IMPLEMENTASI ENKRIPSI DATA BERBASIS ALGORITMA DATA ENCRYPTION STANDARD (DES) DENGAN BAHASA C

Diajukan untuk memenuhi tugas kurikuler mata kuliah
TEKNOLOGI INFORMASI (EL-517)

Oleh
Dian Tresna Nugraha
13296119

JURUSAN TEKNIK ELEKTRO
INSTITUT TEKNOLOGI BANDUNG
1999

DAFTAR ISI

1 PERKENALAN

DES, singkatan dari Data Encryption Standard, merupakan nama dari sebuah algoritma pengenkrip data (DEA : Data Encryption Algorithm) yang dikeluarkan oleh Federal Information Processing Standard (FIPS) 46 – 1 Amerika Serikat. Algoritma dasarnya sendiri (dikenal dengan nama Lucifer) dikembangkan oleh IBM, NSA, dan NBS yang berperan penting dalam pengembangan bagian akhir algoritmanya.
DEA dan DES telah dipelajari secara ekstensif sejak publikasi pertamanya, dan diketahui sebagai algoritma simetrik yang paling baik dan paling banyak digunakan di dunia.

DES memiliki blok kunci 64-bit dan menggunakan kunci 56-bit selama eksekusi. Pada awalnya didesain untuk implementasi secara hardware. Penggunaan dalam sistem komunikasi mengharuskan pengirim dan penerima memiliki kunci rahasia yang sama, yang dapat digunakan untuk mengenkrip dan mendekrip data yang dikirim/diterima. DES juga dapat digunakan untuk enkripsi data-data pribadi dalam harddisk. Namun, luasnya pemakaian DES belum mencakup lingkungan multiuser. Pada kondisi ini public-key cryptography lebih sesuai untuk digunakan.

NIST Amerika sendiri mensertifikasi ulang DES setiap lima tahun. DES terakhir kali disertifikasi ulang pada 1993. Kini NIST tidak lagi mensertifikasi DES disebabkan banyaknya kelemahan DES dan adanya pengembangan algoritma baru, yaitu Advanced Encryption Standard (AES). Di masa yang akan datang, diharapkan AES dapat mengembangkan DES.

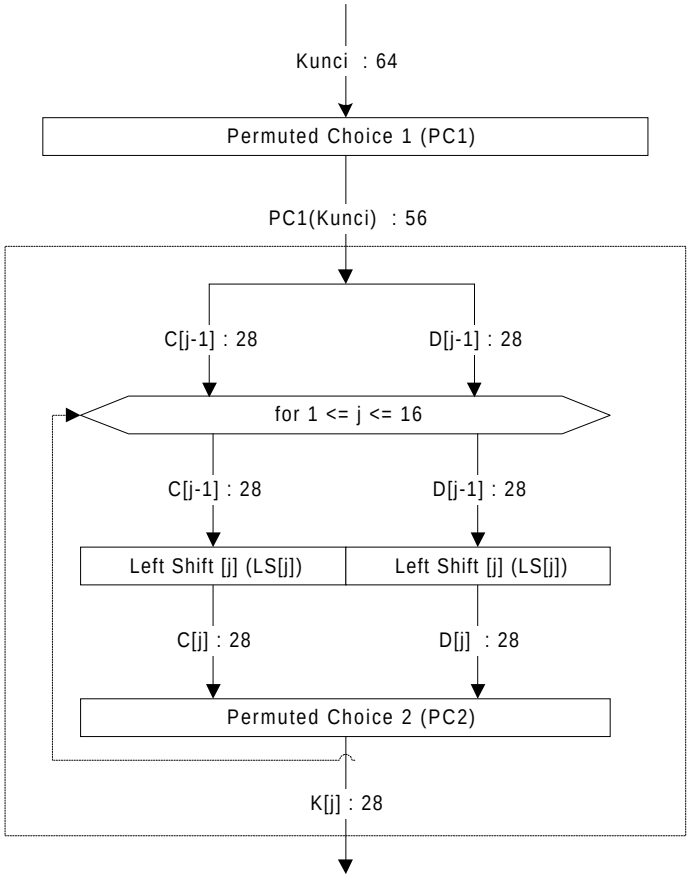
2 ALGORITMA UTAMA DES

Algoritma utama DES terbagi menjadi kelompok – kelompok :

1. Pemrosesan Kunci
2. Enkripsi data 64-bit; dan
3. Dekripsi data 64-bit.

Secara umum, memenuhi diagram blok dan persamaan berikut :

Pemrosesan kunci :



Gambar 1 Diagram Blok Pemrosesan Kunci

C[0]D[0] = PC1(kunci)

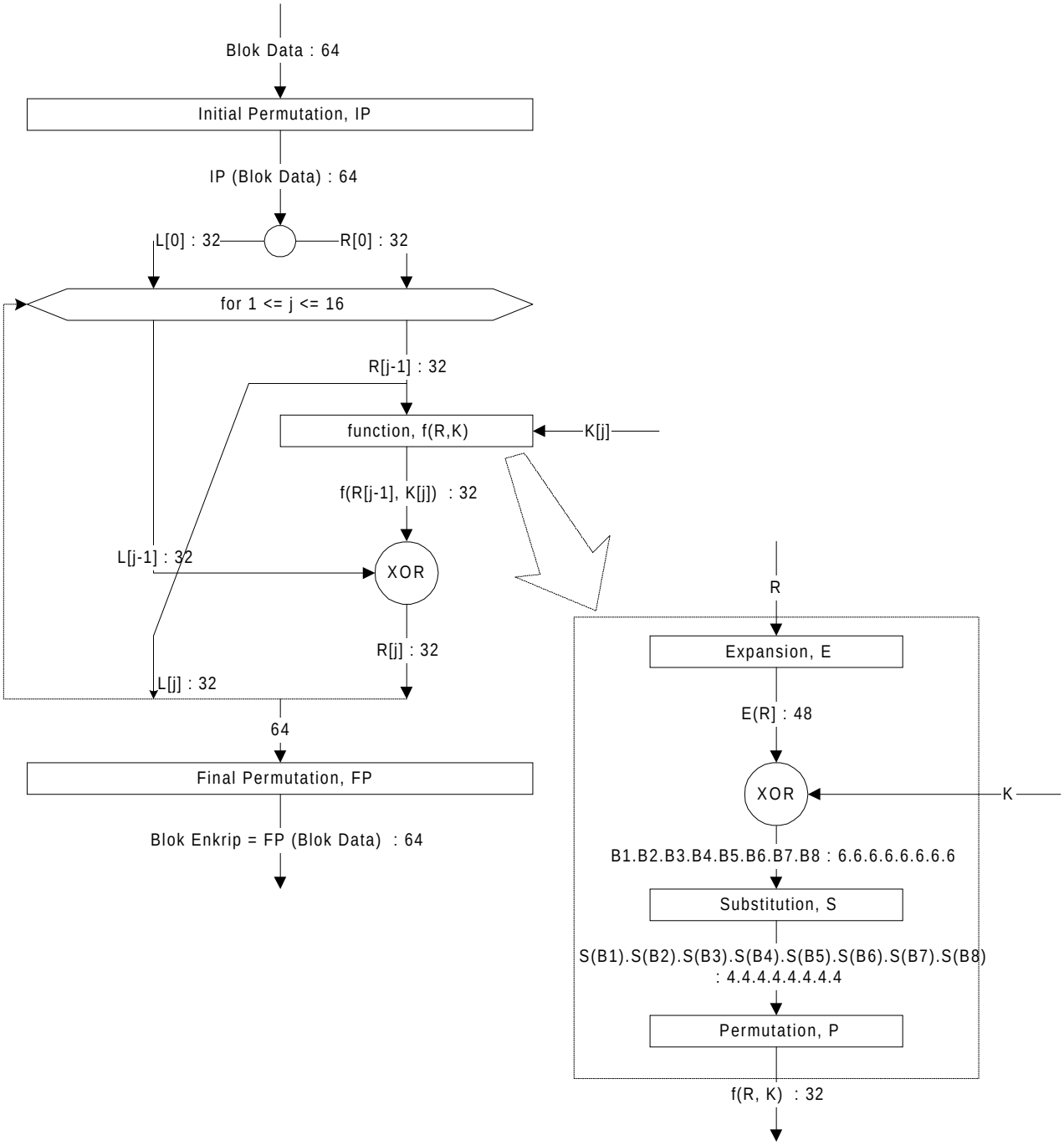
for 1 <= i <= 16

C[i] = LS[i](C[i-1])

D[i] = LS[i](D[i-1])

K[i] = PC2(C[i]D[i])

Enkripsi:



Gambar 2 Diagram Blok Proses Enkripsi

L[0]R[0] = IP(blok data)

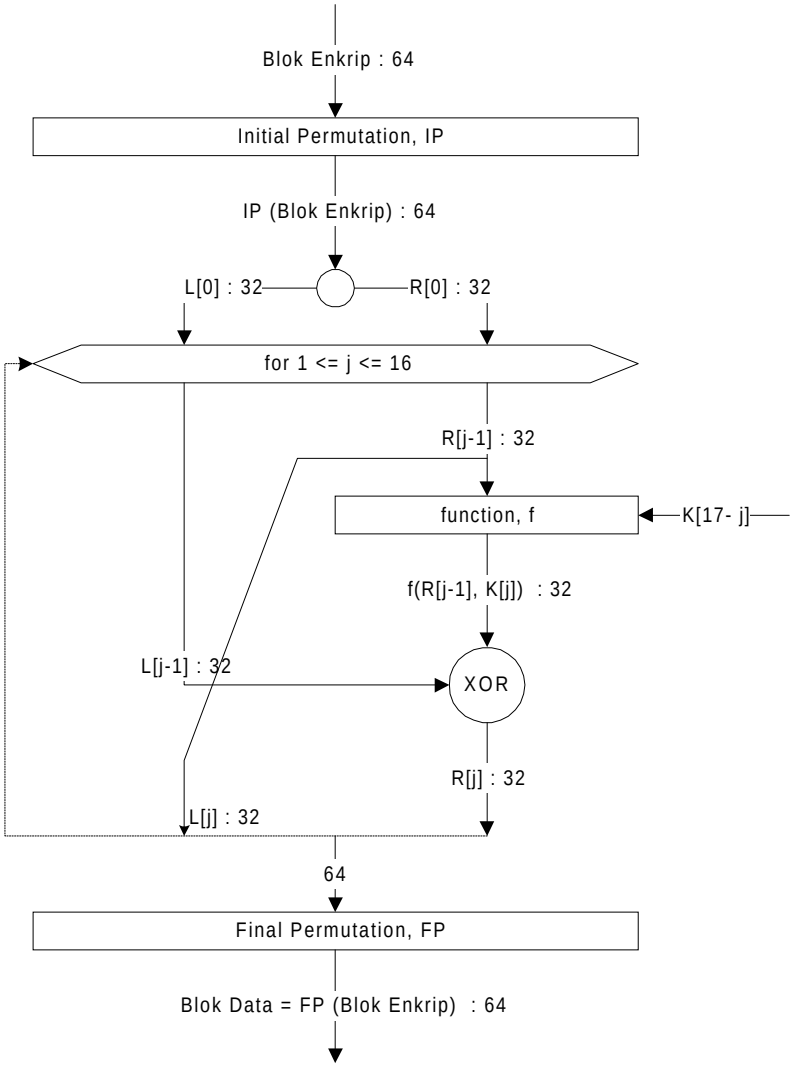
for 1 <= i <= 16

L[i] = R[i-1]

R[i] = L[i-1] xor f(R[i-1], K[i])

blok enkrip = FP(R[16]L[16])

Dekripsi:



Gambar 3 Diagram Blok Proses Dekripsi

R[16]L[16] = IP(blok enkrip)

for 1 <= i <= 16

R[i-1] = L[i]

L[i-1] = R[i] xor f(L[i], K[i])

blok data= FP(L[0]R[0])

2.1 PEMROSESAN KUNCI

1. Meminta sebuah kunci 64-bit (8 karakter) dari pengguna. Setiap bit ke 8 digunakan sebagai bit parity.
2. Penjadwalan kunci rahasia (secret key – scheduling) dimaksudkan untuk menyusun 16 buah kunci yang akan dimasukkan pada setiap iterasi DES, baik pada enkripsi maupun dekripsi

- Permutasi dilakukan pada kunci 64-bit. Pada tahapan ini, bit-bit parity tidak dilibatkan, sehingga bit kunci tereduksi menjadi 56-bit. Bit 1 pada kunci 56 merupakan bit 57 kunci awalnya, bit 2 adalah bit 49, dan seterusnya hingga bit 56 adalah bit 4 kunci 64. Tabel permutasi berikut menunjukkan posisi bit hasil permutasi yang dikenal dengan nama Permuted Choice 1 (PC-1)

Permuted Choice 1 (PC-1)

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

- Output PC-1 kemudian dibagi menjadi dua bagian. 28-bit pertama disebut C[0] dan 28-bit terakhir disebut D[0].
- Dari C[0] dan D[0] kemudian dihitung sub-sub kunci untuk setiap iterasi, yang dimulai dengan j = 1.
- Untuk setiap j, rotasi ke kiri sekali atau dua kali dijalankan pada C[j – 1] dan D[j –1] untuk mendapatkan C[j] dan D[j]. Tabel berikut menunjukkan step rotasi yang dilakukan pada setiap iterasi.

Iterasi ke -	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Jumlah step	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

- Pada setiap hasil C[j]D[j], kunci untuk iterasi ke-j didapat dengan melakukan permutasi kembali pada C[j]D[j]. Permutasi tersebut dikenal dengan nama Permuted Choice 2 (PC-2) dan ditunjukkan pada tabel berikut

Permuted Choice 2 (PC-2)

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

- Iterasi dilakukan terus hingga ke-16 kunci berhasil disusun.

2.2 ENKRIPSI DATA 64-BIT

1. Ambil blok data 64-bit. Apabila blok data kurang dari 64-bit, maka penambahan harus dilakukan agar memadai untuk penggunaan.

2. Permutasi awal (Initial Permutation) dilakukan pada blok data tersebut dengan mengacu pada tabel berikut

Initial Permutation (IP)

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

3. Blok data dibagi menjadi dua bagian. 32-bit pertama disebut L[0] dan 32-bit kedua disebut R[0]
4. Ke-16 sub kunci dioperasikan dengan blok data, dimulai dengan j = 1, dengan cara :

4.1 R[j – 1] dikembangkan menjadi 48-bit menurut fungsi pemilihan ekspansi berikut

Expansion (E)

32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

4.2 E(R[j – 1]) di-XOR dengan K[j]

4.3 Hasil E(R[j – 1]) XOR K[j] dipecah menjadi delapan blok 6-bit. Kelompok bit 1-6 disebut B[1], bit 7-12 disebut B[2], dan seterusnya bit 43-48 disebut B[8].

4.4 Jumlah bit dikurangi dengan penukaran nilai-nilai yang ada dalam tabel S untuk setiap B[j]. Dimulai dengan j = 1, setiap nilai dalam tabel S memiliki 4 bit.

4.4.1 Ambil bit ke 1 dan ke 6 dari B[j] bersama-sama menjadi nilai 2 bit, misalkan m, yang menunjukkan baris dalam tabel S[j] .

4.4.2 Ambil bit ke 2 hingga 5 dari B[j] sebagai nilai 4 bit, misalkan n, yang menunjukkan kolom dalam S[j].

4.4.3 Hasil proses ini adalah S[j][m][n] untuk setiap B[j] sehingga iterasi yang diperlukan sebanyak 8 kali.

Substitution Box 1 (S[1])

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S[2]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S[3]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S[4]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S[5]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S[6]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S[7]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S[8]

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

4.5 Permutasi dilakukan kembali pada gabungan hasil substitusi di atas S[1][m1][n1] hingga S[8][m2][n2] dengan menggunakan tabel berikut.

Permutation P

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

4.6 Hasil permutasi kemudian di XOR dengan L[j-1], selanjutnya hasil ini menjadi R[j]

$$R[i] = L[i-1] \text{ XOR } P(S[1](B[1])...S[8](B[8]))$$

, yang mana B[j] merupakan blok 6-bit hasil E(R[i-1]) XOR K[i]. Fungsi ini biasa ditulis pula sebagai

$$R[i] = L[i-1] \text{ XOR } f(R[i-1], K[i]).)$$

4.7 L[i] = R[i-1].

4.8 Loop kembali ke 4.1 hingga K[16].

5. Permutasi akhir dilakukan kembali dengan tabel permutasi yang merupakan invers dari permutasi awal. Tabel ini disebut tabel permutasi akhir atau tabel inver permutasi awal (IPinverse)

Final Permutation (IP**-1)

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26

33 1 41 9 49 17 57 25

2.3 DEKRIPSI DATA 64-BIT

Pada bagian sebelumnya (2.2) telah diuraikan algoritma DES dalam mengenkrip satu blok data 64-bit. Untuk dekripsi, proses yang sama dilakukan kembali, hanya saja digunakan kunci K[j] dalam urutan yang berlawanan, yaitu memasukkan K[16] terlebih dahulu, kemudian K[16], seterusnya hingga K[1].

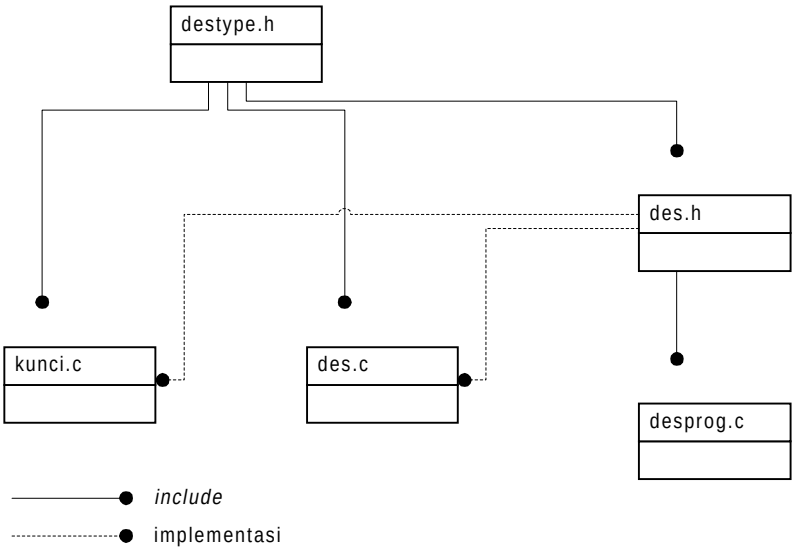
3 IMPLEMENTASI DES

3.1 STRUKTUR FILE SUMBER PROGRAM

Implementasi dengan bahasa C kali ini dilakukan pada Borland C++ 3.1 DOS. Terdapat file yang terpisah, yaitu :

- destype.h : mendefinisikan tipe-tipe data yang akan digunakan
- des.h : mendefinisikan prototipe-prototipe untuk integrasi fungsi yang ada dalam file yang berbeda
- kunci.c : mengimplementasikan fungsi-fungsi pemroses kunci
- des.c : mengimplementasikan fungsi-fungsi pemroses data
- desprog.c : program utama yang menghasilkan file *executable*

Kita juga dapat menggambarkan hierarki penyertaan file satu sama lain (`#include` files)



Gambar 4 Hierarki File Sumber dalam Proyek

Dari gambar di atas kita dapat melihat bahwa file `kunci.c`, `des.c`, dan `des.h` memerlukan `destype.h` yang mendefinisikan tipe-tipe data yang digunakan. File `desprog.c` memerlukan des.h untuk menemukan prototipe fungsi-fungsi yang diimplementasikan dalam file `kunci.c` dan `des.c`.

3.2 DEFINISI TIPE-TIPE DATA

Berdasarkan algoritma yang digunakan, terdapat beberapa variabel utama yang dapat dijadikan acuan,yaitu :

- blok data dan kunci masukan memiliki ukuran 64 bit
- blok data dibagi menjadi bagian ‘kiri’ dan ‘kanan’ masing-masing berukuran 32 bit
- kunci hasil PC1 berukuran 56 bit dibagi menjadi C dan D masing-masing berukuran 28 bit

- set kunci pada setiap iterasi berukuran 48 bit

u8 dan pu8

Blok data dan kunci dideklarasikan sebagai array u8 (1 byte = 8 bit), sehingga nantinya diperlukan 8 buah u8 beserta pointernya.

```
//............
typedef unsigned char u8;
typedef u8 *pu8;
//............
```

bit, pbit, kunci56, dan pkunci56

Pada kunci hasil PC1, untuk memudahkan proses permutasi, digunakan sebuah vektor bit yang diwakili oleh satu komponen bertipe data 'bit'. Walaupun dalam deklarasi berukuran 1 byte, namun dalam penggunaannya hanya akan bernilai 0 atau 1.

```
//............
typedef unsigned char bit;
typedef bit *pbit;

typedef struct {bit bits[NBITKUNCI56];} kunci56;
typedef kunci56 *pkunci56;
//............
```

kunci48, dan pkunci48

Pada kunci hasil PC2, digunakan array sebesar 8 byte yang masing-masing byte menyimpan informasi 6 bit informasi. Setiap byte yang digunakan untuk menyimpan 6 bit informasi informasi tersebut nantinya digunakan langsung untuk di-XOR dengan data hasil blok Ekspansi (E).

```
//............
typedef unsigned char u6;
typedef u6 *pu6;
typedef u6 kunci48[8];
typedef kunci48 *pkunci48;
//............



#define N_ITERASI 16
// definisi blok DES

typedef unsigned long u32;
typedef struct {
        u32 kiri;
        u32 kanan;
} desblok;
typedef desblok *pdesblok;

//............
```

3.3 PEMROSESAN KUNCI

Seperti telah diuraikan di atas, pemrosesan kunci dilakukan dengan beberapa blok fungsi utama, yaitu : Permuted Choice 1 (PC1), Permuted Choice 2 (PC2), dan ShiftKiri.

3.3.1 Fungsi pemroses kunci

Fungsi ini menggunakan 64-bit (8 byte) kunci input untuk dikonversi menjadi 16 set kunci yang akan dipergunakan dalam masing-masing proses Enkripsi maupun Dekripsi.

```
void ProsesKunci(pkunci48 KunciSet, pu8 Kunci)
{
   kunci56 temp;
   pbit ptemp = (pbit) &temp;

   int i,j;
   PC1(&temp, Kunci);

   for (i = 0; i < N_ITERASI; i++)
   {
      for (j = 0; j < tabShift[i]; j++)
      {
         ShiftKiri(ptemp, N_2BITKUNCI56);
        ShiftKiri(ptemp + N_2BITKUNCI56, N_2BITKUNCI56);
      }
      PC2(KunciSet[i], &temp);
   }

}
```

3.3.2 Permuted Choice 1 (PC1)

Susunan indeks bit yang menjadi tolok ukur PC1 dapat dideklarasikan sebagai berikut :

```
//#define NBITKUNCI56
int tabPC1[NBITKUNCI56] = {
   57, 49, 41, 33, 25, 17,  9,  1,
   58, 50, 42, 34, 26, 18, 10,  2,
   59, 51, 43, 35, 27, 19, 11,  3,
   60, 52, 44, 36,
   63, 55, 47, 39, 31, 23, 15,  7,
   62, 54, 46, 38, 30, 22, 14,  6,
   61, 53, 45, 37, 29, 21, 13,  5,
                   28, 20, 12,  4
};
//.................
```

Langkah pertama adalah dengan memecah bit-bit pada input kunci menjadi array bit. Kemudian, dengan menggunakan tabel indeks di atas, array bit tersebut dipermutasi.

```
void PC1(pkunci56 d, pu8 kunci)
{
   int j,k;
   register u8 b;
   pbit temp = (pbit) malloc (BYTEBLOK*BYTEBLOK);

   temp += 7;

// ekstraksi bit
   for (j = 0; j < BYTEBLOK; j++)
   {
       b = kunci[j];
       for (k = 0; k < BYTEBLOK; k++)
       {
              *temp-- = b & 1;
              b >>= 1;
       }
       temp += 16;
   }
   temp -= (8+BYTEBLOK*BYTEBLOK);

// permutasikan
   for (j = 0; j < NBITKUNCI56; j++)
   {
      d->bits[j] = temp[tabPC1[j]];
   }
   free(temp);
}
```

3.3.3 Permuted Choice 2 (PC2)

Pada fungsi ini, selain bit-bit dari C dan D dipermutasi, dilakukan juga pemuatan bit ke dalam satu byte set kunci. Seperti telah kita definisikan di atas, satu byte set kunci hanya memuat 6 bit informasi kunci.

```
void PC2(kunci48 d, pkunci56 s)
{
   register u6 i;
   pbit ps = (pbit) s;
   pu6 pd = (pu6) d;

   i =ps[13]; i<<=1; i|=ps[16]; i<<=1; i|=ps[10]; i<<=1;
```

```
   i|=ps[23]; i<<=1; i|=ps[0];  i<<=1; i|=ps[4];
   *pd++=i;
   i =ps[2];  i<<=1; i|=ps[27]; i<<=1; i|=ps[14]; i<<=1;
   i|=ps[5];  i<<=1; i|=ps[20]; i<<=1; i|=ps[9];
   *pd++=i;
   i =ps[22]; i<<=1; i|=ps[18]; i<<=1; i|=ps[11]; i<<=1;
   i|=ps[3];  i<<=1; i|=ps[25]; i<<=1; i|=ps[7];
   *pd++=i;
   i =ps[15]; i<<=1; i|=ps[6];  i<<=1; i|=ps[26]; i<<=1;
   i|=ps[19]; i<<=1; i|=ps[12]; i<<=1; i|=ps[1];
   *pd++=i;
   i =ps[40]; i<<=1; i|=ps[51]; i<<=1; i|=ps[30]; i<<=1;
   i|=ps[36]; i<<=1; i|=ps[46]; i<<=1; i|=ps[54];
   *pd++=i;
   i =ps[29]; i<<=1; i|=ps[39]; i<<=1; i|=ps[50]; i<<=1;
   i|=ps[44]; i<<=1; i|=ps[32]; i<<=1; i|=ps[47];
   *pd++=i;
   i =ps[43]; i<<=1; i|=ps[48]; i<<=1; i|=ps[38]; i<<=1;
   i|=ps[55]; i<<=1; i|=ps[33]; i<<=1; i|=ps[52];
   *pd++=i;
   i =ps[45]; i<<=1; i|=ps[41]; i<<=1; i|=ps[49]; i<<=1;
   i|=ps[35]; i<<=1; i|=ps[28]; i<<=1; i|=ps[31];
   *pd=i;
}
```

3.3.4 Shift Kiri

Sebenarnya proses yang dilakukan adalah *memutar* bit (rotation), bukan *menggeser (Shift)*. Input yang akan diputar adalah 28 array bit dari kunci hasil PC1 yang digeser secara periodik berdasarkan tabel pergeseran yang telah ditentukan. Penentuan jumlah pergeseran sendiri dipanggil dari fungsi ProsesKunci yang telah ditampilkan di bagian 3.3.1)

```
void ShiftKiri(pbit data, int len) // sebetulnya diputar: rotate.
{
   register bit i = *data;

   memcpy(data, data + 1, len-1);
   data[len-1] = i;
}
```

3.4 PEMROSESAN DATA

Setelah kunci berhasil diproses, maka langkah selanjutnya adalah memroses blok data, tentunya dengan menyertakan kunci di atas.

3.4.1 Fungsi pemroses data

Dalam fungsi ini, masukan dan keluaran data berupa blok data masing-masing 8 byte yang disertai oleh KunciSet yang memuat keseluruhan set kunci yang siap digunakan dalam setiap iterasi DES. Parameter mode digunakan untuk menentukan jenis proses baik Enkripsi maupun Dekripsi.

```
void DESenc (pu8 out, pu8 in, pkunci48 KunciSet, int mode)
{
   int i;
   u32 res;
   desblok dtemp,BlokBlok;

   data2blok(&BlokBlok, in);
   IP (&dtemp, &BlokBlok);
   if (mode) // Enkripsi
   for (i = 0; i< N_ITERASI ; i++)
   {
        res = Fungsi (dtemp.kanan, KunciSet[i]);
        res ^= dtemp.kiri;
        dtemp.kiri = dtemp.kanan;
        dtemp.kanan = res;
   }
   else // Dekripsi
   for (i = N_ITERASI-1; i>=0; i--)
   {
        res = Fungsi (dtemp.kanan, KunciSet[i]);
        res ^= dtemp.kiri;
        dtemp.kiri = dtemp.kanan;
        dtemp.kanan = res;
   }
   res = dtemp.kiri;
   dtemp.kiri = dtemp.kanan;
   dtemp.kanan = res;
   FP (&BlokBlok, &dtemp);
   blok2data(out, &BlokBlok);
}
```

3.4.2 IP dan FP

Initial Permutation (IP) dan Final Permutation (FP) merupakan suatu proses pengacakan blok data yang saling invers. Keduanya diimplementasikan dalam kode-kode sebagai berikut :

```
#define PERM_OP(a,b,t,n,m) ((t)=((((a)>>(n))^(b))&(m)),\
        (b)^=(t),\
        (a)^=((t)<<(n)))
```

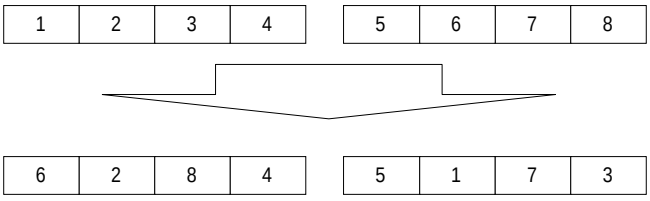
```
void IP (pdesblok out, pdesblok in)
{
   register u32 l = in->kiri;
   register u32 r = in->kanan;
   register u32 t;

   PERM_OP(l,r,t, 4,0x0f0f0f0f);
   PERM_OP(l,r,t,16,0x0000ffff);
   PERM_OP(r,l,t, 2,0x33333333);
   PERM_OP(r,l,t, 8,0x00ff00ff);
   PERM_OP(l,r,t, 1,0x55555555);

   out->kiri = l&0xffffffff;
   out->kanan = r&0xffffffff;
}

void FP (pdesblok out, pdesblok in)
{
   register u32 t;
   register u32 l = in->kiri;
   register u32 r = in->kanan;
   PERM_OP(l,r,t, 1,0x55555555);
   PERM_OP(r,l,t, 8,0x00ff00ff);
   PERM_OP(r,l,t, 2,0x33333333);
   PERM_OP(l,r,t,16,0x0000ffff);
   PERM_OP(l,r,t, 4,0x0f0f0f0f);
   out->kiri = l&0xffffffff;
   out->kanan = r&0xffffffff;
}
```

IP dan FP diimplementasikan dengan menggunakan macro PERM_OP yang berfungsi untuk menukar posisi (swapping) blok-blok bit dalam satu blok data. Dalam beberapa percobaan, akhirnya didapat bahwa lima buah PERM_OP pada desblok memungkinkan terjadinya IP dan FP.



Gambar 5 Ilustrasi PERM_OP dengan ukuran 8 bit pada blok data 8 byte

3.4.3 "Fungsi"

Fungsi ini menggabungkan beberapa blok : Expansion, Substitution, dan Permutation menjadi satu. SpBox merupakan sebuah *lookup table* dari Substitution dan Permutation dengan variabel lookup 6 bit kunci yang di-XOR dengan 6 bit hasil Ekspansi.

```
u32 Fungsi (u32 inp, kunci48 Kunci)
{
   register u32      p, r, q;
   register pbit key = (pbit) Kunci;

   p   = inp; p >>= 27;
   q   = p;   q  &= 3;  q <<= 4;
   r   = inp; r <<= 5;
   p  |= r;

   r  = spBox[0][key[0] ^ (p & 63)]; p >>= 4;
   r |= spBox[7][key[7] ^ (p & 63)]; p >>= 4;
   r |= spBox[6][key[6] ^ (p & 63)]; p >>= 4;
   r |= spBox[5][key[5] ^ (p & 63)]; p >>= 4;
   r |= spBox[4][key[4] ^ (p & 63)]; p >>= 4;
   r |= spBox[3][key[3] ^ (p & 63)]; p >>= 4;
   r |= spBox[2][key[2] ^ (p & 63)]; p >>= 4;
   r |= spBox[1][key[1] ^ (p | q)];

   return r;
}
```

3.5 PROGRAM DES

Fungsi-fungsi DES sebelumnya digunakan dalam suatu program dengan spesifikasi sebagai berikut :

- *Command line* enkriptor/dekriptor dengan format perintah :

  Des.exe <inputfile> <outputfile> <flag>

  Flag :

  -e : enkripsi

  -d : dekripsi
- Ukuran buffer 8192 byte
- Mode operasi Electronic Code Book (ECB)

Kode sumber untuk program DES adalah sebagai berikut :

```
#include <fcntl.h>
#include <time.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <mem.h>
#include "des.h"
```

```
u8 *Cp[7] = {
" X:\des <inputfile> <outputfile> <flag>",
" -e : Enkrip",
" -d : Dekrip",
" Implementasi Data Encryption Standar dgn C",
" Dian Tresna Nugraha, 13296119",
" (c) 1999",
NULL};

// buffer-buffer

#define BUFFSIZE 8192

pkunci48 KunciSet;
u8       Kunci[9];
pu8      ibuf, obuf;

// I/O file
FILE *fin, *fout;


int mf, kf;        //flags

char *probstr[6] = {
"Error 0 : Gagal mengalokasi memori",
"Error 1 : Kunci atau Data hasil Enkripsi Tidak Cocok!",
"Enkripsi / Dekripsi Berhasil!",
"Error 2 : Command line tidak lengkap!",
"Error 3 : Nama file tidak diketahui!"
"Error 4 : Tidak dapat membuka file!"

};

int main(int argc, char **argv)
{
   //argumen
   char *p,pp;
   time_t init_t, final_t;
   double elapsed_t;

   //iterator
   int i, j;

   //counter file & buffer
```

```
char *namain = NULL, *namaout = NULL;
int ex = 0, ypos = 0;
long pjofile=0, pjifile=0, pjdata=0, sisa, ntul=0;

//inisiasi buffer
int probno = 0;
if ((ibuf = (pu8) malloc (BUFFSIZE+8)) == NULL) goto problem;
else if
     ((obuf = (pu8) malloc (BUFFSIZE+8)) == NULL) goto problem;
else if
     ((KunciSet = (pkunci48) malloc (N_ITERASI*sizeof(kunci48)))== NULL )
     goto problem;

for (i=1; i<argc; i++)
{
     p=argv[i];
     if ((p[0] == '-') && (p[1] != '\0'))
     {
         p++;
         while (*p)
         {
             switch (pp=*(p++))
             {
                 case 'e':
                 case 'E':
                      mf = 1;
                      break;
                 case 'd':
                 case 'D':
                      mf = 0;
                      break;
                 default :
                      ex = 1;
             }
         }
     }
     else
     {
         if (namain == NULL)
            namain=argv[i];
         else if (namaout == NULL)
              namaout=argv[i];
         else
         {
```

```
                ex = 1;
            }
        }
    }
    if (ex) {
       probno = 3;
    }


    printf("Masukkan kunci : ");
    gets(Kunci);

    ProsesKunci(KunciSet, Kunci);

// pembacaan & penulisan pada file.........

    if (((fin = fopen(namain, "rb")) == NULL) ||
       ((fout = fopen(namaout, "wb")) == NULL))
    {
         probno = 5;
         goto problem;
    }

    setmode(fileno(fin),O_BINARY);
    setmode(fileno(fout),O_BINARY);

    init_t = time(NULL);
    for (pjifile=0;;)
    {
         pjdata = fread(ibuf, 1, BUFFSIZE, fin);  //1024  55
         pjifile += pjdata;
         sisa = pjdata % BYTEBLOK;                          //0     7
         // masukkan pad-pad...........
         if ((ex=feof(fin)) && mf)
            for (i = 0; i < 8 - sisa; i++)                    //0 it  1 it
                ibuf[pjdata++] = sisa | 0xF0;                 //1024  56

         for (i = 0; i < pjdata; i+=BYTEBLOK)
         {
             DESenc (obuf+i,ibuf+i,KunciSet, mf);
                    }

         if (!mf && ex)
         {
```

```
            i = pjdata;
            sisa = obuf[--i];
            while (sisa == obuf[--i]);
            if ((i+=1)%8 != (sisa & 0x0F))
            {
                probno = 1;
                goto problem;
            }

            pjdata = i;
        }
        pjofile += ntul = fwrite(obuf, 1, pjdata, fout);
        if (ex)
        {
           final_t = time(NULL);
           elapsed_t = difftime(final_t,init_t);
           printf("\nByte  :  %ld  baca  %ld  tulis\n%f  detik\n",  pjifile,
pjofile,elapsed_t);
           probno = 2;
           goto problem;
        }
   }


   problem :
     fclose (fout);
     fclose (fin);
     switch (probno)
     {
            case 3:
                  for(i=0;*Cp;i++)
                     puts(Cp[i]);
                  break;
            default :
                  puts(probstr[probno]);
                  break ;
     }
     puts("\n");
     free(KunciSet);
     free(ibuf);
     free(obuf);
     exit(0);
}
```

4 COMPILE DAN PENGUJIAN PROGRAM

Program dibuat oleh compiler 16 bit Borland C++ versi 3.1 untuk DOS. Tidak ada pesan kesalahan yang muncul. Output pada layar monitor hasil eksekusi program adalah :

```
C>DES.EXE des.exe des.enc -e
Masukkan kunci : DianTN..

Byte : 40736 baca 40744 tulis
0.000000 detik
Enkripsi / Dekripsi Berhasil!
```

Kemudian program diuji pada beberapa file yang berbeda. Tabel berikut merupakan hasil pengujian program untuk beberapa ukuran file yang berbeda :

Ukuran (Byte)	Waktu (Detik)	Byte/detik
233984	3	77994.66667
598842	8	74855.25
1116642	15	74442.8
7665860	109	70328.99083
Ratarata		**74405.42687**

Karena sifat enkripsi/dekripsi tidak menghilangkan satu informasi pun, maka program ini dapat dan telah diuji pada file-file terkompres seperti rar, arj, maupun zip. Hasilnya, file-file tersebut dapat dibuka kembali dengan baik oleh program dekompresi masing-masing.

5 KESIMPULAN DAN PENUTUP

Dari hasil pengujian implementasi program dapat disimpulkan bahwa proyek berhasil mencapai sasaran. Aplikasi lebih lanjut dari program hasil implementasi ini dapat diterapkan pada berbagai sistem komunikasi komputer seperti LAN, internet, ataupun untuk keperluan pribadi.

6 PUSTAKA

1. mfischer@blue.weeg.uiowa.edu (Matthew Fischer). *How to implement the Data Encryption Standard (DES).*1995
2. http://www.rsa.com/rsalabs/
3. http://www.ssh.fi/
4. eay@psych.psy.uq.oz.au (Eric Young). *DES implementation*. 1993

7 LAMPIRAN 1 : DES OPERATING MODES

To encrypt or decrypt more than 64 bits there are four official modes (defined in FIPS PUB 81). One is to go through the above-described process for each block in succession. This is called Electronic Codebook (ECB) mode. A stronger method is to exclusive-or each plaintext block with the preceding ciphertext block prior to encryption. (The first block is exclusive-or'ed with a secret 64-bit initialization vector (IV).) This is called Cipher Block Chaining (CBC) mode. The other two modes are Output Feedback (OFB) and Cipher Feedback (CFB).

When it comes to padding the data block, there are several options. One is to simply append zeros. Two suggested by FIPS PUB 81 are, if the data is binary data, fill up the block with bits that are the opposite of the last bit of data, or, if the data is ASCII data, fill up the block with random bytes and put the ASCII character for the number of pad bytes in the last byte of the block. Another technique is to pad the block with random bytes and in the last 3 bits store the original number of data bytes.

The DES algorithm can also be used to calculate checksums up to 64 bits long (see FIPS PUB 113). If the number of data bits to be checksummed is not a multiple of 64, the last data block should be padded with zeros. If the data is ASCII data, the first bit of each byte should be set to 0. The data is then encrypted in CBC mode with IV = 0. The leftmost n bits (where 16 <= n <= 64, and n is a multiple of 8) of the final ciphertext block are an n-bit checksum.

8 LAMPIRAN 2 : KODE SUMBER PROGRAM (LENGKAP)

8.1 DESTYPE.H

```
#ifndef _DESTYPE_H
#define _DESTYPE_H 1

#define BYTEBLOK 8
#define NBITKUNCI56 56
#define N_2BITKUNCI56 28

typedef unsigned char u8;
typedef u8 *pu8;

typedef unsigned char bit;
typedef bit *pbit;

typedef struct {bit bits[NBITKUNCI56];} kunci56;
typedef kunci56 *pkunci56;

typedef struct {bit bits[N_2BITKUNCI56];} kunci28;
typedef kunci28 *pkunci28;

typedef unsigned char u6;
typedef u6 *pu6;
typedef u6 kunci48[8];
typedef kunci48 *pkunci48;

#define N_ITERASI 16
// definisi blok DES

typedef unsigned long u32;
typedef struct {
        u32 kiri;
        u32 kanan;
} desblok;
typedef desblok *pdesblok;

extern u32 spBox [8][64];
```

```
#endif
```

8.2 DES.H

```
#ifndef _DES_H
#define _DES_H 1

#include "destype.h"

// prototype fungsi-fungsi pemroses Kunci

void PC1(pkunci56 d, pu8 kunci);
void PC2(kunci48 d, pkunci56 s);
void ShiftKiri(pbit data, int len); // sebetulnya diputer: rotate.
void ProsesKunci(pkunci48 KunciSet, pu8 Kunci);


// prototype fungsi-fungsi DES

void data2blok (pdesblok Blok, register pu8 input);
void IP (pdesblok out, pdesblok in);
// Fungsi (pdesblok out, pdesblok in, pkunci48 Kunci);
u32 Fungsi (u32 inp, kunci48 Kunci);
void FP (pdesblok out, pdesblok in);
void blok2data (pu8 output, pdesblok Blok);

void DESenc (pu8 out, pu8 in, pkunci48 KunciSet, int mode);


#endif
```

8.3 KUNCI.C

```
#include <mem.h>
#include <stdlib.h>
#include "des.h"

int tabShift[N_ITERASI] = {
   1, 1, 2, 2, 2, 2, 2, 2,
   1, 2, 2, 2, 2, 2, 2, 1,
};

int tabPC1[NBITKUNCI56] = {
   57, 49, 41, 33, 25, 17,  9,  1,
   58, 50, 42, 34, 26, 18, 10,  2,
```

```
   59, 51, 43, 35, 27, 19, 11,  3,
   60, 52, 44, 36,
   63, 55, 47, 39, 31, 23, 15,  7,
   62, 54, 46, 38, 30, 22, 14,  6,
   61, 53, 45, 37, 29, 21, 13,  5,
                   28, 20, 12,  4
};

void PC1(pkunci56 d, pu8 kunci)
{

   int j,k;
   register u8 b;
   pbit temp = (pbit) malloc (BYTEBLOK*BYTEBLOK);

   temp += 7;

   for (j = 0; j < BYTEBLOK; j++)
   {
       b = kunci[j];
       for (k = 0; k < BYTEBLOK; k++)
       {
              *temp-- = b & 1;
              b >>= 1;
       }
       temp += 16;
   }
   temp -= (8+BYTEBLOK*BYTEBLOK);

   for (j = 0; j < NBITKUNCI56; j++)
   {
      d->bits[j] = temp[tabPC1[j]];
   }

   free(temp);

}

void PC2(kunci48 d, pkunci56 s)
{
   register u6 i;
   pbit ps = (pbit) s;
   pu6 pd = (pu6) d;
```

```
   i =ps[13]; i<<=1; i|=ps[16]; i<<=1; i|=ps[10]; i<<=1;
   i|=ps[23]; i<<=1; i|=ps[0];  i<<=1; i|=ps[4];
   *pd++=i;
   i =ps[2];  i<<=1; i|=ps[27]; i<<=1; i|=ps[14]; i<<=1;
   i|=ps[5];  i<<=1; i|=ps[20]; i<<=1; i|=ps[9];
   *pd++=i;
   i =ps[22]; i<<=1; i|=ps[18]; i<<=1; i|=ps[11]; i<<=1;
   i|=ps[3];  i<<=1; i|=ps[25]; i<<=1; i|=ps[7];
   *pd++=i;
   i =ps[15]; i<<=1; i|=ps[6];  i<<=1; i|=ps[26]; i<<=1;
   i|=ps[19]; i<<=1; i|=ps[12]; i<<=1; i|=ps[1];
   *pd++=i;
   i =ps[40]; i<<=1; i|=ps[51]; i<<=1; i|=ps[30]; i<<=1;
   i|=ps[36]; i<<=1; i|=ps[46]; i<<=1; i|=ps[54];
   *pd++=i;
   i =ps[29]; i<<=1; i|=ps[39]; i<<=1; i|=ps[50]; i<<=1;
   i|=ps[44]; i<<=1; i|=ps[32]; i<<=1; i|=ps[47];
   *pd++=i;
   i =ps[43]; i<<=1; i|=ps[48]; i<<=1; i|=ps[38]; i<<=1;
   i|=ps[55]; i<<=1; i|=ps[33]; i<<=1; i|=ps[52];
   *pd++=i;
   i =ps[45]; i<<=1; i|=ps[41]; i<<=1; i|=ps[49]; i<<=1;
   i|=ps[35]; i<<=1; i|=ps[28]; i<<=1; i|=ps[31];
   *pd=i;

}

void ShiftKiri(pbit data, int len) // sebetulnya diputer: rotate.
{
   register bit i = *data;

   memcpy(data, data + 1, len-1);
   data[len-1] = i;

}

void ProsesKunci(pkunci48 KunciSet, pu8 Kunci)
{
   kunci56 temp;
   pbit ptemp = (pbit) &temp;

   int i,j;
   PC1(&temp, Kunci);
```

```
   for (i = 0; i < N_ITERASI; i++)
   {
      for (j = 0; j < tabShift[i]; j++)
      {
              ShiftKiri(ptemp, N_2BITKUNCI56);
       ShiftKiri(ptemp + N_2BITKUNCI56, N_2BITKUNCI56);
      }
      PC2(KunciSet[i], &temp);
   }

}
```

8.4 SPBOX.C

```
/*Tabel Permutasi SP (di-generate secara otomatis)*/

#include "destype.h"



u32    spBox[8][64] = {
       {
              0x00808200,0x00000000,0x00008000,0x00808202,
              0x00808002,0x00008202,0x00000002,0x00008000,
              0x00000200,0x00808200,0x00808202,0x00000200,
              0x00800202,0x00808002,0x00800000,0x00000002,
              0x00000202,0x00800200,0x00800200,0x00008200,
              0x00008200,0x00808000,0x00808000,0x00800202,
              0x00008002,0x00800002,0x00800002,0x00008002,
              0x00000000,0x00000202,0x00008202,0x00800000,
              0x00008000,0x00808202,0x00000002,0x00808000,
              0x00808200,0x00800000,0x00800000,0x00000200,
              0x00808002,0x00008000,0x00008200,0x00800002,
              0x00000200,0x00000002,0x00800202,0x00008202,
              0x00808202,0x00008002,0x00808000,0x00800202,
              0x00800002,0x00000202,0x00008202,0x00808200,
              0x00000202,0x00800200,0x00800200,0x00000000,
              0x00008002,0x00008200,0x00000000,0x00808002
       },     {
              0x40084010,0x40004000,0x00004000,0x00084010,
              0x00080000,0x00000010,0x40080010,0x40004010,
              0x40000010,0x40084010,0x40084000,0x40000000,
              0x40004000,0x00080000,0x00000010,0x40080010,
              0x00084000,0x00080010,0x40004010,0x00000000,
              0x40000000,0x00004000,0x00084010,0x40080000,
```

```
        0x00080010,0x40000010,0x00000000,0x00084000,
        0x00004010,0x40084000,0x40080000,0x00004010,
        0x00000000,0x00084010,0x40080010,0x00080000,
        0x40004010,0x40080000,0x40084000,0x00004000,
        0x40080000,0x40004000,0x00000010,0x40084010,
        0x00084010,0x00000010,0x00004000,0x40000000,
        0x00004010,0x40084000,0x00080000,0x40000010,
        0x00080010,0x40004010,0x40000010,0x00080010,
        0x00084000,0x00000000,0x40004000,0x00004010,
        0x40000000,0x40080010,0x40084010,0x00084000
    },  {
        0x00000104,0x04010100,0x00000000,0x04010004,
        0x04000100,0x00000000,0x00010104,0x04000100,
        0x00010004,0x04000004,0x04000004,0x00010000,
        0x04010104,0x00010004,0x04010000,0x00000104,
        0x04000000,0x00000004,0x04010100,0x00000100,
        0x00010100,0x04010000,0x04010004,0x00010104,
        0x04000104,0x00010100,0x00010000,0x04000104,
        0x00000004,0x04010104,0x00000100,0x04000000,
        0x04010100,0x04000000,0x00010004,0x00000104,
        0x00010000,0x04010100,0x04000100,0x00000000,
        0x00000100,0x00010004,0x04010104,0x04000100,
        0x04000004,0x00000100,0x00000000,0x04010004,
        0x04000104,0x00010000,0x04000000,0x04010104,
        0x00000004,0x00010104,0x00010100,0x04000004,
        0x04010000,0x04000104,0x00000104,0x04010000,
        0x00010104,0x00000004,0x04010004,0x00010100
    },  {
        0x80401000,0x80001040,0x80001040,0x00000040,
        0x00401040,0x80400040,0x80400000,0x80001000,
        0x00000000,0x00401000,0x00401000,0x80401040,
        0x80000040,0x00000000,0x00400040,0x80400000,
        0x80000000,0x00001000,0x00400000,0x80401000,
        0x00000040,0x00400000,0x80001000,0x00001040,
        0x80400040,0x80000000,0x00001040,0x00400040,
        0x00001000,0x00401040,0x80401040,0x80000040,
        0x00400040,0x80400000,0x00401000,0x80401040,
        0x80000040,0x00000000,0x00000000,0x00401000,
        0x00001040,0x00400040,0x80400040,0x80000000,
        0x80401000,0x80001040,0x80001040,0x00000040,
        0x80401040,0x80000040,0x80000000,0x00001000,
        0x80400000,0x80001000,0x00401040,0x80400040,
        0x80001000,0x00001040,0x00400000,0x80401000,
        0x00000040,0x00400000,0x00001000,0x00401040
```

```
},     {
       0x00000080,0x01040080,0x01040000,0x21000080,
       0x00040000,0x00000080,0x20000000,0x01040000,
       0x20040080,0x00040000,0x01000080,0x20040080,
       0x21000080,0x21040000,0x00040080,0x20000000,
       0x01000000,0x20040000,0x20040000,0x00000000,
       0x20000080,0x21040080,0x21040080,0x01000080,
       0x21040000,0x20000080,0x00000000,0x21000000,
       0x01040080,0x01000000,0x21000000,0x00040080,
       0x00040000,0x21000080,0x00000080,0x01000000,
       0x20000000,0x01040000,0x21000080,0x20040080,
       0x01000080,0x20000000,0x21040000,0x01040080,
       0x20040080,0x00000080,0x01000000,0x21040000,
       0x21040080,0x00040080,0x21000000,0x21040080,
       0x01040000,0x00000000,0x20040000,0x21000000,
       0x00040080,0x01000080,0x20000080,0x00040000,
       0x00000000,0x20040000,0x01040080,0x20000080
},     {
       0x10000008,0x10200000,0x00002000,0x10202008,
       0x10200000,0x00000008,0x10202008,0x00200000,
       0x10002000,0x00202008,0x00200000,0x10000008,
       0x00200008,0x10002000,0x10000000,0x00002008,
       0x00000000,0x00200008,0x10002008,0x00002000,
       0x00202000,0x10002008,0x00000008,0x10200008,
       0x10200008,0x00000000,0x00202008,0x10202000,
       0x00002008,0x00202000,0x10202000,0x10000000,
       0x10002000,0x00000008,0x10200008,0x00202000,
       0x10202008,0x00200000,0x00002008,0x10000008,
       0x00200000,0x10002000,0x10000000,0x00002008,
       0x10000008,0x10202008,0x00202000,0x10200000,
       0x00202008,0x10202000,0x00000000,0x10200008,
       0x00000008,0x00002000,0x10200000,0x00202008,
       0x00002000,0x00200008,0x10002008,0x00000000,
       0x10202000,0x10000000,0x00200008,0x10002008
},     {
       0x00100000,0x02100001,0x02000401,0x00000000,
       0x00000400,0x02000401,0x00100401,0x02100400,
       0x02100401,0x00100000,0x00000000,0x02000001,
       0x00000001,0x02000000,0x02100001,0x00000401,
       0x02000400,0x00100401,0x00100001,0x02000400,
       0x02000001,0x02100000,0x02100400,0x00100001,
       0x02100000,0x00000400,0x00000401,0x02100401,
       0x00100400,0x00000001,0x02000000,0x00100400,
       0x02000000,0x00100400,0x00100000,0x02000401,
```

```
            0x02000401,0x02100001,0x02100001,0x00000001,
            0x00100001,0x02000000,0x02000400,0x00100000,
            0x02100400,0x00000401,0x00100401,0x02100400,
            0x00000401,0x02000001,0x02100401,0x02100000,
            0x00100400,0x00000000,0x00000001,0x02100401,
            0x00000000,0x00100401,0x02100000,0x00000400,
            0x02000001,0x02000400,0x00000400,0x00100001
      },    {
            0x08000820,0x00000800,0x00020000,0x08020820,
            0x08000000,0x08000820,0x00000020,0x08000000,
            0x00020020,0x08020000,0x08020820,0x00020800,
            0x08020800,0x00020820,0x00000800,0x00000020,
            0x08020000,0x08000020,0x08000800,0x00000820,
            0x00020800,0x00020020,0x08020020,0x08020800,
            0x00000820,0x00000000,0x00000000,0x08020020,
            0x08000020,0x08000800,0x00020820,0x00020000,
            0x00020820,0x00020000,0x08020800,0x00000800,
            0x00000020,0x08020020,0x00000800,0x00020820,
            0x08000800,0x00000020,0x08000020,0x08020000,
            0x08020020,0x08000000,0x00020000,0x08000820,
            0x00000000,0x08020820,0x00020020,0x08000020,
            0x08020000,0x08000800,0x08000820,0x00000000,
            0x08020820,0x00020800,0x00020800,0x00000820,
            0x00000820,0x00020020,0x08000000,0x08020800
      }};
```

8.5 DESPROG.C

```
#include <fcntl.h>
#include <time.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <mem.h>
#include "des.h"

u8 *Cp[7] = {
" X:\des <inputfile> <outputfile> <flag>",
" -e : Enkrip",
" -d : Dekrip",
" Implementasi Data Encryption Standar dgn C",
" Dian Tresna Nugraha, 13296119",
" (c) 1999",
NULL};
```

```
// buffer-buffer

#define BUFFSIZE 8192

pkunci48 KunciSet;
u8       Kunci[9];
pu8      ibuf, obuf;

// I/O file
FILE *fin, *fout;


int mf, kf;        //flags

char *probstr[6] = {
"Error 0 : Gagal mengalokasi memori",
"Error 1 : Kunci atau Data hasil Enkripsi Tidak Cocok!",
"Enkripsi / Dekripsi Berhasil!",
"Error 2 : Command line tidak lengkap!",
"Error 3 : Nama file tidak diketahui!"
"Error 4 : Tidak dapat membuka file!"

};

int main(int argc, char **argv)
{
   //argumen
   char *p,pp;
   time_t init_t, final_t;
   double elapsed_t;

   //iterator
   int i, j;

   //counter file & buffer
   char *namain = NULL, *namaout = NULL;
   int ex = 0, ypos = 0;
   long pjofile=0, pjifile=0, pjdata=0, sisa, ntul=0;

   //inisiasi buffer
   int probno = 0;
   if ((ibuf = (pu8) malloc (BUFFSIZE+8)) == NULL) goto problem;
   else if
        ((obuf = (pu8) malloc (BUFFSIZE+8)) == NULL) goto problem;
```

```
else if
     ((KunciSet = (pkunci48) malloc (N_ITERASI*sizeof(kunci48)))== NULL )
     goto problem;

for (i=1; i<argc; i++)
{
     p=argv[i];
     if ((p[0] == '-') && (p[1] != '\0'))
     {
         p++;
         while (*p)
         {
             switch (pp=*(p++))
             {
                 case 'e':
                 case 'E':
                      mf = 1;
                      break;
                 case 'd':
                 case 'D':
                      mf = 0;
                      break;
                 default :
                      ex = 1;
             }
         }
     }
     else
     {
         if (namain == NULL)
            namain=argv[i];
         else if (namaout == NULL)
              namaout=argv[i];
         else
         {
             ex = 1;
         }
     }
}
if (ex) {
   probno = 3;
}
```

```
   printf("Masukkan kunci : ");
   gets(Kunci);

   ProsesKunci(KunciSet, Kunci);

// pembacaan & penulisan pada file.........

   if (((fin = fopen(namain, "rb")) == NULL) ||
      ((fout = fopen(namaout, "wb")) == NULL))
   {
        probno = 5;
        goto problem;
   }

   setmode(fileno(fin),O_BINARY);
   setmode(fileno(fout),O_BINARY);

   init_t = time(NULL);
   for (pjifile=0;;)
   {
        pjdata = fread(ibuf, 1, BUFFSIZE, fin);  //1024  55
        pjifile += pjdata;
        sisa = pjdata % BYTEBLOK;                          //0     7
        // masukkan pad-pad...........
        if ((ex=feof(fin)) && mf)
           for (i = 0; i < 8 - sisa; i++)                     //0 it  1 it
               ibuf[pjdata++] = sisa | 0xF0;                  //1024  56

        for (i = 0; i < pjdata; i+=BYTEBLOK)
        {
            DESenc (obuf+i,ibuf+i,KunciSet, mf);
                  }

        if (!mf && ex)
        {
            i = pjdata;
            sisa = obuf[--i];
            while (sisa == obuf[--i]);
            if ((i+=1)%8 != (sisa & 0x0F))
            {
                probno = 1;
                goto problem;
            }
```

```
            pjdata = i;
        }

        pjofile += ntul = fwrite(obuf, 1, pjdata, fout);

        if (ex)
        {
           final_t = time(NULL);
           elapsed_t = difftime(final_t,init_t);
           printf("\nByte  :  %ld  baca  %ld  tulis\n%f  detik\n",  pjifile,
pjofile,elapsed_t);
           probno = 2;
           goto problem;
        }
   }


   problem :
     fclose (fout);
     fclose (fin);
     switch (probno)
     {
            case 3:
                  for(i=0;*Cp;i++)
                     puts(Cp[i]);
                  break;
            default :
                  puts(probstr[probno]);
                  break ;
     }
     puts("\n");
     free(KunciSet);
     free(ibuf);
     free(obuf);
     exit(0);
}
```

8.6 DES.C

```
#include "des.h"


// implementasi fungsi-fungsi DES

void data2blok (pdesblok Blok, register pu8 input)
```

```
{
   register u32  d;

   d  = *input++; d<<= 8;
   d |= *input++; d<<= 8;
   d |= *input++; d<<= 8;
   d |= *input++;
   Blok->kiri = d;
   d  = *input++; d<<= 8;
   d |= *input++; d<<= 8;
   d |= *input++; d<<= 8;
   d |= *input++;
   Blok->kanan = d;

}

#define PERM_OP(a,b,t,n,m) ((t)=((((a)>>(n))^(b))&(m)),\
        (b)^=(t),\
        (a)^=((t)<<(n)))

void IP (pdesblok out, pdesblok in)
{
   register u32 l = in->kiri;
   register u32 r = in->kanan;
   register u32 t;

   PERM_OP(l,r,t, 4,0x0f0f0f0f);
   PERM_OP(l,r,t,16,0x0000ffff);
   PERM_OP(r,l,t, 2,0x33333333);
   PERM_OP(r,l,t, 8,0x00ff00ff);
   PERM_OP(l,r,t, 1,0x55555555);

   out->kiri = l&0xffffffff;
   out->kanan = r&0xffffffff;
}

void FP (pdesblok out, pdesblok in)
{
   register u32 t;
   register u32 l = in->kiri;
   register u32 r = in->kanan;

   PERM_OP(l,r,t, 1,0x55555555);
   PERM_OP(r,l,t, 8,0x00ff00ff);
```

```
   PERM_OP(r,l,t, 2,0x33333333);
   PERM_OP(l,r,t,16,0x0000ffff);
   PERM_OP(l,r,t, 4,0x0f0f0f0f);

   out->kiri = l&0xffffffff;
   out->kanan = r&0xffffffff;

}

u32 Fungsi (u32 inp, kunci48 Kunci)
{
   register u32      p, r, q;
   register pbit key = (pbit) Kunci;

   p   = inp; p >>= 27;
   q   = p;   q  &= 3;  q <<= 4;
   r   = inp; r <<= 5;
   p  |= r;

   r  = spBox[0][key[0] ^ (p & 63)]; p >>= 4;
   r |= spBox[7][key[7] ^ (p & 63)]; p >>= 4;
   r |= spBox[6][key[6] ^ (p & 63)]; p >>= 4;
   r |= spBox[5][key[5] ^ (p & 63)]; p >>= 4;
   r |= spBox[4][key[4] ^ (p & 63)]; p >>= 4;
   r |= spBox[3][key[3] ^ (p & 63)]; p >>= 4;
   r |= spBox[2][key[2] ^ (p & 63)]; p >>= 4;
   r |= spBox[1][key[1] ^ (p | q)];

   return r;

}

void blok2data (pu8 output, pdesblok Blok)
{
   register pu8  dp = output + 8;
   register u32 s;

   s = Blok->kanan;
   *--dp = (u8) s; s >>= 8;
   *--dp = (u8) s; s >>= 8;
   *--dp = (u8) s; s >>= 8;
   *--dp = s;

   s = Blok->kiri;
```

```
   *--dp = (u8) s; s >>= 8;
   *--dp = (u8) s; s >>= 8;
   *--dp = (u8) s; s >>= 8;
   *--dp = s;

}

void DESenc (pu8 out, pu8 in, pkunci48 KunciSet, int mode)
{
   int i;
   u32 res;
   desblok dtemp,BlokBlok;

   data2blok(&BlokBlok, in);
   IP (&dtemp, &BlokBlok);

   if (mode)
   for (i = 0; i< N_ITERASI ; i++)
   {
        res = Fungsi (dtemp.kanan, KunciSet[i]);
        res ^= dtemp.kiri;
        dtemp.kiri = dtemp.kanan;
        dtemp.kanan = res;
   }

   else
   for (i = N_ITERASI-1; i>=0; i--)
   {
        res = Fungsi (dtemp.kanan, KunciSet[i]);
        res ^= dtemp.kiri;
        dtemp.kiri = dtemp.kanan;
        dtemp.kanan = res;
   }

   res = dtemp.kiri;
   dtemp.kiri = dtemp.kanan;
   dtemp.kanan = res;

   FP (&BlokBlok, &dtemp);
   blok2data(out, &BlokBlok);

}
```

9 LAMPIRAN 3 :

PROPOSAL PROYEK

"IMPLEMENTASI ENKRIPSI DATA BERBASIS ALGORITMA DATA ENCRYPTION STANDARD (DES) DENGAN BAHASA C"

Diajukan untuk memenuhi tugas kurikuler mata kuliah Teknologi Informasi EL-517

Oleh

Dian Tresna Nugraha (13296119)

Pembimbing

Onno W. Purbo

Abstrak

Komunikasi internet kini telah meluas. Fasilitas-fasilitas yang dimiliki oleh internet diantaranya adalah : World Wide Web (WWW), File Transfer Protocol (FTP), Electronic Mail (E-Mail), Newsgroup, dan sebagainya. Keamanan data menjadi isu utama dengan keberadaan hacker yang selalu ingin mengetahui informasi dengan cara yang tidak semestinya. Untuk itu, diperlukan proses pengamanan data baik pada saat penyimpanan maupun pada saat transfer informasi. Enkripsi data merupakan cara efektif untuk menyembunyikan informasi sebenarnya yang terdapat pada suatu data. Algoritma Data Encryption Standard (DES) merupakan salah satu cara enkripsi data blok per 64 bit dan dengan kunci 64-bit pula.

19 September 1999