Enkripsi Dan Dekripsi *File* Dengan Algoritma *Blowfish*

Andri Saputra[1], Andika Widyanto[2]
Program Studi Teknik Informatika, STMIK PALCOMTECH Palembang
STMIK PALCOMTECH, Jl. Basuki Rahmat No.5 Palembang
e-mail : andri.saputra182@gmail.com[1], lorickdikz@gmail.com[2]

Abstract

A Belgian company usually associated with important data that is confidential to safeguard data, problems that often occur when this is essential as well as theft of data leak data conducted by parties who are not responsible, or retrieve the data without the express permission of the owner. This often happens at companies where there is company confidential data that needs safeguarding data so that it is not misused. Of these problems created a system of encryption and decryption of files using Blowfish Algorithm in securing data or information in the form of a file with encrypt plaintext ciphertext files into a file, so that it cannot be read or understood by others, as well as the data can not be misused and manipulated. These applications can serve as objectives, namely to secure data or information in the form of a file with encrypt plaintext file into ciphertext files, nor vice versa.

***Keywords* :** *blowfish algorithm, plaintext, ciphertext, enkription, dekription*

Abstrak

Sebuah perusaahan biasanya terkait dengan data-data penting yang sifatnya rahasia yang perlu pengamanan data, permasalahan yang sering terjadi saat ini adalah pencurian data penting serta mengalami kebocoran data yang dilakukan oleh pihak-pihak yang tidak bertanggung jawab, atau mengambil data tanpa seizin pemiliknya. Hal ini sering terjadi pada perusahaan-perusahaan dimana terdapat data-data rahasia perusahaan yang perlu pengamanan data sehingga tidak disalah gunakan. Dari permasalahan tersebut dibuat sebuah sistem enkripsi dan dekripsi file menggunakan Algoritma *Blowfish* dalam mengamankan data atau informasi yang berupa *file* dengan mengenkripsi *plaintext file* menjadi sebuah *ciphertext file*, sehingga tidak dapat dibaca atau dimengerti oleh orang lain, serta data tersebut tidak dapat disalah gunakan dan dimanipulasi. Aplikasi ini dapat berfungsi seperti tujuan, yaitu untuk mengamankan data atau informasi yang berupa *file* dengan mengenkripsi *plaintext file* menjadi *ciphertext file*, begitu juga sebaliknya.

Kata kunci : algoritma *blowfish, plaintext, ciphertext*, enkripsi, dekripsi

1. Pendahuluan

Dunia dan teknologi informasi masa kini semakin berkembang, semua informasi terkirim dengan bebas melalui suatu jaringan dengan tingkat keamanan yang relatif rendah. Untuk itulah peranan teknologi khususnya dalam keamanan informasi benar-benar dibutuhkan. Keamanan informasi merupakan bagian dari sebuah sistem di dalam sebuah jaringan komputer terutama yang terhubung dengan internet. Oleh karena itu, informasi penting atau data rahasia yang dikirim harus disandikan agar tidak dapat dibaca oleh orang lain.

Sebuah perusaahan biasanya terkait dengan data-data penting yang sifatnya rahasia yang perlu pengamanan. Permasalahan yang sering terjadi saat ini adalah pencurian data penting serta mengalami kebocoran data yang dilakukan oleh pihak-pihak yang tidak bertanggung jawab, atau mengambil data tanpa seizin pemiliknya. Hal ini sering terjadi pada perusahaan-perusahaan dimana terdapat data-data rahasia perusahaan yang perlu pengamanan data sehingga tidak disalah gunakan. Dari permasalahan tersebut peneliti tertarik untuk membuat sebuah aplikasi enkripsi dan dekripsi file menggunakan algoritma *blowfish* dengan tujuan untuk mengamankan data atau informasi yang berupa *file* dengan mengenkripsi *plaintext file* menjadi *ciphertext file*, begitu juga sebaliknya, sehingga isi dari *file*/*plaintext file* tersebut tidak dapat dibaca atau dimengerti oleh orang lain, serta tidak dapat disalah gunakan dan dimanipulasi.

Penelitian terdahulu, disumber lainnya[1] dihasilkan bahwa Algoritma kriptografi AES Rijndael adalah algoritma kriptografi yang cukup handal hingga saat ini. Pada tahun 2006, National Security Agency(NSA) pernah menyatakan bahwa AES cukup aman digunakan untuk mengamankan data-data pemerintah Amerika Serikat yang bukan tergolong sangat rahasia. Hingga tahun 2006 serangan terbaik terhadap algoritma Rijndael hanya berhasil menembus putaran ke-7 untuk kunci 128 bit, putaran ke-8 untuk kunci 192 bit, dan putaran ke-9 untuk kunci 256 bit. Dengan melihat jumlah putaran yang berhasil ditembus, tidaklah tidak mungkin suatu hari algoritma ini dapat dengan mudah ditembus. Namun demikian algoritma Rijndael masih dipandang algoritma yang cukup handal.

Disumber lainnya[2] dihasilkan aplikasi untuk dapat mengetahui proses enkripsi dan dekripsi menggunakan algoritma elgamal. Algoritma elgamal juga bisa dimanfaatkan untuk mengirimkan sebuah pesan yang sifatnya sangat rahasia, yaitu dengan kunci dari sebuah kriptografi simetris. Pada algoritma elgamal suatu *plaintext* yang sama akan dienkripsi menjadi *ciphertext* yang berbeda-beda. Hal ini dikarenakan pemilihan bilangan k yang acak. Akan tetapi, walaupun *ciphertext* yang diperoleh berbeda-beda pada proses dekripsi akan diperoleh *plaintext* yang sama.

Disumber lainnya[3] dihasilkan dimana keamanan informasi yang tersimpan dalam database dapat ditingkatkan dengan adanya enkripsi yang diimplementasikan dalam suatu sistem informasi, informasi yang bersifat rahasia hanya dapat dibaca oleh pihak yang berkepentingan melalui proses enkripsi.

Disumber lainnya[4] dihasilkan dengan adanya cara pengamanan ini, pengembang aplikasi yang menggunakan bahasa pemograman PHP dapat menyembunyikan skrip php supaya tidak mudah disalin, diubah sebagian/seluruhnya oleh orang yang tidak berhak, integritas dari aplikasi yang telah dienkripsi akan lebih terjaga, karena skrip yang sudah dienkripsi tidak dapat diubah.

Disumber lainnya[5] dihasilkan kriptografi *password* dienkripsi dan didekripsi menggunakan modifikasi metode *affine cipher*. Karakter *password* yang dapat dienkripsi dan didekripsi yaitu alfabet dan bilangan asli. Kriptografi *password* merupakan perangkat lunak untuk mengenkripsi dan dekripsi sebuah *password* agar tidak dapat dibaca oleh orang yang tidak berhak walaupun dengan melihat kode sumber programnya.

Disumber lainnya[6] dihasilkan aplikasi yang berfungsi untuk merubah sebuah data elektronik menjadi sandi-sandi yang tidak dapat dibaca dengan metode kriptografi asimetris sehingga kerahasiaannya dapat dijaga. data hasil enkripsi sangat sulit untuk dimengerti dan diterjemahkan, karena banyaknya operasi logika yang harus dilewati serta algoritma yang dibuat masih belum terpublikasi secara umum.

2. Tinjauan Pustaka

Blowfish sendiri menggunakan metode *cipher* blok, dimana selama proses enkripsi dan dekripsi, *blowfish* bekerja dengan membagi pesan-pesan menjadi blok-blok bit dengan ukuran sama panjang yaitu 64-bit dengan panjang kunci bervariasi yang mengenkripsikan data ke dalam 8 *byte* blok. Pesan yang bukan merupakan kelipatan 8 *byte* akan ditambahkan dengan bit-bit tambahan (*padding)* sehinggan ukuran tiap blok sama. Algoritma Blowfish terdiri atas dua bagian, yaitu ekspansi kunci dan enkripsi data[7]:

a. Ekspansi Kunci (*Key Expansion*)

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

$K_{u1},K_{u2},.....,K_{uj}$ $1 \le j \le 14$

Kunci-kunci ini yang dibangkitkan (*generate*) dengan menggunakan subkunci yang harus diitung terlebih dahulu sebelum enkripsi atau dekripsi data. Sub-sub kunci yang digunakan terdiri dari :

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

$P_1,P_2,.....,P_{18}$

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

$S_{1,0},S_{1,1},....,S_{1,255}$

$S_{2,0},S_{2,1},....,S_{2,255}$

$S_{3,0},S_{3,1},....,S_{3,255}$

$S_{4,0},S_{4,1},....,S_{4,255}$.

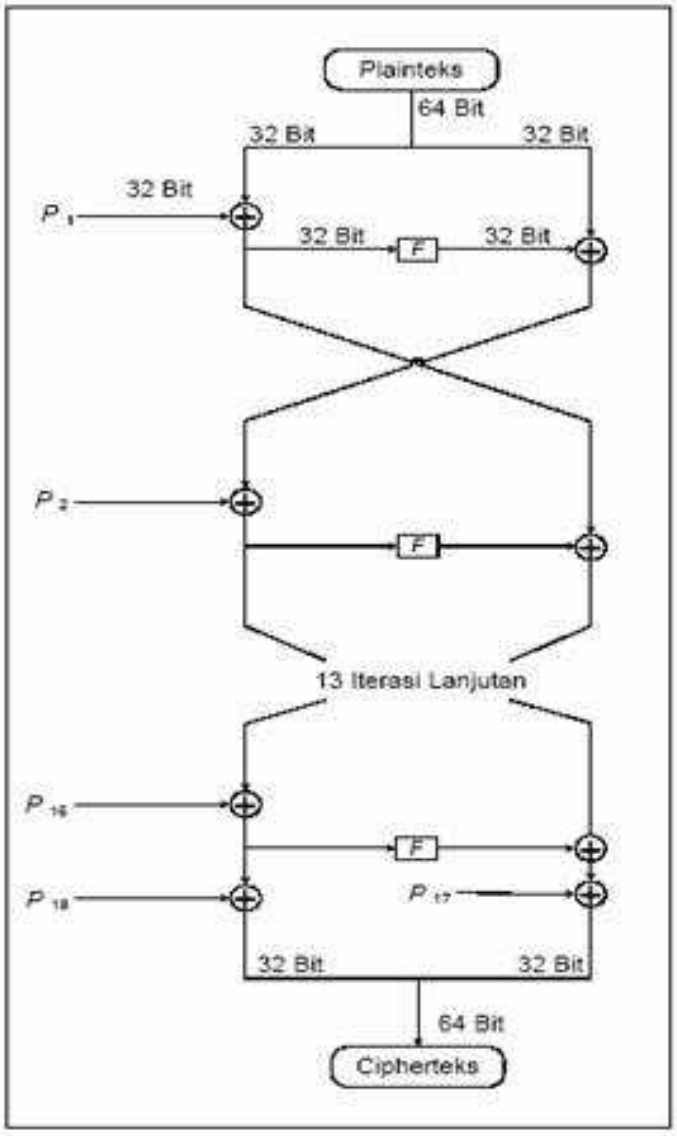
b. Enkripsi Data

Terdiri dari iterasi fungsi sederhana (*Feistel Network*) sebanyak 16 kali putaran (iterasi), masukannya adalah bit elemen data X. Semua operasi adalah penambahan dan XOR pada variabel 32-bit.

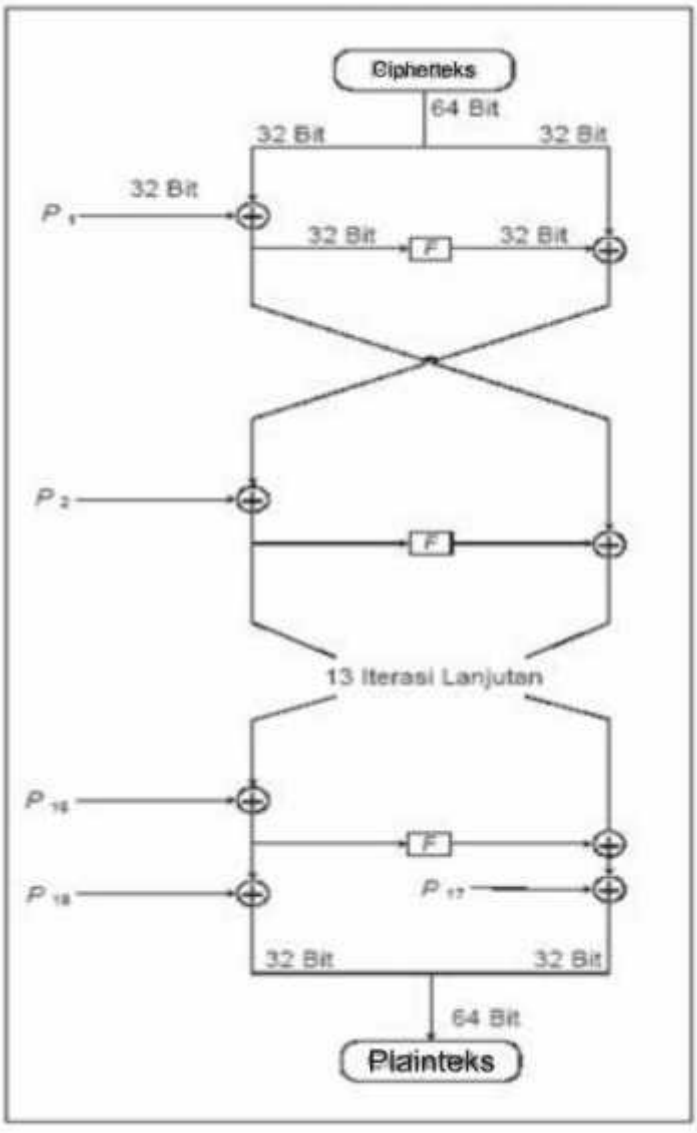
Operasi tambahan lainnya hanyalah empat penelusuran tabel *array* berindeks untuk setiap putaran. Langkahnya adalah seperti berikut :

1. *Plaintext* yang akan dienkripsi diasumsikan sebagai X, X tersebut diambil sebanyak 64-bit, dan apabila kurang dari 64-bit maka kita tambahkan bitnya, supaya dalam operasi nanti sesuai dengan datanya.
2. Bagi X menjadi dua bagian yang masing-masing terdiri dari 32-bit yaitu X_L dan X_R. For i = 1 to 16 :
   (a) $X_L = X_L$ Xor P(i).
   (b) $X_R = F(X_L)$ Xor X_R.
   (c) Tukar $X_L$dan X_R.
3. Setelah iterasi keenam belas, tukar $X_L$dan X_R, lagi untuk membatalkan pertukaran terakhir.
4. Lalu lakukan :
   (a) $X_R = X_R$ Xor P_{17}
   (b) $X_L = X_L$ Xor P_{18}
5. Gabungkan kembali X_L dan X_R untuk mendapatkan *ciphertext*.

Untuk lebih jelasnya berikut gambaran tahapan pada jaringan feistel (*feistel network*) yang digunakan *Blowfish.*



Gambar 1 Blok Diagram Enkripsi Algoritma *Blowfish*



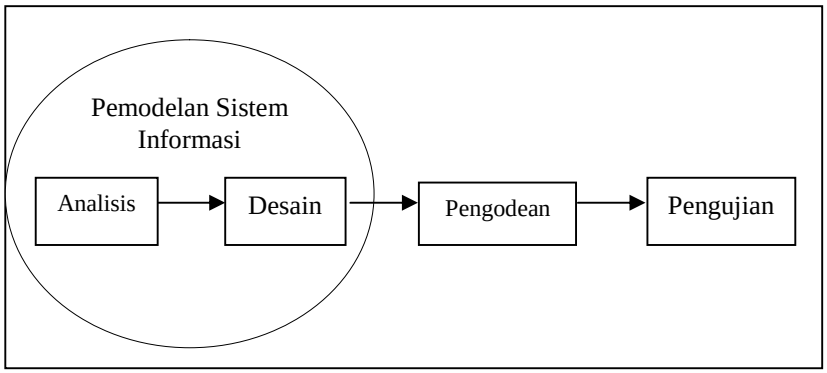
Gambar 2 Blok Diagram Dekripsi Algoritma *Blowfish*.

3. Metode Penelitian

3.1. Teknik Pengembangan Sistem

Pada penelitian ini peneliti menggunakan model pengembangan sistem waterfall, berikut merupakan penjelasan dari tahapan waterfall [8] :

Tahapan **analisis**, merupakan proses pengumpulan kebutuhan dilakukan secara *Intensif* untuk menspesifikasikan kebutuhan perangkat lunak agar dapat dipahami perangkat lunak seperti apa yang dibutuhkan oleh *user*. Tahapan **desain**, merupakan proses multi langkah yang fokus pada desain kebutuhan progaram perangkat lunak termasuk struktur data, arsitektur perangkat lunak, representasi antarmuka, dan prosedur pengodean. Tahapan **pembuatan kode program**, desain harus ditranslasikan ke dalam program perangkat lunak. Hasil dari tahap ini adalah program komputer sesuai dengan desain yang telah dibuat pada tahap desain. Tahapan **pengujian**, merupakan pengujian fokus pada perangkat lunak secara dari segi lojik dan *fungsional* dan memastikan bahwa semua bagian sudah diuji. Hal ini dilakukan untuk meminimalisir kesalahan (*error*) dan memastikan keluaran yang dihasilkan sesuai dengan yang diinginkan. Tahapan **pendukung** (*Support)* atau pemeliharaan (*maintenance*), tidak menutup kemungkinan sebuah perangkat lunak mengalami perubahan ketikas sudah dikirimkan ke *user*. Perubahan bisa terjadi karena adanya kesalahan yang muncul dan tidak terdeteksi saat pengujian atau perangkat lunak harus beadaptasi dengan lingkunngan baru.Tahapannya dapat dilihat pada gambar 3.

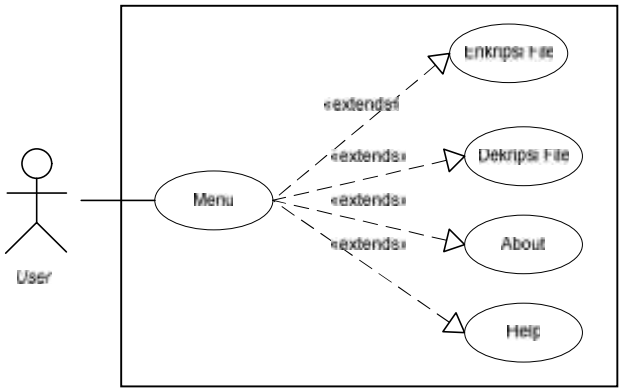


Gambar 3 Tahapan *Waterfall*

3.2. Rancangan Konseptual

1. *Use Case Diagram*

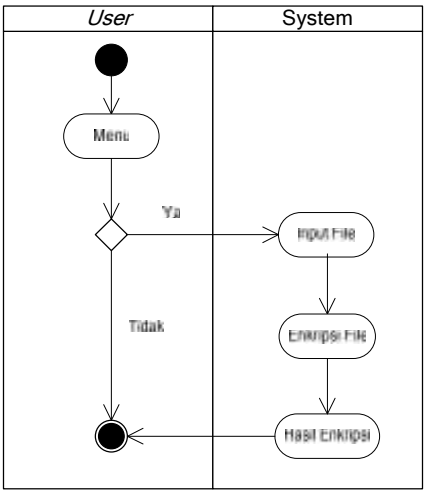
Pada aplikasi ini akan dibahas satu entitas yang berberan sebagai *actor* yaitu *user*. *User* dalam hal ini akan memiliki hak akses ke semua menu dalam penggunaan aplikasi tersebut. *user* memiliki akses untuk melihat menu didalamnya berupa enkripsi *file*, dekripsi *file*,*about*, dan *help*, yang dapat dilihat Pada Gambar 4.



Gambar 4 *Use Case* Diagram

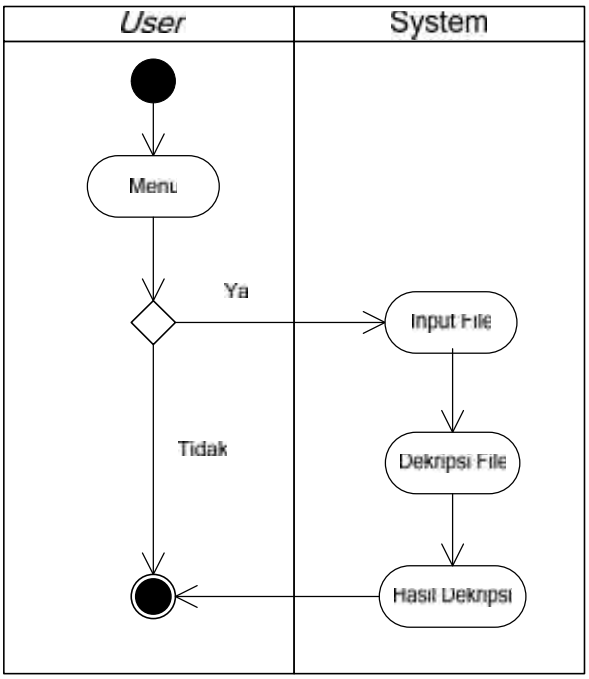
2. *Activity Diagram*

Activity diagram enkripsi ini menggambarkan *user* dapat melakukan proses enkripsi *file* melalui *input file* enkripsi, selanjutnya diproses ke dalam sistem, kemudian *user* akan mendapatkan *file* hasil enkripsi dimana isi *file* tersebut sudah berubah menjadi *ciphertext* sehingga tidak dapat dibaca sebelum *file* tersebut didekripsi ke *plaintext* kembali. Diagram aktivitas dapat dilihat Pada Gambar 5.



Gambar 5 *Activity Diagram* Enkripsi

Activity diagram dekripsi ini menggambarkan *user* dapat melakukan proses dekripsi *file* melalui *input file* dekripsi, selanjutnya *file* diproses ke dalam sistem, kemudian *user* akan mendapatkan *file* hasil dekripsi dimana isi *file* tersebut sudah berubah menjadi *plaintext* sehingga bisa dibaca kembali. Diagram aktivitas dapat dilihat Pada Gambar 6.



Gambar 6 *Activity Diagram* Dekripsi

4. Hasil dan Pembahasan

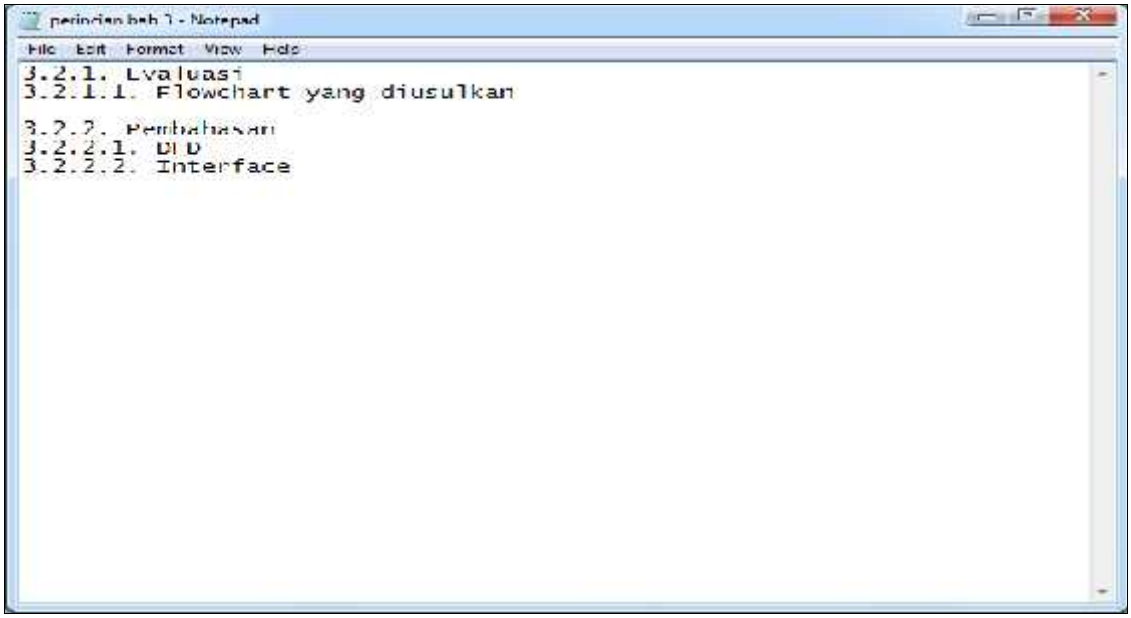
4.1. *Form* proses enkripsi.

Form enkripsi *file* merupakan proses untuk mengenkripsi file, pada *form* tersebut *user* akan melakukan proses pemilihan *file* yang akan dienkripsi dengan cara memilih tombol *browse* kemudian akan masuk ke dalam direktori letak *file* yang akan dienkripsi, selanjutnya user akan melakukan proses input *password* dan *confirm password* dimana *password* dan *confirm password* tersebut digunakan untuk mendekripsikan *file* yang sama setelah dienkripsi, dapat dilihat pada gambar 7.

Gambar 7 Tampilan form enkripsi *file.*

4.2. Tampilan *file* sebelum proses enkripsi

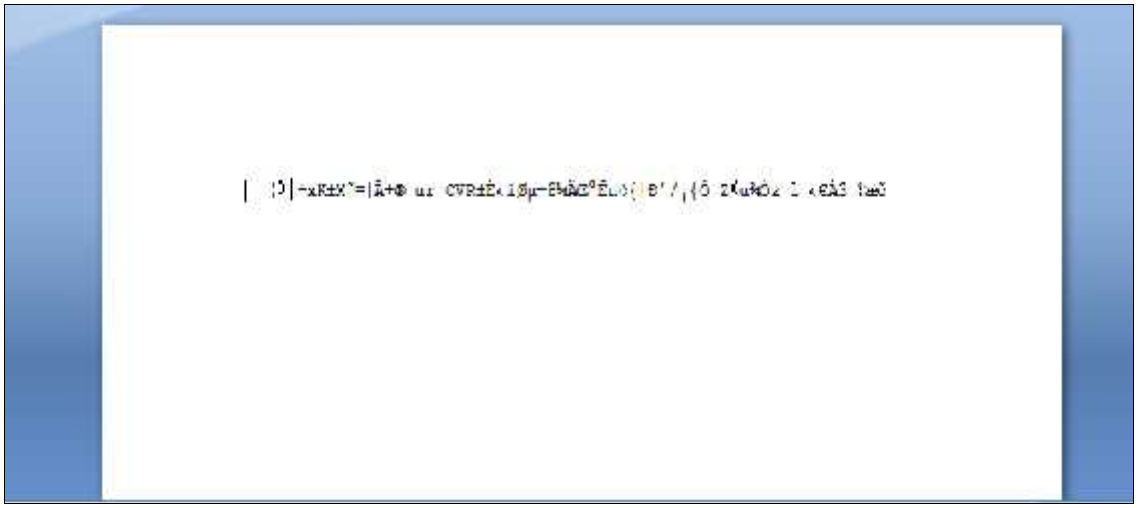
Berikut tampilan *file* sebelum dienkripsi berupa *plaintext*, dapat dilihat pada gambar 8.



Gambar 8 Tampilan isi *file* sebelum dienkripsi.

4.3. Tampilan *file* setelah dienkripsi

Berikut tampilan *file* setelah dienkripsi, isi *file* yang telah berubah menjadi *ciphertext* sehingga isi *file* asli/*Plaintext* tersebut tidak dapat dibaca sebelum didekripsi, lihat pada gambar 9.



Gambar 9 Tampilan isi *file* setelah dienkripsi.

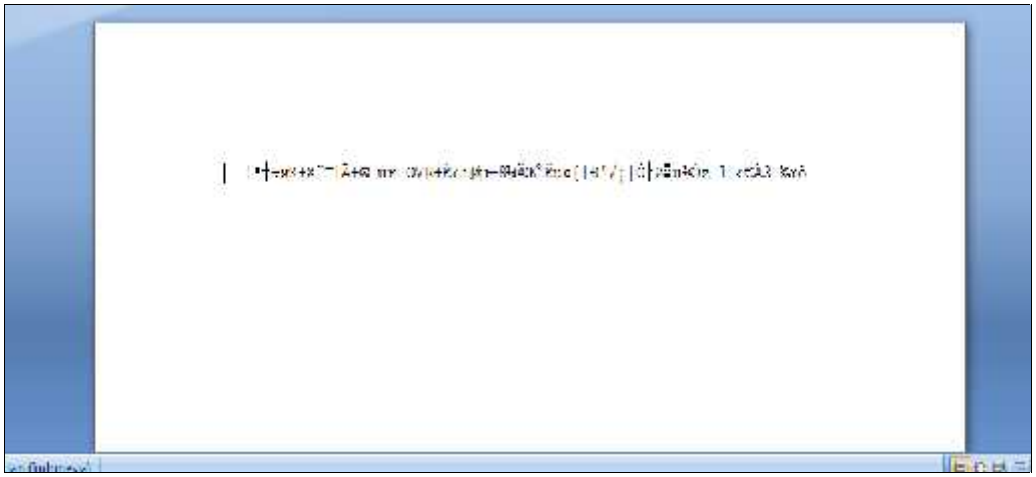
4.4. *Form* dekripsi.

Form dekripsi *file* merupakan proses untuk mendekripsikan *file*, pada *form* tersebut *user* akan melakukan proses pemilihan *file* yang akan didekripsikan dengan cara memilih tombol *browse* kemudian akan masuk ke dalam direktori letak *file* yang akan didekripsi, selanjutnya *user* akan melakukan proses input *password* dan *confirm password* dimana *password* dan *confirm password* tersebut dibuat pada saat mengenkripsi *file*. *Form* dekripsi *file* dapat dilihat pada gambar 10.

Gambar 10 Tampilan *form* dekripsi *File.*

4.5. Tampilan *file* sebelum proses dekripsi

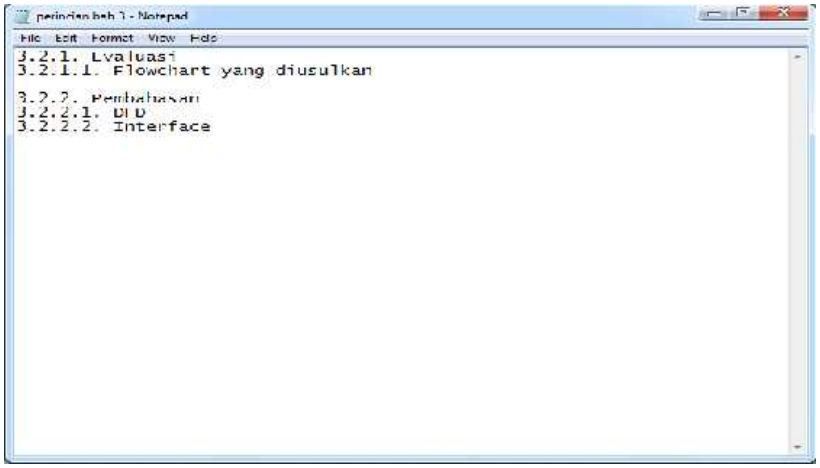
Berikut tampilan *file* sebelum didekripsi berupa *ciphertext*, dapat dilihat pada gambar 11.



Gambar 11 Tampilan *file* sebelum didekripsi.

4.6. Tampilan *file* setelah didekripsi

Berikut tampilan *file* setelah didekripsi, isi *file* yang telah menjadi *plainteks* sehingga isi *file* tersebut bisa dibaca kembali, dapat dilihat pada gambar 12.



Gambar 12 Tampilan *file* setelah didekripsi**.**

5. Kesimpulan

Enkripsi *file* diperlukan dalam meningkatkan pengamanan *file*, terutama untuk menghindari adanya kebocoran data penting pada saat transfer file. Aplikasi yang peneliti buat berhasil dan dapat berfungsi seperti tujuan, yaitu untuk mengamankan data atau informasi yang berupa *file* dengan mengenkripsi *plaintext file* menjadi sebuah *ciphertext file*, sehingga tidak dapat dibaca atau dimengerti oleh orang lain. Aplikasi ini juga telah berhasil mendekripsikan *ciphertext file* menjadi sebuah *plaintext file* kembali dengan *password* yang sama seperti pada proses enkripsi sebelumnya.

Daftar Pustaka

[1] Didi Surian(2006), Algoritma Kriptografi Aes Rijndael. TESLA vol.8, no. 2, 97-101 Jurnal Elektro.
[2] Anandia Zelvina, Syahril Efendi, Dedy Arisandi. (2012), Perancangan Aplikasi Pembelajaran Kriptografi Kunci Publik ElGamal Untuk Mahasiswa. Jurnal Dunia Teknologi Informasi, Vol. 1, No. 1, Universitas Sumatera Utara, Medan.
[3] Novi Dian Natasha, Anang Eko Wicaksono. (2011), Penerapan teknik kriptografi stream-cipher untuk pengaman basis data. Jurnal Basis Data, ICT Research Center UNAS Vol. 6, No. 1, Universitas Nasional, Jakarta Selatan.
[4] Ahmad Timbul Sholeh, Erwin Gunadhi, Asep Deddy Supriatna. (2013), Mengamankan Skrip Pada Bahasa Pemograman Php Dengan Menggunakan Kriptografi Base64, Jurnal STT Garut Vol 10, No. 1, Sekolah Tinggi Teknologi Garut, Garut.
[5] Hartini dan Sri Primaini. (2013), Kriptografi Password Menggunakan Modifikasi Metode Affine Ciphers, Vol 2, No.1, AMIK SIGMA, Palembang.
[6] Munawar. (2012), Perancangan Algoritma Sistem Keamanan Data Menggunakan Metode Kriptografi Asimetris, Jurnal Komputer dan Informatika(KOMPUTA) Vol 1, No. 1, Universitas Komputer Indonesia, Bandung.
[7] Schneier, Bruce. 1996. Applied Cryptography, Second Edition. New York : John Wiley & Son.
[8] Shalahuddin M Rosa A.S. 2011 Rekayasa Perangkat Lunak. Bandung : Modula.