STUDI DAN IMPLEMENTASI ALGORITMA BLOWFISH UNTUK APLIKSI ENKRIPSI DAN DEKRIPSI FILE

Ratih – NIM: 13503016

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

E-mail : if13016@students.if.itb.ac.id

Abstrak

Makalah ini berisi pembahasan mengenai algoritma kunci simetri dengan *cipher* blok, Blowfish. Blowfish merupakan salah satu algoritma yang dibuat oleh Bruce Schneier pada tahun 1993 sebagai salah satu alternatif untuk algoritma enkripsi. Blowfish termasuk kedalam *cipher* blok yang masih dianggap aman karena belum ditemukan attack yang benar-benar dapat mematahkannya.

Pada makalah ini juga akan dibahas mengenai implementasi algoritma Blowfish pada sebuah aplikasi yang akan melakukan enkripsi dan dekripsi pada teks yang diketikkan langsung pada aplikasi tersebut.

Aplikasi yang akan dibuat bernama Ikan Kembung, yang akan memanfaatkan *source code* dari Bruce Schneier sendiri. Namun, untuk kepentingan pengembangan aplikasi akan terdapat beberapa penyesuaian pada algoritma tersebut. Aplikasi tersebut diharapkan dapat menerapkan algoritma Blowfish secara optimal, oleh karena itu akan dibahas mengenai karakteristik algoritma Blowfish yang dapat membuat algoritma ini berjalan secara optimal.

Dengan adanya implementasi itu pula, diharapkan dapat menjadi contoh penggunaan algoritma Blowfish untuk proses enkripsi yang sesungguhnya. Aplikasi Ikan Kembung hanya merupakan aplikasi sederhana yang hanya dapat melakukan enkripsi pada file teks, namun hal tersebut dianggap cukup oleh penulis untuk memberi contoh penggunaan algoritma Blowfish dalam proses enkripsi.

Kata kunci: algoritma kunci simetri, *cipher* Blok,algoritma Blowfish, enkripsi, dekripsi.

1. Pendahuluan

Kriptografi telah menjadi bagian penting dalam dunia teknologi informasi saat ini. Hampir semua penerapan teknologi informasi menggunakan kriptografi sebagai alat untuk menjamin keamanan dan kerahasiaan informasi. Karena itu pulalah, kriptografi menjadi ilmu yang berkembang pesat. Dalam waktu singkat, amat banyak bermunculan algoritma-algoritma baru yang dianggap lebih unggul daripada pendahulunya.

Namun, tetap saja *cipher* yang digunakan tidak lepas dari penemuan lama. Algoritma kunci simetri termasuk algoritma yang masih sering digunakan dalam pembuatan algoritma kriptografi. Pada saat ini, algoritma enkripsi kunci simetri yang banyak digunakan adalah algoritma blok, yang beroperasi pada suatu potongan pesan (blok) yang berukuran sama (biasanya 64 bit) pada suatu saat.

Selain algoritma blok, ada juga algoritma aliran yang beroperasi pada potongan data yang bervariasi. Dibandingkan dengan *cipher* aliran, baik dalam desain dan penerapan, *cipher* blok dianggap lebih rumit.

Semenjak pertama kali ditemukan, telah banyak penemuan-penemuan baru dalam penerapan *cipher* blok. Salah satunya adalah algoritma Blowfish yang dirancang oleh Bruce Schneier pada tahun 1993. Sejak saat itu, telah dilakukan berbagai macam analisis, dan perlahan-lahan mulai mendapat penerimaan sebagai algoritma enkripsi yang kuat. Sampai saat ini dianggap masih belum ada *attack* yang dapat memecahkan Blowfish. Blowfish adalah algoritma yang tidak dipatenkan dan *license-free*, dan tersedia secara gratis untuk berbagai macam kegunaan.

Pada saat Blowfish dirancang, diharapkan mempunyai kriteria perancangan sebagai berikut:

1. Cepat, Blowfish melakukan enkripsi data pada *microprocessors* 32-bit. dengan rate 26 *clock cycles* per *byte*.
2. *Compact*, Blowfish dapat dijalankan pada memori kurang dari 5K.
3. Sederhana, Blowfish hanya menggunakan operasi-operasi Sederhana, Blowfish hanya

menggunakan operasi-operasi sederhana: penambahan, XOR, dan *lookup* tabel pada operan 32-bit.

4. Memiliki tingkat keamanan yang bervariasi, panjang kunci yang digunakan oleh Blowfish dapat bervariasi dan bisa sampai sepanjang 448 bit.

Namun, dalam penerapannya sering kali algortima ini menjadi tidak optimal. Karena strategi implementasi yang tidak tepat. Algoritma Blowfish akan lebih optimal jika digunakan untuk aplikasi yang tidak sering berganti kunci, seperti jaringan komunikasi atau enkripsi file otomatis. Selain itu, karena algoritma ini membutuhkan memori yang besar, maka algoritma ini tidak dapat diterapkan untuk aplikasi yang memiliki memori kecil seperti *smart card*. Panjang kunci yang digunakan, juga mempengaruhi keamanan penerapan algoritma ini.

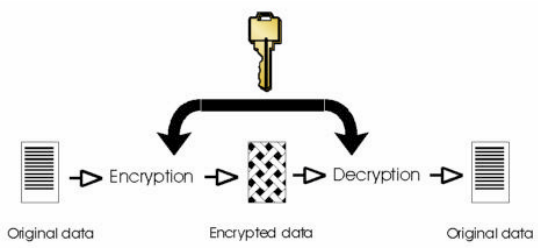
Makalah ini bertujuan untuk dapat mempelajari algoritma Blowfish secara keseluruhan agar dapat memahami cara kerja dan struktur algoritmanya, kemudian menerapkan strategi perancangan Blowfish sehingga algoritma ini dapat berjalan secara optimal. Atau paling tidak dapat melakukan enkripsi dengan baik pada aplikasi yang akan dibuat.

Aplikasi tersebut antara lain dibuat untuk menggambarkan penggunaan algoritma Blowfish pada proses enkripsi teks yang sehingga diharapkan dapat lebih mudah memahami cara kerja dan struktur algoritma Blowfish.

2. Deskripsi Algoritma Kunci Simetri

Algoritma kunci simetri merupakan metode enkripsi yang menggunakan kunci yang sama untuk enkripsi dan dekripsi, seperti ditunjukkan pada Gambar 1.

Tipe enkripsi seperti ini cepat dan cocok untuk data yang tidak perlu dibagi melalui jaringan, misalnya untuk data pada PC, atau enkripsi *hard disk*



Gambar 1. Algoritma Kunci Simetri

Algoritma kriptografi kunci simetri dapat dianggap kuat, karena fungsi yang digunakan untuk membentuk *cipher* dengan menggunakan kunci merupakan fungsi yang *non-reversible.*

Seperti fungsi
$E_k(P) = C$,
Dimana E merupakan fungsi enkripsi. K adalah kunci yang digunakan untuk melakukan proses enkripsi. P adalah plainteks, C adalah cipherteks yang dihasilkan.

Kemudian fungsi dekripsi dilambangkan dengan persamaan fungsi
$D_k(C) = P$,
Dimana D merupakan fungsi yang digunakan untuk melakukan dekripsi pesan dengan komponen seperti diatas.

Pada kriptografi kunci simetri, mungkin saja fungsi E dan D merupakan satu fungsi yang sama, fungsi yang seperti ini akan bergantung pada panjang kunci dalam hal keamanan. Semakin panjang kunci, maka semakin aman, meskipun 64 bit dapat dianggap memiliki tingkat keamanan yang cukup baik, namun penggunaan 128 atau 256 bit tentu saja lebih baik untuk digunakan. Panjang kunci akan mengurangi kemungkinan berhasilnya serangan-serangan seperti *brute force key search, plaintext attack, chose plaintext attack, differential plaintext attack* dan lain sebagainya.

Algoritma kriptografi (*cipher*) simetri dapat dikelompokkan menjadi dua kategori, yaitu:

1. *Cipher* aliran (s*tream cipher*)
   Algoritma kriptografi beroperasi pada plainteks/cipherteks dalam bentuk bit tunggal, yang dalam hal ini rangkaian bit dienkripsikan/didekripsikan bit per bit.
2. *Cipher* blok (*block cipher*)
   Algoritma kriptografi beroperasi pada plainteks/cipherteks dalam bentuk blok bit, yang dalam hal ini rangkaian bit dibagi menjadi blok-blok bit yang

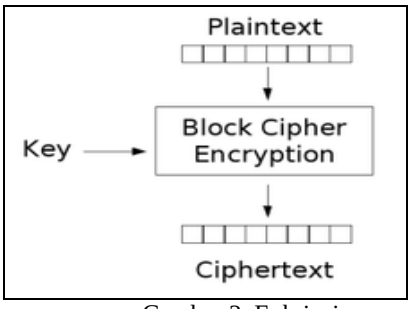
panjangnya sudah ditentukan sebelumnya.

Pada makalah ini penulis akan memfokuskan pembahasan mengenai *cipher* blok, karena algoritma Blowfish merupakan algoritma yang menerapkan *cipher* blok.

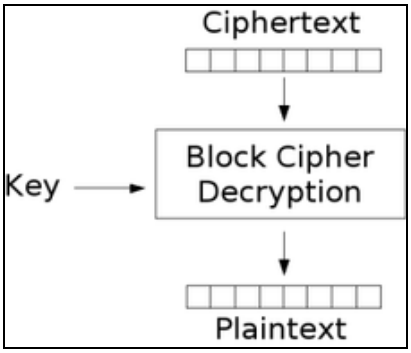
2.1. *Cipher* Blok

Pada *cipher* blok, plainteks dibagi menjadi beberapa blok dengan panjang tetap. Ketika melakukan enkripsi, *cipher* blok mungkin saja menerima input 128-bit plainteks dan mengeluarkan 128-bit keluaran cipherteks. Transformasi selengkapnya dikontrol menggunakan masukan kedua- yaitu kunci. Begitu pula halnya dengan dekripsi, algoritma untuk melakukan dekripsi akan menerima masukan 128-bit cipherteks dan kunci kemudian menghasilkan keluaran 128-bit plainteks aslinya.

Untuk lebih jelasnya, proses enkripsi dan dekripsi pada *cipher* blok dapat dilihat pada Gambar 2 dan Gambar 3.



Gambar 2. Enkripsi



Gambar 3. Dekripsi

Untuk melakukan enkripsi terhadap pesan yang lebih panjang dari ukuran blok, pada cipher blok, digunakan mode-mode tertentu.

2.1.1. Empat mode Operasi *Cipher* Blok

Empat mode operasi yang lazim diterapkan pada sistem blok *cipher* adalah:

1. *Electronic Code Book* (ECB)
   Pada mode ini, setiap blok plainteks P_i dienkripsi secara individual dan independen menjadi blok cipherteks C_i. Secara matematis, enkripsi dengan mode *ECB* dinyatakan sebagai

   $$C_i = E_k(P_i)$$

   dan dekripsi sebagai

   $$P_i = D_k(C_i)$$

   yang dalam hal ini, P_i dan C_i masing-masing blok plainteks dan cipherteks ke-*i*.
   Pada mode ECB, blok plainteks yang sama selalu dienkripsi menjadi blok cipherteks yang sama.
   Ada kemungkinan panjang plainteks tidak habis dibagi dengan panjang ukuran blok yang ditetapkan. Hal ini mengakibatkan blok terakhir berukuran lebih pendek daripada blok-blok lainnya. Satu cara untuk mengatasi hal ini adalah dengan *padding*, yaitu menambahkan blok terakhir dengan pola bit yang teratur agar panjangnya sama dengan ukuran blok yang ditetapkan.

2. *Cipher Block Chaining* (CBC)
   Mode ini menerapkan mekanisme umpan balik (*feedback*) pada sebuah blok, yang dalam hal ini hasil enkripsi blok sebelumnya diumpanbalikkan ke dalam enkripsi blok yang *current*. Caranya, blok plainteks yang *current* di-*XOR*-kan terlebih dahulu dengan blok cipherteks hasil enkripsi sebelumnya, selanjutnya hasil peng-*XOR*-an ini masuk ke dalam fungsi enkripsi. Dengan mode *CBC*, setiap blok cipherteks bergantung tidak hanya pada blok plainteksnya tetapi juga pada seluruh blok plainteks sebelumnya.
   Dekripsi dilakukan dengan memasukkan blok cipherteks yang *current* ke fungsi dekripsi, kemudian meng-*XOR*-kan hasilnya dengan blok cipherteks sebelumnya. Dalam hal ini, blok cipherteks sebelumnya berfungsi sebagai umpan maju (*feedforward*) pada akhir proses dekripsi.

Secara matematis, enkripsi dengan mode *CBC* dinyatakan sebagai

$$C_i = E_k(P_1 \oplus C_{i-1})$$

dan dekripsi sebagai

$$P_i = D_k(C_i) \oplus C_{i-1}$$

Yang dalam hal ini, C_0 = *IV* (*initialization vector*). *IV* dapat diberikan oleh pengguna atau dibangkitkan secara acak oleh program. Jadi, untuk menghasilkan blok cipherteks pertama (C_1), *IV* digunakan untuk menggantikan blok cipherteks sebelumnya, C_0. Sebaliknya pada dekripsi, blok plainteks diperoleh dengan cara meng-*XOR*-kan *IV* dengan hasil dekripsi terhadap blok cipherteks pertama.

Pada mode *CBC*, blok plainteks yang sama menghasilkan blok cipherteks yang berbeda hanya jika blok-blok plainteks sebelumnya berbeda.

3. *Cipher Feedback* (CFB)

   Pada mode CFB, data dienkripsikan dalam unit yang lebih keicl daripada ukuran blok. Sehingga enkripsi dapat dilakukan meskipun data yang diterima belum lengkap.

   Unit yang dienkripsikan dapt berupa bit per bit (seperti cipher aliran), 2 bit, 3 bit, dan seterusnya.

   Secara umum, CFB n-bit mengenkripsi plainteks sebanyak n bit setipa kalinya, yang mana $n \leq m$ (m = ukuran blok).

   Mode CFB membutuhkan sebuah antrian (*queue*) yang berukuran sama dengan ukuran blok masukan.

   Secara formal, mode *CFB n*-bit dapat dinyatakan sebagai:

   Proses Enkripsi:

   $$C_i = P_i \oplus MSB_m(E_k(X_i))$$
   $$X_{i+1} = LSB_{m-n}(X_i) \,||\, C_i$$

   Proses Dekripsi:

   $P_i = C_i \oplus MSB_m(D_k(X_i))$
   $X_{i+1} = LSB_{m-n}(X_i) \,||\, C_i$

   yang dalam hal ini:
   X_i = isi antrian dengan X_i adalah *IV*
   E = fungsi enkripsi dengan algoritma *cipher* blok
   D = fungsi dekripsi dengan algoritma *cipher* blok
   K = kunci
   m = panjang blok enkripsi/dekripsi
   n = panjang unit enkripsi/dekripsi
   || = operator penyambungan (*concatenation*)
   MSB = Most Significant Byte
   LSB = Least Significant Byte

   Ada satu kelemahan yang cukup fatal pada mode CFB, yaitu adanya perambatan kesalahan. Kesalahan 1-bit pada blok plainteks akan merambat pada blok-blok cipherteks yang berkoresponden dan blok-blok cipherteks selanjutnya pada proses enkripsi. Hal yang kebalikan juga terjadi pada proses dekripsi.

4. *Output Feedback* (OFB)

   Mode OFB mirip dengan mode CFB, kecuali n-bit dari hasil enkripsi terhadap antrian disalin menjadi elemen posisi paling kanan di antrian.

   Dekripsi dilakukan sebagai kebalikan dari proses enkripsi.

   Juga merupakan mode aliran, tapi dimaksudkan untuk digunakan ketika umpan balik dari kesalahan menjadi sebuah masalah, atau ketika proses enkripsi ingin dilakukan sebelum keseluruhan pesan diterima.

3. Dekripsi Algoritma Blowfish

3.1. Struktur Algoritma Blowfish

Blowfish adalah algoritma kunci simetri, yang berarti menggunakan kunci yang sama untuk melakukan enkripsi dan dekripsi file. Blowfish juga merupakan *cipher* blok, yang berarti selama proses enkripsi dan dekripsi, Blowfish akan membagi pesan menjadi blok-blok dengan ukuran yang sama panjang. Panjang blok untuk algoritma Blowfish adalah 64-bit. Pesan yang bukan merupakan kelipatan delapan *byte* akan ditambahkan bit-bit tambahan (*padding*) sehingga ukuran untuk tiap blok sama.

Algoritma dalam Blowfish terbagi menjadi dua bagian, yaitu *key expansion* dan *data encryption*.

Proses *key expansion* akan melakukan konversi sebuah kunci mulai dari 56 byte sampai beberapa *array sub* kunci dengan total mencapai 4168 byte.

Proses *data encryption* terjadi pada jaringan feistel, mengandung fungsi pengulangan sederhana sebanyak enam belas kali. Setiap iterasi, terdiri dari sebuah permutasi yang tidak bergantung pada kunci dan sebuah substitusi yang tidak bergantung pada data dan kunci. Semua operasi merupakan penambahan dan

XOR pada word 32-bit. Operasi penambahan yang dilakukan hanya merupakan empat indeks array data lookup pada setiap iterasi.

Pada algoritma Blowfish, digunakan banyak *subkey*. Kunci-kunci ini harus dihitung atau dibangkitkan terlebih dahulu sebelum dilakukan enkripsi atau dekripsi data.

Kunci- kunci yang digunakan antara lain terdiri dari, 18 buah 32-bit *subkey* yang tergabung dalam P-array (P1, P2, ..., P18). Selain itu, ada pula empat 32-bit S-box yang masing-masingnya memiliki 256 entri : S1,0, S1,1,..., S1,255; S2,0, S2,1,..,, S2,255; S3,0, S3,1,..., S3,255; S4,0, S4,1,..,, S4,255.

Pada jaringan feistel, Blowfish memiliki 16 iterasi, masukannya adalah 64-bit elemen data, X. Untuk melakukan proses enkripsi:

1. Bagi X menjadi dua bagian yang masing-masing terdiri dari 32-bit: X_L, X_R.
2. For i = 1 to 16:

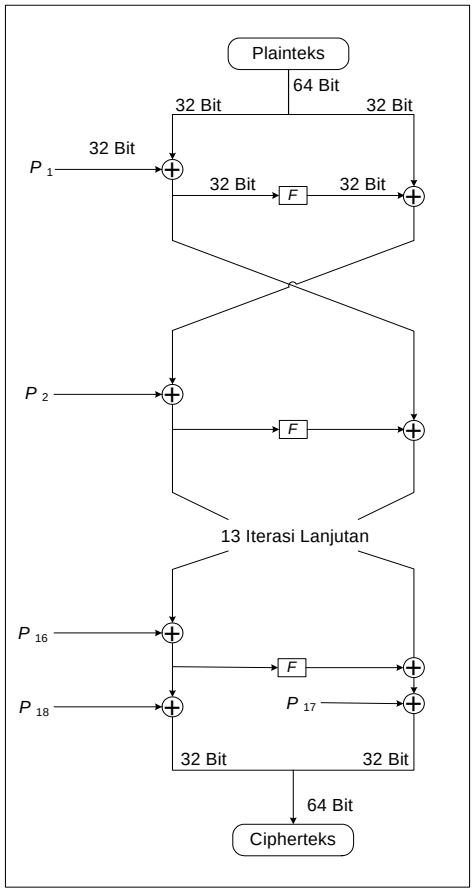
   $X_L = X_L$ XOR Pi
   $X_R = F(X_L)$ XOR X_R
   Tukar X_L dan X_R

3. Setelah iterasi ke-enam belas, tukar X_L dan X_R lagi untuk melakukan *undo* pertukaran terakhir.
4. Lalu lakukan

   $X_R = X_R$ XOR P17
   $X_L = X_L$ XOR P1

5. Terakhir, gabungkan kembali X_L dan X_R untuk mendapatkan cipherteks.

Untuk lebih jelasnya, gambaran tahapan pada jaringan feistel yang digunakan Blowfish adalah seperti pada Gambar 4.

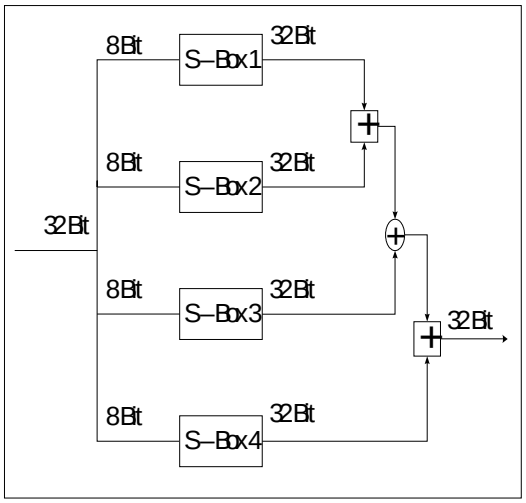


Gambar 4. Jaringan Feistel

Pada langkah kedua, telah dituliskan mengenai penggunaan fungsi F. Fungsi F adalah:

Bagi X_L menjadi empat bagian 8-bit: a,b,c dan d. $F(X_L) = ((S1,a + S2,b \bmod 2^{32}) \text{ XOR } S3,c) + S4,d \bmod 232$.

Agar dapat lebih memahami fungsi F, tahapannya dapat dilihat pada Gambar 5.



Gambar 5. Fungsi F

Algoritma Blowfish memiliki keunikan dalam hal proses dekripsi, yaitu proses dekripsi dilakukan dengan urutan yang sama persis dengan proses enkripsi, hanya saja pada proses dekripsi P1, P2, …, P18 digunakan dalam urutan yang terbalik.

Sebelumnya, telah dijelaskan mengenai penggunaan subkey didalam Blowfish. Sekarang, akan dijelaskan mengenai cara menghitung atau membangkitkan subkey:

1. Inisialisasi P-array yang pertama dan juga empat S-box, berurutan, dengan string yang telah pasti. String tersebut terdiri dari digit-digit heksadesimal dari pi, tidak termasuk angka tiga diawal.

   Contoh :

   P1 = 0x243f6a88
   P2 = 0x85a308d3
   P3 = 0x13198a2e
   P4 = 0x03707344

2. XOR pi dengan 32-bit pertama dari kunci, XOR p2 dengan 32-bit kedua dari kunci , dan seterusnya untuk seluruh bit dari kunci (sampai p18). Ulangi siklus seluruh bit kunci secara berurutan sampai seluruh P-array telah di-XOR-kan dengan bit-bit kunci.
3. Enkripsikan string yang seluruhnya nol (*all-zero*) dengan algoritma Blowfish, menggunakan *subkey* yang telah dideskripsikan di langkah (1) dan (2).
4. Gantikan p1 dan p2 dengan hasil dari langkah (3).
5. Enkripsikan hasil dari tahap (3) menggunakan algoritma Blowfish dengan *subkey* yang telah dimodifikasi.
6. Gantikan p3 dan p4 dengan hasil dari langkah (5)
7. Lanjutkan tahapan-tahapan diatas, gantikan seluruh elemen dari P- array dan kemudian keempat S-box secara berurutan, dengan hsil keluaran algoritma Blowfish yang terus menerus berubah.

Secara keseluruhan, 521 iterasi dibutuhkan untuk membangkitkan seluruh *subkey*. Aplikasi dapat menyimpan seluruh *subkey*, agar tidak perlu mengeksekusi proses ini secara berulang kali setiap iterasi.

3.2. Informasi Lain yang Terkait

Karena algoritma ini merupakan algoritma yang sudah termasuk lama, tentu saja ada beberapa usaha kriptanalisis yang dilakukan terhadap algoritma ini, antara lain adalah:

1. John Kelsey membuat sebuah *attack* yang dapat mematahkan 3 iterasi Blowfish, namun tidak dapat mengembangkannya lebih lanjut. *Attack* ini melakukan eksploitasi pada fungsi f dan fakta bahwa penambahan mod232 dan XOR tidak *commute*.
2. Vikramjit Singh Chhabra mencari cara untuk menerapkan *brute-force key search machine*
3. Serge Vaudenay melakukan penelitian pada varian Blowfish yang telah disederhanakan, dengan S-aBox yang diketahui, dan tidak *key-dependent*. Untuk varian ini, *attack* yang berbeda dapat menemukan P-array dengan 28r+1 plainteks yang telah dipilih (r merupakan jumlah iterasi). *Attack* ini tidak mungkin dilakukan pada Blowfish dengan 8-iterasi dan lebih, karena lebih banyak plainteks yang dibutuhkan daripada yang dapat dibangkitkan dengan 64-bit *cipher* blok.
4. Tesis Ph.D milik Vincent Rijmen mencantumkan second-order differential attack pada 4 iterasi Blowfish. Namun, attack tersebut tidak dapat dilanjutkan lagi untuk iterasi selanjutnya.

Untuk beberapa *weak key* (*weak key* merupakan kunci yang tidak baik yang jika digunakan dalam proses enkripsi akan mengakibatkan hasil enkripsi dengan tingkat keamanan yang buruk), yang membangkitkan S-Box yang lemah (kemungkinan untuk mendapatkannya adalah 1 banding 2^{14}) *attack* yang sama hanya membutuhkan 24r+1 plainteks yang telah diketahui untuk menemukan P-array (dengan asumsi S-box telah diketahui).

Dengan S-Box yang tidak diketahui, *attack* ini dapat mendeteksi apakah *weak key* sedang digunakan, tetapi tidak dapat mengetahui apa *weak key* tersebut (S-Box, P-array, dan kunci semuanya tidak diketahui). *Attack* ini hanya dapat berfungsi pada varian dengan jumlah iterasi yang dikurangi. Dengan kata lain, *attack* ini sama sekali tidak efektif untuk digunakan terhadap Blowfish dengan 16 iterasi.

Meskipun demikian, penemuan akan adanya *weak key* didalam Blowfish signifikan. *Weak key* pada Blowfish akan memberikan kedua masukan untuk S-Box sama persis. Sama sekali tidak ada cara untuk menangani pengecekan terhadap penggunaan *weak key* sebelum tahapan *key expansion*.

4. Penerapan Algoritma Blowfish

Algoritma Blowfish merupakan algoritma yang cukup sederhana. Namun, untuk dapat lebih memahami konsep algoritma Blowfish, penulis akan melakukan penerapan algoritma Blowfish dalam bentuk aplikasi yang akan melakukan enkripsi dan dekripsi file secara otomatis

4.1. Strategi Penerapan Perangkat Lunak

Perangkat lunak ini diharapkan dapat melakukan proses enkripsi secara optimal, maka penerapan yang dilakukan harus mempertimbangkan beberapa aspek yang dianggap mempengaruhi kecepatan dan keamanannya.

4.1.1. Strategi Perancangan Perangkat Lunak

Hal-hal yang harus dipertimbangkan dalam perancangan perangkat lunak yang menerapkan algoritma Blowfish antara lain adalah prinsip perancangan algoritma Blowfish (sifat dasar algoritma Blowfish), yaitu:

1. lebih optimal jika digunakan untuk aplikasi yang tidak sering berganti kunci.
2. lebih cepat jika diterapkan pada 32-bit mikroprosessor dengan *data cache* yang besar. Namun, karena hal ini merupakan batasan teknis sehingga tidak akan dipakai dalam strategi perancangan.
3. Lebih aman jika diterapkan tanpa adanya pengurangan jumlah iterasi (dengan asumsi *user* tidak menggunakan *weak key*).

Setelah mencari beberapa referensi, ternyata Blowfish tidak memiliki karakteristik lainnya yang harus dipenuhi. Ternyata selama aplikasi yang digunakan tidak sering berganti kunci, Blowfish merupakan algoritma yang cepat, dan selama tidak dilakukan pengurangan jumlah iterasi (dengan asumsi tidak ada penggunaan *weak key* oleh *user*) Blowfish merupakan algoritma enkripsi yang aman.

Berikut perbandingan kecepatan algoritma Blowfish dengan beberapa algoritma *cipher* blok lainnya:

Algoritma	Clock cycles tiap iterasi	Jumlah iterasi	Jumlah clock cycles per byte terenkripsi
Blowfish	9	16	18
Khufu/ Khafre	5	32	20
RC5	12	16	23
DES	18	16	45
IDEA	50	8	50
Triple-DES	18	48	108

Tabel 1. Perbandingan kecepatan

Tingkat keamanan Blowfish terbukti dengan belum adanya *attack* yang mampu mematahkan algoritma enkripsi ini.

4.2. Deskripsi Singkat Perangkat Lunak

Berdasarkan pertimbangan diatas, maka penulis memutuskan perangkat lunak yang akan dikembangkan adalah sebuah enkripsi file teks otomatis, agar kunci yang digunakan untuk melakukan suatu enkripsi pada suatu saat tidak berubah-ubah. Pada aplikasi ini, untuk melakukan proses enkripsi *user* diharuskan memasukkan kata kunci yang diinginkan, yang kemudian akan digunakan untuk melakukan proses enkripsi. Untuk enkripsi satu file atau satu teks, hanya

dibutuhkan satu kunci. Lebih lengkapnya, dapat dilihat pada deskripsi perangkat lunak berikut ini:

Perangkat lunak yang akan dibuat adalah perangkat lunak untuk melakukan enkripsi file teks (hanya dengan ekstensi txt, bukan doc) dengan pemanfaatan algoritma Blowfish.

Perangkat lunak ini diharapkan dapat menerapkan algoritma Blowfish secara optimal sehingga baik dalam kecepatan maupun keamanan dapat dianggap baik.

Perangkat lunak ini akan dapat membuka file teks yang ingin dienkripsi dan menampilkan data dalam file tersebut dalam bentuk teks biasa, namun perlu diperhatikan sekali lagi hanya file dengan ekstensi txt yang dapat dibuka. Selain itu, aplikasi ini juga dapat melakukan enkripsi pada kata-kata yang ditulis secara langsung pada 'textbox' yang ada pada aplikasi.

Aplikasi ini akan melakukan proses enkripsi dan dekripsi secara otomatis. *User* hanya perlu memasukkan kata kunci *(password*) yang diinginkan, kemudian tinggal menekan tombol 'encrypt' dan 'decrypt' yang terdapat pada aplikasi ini. Sedangkan untuk proses dekripsi, *user* akan diminta memasukkan kembali kata kunci yang diketikkannya ketika file tersebut dienkripsi.

Setelah proses enkripsi dilakukan *user* akan diminta untuk menyimpan hasil enkripsi tersebut agar dapat didekripsi jika diperlukan.

Aplikasi ini penulis beri nama 'Ikan Kembung' yang penulis ambil dari terjemahan langsung nama algoritma Blowfish ke dalam bahasa Indonesia.

Aplikasi ini akan melakukan proses enkripsi dengan mode operasi ECB (*Electronic Code Book*)

4.2.1. Proses Penerapan Perangkat Lunak

Proses perancangan perangkat lunak ikan kembung tidak mengikuti suatu metode tertentu, melainkan langkah-langkah yang penulis terapkan sendiri.

Tahapan penerapan perangkat lunak Ikan Kembung:

1. Penentuan spesifikasi perangkat lunak, yaitu:
   a. Dapat menerima tulisan *user* secara langsung pada 'textbox' dan langsung melakukan enkripsi
   b. Mode operasi enkripsi menggunakan ECB (*electronic code book*)
   c. Semua hasil enkripsi akan ditampilkan terlebih dahulu pada 'textbox' yang ada pada aplikasi
   d. Setelah proses enkripsi dilakukan, aplikasi akan secara otomatis menghapus plainteks dan *key* yang dimasukkan *user*
2. Perancangan antarmuka yang dapat memudahkan *user* dan memenuhi seluruh fungsi yang dibutuhkan oleh aplikasi ini (dapat dilihat pada bagian antarmuka aplikasi).
3. Proses konstruksi perangkat lunak (*coding)*. Meskipun dikatakan *coding*, namun sebenarnya penulis hanya menerjemahkan kode yang diberikan oleh Bruce Schneier dalam bukunya, *Applied Cryptography*. Pada bagian belakang buku teresbut, terdapat *source code* Blowfish didalam bahasa C.

Pada kode yang dibuat oleh Bruce Schneier, merupakan kode standard yang tidak menggunakan tampilan aplikasi, melainkan hanya menggunakan main program yang akan dijalankan ketika keseluruhan program tersebut dieksekusi.

Untuk menerapkan algoritma ini kedalam bentuk aplikasi yang memiliki antarmuka, dan bahasa basic diperlukan penanganan lebih lanjut.

Contoh kode Ikan kembung, pada bagian algoritma Blowfish saja:

```
Private Function blf_F(x As Long) As Long
   Dim a As Byte, b As Byte, C As Byte, d As Byte
   Dim y As Long

   Call uwSplit(x, a, b, C, d)
   y = uw_WordAdd(blf_S(0, a), blf_S(1, b))
   y = y Xor blf_S(2, C)
   y = uw_WordAdd(y, blf_S(3, d))
   blf_F = y
End Function
```

Kode diatas merupakan fungsi F, yang digunakan pada jaringan feistel.

Kode yang digunakan oleh Bruce Schneier:

```
unsigned long F(blf_ctx *bc,
unsigned long x)
{
   unsigned long a;
   unsigned long b;
   unsigned long c;
   unsigned long d;
   unsigned long y;

   d = x & 0x00FF;
   x >>= 8;
   c = x & 0x00FF;
   x >>= 8;
   b = x & 0x00FF;
   x >>= 8;
   a = x & 0x00FF;
   y = bc->S[0][a] + bc-
>S[1][b];
   y = y ^ bc->S[2][c];
   y = y + bc->S[3][d];

   return y;
}
```

Dapat dilihat bahwa kodenya menjadi berbeda.

Berikut adalah kode untuk proses enciphering dan dechipering

```
Public Function blf_EncipherBlock(xL As
Long, xR As Long)   'fungsi enchiper
   Dim i As Integer
   Dim temp As Long

   For i = 0 To ncIterasi - 1      'jaringan feistel
      xL = xL Xor blf_P(i)
      xR = blf_F(xL) Xor xR
      temp = xL
      xL = xR
      xR = temp
   Next

   temp = xL
   xL = xR
   xR = temp

   xR = xR Xor blf_P(ncIterasi)
   xL = xL Xor blf_P(ncIterasi + 1)

End Function

Public Function blf_DecipherBlock(xL As
Long, xR As Long)   'fungsi dechiper

Dim i As Integer
   Dim temp As Long

   For i = ncIterasi + 1 To 2 Step -1
      xL = xL Xor blf_P(i)
      xR = blf_F(xL) Xor xR
      temp = xL
      xL = xR
      xR = temp
   Next

   temp = xL
   xL = xR
   xR = temp

   xR = xR Xor blf_P(1)
   xL = xL Xor blf_P(0)

End Function
```

Pada kode diatas, ncIterasi merupakan konstanta jumlah iterasi yang digunakan pada aplikasi Ikan Kembung, yaitu sebanyak 16 kali mengikuti algoritma Blowfish yang dikembangkan oleh Bruce schneier.

Berikut kode yang digunakan oleh Bruce Schneier.

```
void Blowfish_encipher(blf_ctx
*bc, unsigned long *xl,
unsigned long *xr)
{
   unsigned long  Xl;
   unsigned long  Xr;
   unsigned long  temp;
   short          i;

   Xl = *xl;
   Xr = *xr;

   for (i = 0; i < N; ++i)
   {
      Xl = Xl ^ bc->P[i];
      Xr = F(bc, Xl) ^ Xr;

      temp = Xl;
      Xl = Xr;
      Xr = temp;
   }

   temp = Xl;
   Xl = Xr;
   Xr = temp;

   Xr = Xr ^ bc->P[N];
   Xl = Xl ^ bc->P[N + 1];

   *xl = Xl;
   *xr = Xr;
}

void Blowfish_decipher(blf_ctx
*bc, unsigned long *xl,
unsigned long *xr)
{
   unsigned long  Xl;
```

```
   unsigned long  Xr;
   unsigned long  temp;
   short          i;

   Xl = *xl;
   Xr = *xr;

   for (i = N + 1; i > 1; --i)
   {
      Xl = Xl ^ bc->P[i];
      Xr = F(bc, Xl) ^ Xr;

      /* Exchange Xl and Xr */
      temp = Xl;
      Xl = Xr;
      Xr = temp;
   }

   /* Exchange Xl and Xr */
   temp = Xl;
   Xl = Xr;
   Xr = temp;

   Xr = Xr ^ bc->P[1];
   Xl = Xl ^ bc->P[0];

   *xl = Xl;
   *xr = Xr;
}
```

Dapat dilihat bahwa kedua kode memiliki kemiripan dalam hal algoritma namun berbeda cara penerapannya.

Lebih lengkapnya kode aplikasi Ikan Kembung dapat dilihat pada *source code* yang saya sertakan pada CD makalah ini.

5. Penjelasan singkat aplikasi

Aplikasi Ikan Kembung merupakan aplikasi berbasis desktop yang dikembangkan pada kakas pengembangan Visual Basic 6.0 dengan bahasa pengembangan Basic. Aplikasi ini berbasis GUI, dengan kata lain sudah memiliki *user interface* dalam bentuk grafis. Hal ini bertujuan untuk memudahkan *user*. Tampilan awal aplikasi ini dapat dilihat pada halaman Antarmuka Aplikasi pada Gambar 6.

Untuk melakukan enkripsi dengan aplikasi Ikan Kembung, ada beberapa tahapan yang harus dilakukan.

Tahapan proses enkripsi pada Ikan Kembung, yaitu:

1. Mengetikkan kata yang ingin dienkripsi pada ’textbox’ plainteks yang terdapat pada aplikasi.
2. Mengetikkan key pada ’textbox’ key aplikasi. Jika *user* tidak memasukkan key, maka aplikasi akan mengeluarkan peringatan dalam bentuk ’message box’.
3. Menekan tombol ‘encrypt’

Contoh langkah proses enkripsi dapat dilihat pada halaman Antarmuka Aplikasi pada Gambar 7.

Setelah langkah-langkah tersebut dilakukan, aplikasi akan memulai proses enkripsi. Kemudian akan menampilkan cipherteks di dalam ’textbox’ untuk cipherteks. Contoh hasil enkripsi dapat dilihat pada halaman Antarmuka Aplikasi pada Gambar 8.

Pada Gambar 8 dapat dilihat bahwa setelah proses enkripsi dilakukan, plainteks dan kunci dihilangkan.

Setelah melakukan enkripsi, *user* dapat langsung melakukan proses dekripsi dengan cara:

1. Mengetikkan key pada ’textbox’ key yang terdapat pada aplikasi.
2. menekan tombol ’decrypt’

Contoh langkah proses dekripsi dapat dilihat pada halaman Antarmuka Aplikasi pada Gambar 9.

Jika *user* tidak memasukkan key pada ’textbox’ key, maka apliksi akan mengeluarkan pesan kesalahan seperti dapat dilihat pada Gambar 10.

Setelah itu, aplikasi akan melakukan proses dekripsi pada cipherteks yang terdapat pada ’textbox’ cipherteks. Contoh hasil dekripsi dapat dilihat pada halaman Antarmuka Aplikasi pada Gambar 11.

Keterangan mengenai aplikasi dapat dilihat pada form ’About’ dan ’Help’

6. Pengujian Perangkat Lunak

Perangkat lunak Ikan Kembung akan diuji untuk mengetahui kecepatannya jika dibandingkan dengan algoritma lain. Aplikasi pembandingnya menggunakan algoritma buatan mahasiswa teknik informatika yang digunakan sebagai contoh pada tugas kriptografi.

Aplikasi tersebut bernama Diencrypt, yang dibuat oleh :

1. 13501006 Rizki Yulianto
2. 13501027 Widhiyo Sudiyono

3. 13501072 Anugrah Redja kusuma

Sebenarnya penulis kurang mengetahui mengenai algoritma yang diterapkan pada aplikasi Diencrypt, namun penulis yakin algoritma yang digunakan berbeda dengan algoritma Blowfish. Selain itu alasan memilih aplikasi Diencrypt adalah, karena sudah ada data perbandingan dengan algoritma cipher blok lainnya (lihat Tabel 1) maka menurut penulis lebih menarik jika membandingkan dengan algoritma yang belum ada datanya.

Tetapi perlu diperhatikan perbandingan yang penulis lakukan berdasarkan pengamatan penulis semata. Tidak ada dasar ilmiah maupun alat untuk mengukur kecepatan kedua aplikasi secara tepat.

6.1. Perancangan Kasus Uji Pengujian Perangkat Lunak

Berdasarkan teknik pengujian yang diterapkan penulis, maka dirancang kasus-kasus uji sebagai berikut:

1. Kasus Uji 1
   Kasus Uji 1 bertujuan untuk mengetahui seberapa cepat aplikasi Ikan Kembung dan Diencrypt melakukan proses enkripsi terhadap teks yang telah disediakan

Teks yang akan digunakan untuk menguji kedua aplikasi :

Vigènere Cipher

•Termasuk ke dalam *cipher* abjad-majemuk (*polyalpabetic substitution cipher*).

•Dipublikasikan oleh diplomat (sekaligus seorang kriptologis) Perancis, Blaise de Vigènere pada abad 16 (tahun 1586).

•Tetapi sebenarnya Giovan Batista Belaso telah menggambarkannya pertama kali pada tahun 1553 seperti ditulis di dalam bukunya *La Cifra del Sig. Giovan Batista Belaso*

•Algoritma tersebut baru dikenal luas 200 tahun kemudian yang oleh penemunya *cipher* tersebut kemudian dinamakan *Vigènere Cipher*

•*Cipher* ini berhasil dipecahkan oleh Babbage dan Kasiski pada pertengahan Abad 19.

•*Vigènere Cipher* digunakan oleh Tentara Konfiderasi (*Confederate Army*) pada Perang Sipil Amerika (*American Civil war*).

•Perang Sipil terjadi setelah *Vigènere Cipher* berhasil dipecahkan.

Aplikasi Ikan Kembung memiliki banyak perbedaan dengan aplikasi Diencrypt, antara lain:

1. Aplikasi Diencypt hanya dapat melakukan enkripsi pada file, sedangkan aplikasi Blowfish melakukan enkripsi pada teks yang ditulis langsung pada aplikasi.
2. Aplikasi Diencrypt akan meminta *user* menyimpan file hasil dekripsi terlebih dahulu sebelum menampilkan hasil dekripsi
3. Proses dekripsi harus dilakukan dari file.

6.2. Evaluasi Hasil Pengujian Perangkat Lunak

Berdasarkan pengujian yang telah dilakukan, penulis memutuskan bahwa kedua aplikasi memiliki kecepatan yang hampir sama.

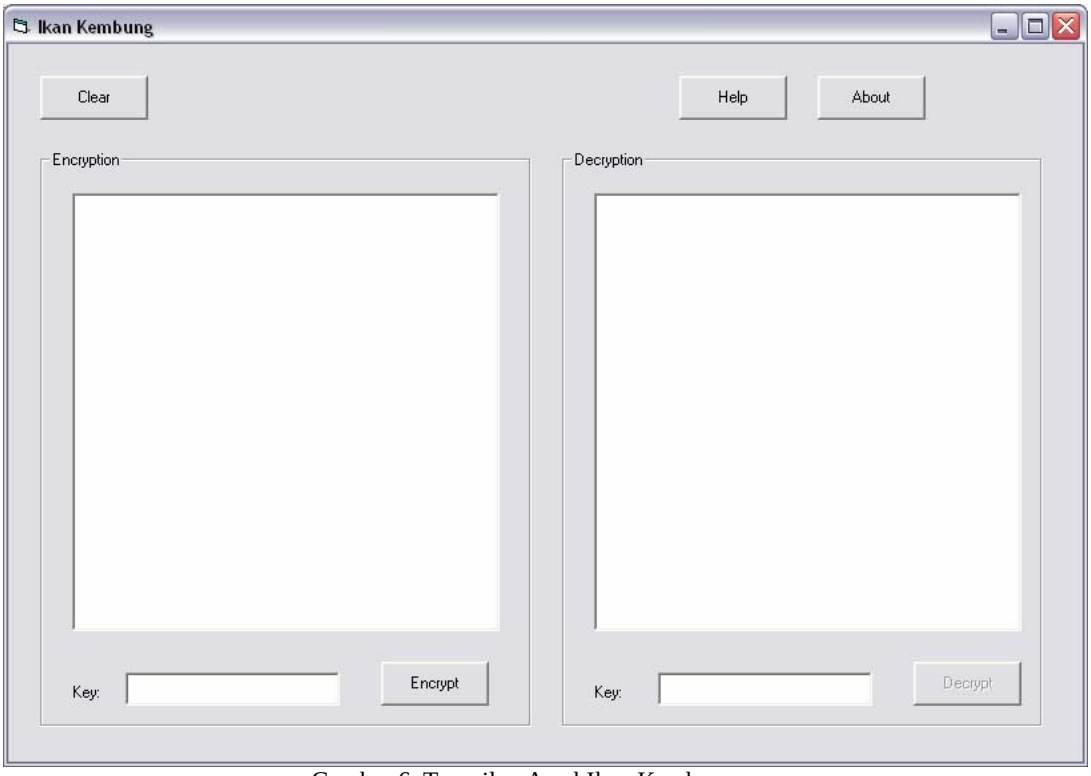
Namun hasil tersebut tetap kurang dapat dipercaya, karena hanya merupakan pengamatan penulis, ada juga pengaruh dari berbedanya kakas pengembangan yang digunakan.

7. Kesimpulan

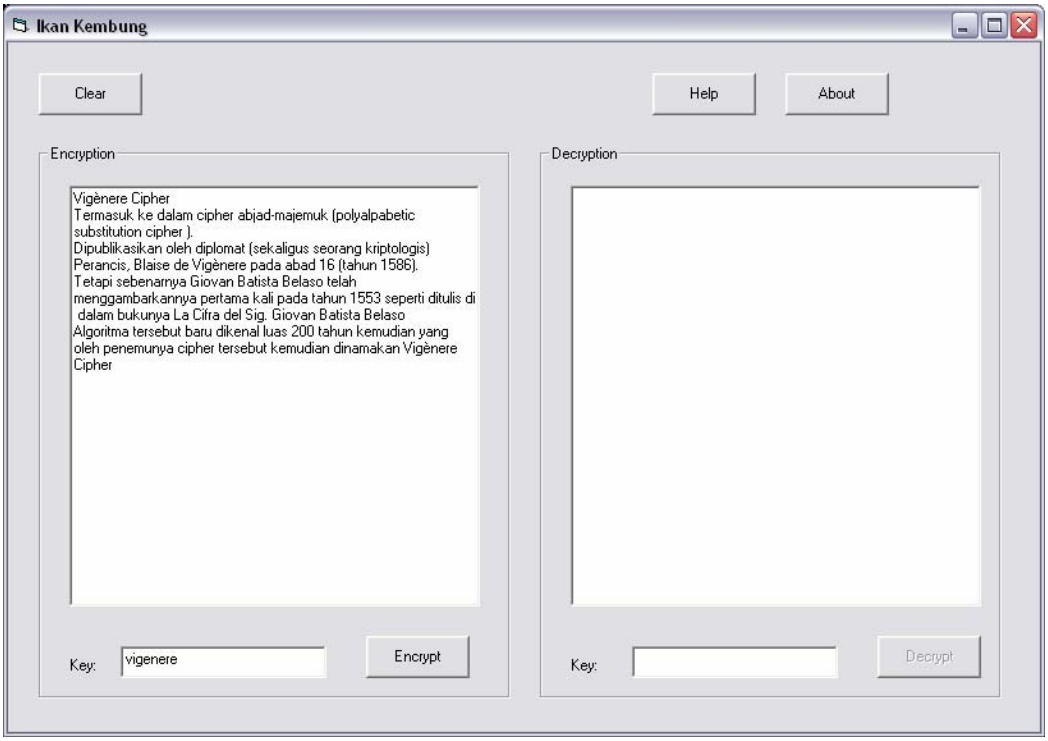
Kesimpulan yang dapat diambil dari studi dan implementasi algoritma Blowfish untuk aplikasi enkripsi dan dekripsi file ini adalah:

1. *Blowfish* merupakan salah satu solusi yang baik untuk mengatasi masalah keamanan dan kerahasiaan data yang pada umumnya diterapkan dalam saluran komunikasi dan file.
2. Algoritma Blowfish merupakan algoritma yang sederhana dan menggunakan jaringan feistel
3. Algoritma Blowfish dapat diimplementasikan dengan bahasa apapun (selain C) namun harus disesuaikan cara penulisan maupun tipe-tipe datanya.
4. Implementasi algoritma Blowfish yang optimal dapat dilakukan dengan aplikasi yang tidak sering berubah-ubah kunci.
5. Tingkat keamanan algoritma blowfish ditentukan oleh jumlah iterasi dan panjang kunci yang digunakan
6. Blowfish memiliki *Weak Key* meskipun amat jarang kemungkinan terjadinya.

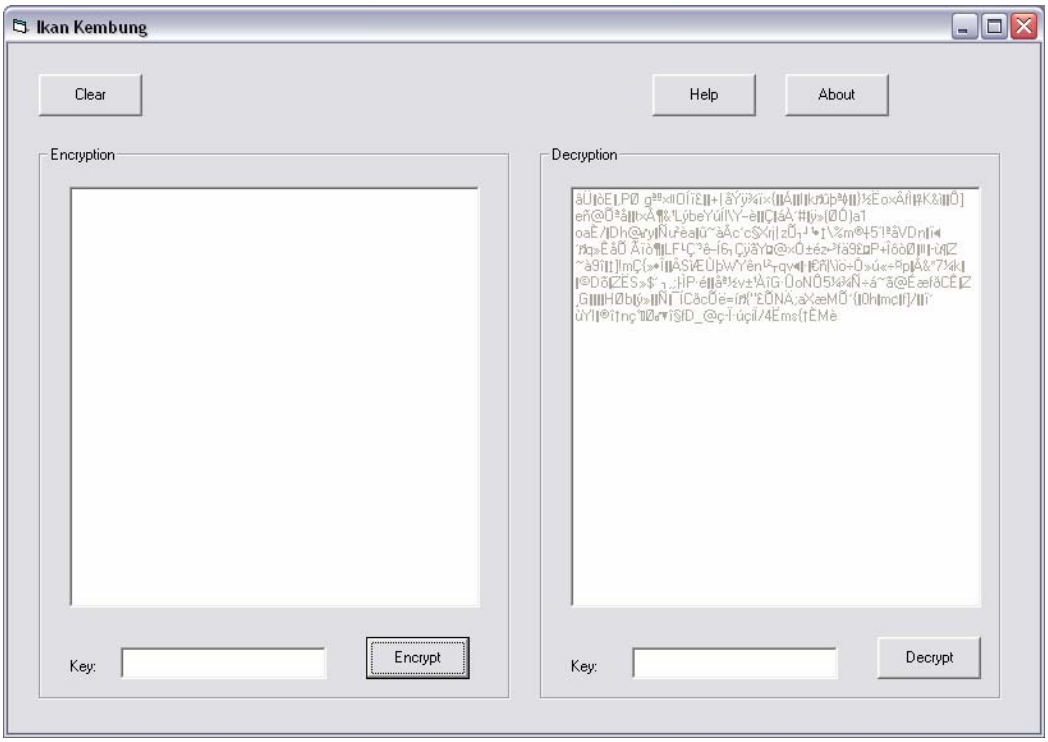
Antarmuka Aplikasi



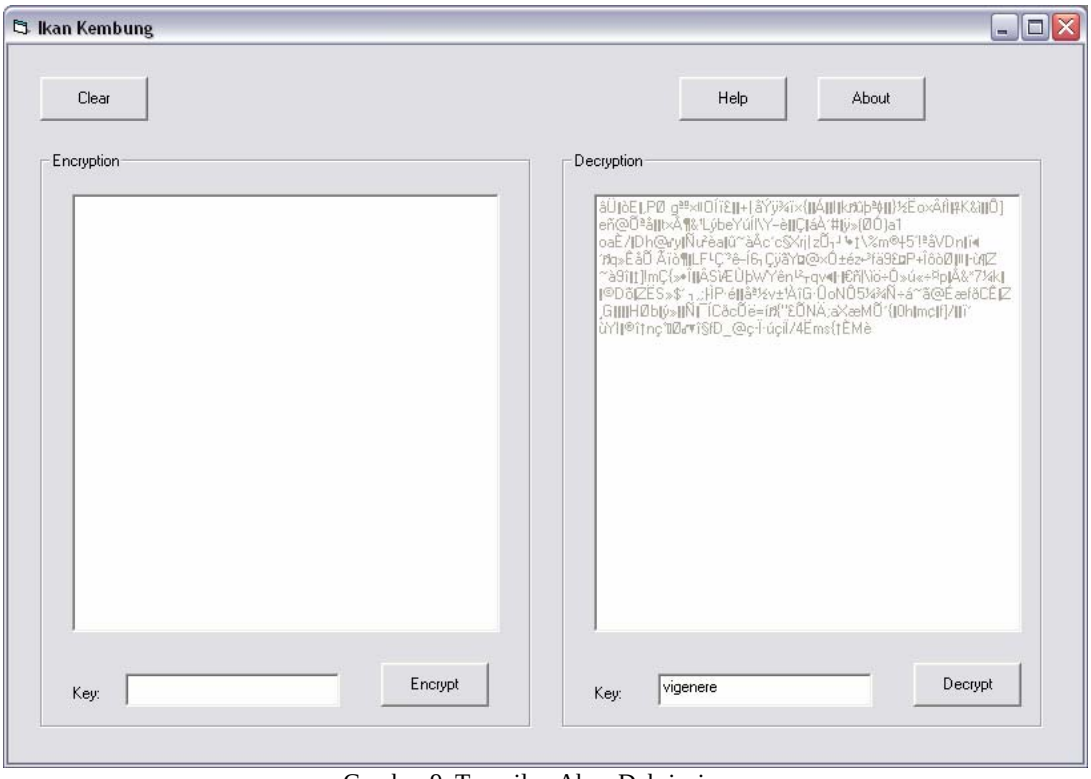
Gambar 6. Tampilan Awal Ikan Kembung



Gambar 7. Tampilan Akan Enkripsi

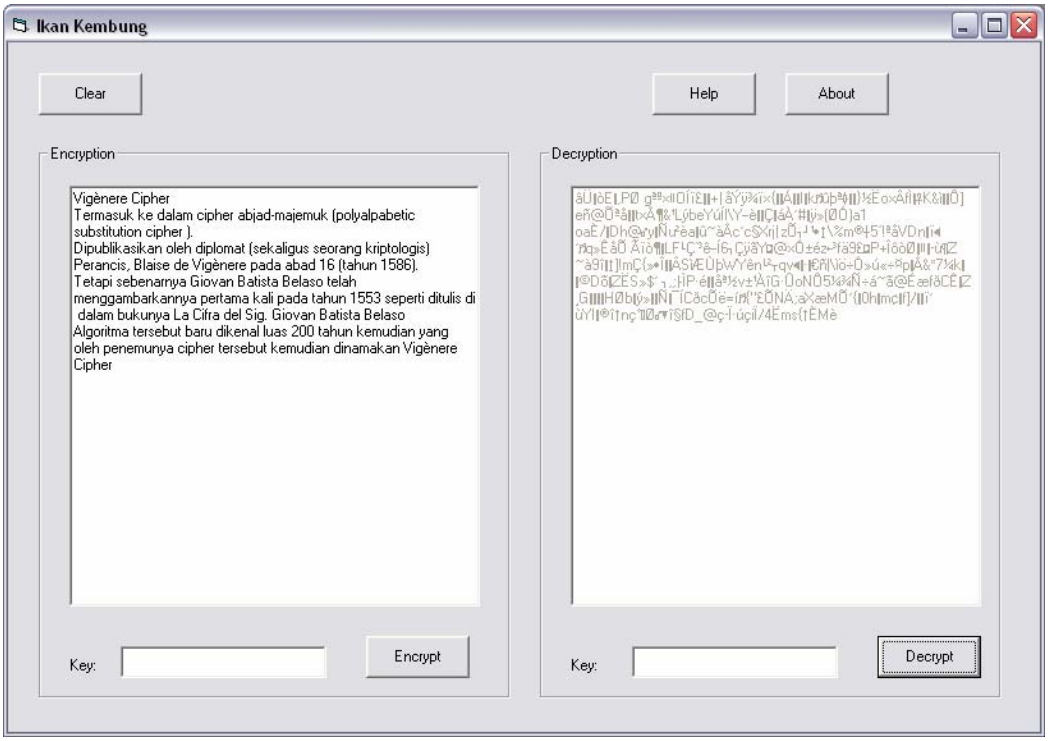


Gambar 8. Tampilan Sesudah Enkripsi

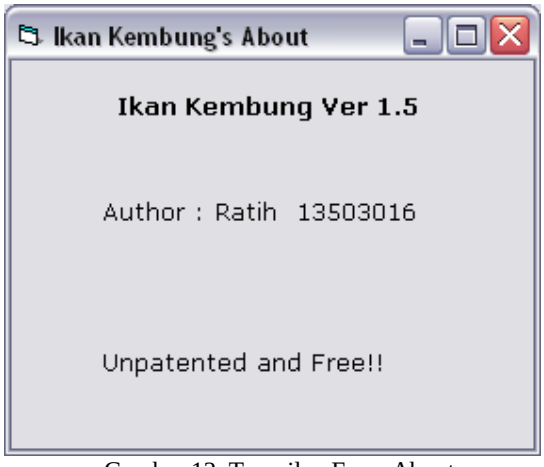


Gambar 9. Tampilan Akan Dekripsi

Gambar 10. Peringatan Memasukkan *Key*



Gambar 11. Tampilan sesudah Dekripsi



Gambar 12. Tampilan Form About

DAFTAR PUSTAKA

[1] Munir, Rinaldi. (2004). Bahan Kuliah IF5054 Kriptografi. Departemen Teknik Informatika, Institut Teknologi Bandung.

[2] Schneier, Bruce. (1996). Applied Cryptography 2nd. John Wiley & Sons.

[3] Wikipedia (2006), *http://en.wikipedia.org/wiki/Cryptograpy*. Tanggal akses : 5 Oktober 2006 pukul 12.00

[4] Modern Private Key Ciphers (1996), *http://williamstallings.com/Extras/Security-Notes/lectures/index.html*. Tanggal akses: 5 Oktober 2006 pukul 12.00

[5] Wikipedia (2006), *http://en.wikipedia.org/wiki/Blowfish_(cipher)*. Tanggal akses : 5 Oktober 2006 pukul 12.00

[6] Bruce Schneier (1994), *http://www.schneier.com/paper-blowfish-fse.html*. Tanggal akses : 5 Oktober 2006 pukul 12.00