ENKRIPSI FILE
MENGGUNAKAN METODE IDEA 64-BIT

Skripsi

Diajukan untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Teknik
Jurusan Teknik Informatika

Oleh :

Nama : Jerry Ercolesea Ho

NIM : 005314011

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS SANATA DHARMA
YOGYAKARTA

2004

FILE ENCRYPTION
USING IDEA ALGORITHM

By :

Name : Jerry Ercolesea Ho

N.I.M : 005314011

INFORMATION TECHNOLOGY
SANATA DHARMA UNIVERSITY
YOGYAKARTA

2004

HALAMAN PERSETUJUAN

SKRIPSI

ENKRIPSI FILE MENGGUNAKAN METODE IDEA 64-BIT

Oleh :
Jerry Ercolesea Ho
NIM : 0053140011

Telah disetujui oleh:

Pembimbing I

(Agnes Maria Polina, S.Kom, M.Sc.) Tanggal : 25 Okt 2004

Pembimbing II

(Puspaningtyas Sanjoyo Adi, S.T.) Tanggal : 23-10-2004

HALAMAN PENGESAHAN

SKRIPSI

FILE ENCRYPTION USING IDEA ALGORITHM

dipersiapkan dan disusun oleh:

Jerry Ercolesea Ho

NIM : 005314011

Telah dipertahankan di depan Panitia Penguji

pada tanggal 18 Oktober 2004 dan dinyatakan memenuhi syarat.

Susunan Panitia Penguji

Nama Lengkap	Tanda Tangan
Penguji I : Agnes Maria Polina, S.Kom., M.Sc.	.....
Penguji II : Puspaningtyas Sanjoyo Adi, S.T.	.....
Penguji III : Ridowati Gunawan, S.Kom., M.T.	.....
Penguji IV : DS. Bambang Soelistijanto, S.T.	.....

Yogyakarta, 18 Oktober 2004

Fakultas Teknik

Universitas Sanata Dharma

Dekan

Ir.Greg.Heliarko,SJ.,SS.,B.ST.,MA.,M.SC.

Skripsi ini dipersembahkan untuk :

My Great Lord, Jesus Christ

Papa

Mama

Jimmy

Elke

Terima kasih atas segala

Cinta

Dukungan

Kepercayaan

yang telah diberikan.

Jangan Pernah Berhenti untuk

Terus Menjadi Lebih Baik dari Dirimu Saat Ini.

Percaya pada Kemampuanmu Sendiri,

Ketakutan Hanya Mendatangkan Keterpurukan.

Percayalah pada Kebenaran maka Kebenaran akan Membelamu.

KATA PENGANTAR

Keamanan merupakan salah satu faktor yang penting dalam kehidupan. Hal serupa juga berlaku dalam pemakaian komputer. Para pengguna menginginkan data-data rahasia dapat disimpan dengan baik sehingga tidak semua orang dapat melihat informasi penting di dalamnya. Atas dasar itulah, maka penulis tertarik untuk membuat suatu program untuk menjaga keamanan data. Salah satu cara untuk mengatasi masalah tersebut adalah enkripsi. Enkripsi merupakan suatu proses mengacak isi data sehingga data tersebut tidak dapat dikenali. Banyak sekali algoritma enkripsi yang telah dibuat oleh para kriptografer. Keunggulan dari algoritma enkripsi biasanya dinilai dari banyaknya masukan kunci yang dipakai dalam proses enkripsinya, sehingga pemberian nilai masukan terhadap kunci dalam melakukan proses enkripsi juga membawa peranan penting. Kunci yang dimasukkan oleh pengguna sebaiknya merupakan karakter yang acak, karena algoritma sebaik apapun jika tidak ditunjang dengan masukan kunci yang sulit dikenali, maka akan sia-sia saja.

Pada kesempatan ini juga, saya ingin mengucapkan terima kasih secara tertulis kepada pihak-pihak berikut:

1. Ir. Greg. Heliarko, SJ., SS., B.ST., MA., M.SC. selaku Dekan Teknik Universitas Sanata Dharma.
2. Agnes Maria Polina, S.Kom, M.Sc. selaku Kaprodi Teknik Informatika Universitas Sanata Dharma sekaligus sebagai dosen pembimbing I, yang telah membantu saya dalam proses penulisan karya tulis ini.

3. Puspaningtyas Sanjoyo Adi, S.T. selaku dosen pembimbing II, yang telah memberikan masukan-masukan dalam mengembangkan sistem yang saya buat dan juga proses penulisan karya tulis ini.
4. Papa, Mama, Jimmy, Elke, atas semua yang telah diberikan kepada saya.
5. Pak Riyad yang pernah menjadi dosen pembimbing saya, yang juga telah memberi saya masukan dalam mengembangkan sistem. Andalah dosen yang pertama kali melihat program yang saya buat. Saya sudah perbaiki program saya sesuai permintaan Bapak, semoga Anda puas dengan program yang saya buat.
6. Pak Wawan, Bu Rido, Pak Antok, Pak Donny, Pak Agung, Bu Merry, dan segenap dosen Teknik Informatika Sanata Dharma, atas segala ilmu yang telah diberikan kepada saya.
7. Pak Belle, Mas Danang, dan Mas Catur yang telah membantu saya dalam kelancaran proses studi saya.
8. Teman-teman kuliah yang selalu berada di dekat saya, yang telah memberikan saya kenangan indah dan berbagi ilmu.
9. Nicko, my best friend, atas dukungan, masukan, pengetahuan-pengetahuan yang tidak saya dapat di kelas dan kebersamaannya.
10. dan kepada pihak-pihak yang tidak disebutkan namun turut membantu dalam proses pengembangan karya tulis ini.

Yogyakarta, 20 Oktober 2004

Jerry Ercolesea Ho

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

INTISARI

Topik penulisan ini adalah tentang program enkripsi dan dekripsi suatu berkas. Enkripsi terhadap suatu berkas dilakukan untuk mengubah nilai tiap *byte*-nya sehingga isi pokok dari berkas tersebut dapat disamarkan. Sedangkan proses dekripsi dilakukan untuk mengembalikan isi pokok dari berkas sesuai dengan aslinya. Metode enkripsi yang digunakan adalah metode enkripsi dengan kunci tunggal menggunakan algoritma blok, IDEA (*International Data Encryption Algorithm*). IDEA dikembangkan pada tahun 1990 di Swiss oleh kriptografer ternama, James Massey dan Xuejia Lai. Algoritma ini bekerja pada blok-blok *plaintext* 64 bit. Panjang kunci yang digunakan adalah 128 bit.

Pada waktu pengguna akan melakukan enkripsi ataupun dekripsi, pengguna harus terlebih dahulu memilih berkas yang akan diproses. Berkas yang dipilih merupakan berkas dokumen dan grafis dengan tipe apapun. Hal ini disesuaikan dengan batasan uji coba yang telah ditetapkan, meskipun tidak menutup kemungkinan pengguna dapat mengenkripsi maupun mendekripsi berkas lain. Setelah pengguna telah memilih berkas yang akan diproses, maka selanjutnya pengguna menetapkan kunci yang akan dipakai dan mengkonfirmasikannya kembali. Kemudian pengguna dapat menentukan apakah berkas sumber akan dihapus atau tidak, serta menentukan apakah berkas hasil enkripsi atau dekripsi akan disimpan dalam direktori yang sama dengan berkas sumber atau tidak. Setelah itu proses enkripsi atau dekripsi dapat mulai dilakukan. Fasilitas yang ditambahkan dalam program ini adalah penambahan menu dalam *context menu Windows*. Fasilitas ini ditambahkan supaya pengguna dapat melakukan enkripsi dan dekripsi berkas secara langsung dengan me-*click* kanan berkas tersebut kemudian memilih proses yang akan dilakukan. Program ini dikembangkan dengan menggunakan *Microsoft Visual Basic* 6.0 dan *Windows* API (*Application Programming Interface*).

Dari hasil uji coba tampak bahwa program ini ternyata terbukti bekerja sesuai dengan algoritma IDEA 64-bit, proses enkripsi dan dekripsi terhadap semua jenis berkas dokumen dan grafis juga dapat berjalan dengan baik, dan waktu yang diperlukan untuk melakukan enkripsi dan dekripsi akan linier dengan ukuran berkas yang dipilih.

ABSTRACT

This project was about making a file encryption programming. Encryption for any files is done by changing every single byte of this value, so that substance of that file can be hided. Decryption is the reversed process of encryption. It changes the substance back into the original one. This program is using one of the symmetric algorithm that operates on the plaintext in group of bits, IDEA (*International Data Encryption Algorithm*). IDEA is developed by James Massey and Xuejia Lai in 1990, Swiss. This algorithm operates on 64-bit plaintext blocks. The key is 128 bits long.

When user wants to encrypt or decrypt, user must first choose the file which will be processed. It is made according to the border of the testing that have been decided, although it is not possible that user can encrypt and decrypt the other one. After user have choosed the file that will be processed, user can define the key that will be used and confirm it again. Then user can decide whether erase the source file or not, and decide whether the encrypted or decrypted file will be saved in the same directory with the source file or not. After that, encryption and decryption can be executed. There is a facility that was added on context menu *Windows*, so that the user can do encryption and decryption file just by right-click the file and then choose the process that he/she wants to do. This program is developed using *Microsoft Visual Basic* 6.0 and *Windows* API (*Application Programming Interface*).

From the test result, it seems that this program has been approved working like IDEA 64-bit algorithm, encryption and decryption can be ran so well to any type of files, and the time is needed to encrypt and decrypt will be linear with the size of the chosen file.

BAB I
PENDAHULUAN

1.1 Latar Belakang Masalah

Keamanan merupakan suatu hal yang sangat penting dalam kehidupan sehari-hari. Faktor keamanan inilah yang selalu dilihat dan dicari oleh banyak orang. Begitu juga dalam penggunaan komputer, keamanan sangat diperlukan dalam menjaga kerahasiaan berkas-berkas penting pemiliknya. Banyak program pengaman telah dibuat untuk menangani masalah ini dan program tersebut semakin lama semakin canggih. Tidak hanya menggunakan perangkat-lunak tetapi juga menggunakan perangkat-keras di dalam penerapannya.

Dari latar belakang itulah, penulis tertarik untuk membuat suatu perangkat-lunak yang dapat memproteksi berkas-berkas penting yang telah disimpan dalam *harddisk*. Proteksi tersebut akan dilakukan dengan cara mengacak isi dari berkas, hal ini sering disebut sebagai enkripsi. Algoritma enkripsi yang dipilih dalam tugas akhir ini adalah algoritma blok enkripsi dengan kunci-tunggal, IDEA (*International Data Encryption Algorithm*).

Fasilitas enkripsi ini akan ditempatkan pada bagian *context menu* dalam sistem operasi *Windows*. *Context menu* merupakan menu yang terdapat dalam *Windows* yang akan ditampilkan pada saat pengguna mengklik-kanan berkas terpilih. Untuk mengaktifkan program proteksi ini, pengguna dapat langsung mengklik-kiri pilihan enkripsi atau dekripsi.

Pilihan enkripsi akan mengacak isi dari berkas, sedang dekripsi akan mengembalikan isi dari berkas seperti semula. Pada saat melakukan enkripsi, pengguna diminta mengisikan kata kunci yang digunakan untuk mengacak isi dari berkas, yang sekaligus juga akan digunakan untuk mengembalikan isi dari berkas itu sewaktu pengguna melakukan dekripsi.

Dengan adanya program ini, pemilik komputer diharapkan dapat merasa lebih aman dengan terjaganya kerahasiaan berkas-berkas yang ada di dalam komputer miliknya.

1.2 Rumusan Masalah

Pertanyaan yang timbul dalam rencana pembuatan program enkripsi *file* ini diantaranya:

1. Bagaimana mengimplementasikan enkripsi *file* untuk mengamankan berkas?
2. Bagaimana mengimplementasikan algoritma IDEA untuk enkripsi *file*?
3. Bagaimana mengimplementasikan aplikasi enkripsi *file* yang disatukan dengan sistem *registry Windows*?
4. Apakah program dapat digunakan dengan baik untuk mengenkripsi maupun mendekripsi berkas dokumen dan berkas grafis?
5. Apakah ada perbedaan waktu proses untuk tipe berkas yang sama dengan ukuran yang berbeda?

1.3 Batasan Masalah

Terdapat batasan masalah pada pembuatan sistem ini mengenai kemampuan dari program yang dibuat dan kegiatan uji cobanya seperti:

1. Sistem dibuat untuk pemakaian *single user*.
2. Kunci yang dipakai tidak disimpan dalam suatu berkas tertentu ataupun disimpan dalam berkas yang terenkripsi. Hal ini mengakibatkan berkas yang terenkripsi tidak akan kembali seperti aslinya, jika pengguna melakukan kesalahan dalam memasukkan kunci sewaktu mendekripsi berkas tersebut.
3. Uji coba hanya dilakukan pada berkas dokumen (.txt, .doc, .rtf) dan file grafis (.jpg, .bmp, .psd).

1.4 Tujuan Penulisan

Manfaat yang diharapkan dari tugas akhir ini adalah:

1. Membuat program yang dapat menjaga kerahasiaan berkas.
2. Menerapkan enkripsi dengan algoritma IDEA dalam sistem keamanan berkas.
3. Mengetahui lebih banyak tentang *register-register* dalam *Windows*.

1.5 Metodologi Penelitian

a. Studi Pustaka

Hal pertama yang dilakukan adalah mempelajari metode-metode enkripsi yang ada dan memilih yang terbaik, yang juga disesuaikan dengan kemampuan pemahaman dari penulis. Menurut Bruce Schneier (1994), metode enkripsi dengan algoritma blok yang terbaik adalah IDEA. Setelah ditemukan metode enkripsi yang tepat, maka langkah selanjutnya adalah mempelajari struktur, isi, dan fungsi dari register-register dalam *Windows*. Kemudian mempelajari penggunaan kode-kode program dalam VB yang berkaitan dengan operasi terhadap suatu berkas dan register.

Operasi terhadap suatu berkas dan register ternyata dapat ditemui dalam pemrograman *Windows* API (*Application Programming Interface*). Sumber pustaka didapat dari buku-buku pendukung dan *internet*.

b. Uji Listing Program Yang Beredar

Penulis juga melakukan pengamatan dengan cara mempelajari kode-kode program dari berbagai macam metode enkripsi blok yang telah ada. Hal ini dilakukan untuk mengetahui cara kerja dari suatu enkripsi dalam lingkungan pemrograman secara garis besar. Kode-kode program ini juga didapat dari buku-buku panduan dan *internet.*

c. Membuat Program untuk Enkripsi dan Dekripsi Berkas dengan Menggunakan Metode IDEA 64-bit

Setelah dilakukan studi terhadap metode enkripsi IDEA dan pengamatan terhadap kode-kode program dari berbagai macam metode enkripsi blok, maka penulis mulai membuat program enkripsi dan dekripsi untuk masukan yang berupa teks. Setelah program ini berjalan dengan baik terhadap suatu teks, penulis mulai mengembangkan program ini untuk diimplementasikan terhadap suatu berkas.

d. Menguji Coba Program

Uji coba terhadap program enkripsi dan dekripsi ini dilakukan untuk:

1) Membuktikan bahwa program berjalan sesuai dengan algoritma yang ada.
2) Menganalisis apakah program berjalan dengan baik terhadap berkas dokumen maupun berkas grafis.
3) Menganalisis apakah ukuran suatu berkas berpengaruh terhadap waktu proses program ini.

1.6 Sistematika Penulisan

BAB I PENDAHULUAN

Mendeskripsikan mengenai latar belakang masalah, rumusan masalah, batasan masalah, tujuan penulisan, sistematika penulisan, metodologi penelitian, dan Ketentuan dalam penulisan.

BAB II LANDASAN TEORI

Penjelasan mengenai teori yang digunakan penulis dalam proses pembuatan karya tulis ini.

BAB III ANALISIS dan PERANCANGAN SISTEM

Deskripsi mengenai analisis dan rancangan sistem yang dibuat.

BAB IV IMPLEMENTASI

Deskripsi mengenai tahap penulisan program dan implementasi rancangan sistem ke sistem yang sesungguhnya.

BAB V UJI COBA dan ANALISA HASIL

Deskripsi mengenai uji coba terhadap program dan analisa hasil dari uji coba tersebut.

BAB V KESIMPULAN dan SARAN

DAFTAR PUSTAKA

LAMPIRAN

BAB II

LANDASAN TEORI

2.1 Enkripsi

2.1.1 Pengenalan Enkripsi

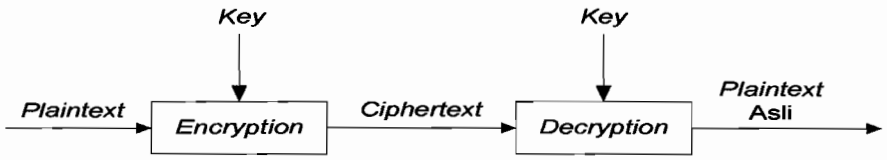
Beberapa definisi dari enkripsi dapat dinyatakan sebagai berikut:

1) Suatu proses yang melakukan perubahan kode-kode dari yang bisa dimengerti menjadi kode-kode yang tidak bisa dimengerti (Wahana Komputer, 2003).
2) Suatu proses menyamarkan pesan dengan cara menyembunyikan isi pokoknya (Schneier, 1994).
3) Suatu proses mengkonversi/mengubah data ke dalam bentuk kode-kode tertentu, dengan tujuan informasi yang disimpan maupun ditransmisikan melalui jaringan yang tidak aman (misalnya *Internet*), tidak dapat dibaca oleh siapapun kecuali oleh orang-orang yang berhak.

Pesan/data yang telah dienkripsi disebut *ciphertext*. Untuk mengembalikan isi dari pesan/data menjadi kode-kode yang bisa dimengerti, sehingga bisa terbaca kembali, maka dilakukan suatu proses dekripsi. Pesan/data yang belum dienkripsi atau telah didekripsi disebut *plaintext*. Proses perubahan pesan/data diatas dapat digambarkan sebagai berikut:

Gambar 2.1 Proses enkripsi-dekripsi

Algoritma kriptografi, atau sering disebut juga *cipher*, merupakan fungsi matematis yang digunakan untuk proses enkripsi dan dekripsi. Supaya benar-benar aman, semua algoritma enkripsi yang moderen menggunakan suatu kunci. Kunci ini dapat menerima satu atau banyak nilai. Semakin besar nilai kunci tersebut, semakin aman pula pesan/data yang terenkripsi. Tetapi penentuan besar nilai kunci ini didasarkan pada algoritma enkripsi yang berlaku. Nilai kunci yang terlalu besar bagi suatu algoritma akan membuat proses enkripsi menjadi semakin lambat. Jadi, keamanan suatu data yang terenkripsi akan meliputi dua bagian yang sangat penting: kekuatan atau kehandalan algoritma enkripsi dan kerahasiaan kunci. Proses enkripsi-dekripsi dengan menggunakan kunci dapat digambarkan sebagai berikut:



Gambar 2.2 Proses enkripsi-dekripsi dengan kunci

Ada dua bentuk umum dari algoritma enkripsi berdasarkan kunci yang dipakai, yaitu: simetris dan kunci-publik (Schneier, 1994).

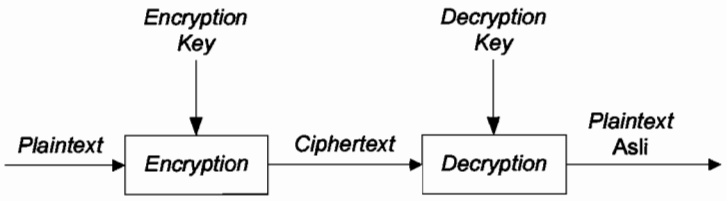
1) Algoritma simetris

   Algoritma simetris adalah algoritma yang menggunakan kunci yang sama dalam proses enkripsi maupun dekripsi. Algoritma ini juga sering disebut algoritma kunci-rahasia, algoritma kunci-tunggal, atau algoritma satu-kunci.

Algoritma simetris dapat dibagi menjadi dua kategori. Kategori pertama adalah algoritma simetris yang mengenkripsi *plaintext* ke dalam suatu bit tunggal dalam suatu waktu. Algoritma ini disebut algoritma *stream* atau *stream ciphers*. Kategori kedua adalah algoritma simetris yang mengenkripsi *plaintext* ke dalam suatu blok bit. Algoritma ini disebut algoritma blok atau *block ciphers*.

2) Algoritma kunci-publik

Algoritma kunci-publik adalah algoritma yang menggunakan kunci yang berbeda antara proses enkripsi dengan proses dekripsinya. Algoritma ini dapat digambarkan sebagai berikut:



Gambar 2.3 Proses enkripsi-dekripsi dengan algoritma kunci-publik

2.1.2 IDEA (International Data Encryption Algorithm)

2.1.2.1 Pengantar

IDEA dikembangkan pada tahun 1990 di Swiss oleh kriptografer ternama, James Massey dan Xuejia Lai. IDEA adalah algoritma blok. Algoritma ini bekerja pada blok-blok *plaintext* 64 bit. Panjang kunci yang digunakan adalah 128 bit. Algoritma ini dipakai dalam proses enkripsi dan proses dekripsi (Schneier, 1994). Filosofi disain yang melatar-belakangi algoritma ini adalah penggabungan operasi dari kelompok aljabar yang berbeda.

Ada tiga kelompok aljabar yang pengoperasiannya digabungkan, yaitu: XOR, penambahan modulo 2^{16} (penambahan dengan mengabaikan *overflow*), dan perkalian modulo $2^{16} + 1$ (perkalian dengan mengabaikan *overflow*). Semua operasi ini akan bekerja pada sub-blok 16 bit.

2.1.2.2 Cara Kerja IDEA

Ada empat proses yang terjadi sewaktu melakukan enkripsi berkas, yaitu: inisialisasi kunci enkripsi, membaca data asli, operasi pengacakan dan pengembalian data, serta menulis data terenkripsi.

1) Inisialisasi kunci enkrispi

Algoritma ini awalnya menggunakan blok kunci 128 bit. Blok tersebut kemudian dibagi menjadi delapan sub-blok 16 bit. Delapan sub-blok kunci tersebut diperoleh dari kunci masukan yang berupa karakter. Karakter ini terlebih dahulu harus dijadikan bilangan *integer* sesuai nilainya dalam sistem pengkodean ASCII (*The American Standard Code for International Interchange*). Tiap satu karakter mempunyai besar 8 bit atau 1 *byte*. Padahal nilai dari kunci ini akan dimasukkan dalam perhitungan dengan blok data yang besarnya 16 bit atau 2 *byte*, sehingga perlu dilakukan penggeseran bit ke kiri sebanyak 8 bit supaya menjadi blok kunci 16 bit. Cara ini hanya berlaku untuk karakter pertama pada kunci, untuk karakter yang lain caranya dengan menggeser karakter ke-(n) ke kiri sebanyak 8 bit dan me-OR-kannya dengan kunci ke-(n-1) yang digeser ke kanan sebanyak 8 bit. Contoh penggambaran proses ini adalah sebagai berikut:

Kunci : 1234

Konversi kunci ke dalam ASCII:

1	49
2	50
3	51
4	52

Jika blok kunci ke-x = Bk(x), maka

Bk(0) = | 49 | 0 | = 49<<8 = 12544

Bk(1) = | 50 | 49 | = (50<<8) OR 49 = 12849

Bk(2) = | 51 | 50 | = (51<<8) OR 50 = 13106

Bk(3) = | 52 | 51 | = (52<<8) OR 51 = 13363

Lambang : << adalah operasi penggeseran bit ke kiri dan >> adalah operasi penggeseran bit ke kanan.

Dari contoh tersebut dapat dilihat bahwa hanya terdapat empat sub-blok kunci, padahal sub-blok kunci yang dibutuhkan adalah delapan, maka empat sub-blok kunci yang lain diset menjadi 0. Jadi delapan sub-blok kunci yang pertama adalah:

Bk(0) = 12544 Bk(4) = 0

Bk(1) = 12849 Bk(5) = 0

Bk(2) = 13106 Bk(6) = 0

Bk(3) = 13363 Bk(7) = 0

Total sub-blok kunci yang dibutuhkan dalam algoritma ini adalah 52. Enam sub-blok kunci digunakan dalam delapan putaran operasi aljabar dan empat sub-blok kunci lainnya akan digunakan pada akhir proses.

Blok-blok kunci yang digunakan tidak sama dengan blok kunci awal. Blok kunci awal akan dirotasikan sebanyak 25 bit ke kiri. Cara merotasikannnya adalah sebagai berikut:

Jika sub-blok kunci ke-x = Bk(x) dan y adalah himpunan bilangan kelipatan 8 dari deretan bilangan x dengan bilangan awal sama dengan 0, maka:

$0 \leq i \leq 7$ dan $7 < x < 52$;

y = {0, 8, 16, 24, 32, 40, 48}, tiap satu bilangan y ke-n digunakan untuk delapan bilangan x ke-n pula.

Bk(x) = Bk(((i + 1) AND 7) + y) << 9 OR Bk(((i + 2) AND 7) + y) >> 7

Harus ditegaskan kembali bahwa algoritma ini akan selalu mengabaikan kondisi *overflow* pada setiap operasinya. Berikut adalah contoh dari proses rotasi ini.

Bk(8) = Bk(((0+1) AND 7) + 0) << 9 OR Bk(((0+2) AND 7) + y) >> 7

Bk(8) = Bk(1) << 9 OR Bk(2) >> 7

Bk(8) = 12849 << 9 OR 13106 >> 7

Bk(8) = 25088 OR 102 = 25190

Jawaban dari contoh di atas akan sama dengan jawaban yang diperoleh dengan cara merotasikan blok kunci 25 bit ke kiri secara langsung. Pembuktiannya adalah sebagai berikut:

31	00	32	31	33	32	34	33	00	00	00	00	00	00	00	00

Konversi ke biner menjadi:

0011 0001	0000 0000	0011 0010	0011 0001	0011 0011	0011 0010	0011 0100	0011 0011	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000

Rotasi 25 bit ke kiri menjadi:

0110 0010	0110 0110	0110 0100	0110 1000	0110 0110	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000	0110 0010	0000 0000	0110 0100

Konversi ke heksadesimal menjadi:

62	66	64	68	66	00	00	00	00	00	00	00	00	62	00	64

Terbukti bahwa Bk(8) = 6266 = 25190.

Proses rotasi blok kunci ini akan berlaku sama hingga mendapatkan 52 sub-blok kunci seperti terlihat pada lampiran A dan B.

2) Membaca data asli

Algoritma ini memerlukan 64 bit blok data dalam pengoperasiannya. Blok data ini dibagi menjadi empat sub-blok data 16 bit yang akan menjadi masukan untuk putaran pertama pada operasi yang berlaku dalam algoritma ini. Tiap sub-blok data diperoleh dengan cara membaca data sebanyak 1 *byte* dan menggesernya 8 bit ke kiri. Kemudian 1 *byte* data berikutnya dibaca dan di-OR-kan dengan sub-blok data sebelumnya. Sehingga dalam satu sub-blok data berisi satu atau dua karakter. Berikut adalah contoh proses membaca data asli:

Data : Dimana

Konversi data ke ASCII:

D	68
i	105
m	109
a	97
n	110
a	97

Jika blok data ke-x = Bd(x), maka:

Bd(0) = | D | i | = (68<<8) OR 105 = 17513

Bd(1) = | m | a | = (109<<8) OR 97 = 28001

Bd(2) = | n | a | = (110<<8) OR 97 = 28257

Dari contoh tersebut dapat dilihat bahwa hanya terdapat tiga sub-blok data, padahal sub-blok data yang dibutuhkan adalah empat, maka satu sub-blok data yang tersisa diset menjadi 0. Jadi empat sub-blok data tersebut adalah:

Bd(0) = 17513

Bd(1) = 28001

Bd(2) = 28257

Bd(3) = 0

Jika data yang ada lebih dari 8 *byte*, maka dilakukan pembacaan data sebanyak 8 *byte* lagi. Hal ini dapat dilakukan setelah proses enkripsi terhadap data 8 *byte* sebelumnya selesai. Proses ini akan dilakukan hingga data yang ada telah terenkripsi semua.

3) Operasi pengacakan data

Operasi ini dapat dilakukan setelah sub-blok kunci dan sub-blok data diketahui. Ada totalnya delapan putaran. Dalam tiap putaran, empat sub-blok data akan di-XOR-kan, ditambah, dan dikali dengan yang lainnya dan dengan enam buah sub-blok kunci.

Jika sub-blok kunci adalah K_i dan sub-blok data adalah X_n, maka tiap putaran akan melewati operasi-operasi aljabar yang sama seperti di bawah ini:

$X_1 = X_1 * K_1$
$X_2 = X_2 + K_2$
$X_3 = X_3 + K_3$
$X_4 = X_4 * K_4$
$t_1 = X_1 \text{ XOR } X_3$
$t_2 = X_2 \text{ XOR } X_4$
$t_1 = t_1 * K_5$
$t_2 = t_2 + t_1$
$t_2 = t_2 * K_6$
$t_1 = t_1 + t_2$

$X_1 = X_1$ XOR t_2
$X_3 = X_3$ XOR t_2
$X_2 = X_2$ XOR t_1
$X_4 = X_4$ XOR t_1

Keluaran dari putaran tersebut adalah X_1, X_2, X_3, dan X_4. Untuk putaran selanjutnya, masukannya akan menjadi:

$X_1 = X_1$
$X_2 = X_3$
$X_3 = X_2$
$X_4 = X_4$

Namun proses pergantian di atas tidak dilakukan untuk putaran terakhir. Setelah delapan putaran, ada transformasi keluaran terakhir sebagai berikut:

1) $X_1 = X_1 * K_1$
2) $X_2 = X_2 + K_2$
3) $X_3 = X_3 + K_3$
4) $X_4 = X_4 + K_4$

Akhirnya, empat sub-blok tersebut dipasang kembali untuk menghasilkan *ciphertext/plaintext*. Jika masukan sub-blok kunci dan sub-blok data sama seperti dicontohkan sebelumnya, maka akan dihasilkan data terenkripsi sebagai berikut:

$X_1 = 45154$
$X_2 = 25559$
$X_3 = 11565$
$X_4 = 29909$

Hasil diatas diperoleh dari perhitungan yang dapat dilihat dalam lampiran C dan D.

4) Menulis data terenkripsi

Hasil dari operasi pengacakan data adalah empat sub-blok data terenkripsi. Keempat sub-blok data ini akan ditulis per *byte*. Cara penulisannya adalah dengan menggeser ke kanan nilai sub-blok data sebanyak 8 bit untuk menghasilkan nilai *byte* pertama dari sub-blok data, dan me-AND-kan nilai sub-blok data tadi dengan 255 untuk menghasilkan nilai *byte* kedua dari sub-blok data ini. Berikut adalah contoh dari penulisan data ini:

$X_1 = 45154$

Nilai *byte* pertama = $X_1 >> 8 = 45154>>8 = 176$

Nilai *byte* kedua = X_1 AND 255 = 45154 AND 255 = 98

$X_2 = 25559$

Nilai *byte* ketiga = $X_2 >> 8 = 25559>>8 = 99$

Nilai *byte* keempat = X_2 AND 255 = 25559 AND 255 = 215

$X_3 = 11565$

Nilai *byte* kelima = $X_3 >> 8 = 11565>>8 = 45$

Nilai *byte* keenam = X_3 AND 255 = 11565 AND 255 = 45

$X_4 = 29909$

Nilai *byte* ketujuh = $X_4 >> 8 = 29909>>8 = 116$

Nilai *byte* kedelapan = X_4 AND 255 = 29909 AND 255 = 213

176	▒
98	b
99	c
215	╫
45	-
45	-
116	t
213	F

Untuk mendekripsi suatu berkas juga terdapat empat proses yang harus dilalui, yaitu: inisialisasi kunci dekripsi, membaca data terenkripsi, operasi pengembalian data, serta menulis data asli.

1) Inisialisasi kunci dekripsi

Inisialisasi kunci pada proses dekripsi dilakukan dengan cara mencari terlebih dahulu 52 sub-blok kunci yang dipakai pada proses enkripsi, kemudian 52 sub-blok kunci tersebut diubah penjadwalannya seperti yang terlihat pada Tabel 2.1.

Tabel 2.1 Tabel perbandingan sub-blok kunci enkripsi dengan dekripsi

PUTARAN	SUB-BLOK KUNCI ENKRIPSI	SUB-BLOK KUNCI DEKRIPSI
1	Z_{11} Z_{12} Z_{13} Z_{14} Z_{15} Z_{16}	$1/Z_{91}$ $-Z_{92}$ $-Z_{93}$ $1/Z_{94}$ Z_{85} Z_{86}
2	Z_{21} Z_{22} Z_{23} Z_{24} Z_{25} Z_{26}	$1/Z_{81}$ $-Z_{82}$ $-Z_{83}$ $1/Z_{84}$ Z_{75} Z_{76}
3	Z_{31} Z_{32} Z_{33} Z_{34} Z_{35} Z_{36}	$1/Z_{71}$ $-Z_{72}$ $-Z_{73}$ $1/Z_{74}$ Z_{65} Z_{66}
4	Z_{41} Z_{42} Z_{43} Z_{44} Z_{45} Z_{46}	$1/Z_{61}$ $-Z_{62}$ $-Z_{63}$ $1/Z_{64}$ Z_{55} Z_{56}
5	Z_{51} Z_{52} Z_{53} Z_{54} Z_{55} Z_{56}	$1/Z_{51}$ $-Z_{52}$ $-Z_{53}$ $1/Z_{54}$ Z_{45} Z_{46}
6	Z_{61} Z_{62} Z_{63} Z_{64} Z_{65} Z_{66}	$1/Z_{41}$ $-Z_{42}$ $-Z_{43}$ $1/Z_{44}$ Z_{35} Z_{36}
7	Z_{71} Z_{72} Z_{73} Z_{74} Z_{75} Z_{76}	$1/Z_{31}$ $-Z_{32}$ $-Z_{33}$ $1/Z_{34}$ Z_{25} Z_{26}
8	Z_{81} Z_{82} Z_{83} Z_{84} Z_{85} Z_{86}	$1/Z_{21}$ $-Z_{22}$ $-Z_{23}$ $1/Z_{24}$ Z_{15} Z_{16}
Output	Z_{91} Z_{92} Z_{93} Z_{94}	$1/Z_{11}$ $-Z_{12}$ $-Z_{13}$ $1/Z_{14}$

Contoh inisialisasi kunci dekripsi dapat dilihat dalam lampiran D dan E.

2) Membaca data terenkripsi

Proses pembacaan data terenkripsi sama dengan proses pembacaan data asli. Yang membedakannya adalah proses pembacaan data terenkripsi akan selalu dilakukan tepat 8 *byte* sehingga empat sub-blok data akan selalu bernilai lebih dari 0. Berikut adalah contoh proses membaca data terenkripsi:

Data : ▒bc╫--t F

Konversi data ke ASCII:

▒	176
b	98
C	99
╫	215

-	45
-	45
t	116
F	213

Jika blok data ke-x = Bd(x), maka:

Bd(0) = (176<<8) OR 98 = 45154

Bd(1) = (99<<8) OR 215 = 25559

Bd(2) = (45<<8) OR 45 = 11565

Bd(3) = (116<<8) OR 213 = 29909

3) Operasi pengembalian data

Operasi pengembalian data ini sama dengan operasi pengacakan data sewaktu melakukan enkripsi. Jika masukan sub-blok data dan sub-blok kunci sesuai dengan yang dicontohkan sebelumnya, maka akan dihasilkan data asli sebagai berikut:

$X_1 = 17513$

$X_2 = 28001$

$X_3 = 28257$

$X_4 = 0$

Hasil tersebut diperoleh dari perhitungan yang dapat dilihat dalam lampiran E dan F.

4) Menulis data asli

Proses penulisan data asli sama dengan proses penulisan data terenkripsi. Yang membedakannya adalah proses pembacaan data asli akan dilakukan sebanyak ukuran data asli. Berikut adalah contoh proses menulis data asli:

$X_1 = 17513$

Nilai *byte* pertama = X_1 >> 8 = 17513>>8 = 68

Nilai *byte* kedua = X_1 AND 255 = 17513 AND 255 = 105

$X_2 = 28001$

Nilai *byte* ketiga = X_2 >> 8 = 28001>>8 = 109

Nilai *byte* keempat = X_2 AND 255 = 28001 AND 255 = 97

$X_3 = 28257$

Nilai *byte* kelima = X_3 >> 8 = 28257>>8 = 110

Nilai *byte* keenam = X_3 AND 255 = 28257 AND 255 = 97

68	D
105	i
109	m
97	a
110	n
97	a

2.2 Pengenalan *Registry Windows*

2.2.1 Pendahuluan

Registry Windows adalah basis data terpusat yang dipakai oleh *Windows* dan aplikasi-aplikasi *Windows* untuk menyimpan informasi konfigurasinya. Informasi ini berisi sistem perangkat-keras, perangkat-lunak, dan aplikasi komunikasi. Setiap program yang ada dapat mengubah dan mengambil informasi ini bilamana diperlukan (Amperiyanto, 2001).

Registry secara prinsip akan menyimpan dua jenis informasi yang mendasar, yaitu:

a. Informasi global

   Informasi ini penting untuk operasi dasar dari perangkat-keras dan perangkat-lunak pada komputer.

b. Informasi spesifikasi pengguna

Bagian ini berisi pengaturan kostumasi pemakai, seperti tampilan dari *desktop Windows* dan pilihan-pilihan yang telah diatur oleh pemakai.

Registry sendiri disimpan dalam dua berkas, yaitu: system.dat dan user.dat.

a. System.dat

System.dat akan diakses oleh *Windows* pada saat komputer melakukan *booting*. Berkas ini berisi data-data khusus yang berhubungan dengan sistem komputer.

b. User.dat

User.dat akan diakses oleh *Windows* setelah system.dat diakses. Berkas ini berisi informasi pemakai yang spesifik, seperti *password* dan preferensi aplikasi individual.

2.2.2 Kunci *Registry*

Registry secara global terdiri dari enam kategori informasi yang disebut dengan kunci induk (*root keys*). Kunci induk ini seperti kontainer data, setiap informasi terhubung dengan aspek tertentu pada sistem komputer. Kunci induk ini dapat berisi nilai (*value*), kunci itu sendiri (*key*), atau sub-kunci (*subkey*). Kunci induk ini sering disebut juga sebagai *handle keys*.

Enam kunci induk dalam *registry Windows* adalah:

1) HKEY_CLASSES_ROOT

Bagian ini merupakan tempat dimana seluruh asosiasi berkas disimpan. Beberapa sub-kunci yang ada dalam bagian ini, yaitu:

a. *Default Icon* merupakan sub-kunci yang berisi *icon* untuk suatu jenis berkas.

b. Shell digunakan untuk menyimpan perintah manipulasi berkas. Perintah yang sering digunakan adalah membuka suatu berkas dengan ekstensi tertentu dan menampilkan perintah tersebut dalam context menu. Namun dalam sub-kunci ini harus terlebih dahulu ditambahkan sub-kunci yang dijadikan penanda perintah atau penamaan perintah, dan di bawah sub-kunci tadi ditambahkan sub-kunci dengan nama bawaan dari Windows, command. Dalam sub-kunci command inilah terdapat nilai yang akan menunjuk pada berkas eksekusi.

2) HKEY_CURENT_USER

Bagian ini menyimpan pengaturan individual untuk setiap pemakai komputer.

3) HKEY_LOCAL_MACHINE

Bagian ini menyimpan pengaturan global yang berhubungan dengan perangkat-keras, perangkat-lunak, dan komunikasi.

4) HKEY_USERS

Bagian ini menyimpan pengaturan spesifikasi pemakai. Pengaturan dapat berupa tampilan *Windows* maupun pengaturan-pengaturan yang dilakukan pada program-program aplikasi.

5) HKEY_CURENT_CONFIG

Bagian ini menyimpan informasi perangkat-keras yang terpasang dalam komputer.

6) HKEY_DYN_DATA

Kunci induk ini berisi informasi yang bersifat dinamis. Contohnya adalah informasi perangkat *Plug and Play*.

2.2.3 Nilai Registry

Nilai dalam *registry Windows* dapat dibagi menjadi dua, yaitu: nama yang dipakai sebagai identifikasi dari nilai, dan data yang digunakan untuk menyimpan informasi nilai. Tipe data dari nilai *registry* ada tiga macam:

a. String

Masukan data nilai yang berupa data teks, seperti informasi *path* dari suatu aplikasi.

b. DWORD

Masukan data nilai yang berupa angka. Angka yang bisa dimasukkan berupa desimal atau heksadesimal.

c. Binary

Masukan data nilai yang berupa angka 0 dan 1 saja.

2.3 Pengenalan VB6

Program VB6 adalah pemrograman berbasis *Microsoft Windows*. VB6 adalah bahasa pemrograman yang digunakan untuk membuat aplikasi *Windows* yang berbasis grafis (GUI-*Graphical User Interface*). VB merupakan *event-driven programming* (pemrograman terkendali kejadian) artinya program menunggu sampai adanya respon dari pemakai berupa kejadian (*event*) tertentu, seperti tombol diklik, menu dipilih, dan lain-lain. Ketika *event* terdeteksi, kode yang berhubungan dengan *event* (*event procedure*) akan dijalankan. VB juga termasuk bahasa pemrograman *Object Oriented Programming* (OOP), yaitu pemrograman yang berorientasi objek. Penulisan kode program dalam VB tidak memperhatikan huruf besar atau kecil.

2.3.1 Struktur Aplikasi VB

Struktur aplikasi VB ini dapat dilihat pada Tabel 2.2 berikut:

Tabel 2.2 Struktur aplikasi VB

Nama Bagian	Keterangan
Form	Jendela untuk membuat tampilan (*user interface*).
Control	Tampilan berbasis grafis yang dimasukkan pada *form* untuk membuat interaksi dengan pemakai (*text box*, *label*, *scroll bar*, tombol *command*).
Properties	Nilai/karakteristik yang dimiliki oleh sebuah objek VB. Contoh : *Name*, *Captions*, *Size*, *Color*, *Position* dan *Text*. VB menerapkan properti standar (*default*). *Properties* dapat diubah saat mendesain program atau run time.
Metode	Serangkaian perintah yang sudah tersedia pada suatu objek yang dapat diminta untuk mengerjakan tugas khusus.
Event Procedure	Kode yang berhubungan dengan suatu objek. Kode ini akan dieksekusi ketika ada respon dari pemakai berupa *event* tertentu.
General Procedure	Kode yang tak berhubungan dengan suatu objek. Kode ini harus diminta oleh aplikasi.
Modul	Kumpulan dari prosedur umum, deklarasi variabel dan definisi konstanta yang digunakan oleh aplikasi.

2.3.2 Langkah-langkah Mengembangkan Aplikasi

1. Membuat user interface/tampilan.
2. Mengatur properti.
3. Menulis kode program.

2.3.3 Dasar-dasar pemrograman VB

Dasar pemrograman dari VB hampir sama dengan bahasa pemrograman tingkat tinggi yang lain, yang berbeda hanya struktur penulisannya saja.

2.3.3.1 Variabel dan Konstanta

Variabel dan konstanta digunakan untuk menampung data sementara. Variabel dan konstanta tersebut lebih merupakan penamaan untuk suatu alamat dalam memori sistem. Ukuran penyimpanan variabel dan konstanta dalam memori diukur dalam satuan *byte*. VB mengenal banyak tipe data yang dapat dilihat pada Tabel 2.3.

Tabel 2.3 Tipe-tipe variabel dan konstanta pada VB

Tipe Data	Ukuran	Kisaran	Contoh
Integer	2 *byte*	-32.768 sampai 32.768	`Dim Bird%` `Bird% = 37`
Long Integer	4 *byte*	-2.147.483.648 sampai 2.147.483.647	`Dim Loan&` `Loan& = 350000`
Single-precision floating point	4 *byte*	-3.402823E38 sampai 3.402823E38	`Dim Price!` `Price! = 899.99`
Double-precision	8 *byte*	-1.79769313486232D308 sampai	`Dim Pi#` `Pi# = 3.1415926535`
Currency	8 *byte*	-92233720368547735808 sampai	`Dim Debt@` `Debt@ = 7600300.50`
String	1 *byte* per karakter	0 sampai 65.535	`Dim Dog$` `Dog$ = "pointer"`
Boolean	2 *byte*	True atau False	`Dim Flag as Boolean` `Flag = True`
Date	8 *byte*	1 Januari 100 sampai 31 Desember 9999	`Dim Birthday as Date` `Birthday = #3-1-63#`
Variant	16 *byte* (untuk angka); 22 *byte* + 1 *byte* (untuk string)	Semua tipe data	`Dim Total` `Total = 289.13`

Deklarasi variabel pada VB dapat digantikan dengan menambahkan suatu karakter tertentu di akhir nama variabelnya. Sehingga pendeklarasian tipe data suatu variabel di bawah ini adalah sama:

- `Dim I as Integer`
- `Dim I%`

VB mengijinkan pemakainya untuk membuat sendiri tipe datanya. Hal ini sangat berguna untuk mengelompokkan data–data yang terbagi menjadi beberapa kategori. Pembuatan tipe data tersebut menggunakan perintah '`Type`'.

Contoh pemakaian perintah '`Type`' adalah sebagai berikut:

```
Type Employee
      Name As String
      DateOfBirth As Date
      HireDate As Date
End Type
```

Tipe data yang berkelompok tersebut digunakan dengan cara:

```
Dim ProductManager as Employee
ProductManager.Name = "Erick Cody"
ProductManager.DateOfBirth = #3-6-1970#
ProductManager.HireDate = #8-1-2001#
```

Konstanta adalah serupa dengan variabel, namun nilai dari suatu konstanta telah diisikan di awal pendeklarasiannya. Nilai tersebut tidak dapat diubah selama program berjalan. Contoh pendeklarasian konstanta adalah sebagai berikut:

- `Const Dim I as Integer`
- `Const I = 100`

2.3.3.2 Operator dalam VB

Operator umum yang disediakan oleh VB adalah:

Tabel 2.4 Operator umum dalam VB

Operator	Kegunaan	Contoh
+	Penambahan	10 + 8 = 18
-	Pengurangan	10 – 8 = 2
*	Perkalian	10 * 8 = 80
/	Pembagian	10 / 8 = 1.25
\	Pembagian integer (angka bulat)	10 \ 8 = 1
Mod	Sisa pembagian	10 Mod 8 = 2
^	Pangkat	10 ^ 2 = 100
&	Menggabungkan string	"Hello" & "World" = "HelloWorld"

Khusus untuk operator matematis, urutan operator berguna saat menggunakan banyak operator matematis pada sebuah baris perintah. Maka VB pun dilengkapi dengan urutan eksekusi operator seperti yang dapat dilihat pada Tabel 2.5 di bawah ini.

Tabel 2.5 Prioritas operator umum pada VB

Prioritas	Operator
1	( )
2	^
3	-
4	* dan /
5	\
6	Mod
7	+ dan -

Jika operator dengan prioritas yang sama ditemukan, maka urutan operasinya dimulai dari paling depan. Contoh:

```
Total = 100 / 2 * 2
```

Akan menghasilkan nilai 100 bukan 25.

Operator lain yang disediakan dalam VB adalah operator perbandingan (Tabel 2.6) dan operator logika (Tabel 2.7).

Tabel 2.6 Operator perbandingan pada VB

Operator	Arti	Contoh
=	Sama dengan	10 = 10 adalah `True`
<>	Tidak sama dengan	10 <> 10 adalah `False`
>	Lebih besar dari	10 > 10 adalah `False`
<	Lebih kecil dari	10 < 10 adalah `False`
>=	Lebih besar dari atau sama dengan	10 >= 10 adalah `True`
<=	Lebih kecil dari atau sama dengan	10 <= 10 adalah `True`

Tabel 2.7 Operator logika pada VB

Operator	Keterangan
AND	Jika kedua ekspresi bernilai True, hasilnya akan True. Jika tidak, hasilnya False.
OR	Jika salah satu atau kedua ekspresi bernilai True, hasilnya bernilai True.
NOT	Jika ekspresi bernilai True, hasilnya False. Jika ekspresi bernilai False, hasilnya True.
XOR	Jika salah satu eksresi bernilai True, hasilnya True. Jika ekspresi bernilai True atau False,

Operator-operator tersebut juga memiliki prioritas jika digunakan secara bersamaan untuk suatu ekspresi yang kompleks. VB akan menguji operator matematis terlebih dahulu, kemudian operator perbandingan, dan operator logika pada urutan terakhir.

2.3.3.3 If then Else

Struktur ini disebut juga dengan *Branch Structure*, merupakan struktur percabangan dimana suatu ekspresi akan dikerjakan bila kondisinya terpenuhi. Tetapi, jika kondisinya tidak terpenuhi maka ekspresi yang lainnya akan dikerjakan.

Kondisi yang diperlukan oleh perintah ini adalah kondisi benar (`True`) atau kondisi salah (`False`). Salah satu cara untuk memperoleh kondisi adalah dengan menggunakan operator–operator perbandingan. Sedangkan untuk dapat menggabungkan beberapa kondisi sekaligus dalam sebuah percabangan, maka VB dilengkapi dengan operator logika.

Format dasar penulisan perintah ini adalah:

```
If (condition) Then
     (statement1)
Else
     (statement2)
End If
```

`Condition` adalah bagian yang menentukan pengambilan keputusan. Jika `Condition` bernilai `True` maka `statement1` akan dijalankan, sebaliknya bila `Condition` bernilai `False` maka `statement2` yang akan dieksekusi.

Perintah `Else` tidak harus mengikuti perintah `If`. Perintah `Else` digunakan bila ada `statement` dengan `condition` yang bernilai `False`. Contoh dari penggunaan struktur `If-Else` adalah sebagai berikut:

```
Dim X As Integer
X = 1000
If X > 100 Then
     Print "X lebih dari 100"
Else
     Print "X kurang dari atau sama dengan 100"
End If
```

Akan menghasilkan: `X lebih dari 100`

2.3.3.4 Select Case

Perintah ini juga digunakan dalam VB untuk percabangan dalam pemrograman. Percabangan dengan perintah ini sedikit berbeda dengan perintah `If`. Dengan perintah `Select Case`, *programmer* memperoleh keuntungan dengan semakin mudahnya membaca program dengan percabangan.

Namun perintah ini tidak dapat melakukan pemeriksaan terhadap banyak kondisi yang berbeda sekaligus karena perintah ini hanya memeriksa kondisi sebuah ekspresi.

2.3.3.5 For Next

Struktur ini digunakan untuk mengulang blok perintah dalam jumlah yang sudah ditentukan. Pada struktur ini, yang perlu dituliskan adalah nilai awal dan akhir variabel penghitung. Nilai variabel penghitung ini akan secara otomatis bertambah atau berkurang setiap kali suatu pengulangan dikerjakan. Format perintah ini adalah sebagai berikut:

```
For counter = awal To akhir Step pertambahan
      (Statement yang ingin diulang)
Next counter
```

`Variabel` adalah suatu variabel *integer* yang digunakan untuk melakukan proses pengulangan dari `awal` sampai `akhir`. `Awal` adalah nilai suatu variabel *integer* untuk menentukan harga awal suatu pengulangan. `Akhir` adalah nilai suatu variabel *integer* untuk menentukan harga akhir suatu pengulangan. `Pertambahan` adalah besarnya nilai perubahan dari nilai awal sampai nilai akhir, sedangkan `ekspresi` merupakan suatu blok perintah yang akan dikerjakan jika kondisi dari proses pengulangan memenuhi syarat.

2.3.3.6 Do Loop

Perintah ini juga dapat menbuat suatu perulangan dalam program. Namun perintah ini menggunakan suatu pemeriksaan terhadap suatu kondisi untuk menentukan suatu perulangan dihentikan.

Sehingga format umum penulisan perintah ini adalah sebagai berikut:

```
Do [While/Until] (condition)
(Statement yang ingin diulang)
Loop
```

atau

```
Do
(Statement yang ingin diulang)
Loop [While/Until] (Condition)
```

Perintah `while/until` dapat diletakkan di akhir perintah atau di awal perintah. Jika `while/until` diletakkan di awal, maka sebelum melakukan perulangan yang pertama, komputer akan melakukan pemeriksaan kondisi terlebih dahulu. Jika kondisi terpenuhi, maka perulangan dapat dilakukan. Kemudian dilakukan pemeriksaan lagi sebelum melakukan perulangan berikutnya. Dengan cara ini dimungkinkan tidak terjadi perulangan sama sekali karena kondisinya tidak terpenuhi pada pemeriksaan pertamanya.

Namun bila perintah `while/until` diletakkan di akhir perintah, maka perulangan akan dilakukan terlebih dahulu sebelum pemeriksaan dilakukan. Sehingga paling tidak akan terjadi sebuah perulangan. Perulangan berikutnya akan dilakukan bila kondisinya terpenuhi.

Perintah `while` dan `until` tidak dapat digunakan bersamaan. Jika menggunakan perintah `while`, maka suatu perulangan akan terus dilakukan selama kondisinya bernilai benar dan akan dihentikan saat kondisi bernilai salah. Sedangkan jika menggunakan perintah `until`, maka perulangan akan terus dilakukan selama kondisinya bernilai salah dan akan dihentikan saat kondisinya bernilai benar.

2.4 Pemrograman *Windows* API dengan VB

2.4.1 Pengantar *Windows* API

Windows API (*Application Programing Interface*) merupakan sekumpulan fungsi-fungsi eksternal yang terdapat dalam berkas-berkas perpustakaan (*library*) *Windows* yang dapat menangani semua operasi yang berhubungan dengan *Windows*, seperti pengaksesan media penyimpanan, grafik *Windows*, kotak dialog, penanganan berkas, mengakses sistem *registry*, dan sebagainya. Selain itu fungsi ini juga memastikan secara konsistem penggunaan semua sumber yang terdapat dalam *Windows*. Semua fungsi *Windows* API hampir terdapat dalam direktori sistem milik *Windows* (biasanya terdapat dalam direktori C:\Windows\System dan C:\Windows, bergantung pada pengaturan pertama instalasi *Windows*), dan paling banyak berekstensi .DLL.

DLL (*Dynamic Link Library*) adalah kode yang sudah dikompilasi dan dapat digunakan oleh program lain. DLL biasanya ditulis dengan bahasa C/C++, *Delphi*, atau bahasa lainnya yang mendukung sistem operasi *Windows*.

Berikut nama *library* milik *Windows* yang paling banyak digunakan dalam *Windows* API (Hadi, 2001).

Tabel 2.8 Deskripsi berkas-berkas DLL

Nama File DLL	Deskripsi File
Advapi32.dll	*Library* yang mendukung fungsi-fungsi keamanan dan rutin-rutin *registry.*
Comdlg32.dll	Standar kotak dialog *Windows.*
Gdi32.dll	Penanganan grafik *Windows.*
Kernel32.dll	Fungsi sistem operasi *Windows* 32-*bit*
Lz32.dll	Fungsi kompresi *file*
Mpr.dll	Fungsi *Intenet*
Netapi32.dll	Fungsi jaringan
Shell32.dll	*Library shell* 32-*bit*
User32.dll	Penanganan rutin *user interface*
Version.dll	Versi *Windows*
Winmm.dll	Fungsi-fungsi *multimedia Windows*
Winspool.drv	Fungsi-fungsi *printer spooler*

2.4.2 Penggunaan *Windows* API dalam VB

Untuk menggunakan fungsi-fungsi *Windows* API dalam VB, maka pertama kali yang harus dilakukan adalah mendeklarasikan fungsi-fungsi yang dibutuhkan di dalam suatu modul. Pendeklarasian fungsi *Windows* API ada dua cara, yaitu: pendeklarasian fungsi yang mengembalikan nilai dan pendeklarasian fungsi yang tidak mengembalikan nilai. Format pendeklarasian fungsi yang mengembalikan nilai adalah sebagai berikut:

```
Declare Function NamaFungsi Lib "NamaLibrary" [Alias _
"AliasFungsi"] ([[ByVal/ByRef] variabel [as type][, _
[ByVal/ByRef] variabel [as type]]...) As FunctionType
```

Keterangan:

- NamaFungsi : Nama fungsi yang ada dalam *library*. Penulisannya bersifat *case-sensitive*.

- NamaLibrary: Nama *library* yang menyimpan nama fungsi yang dideklarasikan. *Programmer* harus mengetahui penyimpanan letak *library*. Jika tidak berada dalam direktori sistem *Windows,* maka letak penyimpanan *library* harus ikut dideklarasikan.

- AliasFungsi : Nama alias dari fungsi yang akan dideklarasikan.

- Variabel : Nama parameter dari fungsi.

- Type : Tipe data dari parameter.

- FunctionType: Tipe data dari nilai balik fungsi.

Sedangkan untuk format pendeklarasian fungsi yang tidak membalikkan nilai adalah sebagai berikut:

```
Declare Sub NamaFungsi Lib "NamaLibrary" [Alias _
"AliasFungsi"] ([[ByVal/ByRef] variabel [as type][, _
[ByVal/ByRef] variabel [as type]]...)
```

Jika pemanggilan dilakukan dalam *form, module*, atau *class module*, sebelum perintah `declare` diberikan kata `Private` atau `Public` untuk mendefinisikan sifat fungsi tersebut. `Private` berarti fungsi tersebut hanya dapat dipanggil oleh fungsi lainnya yang terdapat dalam *form*, *module*, atau *class module* yang mendeklarasikannya, sedangkan `Public` berarti fungsi tersebut dapat

dipanggil oleh fungsi manapun dalam `project`. Contoh dari pendeklarasian fungsi API adalah sebagai berikut:

```
Public Declare Function GetFileSize Lib "kernel32.dll" _
(ByVal hfile As Long,lpFileSizeHigh As Long) As Long
```

Setelah fungsi API dideklarasikan, maka berikut adalah contoh dari pemanggilan fungsi API dalam *form* pada bagian *event* `load`.

```
Private Sub Form1_Load()
Dim x As Long
Dim hfile As Long
Dim highorder As Long
x = GetFileSize(hfile, highorder)
End Sub
```

2.4.3 Fungsi-fungsi *Windows* API

Beberapa fungsi API yang digunakan dalam program adalah fungsi penanganan berkas, fungsi manipulasi *registry*, dan fungsi grafik.

2.4.3.1 Fungsi Penanganan Berkas

Beberapa fungsi penanganan berkas yang digunakan dalam program adalah:

1) Membaca berkas

   Fungsi API yang digunakan untuk membaca data dari berkas yang telah dibuka adalah `ReadFile`. Fungsi ini akan meletakkan data dalam variabel yang dilewatkan sebagai `lpBuffer` dan meletakkan jumlah *byte* dari data langsung ke dalam variabel melalui `lpNumberOfBytesRead`. Sebelumnya, nama berkas harus dibuka terlebih dahulu dengan akses `READ`. Fungsi ini akan membaca dari posisi yang dispesifikasikan oleh *pointer* berkas dan

mengatur *pointer* berkas ke posisi yang ditunjuk setelah semua data dibaca jika berkas bertipe sinkron (*non-overlapped*). Jika berkas bertipe asinkron, pembacaan berkas dimulai dari point yang dispesifikasikan oleh `lpOverlapped`. Fungsi ini akan mengembalikan nilai 0 jika terjadi kesalahan, dan nilai selain itu jika berhasil. Format pendeklarasiannya adalah sebagai berikut:

```
Declare Function ReadFile Lib "kernel32" (ByVal hFile As _
Long, lpBuffer As Any, ByVal nNumberOfBytesToRead As Long, _
lpNumberOfBytesRead As Long, lpOverlapped As OVERLAPPED)As _
Long
```

Keterangan:

Tabel 2.9 Parameter `ReadFile`

`hFile`	*Handle* untuk membaca berkas. Berkas harus mempunyai akses `READ`.
`lpBuffer`	Variabel, *array*, atau tipe data yang menerima data dari berkas.
`nNumberOfBytesToRead`	Nomor *byte* dari data yang dibaca dari berkas dan menyimpannya ke dalam `lpBuffer` (contohnya ukuran dari `lpBuffer`).
`lpNumberOfBytesToRead`	Menerima nomor *byte* dari data secara aktual yang dibaca dari berkas. Jika bernilai 0, maka berarti akhir dari berkas telah tercapai.

2) Membuat berkas

Fungsi API yang digunakan untuk membuat atau membuka berkas pada media penyimpanan untuk akses berikutnya adalah `CreateFile`. Format pendeklarasiannya adalah sebagai berikut:

```
Declare Function CreateFile Lib "kernel32" Alias _
"CreateFileA" (ByVal lpFileName As String, ByVal _
dwDesiredAccess As Long, ByVal dwShareMode As Long, _
lpSecurityAttributes As SECURITY_ATTRIBUTES, ByVal _
dwCreationDisposition As Long, ByVal dwFlagsAndAttributes _
As Long, ByVal hTemplateFile As Long) As Long
```

Keterangan:

Tabel 2.10 Parameter `CreateFile`

`lpFileName`	Nama berkas untuk membuka atau membuat berkas.
`dwDesiredAccess`	Nilai 0 atau salah satu *flag* berikut: • `GENERIC_READ`: untuk membaca data dari berkas. • `GENERIC_WRITE`: untuk menulis data ke berkas.
`dwShareMode`	Nilai 0 atau nilai konstanta *flag* berikut: • `FILE_SHARE_READ`: mode *share* untuk membaca. • `FILE_SHARE_WRITE`: mode *share* untuk menulis.
`lpSecurityAttributes`	Atribut untuk membuat atau membuka berkas. Untuk penggunaan di *Windows* 95 harus menspesifikasikan dengan nilai 0.
`dwCreationDisposition`	*Flag* untuk membuat dan membuka berkas, dan bergantung pada ada atau tidaknya berkas. • `CREATE_NEW`: membuat berkas baru. Fungsi akan gagal, jika berkas tidak ada. • `CREATE_ALWAYS`: membuat berkas baru untuk menimpa berkas lama, jika berkas tersebut tidak ada. • `OPEN_EXISTING`: membuka berkas. Fungsi akan gagal, jika berkas tidak ada. • `OPEN_ALWAYS`: membuka berkas. Jika berkas tidak ada, maka akan dibuatkan.
`dwFlagAndAtrributes`	Kombinasi *flag* berikut menspesifikasikan antara atribut berkas dan akses berkas: `FILE_ATRIBUTE_ARCHIVE`: berkas arsip. `FILE_ATRIBUTE_HIDDEN`: berkas yang disembunyikan. `FILE_ATRIBUTE_NORMAL`: berkas normal (atributnya tidak ada kombinasi dengan atribut lainnya). `FILE_ATRIBUTE_READONLY`: berkas *READONLY*. `FILE_ATRIBUTE_SYSTEM`: berkas sistem.
`hTemplateFile`	*Handle* dari berkas yang terbuka untuk mengkopi atribut yang dispesifikasikan, atau nilai 0 jika tidak ingin mengkopi atribut file.

3) Menulis berkas

Fungsi API yang digunakan untuk menulis data ke berkas yang telah dibuka adalah `WriteFile`. Fungsi ini juga menyimpan nomor *byte* data secara actual yang ditulis ke dalam variabel melalui `lpNumberOfBytesWriten`. Berkas harus dibuka dengan akses `WRITE`. Fungsi ini akan mengembalikan nilai 0 jika terjadi kesalahan, dan nilai 1 jika berhasil. Format pendeklarasian fungsi ini adalah sebagai berikut:

```
Declare Function WriteFile Lib "kernel32" (ByVal hFile As _
Long, lpBuffer As Any, ByVal nNumberOfBytesToWrite As _
Long, lpNumberOfBytesWritten As Long, lpOverlapped As _
OVERLAPPED) As Long
```

Keterangan:

Tabel 2.11 Parameter `WriteFile`

`hFile`	*Handle* berkas untuk menulis berkas.
`lpBuffer`	Data yang akan ditulis ke berkas.
`nNumberOfBytesToWrite`	Nomor *bytes* data untuk ditulis ke berkas.
`lpNumberOfBytesWritten`	Menerima nomor *byte* data actual yang akan ditulis ke berkas.
`lpOverlapped`	Spesifikasi di mana memulai penulisan jika berkas bertipe simultan (*overlapped*).

4) Mengkopi berkas

Fungsi API yang digunakan untuk mengkopi berkas dari satu lokasi ke lokasi lain adalah `CopyFile`. Format pendeklarasiannya adalah sebagai berikut:

```
Declare Function CopyFile Lib "kernel32" Alias "CopyFileA" _
(ByVal lpExistingFileName As String, ByVal lpNewFileName _
As String, ByVal bFailIfExists As Long) As Long
```

Keterangan:

Tabel 2.12 Parameter `CopyFile`

`lpExistingFile`	Nama berkas yang akan dikopikan.
`lpNewFileName`	Nama berkas target.
`bFailIfExisting`	Modus pengkopian. Jika nilaiinya diisi 1, maka akan diaktifkan pengecekan berkas terlebih dahulu sebelum dikopikan, dan nilai 0 untuk mengabaikan modus pengecekan keberadaan berkas.

Pengkopian berkas bergantung pada nilai parameter `bFailIfExist`. Jika nilainya 1, berarti pengkopian dalam modus tidak ditimpa (*overwrite*), dan jika nilainya 0, berarti berkas tujuan akan ditimpa oleh berkas sumber.

Parameter `bFailIfExist` dapat juga digunakan untuk mengecek status berkas target pengkopian, apakah berkas tersebut ada atau tidak. Fungsi ini akan mengembalikan nilai 1 jika berhasil, sebaliknya akan mengembalikan nilai 0 jika terjadi kesalahan.

5) Menghapus berkas secara langsung

Fungsi API yang digunakan untuk manghapus berkas tanpa disimpan dalam tempat penampungan sementara (*Recycle Bin*) adalah `DeleteFile`. Fungsi ini tidak memberikan konfirmasi terlebih dahulu, apakah berkas target akan dihapus atau tidak. Fungsi ini akan mengembalikan nilai 1, jika penghapusan berhasil, dan nilai 0, jika terjadi kesalahan. Format pendeklarasiannya adalah sebagai berikut:

```
Declare Function DeleteFile Lib "kernel32" Alias _
"DeleteFileA" (ByVal lpFileName As String) As Long
```

Keterangan:

- `lpFileName` : Nama berkas yang akan dihapus.

6) Mengambil ukuran berkas

Fungsi API yang digunakan untuk mengambil ukuran berkas yang dispesifikasikan oleh *handle* berkas adalah `GetFileSize`. Format pendeklarasiannya adalah sebagai berikut:

```
Declare Function GetFileSize Lib "kernel32" (ByVal hFile _
As Long, lpFileSizeHigh As Long) As Long
```

Keterangan:

`hFile` : *Handle* berkas yang telah dibuka.

`lpFileSizeHigh` : Variabel yang menampung setengah dari ukuran berkas.

Ukuran berkas dalam ukuran nilai 64 bit yang dibagi dalam dua bagian, yaitu 32 bit bagian atas dan 32 bit bagian bawah. Nilai pertama dimasukkan ke dalam `lpFileSizeHigh` dann nilai kedua dimasukkan ke dalam nilai yang dikembalikan oleh fungsi. Ada dua cara untuk mendapatkan ukuran berkas, yaitu: mengambil dalam nilai biner atau heksadesimal dengan menggunakan rumusan seperti berikut:

```
FileSize = lpFileSizeHigh * 2 ^ 32 + nReturnValue
```

Fungsi ini akan mengembalikan nilai -1 jika terjadi kesalahan, dan mengembalikan nilai setengah dari ukuran berkas jika berhasil. Cara lainnya adalah dengan menggunakan perintah `CreateFile` untuk mendapatkan nilai *handle* berkas dan mengisi nilai `lpFileSizeHigh` sama dengan 0. Cara ini akan mendapatkan ukuran sesungguhnya dari berkas. Tetapi jika nilai `lpFileSizeHigh` berubah, maka terjadi kesalahan dalam ukuran berkas yang melampaui kapasitas memori VB.

7) Mengatur *pointer* berkas

Fungsi API yang digunakan untuk memindahkan posisi *pointer* berkas ke dalam berkas yang dibuka. *Pointer* berkas diidentifikasikan dalam berkas tipe simultan. Berkas harus dibuka dengan mode level baca atau tulis atau kedua-duanya.

Format pendeklarasiannya adalah sebagai berikut:

```
Declare Function SetFilePointer Lib "kernel32" (ByVal _
hFile As Long, ByVal lDistanceToMove As Long, _
lpDistanceToMoveHigh As Long, ByVal dwMoveMethod As Long) _
As Long
```

Keterangan:

Tabel 2.13 Parameter `SetFilePointer`

`hFile`	*Handle* berkas dari berkas yang dibuka.
`lDistanceToMove`	Setengah bagian dari nomor *byte* yang akan dipindahkan ke *pointer* berkas.
`lpDistanceToMoveHigh`	Variabel yang berisi setengah bagian atas dari nomor *byte* yang akan dipindah ke dalam *pointer* berkas.
`dwMoveMethod`	*Flag* yang digunakan untuk menentukan akses berkas. • `FILE_BEGIN`: awal dari berkas. • `FILE_CURRENT`: posisi aktif dari *pointer* berkas. • `FILE_END`: akhir dari berkas.

8) Mengatur akhir dari berkas

Fungsi API yang digunakan untuk menempatkan *pointer* ke akhir berkas adalah `SetEndOfFile`. Format pendeklarasiannya adalah sebagai berikut:

```
Declare Function SetEndOfFile Lib "kernel32" (ByVal hFile _
As Long) As Long
```

Keterangan:

- `hFile` : *Handle* berkas yang telah dibuka.

9) Mengatur atribut berkas

Fungsi API yang digunakan untuk mengatur atribut dari suatu berkas adalah `SetFileAttribute`. Format pendeklarasiannya adalah sebagai berikut:

```
Declare Function SetFileAttributes Lib "kernel32" Alias _
"SetFileAttributesA" (ByVal lpFileName As String, ByVal _
dwFileAttributes As Long) As Long
```

Keterangan:

- `lpFileName` : Nama berkas yang akan diset.
- `dwFileAttributes` : Spesifikasi atribut berkas yang akan ditambahkan.

Atribut berkas dalam *Windows* terdiri atas atribut *Normal*, *Hidden*, *Read Only*, dan *System*. Fungsi ini akan mengembalikan nilai 0 jika terjadi kesalahan, dan nilai lainnya jika berhasil.

2.4.3.2 Fungsi Manipulasi *Registry*

Beberapa fungsi API yang menangani manipulasi *registry* yang digunakan dalam program adalah:

1) Membuat kunci *registry*

Fungsi API yang digunakan untuk membuat kunci baru dalam *registry* adalah `RegCreateKeyEx`. Fungsi ini juga dapat digunakan untuk membuka kunci *registry*, jika ternyata kunci baru yang akan dibuat telah ada dalam basis data *registry Windows*. Fungsi ini akan mengembalikan nilai 0 jika berhasil, dan nilai lainnya jika terjadi kesalahan. Format pendekalarasiannya adalah sebagai berikut:

```
Declare Function RegCreateKeyEx Lib "advapi32.dll" Alias _
"RegCreateKeyExA" (ByVal hKey As Long, ByVal lpSubKey As _
String, ByVal Reserved As Long, ByVal lpClass As String, _
ByVal dwOptions As Long, ByVal samDesired As Long, _
lpSecurityAttributes As SECURITY_ATTRIBUTES, phkResult As _
Long, lpdwDisposition As Long) As Long
```

Keterangan:

Tabel 2.14 Parameter `RegCreateKeyEx`

`hKeyReg`	Salah satu *handle* untuk membuka kunci *registry* di bawah kunci berikut: • `HKEY_CLASSES_ROOT`: kunci yang paling atas dari informasi berkas-berkas yang ada dalam *Windows*. • `HKEY_CURRENT_USER`: kunci yang paling atas dari informasi pengguna komputer yang aktif. • `HKEY_LOCAL_MACHINE`: kunci yang paling atas dari informasi perangkat-keras, perangkat-lunak, dan komunikasi yang digunakan dalam komputer pengguna. • `HKEY_USERS`: kunci paling atas dari informasi sembarang pengguna. • `HKEY_CURRENT_CONFIG`: kunci paling atas dari informasi konfigurasi computer. • `HKEY_DYN_DATA`: kunci paling atas dari informasi data dinamis.
`lpSubKey`	Nama kunci yang akan dibuat.
`reversed`	Nilai tetapan *Windows* yang diset menjadi 0.
`lpClass`	Nama kelas dari objek tipe kunci.
`dwOption`	Jika kunci yang dibuat akan disimpan dalam *registry*, maka nilainya diset menjadi 0. Sebaliknya, nilainya diset menjadi 1. Kunci yang tidak disimpan dalam *registry* akan dihapus sewaktu pengguna keluar dari *Windows*.
`samDesired`	Satu atau lebih *flag* yang menspesifikasikan nilai akses baca atau tulis. • `KEY_READ`: mengizinkan untuk akses pembacaan. • `KEY_SET_VALUE`: mengizinkan data untuk menset data subkunci. • `KEY_WRITE`: mengizinkan untuk mengakses penulisan.
`lpSecurityAttributes`	Untuk *Windows* NT, level keamanan akan disertakan pada kunci. Untuk *Windows* versi lainnya, nilainya diset menjadi 0.
`phkResult`	Variabel yang menerima *handle* dari status membuat atau membuka kunci *registry*.
`lpdwDisposition`	Variabel yang menerima nilai 1 jika kunci berhasil dibuat, dan nilai 2 jika kunci yang akan dibuat telah dibuka.

2) Mengubah nilai *registry*

Fungsi API yang digunakan untuk menulis nilai pada kunci *registry* adalah `RegSetValueEx`. Format pendeklarasiannya adalah sebagai berikut:

```
Declare Function RegSetValueEx Lib "advapi32.dll" Alias _
"RegSetValueExA" (ByVal hKey As Long, ByVal lpValueName As _
String, ByVal Reserved As Long, ByVal dwType As Long, _
lpData As Any, ByVal cbData As Long) As Long
```

Keterangan:

Tabel 2.15 Parameter `RegSetValueEx`

`lKey`	Nama kunci yang akan dibuka yang terdapat nama nilai.
`lpValueName`	Nama nilai yang akan diset.
`Reversed`	Digunakan oleh fungsi. Nilainya ditetapkan menjadi 0.
`dwType`	*Flag* dari operasi penulisan: `REG_BINARY`: data bukan teks yang terdiri atas serangkaian bit. `REG_DWORD`: data *integer* 32 bit. `REG_SZ`: data string dengan `vbNullChar` ("\0").

BAB III

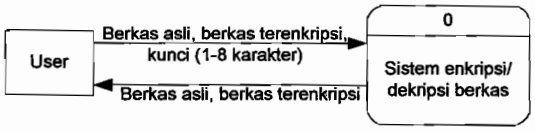
ANALISIS dan PERANCANGAN SISTEM

3.1 Analisis Sistem

3.1.1 DFD (*Data Flow Diagram*)

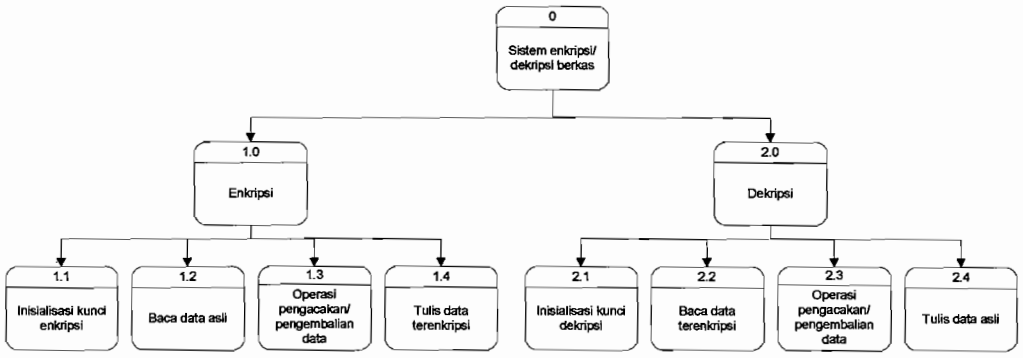
Jika dilihat dari cara kerja IDEA yang telah dijelaskan sebelumnya dalam landasan teori, maka sistem enkripsi dan dekripsi ini dapat dimodelkan dengan menggunakan DFD sebagai berikut:

1) *Context diagram*



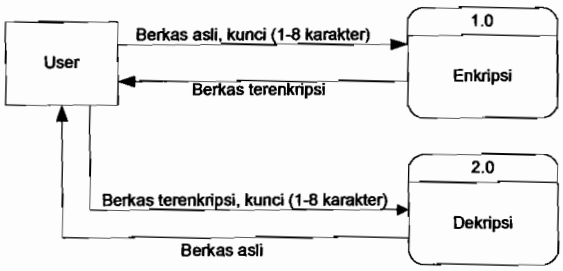
Gambar 3.1 *Context diagram*

2) *Diagram zero* (*level* 1)



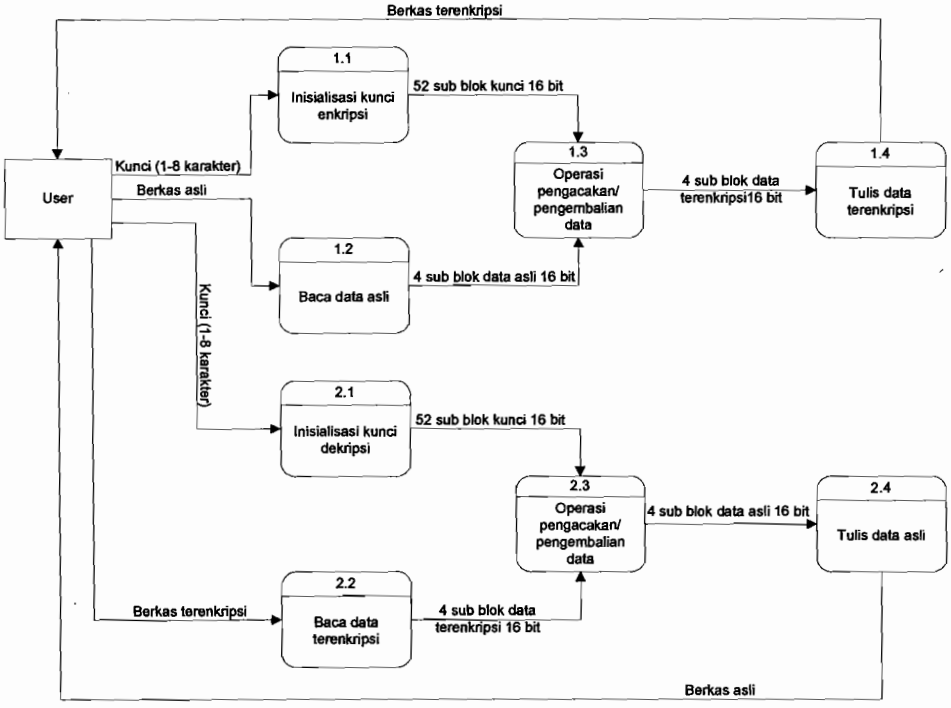
Gambar 3.2 *Diagram zero* (*level* 1)

3) *Diagram zero* (*level* 2)



Gambar 3.3 *Diagram zero* (*level* 2)

4) *Diagram zero* (*level* 3)

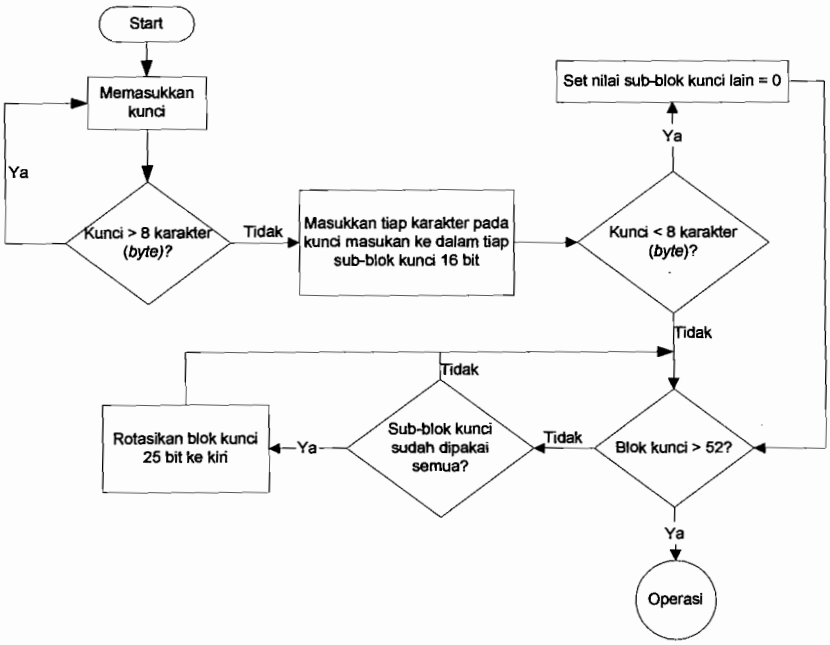


Gambar 3.4 *Diagram zero* (*level* 3)

3.1.2 Diagram Alir

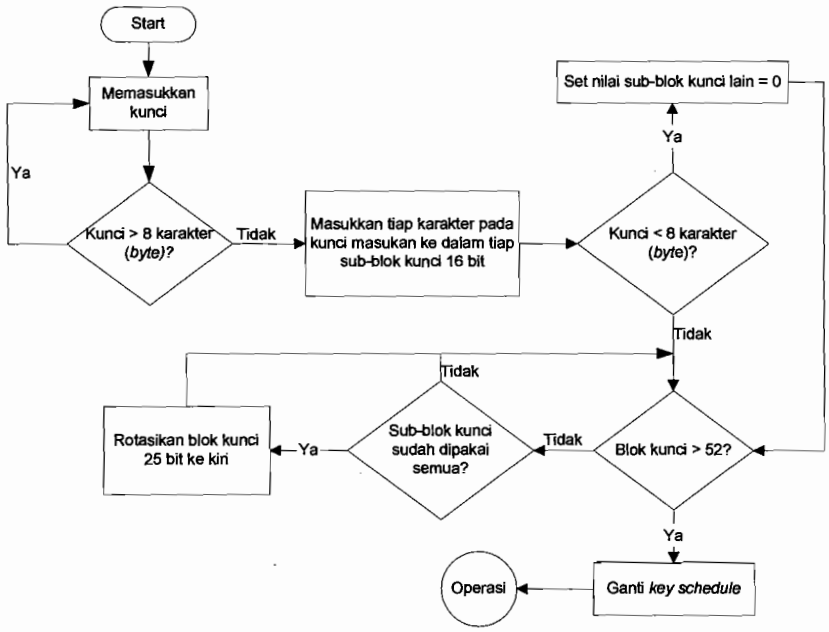
Tiap proses yang ada dalam sistem dapat dijelaskan dengan menggunakan diagram alir, kecuali bagian proses yang berupa operasi aljabar tidak digambarkan. Berikut beberapa diagram alir dari proses yang ada dalam DFD.

1) Inisialisasi kunci enkripsi



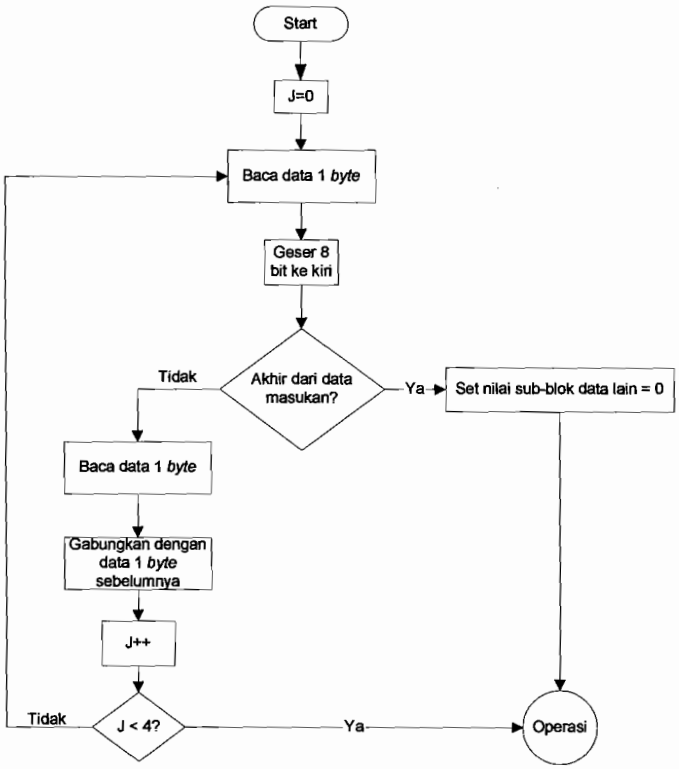
Gambar 3.5 Diagram alir inisialisasi kunci enkripsi

2) Inisialisasi kunci dekripsi



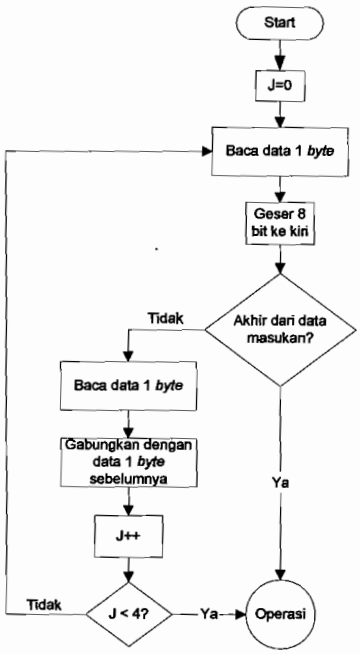
Gambar 3.6 Inisialisasi kunci dekripsi

3) Baca data asli



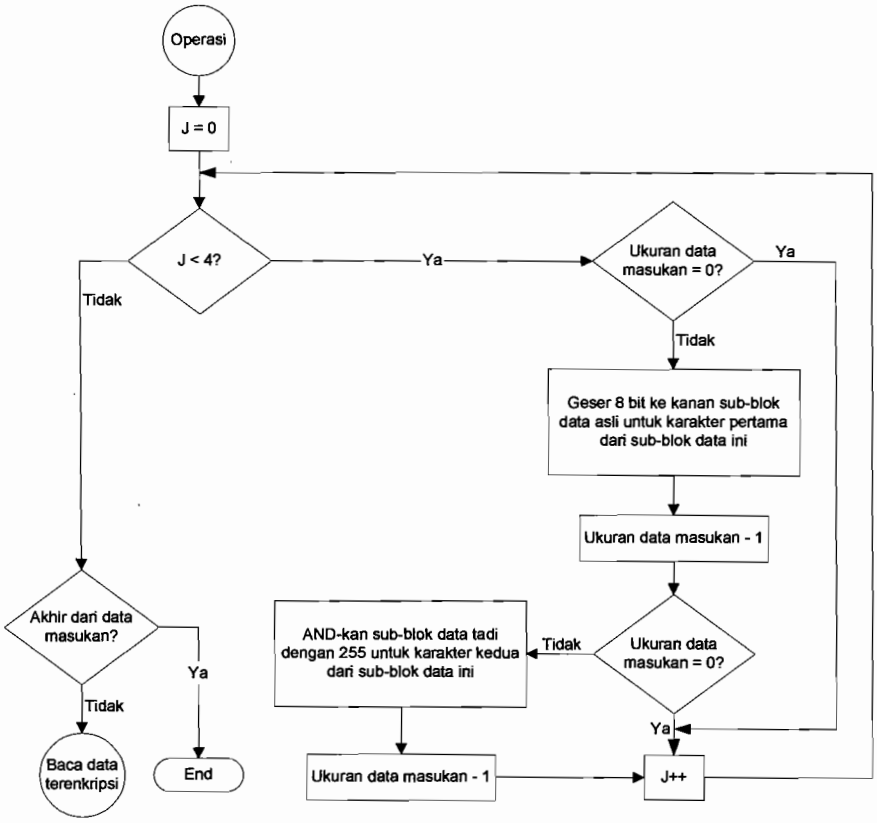
Gambar 3.7 Diagram alir baca data

4) Baca data terenkripsi



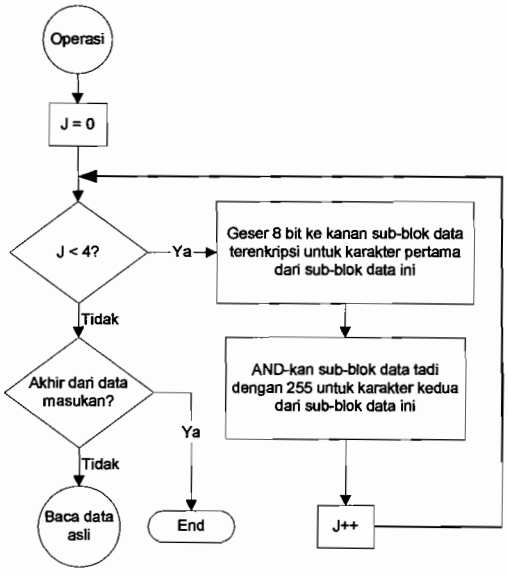
Gambar 3.8 Diagram alir baca data terenkripsi

5) Tulis data asli



Gambar 3.9 Diagram alir tulis data asli

6) Tulis data terenkripsi



Gambar 3.10 Diagram alir tulis data terenkripsi

3.2 Perancangan Sistem

3.2.1 Struktur *Registry* pada Sistem

Sistem merupakan program aplikasi enkripsi dan dekripsi dengan algoritma IDEA yang memiliki fasilitas tambahan, yaitu: menu pilihan enkripsi dan dekripsi pada *Context menu*. Penambahan menu dalam *Context menu* dapat dilakukan dengan melakukan manipulasi pada *registry Windows*. Manipulasi yang dilakukan adalah menambahkan sub kunci pada kunci utama *registry Windows* yang berhubungan dengan penanganan berkas, yaitu: HKEY_CLASSES_ROOT.

Ada empat sub kunci yang dibutuhkan sistem, yaitu:

1) Sub kunci untuk membuka *form* enkripsi dan menampilkannya pada *context menu* serta berlaku untuk semua jenis berkas.
   - Nama : *\shell\Encryption\command
   - Nilai : PathExe & " /e %1"
2) Sub kunci untuk membuka *form* dekripsi dan menampilkannya pada *context menu* serta berlaku untuk berkas yang telah terenkripsi saja.
   - Nama : *\shell\Decryption\command
   - Nilai : PathExe & " /d %1"
3) Sub kunci untuk penanganan berkas yang telah terenkripsi
   - Nama : .enc
   - Nilai : "IDEADec"

4) Sub kunci yang merupakan penghubung dari sub kunci nomor 3) yang akan memberikan *icon* khusus untuk berkas yang telah terenkripsi.

- Nama : IDEAdec\DefaultIcon
- Nilai : PathExe & ",0"

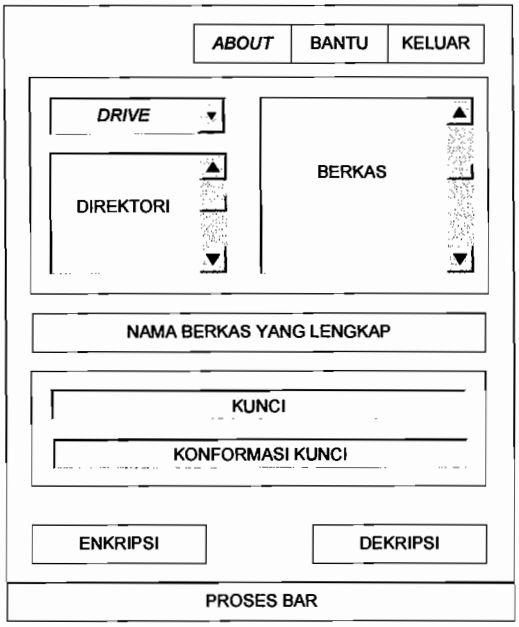
*) PathExe : direktori lengkap dari berkas yang aktif.

3.2.2 Perancangan *User Interface*

Ada lima *form* yang dibuat untuk sistem, yaitu: *form* utama, *form* enkripsi, *form* dekripsi, *form* bantu, dan *form about*.

1) *Form* Utama

Form ini akan terbuka saat program aplikasi ini dijalankan. Segala operasi yang berlaku pada sistem terdapat di dalamnya. Berikut adalah rancangan untuk tampilan *form* utama:



Gambar 3.11 Rancangan form utama

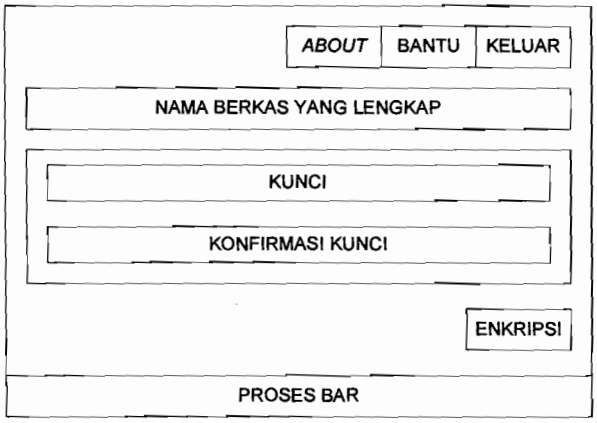
Keterangan Gambar 3.7:

- ABOUT : tombol yang digunakan untuk membuka *form about*.
- BANTU : tombol yang digunakan untuk membuka *form* bantu.
- KELUAR : tombol yang digunakan untuk keluar.
- DRIVE : objek *drive list box* yang berisi *drive-drive* pada komputer pengguna dan merupakan masukan untuk hasil seleksi *drive* yang menjadi tempat penyimpanan berkas yang akan diproses.
- DIREKTORI : objek *dir list box* yang berisi direktori-direktori dalam *drive* terpilih dan merupakan masukan untuk hasil seleksi direktori dari berkas yang akan diproses.
- BERKAS : objek *file list box* yang berisi daftar berkas-berkas yang berada dalam satu direktori dengan berkas yang akan diproses.
- NAMA BERKAS YANG LENGKAP : label dari nama lengkap berkas yang akan diproses. Label ini akan diisi setelah pemakai melakukan *double-click* ke berkas terpilih dalam daftar berkas yang tersedia.
- KUNCI : masukan untuk kunci yang akan diproses.
- KONFIRMASI KUNCI : masukan untuk memastikan kunci yang dimasukkan sama.
- ENKRIPSI : tombol untuk melakukan proses enkripsi terhadap berkas terpilih yang bukan merupakan berkas terenkripsi dan kunci masukan.
- DEKRIPSI : tombol untuk melakukan proses enkripsi terhadap berkas terpilih yang telah terenkripsi dan kunci masukan.

- PROSES BAR : bar yang digunakan untuk menyatakan status dari proses yang berjalan.

2) *Form* Enkripsi

Form ini akan terbuka saat pengguna memilih menu enkripsi pada *context menu* dan berkas yang terpilih bukan berekstensi .enc. Berikut adalah rancangan untuk tampilan *form* enkripsi:

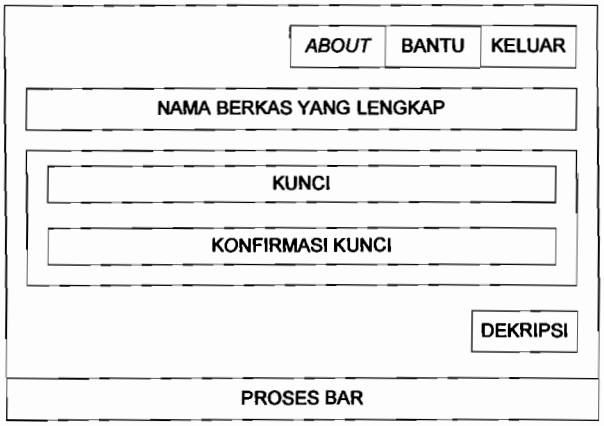


Gambar 3.12 Rancangan *form* enkripsi

Keterangan Gambar 3.8 sama dengan keterangan Gambar 3.7, namun label NAMA BERKAS YANG LENGKAP akan terisi sendiri sesuai dengan nama lengkap berkas yang aktif.

3) *Form* Dekripsi

Form ini akan terbuka saat pengguna memilih menu dekripsi pada *context menu* dan berkas yang terpilih berekstensi .enc. Berikut adalah rancangan untuk tampilan *form* dekripsi:

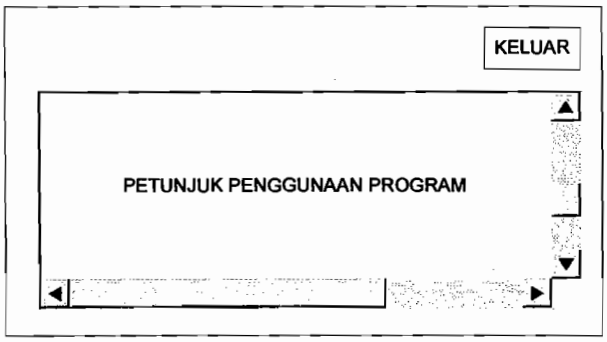


Gambar 3.13 Rancangan *form* dekripsi

Keterangan Gambar 3.9 sama dengan keterangan Gambar 3.7, namun label NAMA BERKAS YANG LENGKAP akan terisi sendiri sesuai dengan nama lengkap berkas yang aktif.

4) *Form* Bantu

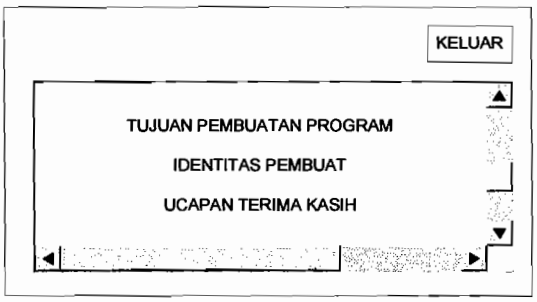
Form ini akan terbuka saat pengguna meng-*click* tombol BANTU. *Form* ini berisi petunjuk penggunaan program aplikasi ini. Berikut adalah rancangan untuk tampilan *form* bantu:



Gambar 3.14 Rancangan *form* bantu

5) *Form About*

Form ini akan terbuka saat pengguna meng-*click* tombol ABOUT. *Form* ini berisi tujuan pembuatan program, identitas pembuat, dan ucapan terima kasih. Berikut adalah rancangan untuk tampilan *form about*:



Gambar 3.15 Rancangan *form about*

3.2.3 Konfigurasi Perangkat-Keras

Rancangan kebutuhan perangkat-keras minimum untuk dapat menjalankan program aplikasi ini adalah:

- Prosesor : Kompatibel IBM dengan teknologi MMX
- Memori : 16 MB
- Kartu Grafik : Mempunyai memori 4MB
- *Harddisk* : 15 MB

3.2.4 Konfigurasi Perangkat-Lunak

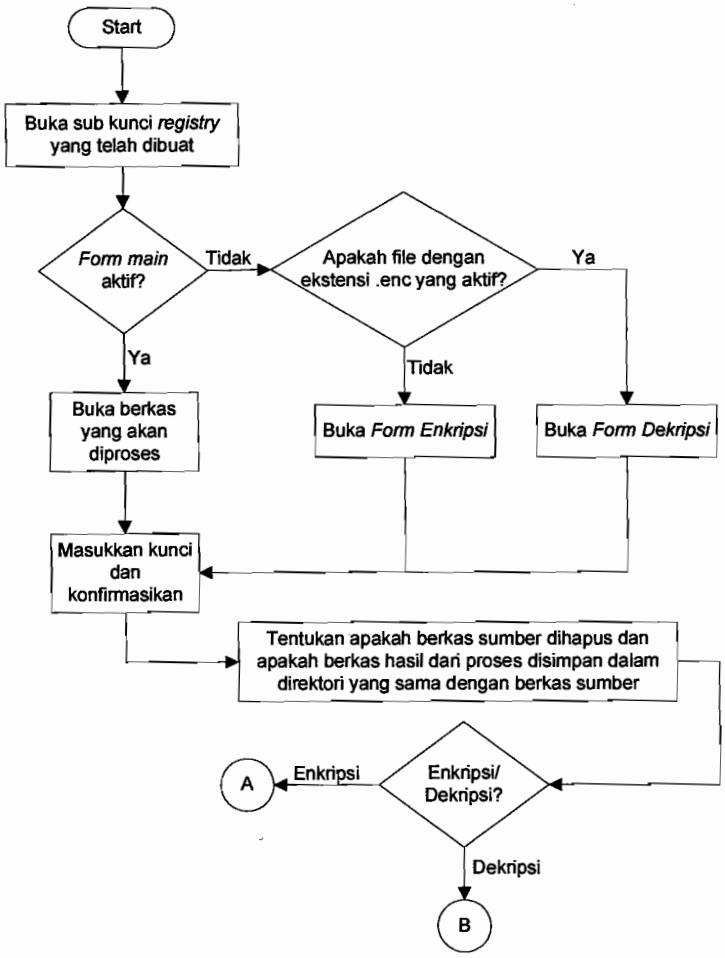
Rancangan kebutuhan perangkat-lunak minimum untuk dapat menjalankan program aplikasi ini adalah:

- Sistem Operasi : *Microsoft Windows* 95/98/ME/2000

BAB IV

IMPLEMENTASI

Ada tiga macam *form* penting yang akan ditampilkan dalam sistem ini, yaitu: *form* utama, *form* enkripsi, dan *form* dekripsi. Ketiga *form* ini akan melalui beberapa proses yang dapat digambarkan seperti diagram alir berikut ini:



Gambar 4.1 Diagram alir sistem

Ada dua fungsi utama yang dikerjakan saat sistem dijalankan di awal program, yaitu:

1) Fungsi untuk membuat sub kunci *registry* yang dipakai untuk menampilkan menu pilihan proses enkripsi dan dekripsi dalam *context menu Windows*. Berikut adalah baris-baris perintah yang membentuk fungsi ini:

```
Private Sub SetRegRightClick(PathExe As String)
Dim secattrPlain As SECURITY_ATTRIBUTES, secattrCipher As
SECURITY_ATTRIBUTES
Dim secattrCipherLnk As SECURITY_ATTRIBUTES, secattrCipherIco As
SECURITY_ATTRIBUTES
Dim XP, XC, XCI, XCL, NewOpenP As Long
Dim NewOpenC As Long, NewOpenCL As Long, NewOpenCI As Long
Dim SubKeyPlain As String
Dim SubKeyCipherIco As String, SubKeyCipher As String,
SubKeyCipherLnk As String
Dim DataPlain As String
Dim DataCipher As String, DataCipherLnk As String, DataCipherIco
As String
Dim hregkeyP As Long, hregkeyC As Long
Dim hregkeyCL As Long, hregkeyCI As Long

secattrCipher.lpSecurityDescriptor = 0
secattrCipher.bInheritHandle = True
secattrCipher.nLength = Len(secattrCipher)

secattrCipherLnk.lpSecurityDescriptor = 0
secattrCipherLnk.bInheritHandle = True
secattrCipherLnk.nLength = Len(secattrCipherLnk)

secattrCipherIco.lpSecurityDescriptor = 0
secattrCipherIco.bInheritHandle = True
secattrCipherIco.nLength = Len(secattrCipherIco)

secattrPlain.lpSecurityDescriptor = 0
secattrPlain.bInheritHandle = True
secattrPlain.nLength = Len(secattrPlain)

SubKeyPlain = "*\shell\Encryption\command"
SubKeyCipher = "*\shell\Decryption\command"
SubKeyCipherLnk = ".enc"
SubKeyCipherIco = "IDEAdec\DefaultIcon"

DataPlain = PathExe & " /e %1"
DataCipherLnk = "IDEAdec"
DataCipherIco = PathExe & ",0"
DataCipher = PathExe & " /d %1"
```

```
XP = RegCreateKeyEx(HKEY_CLASSES_ROOT, SubKeyPlain, 0, "", 0, _
KEY_WRITE, secattrPlain, hregkeyP, NewOpenP)

If XP <> 0 Then
    MsgBox "Failed to create registry!", vbExclamation, "ERROR!!!"
    RegCloseKey hregkeyP
    End
Else
    XP = RegSetValueExString(hregkeyP, "", 0, REG_SZ, DataPlain, _
Len(DataPlain))
    If XP <> 0 Then
        MsgBox "Failed to create registry!", vbExclamation, _
"ERROR!!!"
        RegCloseKey hregkeyP
        End
    End If
End If

XC = RegCreateKeyEx(HKEY_CLASSES_ROOT, SubKeyCipher, 0, "", 0, _
KEY_WRITE, secattrCipher, hregkeyC, NewOpenC)
If XC <> 0 Then
    MsgBox "Failed to create registry!", vbExclamation, "ERROR!!!"
    RegCloseKey hregkeyC
    End
Else
    XC = RegSetValueExString(hregkeyC, "", 0, REG_SZ, DataCipher,_
Len(DataCipher))
    If XC <> 0 Then
        MsgBox "Failed to create registry!", vbExclamation, _
"ERROR!!!"
        RegCloseKey hregkeyC
        End
    End If
End If

XCI = RegCreateKeyEx(HKEY_CLASSES_ROOT, SubKeyCipherIco, 0, "", _
0, KEY_WRITE, secattrCipherIco, hregkeyCI, NewOpenCI)
If XCI <> 0 Then
    MsgBox "Failed to create registry!", vbExclamation, "ERROR!!!"
    RegCloseKey hregkeyCI
    End
Else
    XCI = RegSetValueExString(hregkeyCI, "", 0, REG_SZ,
DataCipherIco, Len(DataCipherIco))
    If XCI <> 0 Then
        MsgBox "Failed to create registry!", vbExclamation, _
"ERROR!!!"
        RegCloseKey hregkeyCI
        End
    End If
End If

XCL = RegCreateKeyEx(HKEY_CLASSES_ROOT, SubKeyCipherLnk, 0, "", _
0, KEY_WRITE, secattrCipherLnk, hregkeyCL, NewOpenCL)
If XCL <> 0 Then
    MsgBox "Failed to create registry!", vbExclamation, "ERROR!!!"
```

```
    RegCloseKey hregkeyCL
    End

Else
    XCL = RegSetValueExString(hregkeyCL, "", 0, REG_SZ, _
DataCipherLnk, Len(DataCipherLnk))
    If XCL <> 0 Then
        MsgBox "Failed to create registry!", vbExclamation, _
"ERROR!!!"
        RegCloseKey hregkeyCL
        End
    End If
End If

RegCloseKey hregkeyP
RegCloseKey hregkeyC
RegCloseKey hregkeyCI
RegCloseKey hregkeyCL
End Sub
```

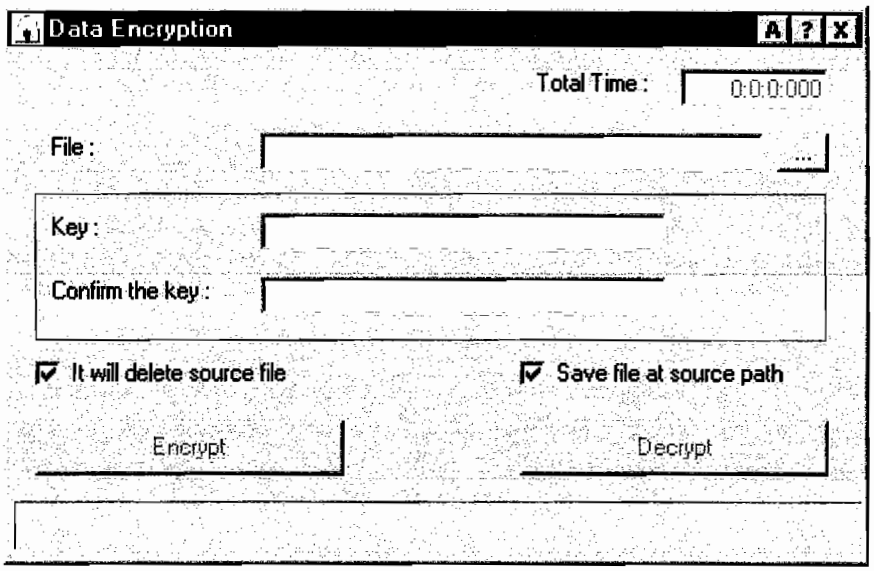
2) Fungsi untuk mengecek *form* mana yang akan diaktifkan dari ketiga *form* tersebut. Berikut adalah baris-baris perintah yang membentuk fungsi ini:

```
Dim temp
Dim cekEnc As String
Dim cekErrEnc As String
temp = Command()
cekEnc = Right(temp, 3)
cekErrEnc = Left(temp, 2)
If temp = "" Then
    Frame1.Enabled = False
    cmdBrws.BackColor = &H8000000F
    cmdEnkrip.BackColor = &H8000000F
    cmdEnkrip.Enabled = False
    cmdDekrip.BackColor = &H8000000F
    cmdDekrip.Enabled = False
    ProgressBar1.Value = 0
Else
    If UCase(cekEnc) = "ENC" Then
        Me.Hide
        If cekErrEnc = "/d" Then
            Decrypt.Show
            Decrypt.lblPath.Caption = Mid(temp, 4, Len(temp) - 3)
        Else
            MsgBox "Failed to Encrypt!" & vbCr & _
"Your file had been encrypted!", vbCritical, "ATTENTION!!!"
            End
        End If
    Else
        Me.Hide
        If cekErrEnc = "/e" Then
            Encrypt.Show
            Encrypt.lblPath.Caption = Mid(temp, 4, Len(temp) - 3)
```

```
        Else
            MsgBox "Failed to Decrypt!" & vbCr & _
"Your file has not been encrypted yet!", vbCritical, _
"ATTENTION!!!"
            End
        End If
    End If
End If
```

4.1 *Form* Utama

Form ini akan ditampilkan sewaktu pengguna mengaktifkan program ini langsung dari berkas eksekusinya. Disinilah semua proses berada. Berikut adalah tampilan implementasi *form* utama:



Gambar 4.2 Tampilan implementasi *form* utama

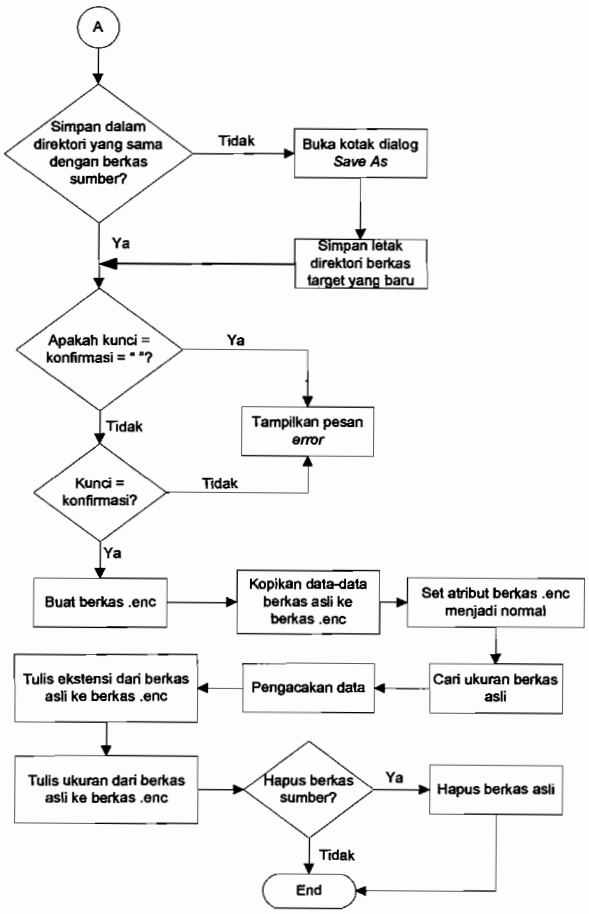
Ada beberapa perubahan tampilan dari awal perancangannya, yaitu:

- Penambahan tombol *browser* disebelah kanan objek *textbox file* yang digunakan untuk membuka *open dialog*, yang dapat mengatasi kelemahan dari objek *combo drive* yang tidak bisa menampilkan berkas-berkas dalam jaringan tetangga.

- Penambahan dua objek *check box* untuk memberikan pilihan bagi pengguna dalam menentukan apakah berkas sumber dihapus atau tidak dan menentukan apakah berkas akan disimpan dalam direktori yang sama dengan berkas sumber atau tidak.
- Penambahan objek *textbox* yang menunjukkan total waktu dari akhir proses.

Hal pertama yang perlu dilakukan pengguna adalah memilih berkas yang akan diproses. Jika berkas tersebut bukan berekstensi .enc, yang artinya berkas tersebut belum pernah dienkripsi, maka tombol Encrypt-lah yang akan aktif. Sebaliknya, tombol Decrypt akan aktif jika pengguna memilih berkas yang berekstensi .enc. Kemudian mengisi kunci pribadi miliknya dan mengkonfirmasikannya kembali. Setelah itu pengguna dapat menentukan apakah akan menghapus berkas sumber atau tidak, dan menentukan berkas hasil dari proses apakah akan disimpan dalam direktori yang sama dengan berkas sumber atau tidak.

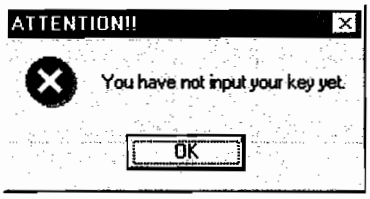
Beberapa proses yang terjadi sewaktu tombol Encrypt di-*click* dapat digambarkan dalam diagram alir berikut ini:



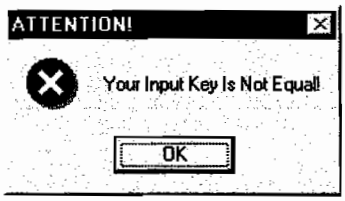
Gambar 4.3 Proses enkripsi berkas

Pada diagram alir proses enkripsi berkas dapat terlihat bahwa pesan kesalahan akan ditampilkan jika pengguna belum mengisikan kunci beserta konfirmasinya, atau kunci yang dimasukkan tidak sesuai dengan masukan konfirmasi kunci.

Berikut adalah tampilan pesan-pesan kesalahan tersebut:



Gambar 4.4 Pesan kesalahan pengguna yang belum memasukkan kunci



Gambar 4.5 Pesan kesalahan masukan kunci yang tidak sama

Khusus untuk proses pengacakan data, terdapat beberapa proses lagi yang terjadi di dalamnya. Proses-proses yang terjadi tersebut telah dijelaskan dalam analisis sistem, yaitu:

1) Inisialisasi kunci

Terdapat dua fungsi utama sewaktu melakukan inisialisasi kunci, yaitu: menetapkan 8 sub-blok kunci pertama dan menetapkan sub-blok kunci yang lain dengan cara merotasikan 25 bit ke kiri 8 sub-blok kunci sebelumnya.

Berikut adalah baris-baris perintah untuk menetapkan 8 sub-blok kunci pertama:

```
Public Sub GetUserKeyString(Arg As String)
Dim x As Integer, y As Integer
y = Len(Arg) - 1
ReDim key(0 To 8) As Integer
For x = 0 To y
    If x < 8 Then

        If x = 0 Then
            key(x) = ShiftBitsLeft(Asc(Mid(Arg, x + 1, 1)), 8)
        Else
            key(x) = ShiftBitsLeft(Asc(Mid(Arg, x + 1, 1)), 8) _
Or ShiftBitsRight(key(x - 1), 8)
        End If
    End If
Next x
If Len(Arg) > 8 Then MsgBox "ONLY first 8 characters of key
used!!", vbExclamation, "ATTENTION!"
If x < 8 Then
    Do While x < 8
        x = x + 1
        key(x) = 0
    Loop
End If
End Sub
```

Berikut adalah baris-baris perintah untuk mendapatkan 52 sub-blok kunci:

```
Private Sub EnkeyIDEA()
Dim i As Integer, j As Integer, x As Integer, y As Integer
ReDim Z(0 To KEYLEN) As Long

For j = 0 To 7
    Z(j) = key(j)
Next j
i = 0
x = 0
For j = 8 To KEYLEN - 1
    i = i + 1
    If j Mod 8 = 0 Then
        y = x * 8
        x = x + 1
    End If
    Z(j) = ShiftBitsLeft(Z((i And 7) + y), 9) Or
ShiftBitsRight(Z(((i + 1) And 7) + y), 7)
    i = i And 7
Next j
End Sub
```

2) Baca data asli

Berikut adalah baris-baris perintah untuk membaca data asli dari berkas:

```
Do While (I < 4)
   InputSubFile(i) = ShiftBitsLeft(ReadBFromFile(TempInFile), 8)
   If EndOfFile <> 0 Then
      temp = ReadBFromFile(TempInFile)
      If temp < 0 Then temp = temp + 256
      InputSubFile(i) = InputSubFile(i) Or temp
      i = i + 1
   End If
   If EndOfFile = 0 Then
      Do While (i < 4)
         InputSubFile(i) = 0
         i = i + 1
      Loop
      GoTo blok
   End If
Loop
```

3) Operasi pengacakan/pengembalian data

Berikut adalah baris-baris perintah yang berisi operasi yang digunakan untuk mengacak dan mengembalikan data:

```
Private Sub BlokIDEA()
Dim x1 As Long, x2 As Long, x3 As Long, x4 As Long, t1 As Long, t2
As Long
Dim r As Integer, i As Integer
r = ROUNDS
i = 0
x1 = InputSubFile(0)
x2 = InputSubFile(1)
x3 = InputSubFile(2)
x4 = InputSubFile(3)

Do
    x1 = MUL(x1, Z(i))
    i = i + 1
    x2 = Low16(x2 + Z(i))
    i = i + 1
    x3 = Low16(x3 + Z(i))
    i = i + 1
    x4 = MUL(x4, Z(i))
    i = i + 1
    t2 = x1 Xor x3
    t2 = MUL(t2, Z(i))
    i = i + 1
    t1 = Low16(t2 + (x2 Xor x4))
    t1 = MUL(t1, Z(i))
    i = i + 1
```

```
    t2 = Low16(t1 + t2)
    x1 = x1 Xor t1
    x4 = x4 Xor t2
    t2 = t2 Xor x2
    x2 = x3 Xor t1
    x3 = t2
    r = r - 1
Loop While (r > 0)
x1 = MUL(x1, Z(i))
i = i + 1


OutSubFile(0) = x1
OutSubFile(1) = Low16(x3 + Z(i))
i = i + 1
OutSubFile(2) = Low16(x2 + Z(i))
i = i + 1
x4 = MUL(x4, Z(i))
OutSubFile(3) = x4
End Sub
```

4) Tulis data terenkripsi

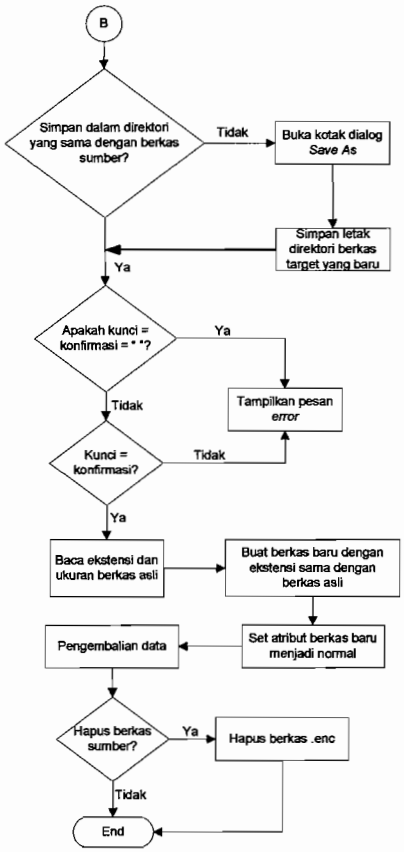
Berikut adalah baris-baris perintah untuk menulis data yang telah dienkripsi ke dalam berkas:

```
For y = 0 To 3
    If y < i Then
       WF = WriteFileB(TempOutFile, ShiftBitsRight(OutSubFile(y),_
8), 1, numwritten, 0)
       If WF = 0 Then
          MsgBox "ERROR while writting file!" & vbCr & vbCr & _
"The process might be failed!", vbExclamation, "ERROR!!!"
                    Exit Sub
                End If

                WF = WriteFileB(TempOutFile, OutSubFile(y) And _
255, 1, numwritten, 0)

                If WF = 0 Then
                    MsgBox "ERROR while writting file!" & vbCr & _
vbCr & "The process might be failed!", _
                        vbExclamation, "ERROR!!!"
                    Exit Sub
                End If
            End If
        Next y
```

Sedangkan proses-proses yang terjadi sewaktu tombol Decrypt di-*click* dapat digambarkan dalam diagram alir berikut ini:



Gambar 4.6 Proses dekripsi berkas

Khusus untuk proses pengembalian data, terdapat juga beberapa proses lagi yang terjadi di dalamnya. Proses-proses yang terjadi tersebut juga telah dijelaskan dalam analisis sistem, yaitu:

1) Inisialisasi kunci dekripsi

Sub-blok kunci dekripsi dapat dicari dengan terlebih dahulu menemukan 52 sub-blok kunci enkripsi yang kemudian diubah penjadwalannya seperti pada tabel 2.1. Berikut adalah baris-baris perintah untuk mengubah penjadwalan dari sub-blok kunci enkripsi:

```
Private Sub DekeyIDEA()
Dim j As Integer, i As Integer, k As Integer
Dim t1 As Long, t2 As Long, t3 As Long
Dim temp As Long
ReDim p(0 To KEYLEN) As Long

i = 0
k = 52

t1 = INVER(Z(i))
i = i + 1
t2 = CheckMin(Z(i))
i = i + 1
t3 = CheckMin(Z(i))
i = i + 1
k = k - 1
p(k) = INVER(Z(i))
i = i + 1
k = k - 1
p(k) = t3
k = k - 1
p(k) = t2
k = k - 1
p(k) = t1
For j = 1 To ROUNDS - 1
    t1 = Z(i)
    i = i + 1
    k = k - 1
    p(k) = Z(i)
    i = i + 1
    k = k - 1
    p(k) = t1
    t1 = INVER(Z(i))
    i = i + 1
    t2 = CheckMin(Z(i))
    i = i + 1
    t3 = CheckMin(Z(i))
    i = i + 1
    k = k - 1
    p(k) = INVER(Z(i))
    i = i + 1
    k = k - 1
    p(k) = t2
    k = k - 1
    p(k) = t3
    k = k - 1
    p(k) = t1
Next j
t1 = Z(i)
i = i + 1
k = k - 1
p(k) = Z(i)
i = i + 1
k = k - 1
p(k) = t1
t1 = INVER(Z(i))
```

```
i = i + 1
t2 = CheckMin(Z(i))
i = i + 1
t3 = CheckMin(Z(i))
i = i + 1
k = k - 1
p(k) = INVER(Z(i))
i = i + 1
k = k - 1
p(k) = t3
k = k - 1
p(k) = t2
k = k - 1
p(k) = t1

'--copy and destroy temp copy--
'ReDim DK(0 To KEYLEN) As Long
i = 0
k = 0
For j = 0 To KEYLEN - 1
    Z(i) = p(k)
    i = i + 1
    p(k) = 0
    k = k + 1
Next j
End Sub
```

2) Baca data terenkripsi

Berikut adalah baris-baris perintah untuk membaca data terenkripsi dari berkas:

```
Do While (i < 4)
   InputSubFile(i) = ShiftBitsLeft(ReadBFromFile(TempInFile), 8)
   If EndOfFile <> 0 Then
      temp = ReadBFromFile(TempInFile)
      If temp < 0 Then temp = temp + 256
      InputSubFile(i) = InputSubFile(i) Or temp
   End If
   If EndOfFile = 0 Then GoTo err
   i = i + 1
Loop
```

3) Operasi pengacakan/pengembalian data

Fungsi yang dipakai sama dengan fungsi untuk operasi pengacakan/pengembalian data sewaktu melakukan enkripsi berkas.

4) Tulis data asli

Berikut adalah baris-baris perintah untuk menulis data asli ke dalam berkas:

```
For y = 0 To 3
   If y < i Then
      If length > 0 Then
         length = length - 1
         n = CheckOver(ShiftBitsRight(OutSubFile(y), 8))
         WF = WriteFileB(TempOutFile, n, Len(n), numwritten, 0)
         If WF = 0 Then
             MsgBox "ERROR while writting file!" & vbCr & vbCr & _
"The process might be failed!", vbExclamation, "ERROR!!!"
             Exit Sub
         End If
      End If
      If length > 0 Then
         length = length - 1
         n = OutSubFile(y) And 255
         WF = WriteFileB(TempOutFile, n, 1, numwritten, 0)
         If WF = 0 Then
             MsgBox "ERROR while writting file!" & vbCr & vbCr & _
"The process might be failed!", vbExclamation, "ERROR!!!"
          Exit Sub
         End If
      End If
      If length = 0 Then
         SetEndOfFile TempOutFile
         GoTo err
      End If
  End If
Next y
```

Ada beberapa fungsi pendukung yang digunakan dalam beberapa fungsi lainnya, yaitu:

1) Fungsi untuk melakukan penggeseran bit

Fungsi ini dibuat karena VB tidak menyediakan operator khusus untuk ini. Ada dua fungsi yang dibuat, yaitu fungsi untuk menggeser bit ke kiri dan fungsi untuk menggeser bit ke kanan.

Berikut adalah baris-baris perintah untuk menggeser bit ke kiri:

```
Private Function ShiftBitsLeft(ByVal dwValue As Long, _
                               ByVal dwBitsLeft As Long) As Long
'bitwise left-shift equivalent to
'C++ << operator
Dim temp As Long
temp = CLng(dwValue * (2 ^ dwBitsLeft))
If temp > 65535 Then temp = temp - (MODSHIFT_L And temp)
ShiftBitsLeft = temp
End Function
```

Berikut adalah baris-baris perintah untuk menggeser bit ke kanan:

```
Private Function ShiftBitsRight(ByVal dwValue As Long, _
                                ByVal dwBitsRight As Long) As Long
'bitwise right-shift equivalent to
'C++ >> operator
ShiftBitsRight = CLng(dwValue \ (2 ^ dwBitsRight))
End Function
```

2) Fungsi untuk melakukan perkalian modulo 2^{16}

Fungsi ini dibuat dalam bentuk berkas .dll dengan menggunakan Visual C++. Hal ini dilakukan karena VB tidak mampu menampung hasil perkalian dalam *unsigned long*. Berikut adalah baris-baris perintahnya:

```
#include "stdafx.h"

#define word32 unsigned long int
#define low16(x) ((x) & 0xffff)
typedef unsigned int uint16;

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserve)
{return TRUE;}

uint16 _stdcall MUL(uint16 a, uint16 b)
{
      word32 p;
      if (a)
      {
            if (b)
            {
                  p = (word32)a*b;
                  b = (uint16)low16(p);
                  a = (uint16)p>>16;
                  if (b < a)
                  { return low16((b - a) + 65537); }
```

```
                    else
                    { return (b - a); }
              }
              else
              {
                    return low16(65537 - a);
              }
        }
        else
              return low16(65537 - b);
}
```

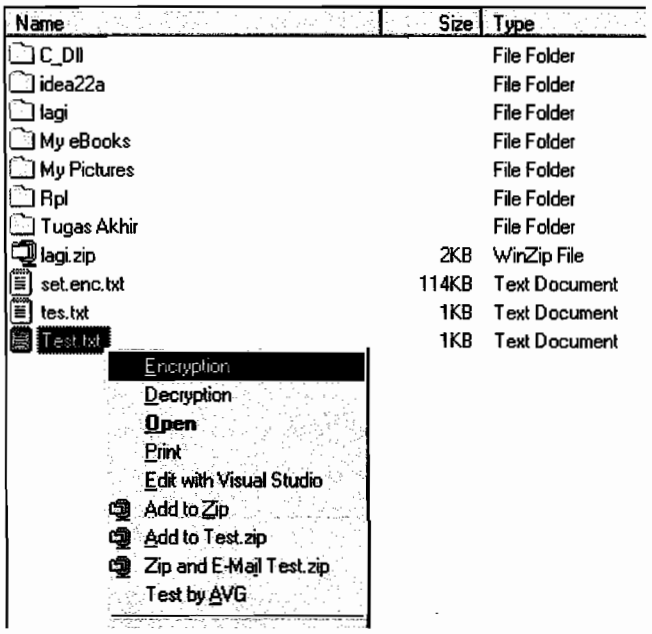
3) Fungsi untuk melakukan invers dari perkalian modulo 2^{16}

Berikut adalah baris-baris perintah untuk melakukan invers:

```
Private Function INVER(x As Long) As Long
Dim t0 As Long, t1 As Long
Dim q As Long, y As Long
If (x <= 1) Then
    INVER = x
    Exit Function
End If
t1 = CLng(MAX_INT \ x)
y = CLng(MAX_INT Mod x)
If y = 1 Then
    INVER = Low16(MAX_INT - t1)
    Exit Function
End If
t0 = 1
Do
q = Int(x \ y)
x = x Mod y
t0 = t0 + (q * t1)
If x = 1 Then
    INVER = t0
    Exit Function
End If
q = Int(y \ x)
y = y Mod x
t1 = t1 + (q * t0)
Loop While (y <> 1)
INVER = Low16(MAX_INT - t1)
End Function
```

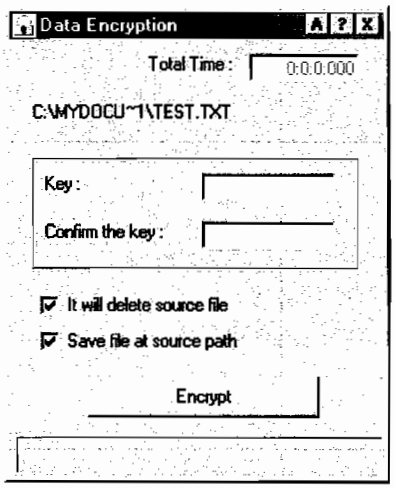
4.2 *Form* Enkripsi

Form ini akan ditampilkan jika pengguna mengaktifkan program ini dengan memilih menu enkripsi pada *context menu Windows* dan berkas yang terpilih merupakan berkas yang belum dienkripsi. Hal ini seperti terlihat pada gambar berikut ini:



Gambar 4.7 Menu enkripsi dalam *context menu Windows*

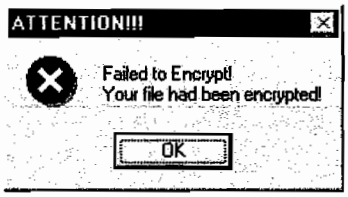
Berikut adalah tampilan implementasi *form* enkripsi:



Gambar 4.8 Tampilan implementasi *form* enkripsi

Seperti pada *form* utama, *form* enkripsi inipun mengalami perubahan dari rancangan awalnya, yaitu terdapat penambahan dua objek *check box* untuk memberikan pilihan bagi pengguna dalam menentukan apakah berkas sumber dihapus atau tidak dan menentukan apakah berkas akan disimpan dalam direktori yang sama dengan berkas sumber atau tidak. Proses yang dijalankan sewaktu tombol Encrypt di-*click* sama dengan proses yang serupa terjadi pada *form* utama.

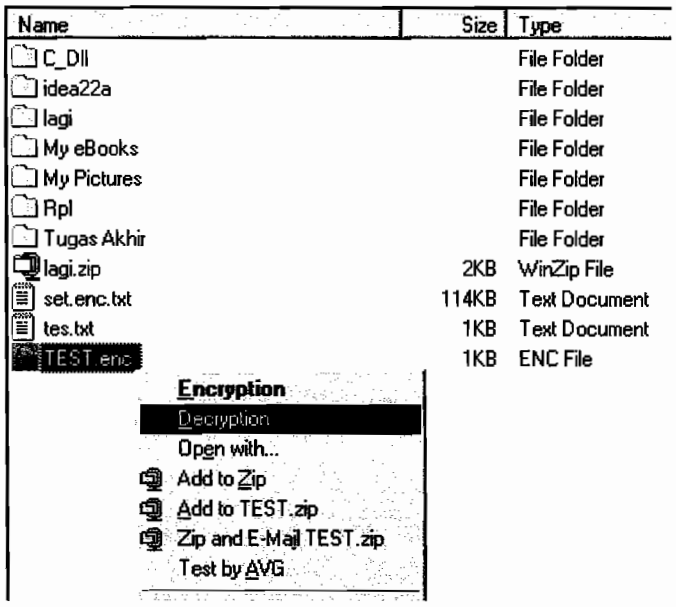
Jika pengguna ternyata mengaktifkan enkripsi pada *context menu Windows* dengan berkas terpilih merupakan berkas yang telah dienkripsi, maka akan ditampilkan pesan kesalahan seperti terlihat pada gambar berikut ini:



Gambar 4.9 Pesan kesalahan enkripsi berkas yang telah terenkripsi

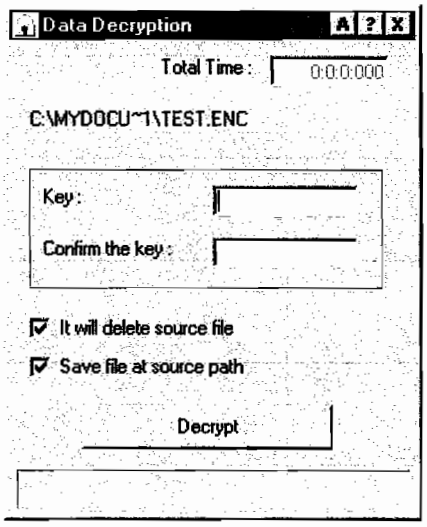
4.3 *Form* Dekripsi

Form ini akan ditampilkan jika pengguna mengaktifkan program ini dengan memilih menu dekripsi pada *context menu Windows* dan berkas yang terpilih merupakan berkas yang telah dienkripsi. Hal ini seperti terlihat pada gambar berikut ini:



Gambar 4.10 Menu dekripsi dalam *context menu Windows*

Berikut adalah tampilan implementasi *form* dekripsi:

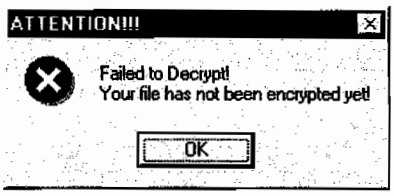


Gambar 4.11 Tampilan implementasi *form* dekripsi

Seperti pada *form* enkripsi, *form* inipun juga mengalami perubahan dari rancangan awalnya, yaitu terdapat penambahan dua objek *check box* untuk memberikan pilihan bagi pengguna dalam menentukan apakah berkas sumber dihapus atau tidak dan menentukan apakah berkas akan disimpan dalam direktori yang sama

dengan berkas sumber atau tidak. Proses yang dijalankan sewaktu tombol Decrypt di-*click* sama dengan proses yang serupa terjadi pada *form* utama.

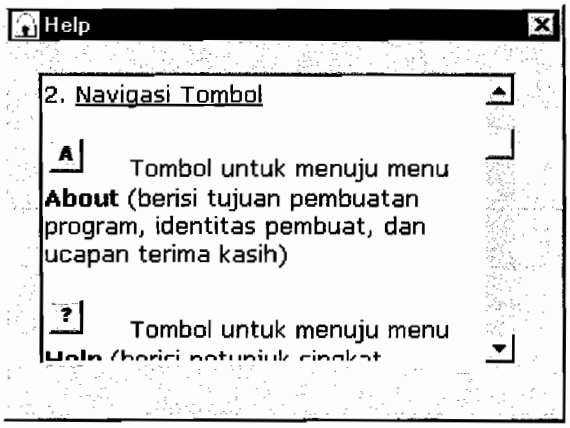
Jika pengguna ternyata mengaktifkan dekripsi pada *context menu Windows* dengan berkas terpilih merupakan berkas yang belum dienkripsi, maka akan ditampilkan pesan kesalahan seperti terlihat pada gambar berikut ini:



Gambar 4.12 Pesan kesalahan dekripsi berkas yang belum terenkripsi

4.4 *Form* Bantu

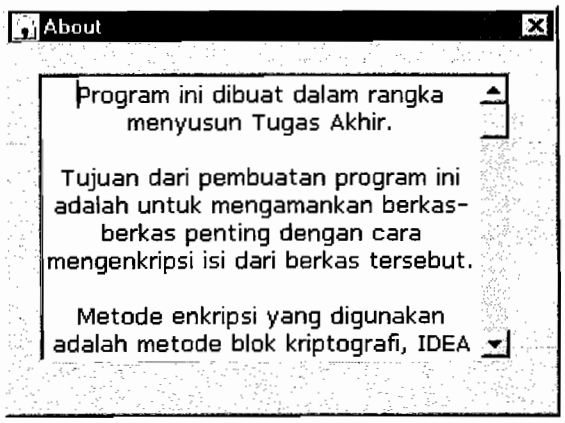
Seperti pada rancangan awalnya, *form* ini tidak mengalami perubahan dan memiliki kegunaan yang sama, yaitu menampilkan petunjuk-petunjuk penggunaan program ini. *Form* ini akan ditampilkan sewaktu pengguna me-*click* tombol ?. Berikut adalah tampilan implementasi *form* bantu:



Gambar 4.13 Tampilan implementasi *form* bantu

4.5 *Form About*

Seperti pada rancangan awalnya, *form* ini juga tidak mengalami perubahan dan memiliki kegunaan yang sama, yaitu menampilkan tujuan pembuatan program, identitas pembuat, dan ucapan terima kasih. *Form* ini akan ditampilkan sewaktu pengguna me-*click* tombol  . Berikut adalah tampilan implementasi *form about*:



Gambar 4.14 Tampilan implementasi *form about*

BAB V

UJI COBA DAN ANALISA HASIL

5.1 Uji Coba

Uji coba terhadap sistem ini dilakukan untuk menjawab beberapa pertanyaan penting sebagai berikut:

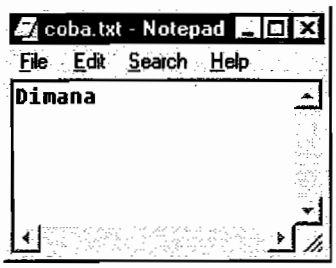
1) Apakah program berjalan sesuai dengan algoritma IDEA yang dipakai?
2) Apakah program dapat mengenkripsi dan mendekripsi dengan baik beberapa berkas dokumen (.txt, .doc, .rtf) dan berkas grafis (.jpg, .bmp, .psd)?
3) Apakah ukuran suatu berkas akan berpengaruh secara signifikan terhadap waktu proses?

5.1.1 Uji Coba Kesesuaian Hasil Program dengan Algoritma IDEA

Uji coba ini dilakukan untuk mengetahui apakah program telah berjalan sesuai dengan algoritma IDEA yang dipakai. Hal ini dilakukan dengan cara membandingkan hasil dari program dengan hasil yang telah dicontohkan dalam landasan teori.

Langkah-langkah dalam uji coba enkripsi berkas sebagai berikut:

1) Membuat berkas coba.txt dengan isi seperti terlihat pada gambar berikut:

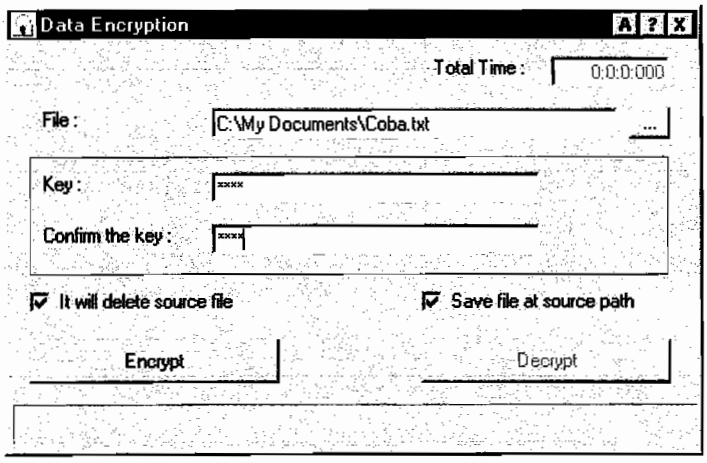


Gambar 5.1 Tampilan berkas coba.txt dalam notepad

2) Menjalankan program enkripsi-dekripsi langsung dari berkas eksekusinya.

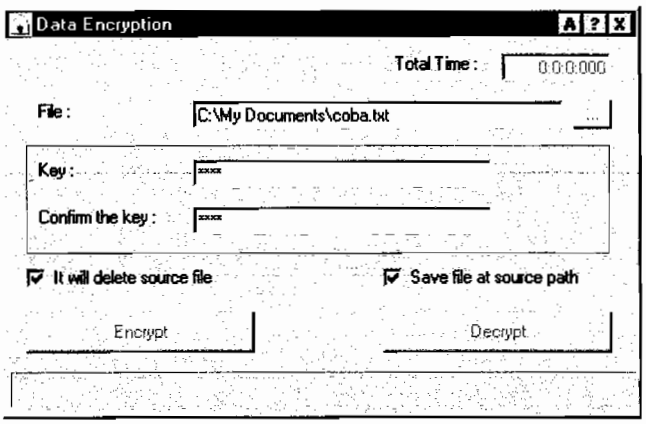
3) Memilih berkas coba.txt.

4) Memasukkan kunci = 1234 dan mengkonfirmasikannya. Tampilannya akan menjadi sebagai berikut:



Gambar 5.2 Uji coba enkripsi berkas coba.txt

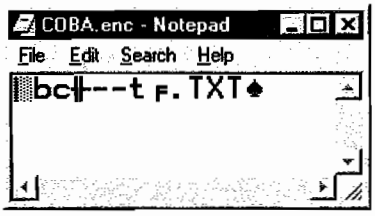
5) Me-*click* tombol Encrypt. Setelah proses enkripsi selesai, maka tampilannya akan berubah menjadi gambar berikut:



Gambar 5.3 Proses enkripsi berkas coba.txt selesai

Gambar 5.4 Pesan sukses dalam melakukan enkripsi berkas

6) Membuka berkas hasil enkripsi (coba.enc) dengan program NOTEPAD.exe untuk melihat hasil enkripsinya. Sebelumnya jenis *font* dalam program NOTEPAD diset sebagai *Terminal*. Berikut tampilan berkas coba.enc:



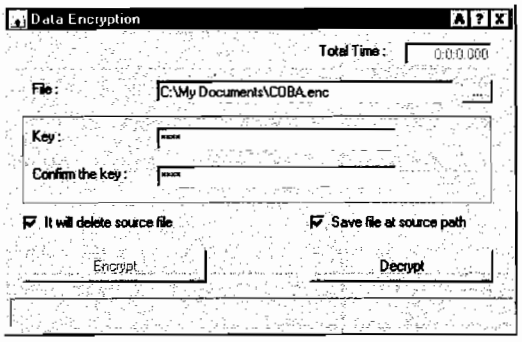
Gambar 5.5 Tampilan berkas coba.enc dalam notepad

Dari tampilan Gambar 5.5 terlihat bahwa hasil percobaan enkripsi berkas coba.txt ini sama dengan hasil enkripsi pada contoh dalam landasan teori (halaman 16).

Hanya saja pada hasil enkripsi berkas ini terdapat tambahan ".TXT" dan karakter "♠", hal ini disebabkan oleh adanya penambahan data setelah proses enkripsi berkas berakhir. Data yang ditambahkan adalah jenis ektensi dari berkas asli (.TXT) dan sisanya merupakan ukuran dari berkas asli yang ditampilkan sebagai karakter "♠".

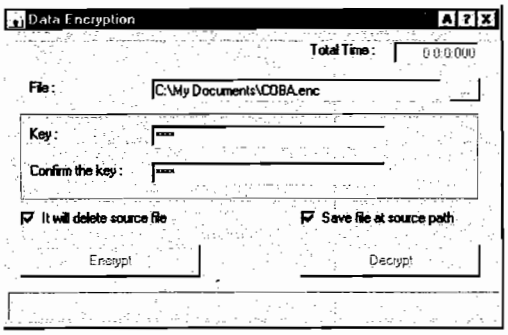
Langkah-langkah dalam uji coba dekripsi berkas sebagai berikut:

1) Menjalankan program enkripsi-dekripsi langsung dari berkas eksekusinya.
2) Memilih berkas coba.enc.
3) Memasukkan kunci = 1234 dan mengkonfirmasikannya. Tampilannya akan menjadi sebagai berikut:

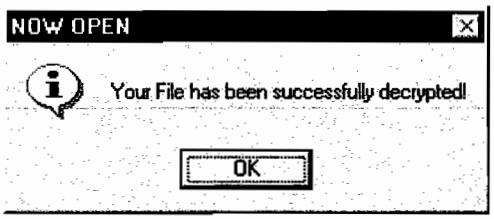


Gambar 5.6 Uji coba dekripsi berkas coba.enc

4) Me-*click* tombol Encrypt. Setelah proses enkripsi selesai, maka tampilannya akan berubah menjadi gambar berikut:

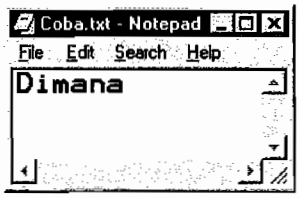


Gambar 5.7 Proses dekripsi berkas coba.txt selesai



Gambar 5.8 Pesan sukses dalam melakukan dekripsi berkas

5) Membuka berkas hasil dekripsi (coba.txt) dengan program NOTEPAD.exe untuk melihat hasil dekripsinya. Berikut tampilan berkas coba.txt setelah dienkripsi dan didekripsi:



Gambar 5.9 Berkas coba.txt setelah didekripsi

Dari tampilan Gambar 5.9 terlihat bahwa program berhasil mengembalikan data aslinya setelah mendekripsi berkas coba.enc. Perbedaan hanya terlihat pada jenis *font*-nya saja. Hal ini disebabkan oleh karena jenis *font* dalam program NOTEPAD tersebut masih diset sebagai *Terminal*.

5.1.2 Uji Coba Berkas Dokumen dan Berkas Grafis

Uji coba ini dilakukan untuk mengetahui apakah program dapat mengenkripsi dan mendekripsi berkas dokumen dan berkas grafis dengan baik, apakah ukuran suatu berkas akan berpengaruh secara signifikan terhadap waktu proses, serta apakah waktu yang dibutuhkan dalam mengenkripsi berkas berbeda secara signifikan dengan waktu yang dibutuhkan untuk mendekripsi berkas.

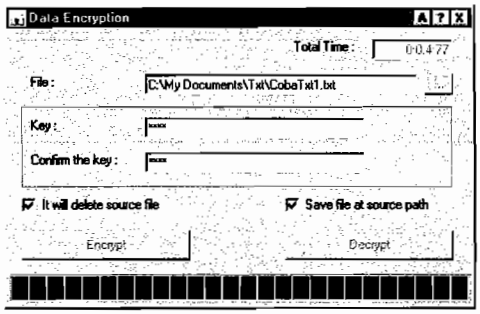
1) Enkripsi berkas dokumen dan berkas grafis

a. Uji coba berkas bertipe txt

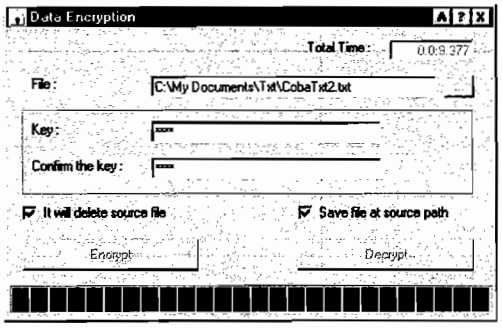
Berkas txt yang akan dienkripsi adalah:

1. CobaTxt1.txt yang berukuran 115.982 *bytes*.
2. CobaTxt2.txt yang berukuran 347.944 *bytes*.
3. CobaTxt3.txt yang berukuran 707.967 *bytes*.

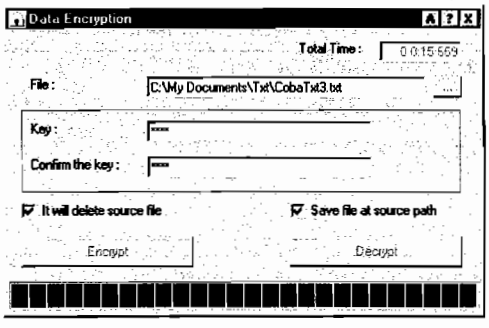
Berikut tampilan hasil enkripsi ketiga berkas tersebut:



Gambar 5.10 Hasil enkripsi berkas CobaTxt1.txt



Gambar 5.11 Hasil enkripsi berkas CobaTxt2.txt



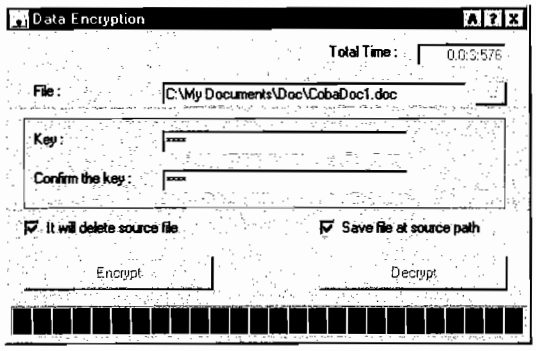
Gambar 5.12 Hasil enkripsi berkas CobaTxt3.txt

b. Uji coba berkas bertipe doc

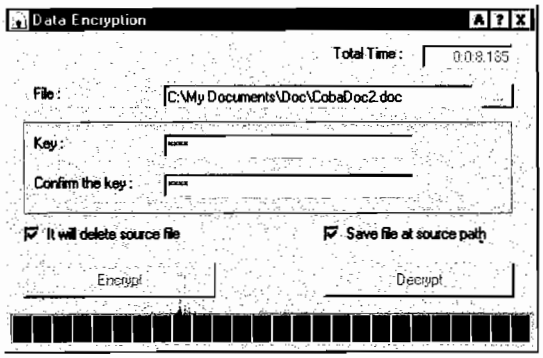
Berkas doc yang akan dienkripsi adalah:

1. CobaDoc1.doc yang berukuran 131.072 *bytes*.
2. CobaDoc2.doc yang berukuran 353.792 *bytes*.
3. CobaDoc3.doc yang berukuran 539.648 *bytes*.

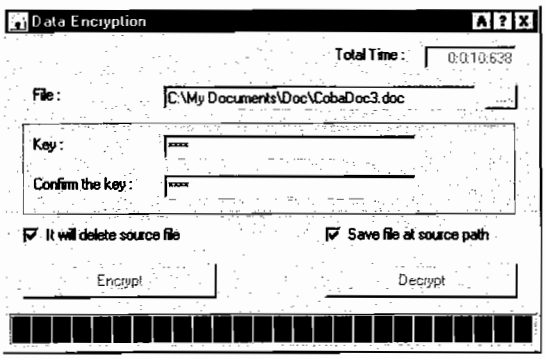
Berikut tampilan hasil enkripsi ketiga berkas tersebut:



Gambar 5.13 Hasil enkripsi berkas CobaDoc1.doc



Gambar 5.14 Hasil enkripsi berkas CobaDoc2.doc



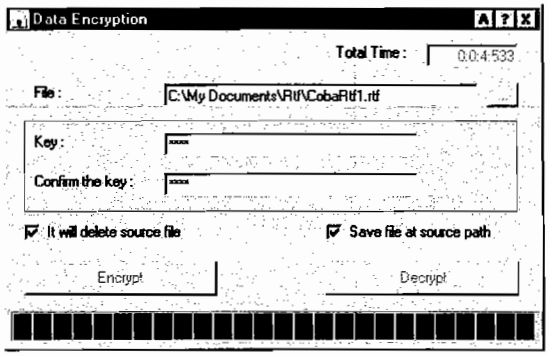
Gambar 5.15 Hasil enkripsi berkas CobaDoc3.doc

c. Uji coba berkas bertipe rtf

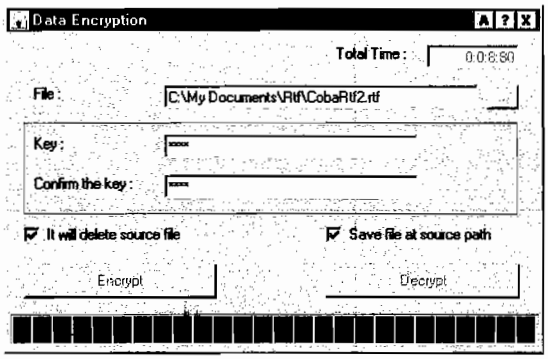
Berkas rtf yang akan dienkripsi adalah:

1. CobaRtf1.rtf yang berukuran 159.207 *bytes*.
2. CobaRtf2.rtf yang berukuran 347.593 *bytes*.
3. CobaRtf3.rtf yang berukuran 543.768 *bytes*.

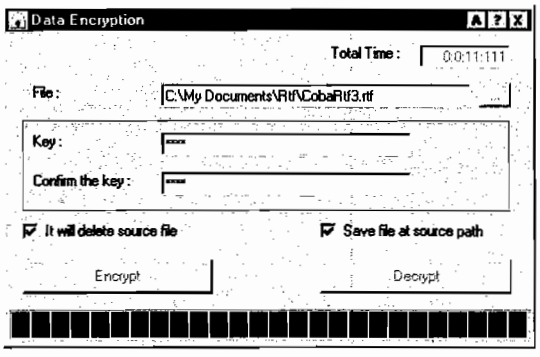
Berikut tampilan hasil enkripsi ketiga berkas tersebut:



Gambar 5.16 Hasil enkripsi berkas CobaRtf1.rtf



Gambar 5.17 Hasil enkripsi berkas CobaRtf2.rtf



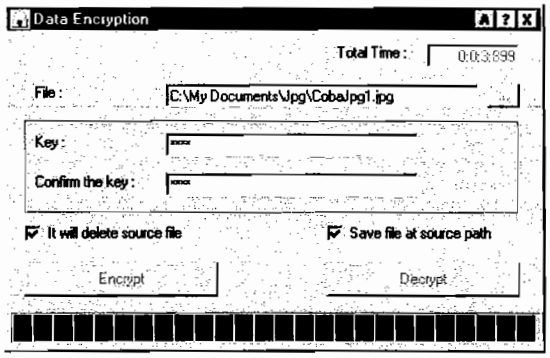
Gambar 5.18 Hasil enkripsi berkas CobaRtf3.rtf

d. Uji coba berkas bertipe jpg

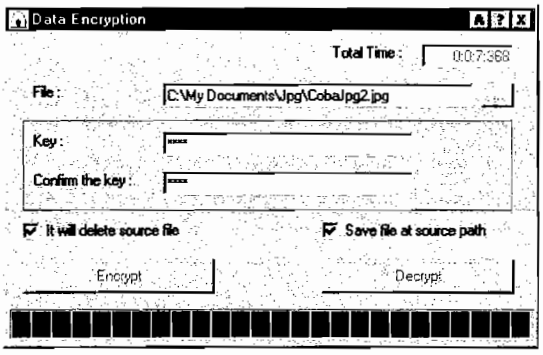
Berkas jpg yang akan dienkripsi adalah:

1. CobaJpg1.jpg yang berukuran 138.705 *bytes*.
2. CobaJpg2.jpg yang berukuran 302.514 *bytes*.
3. CobaJpg3.jpg yang berukuran 612.504 *bytes*.

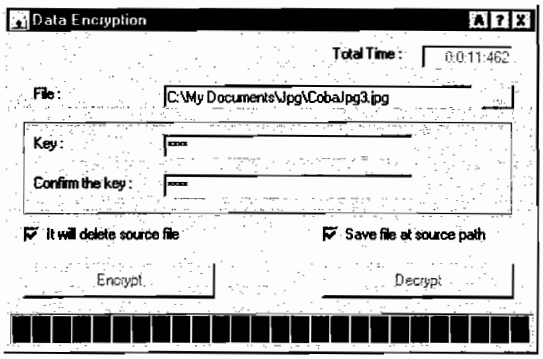
Berikut tampilan hasil enkripsi ketiga berkas tersebut:



Gambar 5.19 Hasil enkripsi berkas CobaJpg1.jpg



Gambar 5.20 Hasil enkripsi berkas CobaJpg2.jpg



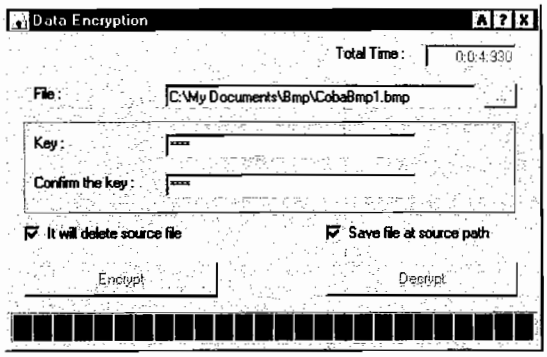
Gambar 5.21 Hasil enkripsi berkas CobaJpg3.jpg

e. Uji coba berkas bertipe bmp

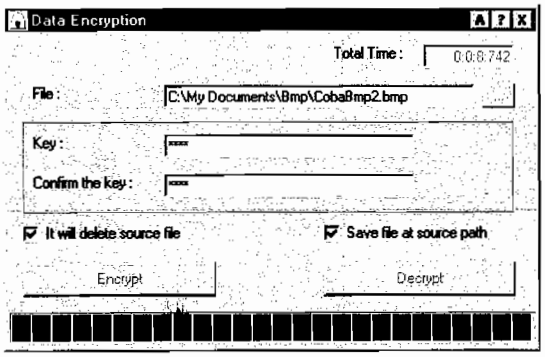
Berkas bmp yang akan dienkripsi adalah:

1. CobaBmp1.bmp yang berukuran 180.056 *bytes*.
2. CobaBmp2.bmp yang berukuran 378.056 *bytes*.
3. CobaBmp3.bmp yang berukuran 648.056 *bytes*.

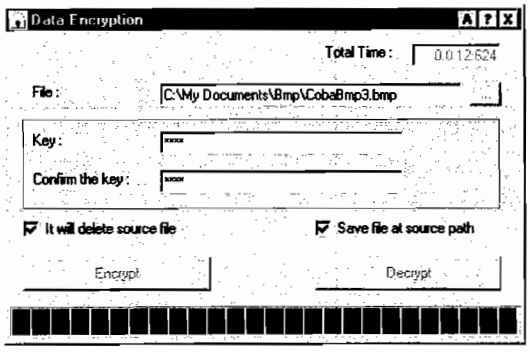
Berikut tampilan hasil enkripsi ketiga berkas tersebut:



Gambar 5.22 Hasil enkripsi berkas CobaBmp1.bmp



Gambar 5.23 Hasil enkripsi berkas CobaBmp2.bmp



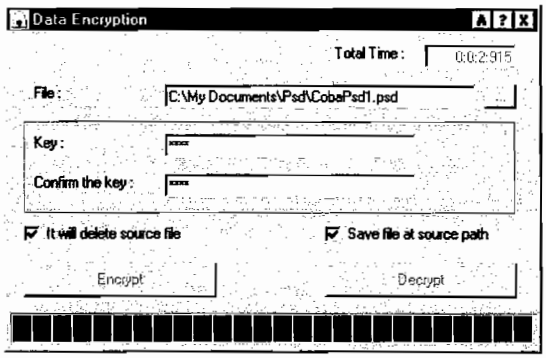
Gambar 5.24 Hasil enkripsi berkas CobaBmp3.bmp

f. Uji coba berkas bertipe psd

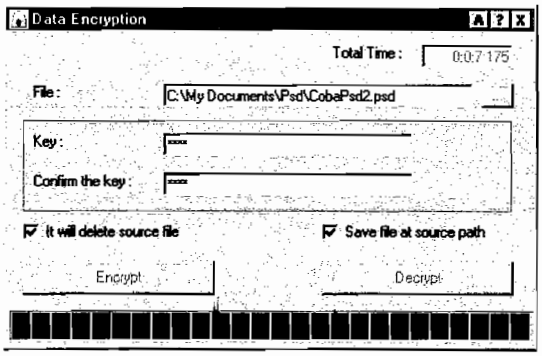
Berkas psd yang akan dienkripsi adalah:

1. CobaPsd1.psd yang berukuran 102.586 *bytes*.
2. CobaPsd2.psd yang berukuran 291.918 *bytes*.
3. CobaPsd3.psd yang berukuran 522.580 *bytes*.

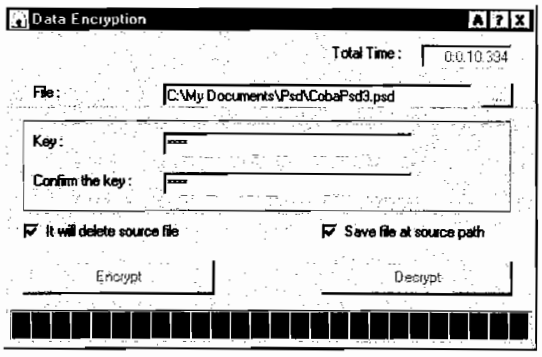
Berikut tampilan hasil enkripsi ketiga berkas tersebut:



Gambar 5.25 Hasil enkripsi berkas CobaPsd1.psd



Gambar 5.26 Hasil enkripsi berkas CobaPsd2.psd



Gambar 5.27 Hasil enkripsi berkas CobaPsd3.psd

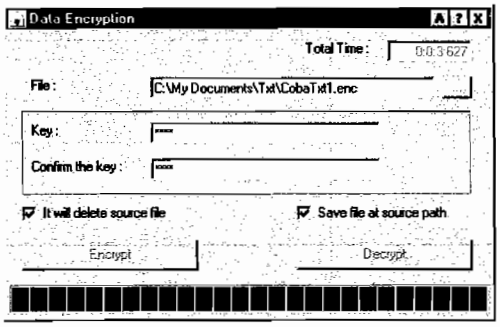
2) Dekripsi berkas dokumen dan berkas grafis

a. Uji coba berkas terenkripsi dari berkas asli bertipe txt

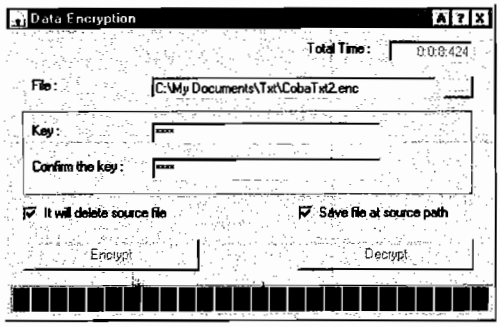
Berkas yang akan didekripsi adalah:

1. CobaTxt1.enc yang berukuran 115.992 *bytes*.
2. CobaTxt2.enc yang berukuran 347.960 *bytes*.
3. CobaTxt3.enc yang berukuran 707.976 *bytes*.

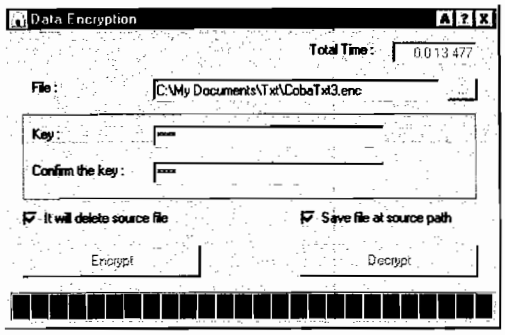
Berikut tampilan hasil dekripsi ketiga berkas tersebut:



Gambar 5.28 Hasil dekripsi berkas CobaTxt1.enc



Gambar 5.29 Hasil dekripsi berkas CobaTxt2.enc



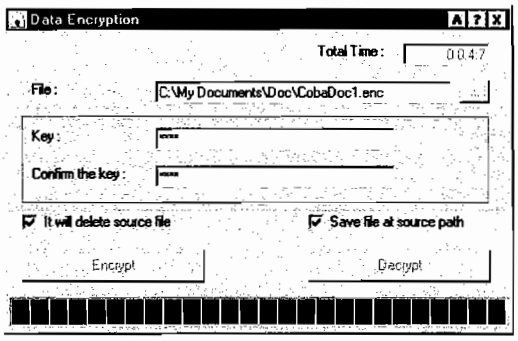
Gambar 5.30 Hasil dekripsi berkas CobaTxt3.enc

b. Uji coba berkas terenkripsi dari berkas asli bertipe doc

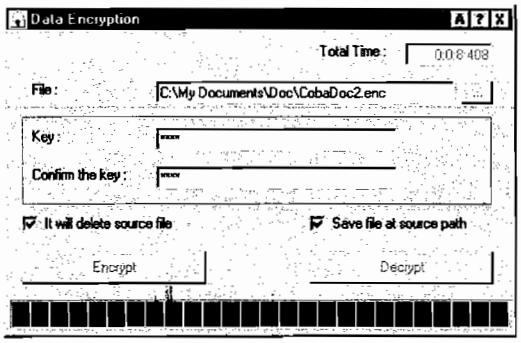
Berkas yang akan didekripsi adalah:

1. CobaDoc1.enc yang berukuran 131.088 *bytes*.
2. CobaDoc2.enc yang berukuran 353.808 *bytes*.
3. CobaDoc3.enc yang berukuran 539.664 *bytes*.

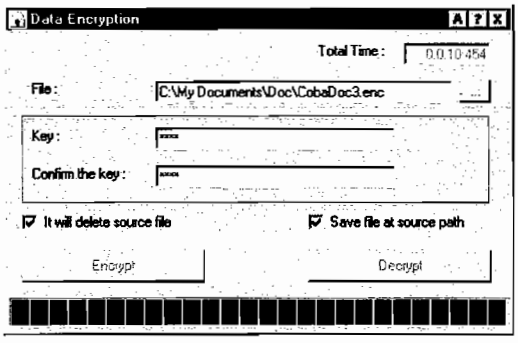
Berikut tampilan hasil dekripsi ketiga berkas tersebut:



Gambar 5.31 Hasil dekripsi berkas CobaDoc1.enc



Gambar 5.32 Hasil dekripsi berkas CobaDoc2.enc



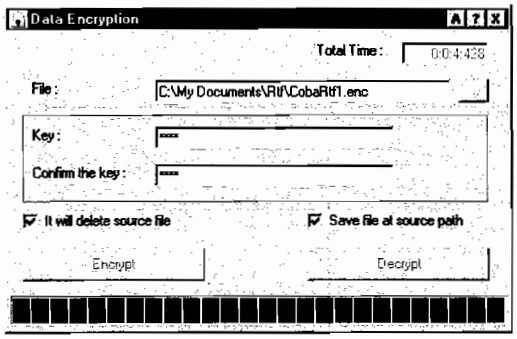
Gambar 5.33 Hasil dekripsi berkas CobaDoc3.enc

c. Uji coba berkas terenkripsi dari berkas asli bertipe rtf

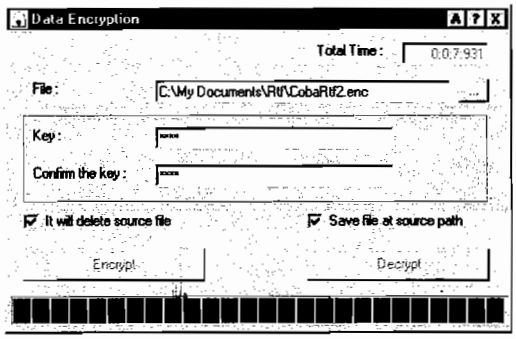
Berkas yang akan didekripsi adalah:

1. CobaRtf1.enc yang berukuran 159.216 *bytes*.
2. CobaRtf2.enc yang berukuran 347.608 *bytes*.
3. CobaRtf3.enc yang berukuran 543.784 *bytes*.

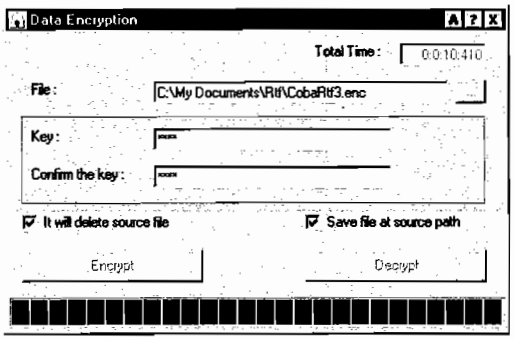
Berikut tampilan hasil dekripsi ketiga berkas tersebut:



Gambar 5.34 Hasil dekripsi berkas CobaRtf1.enc



Gambar 5.35 Hasil dekripsi berkas CobaRtf2.enc



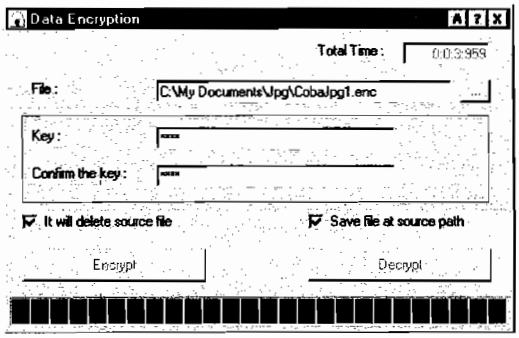
Gambar 5.36 Hasil dekripsi berkas CobaRtf2.enc

d. Uji coba berkas terenkripsi dari berkas asli bertipe jpg

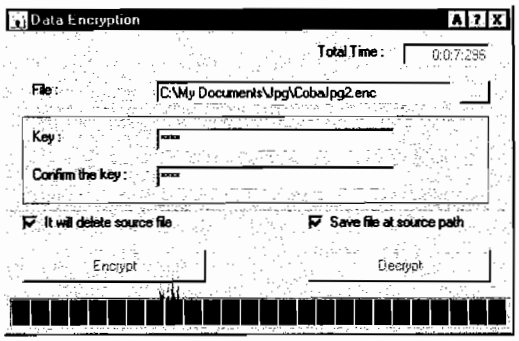
Berkas yang akan didekripsi adalah:

1. CobaJpg1.enc yang berukuran 138.720 *bytes*.
2. CobaJpg2.enc yang berukuran 302.528 *bytes*.
3. CobaJpg3.enc yang berukuran 612.520 *bytes*.

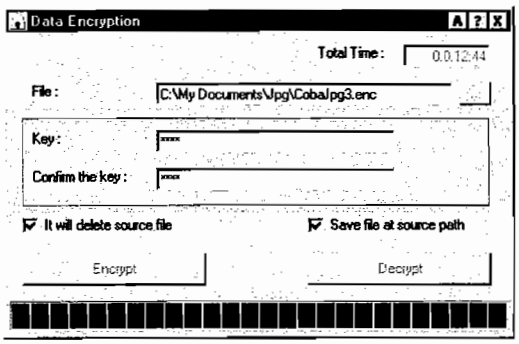
Berikut tampilan hasil dekripsi ketiga berkas tersebut:



Gambar 5.37 Hasil dekripsi berkas CobaJpg1.enc



Gambar 5.38 Hasil dekripsi berkas CobaJpg2.enc



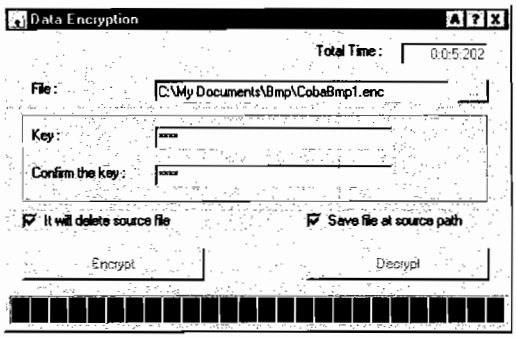
Gambar 5.39 Hasil dekripsi berkas CobaJpg3.enc

e. Uji coba berkas terenkripsi dari berkas asli bertipe bmp

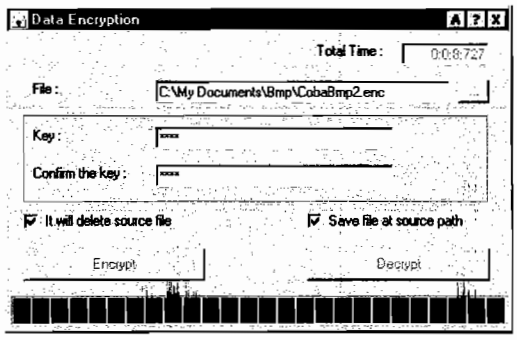
Berkas yang akan didekripsi adalah:

1. CobaBmp1.enc yang berukuran 180.072 *bytes*.
2. CobaBmp2.enc yang berukuran 378.072 *bytes*.
3. CobaBmp3.enc yang berukuran 648.072 *bytes*.

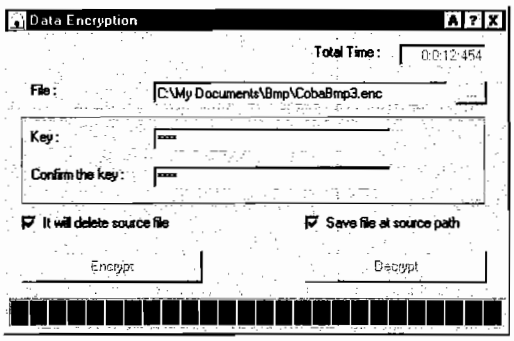
Berikut tampilan hasil dekripsi ketiga berkas tersebut:



Gambar 5.40 Hasil dekripsi berkas CobaBmp1.enc



Gambar 5.41 Hasil dekripsi berkas CobaBmp2.enc



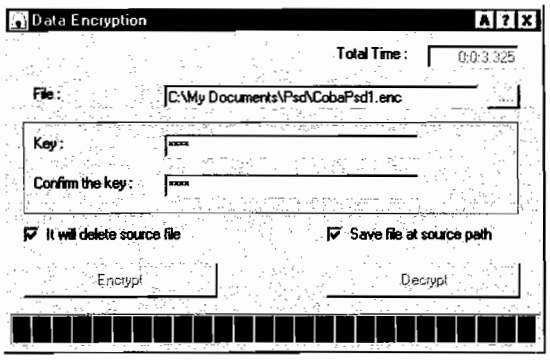
Gambar 5.42 Hasil dekripsi berkas CobaBmp3.enc

f. Uji coba berkas terenkripsi dari berkas asli bertipe psd

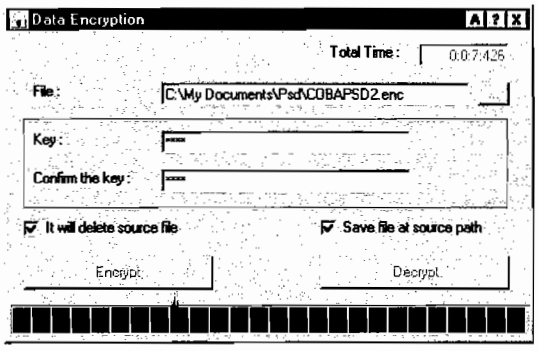
Berkas yang akan didekripsi adalah:

1. CobaPsd1.enc yang berukuran 102.600 *bytes*.
2. CobaPsd2.enc yang berukuran 291.928 *bytes*.
3. CobaPsd3.enc yang berukuran 522.592 *bytes*.

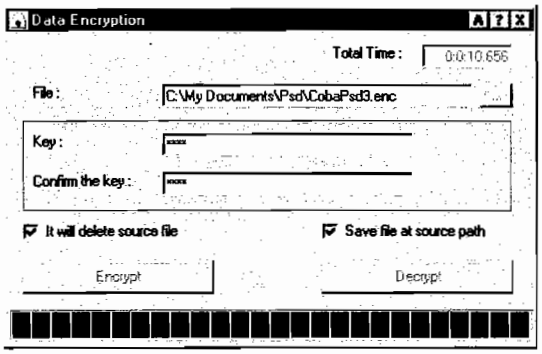
Berikut tampilan hasil dekripsi ketiga berkas tersebut:



Gambar 5.43 Hasil dekripsi berkas CobaPsd1.enc



Gambar 5.44 Hasil dekripsi berkas CobaPsd2.enc



Gambar 5.45 Hasil dekripsi berkas CobaPsd3.enc

Dari hasil coba dapat dijelaskan sebagai berikut:

Tabel 5.1 Tabel uji coba enkripsi berkas dokumen dan berkas grafis

No.	Nama Berkas	Tipe Berkas	Ukuran (*bytes*)	Hasil Enkripsi	Waktu (detik)	Ratio (*byte*/detik)
1	CobaTxt1	txt	115982	Sukses	4.77	24314.885
2	CobaTxt2	txt	347944	Sukses	9.377	37106.111
3	CobaTxt3	txt	707967	Sukses	15.669	45182.654
4	CobaDoc1	doc	131072	Sukses	3.576	36653.244
5	CobaDoc2	doc	353792	Sukses	8.185	43224.435
6	CobaDoc3	doc	539648	Sukses	10.638	50728.332
7	CobaRtf1	rtf	159207	Sukses	4.533	35121.774
8	CobaRtf2	rtf	347593	Sukses	8.8	39499.205
9	CobaRtf3	rtf	543768	Sukses	11.111	48939.609
10	CobaJpg1	jpg	138705	Sukses	3.899	35574.506
11	CobaJpg2	jpg	302514	Sukses	7.368	41057.818
12	CobaJpg3	jpg	612504	Sukses	11.462	53437.794
13	CobaBmp1	bmp	180056	Sukses	4.93	36522.515
14	CobaBmp2	bmp	378056	Sukses	8.742	43245.939
15	CobaBmp3	bmp	648056	Sukses	12.624	51335.234
16	CobaPsd1	psd	102586	Sukses	2.915	35192.453
17	CobaPsd2	psd	291918	Sukses	7.175	40685.436
18	CobaPsd3	psd	522580	Sukses	10.394	50277.083

Tabel 5.2 Tabel uji coba dekripsi berkas dokumen dan berkas grafis

No.	Nama Berkas	Tipe Berkas	Ukuran (*bytes*)	Hasil Dekripsi	Waktu (detik)	Ratio (*byte*/detik)
1	CobaTxt1	enc	115992	Sukses	3.627	31980.149
2	CobaTxt2	enc	347960	Sukses	8.424	41305.793
3	CobaTxt3	enc	707976	Sukses	13.477	52532.166
4	CobaDoc1	enc	131088	Sukses	4.7	27891.064
5	CobaDoc2	enc	353808	Sukses	8.408	42079.924
6	CobaDoc3	enc	539664	Sukses	10.454	51622.728
7	CobaRtf1	enc	159216	Sukses	4.428	35956.640
8	CobaRtf2	enc	347608	Sukses	7.931	43829.025
9	CobaRtf3	enc	543784	Sukses	10.41	52236.695
10	CobaJpg1	enc	138720	Sukses	3.959	35039.151
11	CobaJpg2	enc	302528	Sukses	7.296	41464.912
12	CobaJpg3	enc	612520	Sukses	12.44	49237.942
13	CobaBmp1	enc	180072	Sukses	5.202	34615.917
14	CobaBmp2	enc	378072	Sukses	8.727	43322.104
15	CobaBmp3	enc	648072	Sukses	12.454	52037.257
16	CobaPsd1	enc	102600	Sukses	3.325	30857.143
17	CobaPsd2	enc	291928	Sukses	7.426	39311.608
18	CobaPsd3	enc	522592	Sukses	10.656	49042.042

5.2 Analisa Hasil

Dari hasil uji coba tersebut di atas dapat dilakukan analisa sebagai berikut:

1) Program enkripsi dan dekripsi *file* dengan metode IDEA ini ternyata terbukti bekerja sesuai dengan algoritma yang ada.
2) Program enkripsi *file* dengan metode IDEA dapat berfungsi dengan baik untuk berkas dokumen bertipe txt, doc, dan rtf. Demikian pula dapat berfungsi dengan baik untuk melakukan dekripsi.
3) Program enkripsi *file* dengan metode IDEA dapat berfungsi dengan baik untuk berkas grafis bertipe jpg, bmp, dan psd. Demikian pula dapat berfungsi dengan baik untuk melakukan dekripsi.
4) Bila dilihat dari perbandingan antara ukuran berkas dan waktu yang dibutuhkan untuk proses enkripsi tampak bahwa terdapat hubungan yang linier antara ukuran berkas dengan waktu prosesnya, yaitu semakin besar ukuran berkas semakin besar waktu prosesnya. Hal serupa juga terjadi pada proses dekripsi.

BAB VI

KESIMPULAN DAN SARAN

6.1 Kesimpulan

Kesimpulan dari pembuatan perangkat-lunak sistem enkripsi-dekripsi berkas ini adalah:

1) Algoritma IDEA 64 bit mampu mengenkripsi dan mendekripsi *file* dokumen maupun grafis dengan benar. Hal ini ditunjukkan dari hasil uji coba sesuai dengan teori perhitungan secara manual.
2) Program Enkripsi *File* menggunakan IDEA 64 bit dapat berfungsi dengan baik untuk mengenkripsi dan mendekripsi berkas dokumen maupun berkas grafis. Ini menunjukkan bahwa algoritma IDEA 64 bit dapat bekerja pada semua jenis berkas dokumen maupun berkas grafis.
3) Terdapat hubungan yang linier antara ukuran berkas dengan waktu proses enkripsi dan dekripsi.
4) *Windows* API adalah piranti pengembang yang tepat untuk mengimplementasikan pembuatan program enkripsi *file*. *Windows* API yang dipakai pada aplikasi ini adalah fungsi-fungsi yang berhubungan dengan operasi *file* dan operasi *registry*.

6.2 Saran

Pembuatan program ini masih dapat dikembangkan lagi sehingga dapat mengatasi beberapa kelemahan dari program ini dan bahkan juga dapat digunakan untuk program aplikasi yang lebih besar. Beberapa pengembangan yang dapat dilakukan adalah:

1) Pada proses dekripsi, program dapat mengetahui kunci yang dimasukkan sewaktu melakukan enkripsi sehingga program dapat membatalkan proses dekripsi jika kunci tidak sesuai.
2) Program dapat mengatasi keadaan yang terjadi sewaktu pengguna lupa akan kunci yang pernah dimasukkannya.
3) Program dapat melakukan enkripsi-dekripsi untuk beberapa berkas sekaligus yang kemudian dapat disimpan dalam satu berkas.
4) Program dapat memproteksi berkas yang telah dienkripsi sehingga tidak dapat dihapus maupun diedit isinya.

DAFTAR PUSTAKA

Amperiyanto, T, 2001, "Bermain-main dengan Registry Windows", PT. Elex Media Komputindo, Jakarta.

Hadi, R, 2001, "Pemrograman Windows API dengan Microsoft Visual Basic", PT. Elex Media Komputindo, Jakarta.

Hakim, L, 2003, "Pemrograman Game dengan Visual Basic", Andi Offset, Yogyakarta.

Halvorson, M, 2000, "Step By Step Microsoft Visual Basic 6.0 Professional", PT. Elex Media Komputindo, Jakarta.

Kadir, A, 1995, "Pemrograman C++ Membahas Pemrograman Berorientasi Objek Menggunakan Turbo C++ dan Borland C++", Andi Offset, Yogyakarta.

Menezes, A, van Oorschot, P, Vanstone, S, 1997, "Handbook of Applied Cryptography", CRC Press, Inc.

Schneier, B, 1994, "Applied Cryptography", John Wiley & Sons, Inc., Canada.

Wahana Komputer, Semarang, 2003, "Memahami Model Enkripsi & Security Data", Andi Offset, Yogyakarta.

Yuswanto, 2003, "Pemrograman Dasar Microsoft Visual Basic 6.0", Prestasi Pustaka, Surabaya.

http://www.ussrback.com/crypto/bruce_schneier/applied-crypto/idea22a.zip.html.

LAMPIRAN

ENKRIPSI

Masukan
Kunci = 1234
Data = Dimana

Inisialisasi Kunci

- Konversi kunci ke ASCII

1	49
2	50
3	51
4	52

- Tentukan 8 sub-blok kunci pertama dari masukan kunci
  Jika sub-blok kunci ke-x adalah Z_x, maka:

  Z_1 = 49 << 8 = 12544
  Z_2 = (50<<8) or 49 = 12849
  Z_3 = (51<<8) or 50 = 13106
  Z_4 = (52<<8) or 51 = 13363
  $Z_5 = Z_6 = Z_7 = Z_8 = 0$

- Rotasi 25 bit ke kiri 8 sub-blok kunci pertama untuk mencari 8 sub-blok kunci kedua

  Z_9 = (Z_2 << 9) or (Z_3 >> 7)
  = 25088 or 102
  = 25190
  Z_{10} = (Z_3 << 9) or (Z_4 >> 7)
  = 25600 or 104
  = 25704
  Z_{11} = (Z_4 << 9) or (Z_5 >> 7)
  = 26112 or 0
  = 26112
  Z_{12} = (Z_5 << 9) or (Z_6 >> 7)
  = 0 or 0
  = 0
  Z_{13} = (Z_6 << 9) or (Z_7 >> 7)
  = 0 or 0
  = 0
  Z_{14} = (Z_7 << 9) or (Z_8 >> 7)
  = 0 or 0
  = 0
  Z_{15} = (Z_8 << 9) or (Z_1 >> 7)
  = 0 or 98
  = 98
  Z_{16} = (Z_1 << 9) or (Z_2 >> 7)
  = 0 or 100
  = 100

- Rotasi 25 bit ke kiri 8 sub-blok kunci kedua untuk mencari 8 sub-blok kunci ketiga

  Z_{17} = (Z_{10} << 9) or (Z_{11} >> 7)
  = 53248 or 204
  = 53452
  Z_{18} = (Z_{11} << 9) or (Z_{12} >> 7)
  = 0 or 0
  = 0
  Z_{19} = (Z_{12} << 9) or (Z_{13} >> 7)
  = 0 or 0
  = 0
  Z_{20} = (Z_{13} << 9) or (Z_{14} >> 7)
  = 0 or 0
  = 0
  Z_{21} = (Z_{14} << 9) or (Z_{15} >> 7)
  = 0 or 0
  = 0
  Z_{22} = (Z_{15} << 9) or (Z_{16} >> 7)
  = 50176 or 0
  = 50176
  Z_{23} = (Z_{16} << 9) or (Z_9 >> 7)
  = 51200 or 196
  = 51396
  Z_{24} = (Z_9 << 9) or (Z_{10} >> 7)
  = 52224 or 200
  = 52424

- Rotasi 25 bit ke kiri 8 sub-blok kunci kedua untuk mencari 8 sub-blok kunci keempat

  Z_{25} = (Z_{18} << 9) or (Z_{19} >> 7)
  = 0 or 0
  = 0
  Z_{26} = (Z_{19} << 9) or (Z_{20} >> 7)
  = 0 or 0
  = 0
  Z_{27} = (Z_{20} << 9) or (Z_{21} >> 7)
  = 0 or 0
  = 0
  Z_{28} = (Z_{21} << 9) or (Z_{22} >> 7)
  = 0 or 392
  = 392
  Z_{29} = (Z_{22} << 9) or (Z_{23} >> 7)
  = 0 or 401
  = 401
  Z_{30} = (Z_{23} << 9) or (Z_{24} >> 7)
  = 34816 or 409
  = 35225
  Z_{31} = (Z_{24} << 9) or (Z_{17} >> 7)
  = 36864 or 417
  = 37281
  Z_{32} = (Z_{17} << 9) or (Z_{18} >> 7)
  = 38912 or 0
  = 38912

- Rotasi 25 bit ke kiri 8 sub-blok kunci keempat untuk mencari 8 sub-blok kunci kelima

Z_{33} = (Z_{26} << 9) or (Z_{27} >> 7)
= 0 or 0
= 0
Z_{34} = (Z_{27} << 9) or (Z_{28} >> 7)
= 0 or 3
= 3
Z_{35} = (Z_{28} << 9) or (Z_{29} >> 7)
= 4096 or 3
= 4099
Z_{36} = (Z_{29} << 9) or (Z_{30} >> 7)
= 8704 or 275
= 8979
Z_{37} = (Z_{30} << 9) or (Z_{31} >> 7)
= 12800 or 291
= 13091
Z_{38} = (Z_{31} << 9) or (Z_{32} >> 7)
= 16896 or 304
= 17200
Z_{39} = (Z_{32} << 9) or (Z_{25} >> 7)
= 0 or 0
= 0
Z_{40} = (Z_{25} << 9) or (Z_{26} >> 7)
= 0 or 0
= 0

- Rotasi 25 bit ke kiri 8 sub-blok kunci kelima untuk mencari 8 sub-blok kunci keenam

Z_{41} = (Z_{34} << 9) or (Z_{35} >> 7)
= 1536 or 32
= 1568
Z_{42} = (Z_{35} << 9) or (Z_{36} >> 7)
= 1536 or 70
= 1606
Z_{43} = (Z_{36} << 9) or (Z_{37} >> 7)
= 9728 or 102
= 9830
Z_{44} = (Z_{37} << 9) or (Z_{38} >> 7)
= 17920 or 134
= 18054
Z_{45} = (Z_{38} << 9) or (Z_{39} >> 7)
= 24576 or 0
= 24576
Z_{46} = (Z_{39} << 9) or (Z_{40} >> 7)
= 0 or 0
= 0
Z_{47} = (Z_{40} << 9) or (Z_{33} >> 7)
= 0 or 0
= 0
Z_{48} = (Z_{33} << 9) or (Z_{34} >> 7)
= 0 or 0
= 0

- Rotasi 25 bit ke kiri 8 sub-blok kunci kelima untuk mencari 4 sub-blok kunci terakhir

Z_{49} = (Z_{42} << 9) or (Z_{43} >> 7)
= 35840 or 76
= 35916
Z_{50} = (Z_{43} << 9) or (Z_{44} >> 7)
= 52224 or 141
= 52365
Z_{51} = (Z_{44} << 9) or (Z_{45} >> 7)
= 3264
Z_{52} = (Z_{45} << 9) or (Z_{46} >> 7)
= 0 or 0
= 0

Baca data

- Konversi data ke ASCII

D	68
i	105
m	109
a	97
n	110
a	97

- Baca 1 byte data masukkan dalam blok data
  Blok data = X1, X2, X3, X4
  X_1 = (68 << 8) or 105 = 17513
  X_2 = (109 << 8) or 97 = 28001
  X_3 = (110 << 8) or 97 = 28257
  X_4 = 0

Proses

Round1:
$X_1 = X_1 * Z_1 = 17513 * 12544 = 3048$
$X_2 = X_2 + Z_2 = 28001 + 12849 = 40850$
$X_3 = X_3 + Z_3 = 28257 + 13106 = 41363$
$X_4 = X_4 * Z_4 = 0 * 13363 = 52174$
$T_1 = X_1 \text{ xor } X_3 = 3048 \text{ xor } 41363 = 43643$
$T_2 = X_2 \text{ xor } X_4 = 40850 \text{ xor } 52174 = 21596$
$T_1 = T_1 * Z_5 = 43643 * 0 = 21849$
$T_2 = T_2 + T_1 = 21596 + 21894 = 43490$
$T_2 = T_2 * Z_6 = 43490 * 0 = 22047$
$T_1 = T_1 + T_2 = 21894 + 22047 = 43941$
$X_1 = X_1 \text{ xor } T_2 = 3048 \text{ xor } 22047 = 24055$
$X_4 = X_4 \text{ xor } T_1 = 52174 \text{ xor } 43941 = 24683$
$T_1 = T_1 \text{ xor } X_2 = 43941 \text{ xor } 40850 = 13367$
$X_2 = X_3 \text{ xor } T_2 = 41363 \text{ xor } 22047 = 63372$
$X_3 = T_1 = 13367$

Round2:
$X_1 = X_1 * Z_7 = 24055 * 0 = 41482$
$X_2 = X_2 + Z_8 = 63372 + 0 = 63372$
$X_3 = X_3 + Z_9 = 13367 + 25190 = 38557$
$X_4 = X_4 * Z_{10} = 24683 * 25704 = 53672$
$T_1 = X_1 \text{ xor } X_3 = 41482 \text{ xor } 38557 = 13463$
$T_2 = X_2 \text{ xor } X_4 = 63372 \text{ xor } 53672 = 9764$
$T_1 = T_1 * Z_{11} = 13463 * 26112 = 5388$
$T_2 = T_2 + T_1 = 9764 + 5388 = 15152$
$T_2 = T_2 * Z_{12} = 15152 * 0 = 50385$
$T_1 = T_1 + T_2 = 5388 + 50385 = 55773$
$X_1 = X_1 \text{ xor } T_2 = 41482 \text{ xor } 50385 = 26331$
$X_4 = X_4 \text{ xor } T_1 = 53672 \text{ xor } 55773 = 2165$
$T_1 = T_1 \text{ xor } X_2 = 55773 \text{ xor } 63372 = 11857$
$X_2 = X_3 \text{ xor } T_2 = 38557 \text{ xor } 50385 = 21068$
$X_3 = T_1 = 11857$

Round3:
$X_1 = X_1 * Z_{13} = 26331 * 0 = 39206$
$X_2 = X_2 + Z_{14} = 21068 + 0 = 21068$
$X_3 = X_3 + Z_{15} = 11857 + 98 = 11955$
$X_4 = X_4 * Z_{16} = 2165 * 100 = 19889$
$T_1 = X_1 \text{ xor } X_3 = 39206 \text{ xor } 11955 = 46997$
$T_2 = X_2 \text{ xor } X_4 = 21068 \text{ xor } 19889 = 8189$
$T_1 = T_1 * Z_{17} = 46997 * 53452 = 50434$
$T_2 = T_2 + T_1 = 8189 + 50434 = 58623$
$T_2 = T_2 * Z_{18} = 58623 * 0 = 6914$
$T_1 = T_1 + T_2 = 50434 + 6914 = 57348$
$X_1 = X_1 \text{ xor } T_2 = 39206 \text{ xor } 6914 = 33316$
$X_4 = X_4 \text{ xor } T_1 = 19889 \text{ xor } 57348 = 44469$
$T_1 = T_1 \text{ xor } X_2 = 57348 \text{ xor } 21068 = 45640$
$X_2 = X_3 \text{ xor } T_2 = 11955 \text{ xor } 6914 = 13745$
$X_3 = T_1 = 45640$

Round4:
$X_1 = X_1 * Z_{19} = 33316 * 0 = 32221$
$X_2 = X_2 + Z_{20} = 13745 + 0 = 13745$
$X_3 = X_3 + Z_{21} = 45640 + 0 = 45640$
$X_4 = X_4 * Z_{22} = 44469 * 50176 = 3842$
$T_1 = X_1 \text{ xor } X_3 = 32221 \text{ xor } 45640 = 53141$
$T_2 = X_2 \text{ xor } X_4 = 13745 \text{ xor } 3842 = 15027$
$T_1 = T_1 * Z_{23} = 53141 * 51396 = 45898$
$T_2 = T_2 + T_1 = 15027 + 45898 = 60925$
$T_2 = T_2 * Z_{24} = 60925 * 52424 = 52042$
$T_1 = T_1 + T_2 = 45898 + 52042 = 32404$
$X_1 = X_1 \text{ xor } T_2 = 32221 \text{ xor } 52042 = 46743$
$X_4 = X_4 \text{ xor } T_1 = 3842 \text{ xor } 32404 = 29078$
$T_1 = T_1 \text{ xor } X_2 = 32404 \text{ xor } 13745 = 19237$
$X_2 = X_3 \text{ xor } T_2 = 45640 \text{ xor } 52042 = 30978$
$X_3 = T_1 = 19237$

Round5:
$X_1 = X_1 * Z_{25} = 46743 * 0 = 18749$
$X_2 = X_2 + Z_{26} = 30978 + 0 = 30978$
$X_3 = X_3 + Z_{27} = 19237 + 0 = 19237$
$X_4 = X_4 * Z_{28} = 29078 * 392 = 60675$
$T_1 = X_1 \text{ xor } X_3 = 18794 \text{ xor } 19237 = 591$
$T_2 = X_2 \text{ xor } X_4 = 30978 \text{ xor } 60675 = 37889$
$T_1 = T_1 * Z_{29} = 591 * 401 = 40380$
$T_2 = T_2 + T_1 = 37889 + 40380 = 12733$
$T_2 = T_2 * Z_{30} = 12733 * 35225 = 50234$
$T_1 = T_1 + T_2 = 40380 + 50234 = 25078$
$X_1 = X_1 \text{ xor } T_2 = 18794 \text{ xor } 50234 = 36176$
$X_4 = X_4 \text{ xor } T_1 = 60675 \text{ xor } 25078 = 36085$
$T_1 = T_1 \text{ xor } X_2 = 25078 \text{ xor } 30978 = 6388$
$X_2 = X_3 \text{ xor } T_2 = 19237 \text{ xor } 50234 = 36639$
$X_3 = T_1 = 6388$

Round6:
$X_1 = X_1 * Z_{31} = 36176 * 37281 = 57070$
$X_2 = X_2 + Z_{32} = 36639 + 38912 = 10015$
$X_3 = X_3 + Z_{33} = 6388 + 0 = 6388$
$X_4 = X_4 * Z_{34} = 36085 * 3 = 42718$
$T_1 = X_1 \text{ xor } X_3 = 57070 \text{ xor } 6388 = 50714$
$T_2 = X_2 \text{ xor } X_4 = 10015 \text{ xor } 42718 = 33217$
$T_1 = T_1 * Z_{35} = 50714 * 4099 = 58859$
$T_2 = T_2 + T_1 = 33217 + 58859 = 26540$
$T_2 = T_2 * Z_{36} = 26540 * 8979 = 10128$
$T_1 = T_1 + T_2 = 58859 + 10128 = 3451$
$X_1 = X_1 \text{ xor } T_2 = 57070 \text{ xor } 10128 = 63870$
$X_4 = X_4 \text{ xor } T_1 = 42718 \text{ xor } 3451 = 43941$

$T_1 = T_1$ xor X_2 = 3451 xor 10015 = 10582
$X_2 = X_3$ xor T_2 = 6388 xor 10128 = 16228
$X_3 = T_1$ = 10852

Round7:
$X_1 = X_1 * Z_{37}$ = 63870 * 13091 = 1124
$X_2 = X_2 + Z_{38}$ = 16228 + 17200 = 33428
$X_3 = X_3 + Z_{39}$ = 10852 + 0 = 10852
$X_4 = X_4 * Z_{40}$ = 43941 * 0 = 21596
$T_1 = X_1$ xor X_3 = 1124 xor 10852 = 11776
$T_2 = X_2$ xor X_4 = 33428 xor 21596 = 54984
$T_1 = T_1 * Z_{41}$ = 11776 * 1568 = 48871
$T_2 = T_2 + T_1$ = 54984 + 48871 = 38319
$T_2 = T_2 * Z_{42}$ = 38319 * 1606 = 1071
$T_1 = T_1 + T_2$ = 1071 + 48871 = 49942
$X_1 = X_1$ xor T_2 = 1124 xor 1071 = 75
$X_4 = X_4$ xor T_1 = 21596 xor 49942 = 38730
$T_1 = T_1$ xor X_2 = 49942 xor 33428 = 16770
$X_2 = X_3$ xor T_2 = 10852 xor 1071 = 11851
$X_3 = T_1$ = 16770

Round8:
$X_1 = X_1 * Z_{43}$ = 75 * 9830 = 16343
$X_2 = X_2 + Z_{44}$ = 11851 + 18054 = 29905
$X_3 = X_3 + Z_{45}$ = 16770 + 24576 = 41346
$X_4 = X_4 * Z_{46}$ = 38730 * 0 = 26807
$T_1 = X_1$ xor X_3 = 16343 xor 41346 = 40533
$T_2 = X_2$ xor X_4 = 29905 xor 26807 = 7270
$T_1 = T_1 * Z_{47}$ = 40533 * 0 = 25004
$T_2 = T_2 + T_1$ = 7270 + 25004 = 32274
$T_2 = T_2 * Z_{48}$ = 32274 * 0 = 33263
$T_1 = T_1 + T_2$ = 25004 + 33263 = 58267
$X_1 = X_1$ xor T_2 = 16343 xor 33263 = 48696
$X_4 = X_4$ xor T_1 = 26807 xor 58267 = 35628
$X_2 = T_1$ xor X_2 = 58267 xor 29905 = 38730
$X_3 = X_3$ xor T_2 = 41346 xor 33263 = 8301

Hasil Enkripsi:
$X_1 = X_1 * Z_{49}$ = 48696 * 35916 = 45154
$X_2 = X_2 + Z_{50}$ = 38730 + 52365 = 25559
$X_3 = X_3 + Z_{51}$ = 8301 + 3264 = 11565
$X_4 = X_4 * Z_{52}$ = 35628 * 0 = 29909

Tulis Data
X_1 = 45154

Nilai *byte* pertama = X_1 >> 8 = 45154>>8 = 176

Nilai *byte* kedua = X_1 AND 255 = 45154 AND 255 = 98

X_2 = 25559

Nilai *byte* ketiga = X_2 >> 8 = 25559>>8 = 99

Nilai *byte* keempat = X_2 AND 255 = 25559 AND 255 = 215

X_3 = 11565

Nilai *byte* kelima = X_3 >> 8 = 11565>>8 = 45

Nilai *byte* keenam = X_3 AND 255 = 11565 AND 255 = 45

X_4 = 29909

Nilai *byte* ketujuh = X_4 >> 8 = 29909>>8 = 116

Nilai *byte* kedelapan = X_4 AND 255 = 29909 AND 255 = 213

Tampilan dalam ASCII:

176	▒
98	b
99	c
215	╫
45	-
45	-
116	t
213	╒

DEKRIPSI

Masukan
Kunci = 1234
Data = ▒bc╫-t╒

Inisialisasi kunci
Kunci enkripsi diinverskan sesuai dengan Tabel2.1.
Jika sub-blok kunci dekripsi ke-x adalah P_x, maka:

$P_1 = inv(Z_{49}) = inv(35916) = 64017$
$P_2 = -Z_{50} = -(52365) = 13171$
$P_3 = -Z_{51} = -(3264) = 62272$
$P_4 = inv(Z_{52}) = inv(0) = 0$
$P_5 = Z_{47} = 0$
$P_6 = Z_{48} = 0$
$P_7 = inv(Z_{43}) = inv(9830) = 11914$
$P_8 = -Z_{45} = -(24576) = 40960$
$P_9 = -Z_{44} = -(18054) = 47482$
$P_{10} = inv(Z_{46}) = inv(0) = 0$
$P_{11} = Z_{41} = 1568$
$P_{12} = Z_{42} = 1606$
$P_{13} = inv(Z_{37}) = inv(13091) = 51950$
$P_{14} = -Z_{39} = -(0) = 0$
$P_{15} = -Z_{38} = -(17200) = 48336$
$P_{16} = inv(Z_{40}) = inv(0) = 0$
$P_{17} = Z_{35} = 4099$
$P_{18} = Z_{36} = 8979$
$P_{19} = inv(Z_{31}) = inv(37281) = 23085$
$P_{20} = -Z_{33} = -(0) = 0$
$P_{21} = -Z_{32} = -(38912) = 26624$
$P_{22} = inv(Z_{34}) = inv(3) = 21846$
$P_{23} = Z_{29} = 401$
$P_{24} = Z_{30} = 35225$
$P_{25} = inv(Z_{25}) = inv(0) = 0$
$P_{26} = -Z_{27} = -(0) = 0$
$P_{27} = -Z_{26} = -(0) = 0$
$P_{28} = inv(Z_{28}) = inv(392) = 41295$
$P_{29} = Z_{23} = 51396$
$P_{30} = Z_{24} = 52424$

$P_{31} = inv(Z_{19}) = inv(0) = 0$
$P_{32} = -Z_{21} = -(0) = 0$
$P_{33} = -Z_{20} = -(0) = 0$
$P_{34} = inv(Z_{22}) = inv(50176) = 25411$
$P_{35} = Z_{17} = 53452$
$P_{36} = Z_{18} = 0$
$P_{37} = inv(Z_{13}) = inv(0) = 0$
$P_{38} = -Z_{15} = -(98) = 65438$
$P_{39} = -Z_{14} = -(0) = 0$
$P_{40} = inv(Z_{16}) = inv(100) = 17695$
$P_{41} = Z_{11} = 26112$
$P_{42} = Z_{12} = 0$
$P_{43} = inv(Z_{7}) = inv(0) = 0$
$P_{44} = -Z_{9} = -(25190) = 40346$
$P_{45} = -Z_{8} = -(0) = 0$
$P_{46} = inv(Z_{10}) = inv(25704) = 62740$
$P_{47} = Z_{5} = 0$
$P_{48} = Z_{6} = 0$
$P_{49} = inv(Z_{1}) = inv(12544) = 36107$
$P_{50} = -Z_{2} = -(12849) = 52687$
$P_{51} = -Z_{3} = -(13106) = 52430$
$P_{52} = inv(Z_{4}) = inv(13363) = 42717$

Baca Data
Konversi ke ASCII

▒	176
b	98
C	99
╪	215
-	45
-	45
t	116
F	213

$X_1 = (176<<8)\ OR\ 98 = 45154$

$X_2 = (99<<8)\ OR\ 215 = 25559$

$X_3 = (45<<8)\ OR\ 45 = 11565$

$X_4 = (116<<8)\ OR\ 213 = 29909$

Proses

Round1:
$X_1 = X_1 * P_1 = 45154 * 64017 = 48696$
$X_2 = X_2 + P_2 = 25559 + 13171 = 38730$
$X_3 = X_3 + P_3 = 11565 + 62272 = 8301$
$X_4 = X_4 * P_4 = 29909 * 0 = 35628$
$T_1 = X_1 \text{ xor } X_3 = 48696 \text{ xor } 8301 = 40533$
$T_2 = X_2 \text{ xor } X_4 = 38730 \text{ xor } 35628 = 7270$
$T_1 = T_1 * P_5 = 40533 * 0 = 25004$
$T_2 = T_2 + T_1 = 7270 + 25004 = 32274$
$T_2 = T_2 * P_6 = 32274 * 0 = 33263$
$T_1 = T_1 + T_2 = 25004 + 33263 = 58267$
$X_1 = X_1 \text{ xor } T_2 = 48696 \text{ xor } 33263 = 16343$
$X_4 = X_4 \text{ xor } T_1 = 35628 \text{ xor } 58267 = 26807$
$T_1 = T_1 \text{ xor } X_2 = 58267 \text{ xor } 38730 = 29905$
$X_2 = X_3 \text{ xor } T_2 = 8301 \text{ xor } 33263 = 41346$
$X_3 = T_1 = 29905$

Round2:
$X_1 = X_1 * P_7 = 16343 * 11914 = 75$
$X_2 = X_2 + P_8 = 41346 + 40960 = 16770$
$X_3 = X_3 + P_9 = 29905 + 47482 = 11851$
$X_4 = X_4 * P_{10} = 26807 * 0 = 38730$
$T_1 = X_1 \text{ xor } X_3 = 75 \text{ xor } 11851 = 11776$
$T_2 = X_2 \text{ xor } X_4 = 16770 \text{ xor } 38730 = 54984$
$T_1 = T_1 * P_{11} = 11776 * 1568 = 48871$
$T_2 = T_2 + T_1 = 54984 + 48871 = 38319$
$T_2 = T_2 * P_{12} = 38319 * 1606 = 1071$
$T_1 = T_1 + T_2 = 48871 + 1071 = 49942$
$X_1 = X_1 \text{ xor } T_2 = 75 \text{ xor } 1071 = 1124$
$X_4 = X_4 \text{ xor } T_1 = 38730 \text{ xor } 49942 = 21596$
$T_1 = T_1 \text{ xor } X_2 = 49942 \text{ xor } 16770 = 33428$
$X_2 = X_3 \text{ xor } T_2 = 11851 \text{ xor } 1071 = 10852$
$X_3 = T_1 = 33428$

Round3:
$X_1 = X_1 * P_{13} = 1124 * 51950 = 63870$
$X_2 = X_2 + P_{14} = 10852 + 0 = 10852$
$X_3 = X_3 + P_{15} = 33428 + 48336 = 16228$
$X_4 = X_4 * P_{16} = 21596 * 0 = 43941$
$T_1 = X_1 \text{ xor } X_3 = 63870 \text{ xor } 16228 = 50714$
$T_2 = X_2 \text{ xor } X_4 = 10852 \text{ xor } 43941 = 33217$
$T_1 = T_1 * P_{17} = 50714 * 4099 = 58859$
$T_2 = T_2 + T_1 = 33217 + 58859 = 26540$
$T_2 = T_2 * P_{18} = 26540 * 8979 = 10128$
$T_1 = T_1 + T_2 = 58859 + 10128 = 3451$
$X_1 = X_1 \text{ xor } T_2 = 63870 \text{ xor } 10128 = 57070$
$X_4 = X_4 \text{ xor } T_1 = 43941 \text{ xor } 3451 = 42718$
$T_1 = T_1 \text{ xor } X_2 = 3451 \text{ xor } 10852 = 10015$
$X_2 = X_3 \text{ xor } T_2 = 16228 \text{ xor } 10128 = 6388$
$X_3 = T_1 = 10015$

Round4:
$X_1 = X_1 * P_{19} = 57070 * 23085 = 36176$
$X_2 = X_2 + P_{20} = 6388 + 0 = 6388$
$X_3 = X_3 + P_{21} = 10015 + 26624 = 36639$
$X_4 = X_4 * P_{22} = 42718 * 21846 = 36085$
$T_1 = X_1 \text{ xor } X_3 = 36176 \text{ xor } 36639 = 591$
$T_2 = X_2 \text{ xor } X_4 = 6388 \text{ xor } 36085 = 37889$
$T_1 = T_1 * P_{23} = 591 * 401 = 40380$
$T_2 = T_2 + T_1 = 37889 + 40380 = 12733$
$T_2 = T_2 * P_{24} = 12733 * 35225 = 50234$
$T_1 = T_1 + T_2 = 40380 + 50234 = 25078$
$X_1 = X_1 \text{ xor } T_2 = 36176 \text{ xor } 50234 = 18794$
$X_4 = X_4 \text{ xor } T_1 = 36085 \text{ xor } 25078 = 60675$
$T_1 = T_1 \text{ xor } X_2 = 25078 \text{ xor } 6388 = 30978$
$X_2 = X_3 \text{ xor } T_2 = 36639 \text{ xor } 50234 = 19237$
$X_3 = T_1 = 30978$

Round5:
$X_1 = X_1 * P_{25} = 18794 * 0 = 46743$
$X_2 = X_2 + P_{26} = 19237 + 0 = 19237$
$X_3 = X_3 + P_{27} = 30978 + 0 = 30978$

$X_4 = X_4 * P_{28} = 60675 * 41295 = 29078$
$T_1 = X_1 \text{ xor } X_3 = 46743 \text{ xor } 30978 = 53141$
$T_2 = X_2 \text{ xor } X_4 = 19237 \text{ xor } 29078 = 15027$
$T_1 = T_1 * P_{29} = 53141 * 51396 = 45898$
$T_2 = T_2 + T_1 = 15027 + 45898 = 60925$
$T_2 = T_2 * P_{30} = 60925 * 52424 = 52042$
$T_1 = T_1 + T_2 = 45898 + 52042 = 32404$
$X_1 = X_1 \text{ xor } T_2 = 46743 \text{ xor } 52042 = 32221$
$X_4 = X_4 \text{ xor } T_1 = 29078 \text{ xor } 32404 = 3842$
$T_1 = T_1 \text{ xor } X_2 = 32404 \text{ xor } 19237 = 13745$
$X_2 = X_3 \text{ xor } T_2 = 30978 \text{ xor } 52042 = 45640$
$X_3 = T_1 = 13745$

Round6:
$X_1 = X_1 * P_{31} = 32221 * 0 = 33316$
$X_2 = X_2 + P_{32} = 45640 + 0 = 45640$
$X_3 = X_3 + P_{33} = 13745 + 0 = 13745$
$X_4 = X_4 * P_{34} = 3842 * 25411 = 44469$
$T_1 = X_1 \text{ xor } X_3 = 33316 \text{ xor } 13745 = 46997$
$T_2 = X_2 \text{ xor } X_4 = 45640 \text{ xor } 44469 = 8189$
$T_1 = T_1 * P_{35} = 46997 * 53452 = 50434$
$T_2 = T_2 + T_1 = 8189 + 50434 = 58623$
$T_2 = T_2 * P_{36} = 58623 * 0 = 6914$
$T_1 = T_1 + T_2 = 50434 + 6914 = 57348$
$X_1 = X_1 \text{ xor } T_2 = 33316 \text{ xor } 6914 = 39206$
$X_4 = X_4 \text{ xor } T_1 = 44469 \text{ xor } 57348 = 19889$
$T_1 = T_1 \text{ xor } X_2 = 57348 \text{ xor } 45640 = 21068$
$X_2 = X_3 \text{ xor } T_2 = 13745 \text{ xor } 6914 = 11955$
$X_3 = T_1 = 21068$

Round7:
$X_1 = X_1 * P_{37} = 39206 * 0 = 26331$
$X_2 = X_2 + P_{38} = 11955 + 65438 = 11587$
$X_3 = X_3 + P_{39} = 21068 + 0 = 21068$
$X_4 = X_4 * P_{40} = 19889 * 17695 = 2165$
$T_1 = X_1 \text{ xor } X_3 = 26331 \text{ xor } 21068 = 13463$
$T_2 = X_2 \text{ xor } X_4 = 11857 \text{ xor } 2165 = 9764$
$T_1 = T_1 * P_{41} = 13463 * 26112 = 5388$
$T_2 = T_2 + T_1 = 9764 + 5388 = 15152$
$T_2 = T_2 * P_{42} = 15152 * 0 = 50385$
$T_1 = T_1 + T_2 = 5388 + 50385 = 55773$
$X_1 = X_1 \text{ xor } T_2 = 26331 \text{ xor } 50385 = 41482$
$X_4 = X_4 \text{ xor } T_1 = 2165 \text{ xor } 55773 = 53672$
$T_1 = T_1 \text{ xor } X_2 = 55773 \text{ xor } 11857 = 63372$
$X_2 = X_3 \text{ xor } T_2 = 21068 \text{ xor } 50385 = 38557$
$X_3 = T_1 = 63372$

Round8:
$X_1 = X_1 * P_{43} = 41482 * 0 = 24055$
$X_2 = X_2 + P_{44} = 38557 + 40346 = 13367$
$X_3 = X_3 + P_{45} = 63372 + 0 = 63372$
$X_4 = X_4 * P_{46} = 53672 * 62740 = 24683$
$T_1 = X_1 \text{ xor } X_3 = 24055 \text{ xor } 63372 = 43643$
$T_2 = X_2 \text{ xor } X_4 = 13367 \text{ xor } 24683 = 21596$
$T_1 = T_1 * P_{47} = 43643 * 0 = 21894$
$T_2 = T_2 + T_1 = 21596 + 21894 = 43490$
$T_2 = T_2 * P_{48} = 43490 * 0 = 22047$
$T_1 = T_1 + T_2 = 21849 + 22047 = 43491$
$X_1 = X_1 \text{ xor } T_2 = 24055 \text{ xor } 22047 = 3048$

$X_4 = X_4 \text{ xor } T_1 = 13367 \text{ xor } 43941 = 40850$
$X_2 = T_1 \text{ xor } X_2 = 63372 \text{ xor } 22047 = 41363$
$X_3 = X_3 \text{ xor } T_2 = 24683 \text{ xor } 43941 = 52174$

Hasil Dekripsi:
$X_1 = X_1 * P_{49} = 3048 * 36107 = 17513$
$X_2 = X_2 + P_{50} = 40850 + 52687 = 28001$
$X_3 = X_3 + P_{51} = 41363 + 52430 = 28257$
$X_4 = X_4 * P_{52} = 52174 * 42717 = 0$

Tulis Data

$X_1 = 17513$

Nilai *byte* pertama = $X_1 >> 8 = 17513>>8 = 68$

Nilai *byte* kedua = X_1 AND 255 = 17513 AND 255 = 105

$X_2 = 28001$

Nilai *byte* ketiga = $X_2 >> 8 = 28001>>8 = 109$

Nilai *byte* keempat = X_2 AND 255 = 28001 AND 255 = 97

$X_3 = 28257$

Nilai *byte* kelima = $X_3 >> 8 = 28257>>8 = 110$

Nilai *byte* keenam = X_3 AND 255 = 28257 AND 255 = 97

Tampilan dalam ASCII:

68	D
105	i
109	m
97	a
110	n
97	a