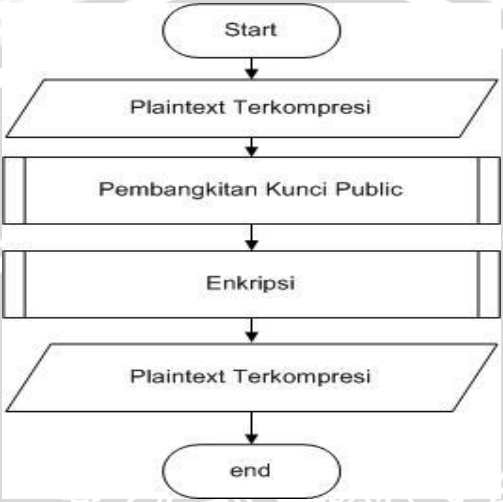


Gambar 3.13 *Flowchart* Proses *Decoding*

3.1.6 Proses Kriptografi

Proses dimulai dari membangkitkan kunci *public* kemudian dilakukan proses enkripsi. Untuk dapat membaca pesan yang telah dilakukan proses enkripsi harus dilakukan proses dekripsi .

Flowchart proses Kriptografi dapat dilihat pada **Gambar 3.14** *Flowchart* Proses Kriptografi.



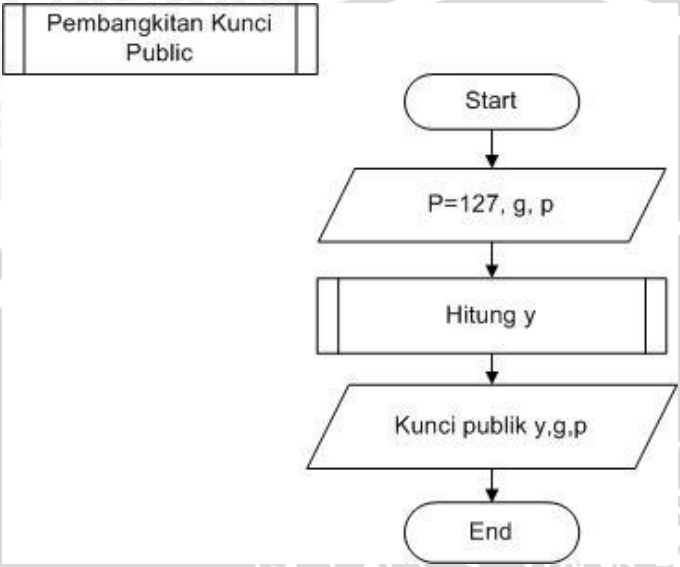
Gambar 3.14 *Flowchart* Proses Kriptografi

3.1.6.1 Proses Pembangkitan Kunci Public

Sebagaimana telah dijelaskan di atas bahwa untuk mengenkripsi pesan (*plaintext*) dibutuhkan kunci *public* milik penerima *email*. Kunci *public* dalam algoritma kriptografi Elgamal merupakan pasangan kunci (y, g, p) yang mana y adalah enkripsi dari kunci *private* (x) penerima *email*, g adalah bilangan acak dan p adalah *range* bilangan ASCII yang digunakan. Adapun syarat pembangkitan nilai g dan x yaitu:

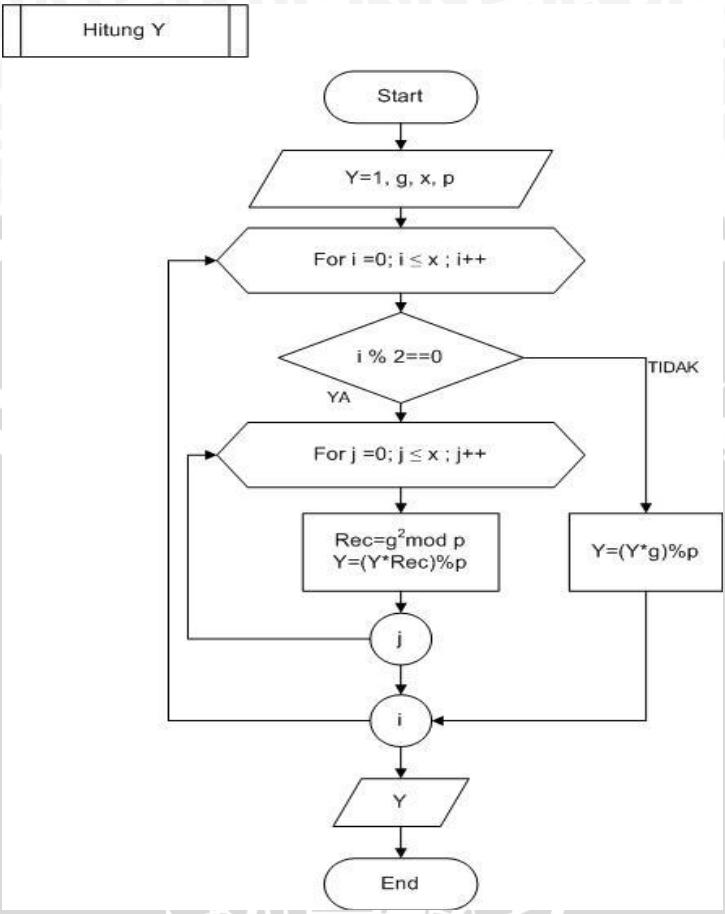
1. $g < p$
2. $1 \leq x \leq p-2$

flowchart pembangkitan kunci *public* dapat dilihat pada **Gambar 3.15** *Flowchart* Pembangkitan Kunci *Public*



Gambar 3.15 *Flowchart* Pembangkitan Kunci *Public*

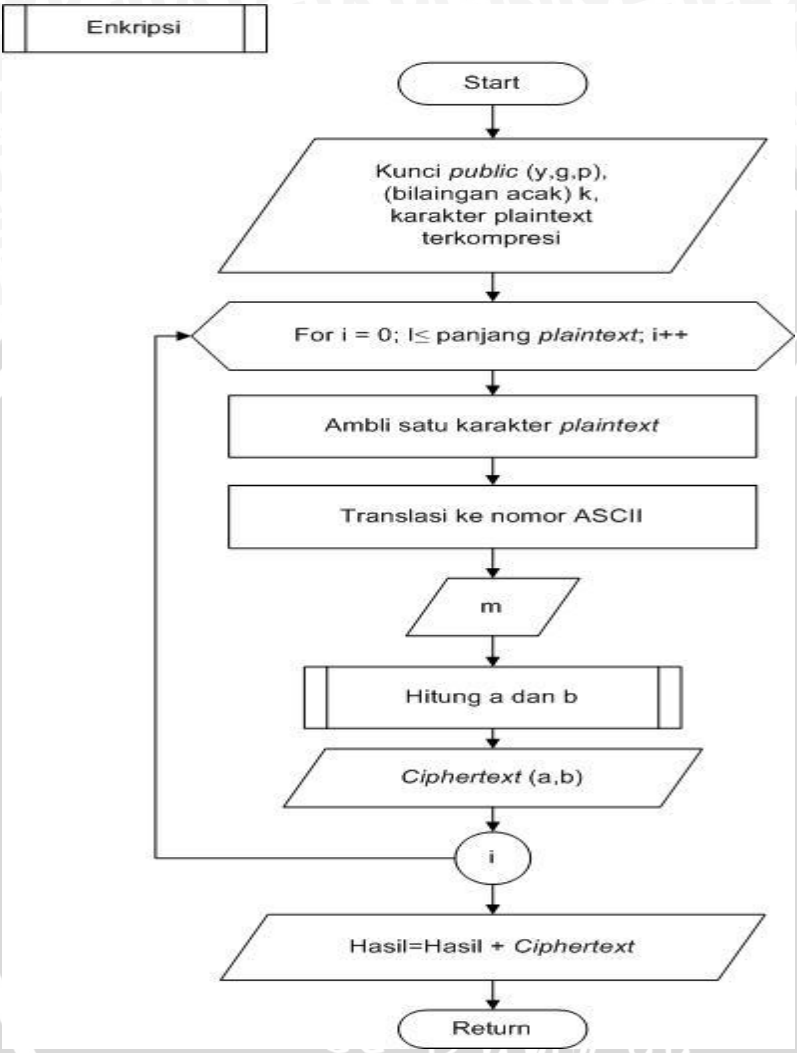
Flowchart penghitungan nilai y dengan metode *divide and conquer* dapat dilihat pada **Gambar 3.16** *Flowchart* Penghitungan Nilai y Menggunakan *divide and conquer*.



Gambar 3.16 *Flowchart* Penghitungan Nilai y Menggunakan *divide and conquer*

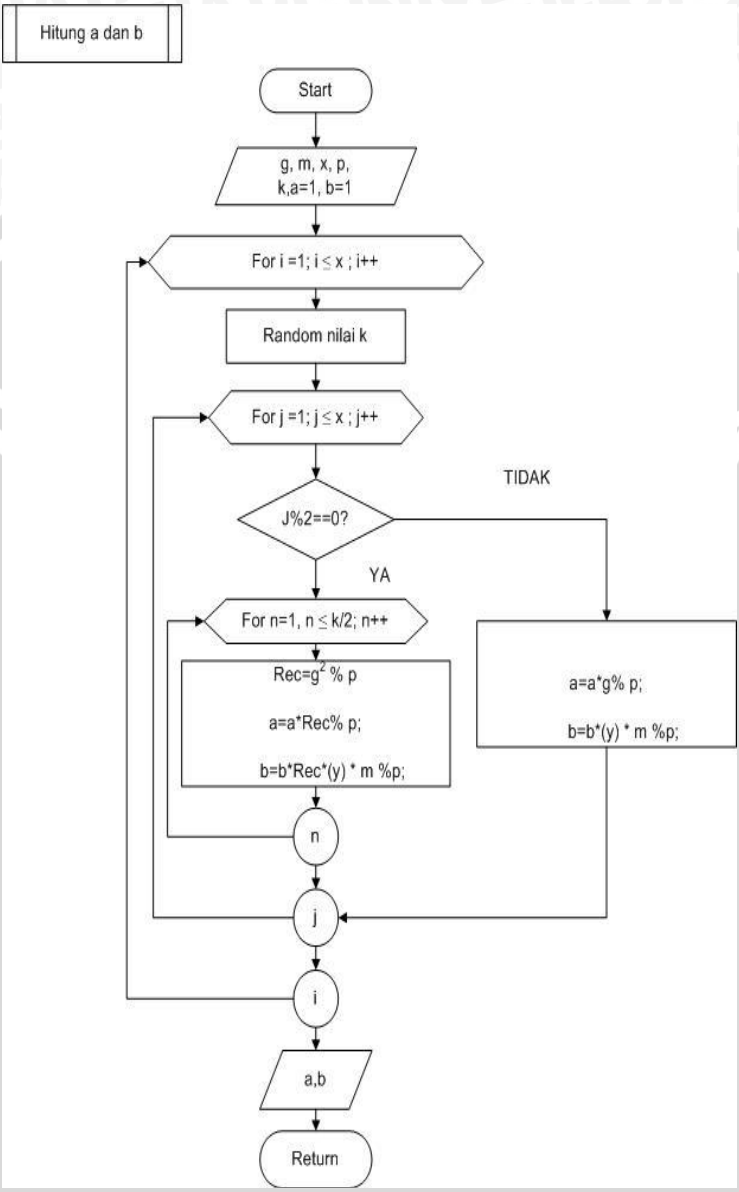
3.1.6.2 Enkripsi Pesan

Setelah pembangkitan kunci *public* selesai dan telah dikirimkan kepada pengirim pesan, barulah proses enkripsi pesan dimulai. Dalam proses enkripsi ini diperlukan translasi karakter *plaintext* ke nomor karakter ASCII yang disimpan dalam variabel m dan juga k yang merupakan bilangan prima. *Flowchart* enkripsi dapat dilihat pada **Gambar 3.17** *Flowchart* Enkripsi.



Gambar 3.17 *Flowchart* Enkripsi

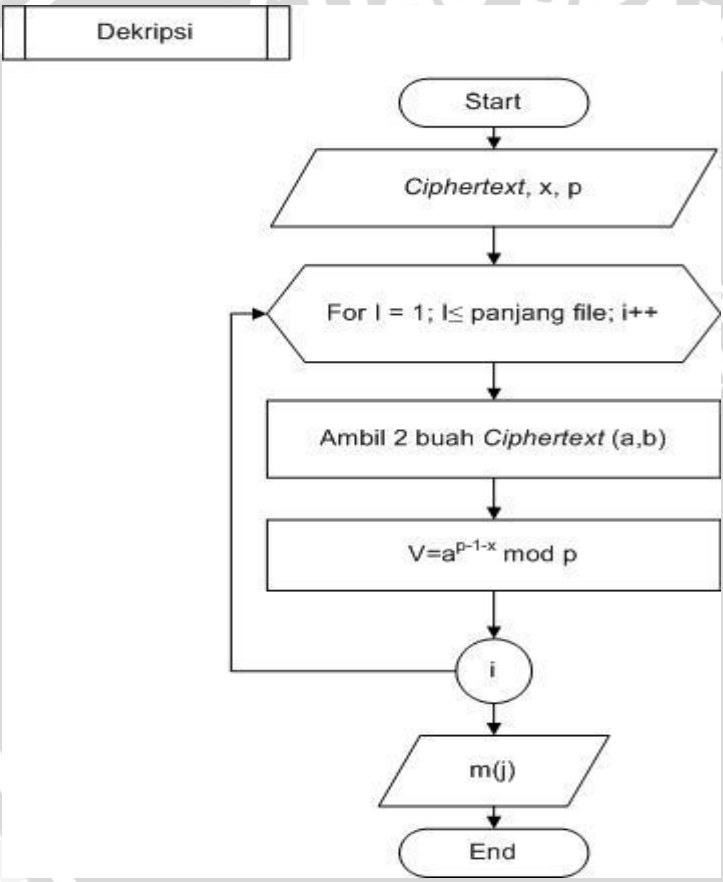
Flowchart penghitungan nilai a dan b dengan metode *divide and conquer* dapat dilihat pada **Gambar 3.18** *Flowchart* Penghitungan Nilai a dan b Menggunakan *divide and conquer*.



Gambar 3.18 *Flowchart* Penghitungan Nilai a dan b Menggunakan *divide and conquer*

3.1.6.3 Proses Dekripsi

Setelah penerima *email* selesai melakukan proses dekompresi *file* menjadi *ciphertext* kembali, langkah selanjutnya adalah mendekripsi *ciphertext*. Dekripsi merupakan kebalikan dari enkripsi. Proses dekripsi bertujuan untuk mengembalikan karakter ASCII hasil proses enkripsi (*ciphertext*) menjadi karakter semula (*plaintext*). *Flowchart* proses dekripsi dapat dilihat pada **Gambar 3.19** Proses Dekripsi.



Gambar 3.19 *Flowchart* Proses Dekripsi

3.2 Penghitungan Manual

Pengirim *email* akan mengirimkan *text*: BRAWIJAYA
Adapun langkah-langkahnya sebagai berikut :

4. Kompresi *Plaintext*.

4.a. Buat tabel kompresi :
Pembuatan tabel kompresi terdiri dari beberapa langkah, yaitu:
1. Menghitung frekuensi karakter.
2. Mengurutkan karakter berdasarkan frekuensi.
3. Membentuk kode Shannon-Fano.
4. Menerjemahkan biner ke karakter ASCII.

4.a.1. Menghitung frekuensi karakter.
Jumlah frekuensi yang telah dihitung dapat dilihat pada **Tabel 3.1** Frekuensi Karakter
.

Tabel 3.1 Frekuensi Karakter

No.	Karakter	Frekuensi	Kode
1	B	1	
2	R	1	
3	A	3	
4	W	1	
5	I	1	
6	J	1	
7	Y	1	

4.a.2 Mengurutkan karakter berdasarkan frekuensi
Pengurutan karakter digunakan untuk membentuk kaki kanan dan kaki kiri. Karakter yang telah terurut dapat dilihat pada **Tabel 3.2** Frekuensi Karakter yang Terurut.

Tabel 3.2 Frekuensi Karakter yang Terurut

No.	Karakter	Frekuensi	Kode
1	A	3	
2	B	1	
3	R	1	
4	W	1	

5	I	1	
6	J	1	
7	Y	1	

4.a.3 Membentuk kode Shannon-Fano

Pembentukan kode Shannon-Fano yaitu dengan memberikan anka 0 untuk kaki kiri dan 1 untuk kaki kanan. Pembentukan kode Shannon-Fano dapat dilihat pada **Tabel 3.3** Pembentukan Kode Shannon-Fano.

Tabel 3.3 Pembentukan Kode Shannon-Fano

No.	Karakter	Frekuensi	Kode
1	A	3	000
2	B	1	001
3	R	1	010
4	W	1	100
5	I	1	101
6	J	1	110
7	Y	1	111

4.b. Menerjemahkan biner ke karakter ASCII

Penterjemahan biner ke karakter ASCII terdiri dari beberapa langkah , yaitu :

1. Mengubah karakter *plaintext* ke kode biner Shannon-Fano.
2. Menambahkan angka 0 pada akhir bilangan biner.
3. Menerjemahkan biner ke karakter ASCII.

4.b.1 Mengubah karakter *plaintext* ke kode biner Shannon-Fano.

Pengubahan karekter *plaintext* ke kode Shannon-Fano dilakukan dengan mengambil kode Shannon-Fano yang bersesuaian dengan karakter *plaintext.*
Sehingga *plaintext* :

BRAWIJAYA

Menjadi :
001 010 000 100 101 110 000 111 000

4.b.2 Menambahkan angka 0 pada akhir bilangan biner

Sebelum melakukan penambahan angka 0, pertama dilakukan penghitungan jumlah karakter biner dari susunan biner yang dihasilkan dari proses di atas.

001 010 000 100 101 110 000 111 000
Jumlah biner = 27

Penambahan angka 0 ditentukan oleh pengurangan 7 dari sisa bagi (mod 7) jumlah biner. Kemudian angka 0 ditambahkan dibelakang kode biner sebanyak hasil pengurangan yang dihasilkan.

Sisa bagi = Jumlah biner mod 7
= 27 mod 7
= 6

Keterangan : Jumlah biner mod 7 dilakukan untuk mengetahui seberapa banyak penambahan angka 0 untuk menyempurnakan biner menjadi 7 bit karakter ASCII.

Penambahan 0 = 7-6
= 1

Setelah dilakukan penambahan angka 0, dilakukan penambahan 0 sebanyak 1.

Susunan biner menjadi :
001 010 000 100 101 110 000 111 000 0

Biner dirapatkan :
00101000010010111000001110000

Dipecah per 7 biner :
0010100 0010010 1110000 1110000

4.b.3 Translasi ke kode ASCII
0010100 0010010 1110000 1110000
Menjadi : 42ÉÉ

5. Penyimpanan *file* kompresi
Mekanisme penyimpanan *file* kompresi dari BRAWIJAYA adalah :
7.A-B-R-W-I-J-Y.3-1-1-1-1-1-1.1.42ÉÉ

Kriptografi :

1. Pembentukan kunci *public* dan kunci *private* oleh komputer penerima. Kunci *private*, x = 64 (bilangan acak) g =13, p =127,
Kunci *public* , $y = g^x \bmod p$
$= 13^7 \bmod 127$
$= 103$
Kunci *public* yang terbentuk (y, g, p) = (103, 13, 127)
2. Konversi Karakter Pesan ke Kode ASCII
Mengkonfersi karakter *text* ke dalam nomor yang sesuai dalam kode ASCII. Karakter yang telah dikonversi ke nomer ASCII dapat dilihat pada **Tabel 3.4** Konversi Karakter Pesan ke Kode ASCII.

Tabel 3.4 Konversi Karakter Pesan ke Kode ASCII

No	Karakter	Plainteks (m)	ASCII
1	7	m1	23
2	.	m2	14
3	A	m3	33
4	-	m4	13
5	B	m5	34
6	-	m6	13
7	R	m7	50
8	-	m8	13
9	W	m9	55
10	-	m10	13
11	I	m11	41
12	-	m12	13
13	J	m13	42

14	-	m14	13
15	Y	m15	57
16	.	m16	14
17	3	m17	19
18	-	m18	13
19	1	m19	17
20	-	m20	13
21	1	m21	17
22	-	m22	13
23	1	m23	17
24	-	m24	13
25	1	m25	17
26	-	m25	13
27	1	m26	17
28	-	m27	13
29	1	m28	17
30	.	m29	14
31	1	m30	17
32	.	m31	14
33	4	m32	20
34	2	m33	18
35	É	m34	112
36	É	m35	112

3. Enkripsi Karakter *Text*

Mengenkripsi karakter *text* berdasarkan nomer ASCII menjadi nomer ASCII yang baru. Nomer ASCII hasil enkripsi dapat dilihat pada **Tabel 3.5** Enkripsi Karakter *Text*.

Keterangan : g=13 , y=103 p=127

Tabel 3.5 Enkripsi Karakter *Text*

No.	(m)	(k)	$a=g^k \bmod p$	$b=y^k m \bmod p$
1	23	73	104	106
2	14	31	71	105
3	33	13	15	114
4	13	11	82	50
5	34	41	18	104

6	13	23	36	87
7	50	107	49	37
8	13	29	41	25
9	55	2	42	24
10	13	37	35	74
11	41	5	72	31
12	13	79	62	44
13	42	109	26	76
14	13	59	9	117
15	57	3	38	7
16	14	31	71	105
17	19	23	36	49
18	13	11	82	50
19	17	7	103	100
20	13	53	11	16
21	17	53	11	60
22	13	13	15	68
23	17	113	17	35
24	13	41	18	107
25	17	13	15	1
26	13	13	15	68
27	17	101	74	115
28	13	113	17	94
29	17	31	71	64
30	14	31	71	105
31	17	11	82	124
32	14	11	82	5
33	20	17	44	118
34	18	37	35	122
35	112	103	60	116
36	112	53	11	89

Chipertext yang terbentuk :
êè…ë/ÆrR2êDwQEI9J8Cjh?^L:l)òF'…ëDQrRçä+0+g/d1C2¼/!/djô1
~…`…ër£r%LûCÜgö+y
6. Dekripsi

Chipertext yang telah di dekripsi dapt dilihat pada Tabel 3.6
Keterangan : P-1-x = 62

Tabel 3.6 Dekripsi

No	(a)	(b)	$a^{p-1-x} mod\ P$	$b.a^{p-1-x} mod\ P$	Karakter
1	104	106	11	23	7
2	71	105	34	14	.
3	15	114	17	33	A
4	82	50	79	13	-
5	18	104	120	34	B
6	36	87	60	13	-
7	49	37	70	50	R
8	41	25	31	13	-
9	42	24	124	55	W
10	35	74	98	13	-
11	72	31	30	41	I
12	62	44	84	13	-
13	26	76	44	42	J
14	9	117	113	13	-
15	38	7	117	57	Y
16	71	105	34	14	.
17	36	49	60	19	3
18	82	50	79	13	-
19	103	100	37	17	1
20	11	16	104	13	-
21	11	60	104	17	1
22	15	68	17	13	-
23	17	35	15	17	1
24	18	107	120	13	-
25	15	1	17	17	1
26	15	68	17	13	-
27	74	115	115	17	1
28	17	94	15	13	-
29	71	64	34	17	1
30	71	105	34	14	.
31	82	124	79	17	1

32	82	5	79	14	.
33	44	118	26	20	4
34	35	122	98	18	2
35	60	116	36	112	É
36	11	89	104	112	É

7. Dekompresi

Proses dekompresi :

7.A-B-R-W-I-J-Y.3-1-1-1-1-1-1.1.42ÉÉ

1. Baca *header* data pertama, yaitu jumlah karakter penyusun *file* asli. Bentuk tabel baru.Pembentukan tabel baru dapat dilihat pada **Tabel 3.7** Dekonpresi.

Tabel 3.7 Dekompresi

No.	Karakter	Frekuensi	Kode
1			
2			
3			
4			
5			
6			
7			

2. Baca *header* data kedua, yaitu karakter penyusun *file* asli. Isikan karakter tersebut ke dalam tabel yang dibentuk. Pengisian karakter dapat dilihat pada **Tabel 3.8** Pengisian Karakter.

Tabel 3.8 Pengisian Karakter

No.	Karakter	Frekuensi	Kode
1	A		
2	B		
3	R		
4	W		
5	I		

6	J		
7	Y		

3. Baca header *file* ketiga, yaitu jumlah frekuensi masing-masing karakter penyusun *file* asli. Pengisian frekuensi dapat dilihat pada **Tabel 3.9** Pengisian Frekuensi.

Tabel 3.9 Pengisian Frekuensi

No.	Karakter	Frekuensi	Kode
1	A	3	
2	B	1	
3	R	1	
4	W	1	
5	I	1	
6	J	1	
7	Y	1	

4. Buat pohon Shannon-Fano kembali. Pembuatan pohon Shannon Fano **Tabel 3.10** Pembuatan Pohon shannon Fano Kembali.

Tabel 3.10 Pembuatan Pohon Shannon Fano Kembali

No.	Karakter	Frekuensi	Kode
1	A	3	000
2	B	1	001
3	R	1	010
4	W	1	100
5	I	1	101
6	J	1	110
7	Y	1	111

5. Baca *header file* keempat, yaitu jumlah penambahan angka 0. Pada header keempat contoh di atas, penambahan angka 0 sebanyak 1.

6. Baca Data pada bagian ke lima, ubah isi *file* menjadi *string* biner, dan lakukan pemotongan penambahan

angka 0 dibelakang *string* biner sebanyak pada *header* empat. Langkah-langkahnya akan dijelaskan sebagai berikut:

1. Data bagian ke lima : 42ÉÉ
2. Data tersebut dirubah ke string binernya, menjadi:

0010100 0010010 1110000 1110000

3. Potong penambahan angka 0 dibelakang sesuai dengan header 4. Dilakukan pemotongan angka 0 sebanyak 1.

   Biner :

0010100 0010010 1110000 111000

Rapatkan:

00101000010010111000011100 0

Pada biner yang telah dipotong penambahan angka 0 nya : (00101000010010111000011100 0) dilakukan proses dekompresi menjadi karakter aslinya. Langkah langkahnya akan dijelaskan sebagai berikut :

Pembacaan dilakukan menggunakan pointer :

00101000010010111000011100 0

Proses:

1. Pointer menunjuk angka 0, angka 0 dicocokkan pada table biner Shannon fano.
2. Hasil = tak ada kode sama, dilanjutkan pointer selanjutnya
3. Pointer menunjuk angka 00, angka 00 dicocokkan pada table Shannon fano.
4. Hasil = tak ada kode sama, dilanjutkan pointer selanjutnya
5. Pointer menunjuk angka 001, angka 001 dicocokkan pada table Shannon fano.
6. Hasil = ada, Karakter B
7. Ganti angka 001 menjadi B

Biner :
00101000010010111000011100 0
Menjadi :
B0100001001011100001110 00

Proses di atas diulang sampai biner habis, sehingga didapatkan kembali kode :
BRAWIJAYA

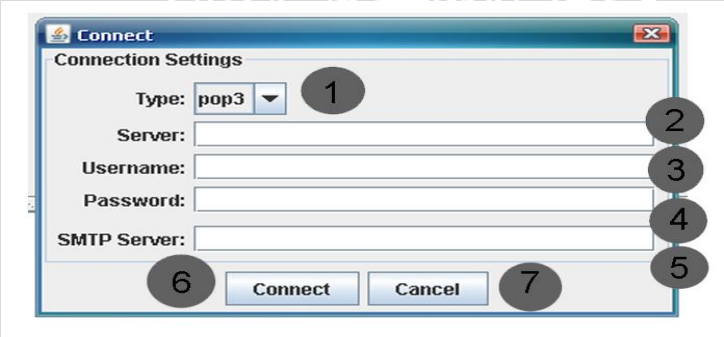
3.3 Rancangan Antar Muka Aplikasi

Rancangan antar muka aplikasi ini terdiri dari :

1. Halaman koneksi.
2. Halaman *create email*.
3. Halaman *inbox.*

3.3.1 Rancangan Halaman Koneksi

Rancangan halaman koneksi dapat dilihat pada Gambar 3.20.



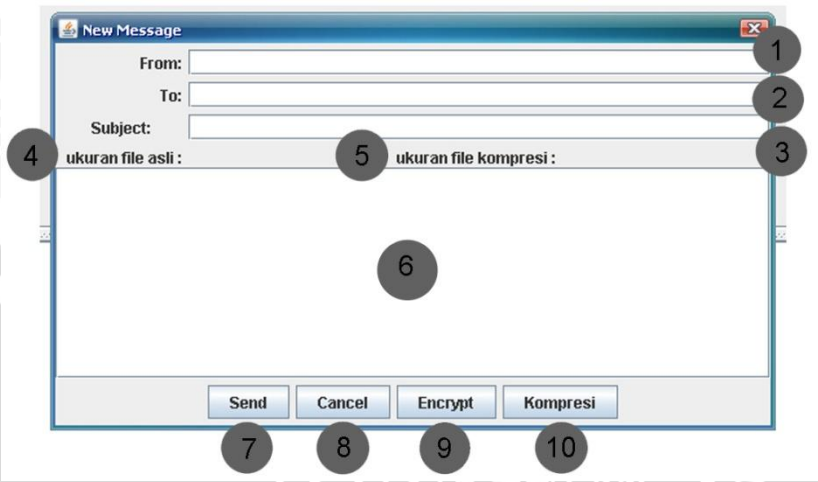
Gambar 3.20 Halaman Koneksi

Keterangan Gambar 3.18**:**

1. Tipe *server POP3* atau *IMAP*.
2. Alamat *server POP3* atau *IMAP*.
3. *Username.*
4. *Pasword.*
5. *Alamat SMTP server*.
6. Tombol koneksi.
7. Tombol untuk membatalkan koneksi.

3.3.2 Rancangan Halaman *Create Email*

Rancangan halaman *create email* dapat dilihat pada Gambar 3.21.



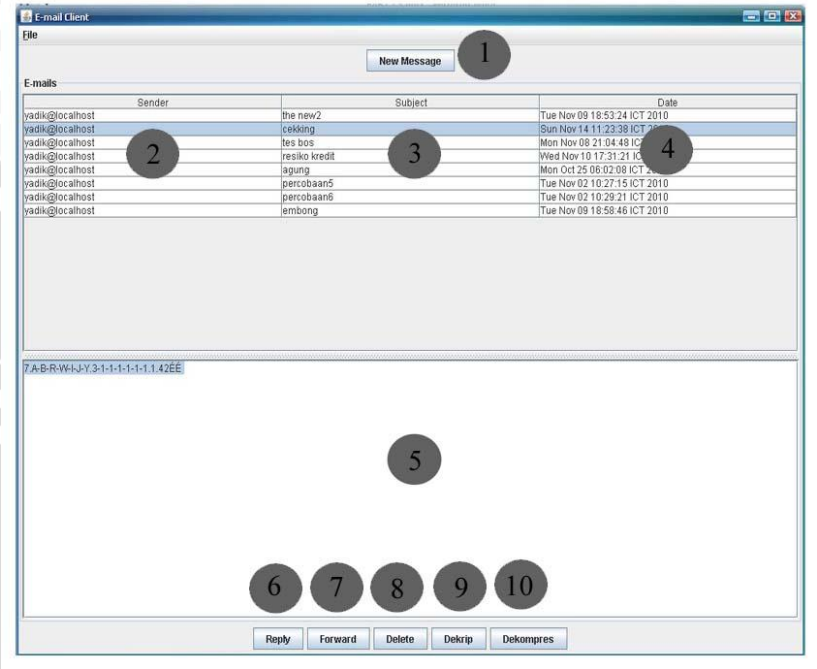
Gambar 3.21 Halaman *Create Email*

Keterangan Gambar 3.21:

1. Inputan alamat asal *email* dikirimkan.
2. Inputan alamat tujuan *email* dikirimkan.
3. *Subject email*.
4. Ukuran *file email*.
5. Ukuran *file email* setelah dikompresi.
6. Isi *email* (plaintext).
7. Tombol untuk mengirim *email*
8. Tombol untuk membatalkan pengiriman.
9 . Tombol untuk mengenkripsi pesan.
10. Tombol untuk mengkompresi pesan.

3.3.3 Rancangan Halaman *Inbox*

Rancangan halaman *inbox* dapat dilihat pada Gambar 3.22.



Gambar 3.22 Halaman *Inbox*

Keterangan :
1.Tombol untuk membuat pesan baru.
2. Alamat pengirim *email*.
3. *subject email*.
4. Tanggal *email* dikirim.
5. isi *email* (*plaintext, chipertext, file* terkompresi).
6. Tombol untuk membalas pesan .
7. Tombol untuk mengirim pesan ke alamat email yang lain.
8. Tombol untuk menghapus pesan.
9. Tombol untuk mendekrip pesan.
10. Tombol untuk mendekompres pesan.

3.4 Rancangan Uji Coba dan Evaluasi Hasil

Uji coba sistem kriptografi Elgamal dan kompresi Shannon-Fano digunakan untuk mengevaluasi keamanan sistem kriptografi Elgamal dan mengukur seberapa efektif kompresi *text* Shannon-Fano.

3.4.1 Perancangan Uji Coba dan Evaluasi Hasil

Pengujian terhadap kompresi Shannon-Fano dilakukan dengan membandingkan *file* sebelum dikompresi dengan *file* yang telah dikompresi dan dihitung rasio kompresinya. Rancangan tabel untuk pengujian kompresi dapat dilihat pada **Tabel 3.11** Rancangan Pengujian Kompresi.

Tabel 3.11 Rancangan Pengujian Kompresi

Ukuran *file* asli	Ukuran *file* setelah dikompresi	Rasio Kompresi

Pengujian dilakukan dalam uji coba proses kriptografi adalah menghitung ukuran (*size*) *file, brute force* dan waktu yang diperlukan untuk memecahkan kriptografi.

Untuk pengujian menggunakan parameter ukuran (*size*) *file* yang diujikan adalah *file* .txt , pada setiap *file* yang diujikan akan dihitung ukuran *file* awal (sebelum dilakukan proses enkripsi maupun dekripsi), ukuran *file* setelah proses enkripsi dan ukuran *file* setelah dekripsi. Tujuan pengukuran ini dilakukan adalah untuk mengetahui seberapa besar pengaruh proses enkripsi terhadap ukuran (*size*) *file*. Rancangan tabel untuk pengujian ukuran *file* dapat dilihat pada **Tabel 3.12** Rancangan Pengujian Ukuran *File*.

Tabel 3.12 Rancangan Pengujian Ukuran *File*

Nama *File* Uji	Ukuran Awal	Ukuran Enkripsi	Ukuran Dekripsi

Pengujian yang akan dilakukan untuk memecahkan kriptografi Elgamal dibagi menjadi dua berdasarkan tingkatan kriptanalis, yaitu:

1. Kriptanalis yang tidak mendapatkan kunci *public*.

Untuk memecahkan kriptografi, kriptanalis harus melakukan *brute force* kunci dekripsi sebanyak kombinasi dari jumlah p (jumlah simbol yang digunakan dalam kriptografi), dalam skripsi ini digunakan p (127) atau 8bit *printable* karakter ASCII dan jumlah kemungkinan kunci x (kunci *private*) yaitu sebanyak 125 x 127 = 15875 kemungkinan.

Rancangan tabel untuk pengujian kriptografi oleh kriptanalis yang mendapatkan kunci *public* dapat dilihat pada **Tabel 3.13** Pengujian serangan *brutoforce* terhadap kunci *private* dengan kunci *public* belum diketahui.

Tabel 3.13 Pengujian serangan *brutoforce* terhadap kunci *private* dengan kunci *public* belum diketahui.

No.	Nama file	Ukuran File	Ukuran Setelah Enkripsi	Ukuran Dekripsi	waktu dekripsi	waktu dekripsi 15875 kunci

2. Kriptanalis yang mendapatkan kunci public.

Untuk memecahkan kriptografi, kriptanalis harus melakukan *brute force* kemungkinan kunci dekripsi sebanyak jumlah P. Dari kunci yang dihasilkan dari proses *brute force* dicari nilai yang kongruen dengan y (kunci *public*) dimana y di dapatkan dari rumus:

$y = g^x \bmod p$

Rancangan tabel untuk pengujian kriptografi oleh kriptanalis yang mendapatkan kunci *public* dapat dilihat pada **Tabel 3.14** Pengujian Mendapatkan Bilangan Kongruen

Tabel 3.14 Pengujian Mendapatkan Bilangan Kongruen

No	$y(2)=g^x \bmod p$	$y2 \equiv y$	Waktu (s)

BAB IV
HASIL DAN PEMBAHASAN

4.1. Lingkungan Implementasi

Lingkungan implementasi yang akan dijelaskan dalam sub bab ini adalah lingkungan implementasi perangkat keras dan perangkat lunak.

4.1.1. Lingkungan Implementasi Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan aplikasi Implementasi Algoritma Kriptografi Elgamal Dan Algoritma kompresi Shannon Fano adalah :

1. Prosesor Intel Celeron 1,6 GHz
2. Memory (RAM) 1GB
3. Harddisk dengan kapasitas 60 GB
4. VGA ATI Radeon 64 Mb

4.1.2. Lingkungan Implementasi Perangkat Lunak

Perangkat Lunak yang digunakan dalam pengembangan aplikasi Implementasi Algoritma Kriptografi Elgamal Dan Algoritma kompresi Shannon Fano ini yaitu :

1. Sistem Operasi yang digunakan adalah *Microsoft Windows XP Professional SP3*
2. Program dibuat dengan menggunakan perangkat lunak *JAVA*
3. Program menggunakan struktur data array.

4.2. Deskripsi Program

4.2.1 Proses Kompresi

4.2.1.1 Proses Pembentukan Tabel Kompresi dan Proses Pengurutan Karakter.

```
private void createTabel()
{
   for(int i=0; i<arrayHuruf.length; i++)
   {
      if (firstNode == null)
      {
         TNode newNode = new TNode();
         newNode.asciiChar = arrayHuruf[i];
         newNode.frekunsi = 1;
         newNode.huruf = (char)(arrayHuruf[i]);
         newNode.nextNode = null;
         firstNode = lastNode = newNode;
      }
      else
      {
         currentNode = firstNode;
         while((currentNode.asciiChar != arrayHuruf[i]) &&
               (currentNode.nextNode != null))
         {
            currentNode = currentNode.nextNode;
         }

         if (currentNode.nextNode == null)
         {
            if (currentNode.asciiChar == arrayHuruf[i])
            {
               currentNode.frekunsi++;
            }
            else
            {
               TNode newNode = new TNode();
               newNode.asciiChar = arrayHuruf[i];
               newNode.frekunsi = 1;
               newNode.huruf = (char)(arrayHuruf[i]);
               newNode.nextNode = null;
               currentNode.nextNode = newNode;
               lastNode = newNode;
          }  }
else
         {
            currentNode.frekunsi++;
}}}}
```

Node dibuat sebanyak jumlah variasi karakter. Node disini adalah object yang menampung variasi karakter, frekuensi karakter dan kode biner yang dihasilkan dari pembuatan pohon Shannon-Fano.

4.2.1.2 Proses Pengurutan Karakter.

```
private void sortingTabel()
   {
      currentNode = firstNode;

      while(currentNode.nextNode != lastNode)
      {
         if ((currentNode == firstNode)
          && (currentNode.frekunsi < currentNode.nextNode.frekunsi))
         {
            firstNode = currentNode.nextNode;
            currentNode.nextNode = firstNode.nextNode;
            firstNode.nextNode = currentNode;

            currentNode = firstNode;
         }
         else if (currentNode.nextNode.frekunsi
          < currentNode.nextNode.nextNode.frekunsi)
         {
            //swap
            TNode bufNode = new TNode();
            bufNode = currentNode.nextNode.nextNode;
            currentNode.nextNode.nextNode = bufNode.nextNode;
            bufNode.nextNode = currentNode.nextNode;
            currentNode.nextNode = bufNode;

            currentNode = firstNode;
         }
         else
            currentNode = currentNode.nextNode;
      }
   }
```

Node diurutkan (*sorting*) berdasarkan jumlah dari masing-masing karakter. Pengurutan dilakukan dari jumlah frekuensi terbesar samapai frekuensi terkecil.

4.2.1.3 Proses Pembentukan Pohon Shannon Fano

```
private void completingTabel()
   {
      int total, ind;
      ind = total = 0;

      currentNode = firstNode;
      while(currentNode != null)
      {
         total = total + currentNode.frekunsi;
         currentNode = currentNode.nextNode;
         ind++;
      }
      totalFrekuensi = total;
      nodeLength = ind;

      ind = total = 0;
      currentNode = firstNode;

      for (currentNode = firstNode; currentNode != null;
       currentNode = currentNode.nextNode)
      {
         total = total + currentNode.frekunsi;
         if ((total*2) >= totalFrekuensi)
         {
            break;
         }
         ind++;
      }

      indexHalf = ind;

      left = indexHalf + 1;
      right = nodeLength - left;

      String[] arrString = new String[left];
      arrString = createBiner(left);
      currentNode = firstNode;
      for (int i=0; i<left; i++)
      {
         currentNode.binerCode = "0" + arrString[i];
         currentNode = currentNode.nextNode;
      }
      arrString = new String[right];
      arrString = createBiner(right);
      for(int i=0; i<right; i++)
      {
```

```
            currentNode.binerCode = "1" + arrString[i];
            if (currentNode != null)
                currentNode = currentNode.nextNode;
        }
    }
```

Seluruh frekuensi karakter dijumlahkan. Frekuensi karakter pertama dijumlahkan dengan frekuensi karakter kedua,kemudian di cek apakah kurang dari atau sama dengan jumlah separuh dari total frekuensi karakter. Apabila masih kurang dari separuh total frekuensi karakter. Maka frekuensi dijumlahkan dengan frekuensi karakter selanjutnya sampai pada batas separuh dari total frekuensi.

Apabila penjumlahan frekuensi melebihi dari separuh total frekuensi dan apabila jumlah frekuensi sebelumnya kurang dari separuh total frekuensi maka frekuensi ditambahkan dengan frekuensi karakter selanjutnya, kemudian penjumlahan frekuensi dihentikan dan node dipisah menjadi dua.

Node pertama yaitu dari karakter pertama sampai pada batas node dipisah menjadi dua dan node kedua adalah sisa dari node. Node pertama di awali dengan nilai 1 dan node kedua diawali dengan nilai 0.

4.2.1.4 Proses Encoding

```
private void createBinCode()
    {
        combineBinCode = new String();
        combineBinCode = "";

        for (int i=0; i<arrayHuruf.length; i++)
        {
            for(currentNode = firstNode; currentNode.asciiChar != arrayHuruf[i];
             currentNode = currentNode.nextNode)
            {

            combineBinCode = combineBinCode + currentNode.binerCode;
        }
              int lenCombineBinCode = combineBinCode.length();
        String addNol = new String();
        headerNol = 7 - (lenCombineBinCode % 7);
```

```
            if (headerNol < 7)
        {
            for (int i=0; i<headerNol; i++)
                addNol = addNol + "0";
        }


else
            headerNol = 0;

        combineBinCode = combineBinCode + addNol;
    }

private void createArrCode()
    {
        char[] bufChar = new char[7];
        int counter, index;
        counter = index = 0;
        int len = combineBinCode.length() / 7;
        arrayCode = new int[len];

for (int i=0; i<combineBinCode.length(); i++)
        {
            if (counter<7)
            {
                bufChar[counter] = combineBinCode.charAt(i);
                counter++;
            }
            if (counter == 7)
              {
                String bufString = new String(bufChar);
                arrayCode[index] = stringBinToInt(bufString);
                index++;
                counter = 0;
              }
}
}
```

Kode biner yang dihasilkan oleh pohon Shannon-Fano digantikan pada setiap karakter penyusun pesan. Sehingga Pesan menjadi susunan kombinasi biner. Dari susunan kombinasi biner tersebut dipecah menjadi 7bit. Apabila pada biner terakhir tidak genap tujuh, maka dilakukan penambahan angka nol sampai biner genap menjadi tujuh. Masing-masing kode biner 7bit tersebut di translasikan ke karakter ASCII.

4.2.1.5 Proses pembuatan header

```
    private void createCompleteCompression()
    {
        String bufStringChar = new String();
        String bufStringFre = new String();
        for(currentNode = firstNode; currentNode != null;

   currentNode = currentNode.nextNode)
        {
            bufStringChar = bufStringChar + currentNode.huruf;
            bufStringFre = bufStringFre + currentNode.frekunsi;
            if (currentNode.nextNode != null)
            {
                bufStringChar = bufStringChar + "-";
                bufStringFre = bufStringFre + "-";
            }

else
            {
                bufStringChar = bufStringChar + ".";
                bufStringFre = bufStringFre + ".";
            }
        }
String bufString = new String();
        bufString = bufString + nodeLength + "." + bufStringChar + bufStringFre
         + headerNol + ".";
        for(int i=0; i<arrayCode.length; i++)
        {
            bufString = bufString + " " + arrayCode[i];
        }
        completeCompression = bufString;
    }
```

Header file berisi banyaknya variasi karakter, variasi karakter, frekuensi karakter, jumlah penambahan angka 0.

4.2.2 Proses Dekompresi

4.2.2.1 Proses Pembentukan Tabel Dekompresi

```
private void parseString()
   {
      int state = 0;  //header jumlah char
      int bufInt, index;
      bufInt = index = 0;
      String arrChar = new String();
      char bufChar;
      totalFrekuensi = 0;
      String bufString = new String();

      for(int i=0; i<stringDecompress.length(); i++)
      {

         if(stringDecompress.charAt(i) == '.')
         {

            if (state == 0)         //header jumlah jenis char
            {
               bufInt = new  Integer(bufString);
               arrayHuruf = new int[bufInt];
               arrayFre = new int[bufInt];
               arrayCode = new String[bufInt];
            }
            else if(state == 1)     //baca char
            {
               bufChar = bufString.charAt(0);
               arrayHuruf[index] = bufChar;
               arrChar += (char)arrayHuruf[index];
            }
            else if(state == 2)     //baca frekuensi
            {
               bufInt = new  Integer(bufString);
               arrayFre[index] = bufInt;
               totalFrekuensi = totalFrekuensi + bufInt;
            }
            else if(state == 3)     //baca header nol
            {
               bufInt = new  Integer(bufString);
               headerNol = bufInt;
            }

           state++;        //change state
            index = 0;
            bufString = new String();
         }
```

```
        else if(stringDecompress.charAt(i) == '-')
        {
           if (state == 1)
           {
              bufChar = bufString.charAt(0);
              arrayHuruf[index] = bufChar;
              arrChar += (char)arrayHuruf[index];
           }
         else if (state == 2)
           {
              bufInt = new  Integer(bufString);
              arrayFre[index] = bufInt;
              totalFrekuensi = totalFrekuensi + bufInt;
           }
           index++;
           bufString = new String();
        }
        else
        {
           bufString = bufString + stringDecompress.charAt(i);
        }
     }

     JOptionPane.showMessageDialog(null, arrChar);
     combineIntCode = bufString;
  }
```

Proses dekompresi diawali dengan pembacaan header pertama yaitu jumlah variasi karakter. Setelah pembacaan header pertama dibuat tabel array yang berisi entitas karakter, frekuensi dan kode biner Shannon fano. Kemudian baca header dua, header dua berisi variasi karakter. Variasi karakter ini diisikan ke tabel array huruf. Kemudian baca header tiga, header tiga berisi jumlah frekuensi masing-masing karakter. Jumlah frekuensi ini diisikan ke tabel array fre.

4.2.2.2 Proses Pembentukan Pohon Shannon Fano

```
private void completingTabel()
{
   int total, ind;
   ind = total = 0;

   for (int i=0; i<arrayFre.length; i++)
   {
      total = total + arrayFre[i];
      if ((total*2) >= totalFrekuensi)
                  {
         break;
      }
      ind++;
   }

   indexHalf = ind;

   left = indexHalf + 1;
   right = arrayFre.length - left;
   String[] arrString = new String[left];
   arrString = createBiner(left);
   for (int i=0; i<left; i++)
   {
      arrayCode[i] = "0" + arrString[i];
   }
   arrString = new String[right];
   arrString = createBiner(right);
   for(int i=0; i<right; i++)
   {
      arrayCode[i+left] = "1" + arrString[i];
   }
}
```

Pohon Shannon fano dibentuk kembali berdasarkan variasi karakter dan jumlah frekuensinya.

4.2.2.3 Proses Dekoding

```
    private void generateOriText()
    {
        JOptionPane.showMessageDialog(null, combineBinCode);
        String bufString = new String();
        boolean beginning = true;
        int len, counter;
        len = counter = 0;
        String binCode, intCode;
        oriText = new String();

        binCode = intCode = "";

        for(int i=0; i<combineBinCode.length(); i++)
        {
            if (beginning)
            {
                bufString = new String();
                if (combineBinCode.charAt(i) == '0')
                {
                    len = cekOrdo(left) + 1;
                }
                else if(combineBinCode.charAt(i) == '1')
                {
                    len = cekOrdo(right) + 1;
                }
                bufString += combineBinCode.charAt(i);
                counter++;
                beginning = false;
            }
if(counter == len)
            {
for (int j=0; j<arrayCode.length; j++)
                {
                    if ((bufString).equals(arrayCode[j]))
                    {
                        binCode += bufString + " ";
                        intCode += arrayHuruf[j] + " ";
                        oriText += (char)arrayHuruf[j];
                        counter = 0;
                        beginning = true;
                        break;
                    }
                }
            }
        }
JOptionPane.showMessageDialog(null,"original text : \n" + oriText
            + "\ncode biner : " + binCode
            + "\ncode angka : " + intCode);
    }
```

Proses decoding dimulai dari translasi hasil kompresi ke kode binernya. Kode biner dari tiap karakter digabungkan dan dicocokkan dengan tabel kompresi. Apabila kode biner cocok dengan biner yang ada di tabel kompresi maka ditranslasikan ke karakter ASCII-nya. Proses berlanjut sampai semua kode biner ditranslasikan.

4.2.3 Proses Kriptografi

4.2.3.1 Proses Pembangkitan Kunci Public (y,g,p)

```
package bikinKunci;
import javax.swing.JOptionPane;

public class bkunci
 {
       private int kuncix;
       private int kunciy;
       private int p = 127;
       private int g = 13;
       public bkunci()
            {
                  this.kuncix = 1;
            }

              public void inputx()
             {
                        String pass =  JOptionPane.showInputDialog(null,"masukkan password
anda dalam text ");
         char [] tamp = pass.toCharArray();
         int pnj = tamp.length;
         for(int i=0;i<pnj;i++)
                        {
                                    int temp = (int)tamp[i];
                                     kuncix = ((kuncix*tamp) % 122)+2 ;
                        }
          System.out.println("password anda adalah "+pass+" kunci privat anda "+kuncix);
                }
public void generateY()
{
           inputx();
           int ky = 0;
           int maks = this.kuncix;
           double hasil = g;
           double rec = 1;
```

```
for(int i=2;i<=maks;i=i+2)
        {
                if((i==2)||(i==4))
                {
                rec = 1;
                hasil = (rec * Math.pow(hasil,2))%p;
                }
                Else
                {
                rec = Math.pow(g, 2)%p;
                hasil = (hasil * rec) % p;
                }

int tempi = i-2;
         }

        if((maks%2)==1)
        {
                hasil = (hasil * g)%p;
                        }
        this.kunciy = (int) hasil;
 }

public void informasi()
{
        generateY();
        JOptionPane.showMessageDialog(null,"kunci public anda adalah "+ this.kunciy +
" kunci private anda adalah "+this.kuncix);
      }
}
```

Pada proses penghitungan kunci *public* (y), diperlukan nilai kunci *private* (x), karena kunci *public* (y) merupakan enkripsi dari kunci *private* (x). Proses penghitungan kunci *public* dilakukan dengan proses *devide and conquer* yaitu memecah proses menjadi proses-proses yang lebih kecil. Proses *devide and conquer* ini dilakukan dengan proses rekursif.

4.2.3.2 Proses Kriptografi (Enkripsi Pesan)

```
// Penghitungan nilai a

package enkripsi;
public class hitungAy
{
   private int k;
   private int p = 127;
   private int g = 13;
   private int hasila;

  public hitungAy()
      {
            k =0;
      }

  public hitungAy(int nilaik)
      {
            this.k = nilaik;
      }

   public void informasi()
      {
             System.out.println("nilai kiriman adalah " +k);
       }
        private void set_a()
      {
            int maks = this.k;
            double hasil = g;
            double rec = 1;
   for(int i=2;i<=maks;i=i+2)
      {
            if((i==2)||(i==4))
            {
            rec = 1;
          hasil = (rec * Math.pow(hasil,2))%p;
            }

else
        {
                rec = Math.pow(g, 2)%p;
                hasil = (hasil * rec) % p;
        }

   int tempi = i-2;
      }
```

```
if((maks%2)==1)
{
        hasil = (hasil * g)%p;
}
        this.hasila = (int) hasil;
        }

    public int get_a (){
        this.set_a();
        return hasila;
    }

}
// Penghitungan nilai b

package enkripsi;
public class hitungB
{
         int y,m,k;
        int p =223;

        public hitungB(int kirY, int kirM ,int kirK)
        {
           this.y = kirY;
           this.m = kirM;
           this.k = kirK;
        }
private double setB()
        {
           double hasil = y;
           int dataku = m;
           double rec = 1;
          int maks = (int) k;
for(int i=2;i<=maks;i=i+2)
        {
            if((i==2)||(i==4))
            {
                        rec = 1;
                         hasil = (rec * Math.pow(hasil,2))%p;
             }
            else
            {
                        rec = Math.pow(y, 2)%p;
                         hasil = (hasil * rec) % p;
             }

                           int tempi = i-2;
        }
```

```
    if((maks%2)==1)
        {
       hasil = (hasil * y)%p;
        }
         hasil = hasil * dataku %p;
          return hasil;
       }

public int getB()
        {
          int kb =0;

         kb =(int) Math.ceil(setB());

          return kb;
       }

       public void informasi()
        {
          System.out.println("Y = "+y + " m = "+m);
       }


}
```

Proses penghitungan nilai a dan b ini adalah proses enkripsi dari tiap karakter pesan. Pada algoritma kriptogrefi Elgamal satu karakter pesan akan dienkripsi menjadi dua karakter yaitu dienkripsi menjadi a dan b. Proses penghitungan nilai adan b ini memiliki kemiripan dengan proses penghitungan nilai y yaitu menggunakan proses *devide and conquer* dengan cara melakukan proses rekursif. Nilai a dan b yang dihasilkan kemudian di rubah menjadi karakter ASCII yang telah disimpan dalam array.

4.2.3.3 Proses Dekripsi

```
public void setDekrip()
{
            int pnjchip = kodechiper.length();
            String plain = "";
             for(int i=0;i<pnjchip;i+=2)
          {
               int j = i;
               Object plainq = (kodechiper.charAt(j));
               j = j+1;
               Object plainq2 = (kodechiper.charAt(j));
              int nila = ceknilai(plainq.toString());
              int nilb = ceknilai(plainq2.toString());
              double v = hitungv(nila);
              int angkami = (int) ((v * nilb) % p);
              String plainteks = daftar[angkami][1];
              plain = plain + "" +plainteks;
          }
            this.plaintext = plain;
        }

        private int ceknilai(String tangkap)
          {
            int hasil = 0;
            for(int i=0;i<daftar.length;i++)
          {
               if(tangkap.equals(daftar[i][1]))
          {
                 hasil = Integer.parseInt(daftar[i][0]);
                  break;
               }
          }
            return hasil;
   }

private double hitungv (int a)
{
      double hasil = a;
      int pangkat = p - x - 1;
      double rec = 1;
       for(int i=2;i<=pangkat;i=i+2)
```

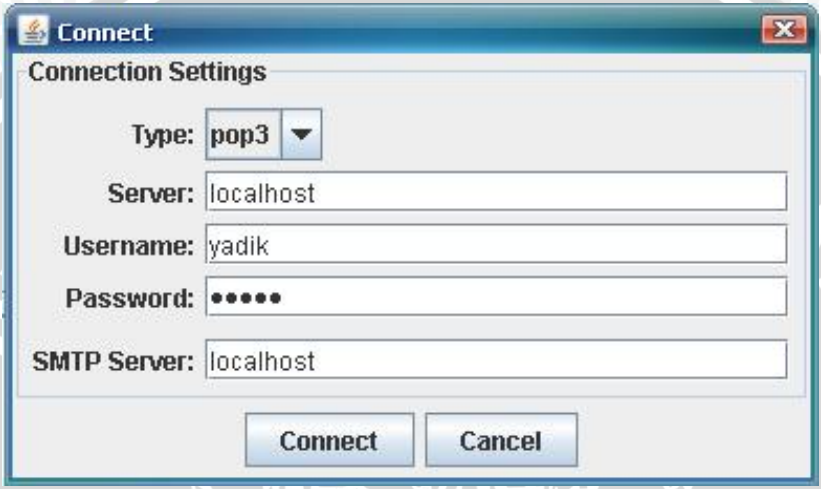
```
        {
        if((i==2)||(i==4))
        {
        rec = 1;
         hasil = (rec * Math.pow(hasil,2))%p;
        }
        Else
        {
        rec = Math.pow(a, 2)%p;
        hasil = (hasil * rec) % p;
        }
    int tempi = i-2;
 }
 if((pangkat%2)==1)
        {
        hasil = (hasil * a)%p;
        }
   return hasil;
}
```

Proses ini diawali dengan mengambil karakter a dan b dari *ciphertext* untuk kemudian dirubah menjadi nomor ASCII-nya, setelah itu baru dilakukan proses dekripsi yang mana inputannya adalah nomor ASCII dari a dan b menjadi satu karakter *plaintext.*

4.3 Antar Muka Aplikasi

4.3.1 Halaman Koneksi

Gambar halaman koneksi dapat dilihat pada gambar 4.1 halaman koneksi.

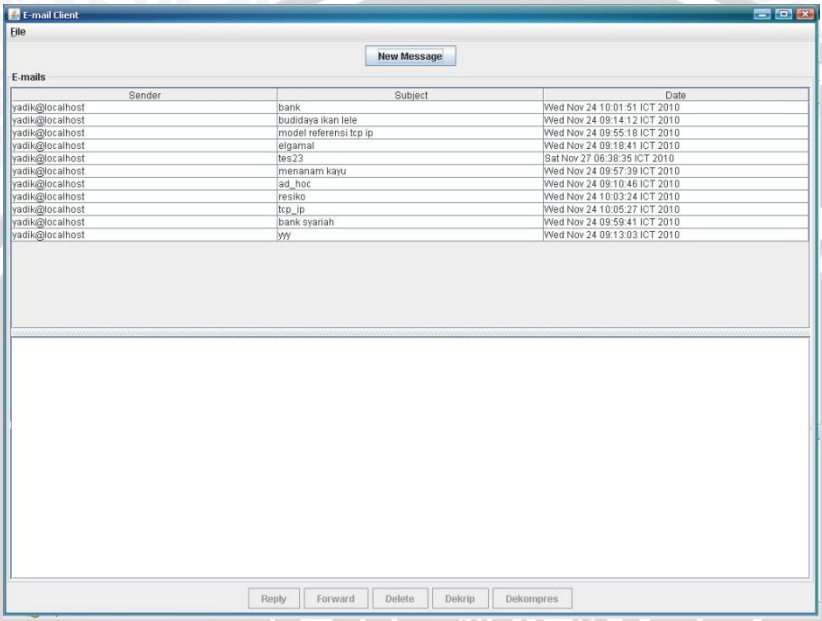


Gambar 4.1 Halaman Koneksi

Pada halaman koneksi ini *user* dapat memilih dua protokol yaitu POP3 ataupun IMAP. Selanjutnya untuk alamat *server* diisi *localhost* begitu juga pada alamat SMTP. *Username* diisi dengan nama *user* dan selanjutnya memasukkan *paswaord* sesuai dengan *password user*.

4.3.2 Halaman *Inbox*

Gambar halaman *inbox* dapat dilihat pada gambar 4.2 halaman *inbox*.

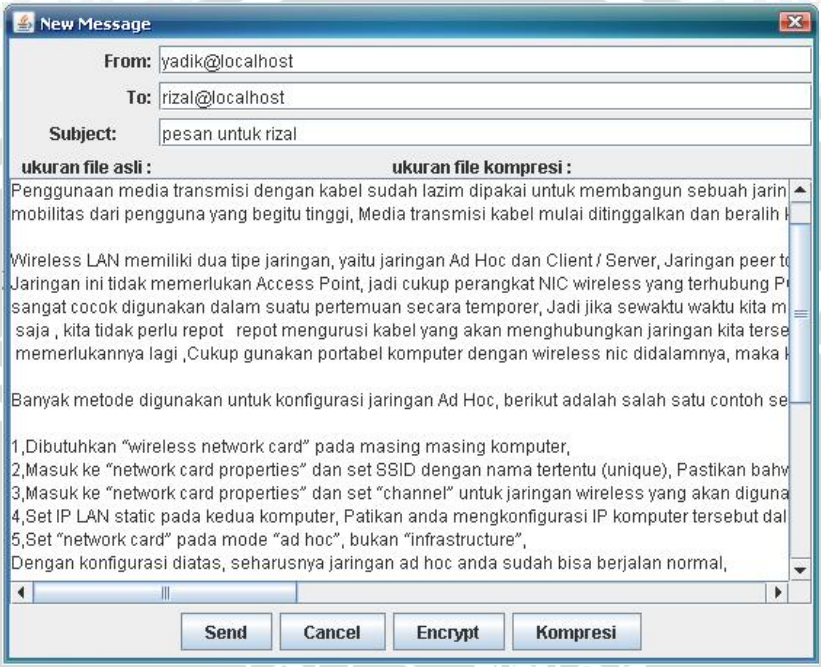


Gambar 4.2 Halaman *Inbox*

Pada halaman ini *user* dapat melihat pesan ataupun mengirim pesan. Apabila *user* akan mengirim pesan *user* dapat mengklik *button New Message* dan apabila user ingin melihat *inbox email*, *user* dapat mengklik baris pada tabel *email*.

4.3.3 Halaman *Create Email* (*New Message*)

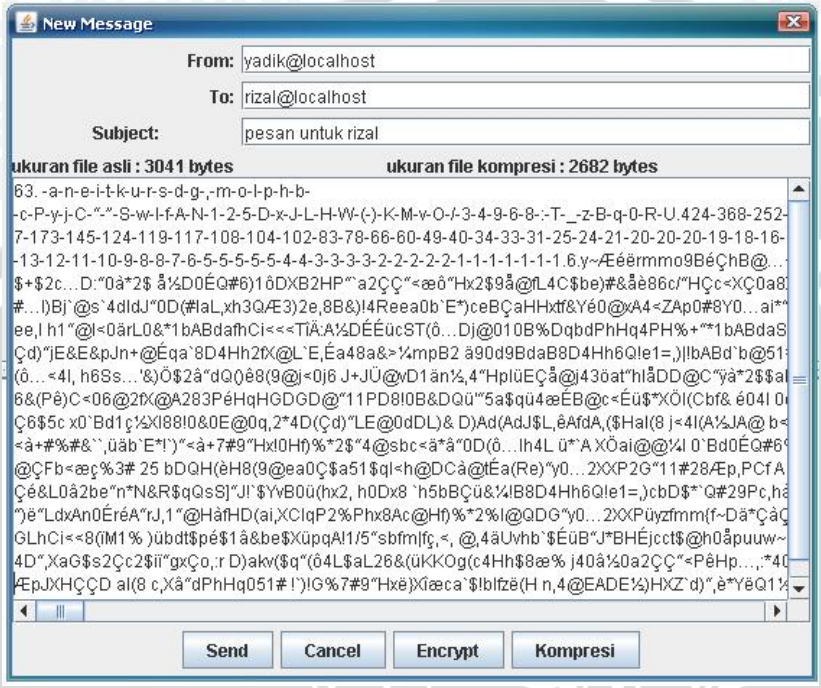
Gambar halaman *create email* (*new message*) dapat dilihat pada gambar 4.3 halaman *create email* (*new message*).



Gambar 4.3 Halaman *Create* Email (*New Message*)

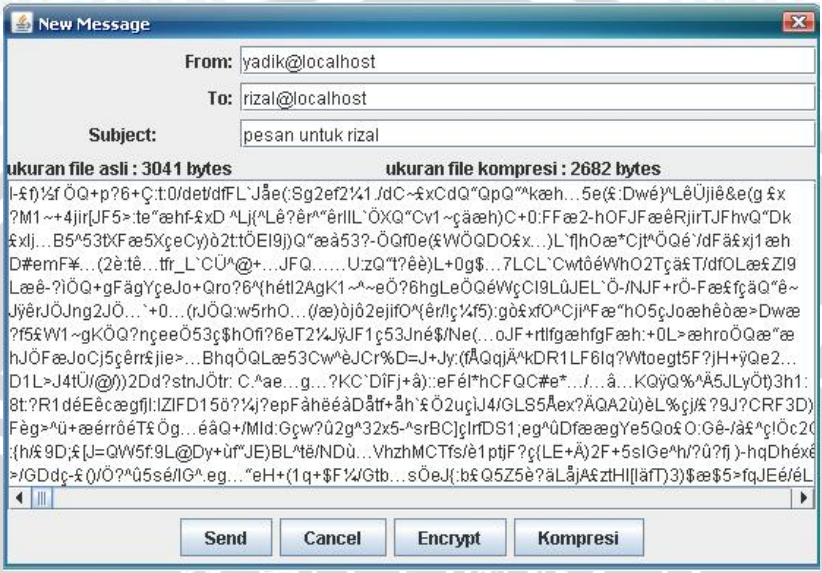
Pada halaman ini *user* memasukkan alamat asal dan alamat tujuan email serta *subject email* (nama email). Setelah itu user dapat menuliskan isi *email*. Setelah email ditulis, *user* dapat mengkompresi pesan tersebut dengan mengklik tombol kompresi untuk menjadi *plaintext* terkompresi. Setelah proses kompresi dilakukan *user* dapat mengenkripsi *plaintext* terkompresi tersebut menjadi *ciphertext*. Kemudian klik *button send* untuk mengirim pesan.

Gambar halaman hasil kompresi dapat dilihat pada gambar 4.4 Halaman Hasil Kompresi.



Gambar 4.4 Halaman Hasil Kompresi

Gambar halaman hasil kompresi dapat dilihat pada gambar 4.5 Halaman Hasil Kompresi.



Gambar 4.5 Halaman Hasil Kompresi

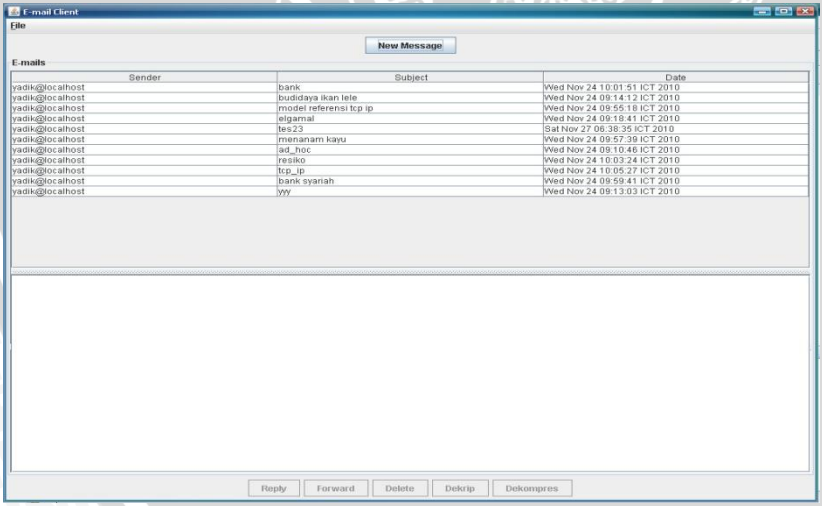
4.3.4 Halaman Koneksi Penerima

Setelah pesan dikirim ,akan dilakukan proses dekripsi dan dekompresi dari *email client* penerima. Seperti halnya pengrim, untuk masuk pada aplikasi *email client* penerima juga harus melakukan *login* (koneksi) dengan menggunakan *username* dan *password*-nya serta *server* menggunakan *localhost*. Gambar halaman koneksi penerima dapat dilihat pada gambar 4.6 Halaman Koneksi Penerima.

Gambar 4.6 Halaman Koneksi Penerima

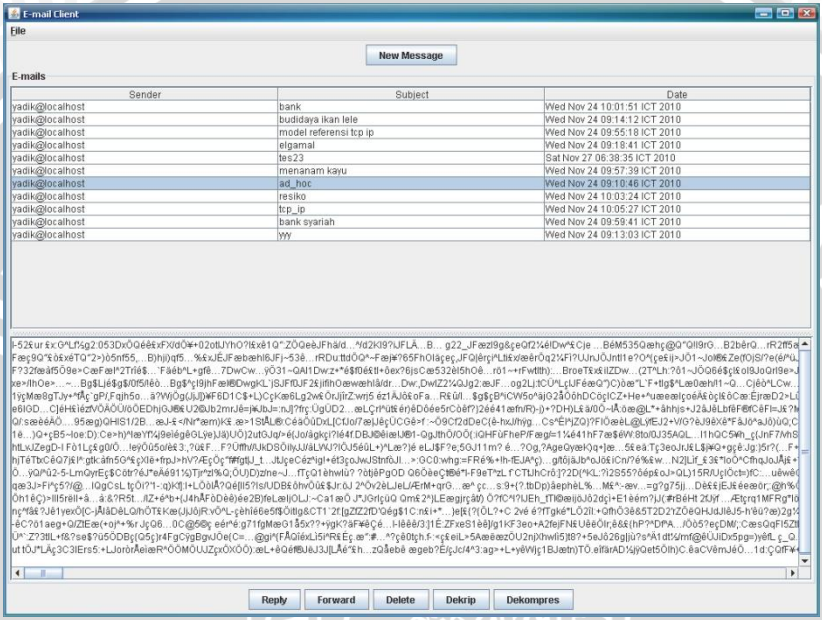
4.3.5 Halaman *Inbox* Penerima

Setelah melakukan koneksi, penerima email selanjutnya penerima akan masuk pada halaman *inbox*. Gambar halaman *inbox* penerima dapat dilihat pada gambar 4.7 Halaman *Inbox* Penerima.



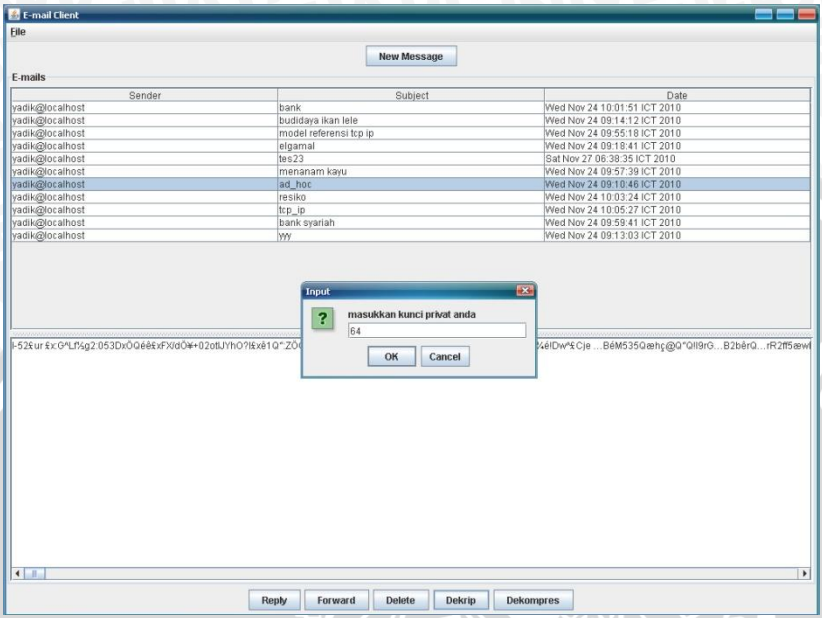
Gambar 4.7 Halaman *Inbox* Penerima

Setelah masuk pada halaman *inbox* klik pesan yang akan dibaca, maka pesan tersebut akan ditampilkan. Akan tetapi pesan yang akan ditampilkan masih dalam bentuk *ciphertext*. Gambar tampilan *email ciphertext* dapat dilihat pada gambar 4.8 Halaman Tampilan *Email Ciphertext*.



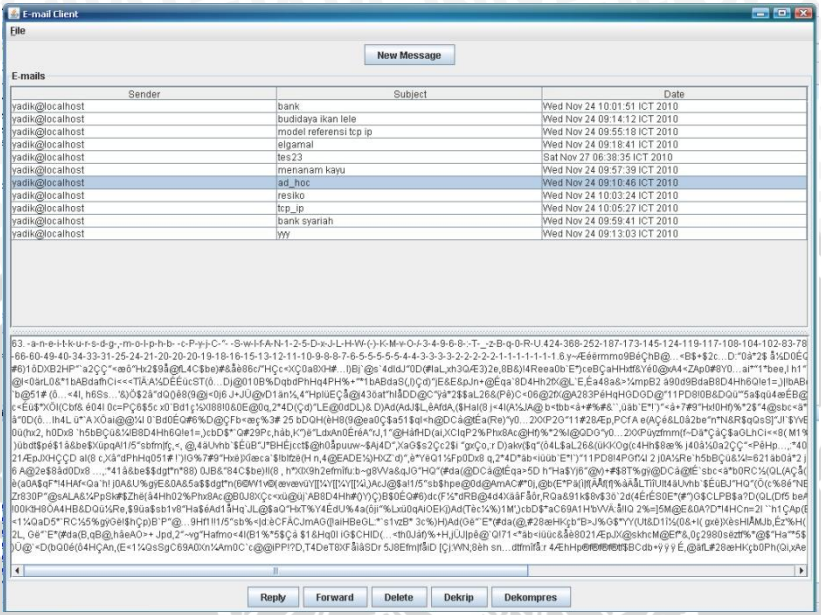
Gambar 4.8 Halaman Tampilan Email *Ciphertext*

Untuk dapat membaca pesan maka penerima *email* harus melakukan proses dekripsi dan dekompresi. Untuk melakukan proses dekripsi maka penerima harus memasukkan kunci *private*-nya. Gambar tampilan *input* kunci *private* dapat dilihat pada gambar 4.9 Halaman Tampilan *Input* Kunci *Private*.



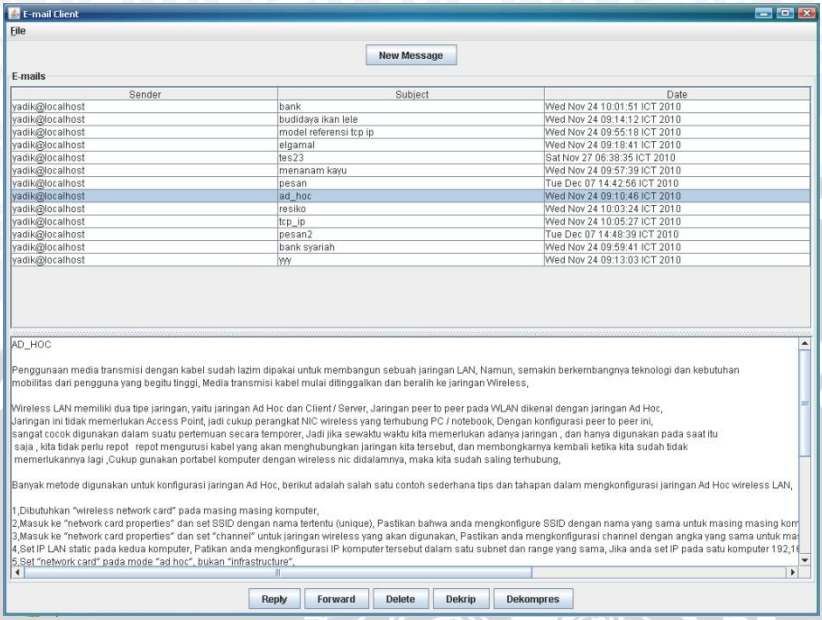
Gambar 4.9 Halaman Tampilan *Input* Kunci *Private*

Setelah pesan *ciphertext* di dekripsi maka *plaintext* terkompresi dapat dilihat. Gambar tampilan *plaintext* terkompresi dapat dilihat pada gambar 4.10 Tampilan *Plaintext* Terkompresi.



Gambar 4.10 Halaman Tampilan *Plaintext* Terkompresi

Setelah *plaintext* terkompresi dapat dilihat, maka langkah selanjutnya mendekompres pesan tersebut untuk menjadi *plantext.* Gambar tampilan *plaintext* dapat dilihat pada gambar 4.11 Tampilan *Plaintext*.



Gambar 4.11 Tampilan *Plaintext*

4.4 Pengujian Algoritma Kompresi Shannon Fano

4.4.1 Hasil pengujian Kompresi Berdasarkan Ukuran (*size*) *file*

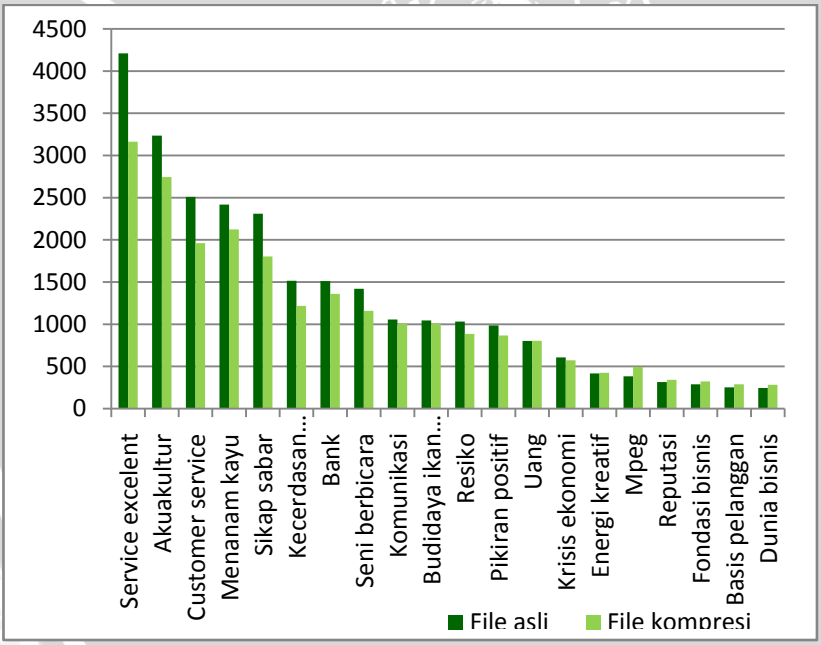
Pada pengujian algoritma Shannon-Fano ini dilakukan dengan menguji data sebanyak dua puluh dokumen. Dari dua puluh dokumen yang diuji, ukuran *file* terbesar yaitu 4210 bytes dan yang terkecil sebesar 245 bytes. Data yang diperoleh dari hasil pengujian dapat dilihat pada **Tabel 4.1** Pengujian Ukuran (*size*)

Tabel 4.1 Pengujian ukuran(*size*)

No.	Nama *File*	Ukuran *File*	Ukuran Setelah Kompresi	Rasio Kompresi
1.	Service excelent	4210 bytes	3164 bytes	75%
2.	Akuakultur	3235 bytes	2744 bytes	85%
3.	Customer service	2509 bytes	1961 bytes	78%
4.	Menanam kayu	2418 bytes	2123 bytes	88%
5.	Sikap sabar	2310 bytes	1803 bytes	78%
6.	Kecerdasan emosi	1504 bytes	1202 bytes	80%
7.	Bank	1511 bytes	1360 bytes	90%
8.	Seni berbicara	1409 bytes	1143 bytes	81%
9.	Komunikasi	1049 bytes	994 bytes	95%
10.	Budidaya ikan lele	1045 bytes	1005 bytes	96%
11.	Resiko	1032 bytes	886 bytes	86%
12.	Pikiran positif	987 bytes	867 bytes	88%
13.	Uang	803 bytes	804 bytes	100.12%
14.	Krisis ekonomi	606 bytes	572 bytes	94%
15.	Energi kreatif	417 bytes	424 bytes	101,7%

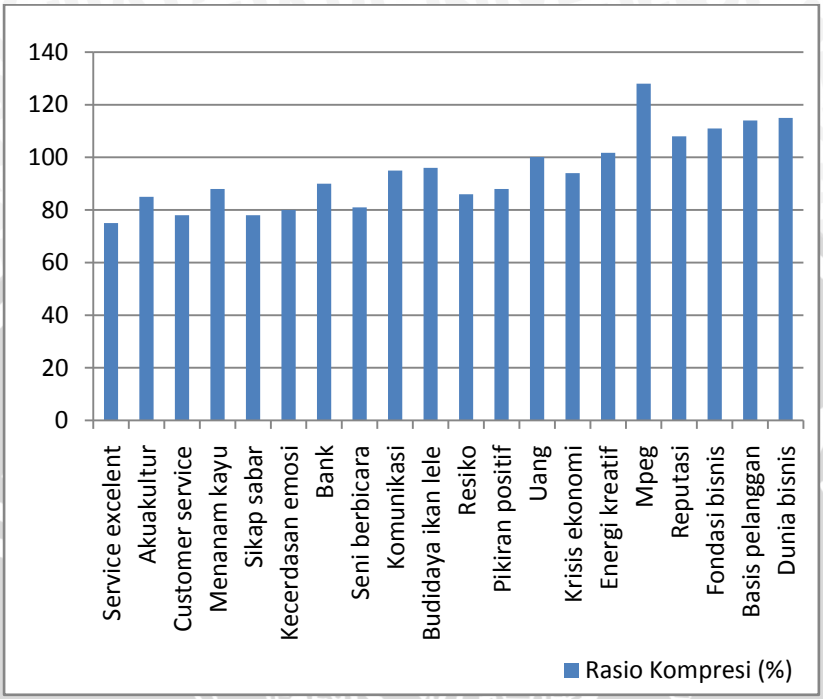
16.	Mpeg	383 bytes	494 bytes	128%
17.	Reputasi	315 bytes	342 bytes	108%
18.	Fondasi bisnis	289 bytes	322 bytes	111%
19.	Basis pelanggan	252 bytes	289 bytes	114%
20.	Dunia bisnis	245 bytes	282 bytes	115%

Dari pengujian data pada sub-bab 4.4.1 maka dapat disusun sebuah grafik yang mengambarkan perbandingan ukuran *file* awal dan , ukuran kompresi untuk setiap file yang diujikan. Grafik perbandingan ukuran *file* dapat dilihat pada **Grafik 4.1** Perbandingan Ukuran *file* asli (*plaintext*) dan *file* kompresi.



Grafik 4.1 Perbandingan Ukuran *file* asli (*plaintext*) dan *file* kompresi

Grafik rasio kompresi dapat dilihat pada **Grafik 4.2** Rasio Kompresi.



Grafik 4.2 Rasio Kompresi

Dari tabel 4.1 dapat kita lihat rasio kompresi terbaik yaitu pada data pertama yaitu data dengan nama "*Service excelent*" yang memiliki ukuran *file* 4210 bytes, ukuran *file* kompresi 3164 bytes dan rasio kompresi sebesar 75%. Rasio kompresi terbaik ke dua didapat pada data ke tiga yaitu data dengan nama "*Customer service*" yang memiliki ukuran *file* 2509 bytes, ukuran file kompresi 1961 bytes dan rasio kompresi sebesar 78%.

Rasio kompresi terjelek yaitu pada data ke enam belas dengan nama "Mpeg" yang memiliki ukuran *file* 383 bytes, ukuran *file* kompresi 494 bytes dan rasio kompresi sebesar 128%. Rasio kompresi terjelek ke dua didapat pada data ke dua puluh dengan nama "Dunia bisnis" yang memiliki ukuran *file* 245 bytes, ukuran *file* kompresi 282 bytes dan rasio kompresi sebesar 115%.

Rata-rata nilai rasio kompresi di atas 417 bytes sampai dengan 4210 bytes = ((75+85+78+88+78+80+90+81+95+96+86+88+100.12 + 94) :14) % = 80.4%

4.4.2 Analisa Hasil Pengujian Algoritma Shannon Fano.

Dari Tabel 4.1 didapatkan rasio kompresi yang sangat bervariasi mulai dari rasio kompresi terbaik yaitu 75% pada data pertama dan rasio terjelek yaitu 128% pada data ke enam belas. Apabila dilihat berdasarkan grafik rasio kompresi didapatkan grafik rasio kompresi yang naik turun. Ini dapat diartikan bahwa ukuran file yang besar belum tentu akan menghasilkan rasio kompresi yang lebih baik dari ukuran *file* yang lebih kecil.

Dapat dilihat pada data ke dua dengan nama "Akuakultur" yang memiliki ukuran *file* sebesar 3235 bytes, ukuran *file* kompresi 2744 bytes dan rasio kompresi sebesar 85% ini tidak lebih besar dari pada data ke tiga dengan nama "*Customer service*" yang memiliki ukuran *file* sebesar 2509 bytes, ukuran *file* kompresi sebesar 1961 bytes dan rasio kompresi sebesar78%. Ini disebabkan karena data ke tiga lebih banyak mempunyai karakter yang sama dari pada data ke dua.

Pada data ke lima belas sampai data ke dua puluh yaitu rentang ukuran *file* 417 bytes sampai 245 bytes didapatkan rasio kompresi minus. Ini dikarenakan beban header data yang besar sehingga menjadikan gabungan header data dan data kompresi lebih besar dari pada ukuran *file* aslinya.

Berdasarkan penjelasan yang telah diuraikan di atas dapat disimpulkan bahwa algoritma kompresi teks Shannon-Fano pada program ini akan efektif apabila data yang akan dikompresi memiliki ukuran lebih dari 417 bytes dan rata-rata rasio kompresinya 80.4%. Algoritma kompresi Shannon-Fano akan lebih optimal apabila dalam ebuah *file* banyak terdapat kesamaan karakter *text.*

4.5 Pengujian Algoritma Kriptografi Elgamal

4.5.1 Hasil Pengujian Enkripsi Berdasarkan Ukuran *file* (*size*)

Pada pengujian algoritma kriptografi Elgamal berdasarkan ukuran *file* dilakukan dengan menguji data sebanyak sepuluh dokumen. Pengujian akan dilakukan dua kali, Data pertama yang akan diuji adalah data *file* asli sebelum dilakukan kompresi dan pengujian kedua dilakukan pada data yang telah dikompresi. Data yang diperoleh dari hasil pengujian enkripsi *file* asli dapat dilihat pada **Tabel 4.2** Pengujian Perbandingan Ukuran *File* Asli Dan *File* Enkripsi (*size*).

Tabel 4.2 Pengujian Perbandingan Ukuran *File* Asli Dan *File* Enkripsi (*size*)

No.	Nama *File*	Ukuran *File*	Ukuran Setelah Enkripsi	Ukuran Setelah Dekripsi
1.	Service excelent	4210 bytes	8420 bytes	4210 bytes
2.	Akuakultur	3235 bytes	6470 bytes	3235 bytes
3.	Customer service	2509 bytes	5018 bytes	2509 bytes
4.	Menanam kayu	2418 bytes	4836 bytes	2418 bytes
5.	Sikap sabar	2310 bytes	4620 bytes	2310 bytes
6.	Kecerdasan emosi	1504 bytes	3008 bytes	1504 bytes
7.	Bank	1511 bytes	3022 bytes	1511 bytes
8.	Seni berbicara	1409 bytes	2818 bytes	1409 bytes
9.	Komunikasi	1049 bytes	2098 bytes	1049bytes
10.	Budidaya ikan lele	1045 bytes	2090 bytes	1045 bytes

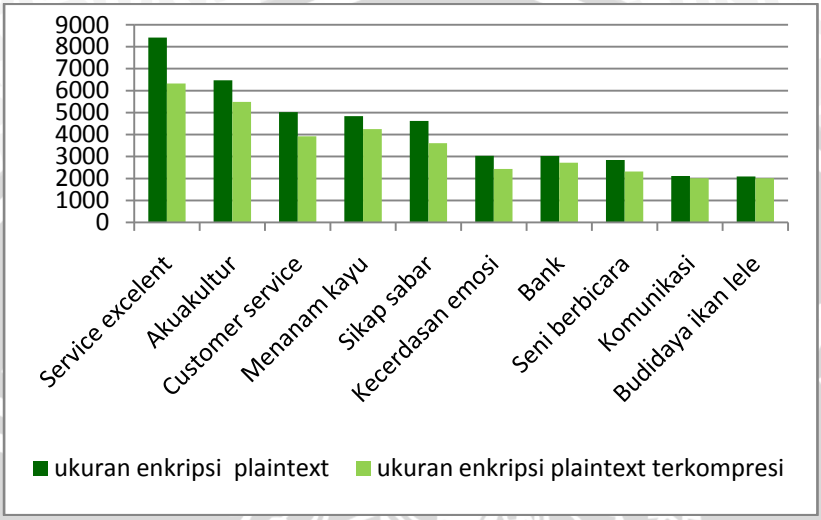
Dari data pada tabel 4.2 kita dapat melihat bahwa ukuran *file* enkripsi akan menjadi dua kali lipat dari ukuran *file* asli. Ini akan menyebabkan *file* hasil enkripsi menyebabkan pemborosan memory. Untuk itu diperlukan kompresi untuk mengecilkan ukuran *file* enkripsi. Data yang diperoleh dari hasil pengujian enkripsi pada *file text* terkompresi dapat dilihat pada **Tabel 4.3** Pengujian Perbandingan Ukuran *File* Kompresi Dan *File* Enkripsi (*size*).

Tabel 4.3 Pengujian Perbandingan Ukuran *File* Kompresi Dan *File* Enkripsi (*size*)

No.	Nama *File*	Ukuran *File* Kompresi	Ukuran Setelah Enkripsi	Ukuran Setelah Dekripsi
1.	Service excelent	3164 bytes	6328 bytes	3164 bytes
2.	Akuakultur	2744 bytes	5488 bytes	2744 bytes
3.	Customer service	1961 bytes	3922 bytes	1961 bytes
4.	Menanam kayu	2123 bytes	4246 bytes	2123 bytes
5.	Sikap sabar	1803 bytes	3606 bytes	1803 bytes
6.	Kecerdasan emosi	1202 bytes	2404 bytes	1217 bytes
7.	Bank	1360 bytes	2720 bytes	1360 bytes
8.	Seni berbicara	1143 bytes	2286 bytes	1158 bytes
9.	Komunikasi	994 bytes	1988 bytes	994 bytes
10.	Budidaya ikan lele	1005 bytes	2010 bytes	1005 bytes

Dari tabel 4.3 dapat kita lihat bahwa ukuran *file* enkripsi dari *plaintext* terkompresi menunjukkan ukuran yang lebih kecil dari pada *file* enkripsi dari *plaintext*. Untuk lebih jelasnya akan dibandingkan ukuran *file* enkripsi pada *plaintext* dan *plaintext* terkompresi. Perbandingan *file* enkripsi *plaintext* dan *file* enkripsi *plaintext*

terkompresi dapat dilihat pada **Grafik 4.4** Perbandingan Ukuran *File* Enkripsi *Plaintext* Dan *File* Enkripsi *Plaintext* Terkompresi.



Grafik 4.4 Perbandingan Ukuran *File* Enkripsi *Plaintext* Dan *File* Enkripsi *Plaintext* Terkompresi

Berdasarkan grafik 4.4 dapat dilihat bahwa ukuran *file* enkripsi dari *plaintext* terkompresi menunjukkan ukuran yang lebih kecil dari pada ukuran file enkripsi *plaintext.*

4.5.2 Analisa Hasil Pengujian Enkripsi Berdasarkan Ukuran *File* (*size*).

Dari tabel 4.2 kita dapat melihat bahwa ukuran *file* asli (*plaintext*) apabila dilakukan proses enkripsi, *ciphertext* akan menjadi dua kali lipat dari ukuran *plaintext*-nya. Ini diakibatkan karena proses enkripsi algoritma kriptografi Elgamal akan menghasilkan dua karakter untuk setiap satu karakter *plaintext*-nya.

Dengan besarnya karakter *ciphertext* yang dihasilkan maka digunakan algoritma kompresi Shannon Fano untuk memperkecil ukuran *file*. Pada Grafik 4.4 kita dapat melihat bahwa apabila digunakan *plaintext* terkompresi sebagai inputan proses enkripsi maka akan terjadi penurunan ukuran *file* enkripsi.

Untuk proses dekripsi, didapatkan ukuran *file* yang sama dengan *file plaintext*, ini dapat diartikan bahwa pada saat dilakukan proses enkripsi-dekripsi tidak terjadi kehilangan informasi.

Berdasarkan penjelasan yang telah diuraikan di atas dapat disimpulkan bahwa algoritma kompresi Shannon Fano efektif untuk mengkompresi *plaintext* dan penggunaan algoritma kompresi Shannon-Fano efektif digabungkan dengan algoritma kriptografi Elgamal.

4.6. Analisis Kekuatan dari Metode Elgamal

Bagian ini, akan dibahas mengenai kekuatan keamanan metode kriptografi Elgamal. Pada skripsi ini digunakan dua cara untuk menyerang kriptografi elgamal yaitu *brute force* dengan mencoba semua kemungkinan kunci dekripsi (ini dilakukan apabila penyerang tidak mengetahui kunci *public*) dan *brute force* untuk mencari nilai yang kongruen dengan nilai y dari kunci *public* (y,g,p)

4.6.1. Serangan *Bruteforce* terhadap Kunci *Private* jika kunci *public* belum diketahui.

Kemungkinan serangan *man in the midle attack* atau seseorang penyadap yang berusaha menangkap pesan selama pesan ditransmisikan dengan tujuan mendapatkan informasi sebanyak-banyaknya mengenai sistem kriptografi yang digunakan untuk berkomunikasi adalah sangat mungkin terjadi jika saluran komunikasi itu tidak aman. Apabila penyadap barhasil mendapatkan pesan rahasia tanpa mengetahui kunci *public* dari pesan tersebut maka penyadap harus melakukan *bruteforce* kunci dekripsi sebanyak kombinasi dari jumlah kemungkinan kunci *private* (x) dan nilai p.

Kemungkinan kunci *private* (x) adalah $1 \leq x \leq p-2$, dalam skripsi ini digunakan nilai p maksimum yaitu 127, jadi nilai terbesar untuk x adalah 125. Kombinasi dari nilai x dan p yaitu sebanyak 125*127 = 15875. Jadi penyadap harus mencoba 15875 kemungkinan kunci. Berdasarkan percobaan yang dilakukan dapat dilihat lama waktu dekripsi pada **Tabel 4.4** Pengujian serangan *brutoforce* terhadap kunci *private* dengan kunci *public* belum diketahui.

Tabel 4.4 Pengujian serangan *brutoforce* terhadap kunci *private* dengan kunci *public* belum diketahui.

No.	Nama file	Ukuran File	Ukuran Setelah Enkripsi	Ukuran Dekripsi	waktu dekripsi+ dekompresi (*second*)	waktu dekripsi 15875 kunci
1.	Service excelent	3164 bytes	6328 bytes	3164 bytes	3.125s+ 5.500s = 8.625s	38jam
2.	Akuakultur	2744 bytes	5488 bytes	2744 bytes	3.17s+ 3.750s = 6.92s	30.5jam
3.	Customer service	1961 bytes	3922 bytes	1961 bytes	5.172s+ 1.812s = 6.984s	30.8jam
4.	Menanam kayu	2123 bytes	4246 bytes	2123 bytes	3.360s+ 1.938s = 5.298s	23.3jam
5.	Sikap sabar	1803 bytes	3606 bytes	1803 bytes	5.547s+ 1.579s = 7.126s	31.4jam
6.	Kecerdasan emosi	1217 bytes	2434 bytes	1217 bytes	5.328s+ 0.703s = 6.031s	26.6jam
7.	Bank	1360 bytes	2720 bytes	1360 bytes	4.156s+ 0.921s = 5.077s	22.4jam
8.	Seni berbicara	1158 bytes	2316 bytes	1158 bytes	4.875s+ 0.625s = 5.5s	24.2jam
9.	Komunikasi	1006 bytes	2012 bytes	1006 bytes	3.406s+ 0.406s = 3.812s	16.8jam
10.	Budidaya ikan lele	1045 bytes	2090 bytes	1045 bytes	6.157s+ 0.422s = 6.579s	29jam

Rata-rata waktu dekripsi + dekompresi 1 kunci = (8.625+6.92 +6.984+5.298+7.126+6.031+5.077+5.5+3.812+6.579):10 = 6.2s

Rata-rata waktu dekripsi + dekompresi 15875 kunci = (38+30.5+30.8+23.3+31.4+ 26.6+22.4+24.2+16.8+29):10=27.3 jam.

Berdasarkan data hasil uji coba *bruteforce* pada tabel 4.9 tikuntuk mendekripsi dan mendekompresi satu *file* dibutuhkan waktu tiga sampai empat detik. Untuk dekripsi menggunakan semua kemungkinan kunci dibutuhkan waktu rata-rata 27.3 jam.

4.6.2. Serangan *Bruteforce* terhadap kunci *private* jika kunci *public* diketahui.

Dalam kasus ini seorang penyadap telah berhasil mendapatkan pesan dan kunci *public*-nya (y,g,p). Maka apabila penyadap ingin mengetahui isi pesan tersebut penyadap harus berhasil menemukan kunci *private* (x) yaitu dengan cara mencari nilai yang kongruen dengan y karena nilai y adalah enkripsi dari kunci *private* (x). Di mana rumus untuk mencari nilai y dapat dilihat pada rumus berikut : $y = g^x \bmod p$

Karena diasumsikan penyadap telah berhasil mendapatkan kunci *public* (y,g,p) maka, penyadap melakukan *bruteforce* sebanyak jumlah kemungkinan nilai x yaitu $1 \leq x \leq p-2$. Dalam skripsi ini digunakan nilai p maksimum = 127, jadi kemungkinan jumlah x tebesar adalah 125.

Sebagai contoh, beberapa *file* uji di enkripsi dengan kunci *public* y,g,p (103,13,127). Untuk dapat mendekripsi pesan, maka penyadap harus bisa mendapatkan nilai x yang apabila dimasukkan ke dalam rumus $y(2)=g^x \bmod p$ menghasilkan bilaingan yang kongruen dengan nilai y. Berdasarkan contoh di atas, maka dicari nilai yang kongruen dengan 103. Untuk percobaan *bruteforce* dapat dilihat pada **Tabel 4.5** Pengujian pencarian kunci *private* (x) menggunakan *bruteforce attack*.

Tabel 4.5 Pengujian Pencarian Kunci Private (x) Menggunakan *Bruteforce Attack*.

No	Nilai x	$y(2)=g^x \bmod p$	$y(2) \equiv y$	No	Nilai x	$y(2)=g^x \bmod p$	$y(2) \equiv y$
1	1	13	Tidak	64	64	13	Tidak
2	2	42	Tidak	65	65	42	Tidak
3	3	38	Tidak	66	66	38	Tidak
4	4	113	Tidak	67	67	113	Tidak
5	5	72	Tidak	68	68	72	Tidak
6	6	47	Tidak	69	69	47	Tidak
7	7	103	Ya	70	70	103	Ya
8	8	69	Tidak	71	71	69	Tidak
9	9	8	Tidak	72	72	8	Tidak
10	10	104	Tidak	73	73	104	Tidak
11	11	82	Tidak	74	74	82	Tidak
12	12	50	Tidak	75	75	50	Tidak
13	13	15	Tidak	76	76	15	Tidak
14	14	68	Tidak	77	77	68	Tidak
15	15	122	Tidak	78	78	122	Tidak
16	16	62	Tidak	79	79	62	Tidak
17	17	44	Tidak	80	80	44	Tidak
18	18	64	Tidak	81	81	64	Tidak
19	19	70	Tidak	82	82	70	Tidak
20	20	21	Tidak	83	83	21	Tidak
21	21	19	Tidak	84	84	19	Tidak
22	22	120	Tidak	85	85	120	Tidak
23	23	36	Tidak	86	86	36	Tidak
24	24	87	Tidak	87	87	87	Tidak
25	25	115	Tidak	88	88	115	Tidak
26	26	98	Tidak	89	89	98	Tidak
27	27	4	Tidak	90	90	4	Tidak
28	28	52	Tidak	91	91	52	Tidak
29	29	41	Tidak	92	92	41	Tidak
30	30	25	Tidak	93	93	25	Tidak
31	31	71	Tidak	94	94	71	Tidak
32	32	34	Tidak	95	95	34	Tidak
33	33	61	Tidak	96	96	61	Tidak

34	34	31	Tidak	97	97	31	Tidak
35	35	22	Tidak	98	98	22	Tidak
36	36	32	Tidak	99	99	32	Tidak
37	37	35	Tidak	100	100	35	Tidak
38	38	74	Tidak	101	101	74	Tidak
39	39	73	Tidak	102	102	73	Tidak
40	40	60	Tidak	103	103	60	Tidak
41	41	18	Tidak	104	104	18	Tidak
42	42	107	Tidak	105	105	107	Tidak
43	43	121	Tidak	106	106	121	Tidak
44	44	49	Tidak	107	107	49	Tidak
45	45	2	Tidak	108	108	2	Tidak
46	46	26	Tidak	109	109	26	Tidak
47	47	84	Tidak	110	110	84	Tidak
48	48	76	Tidak	111	111	76	Tidak
49	49	99	Tidak	112	112	99	Tidak
50	50	17	Tidak	113	113	17	Tidak
51	51	94	Tidak	114	114	94	Tidak
52	52	79	Tidak	115	115	79	Tidak
53	53	11	Tidak	116	116	11	Tidak
54	54	16	Tidak	117	117	16	Tidak
55	55	81	Tidak	118	118	81	Tidak
56	56	37	Tidak	119	119	37	Tidak
57	57	100	Tidak	120	120	100	Tidak
58	58	30	Tidak	121	121	30	Tidak
59	59	9	Tidak	122	122	9	Tidak
60	60	117	Tidak	123	123	117	Tidak
61	61	124	Tidak	124	124	124	Tidak
62	62	88	Tidak	125	125	88	Tidak
63	63	1	Tidak				

Dari hasil percobaan pada tabel 4.10 didapat dua buah bilangan yang kongruen dengan y, yaitu 7 dan 70. Dari dua bilangan kongruen yang didapat, dilakukan proses dekripsi dengan dua kunci tersebut. Lama waktu proses dekripsi dapat dilihat pada **Tabel 4.6** Lama Waktu Dekripsi Menggunakan Dua Kunci.

Tabel 4.6 Lama Waktu Dekripsi Menggunakan Dua Kunci.

No.	Nama file	Ukuran File	Ukuran Setelah Enkripsi	Ukuran Dekripsi	waktu dekripsi+ dekompresi (second)	waktu dekripsi 2 kunci (second)
1.	Service excelent	3164 bytes	6328 bytes	3164 bytes	8.625s	17.25s
2.	Akuakultur	2744 bytes	5488 bytes	2744 bytes	6.92s	13.84s
3.	Customer service	1961 bytes	3922 bytes	1961 bytes	6.984s	13.968s
4.	Menanam kayu	2123 bytes	4246 bytes	2123 bytes	5.298s	10.596s
5.	Sikap sabar	1803 bytes	3606 bytes	1803 bytes	7.126s	14.252s
6.	Kecerdasan emosi	1202 bytes	2404 bytes	1202 bytes	6.031	12.062s
7.	Bank	1360 bytes	2720 bytes	1360 bytes	5.077s	10.154s
8.	Seni berbicara	1143 bytes	2286 bytes	1143 bytes	5.5s	11s
9.	Komunikasi	994 bytes	1988 bytes	994 bytes	3.812s	7.624s
10.	Budidaya ikan lele	1005 bytes	2010 bytes	1005 bytes	6.579s	13.158s

Rata-rata waktu dekripsi + dekompresi 2 kunci = (17.25+13.84+13.968+10.596+14.252+12.062+10.154+11+7.624+ 13.158) :10 = 12.39 s.

Dengan menggunakan pencarian nilai y(2) yang kongruen dengan kunci *public* y maka, proses dekripsi pesan akan jauh lebih singkat yaitu dari rata-rata waktu yang dibutuhkan 27.3jam menjadi 12.39 detik. Hal ini dikarenakan pencarian nilai y(2) yang kongruen dengan nilai kunci *public* y akan memperkecil jumlah kemungkinan kunci yang akan dicoba untuk melakukan proses dekripsi.

4.6.3 Analisa Hasil Pengujian Enkripsi Berdasarkan Ukuran *File* (*size*).

Proses dekripsi pada dua percobaan *bruteforce*, baik untuk *bruteforce* terhadap kunci *private* dari kunci *public* yang belum ataupun sudah diketahui ini masih belum aman, ini disebabkan karena jumlah karakter kunci *public* (p) yang digunakan adalah 2^7 karakter ASCII. Penggunaan kunci *public* (p) sebanyak 2^7 karakter ASCII ini dikarenakan berdasarkan percobaan yang dilakukan untuk karakter ASCII 2^8 tidak semua dapat dikirim menggunakan email *client*. Begitu juga dengan karakter *unicode*.

Proses dekripsi akan lebih lama dan lebih efektif apabila rentang kunci *public* (p) (huruf ASCII yang digunakan untuk enkripsi) adalah sebanyak karakter *unicode* sebanyak 2^{16} karakter atau sebanyak 65.536 karakter. Untuk jumlah kunci *public* (p) sebanyak 65.536 maka banyaknya jumlah kunci *private* (x) , yang mana jumlah kunci *private* (x) adalah $1 \leq x \leq p-2$ yaitu sebanyak 65.534 kemunkinan.

Dengan menggunakan kunci *public* (p) sebanyak 2^{16} karakter, maka untuk dapat mendekripsi dan mendekompresi pesan secara *bruteforce* dibutuhkan kombinasi kunci 65.536 x 65.534 = 4.294.836.224 kemungkinan. Untuk mendekripsi *file* tanpa mengetahui kunci *public* dibutuhkan waktu = (rata-rata waktu dekripsi+dekomresi 1 kunci) 6.2s*4.294.836.224 = 26627984588.8s = 865 tahun.

Berdasarkan penjelasan di atas dapat disimpulkan bahwa penggunaan karakter kunci *public* (p) sebanyak 2^7 karakter ASCII ini belum aman, akan lebih aman apabila digunakan 2^{16} karakter *unicode*.

Dari percobaan di atas dapat disimpulkan baik untuk *brute force* terhadap kunci *private* dari kunci *public* yang belum ataupun sudah diketahui akan lebih aman jika kunci *public* (p) menggunakan karakter 2^{16} *unicode*.

BAB V
PENUTUP

5.1 KESIMPULAN

Kesimpulan yang didapat dari pelaksanaan kegiatan penulisan skripsi ini adalah :

1. Iimplementasi algoritma kompresi Shanon Fano pada *plaintext* menghasilkan *file* yang memilki ukuran lebih kecil dan *file* yang dihasilkan dapat dilakukan proses dekompresi.
2. Implementasi algoritma kriptografi Elgamal pada pesan (*plaintext* terkompresi) dengan kunci *public* menghasilkan *ciphertext* yang dapat dilakukan proses dekripsi.
3. Aplikasi *email client* yang telah dibuat dapat mengirim *email* yang sudah dikompresi,dienkripsi dan dapat melakukan proses dekripsi dan dekompresi .

5.2 SARAN

Saran yang bisa disampaikan dari penulisan skripsi ini adalah :

1. Aplikasi penerapan algoritma kriptografi Elgamal dan algoritma kompresi Shannon Fano dalam tulisan ini baru diterapkan pada file text saja, sehingga untuk pengembangan lebih lanjut lagi diperlukan untuk melakukan pengujian terhadap file gambar (BMP, JPG, GIF, dsb), file video dan audio (mp3, avi, mp4, mpeg, dsb) sehingga perbandingan pengujian menjadi lebih bervariasi.
2. Dalam penerapan algoritma Elgamal akan lebih aman apabila digunakan 2^{16} kerakter *Unicode*. Dalam percobaan yang dilakukan pada skripsi ini belum digunakan karakter *Unicode* karena keterbatasan *resource* program yang digunakan tidak mensupport sehingga diharapkan pengembangannya menggunakan 2^{16} karakter *unicode*.

DAFTAR PUSTAKA

Brain, Marshall .2009.*How E-mail Works*. www.howstuffworks.com. di akses pada bulan November 2009.

Fauzi, Rahmad. 2003. *Analisis Beberapa Teknik Coding*. Sumatera Utara. Universitas Sumatera Utara. Teknik Elektro.

Febrian, Jack. 2007. *Data Digital*.www. artikel.total.or.id. di akses pada bulan November 2009.

Jaya, A. 2003. *Penyandian data dengan algoritma kriptografi noekeon*.http://www.cert.or.id/~budi/courses/ec7010/2003/paper-imadeari.pdf.Diakses pada tanggal 7 Januari 2008.

Linawati, dan Panggabean, Henry. 2004. *Perbandingan Kinerja Algoritma Huffman, LZW dan DMC Pada Berbagai Tipe File*. Bandung. Universitas Katolik Parahyangan. Ilmu Komputer.

Menezes, Oorcshot, and Vanstone, 1996, *Handbook of Applied Cryptography*, CRC Press, Inc. USA.

Munir, Renaldi. 2006. *Kriptografi*. Bandung. Informatika.

Raharjo, Budi. 2005. Keamanan Sistem Informasi Berbasis Internet. Bandung. PT. Insan Infonesia.

Rhee, M.Y. (2003). *Cryptograhpy Principles, Algorithms and Protocol*, John Wiley and Sons Inc Richardson, I. (2007). *An Overview of H.264 Advanced Video Coding*, terdapat di www.vcodex.com di akses pada bulan September 2005.

Riyanto, Zaki, Muhammad. 2007. *Pengamanan Pesan Rahasia Menggunakan Algoritma Kriptografi Egamal Atas Grup Pergandaan Zp**. Universitas Gajah Mada. Matematika. Yogyakarta.

Stallings, William. 2003.*Criptography And Network Security*. New Jersey. Pearson Education.Inc.

Sagita, A. 2005. *Simulasi Penerapan Metoda Elliptic Curve Cryptography (Ecc) Untuk Mengatasi Kelemahan Sistem Keamanan Jaringan Gsm.* STT Telkom. Bandung.

Tino, D. "Pengantar Teknologi Informasi – Pengkodean". http://www.dwiantoro.com/documents/Modul_5_PTI.pdf. Tanggal akses : 21 Oktober 2009.

Wiryadinata, Romi. *Data Compression Coding Using Static And Dynamic Methode Of Shannon-Fano Algorithm*. Yogyakarta. Universitas Gajah Mada.

LAMPIRAN 1

TABEL ASCII 7 BIT

Desimal	Simbol	Desimal	Simbol
0	NUL	30	RS
1	SOH	31	US
2	STX	32	
3	ETX	33	!
4	EOT	34	”
5	ENQ	35	#
6	ACK	36	$
7	BEL	37	%
8	BS	38	&
9	HT	39	’
10	LF	40	(
11	VT	41	)
12	FF	42	*
13	CR	43	+
14	SO	44	,
15	SI	45	-
16	DLE	46	.
17	DC1	47	/
18	DC2	48	0
19	DC3	49	1
20	DC4	50	2
21	NAK	51	3
22	SYN	52	4
23	ETB	53	5
24	CAN	54	6
25	EM	55	7
26	SUB	56	8
27	ESC	57	9
28	FS	58	:
29	GS	59	;

60	<	94	^
61	=	95	_
62	>	96	`
63	?	97	a
64	@	98	b
65	A	99	c
66	B	100	d
67	C	101	e
68	D	102	f
69	E	103	g
70	F	104	h
71	G	105	i
72	H	106	j
73	I	107	v
74	J	108	l
75	K	109	m
76	L	110	n
77	M	111	o
78	N	112	p
79	O	113	q
80	P	114	r
81	Q	115	s
82	R	116	t
83	S	117	u
84	T	118	v
85	U	119	w
86	V	120	x
87	W	121	y
88	X	122	z
89	Y	123	{
90	Z	124	|
91	[	125	}
92	\	126	~
93	]	127	DEL

LAMPIRAN 2

TABEL ASCII 7 BIT (Modifikasi)

Desimal	Simbol	Desimal	Simbol
0		30	>
1	!	31	?
2	”	32	@
3	#	33	A
4	$	34	B
5	%	35	C
6	&	36	D
7	’	37	E
8	(	38	F
9	)	39	G
10	*	40	H
11	+	41	I
12	,	42	J
13	-	43	K
14	.	44	L
15	/	45	M
16	0	46	N
17	1	47	O
18	2	48	P
19	3	49	Q
20	4	50	R
21	5	51	S
22	6	52	T
23	7	53	U
24	8	54	V
25	9	55	W
26	:	56	X
27	;	57	Y
28	<	58	Z
29	=	59	[

60	**G**	94	**Í**
61	^	95	**®**
62	_	96	**Ç**
63	`	97	**Ü**
64	**A**	98	**É**
65	**B**	99	**Â**
66	**C**	100	**Ä**
67	**D**	101	**À**
68	**E**	102	**Å**
69	**F**	103	**Ç**
70	…	104	**Ê**
71	**H**	105	**Ë**
72	**I**	106	**È**
73	**J**	107	**Ï**
74	V	108	**Î**
75	**L**	109	**Ì**
76	**M**	110	**Ä**
77	**N**	111	**Å**
78	**O**	112	**É**
79	**P**	113	**Æ**
80	**Q**	114	**Æ**
81	**R**	115	**Ô**
82	**S**	116	**Ö**
83	**T**	117	**Ò**
84	**U**	118	**Û**
85	**V**	119	**Ù**
86	**W**	120	**¯**
87	**X**	121	**Ö**
88	**Y**	122	**Ü**
89	**Z**	123	**Á**
90	**{**	124	**£**
91	|	125	**¥**
92	**}**	126	**¯**
93	**~**	127	½


UNIVERSITAS BRAWIJAYA