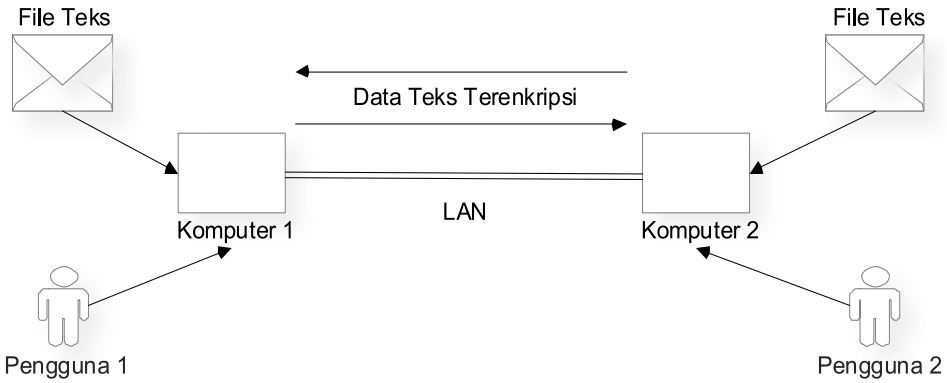
BAB 3

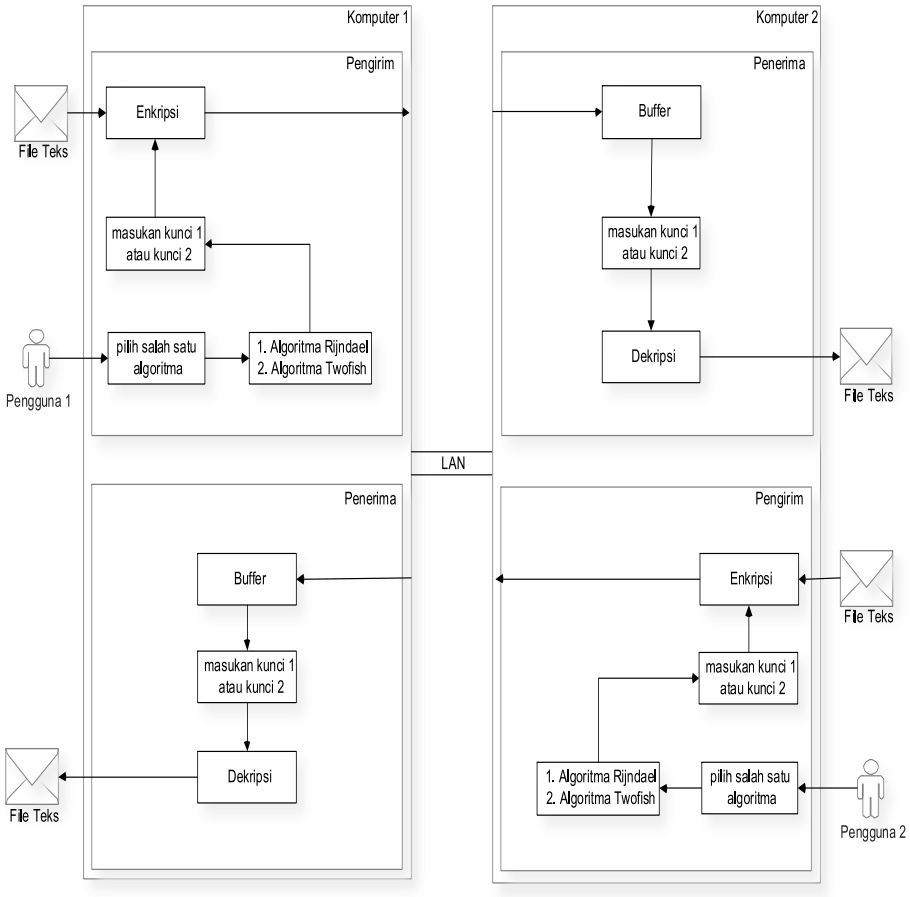
ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Sistem

Analisis adalah penguraian dari suatu pembahasan, dalam hal ini pembahasan mengenai analisis perbandingan algoritma *rijndael* dan algoritma *twofish* pada proses pengiriman data teks menggunakan jaringan *Local Area Network* (*LAN*), yang berguna untuk mengetahui proses enkripsi dan dekripsi dari kedua algoritma tersebut. Permasalahan utama yang ada pada perangkat lunak adalah mengenai enkripsi dan dekripsi. Bagaimana melakukan enkripsi terhadap sebuah pesan teks pada proses pengiriman secara *streaming* dengan menggunakan jaringan *Local Area Network* (*LAN*) dan dekripsi terhadap data teks yang sudah berisi *ciphertext* sehingga dapat ditampilkan ke *user* dalam *plaintext*. Dimana data yang ada pada komputer boleh jadi merupakan data yang sangat penting dan rahasia yang tidak dimungkinkan orang lain mengaksesnya. Dengan mengimplementasikan kriptografi sebuah data menjadi tidak terbaca atau tidak dimengerti oleh pihak yang tidak berwenang karena suatu proses perubahan terhadap data itu sendiri dengan sandi atau kode tertentu. Sebuah kunci digunakan dalam algoritma untuk menyandikan semua data menjadi *cryptogram*. Dengan begitu, sebuah data hanya akan diakses oleh orang yang mengetahui kunci saja dan kemungkinan dicuri atau dirusak telah diminimalisasi.



Gambar 3.1 Gambaran Umum Sistem



Gambar 3.2 Arsitektur Detail Sistem

Pada bagian analisis sistem ini akan dibahas tentang analisis masalah, analisis *file text*, analisis kriptografi, analisis non fungsional, dan analisis kebutuhan fungsional.

3.1.1 Analisis Masalah

Kriptografi bertujuan untuk menjaga kerahasiaan informasi yang terkandung dalam data sehingga informasi tersebut tidak dapat diketahui oleh pihak yang tidak berkepentingan.

Kebanyakan kriptografi diterapkan pada *file* yang memiliki *format .txt, .doc*, dan *.pdf*. Masalah yang timbul adalah bagaimana jika data/pesan rahasia tersebut yang ada di komputer tidak mudah dapat dibaca oleh orang lain yang tidak berkepentingan di jaringan *Local Area Network (LAN)*, sehingga dapat menjadi suatu hambatan apabila komputer kita sudah terhubung di jaringan *Local Area Network (LAN)* pada saat mendistribusikan *file* teks yang berisi data/pesan rahasia tersebut.

Pada sistem yang akan direalisasikan ini akan mengatasi masalah tersebut, yaitu merubah isi data dari bentuk pesan asli (*plaintext*) menjadi bentuk pesan yang disandikan (*chipertext*) yang memiliki *format .txt, .doc,* dan *.pdf* pada saat data sedang dikirimkan ke komputer tujuan di jaringan *Local Area Network (LAN).*

3.1.2 Analisis Kriptografi

Permasalahan yang akan diselesaikan dalam pelaksanaan tugas akhir ini adalah menerapkan algoritma *Rijndael* dan algoritma *Twofish* agar dapat

digunakan untuk melakukan enkripsi teks dalam proses pengiriman pesan berupa teks antar dua buah komputer melalui jaringan. Pengiriman teks bersifat dua arah.

Untuk lebih jelasnya, akan dibahas 2 metode pada proses enkripsi dan dekripsi sebuah pesan teks, yaitu :

1. Algoritma *Rijndael*
2. Algoritma *Twofish*

3.1.2.1 Algoritma *Rijndael*

Rijndael Adalah algoritma yang beroperasi dalam *byte*, bukan dalam *bit*. Algoritma ini mampu melakukan enkripsi terhadap *plaintext* sebesar 16 *byte* atau 128 *bit*.

Algoritma *Rijndael* juga melakukan putaran enkripsi (*enciphering*) sebanyak 10 putaran namun bukan putaran yang merupakan jaringan *Feistel*. *Enciphering* pada *Rijndael* melibatkan empat proses yaitu :

1. *Sub Bytes*
2. *Shift Rows*
3. *Mix Columns*
4. *Add Round Key*

Secara umum, proses enkripsi dilakukan dengan *initial round* yaitu melakukan *XOR* antara *state* awal yang masih berupa *plaintext* dengan *cipher key*. Kemudian melakukan keempat proses diatas sebanyak 9 kali putaran, dan terakhir adalah *final round* yang melibatkan proses *sub bytes, shift rows,* dan *add round key*.

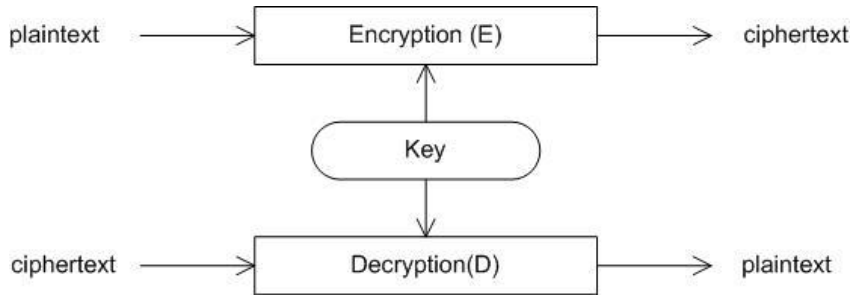
3.1.2.1.1 Panjang Kunci (*Key*) Algoritma *Rijndael*

Panjang kunci untuk keamanan dari sebuah teknik penyandian tergantung dari dua hal yaitu algoritma penyandian dan panjang kunci (*key*). Algoritma *Rijndael* sangat menentukan kekuatan dari sebuah teknik penyandian, tetapi panjang kunci juga tidak kalah penting dalam menentukan kekuatan sebuah teknik penyandian.

Dengan melihat situasi ini, maka kriptografi yang baik akan memilih untuk menggunakan sepanjang mungkin kunci yang akan digunakan, namun hal ini tidak dapat diterapkan begitu saja. Semakin panjang kunci, semakin lama pula waktu yang digunakan oleh komputer untuk melakukan proses enkripsi. Oleh sebab itu, panjang kunci yang akan digunakan hendaknya memperhatikan 3 hal, yaitu seberapa penting data yang akan dirahasiakan, berapa lama waktu yang dibutuhkan agar data tersebut tetap aman, dan seberapa kuat kemampuan kriptanalis dalam memecahkan teknik penyandian kita. Untuk algoritma *Rijndael* ini sendiri memakai kunci dengan panjang 128 bit karena panjang kunci ini dianggap paling optimal untuk saat ini.

3.1.2.1.2 Kunci Simetri Algoritma *Rijndael*

Kriptografi *secret key* adalah kriptografi yang hanya melibatkan satu kunci dalam proses enkripsi dan dekripsi. Proses dekripsi dalam kriptografi *secret key* ini adalah kebalikan dari proses enkripsi.



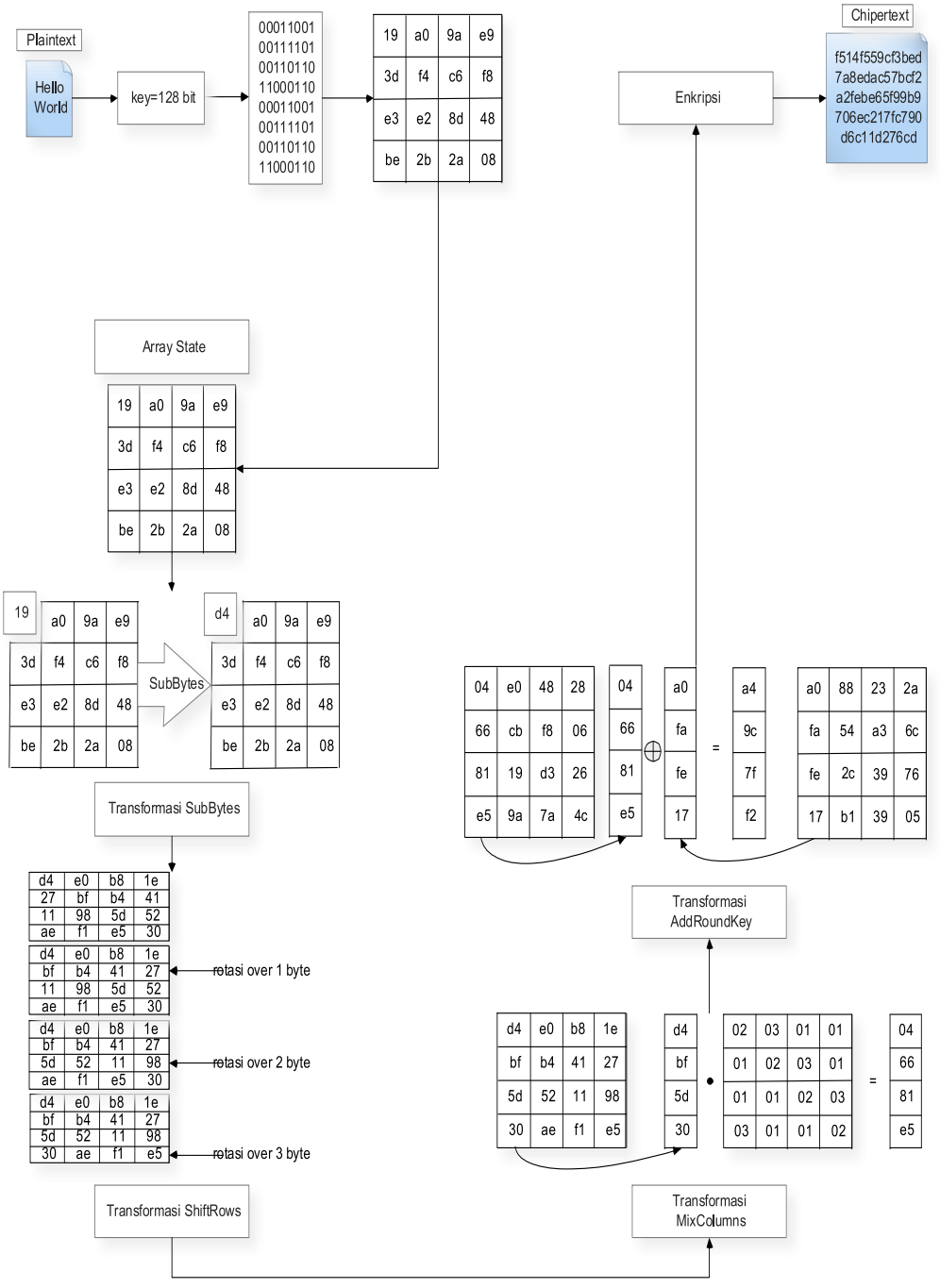
Gambar 3.3 Kriptografi Simetris *Rijndael*

Kriptografi *secret key* seringkali disebut sebagai kriptografi konvensional atau kriptografi simetris (*Symmetric Cryptography*) dimana proses dekripsi adalah kebalikan dari proses enkripsi dan menggunakan kunci yang sama.

Kriptografi simetris dapat dibagi menjadi dua, yaitu penyandian blok dan penyandian alir. Penyandian blok bekerja pada suatu data yang terkelompok menjadi blok-blok data atau kelompok data dengan panjang data yang telah ditentukan. Pada penyandian blok, data yang masuk akan dipecah-pecah menjadi blok data yang telah ditentukan ukurannya. Penyandian alir bekerja pada suatu data bit tunggal atau terkadang dalam satu byte. Jadi format data yang mengalami proses enkripsi dan dekripsi adalah berupa aliran bit-bit data.

Algoritma *Rijndael* yang ada pada saat ini kebanyakan bekerja untuk penyandian blok karena kebanyakan proses pengiriman data pada saat ini menggunakan blok-blok data yang telah ditentukan ukurannya untuk kemudian dikirim melalui jaringan *Local Area Network* (*LAN*).

Cara penggunaan metode *Rijndael* pada kriptografi berupa *file* teks seperti berikut:



Gambar 3.4 Proses Enkripsi *Rijndael*

Pada proses enkripsi pesan terdapat tugas-tugas seperti berikut :

1. *Load file* teks yang akan dienkripsi. Format teks yang digunakan adalah *.txt, .doc,* dan *.pdf*. Sedangkan untuk ukuran *file*nya tidak ada batasannya, semakin besar ukuran *file*nya maka semakin banyak pula pesan yang dapat dienkripsi.
2. Pada proses enkripsi dan dekripsi *Rijndael* proses *Key Schedule* sama, mengambil kunci *cipher* dan melakukan rutin ekspansi kunci (*key expansion*) untuk membentuk *key schedule*. Ekspansi kunci menghasilkan total Nb(Nr+1) *word*. Algoritma ini membutuhkan set awal *key* yang terdiri dari Nb *word*, dan setiap *round* Nr membutuhkan data kunci sebanyak Nb *word*. Hasil *key schedule* terdiri dari *array* 4 *byte word* linear yang dinotasikan dengan [wi]. *SubWord* adalah fungsi yang mengambil 4 *byte word* input dan mengaplikasikan S-Box ke tiap-tiap data 4 *byte* untuk menghasilkan *word output*. Fungsi *RotWord* mengambil *word* [a0, a1, a2, a3] sebagai *input*, melakukan permutasi siklik, dan mengembalikan *word* [a1,a2,a3,a0].

01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

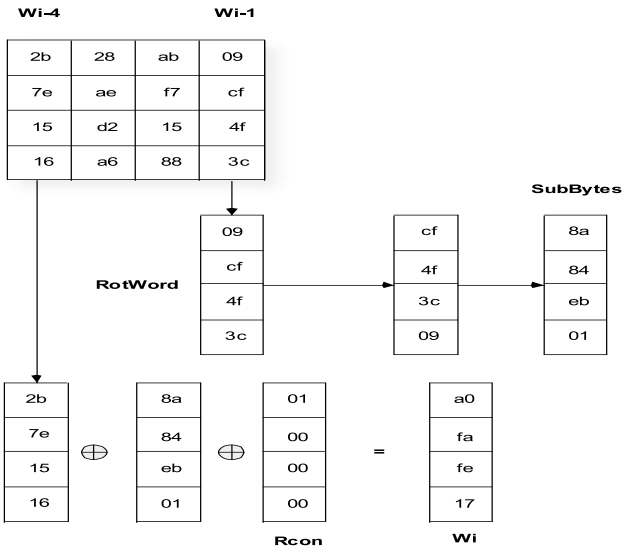
Gambar 3.5 *Rcon*

Dari Gambar 3.5 dapat dilihat bahwa *Rcon*[i] terdiri dari nilai-nilai yang diberikan oleh [xi-1, {00}, {00}, {00}], dengan xi-1 sebagai pangkat dari x (x dinotasikan sebagai {02}).

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

Gambar 3.6 *Cipher Key*

Dari Gambar 3.6 dapat dilihat bahwa kunci kriptografi yang digunakan oleh rutin untuk Ekspansi Kunci menghasilkan satu set *round Keys*, dapat digambarkan sebagai *array* persegi panjang *byte*, yang memiliki empat baris dan kolom Nk.



Gambar 3.7 Proses *Key Schedule*

Dari Gambar 3.7 dapat dilihat bahwa *word* ke *Nk* pertama pada ekspansi kunci berisi kunci *cipher*. Setiap *word* berikutnya, w[i], sama dengan XOR dari *word* sebelumnya, w[i-1] dan *word Nk* yang ada pada posisi sebelumnya, w[i-Nk]. Untuk *word* pada posisi yang merupakan kelipatan *Nk*, sebuah transformasi diaplikasikan pada w[i-1] sebelum *XOR*, lalu dilanjutkan oleh *XOR* dengan konstanta*round*, *Rcon*[i]. Transformasi ini terdiri dari pergeseran siklik dari *byte* data dalam suatu *word RotWord*, lalu diikuti aplikasi dari *lookup* tabel untuk semua 4 *byte* data dari *word SubWord*.

3. Selama kalkulasi *plainteks* menjadi *cipherteks*, status sekarang dari data disimpan di dalam *array of byte* dua dimensi, *state*, yang berukuran *NROWS x NCOLS*. Elemen *array state* diacu sebagai berikut :

$$s[r,c] = in[r+4c] \text{ untuk } 0 \leq r < 4 \text{ dan } 0 \leq c < Nb$$

19	a0	9a	e9
3d	f4	c6	f8
e3	e2	8d	48
be	2b	2a	08

Gambar 3.8 Pengisian *Array State*

Dari Gambar 3.8 dapat dilihat bahwa(Nb adalah panjang blok dibagi 32). Pada AES, Nb = 128/32 = 4. Tiap elemen dari *array state* diisi dengan 8 *bit* teks (1 *byte*) dalam notasi *HEX*.

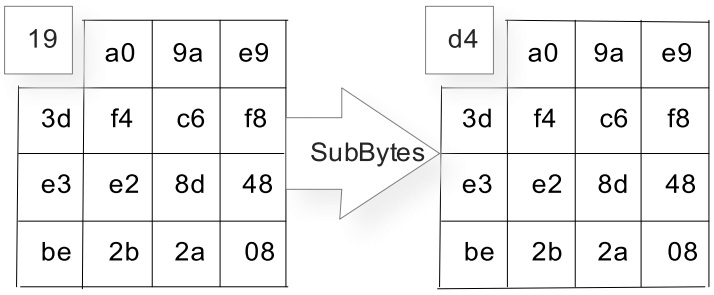
4. Setelah pengisian *array state* selesai, lakukan transformasi *SubBytes* dimana transformasi *byte* pada setiap elemen pada *state* akan dipetakan dengan menggunakan sebuah tabel substitusi (*S-Box*). Tabel substitusi *S-Box* akan dipaparkan dalam Tabel 3.1.

Tabel 3.1 *S-Box* untuk *SubBytes*

hex	y															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	**d4**	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	3	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

x

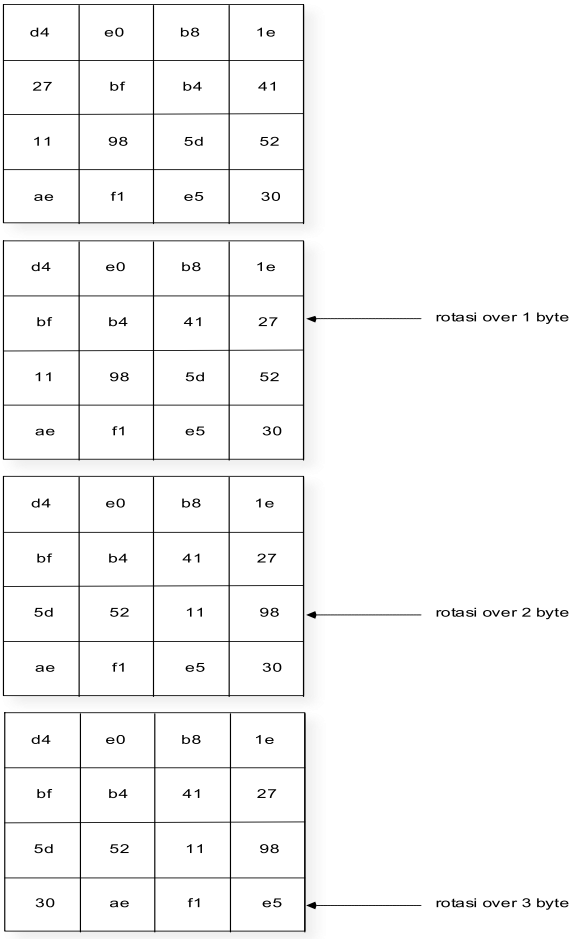
Berikut hasil *array state* yang sudah di transformasi *SubBytes* :



Gambar 3.9 Transformasi *SubBytes*

Dari Gambar 3.9 dapat dilihat bahwa *S-box* yang digunakan dalam *SubBytes*() transformasi disajikan dalam bentuk heksadesimal pada Tabel 3.1. Sebagai contoh, jika = 1,9s {D4}, maka nilai substitusi akan ditentukan oleh perempatan dari baris dengan indeks '1' dan kolom dengan indeks '9'.

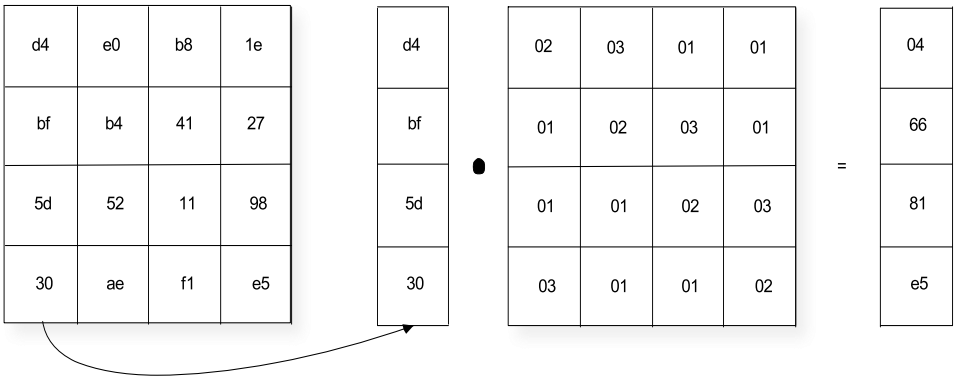
5. Lakukan transformasi *Shiftrows* yang pada dasarnya adalah proses pergeseran *bit* dimana *bit* paling kiri akan dipindahkan menjadi *bit* paling kanan (rotasi *bit*). Proses pergeseran *ShiftRows* ditunjukan dalam gambar 3.10 berikut :



Gambar 3.10 Transformasi *ShiftRows*

6. Setelah proses transformasi *ShiftRows* selesai, lakukan pengacakan *array state* dengan cara melakukan perkalian matriks yang merupakan transformasi dari perkalian polinom antara tiap kolom dengan polinom 4 suku pada $GF(2^8)$,

$$a(x) \bmod (x4 + 1).$$



Gambar 3.11 *MixColumns*

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

Gambar 3.12 *Galois Field By a Given Matrix*

04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

Gambar 3.13 Hasil Akhir *MixColumns*

7. Pada proses enkripsi dan dekripsi *Rijndael* proses *AddRoundKey* sama, sebuah *round key* ditambahkan pada *state* dengan operasi *XOR*. Setiap *round key* terdiri dari *Nb word* dimana tiap word tersebut akan dijumlahkan dengan *word* atau kolom yang bersesuaian dari *state* sehingga :

$$[S'_{0,c},S'_{1,c},S'_{2,c},S'_{3,c}] = [S_{0,c},S_{1,c},S_{2,c},S_{3,c}] \oplus [W_{round*Nb+c}] \text{ untuk } 0 \leq c \leq Nb$$

[Wi] adalah word dari key yang bersesuaian dimana :

$$i = round*Nb+c$$

Transformasi *AddRoundKey* pada proses enkripsi pertama kali pada *round* = 0 untuk *round* selanjutnya *round= round* + 1.

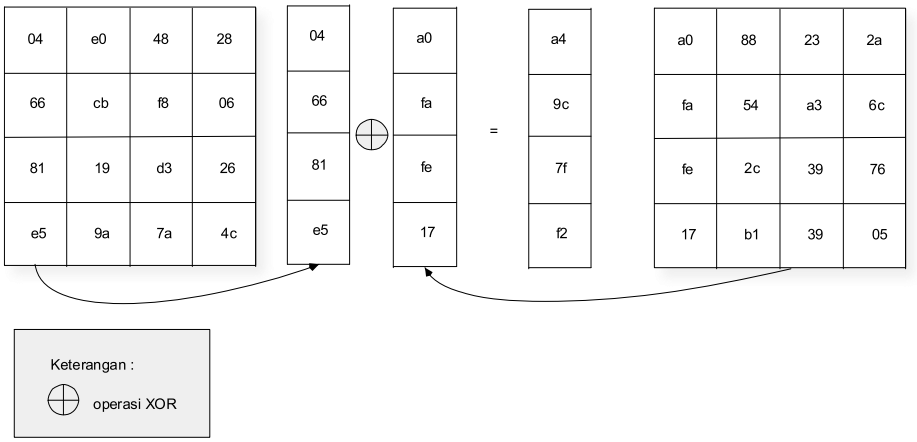
04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

Gambar 3.14 *Array State* Sekarang

a0	88	23	2a
fa	54	a3	6c
fe	2c	39	76
17	b1	39	05

Gambar 3.15 *Round Key*

Berikut hasil dari *array state* dan *round key* yang sudah di transformasi *AddRoundKey*:



Gambar 3.16 Transformasi *AddRoundKey*

a4	68	6b	02
9c	9f	5b	6a
7f	35	ea	50
f2	2b	43	49

Gambar 3.17 Hasil Akhir *AddRoundKey*

8. Setelah selesai melakukan semua langkah diatas maka simpan *file* tersebut sebagai *ciphertek*s (*file* teks yang sudah terenkripsi).

Sebuah *pseudo code* untuk tipe data pada algoritma *Rijndael* akan tampak seperti berikut ini :

```
int main(int argc, char **argv)
{
  unsigned long rk[RKLENGTH(KEYBITS)];
  unsigned char key[KEYLENGTH(KEYBITS)];
  int i;
  int nrounds;
  char *password;
  FILE *output;
  if (argc < 3)
  {
    fputs("Missing argument\n", stderr);
    return 1;
  }
  password = argv[1];
  for (i = 0; i < sizeof(key); i++)
    key[i] = *password != 0 ? *password++ : 0;
  output = fopen(argv[2], "wb");
  if (output == NULL)
  {
    fputs("File write error", stderr);
    return 1;
  }
```

Gambar 3.18 *Pseudo Code* Tipe Data Pada Algoritma *Rijndael*

Sebuah *pseudo code* untuk panjang kunci pada algoritma *Rijndael* akan tampak seperti berikut ini :

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
  word temp

  i = 0

  while (i < Nk)
    w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
    i = i+1
  end while

  i = Nk

  while (i < Nb * (Nr+1)]
    temp = w[i-1]
    if (i mod Nk = 0)
       temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
    else if (Nk > 6 and i mod Nk = 4)
       temp = SubWord(temp)
    end if
    w[i] = w[i-Nk] xor temp
    i = i + 1
  end while
end
```

Gambar 3.19 *Pseudo Code* Panjang Kunci Pada Algoritma *Rijndael*

Sebuah *pseudo code* untuk proses enkripsi pada algoritma *Rijndael* akan tampak seperti berikut ini :

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])

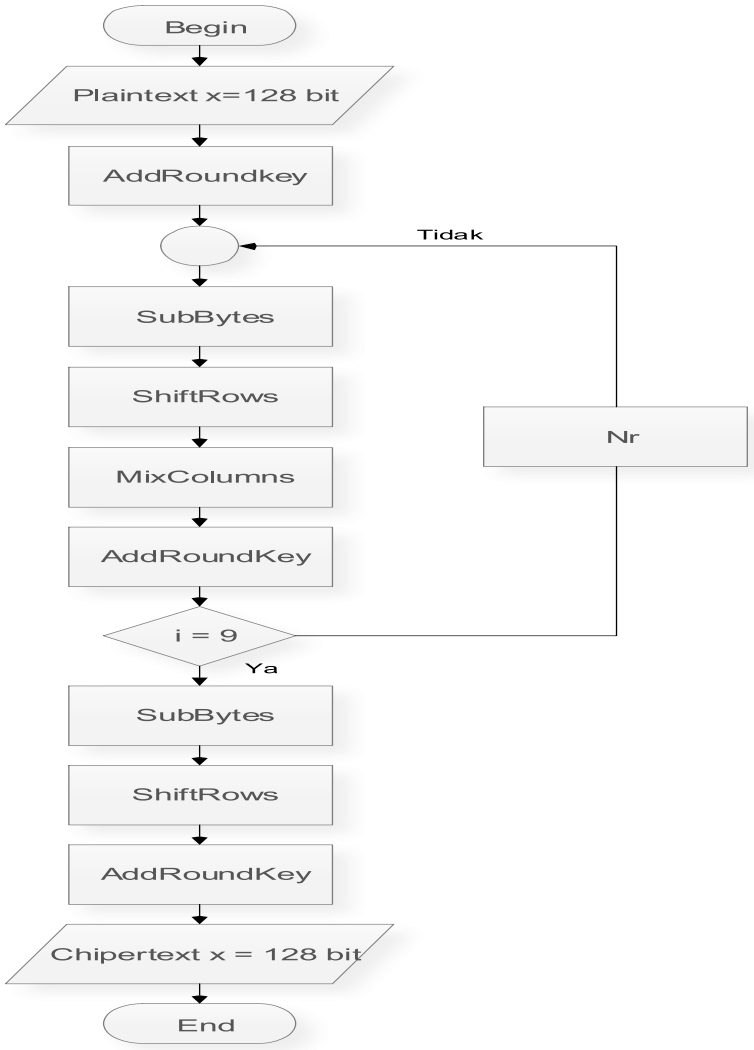
   for round = 1 step 1 to Nr-1
     SubBytes(state)
     ShiftRows(state)
     MixColumns(state)
     AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Gambar 3.20 *Pseudo Code* Proses Enkripsi Algoritma *Rijndael*

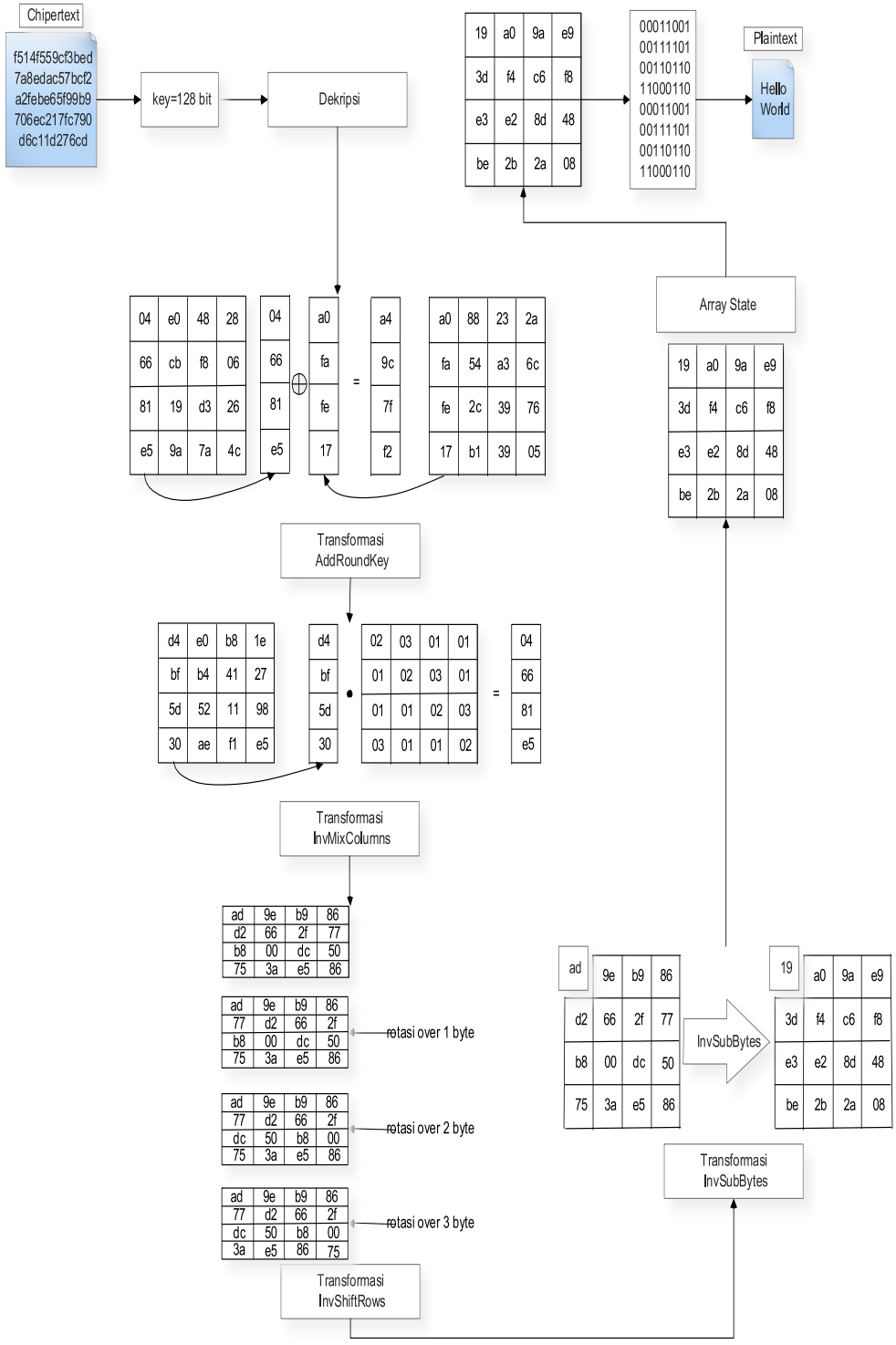
Sebuah *flowchart* untuk proses enkripsi pada algoritma *Rijndael* akan tampak seperti berikut ini:



Gambar 3.21 *Flowchart* Proses Enkripsi Algoritma *Rijndael*

Bila dilihat dari langkah-langkah diatas untuk melakukan kriptografi atau penyandian pesan dengan menggunakan metoda *Rijndael* adalah lakukan kalkulasi pada *plainteks* menjadi *cipherteks* dari data yang akan disimpan di dalam *array of byte* dua dimensi. Tiap elemen dari *array state* tersebut harus diisi dengan 8 *bit* teks (1 *byte*) dalam notasi *HEXADESIMAL*. Setelah pengisian *array*

state selesai, lakukan transformasi *SubBytes* yang dimana transformasi *byte* pada setiap elemen pada *state* akan dipetakan dengan menggunakan sebuah tabel substitusi (*S-Box*). Setelah selesai tahap transformasi *SubBytes*, lakukan transformasi *ShiftRows* yang berfungsi sebagaiproses pergeseran *bit* dimana *bit* paling kiri akan dipindahkan menjadi *bit* paling kanan (rotasi *bit*). Setelah proses rotasi *bit* sudah selesai, lakukan pengacakan *array state* dengan melakukan perkalian matriks menggunakan *galois field*. Setelah perhitungan perkalian matriks dengan *galois field* selesai, lakukan penambahan pada *state* dengan operasi *XOR*. Setelah selesai melakukan semua prosesmaka *output* yang dihasilkan adalah *file* teks yang sudah menjadi *cipherteks*.



Gambar 3.22 Proses Dekripsi *Rijndael*

Pada proses dekripsi pesan terdapat tugas-tugas seperti berikut :

1. *Load file* teks yang telah dienkripsi dengan menggunakan algoritma *Rijndael* untuk di dekripsi agar mudah dibaca pesan yang tersimpan didalamnya.
2. Pada proses enkripsi dan dekripsi *Rijndael* proses *Key Schedule* sama, mengambil kunci *cipher* dan melakukan rutin ekspansi kunci (*key expansion*) untuk membentuk *key schedule*. Ekspansi kunci menghasilkan total Nb(Nr+1) *word*. Algoritma ini membutuhkan set awal *key* yang terdiri dari Nb *word*, dan setiap *round* Nr membutuhkan data kunci sebanyak Nb *word*. Hasil *key schedule* terdiri dari *array* 4 *byte word* linear yang dinotasikan dengan [wi]. *SubWord* adalah fungsi yang mengambil 4 *byte word* input dan mengaplikasikan *S-Box* ke tiap-tiap data 4 *byte* untuk menghasilkan *word output*. Fungsi *RotWord* mengambil *word* [a0, a1, a2, a3] sebagai *input*, melakukan permutasi siklik, dan mengembalikan *word* [a1, a2, a3, a0]. *Rcon*[i] terdiri dari nilai-nilai yang diberikan oleh [xi-1, {00}, {00}, {00}], dengan xi-1 sebagai pangkat dari x (x dinotasikan sebagai {02}).
3. Pada proses enkripsi dan dekripsi *Rijndael* proses *AddRoundKey* sama, sebuah *round key* ditambahkan pada *state* dengan operasi *XOR*. Setiap *round key* terdiri dari Nb *word* dimana tiap *word* tersebut akan dijumlahkan dengan *word* atau kolom yang bersesuaian dari *state* sehingga :

$$[S'_{0,c},S'_{1,c},S'_{2,c},S'_{3,c}] = [S_{0,c},S_{1,c},S_{2,c},S_{3,c}] \oplus [W_{round*Nb+c}] \text{ untuk } 0 \leq c \leq Nb$$

[Wi] adalah word dari key yang bersesuaian dimana :

$$i = round*Nb+c$$

Transformasi *AddRoundKey* pada proses enkripsi pertama kali pada *round* = 0 untuk *round* selanjutnya *round*= *round* + 1.

4. Setelah melakukan proses *AddRoundKey*, lakukan pada setiap kolom dalam *state* dikalikan dengan matriks perkalian dalam AES. Perkalian dalam matriks dapat dituliskan :

$$\begin{bmatrix} S'0,c \\ S'1,c \\ S'2,c \\ S'3,c \end{bmatrix} = \begin{bmatrix} 0E\ 0B\ 0D\ 09 \\ 09\ 0E\ 0B\ OD \\ 0D\ 09\ 0E\ 0B \\ 0B\ OD\ 09\ 0E \end{bmatrix} \begin{bmatrix} S0,c \\ S1,c \\ S2,c \\ S3,c \end{bmatrix}$$

Hasil dari perkalian dalam matriks adalah :

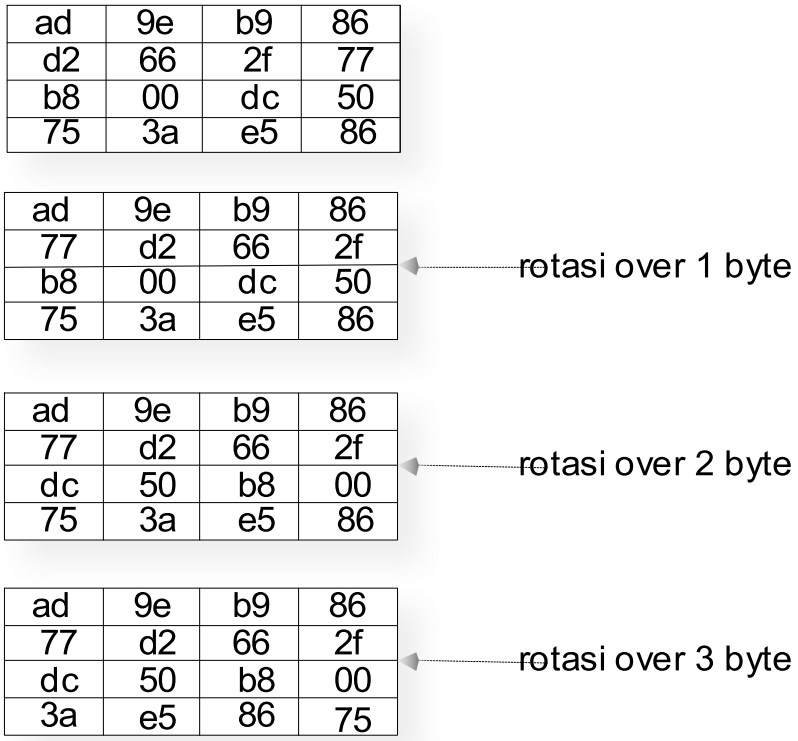
$$S'0,c = (\{0E\} \bullet S\,0,c) \oplus (\{oB\} \bullet S\,1,c) \oplus (\{oD\} \bullet S\,2,c) \oplus (\{o9\} \bullet S\,3,c)$$

$$S'1,c = (\{09\} \bullet S\,0,c) \oplus (\{oE\} \bullet S\,1,c) \oplus (\{oB\} \bullet S\,2,c) \oplus (\{oD\} \bullet S\,3,c)$$

$$S'2,c = (\{0D\} \bullet S\,0,c) \oplus (\{o9\} \bullet S\,1,c) \oplus (\{oE\} \bullet S\,2,c) \oplus (\{oB\} \bullet S\,3,c)$$

$$S'3,c = (\{0B\} \bullet S\,0,c) \oplus (\{oD\} \bullet S\,1,c) \oplus (\{o9\} \bullet S\,2,c) \oplus (\{oE\} \bullet S\,3,c)$$

5. Setelah melakukan perhitungan pada perkalian matriks di *InvMixColumns* dilanjutkan transformasi *byte* yang berkebalikan dengan transformasi *ShiftRows*, karena pada transformasi *InvShiftRows* dilakukan pergeseran *bit* ke kanan.

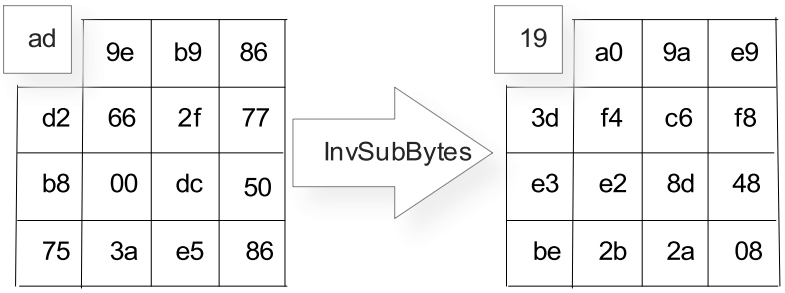


Gambar 3.23 Transformasi *InvShiftRows*

6. Dilanjutkan proses transformasi *InvSubBytes* dimana transformasi *bytes* yang berkebalikan dengan transformasi *SubBytes*, karena pada *InvSubBytes* tiap elemen pada *state* dipetakan dengan menggunakan tabel *Inverse S-Box* yang ditunjukan dalam Tabel 3.2 berikut :

Tabel 3.2 *Inverse S-Box*

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xa	xb	xc	xd	xe	xf
0x	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1x	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2x	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3x	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4x	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5x	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6x	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7x	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8x	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9x	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
ax	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
bx	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
cx	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
dx	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
ex	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
fx	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d



Gambar 3.24 Transformasi *InvSubBytes*

7. Setelah transformasi *InvSubBytes* sudah dilakukan, maka hasil *array state* yang awal enkripsi akan muncul kembali untuk dijadikan status awal dan dikalkulasikan c*ipherteks* menjadi *plainteks* kembali.
8. Setelah selesai melakukan semua langkah diatas maka simpan *file* tersebut sebagai *plainteks* (*file* teks yang sudah didekripsi).

Sebuah *pseudo code* untuk proses dekripsi pada algoritma *Rijndael* akan tampak seperti berikut ini :

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte state[4,Nb]

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   for round = Nr-1 step -1 downto 1
     InvShiftRows(state)
     InvSubBytes(state)
     AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
     InvMixColumns(state)
   end for

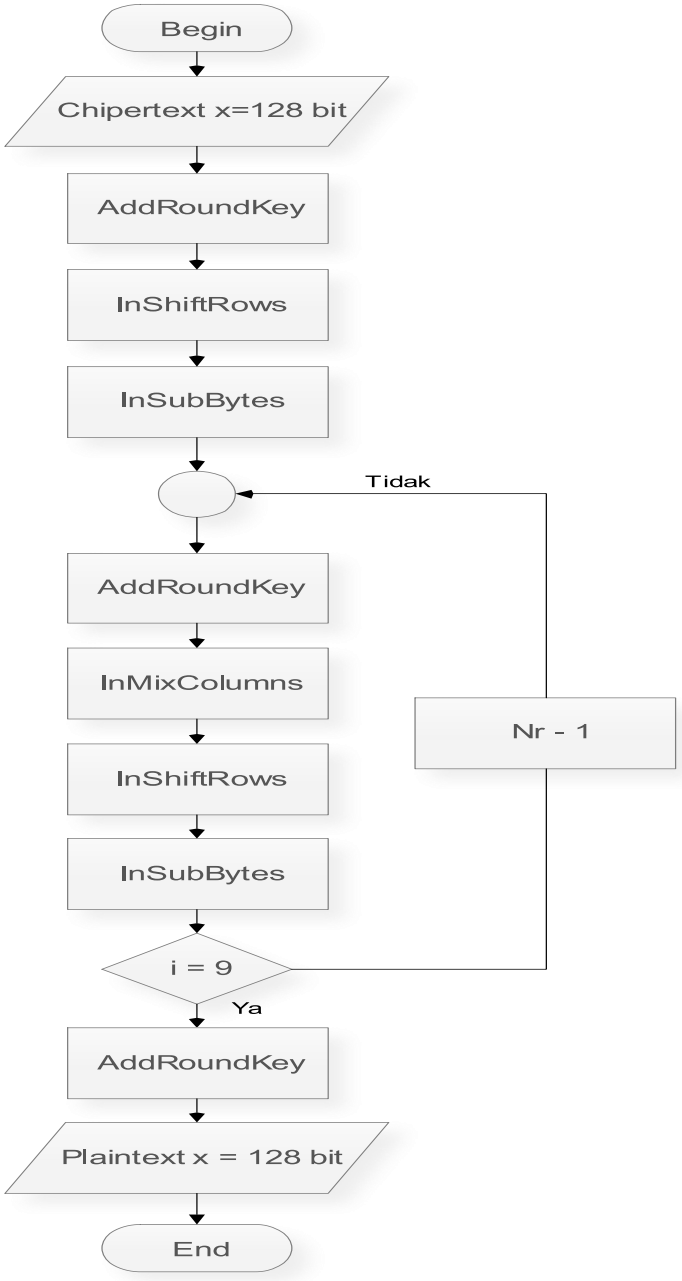
   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])

   out = state
end
```

Gambar 3.25 *Pseudo Code* Proses Dekripsi Algoritma *Rijndael*

\

Sebuah *flowchart* untuk proses dekripsi pada algoritma *Rijndael* akan tampak seperti berikut ini :



Gambar 3.26 *Flowchart* Proses Dekripsi Algoritma *Rijndael*

3.1.2.1.3 Kompleksitas Waktu Algoritma Rijndael

1. Operasi pengisian nilai

   jumlah ← 4, 1 kali

   k ← 1, 1 kali

   jumlah ← 4 + 9, 9*n* kali

   k ← k + 1, *n* kali

   r ← 13/9 1 kali

   Jumlah seluruh operasi pengisian nilai adalah

   $t_1 = 1 + 1 + 9n + n + 1 = 3 + 10n$

2. Operasi penjumlahan

   4 + 9, 9*n* kali

   k + 1 *n* kali

   jumlah seluruh operasi penjumlahan adalah

   $t_2 = 9n + n = 10n$

3. Operasi pembagian

   13/9 1 kali

Dengan demikian, kompleksitas waktu algoritma dihitung berdasarkan jumlah operasi aritmetika dan operasi pengisian nilai adalah

$T(n) = t_1 + t_2 + t_3 = 3 + 10n + 10n + 1 = 20n + 4$

3.1.2.2 Algoritma *Twofish*

Twofish merupakan algoritma yang beroperasi dalam mode blok. Algoritma *Twofish* sendiri merupakan pengembangan dari algoritma *Blowfish*. *Twofish* menggunakan sebuah Struktur *Feistel-like 16-round* dengan tambahan

whitening pada masukan dan keluaran. Satu-satunya unsur *non-Feistel* adalah 1-*bit* rotasi. Perputaran dapat dipindah ke dalam fungsi F untuk membuat suatu struktur *Feistel* murni, tapi memerlukan suatu tambahan perputaran kata-kata yang tepat sebelum langkah keluaran *whitening*.

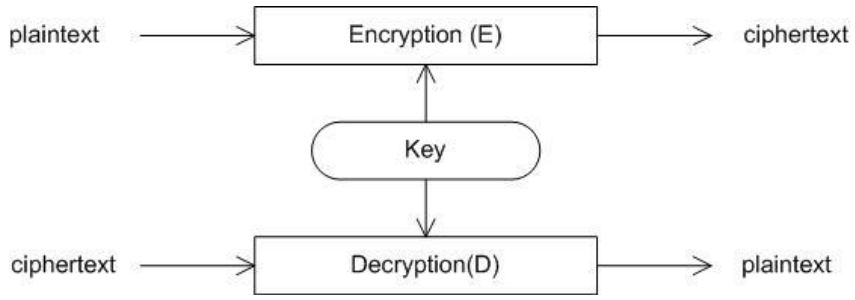
3.1.2.2.1 Panjang Kunci (*Key*) Algoritma *Twofish*

Tingkat keamanan algoritma kunci simetris tipe cipher blok dapat diukur dari tingkat kerumitan algoritma, panjang blok yang digunakan, panjang kunci yang digunakan dan tingkat pengacakan plainteks terhadap cipherteks. Algoritma *twofish* menggunakan jaringan *feistel* dan kotak-S dalam implementasinya. Karena itu tingkat keamanan algoritma ini juga dipengaruhi oleh cara penjadwalan kunci internal dan cara pembangkitan kunci-S.

Kotak-S digunakan algoritma *twofish* dalam fungsi *f* pada jaringan *feistel*. Cara pembangkitan kotak-S ini mempengaruhi tingkat kerumitan pada fungsi *f* tesebut. Ada dua macam pendekatan dalam pembangkitan kotak-S ini. Pertama adalah pembangkitan kotak-S secara statis ini berarti kotak-S yang digunakan tidak bergantung pada plainteks dan kunci yang dimasukkan. Pendekatan kedua adalah pembangkitan kotak-S secara dinamis. Pembangkit dinamis ini biasanya diimplementasikan dengan menggunakan fungsi bilangan acak.

3.1.2.2.2 Kunci Simetri Algoritma *Twofish*

Untuk algoritma *twofish* ini sendiri termasuk kriptografi tipe *cipher* blok dan memiliki kunci simetris dimana proses dekripsi adalah kebalikan dari proses enkripsi dan menggunakan kunci yang sama. Pada gambar dibawah ini menunjukan proses dari kunci simetris itu sendiri.

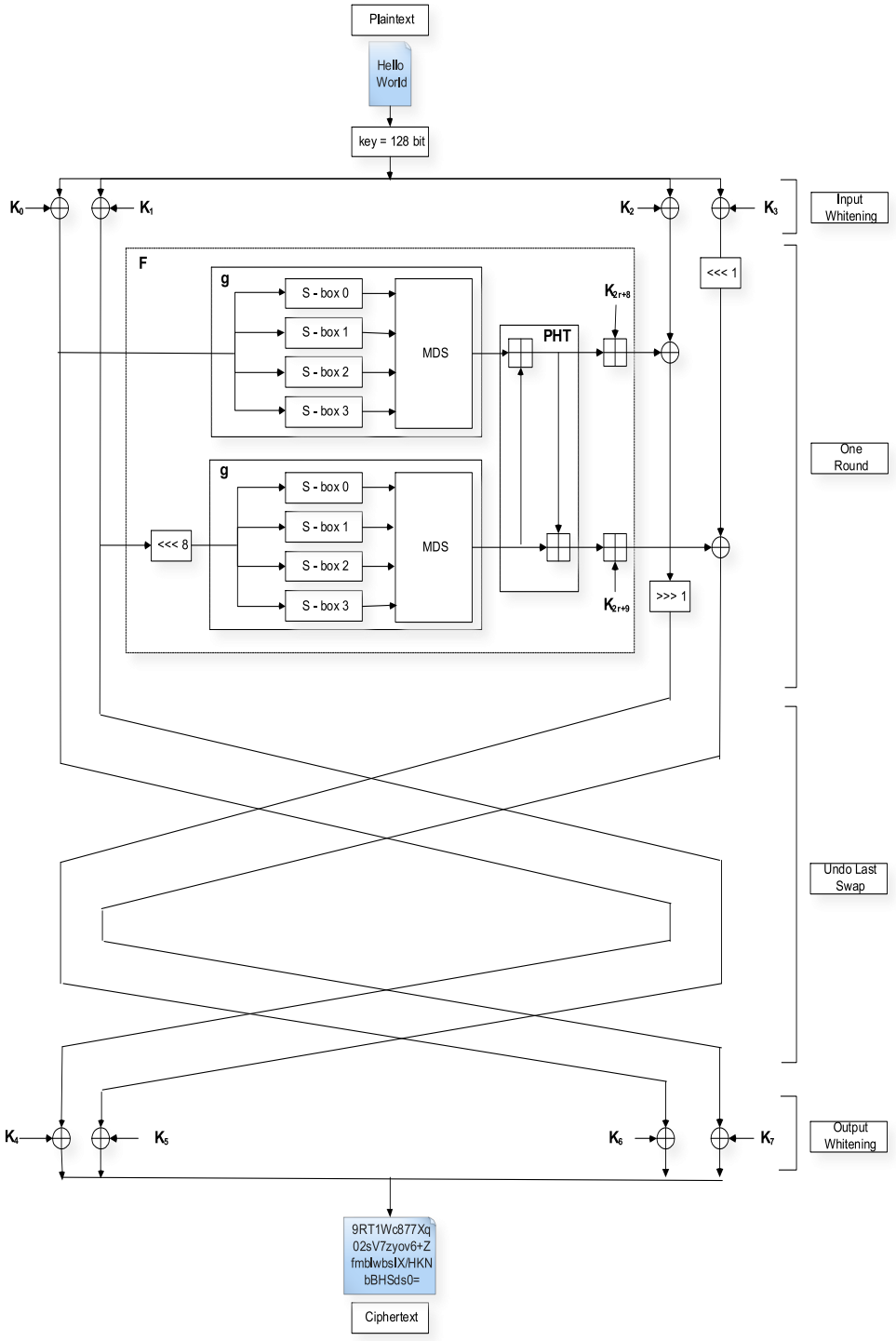


Gambar 3.27 Kriptografi Kunci Simetris *Twofish*

Kriptografi simetris dapat dibagi menjadi dua, yaitu penyandian blok dan penyandian alir. Penyandian blok bekerja pada suatu data yang terkelompok menjadi blok-blok data atau kelompok data dengan panjang data yang telah ditentukan. Pada penyandian blok, data yang masuk akan dipecah-pecah menjadi blok data yang telah ditentukan ukurannya. Penyandian alir bekerja pada suatu data bit tunggal atau terkadang dalam satu *byte*. Jadi format data yang mengalami proses enkripsi dan dekripsi adalah berupa aliran bit-bit data.

Algoritma *Twofish* yang ada pada saat ini kebanyakan bekerja untuk penyandian blok karena kebanyakan proses pengiriman data pada saat ini menggunakan blok-blok data yang telah ditentukan ukurannya untuk kemudian dikirim melalui jaringan *Local Area Network* (*LAN*).

Cara penggunaan metode *Twofish* pada kriptografi berupa *file* teks seperti berikut:



Gambar 3.28 Proses Enkripsi *Twofish*

Pada proses enkripsi pesan terdapat tugas-tugas seperti berikut :

1. *Load file* teks yang akan dienkripsi. Format teks yang digunakanadalah *.txt / .doc / .pdf*. Sedangkan untuk ukuran *file*nya tidak ada batasannya, semakin besar ukuran *filenya* maka semakin banyak pula pesan yang dapat dienkripsi.
2. *Input plainteks* sebesar 128 *bit* akan dibagi menjadi empat *word* yaitu P0, P1, P2, P3 yang masing-masing sebesar 32 *bit*. P0 dan P1 akan menjadi bagian kiri, sedangkan P2 dan P3 akan menjadi bagian kanan. *Plainteks* tersebut dipecah menjadi empat kata sebanyak 32-*bit* dengan menggunakan konversi *little endian*.

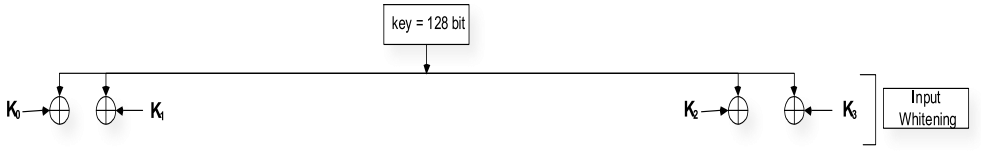
$$P_i = \sum_{j=0}^{3} P(4_i + j)\ 2^{8\,j}$$

dimana i=0,…,3.

3. *Plainteks* akan melalui proses *input whitening* yaitu *input* akan di-*XOR* dengan empat *word* kunci yang telah terjadwal yaitu K0, K1, K2, dan K3. Secara formalnya adalah sebagai berikut :

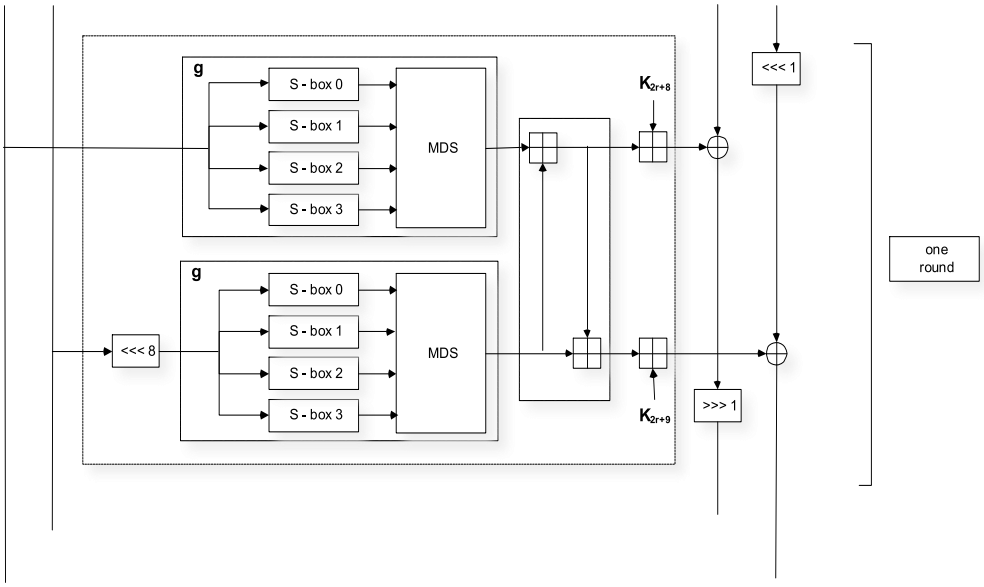
$$R_{o,i} = P_i \text{ xor } K_i$$

dimana i=0,…,3.



Gambar 3.29 Proses *Whitening*

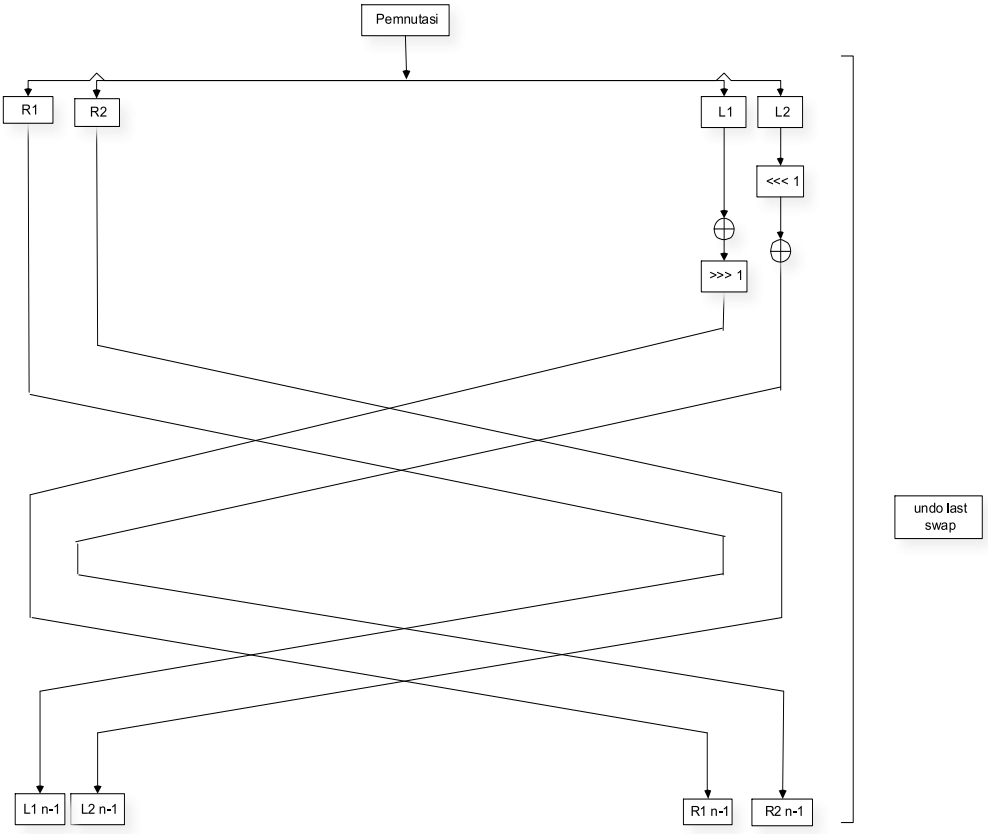
4. Proses berikutnya *input* akan melalui proses pada fungsi *F* yang meliputi didalamnya adalah fungsi *g* dan dilanjutkan dengan *PHT* (*pseudo hadamard transform*), dan dilakukan penambahan hasil *PHT* dengan kunci. Proses fungsi *F* tersebut dilakukan secara bertahap. *R0* dan *R1* yang merupakan hasil *whitening* akan menjadi *input* untuk fungsi *F*.
5. *R0* dan *R1* akan dimasukkan ke dalam fungsi *g* yang merupakan bagian awal dari fungsi *F*. Untuk *R1* sebelum dimasukkan ke dalam fungsi *g* akan dirotasi ke kiri sejauh 8 *bit*. *R0* dan *R1* melalui *S-box* dan selanjutnya akan dikalikan dengan matriks *MDS*. Hasil dari fungsi *g* ini masing-masing menjadi *T0* dan *T1*.
6. *T0* dan *T1* akan melalui proses *PHT* yang merupakan penggabungan *T0* dan *T1* dimana *T0 + T1* dan *T0 + 2T1*. Setelah itu hasil dari *PHT* tersebut masing-masing akan ditambahkan dengan kunci yang sudah terjadwal yaitu *K2r+8* dan *K2r+9*. Hasil dari fungsi *F* adalah *F0* dan *F1*, maka dengan demikian fungsi *F* telah terpenuhi.



Gambar 3.30 Fungsi *F*

Dari Gambar 3.30 Fungsi *F* dapat dilihat bahwa untuk mengambil tiga argumentasi, dua kata masukan *R0* dan *R1*, dan angka bulat *r* digunakan untuk memilih *subkey* yang sesuai. Hasil *whitening* akan melalui fungsi *F* yang mempunyai *output*F0 dan F1 dan masing-masing di-*XOR*kan dengan *R2* dan *R3* (melalui rotasi ke kiri 1 *bit*).

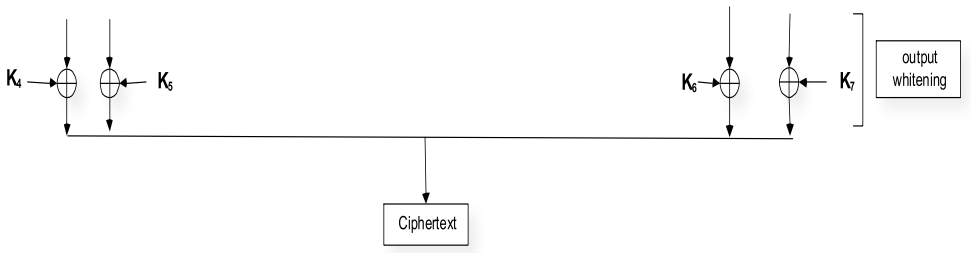
7. Setelah itu *F0* dan *F1* masing-masing di-*XOR* dengan *R2* dan *R3*. Hasil dari *R2 XOR F0* dirotasi ke kanan sejauh 1 *bit*. Sedangkan *R3 XOR F1* , sebelumnya *R3* dirotasi ke kiri sejauh 1 *bit*.
8. Setelah itu, maka akan dilakukan iterasi sebanyak 16 kali. Setiap iterasi sama dengan proses sebelumnya.
9. Hasil dari *swap* blok terakhir adalah penukaran bagian kanan dan kiri yang di-*undo*.



Gambar 3.31 *Swap* Blok Terakhir

Dari Gambar 3.31 dapat dilihat bahwa ada penukaran pada bagian kanan dan kiri yang di argumen pertama (32 *bit word*) disebelah kiri atau kanan dengan jumlah *bit* yang diindikasikan dengan argumen kedua.

10. Hasil dari 16 *round* enkripsi akan melalui *output whitening* yaitu proses peng-*XOR*an 16 *round* enkripsi dengan *K4*, *K5*, *K6*, dan *K7*.



Gambar 3.32 *Output Whitening*

Dari Gambar 3.32 dapat dilihat bahwa setelah proses pertukaran pada bagian kanan dan kiri yang di *undo*, lalu memasuki tahap *output whitening* yang merupakan proses peng-*XOR*an 16 *round* enkripsi.

Sebuah *pseudo code* untuk tipe data pada algoritma *Twofish* akan tampak seperti berikut ini :

```
Public Function typedata(Text As String, Optional Key As String)
As String
    Dim byteArray() As Byte
    byteArray() = StrConv(Text, vbFromUnicode)
    Call typedata(byteArray(), Key)
    typedata = StrConv(byteArray(), vbUnicode)
End Function
```

Gambar 3.33 *Pseudo Code* Tipe Data Pada Algoritma *Twofish*

Sebuah *pseudo code* untuk panjang kunci pada algoritma *Twofish* akan tampak seperti berikut ini :

```
    Dim KeyLength As Long, Key() As Byte
    KeyLength = Len(New_Value) * 8
    Key() = StrConv(New_Value, vbFromUnicode)
    If (KeyLength < MinKeyLength) Then
        ReDim Preserve Key(MinKeyLength \ 8 - 1)
        KeyLength = MinKeyLength
    End If
    If (KeyLength = 128) Then
        ReDim Preserve Key(15)
        KeyLength = 128
    End If
    Call KeyCreate(Key, KeyLength \ 8)
```

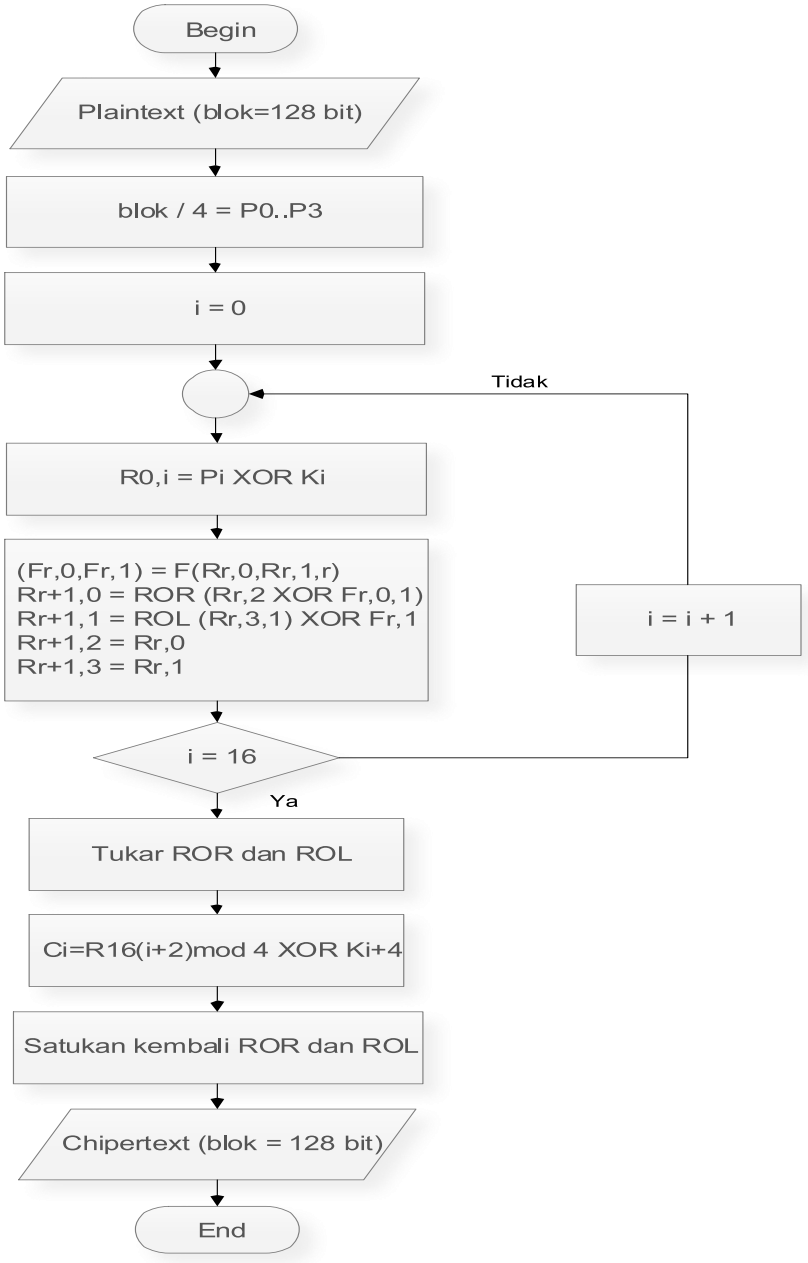
Gambar 3.34 *Pseudo Code* Panjang Kunci Pada Algoritma *Twofish*

Sebuah *pseudo code* untuk proses enkripsi pada algoritma *Twofish* akan tampak seperti berikut ini :

```
    DWord(0) = DWord(0) Xor sKey(INPUT_WHITEN)
    DWord(1) = DWord(1) Xor sKey(INPUT_WHITEN + 1)
    DWord(2) = DWord(2) Xor sKey(INPUT_WHITEN + 2)
    DWord(3) = DWord(3) Xor sKey(INPUT_WHITEN + 3)
    K = ROUND_SUBKEYS
    For r = 0 To (Rounds - 1) Step 2
        If (m_RunningCompiled) Then
            t0 = Fe32(DWord(0), 0)
            t1 = Fe32(DWord(1), 3)
            t0 = t0 + t1
            DWord(2) = Rot1(DWord(2) Xor (t0 + sKey(K)))
            K = K + 1
            DWord(3) = Rot31(DWord(3)) Xor (t0 + t1 +
sKey(K))
            K = K + 1
            t0 = Fe32(DWord(2), 0)
            t1 = Fe32(DWord(3), 3)
            t0 = t0 + t1
            DWord(0) = Rot1(DWord(0) Xor (t0 + sKey(K)))
            K = K + 1
            DWord(1) = Rot31(DWord(1)) Xor (t0 + t1 +
sKey(K))
            K = K + 1
        Else
            t0 = Fe32(DWord(0), 0)
            t1 = Fe32(DWord(1), 3)
            t0 = UnsignedAdd(t0, t1)
            DWord(2) = Rot1(DWord(2) Xor (UnsignedAdd(t0,
sKey(K))))
            K = K + 1
            DWord(3) = Rot31(DWord(3)) Xor
(UnsignedAdd(UnsignedAdd(t0, t1), sKey(K)))
            K = K + 1
            t0 = Fe32(DWord(2), 0)
            t1 = Fe32(DWord(3), 3)
            t0 = UnsignedAdd(t0, t1)
            DWord(0) = Rot1(DWord(0) Xor (UnsignedAdd(t0,
sKey(K))))
            K = K + 1
            DWord(1) = Rot31(DWord(1)) Xor
(UnsignedAdd(UnsignedAdd(t0, t1), sKey(K)))
            K = K + 1
        End If
    Next
    DWord(2) = DWord(2) Xor sKey(OUTPUT_WHITEN)
    DWord(3) = DWord(3) Xor sKey(OUTPUT_WHITEN + 1)
    DWord(4) = DWord(0) Xor sKey(OUTPUT_WHITEN + 2)
    DWord(5) = DWord(1) Xor sKey(OUTPUT_WHITEN + 3)
    Call CopyMem(DWord(0), DWord(2), 16)
```

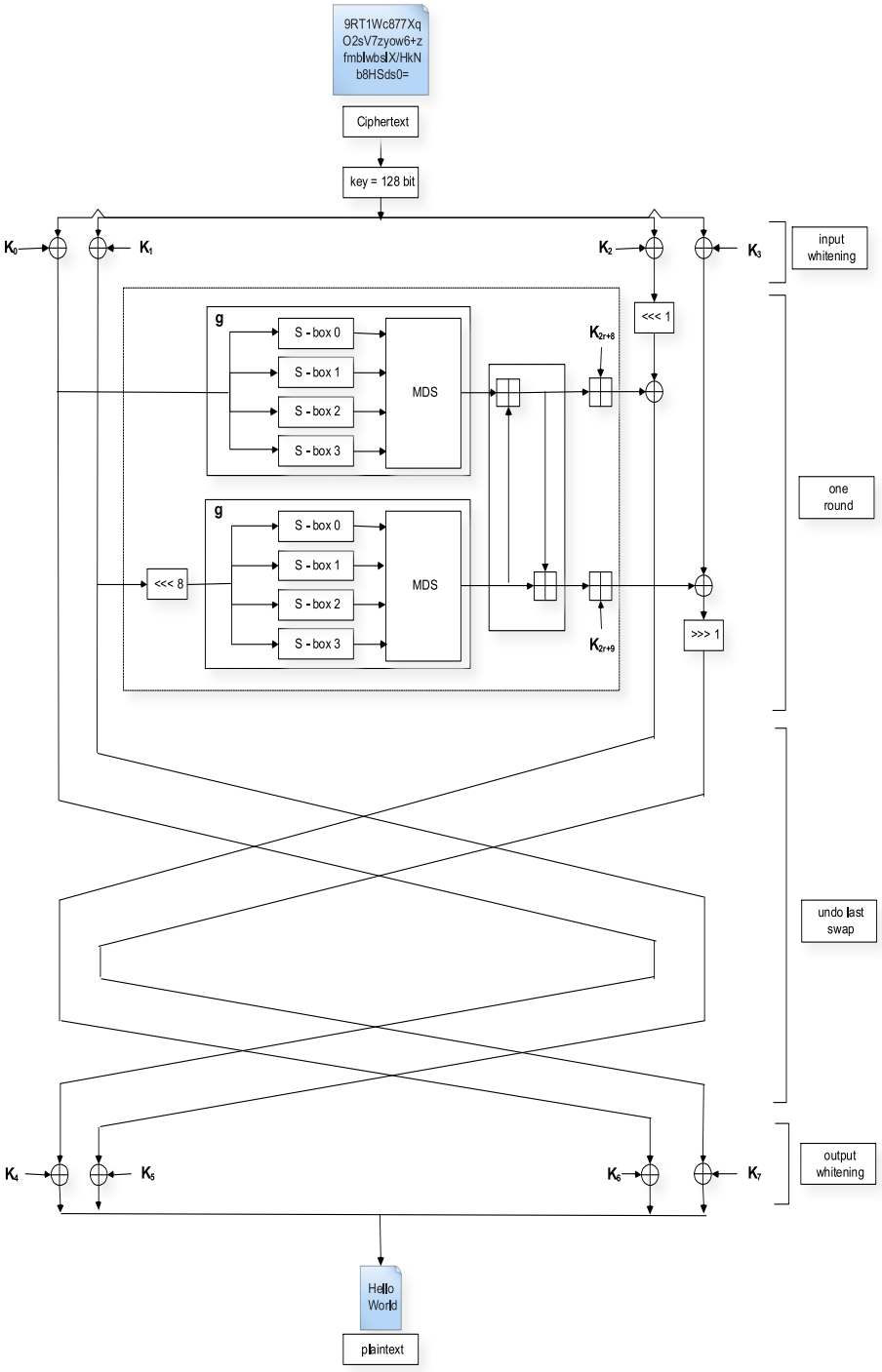
Gambar 3.35 *Pseudo Code* Proses Enkripsi Algoritma *Twofish*

Sebuah *flowchart* untuk proses enkripsi pada algoritma *Twofish* akan tampak seperti berikut ini :



Gambar 3.36 *Flowchart* Proses Enkripsi Algoritma *Twofish*

Bila dilihat dari langkah-langkah diatas untuk melakukan kriptografi atau penyandian pesan dengan menggunakan metoda *Twofish* adalah *plaintext* dipecah menjadi empat kata 32-*bit*. Pada langkah *whitening* masukan terdapat *XOR* dengan empat kata kunci. Ini diikuti oleh enambelas putaran. Pada setiap putaran, dua kata-kata pada sisi kiri digunakan sebagai masukan kepada fungsi *g* (Salah satu darinya diputar pada 8 *bit* pertama). Fungsi *g* terdiri dari empat *byte-wide S-Box key-dependent*, yang diikuti oleh suatu langkah pencampuran linier berdasar pada suatu matriks *MDS*. Hasil kedua fungsi *g* dikombinasikan menggunakan suatu *Pseudo Hadamard Transform (PHT)*, dan ditambahkan dua kata kunci. Kedua hasil ini kemudian di-*XOR* ke dalam kata-kata pada sisi kanan (salah satunya diputar ke kanan 1 *bit* pertama, yang lainnya diputar ke kiri setelahnya). Yang kiri dan kanan dibelah dua kemudian ditukar untuk putaran yang berikutnya, pertukaran yang terakhir adalah dibalik, dan yang empat kata di-*XOR* dengan lebih dari empat kata kunci untuk menghasilkan *ciphertext*.



Gambar 3.37 Proses Dekripsi *Twofish*

Pada proses enkripsi pesan terdapat tugas-tugas seperti berikut :

1. *Load file* teks yang telah dienkripsidengan menggunakan algoritma *Twofish* untuk di dekripsi agar mudah dibaca pesan yang tersimpan didalamnya.
2. *Cipherteks* sebesar 128 *bit* akan dibagi menjadi empat *word* yaitu P0, P1, P2, P3 yang masing-masing sebesar 32 *bit*. P0 dan P1 akan menjadi bagian kiri, sedangkan P2 dan P3 akan menjadi bagian kanan. Plainteks tersebut dipecah menjadi empat kata sebanyak 32-*bit* dengan menggunakan konversi *little endian*.

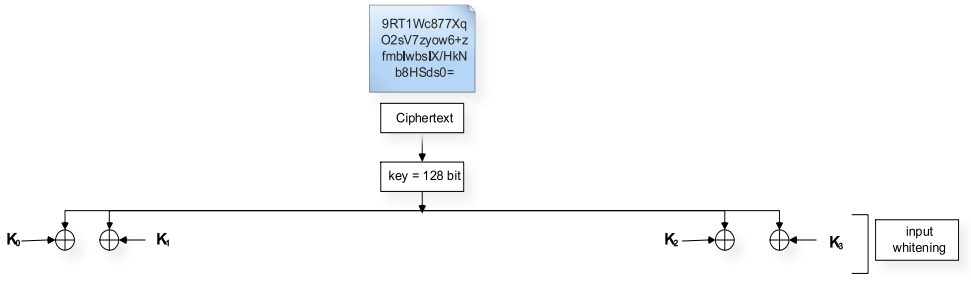
$$P_i = \sum_{j=0}^{3} P(4_i + j)\ 2^{8\,j}$$

dimana i=0,…,3.

3. Pada *ciphertext* akan melalui proses *input whitening* yaitu input akan di-*XOR* dengan empat *word* kunci yang telah terjadwal yaitu K0, K1, K2, dan K3. Secara formalnya adalah sebagai berikut :

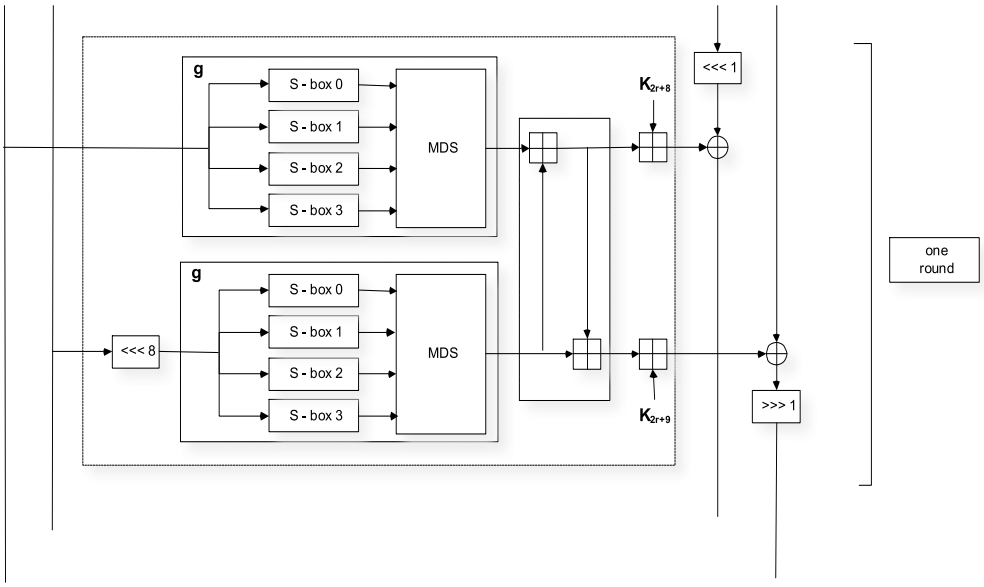
$$Ro,i = Pi \text{ xor } Ki$$

dimana i=0,…,3.



Gambar 3.38 Proses *Input Whitening*

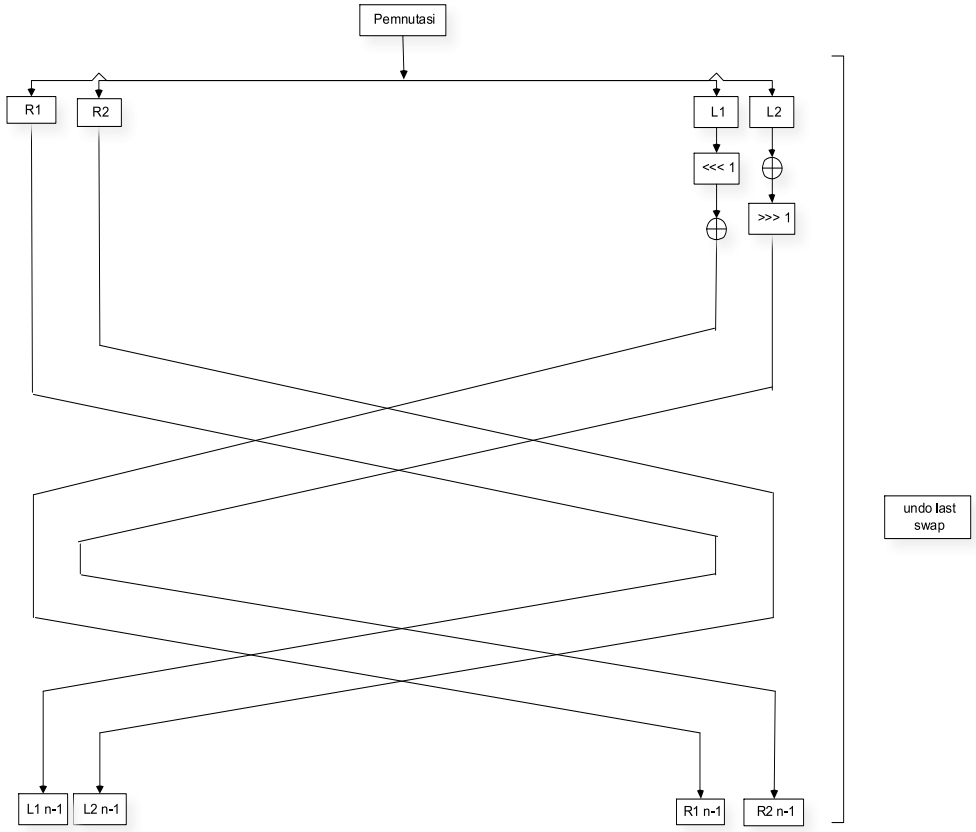
4. Proses berikutnya *input* akan melalui proses pada fungsi *F* yang meliputi didalamnya adalah fungsi *g* dan dilanjutkan dengan *PHT (pseudo hadamard transform)*, dan dilakukan penambahan hasil *PHT* dengan kunci. Proses fungsi *F* tersebut dilakukan secara bertahap. *R0* dan *R1* yang merupakan hasil *whitening* akan menjadi *input* untuk fungsi *F*.
5. *R0* dan *R1* akan dimasukkan ke dalam fungsi *g* yang merupakan bagian awal dari fungsi *F*. Untuk *R1* sebelum dimasukkan ke dalam fungsi *g* akan dirotasi ke kiri sejauh 8 *bit. R0* dan *R1* melalui *S-box* dan selanjutnya akan dikalikan dengan matriks *MDS*. Hasil dari fungsi *g* ini masing-masing menjadi *T0* dan *T1*.
6. *T0* dan *T1* akan melalui proses *PHT* yang merupakan penggabungan *T0* dan *T1* dimana *T0 + T1* dan *T0 + 2T1*. Setelah itu hasil dari PHT tersebut masing-masing akan ditambahkan dengan kunci yang sudah terjadwal yaitu *K2r+8* dan *K2r+9*. Hasil dari fungsi *F* adalah *F0* dan *F1*, maka dengan demikian fungsi *F* telah terpenuhi.



Gambar 3.39 Proses Rotasi Pada Fungsi *F*

Dari Gambar 3.39 Proses Rotasi Pada Fungsi *F* dapat dilihat bahwa untuk mengambil tiga argumentasi, dua kata masukan *R0* dan *R1*, dan angka bulat *r* digunakan untuk memilih *subkey* yang sesuai. Hasil *whitening* akan melalui fungsi *F* yang mempunyai *outputF0* dan *F1* dan masing-masing di-*XOR*kan dengan *R2* dan *R3* (melalui rotasi kekanan 1 *bit*).

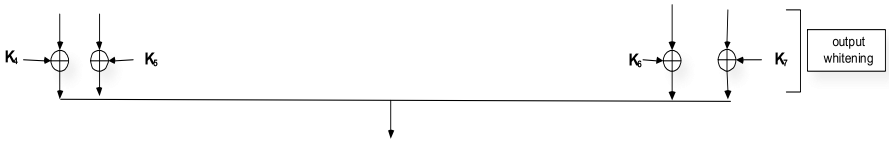
7. Setelah itu *F0* dan *F1* masing-masing di-*XOR* dengan *R2* dan *R3*. Hasil dari *R2 XOR F0* dirotasi ke kiri sejauh 1 *bit*. Sedangkan *R3 XOR F1* , sebelumnya *R3* dirotasi ke kanan sejauh 1 *bit*.
8. Setelah itu, maka akan dilakukan iterasi sebanyak 16 kali. Setiap iterasi sama dengan proses sebelumnya.
9. Hasil dari *swap* blok terakhir adalah penukaran bagian kanan dan kiri yang di-*undo*.



Gambar 3.40 Proses *Undo Last Swap*

Dari Gambar 3.40 dapat dilihat bahwa ada penukaran pada bagian kanan dan kiri yang di argumen pertama (32 *bit word*) disebelah kiri atau kanan dengan jumlah *bit* yang diindikasikan dengan argumen kedua.

10. Hasil dari 16 *round* enkripsi akan melalui *output whitening* yaitu proses peng-*XOR*an 16 *round* enkripsi dengan K4, K5, K6, dan K7.



Gambar 3.41 Proses *Output Whitening*

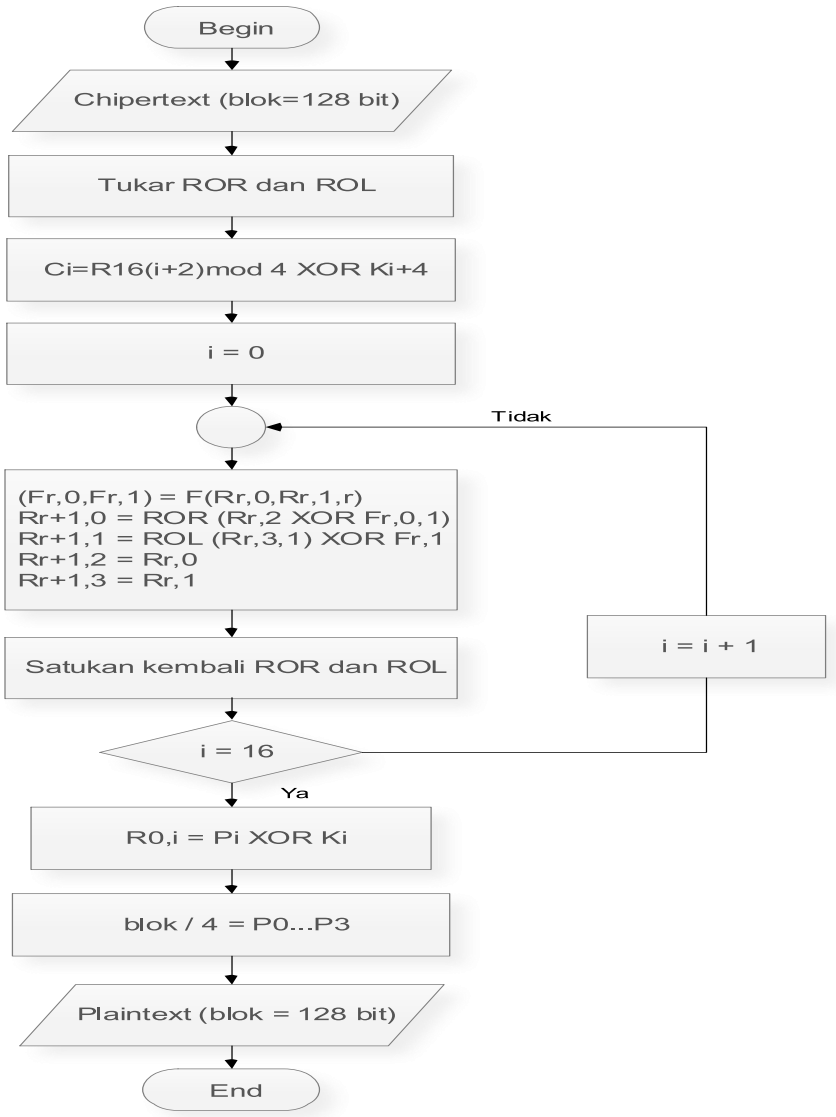
Dari Gambar 3.41 dapat dilihat bahwa setelah proses pertukaran pada bagian kanan dan kiri yang di *undo*, lalu memasuki tahap *output whitening* yang merupakan proses peng-*XOR*an 16 *round* enkripsi.

Sebuah *pseudo code* untukproses dekripsi pada algoritma *Twofish* akan tampak seperti berikut ini :

```
    DWord(2) = DWord(2) Xor sKey(OUTPUT_WHITEN)
    DWord(3) = DWord(3) Xor sKey(OUTPUT_WHITEN + 1)
    DWord(0) = DWord(4) Xor sKey(OUTPUT_WHITEN + 2)
    DWord(1) = DWord(5) Xor sKey(OUTPUT_WHITEN + 3)
    K = ROUND_SUBKEYS + 2 * Rounds - 1
    For r = 0 To Rounds - 1 Step 2
        If (m_RunningCompiled) Then
            t0 = Fe32(DWord(2), 0)
            t1 = Fe32(DWord(3), 3)
            t0 = t0 + t1
            DWord(1) = Rot1(DWord(1) Xor (t0 + t1 + sKey(K)))
            K = K - 1
            DWord(0) = Rot31(DWord(0)) Xor (t0 + sKey(K))
            K = K - 1
            t0 = Fe32(DWord(0), 0)
            t1 = Fe32(DWord(1), 3)
            t0 = t0 + t1
            DWord(3) = Rot1(DWord(3) Xor (t0 + t1 + sKey(K)))
            K = K - 1
            DWord(2) = Rot31(DWord(2)) Xor (t0 + sKey(K))
            K = K - 1
        Else
            t0 = Fe32(DWord(2), 0)
            t1 = Fe32(DWord(3), 3)
            t0 = UnsignedAdd(t0, t1)
            DWord(1) = Rot1(DWord(1) Xor
(UnsignedAdd(UnsignedAdd(t0, t1), sKey(K))))
            K = K - 1
            DWord(0) = Rot31(DWord(0)) Xor (UnsignedAdd(t0,
sKey(K)))
            K = K - 1
            t0 = Fe32(DWord(0), 0)
            t1 = Fe32(DWord(1), 3)
            t0 = UnsignedAdd(t0, t1)
            DWord(3) = Rot1(DWord(3) Xor
(UnsignedAdd(UnsignedAdd(t0, t1), sKey(K))))
            K = K - 1
            DWord(2) = Rot31(DWord(2)) Xor (UnsignedAdd(t0,
sKey(K)))
            K = K - 1
        End If
    Next
    DWord(0) = DWord(0) Xor sKey(INPUT_WHITEN)
    DWord(1) = DWord(1) Xor sKey(INPUT_WHITEN + 1)
    DWord(2) = DWord(2) Xor sKey(INPUT_WHITEN + 2)
    DWord(3) = DWord(3) Xor sKey(INPUT_WHITEN + 3)
```

Gambar 3.42 *Pseudo Code* Proses Dekripsi Algoritma *Twofish*

Sebuah *flowchart* untuk proses dekripsi pada algoritma *Twofish* akan tampak seperti berikut ini :



Gambar 3.43 *Flowchart* Proses Dekripsi Algoritma *Twofish*

Bila dilihat dari langkah-langkah diatas untuk melakukan kriptografi atau penyandian pesan dengan menggunakan metoda *Twofish* adalah *cipherteks* dipecah menjadi empat kata 32-*bit.* Pada langkah *whitening* masukan terdapat *XOR* dengan empat kata kunci. Ini diikuti oleh enambelas putaran. Pada setiap

putaran, dua kata-kata pada sisi kiri digunakan sebagai masukan kepada fungsi *g* (Salah satu darinya diputar pada 8 *bit* pertama). Fungsi *g* terdiri dari empat *byte-wide S-Box key-dependent*, yang diikuti oleh suatu langkah pencampuran linier berdasar pada suatu matriks *MDS*. Hasil kedua fungsi *g* dikombinasikan menggunakan suatu *Pseudo Hadamard Transform (PHT)*, dan ditambahkan dua kata kunci. Kedua hasil ini kemudian di-*XOR* ke dalam kata-kata pada sisi kiri (salah satunya diputar ke kiri 1 *bit* pertama, yang lainnya diputar ke kanan setelahnya). Yang kiri dan kanan dibelah dua kemudian ditukar untuk putaran yang berikutnya, pertukaran yang terakhir adalah dibalik, dan yang empat kata di-*XOR* dengan lebih dari empat kata kunci untuk menghasilkan *plainteks*.

3.1.2.2.3 Kompleksitas Waktu Algoritma Twofish

1. Operasi pengisian nilai

   jumlah ← 3, 1 kali

   k ← 1, 1 kali

   jumlah ← 3 + 16, 16*n* kali

   k ← k + 1, *n* kali

   r ← 19/16 1 kali

   Jumlah seluruh operasi pengisian nilai adalah

   $t_1 = 1 + 1 + 16n + n + 1 = 3 + 17n$

2. Operasi penjumlahan

   3 + 16, 16*n* kali

   k + 1 *n* kali

jumlah seluruh operasi penjumlahan adalah

$t_2 = 16n + n = 17n$

3. Operasi pembagian

13/9 1 kali

Dengan demikian, kompleksitas waktu algoritma dihitung berdasarkan jumlah operasi aritmetika dan operasi pengisian nilai adalah

$T(n) = t_1 + t_2 + t_3 = 3 + 17n + 17n + 1 = 34n + 4$

3.1.2.3 Perbandingan Algoritma *Rijndael* dan Algoritma *Twofish*

Khusus untuk algoritma *Rijndael* dan *Twofish* yang menggunakan kotak-S, terdapat beberapa hal tambahan yang dapat mempengaruhi kekuatan algoritma. Kekuatan kotak-S pada dasarnya didukung oleh penjadwalan kunci internal yang sangat acak dan rumit. Semakin acak hasil penjadwalan kunci internal yang dilakukan akan mempersulit kriptanalisis untuk melakukan penyerangan.

Ada dua macam pendekatan dalam pembangkitan kotak-S ini. Pertama adalah pembangkitan kotak-S secara statis. Pembangkitan secara statis ini berarti kotak-S yang digunakan tidak bergantung pada plainteks dan kunci yang dimasukkan. Pendekatan kedua adalah pembangkitan kotak-S secara dinamis. Pembangkitan dinamis ini biasanya diimplementasikan dengan menggunakan fungsi bilangan acak.Pada algoritma *Rijndael* dan *Twofish*, kotak-S dibangkitkan berdasarkan kunci yang dimasukkan.

Table 3.3 Perbandingan Komponen Keamanan Algoritma *Rijndael* dan *Twofish*

Komponen	*Rijndael*	*Twofish*
Algoritma	Proses perulangan selama 10 putaran; menggunakan kotak-S, teknik *Galois Field, Sub Bytes, Shift Row, Mix Columns, dan Add Round Key*; menggunakan operasi penambahan, *XOR*, matriks, aljabar polinom, dan *lookup table*	Proses perulangan selama 16 putaran; menggunakan jaringan *Feistel*, kotak-S, matriks *MDS*, teknik *Pseudo-Hadamard Transform (PHT)*, dan teknik *whitening*; menggunakan operasi penambahan, *XOR*, *lookup table*, aljabar *vector*, aljabar polinom dan aljabar medan; menggunakan fungsi dengan derajat aljabar yang tinggi (merupakan gabungan dari banyak kelompok aljabar)
Panjang Blok	128 *bit*	128 *bit*
Panjang Kunci	Bervariasi, panjang maksimal 256 *bit*	Bervariasi, panjang maksimal 256 *bit*
Tingkat Pengacakan	Sangat rumit, karena menggunakan teknik *enciphering* selama 10 putaran dan 16 kotak-S yang bergantung pada kunci, teknik *Galois Field, Sub Bytes, Shift Rows, Mix Columns, dan Add Round Key*	Rumit, karena menggunakan jaringan *Feistel* 16 putaran, 4 kotak-S yang bergantung pada kunci, matriks *MDS*, teknik *PHT*, teknik *whitening*

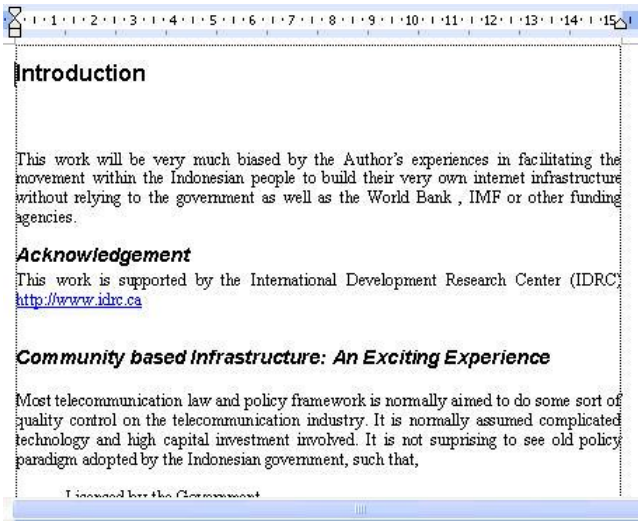
Penjadwalan Kunci Internal	Penjadwalan kunci internal bergantung pada kunci eksternal; cara pembangkitan dengan menggunakan operasi penambahan, *XOR*, matriks, aljabar polinom, dan *lookup table*	Penjadwalan kunci internal bergantung pada kunci eksternal; cara pembangkitan kunci dengan menggunakan operasi penambahan, *XOR*, aljabar vektor, aljabar polinom, dan matriks *MDS*
Pembangkitan kotak-S	Cara pembangkitan kotak-S dilakukan dengan operasi penambahan, *XOR*, teknik *Galois Field, Sub Bytes, Shift Row, Mix Columns, dan Add Round Key*; kotak-S bergantung pada kunci	Cara pembangkitan kotak-S dilakukan dengan operasi matriks *MDS* dan aljabar medan; kotak-S bergantung pada kunci

3.1.3 Analisis Data

Analisis data merupakan tahap dimana dilakukannya analisis terhadap data-data apa saja yang diolah dalam sistem atau prosedur sebuah aplikasi, dalam hal ini data yang akan dienkripsi pada aplikasi kriptografi adalah berupa *file doc, txt,* dan *pdf.*

3.1.3.1 *File Text*

Teks merupakan sekumpulan karakter, huruf-huruf, angka-angka (A-Z, a-z, 0-9) dan simbol-simbol lainnya seperti %, &, ^, =, @, £, $, !, * dan lain-lain dengan menggunakan kode *ASCII* setiap karakter dari teks berjumlah 8 *bit* atau 1 *byte*.



Gambar 3.44 *File* Teks

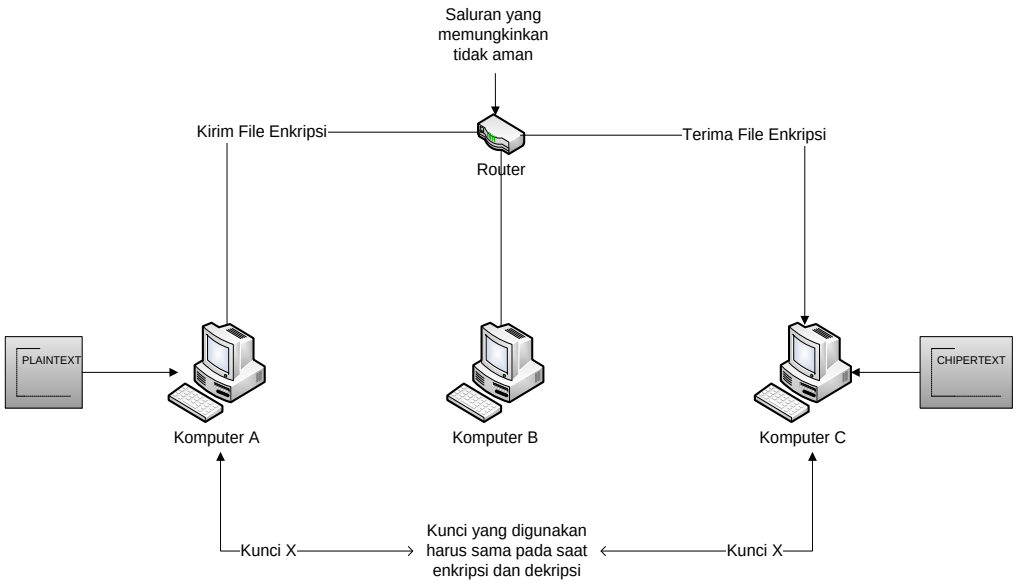
3.1.3.2 Analisis Pengiriman Data

Analisis pengiriman data dan kemanan jaringan merupakan tahap dimana analisis terhadap metode apa saja yang digunakan dalam sebuah proses pengiriman data teks pada jaringan *Local Area Network (LAN)*, dalam hal ini metode yang akan dipakai dalam proses pengiriman data teks pada jaringan *Local Area Network (LAN)* adalah transmisi secara seri, karena proses pengiriman data yang berupa teks akan diolah menjadi bentuk bilangan biner atau *bit*. Dan setiap *bit* hanya bernilai 1 dan 0. Maka dengan menggunakan metode ini proses pengiriman berupa *bit* dikirim secara berurutan dengan menggunakan satu jalur (penghantar), dan penerima kemudian merakit kembali menjadi karakter.

Pada permasalahan yang sedang dibahas yaitu mengenai analisis perbandingan algoritma *Rijndael* dan algoritma *Twofish* pada proses pengiriman data teks menggunakan jaringan *Local Area network (LAN).* Jadi proses *transfer* data dari komputer satu ke komputer lain menggunakan sistem jaringan *Local Area Network (LAN).* Dalam hal ini dengan adanya keamanan jaringan komputer

dengan menggunakan algoritma *Rijndael* dan *Twofish* dapat melindungi pada saat transaksi pengiriman data pada jaringan *Local Area Network (LAN).* Keamanan data pada suatu jaringan *Local Area Network (LAN)* adalah sesuatu yang sangat penting, karena ntuk menghindari data yang kita kirim disalahgunakan oleh orang-orang yang tidak berhak atau berkepentingan.

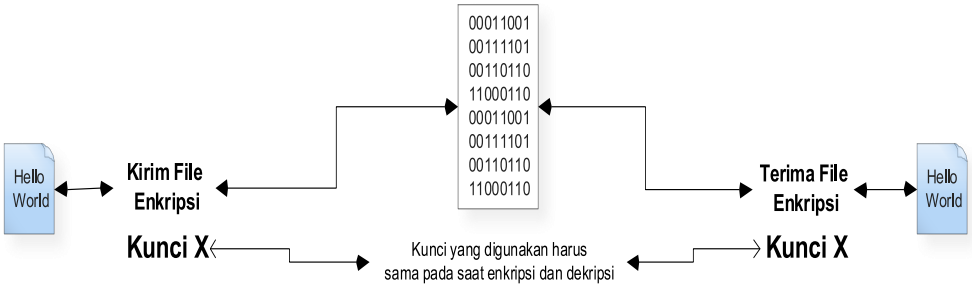
Ditinjau dari masalah yang dihadapi, maka perlu dibangun suatu aplikasi keamanan pengiriman data dengan menggunakan algoritma kriptografi *Rijndael* dan *Twofish* pada saat mengirimkan data melalui jaringan *Local Area Network (LAN)*. Berikut dibawah ini merupakan gambaran dari pengiriman data melalui jaringan *Local Area Network (LAN)* dan permasalahan yang memungkinkan terjadi pada media jaringan *Local Area Network (LAN).*



Gambar 3.45 Pengiriman Data Melalui Jaringan *Local Area Network*

Dari gambar 3.45 tersebut dapat dijelaskan bagaimana melakukan pengiriman sebuah data teks dari komputer A dan B melalui media jaringan *Local Area Network (LAN)*. Untuk melakukannya diperlukan sebuah data yang akan

dikirim dengan dienkripsi serta memasukkan kunci terlebih dahulu, kemudian dikirimkan menurut alamat tujuan yang akan dikirim, dan penerima dari data yang terenkripsi tersebut dapat mendekripsi data tersebut jika mengetahui kunci yang dipakai, maka dari itu penggunaan kunci sebelumnya harus disepakati dan diketahui oleh pengirim dan penerima.



Gambar 3.46 Pengiriman dan Penerimaan Data Berupa *Bit*

Dari gambar 3.46 tersebut dapat dijelaskan sebagai berikut :

Gambar 3.46 tersebut adalah menjelaskan bagaimana melakukan pengiriman dan penerimaan sebuah data teks dengan menggunakan metode transmisi secara seri, yaitu proses pengiriman data yang berupa teks akan diolah menjadi bentuk bilangan biner atau *bit*. Dan setiap *bit* hanya bernilai 1 dan 0. Maka dengan menggunakan metode ini proses pengiriman berupa *bit* dikirim secara berurutan dengan menggunakan satu jalur (penghantar), dan penerima kemudian merakit kembali menjadi karakter.

3.1.4 Analisis Kebutuhan Non Fungsional

Analisis kebutuhan non fungsional bertujuan untuk menghasilkan spesifikasi kebutuhan non fungsional. Spesifikasi kebutuhan non fungsional adalah spesifikasi yang rinci mengenai hal-hal yang akan dilakukan sistem ketika

di implementasikan. Analisis kebutuhan ini diperlukan untuk menentukan keluaran yang akan dihasilkan sistem, masukan yang diperlukan sistem, lingkup proses yang digunakan untuk mengolah masukan menjadi keluaran. Kebutuhan ini terdiri dari kebutuhan perangkat lunak (*software)* dan kebutuhan perangkat keras (*hardware).*

3.1.4.1 Analisis Perangkat Lunak

Kebutuhan *software* merupakan kebutuhan akan perangkat lunak yang digunakan untuk membangun program aplikasi kriptografi ini. Diantara kebutuhan tersebut adalah :

1. Sistem operasi berbasis *windows*, yang akan digunakan adalah sistem operasi *Windows XP SP 3.*
2. *Tools* / bahasa pemrograman, yang akan digunakan adalah *Microsoft Visual Basic* 6.0

3.1.4.2 Analisis Perangkat Keras

Perangkat keras yang dibutuhkan dalam pembuatan aplikasi kriptografi adalah sebagai berikut :

1. *Processor : Intel Pentium III* (800 Mhz)
2. *Motherboard : MSI*
3. *Memory : SDR* 128 *Mb PC* 133
4. *VGA card : NVIDIA Riva TNT* 32 *Mb*
5. *Harddisk : Seagate* 10 *GB*
6. Sebagai alat koneksi jaringan memakai kabel *UTP* (*Unshielded Twisted Pair*) dan konektor *RJ-45*.

3.1.4.3 Analisis Pengguna

Latar belakang pengguna yang akan memakai program aplikasi yang akan dibuat, terbagi menjadi dua pengguna yaitu pengirim pesan dan penerima pesan, serta memiliki kriteria pengguna sebagai berikut :

1. Memiliki kemampuan dasar dalam pengoperasian komputer.
2. Mengenal sistem kriptografi.
3. Mengenal jaringan *Local Area Network (LAN).*

3.1.4.3.1 Pengirim *File* atau Data

Orang yang mengirim *file*, akan melakukan langkah-langkah sebagai berikut :

1. Memilih suatu *file* yang akan dienkripsi dan dikirim.
2. Memilih algoritma untuk proses enkripsi.
3. Memasukkan *IP Address* tujuan pengiriman.
4. Mengkoneksikan jaringan.
5. Mengirim *file* enkripsi untuk 2 algoritma *Rijndael* atau *Twofish*.

3.1.4.3.2 Penerima *File* atau Data

Berikut langkah – langkah yang akan dilakukan oleh penerima pesan atau *file* adalah sebagai berikut :

1. Mengkoneksikan jaringan.
2. Mengambil *file* enkripsi (*Rijndael* atau *Twofish*) yang akan di dekripsi.
3. Melakukan dekripsi.

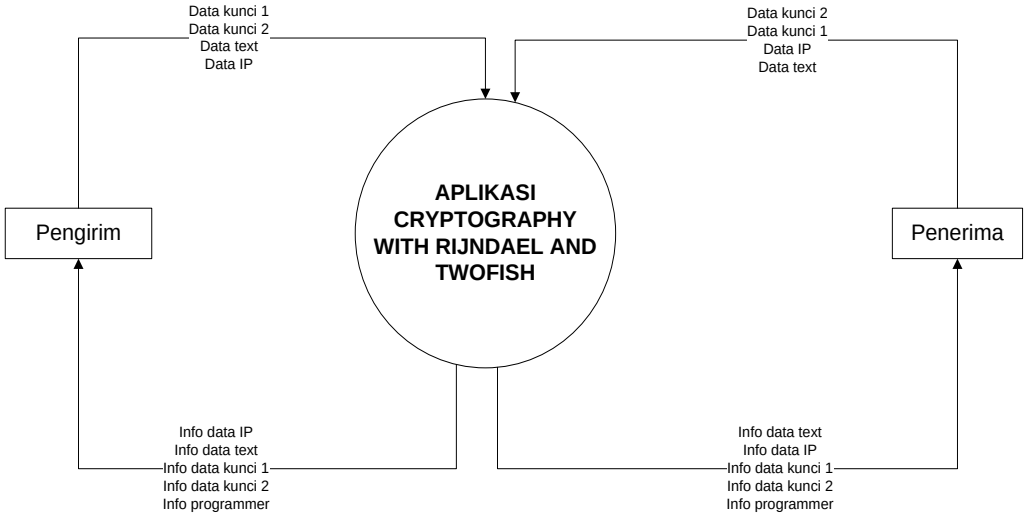
3.1.5 Analisis Kebutuhan Fungsional

Dalam langkah ini dilakukan penentuan entitas-entitas, data-data yang mengalir serta prosedur-prosedur yang bisa dilakukan oleh masing-masing entitas.

3.1.5.1 Diagram Konteks

Diagram konteks merupakan gambaran secara umum mengenai sebuah sistem yang dirancang secara *global*, yaitu suatu diagram yang mempresentasikan atau mengambarkan hubungan antara sistem dengan luar lingkungan luar sistem yang mempengaruhi operasi sistem.

Sistem ditunjukan dalam satu lingkungan yang mengambarkan keseluruhan proses dalam sistem dan hubungannya dengan entitas. Diagram konteks pada sistem kriptografi ini dapat dilihat pada gambar 3.47



Gambar 3.47 Diagram Konteks

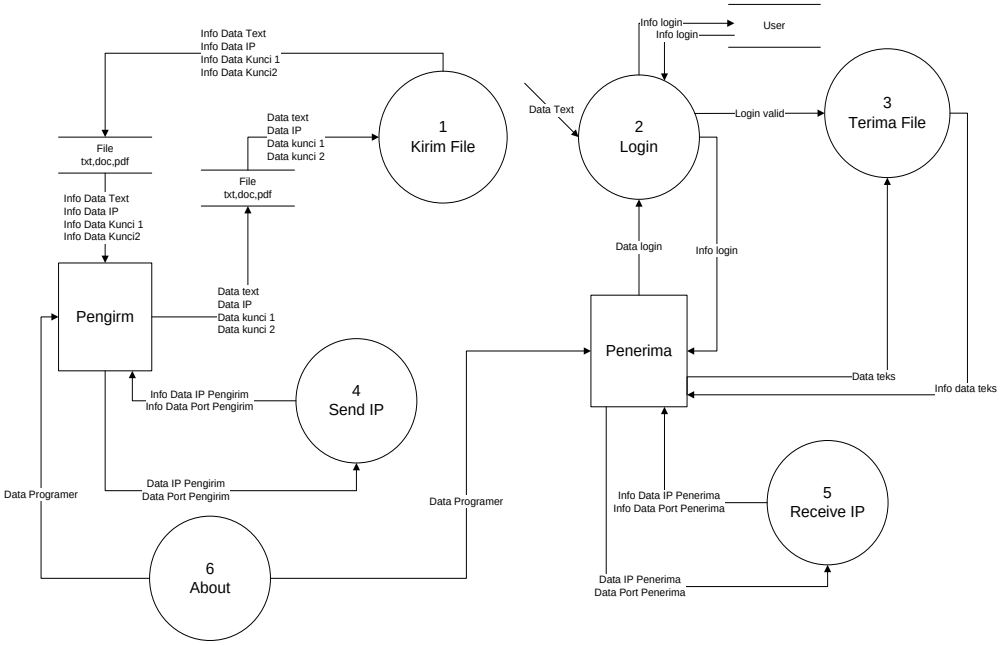
3.1.5.2 *Data Flow Diagram* (DFD)

Data Flow Diagram merupakan representasi grafik dari sebuah sistem yang menunjukan proses-proses dalam sebuah sistem dan aliran data yang masuk dan keluar dari proses tersebut secara logika tanpa harus

mempertimbangkan lingkungan fisik dimana data tersebut mengalir atau lingkungan fisik dimana data tersebut akan disimpan. DFD pada sistem ini dari DFD level 1 sampai DFD Level 2.

3.1.5.2.1 *Data Flow Diagram* (DFD) *Level* 1

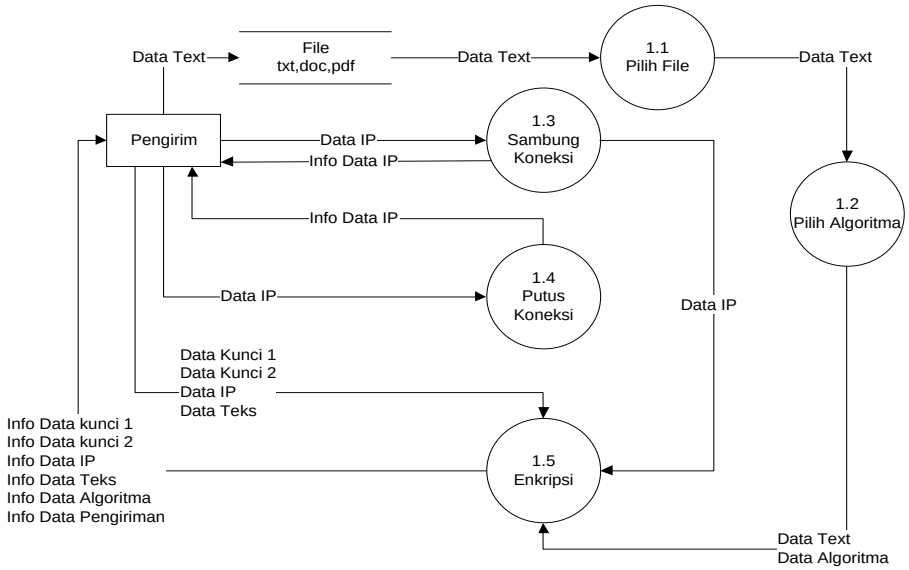
DFD Level 1 memiliki enam proses utama yaitu Kirim File, Login, Terima File, Send IP, Receive IP dan About dengan dua entitas yaitu Pengirim dan Penerima, seperti pada Gambar 3.48



Gambar 3.48 *Data Flow Diagram* Level 1

3.1.5.2.2 *Data Flow Diagram* (DFD) *Level* 2 Proses 1

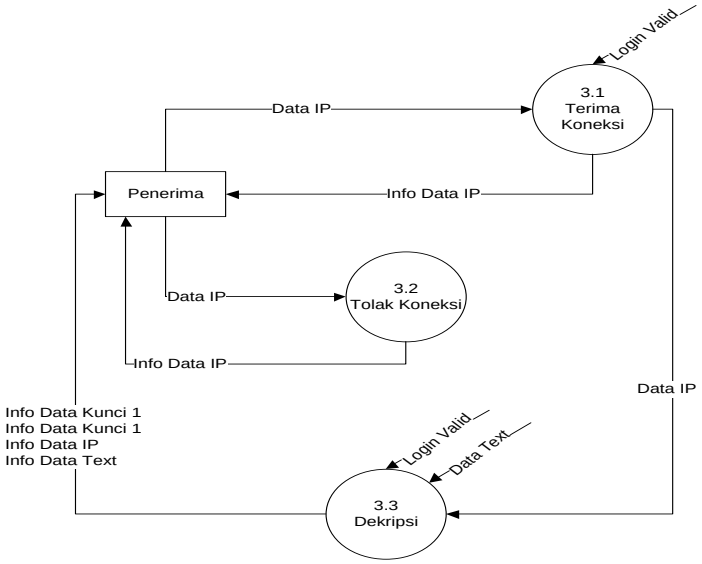
DFD *Level* 2 proses 1 memiliki empat proses utama yaitu Pilih*File*, Pilih Algoritma, Sambung Koneksi, Putus Koneksi dan Enkripsi. Semua proses ini diakses oleh Pengirim seperti terlihat pada Gambar 3.49



Gambar 3.49 *Data Flow Diagram Level* 2 proses 1

3.1.5.2.3 *Data Flow Diagram* (DFD) *Level* 2 proses 3

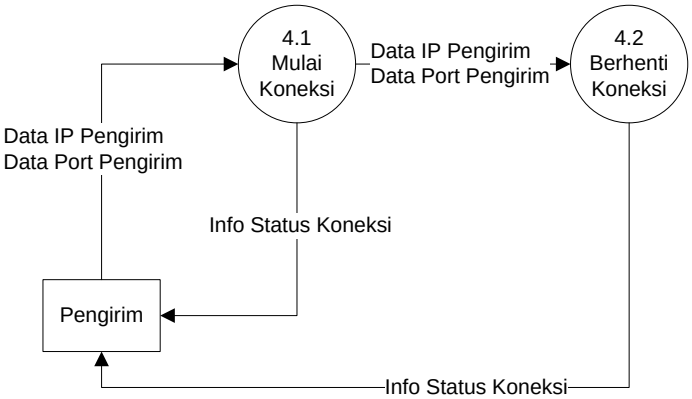
DFD *Level* 2 proses 3 memiliki tiga proses utama yaitu Terima Koneksi, Tolak Koneksi dan Dekripsi. Semua proses ini diakses oleh Penerima seperti terlihat pada Gambar 3.50.



Gambar 3.50 *Data Flow Diagram Level* 2 proses 3

3.1.5.2.4 *Data Flow Diagram* (DFD) *Level* 2 proses 4

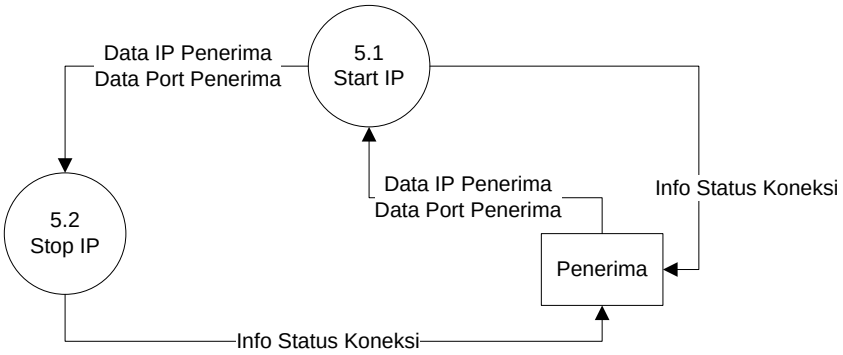
DFD *Level* 2 proses 4 memiliki dua proses utama yaitu Mulai Koneksi dan Berhenti Koneksi seperti terlihat pada Gambar 3.51



Gambar 3.51 *Data Flow Diagram Level* 2 proses 4

3.1.5.2.5 *Data Flow Diagram* (DFD) *Level* 2 proses 5

DFD *Level* 2 proses 5 memiliki dua proses utama yaitu Start IP dan Stop IP seperti terlihat pada Gambar 3.52



Gambar 3.52 Data Flow Diagram Level 2 proses 5

3.1.5.3 Spesifikasi Proses

Spesifikasi proses digunakan untuk menggambarkan semua proses *model* aliran data yang disajikan pada diagram aliran data. Spesifikasi proses yang terdapat pada *Data Flow Diagram* dapat dilihat pada tabel 3.4.

Tabel 3.4 Spesifikasi Proses

No	Proses	Keterangan
1.	No. Proses	1
	Nama Proses	Kirim File
	Sumber (*source*)	Pengirim
	Masukan (*Input)*	Data Text, Data IP Pengirim
	Keluaran (*output*)	Info Data Text, Info Data IP
	Tujuan (*destination*)	Penerima
	Logika Proses	Begin { Pengirim memasukkanData Text, Data IP Pengirim ke sistem} End
2.	No. Proses	2
	Nama Proses	Login
	Sumber (*source*)	Penerima
	Masukan (*Input)*	Data Login
	Keluaran (*output*)	Info Data Login
	Tujuan (*destination*)	Penerima
	Logika Proses	Begin { Penerima memasukkanData Login ke sistem } End
3.	No. Proses	3
	Nama Proses	Terima File
	Sumber (*source*)	Penerima
	Masukan (*Input)*	Data IP Penerima
	Keluaran (*output*)	Info Data IP Penerima
	Tujuan (*destination*)	Penerima
	Logika Proses	Begin { Penerima memasukkanData IP Penerima ke sistem } End
4.	No. Proses	4
	Nama Proses	Send IP
	Sumber (*source*)	Pengirim
	Masukan (*Input)*	Data IP Pengirim, Data Port Pengirim

	Keluaran (*output*)	InfoData IP Pengirim, Info Data Port Pengirim
	Tujuan (*destination*)	Penerima
	Logika Proses	Begin {Pengirim memasukan Data IP dan Data Port ke sistem} End
5.	No. Proses	5
	Nama Proses	Receive IP
	Sumber (*source*)	Penerima
	Masukan (*Input)*	Data IP Penerima, Data Port Penerima
	Keluaran (*output*)	Info Data IP Pengirim, Info Data Port Pengirim
	Tujuan (*destination*)	Pengirim
	Logika Proses	Begin {Penerima memasukan Data IP dan Data Port ke sistem} End
6.	No. Proses	1.1
	Nama Proses	Pilih File
	Sumber (*source*)	Pengirim
	Masukan (*Input)*	Data Text
	Keluaran (*output*)	Info Data Text
	Tujuan (*destination*)	Pengirim
	Logika Proses	Begin {Pengirim memasukan Data Text ke sistem} End
7.	No. Proses	1.2
	Nama Proses	Pilih Algoritma
	Sumber (*source*)	Pengirim
	Masukan (*Input)*	Data Teks, Data Algoritma
	Keluaran (*output*)	Info Data Teks, Input Data Algoritma
	Tujuan (*destination*)	Penerima
	Logika Proses	Begin {Pengirim memilih algoritma ke sistem} End
8.	No. Proses	1.3
	Nama Proses	Sambung Koneksi
	Sumber (*source*)	Pengirim
	Masukan (*Input)*	Data IP Pengirim
	Keluaran (*output*)	Info Data IP Pengirim
	Tujuan (*destination*)	Pengirim
	Logika Proses	Begin {Pengirim memasukan Data IP Pengirim ke sistem}

		End
9.	No. Proses	1.4
	Nama Proses	Putus Koneksi
	Sumber (*source*)	Pengirim
	Masukan (*Input)*	Data IP Pengirim
	Keluaran (*output*)	Info Data IP Pengirim
	Tujuan (*destination*)	Pengirim
	Logika Proses	Begin {Pengirim memasukan Data IP Pengirim ke sistem} End
10.	No. Proses	1.5
	Nama Proses	Enkripsi
	Sumber (*source*)	Pengirim
	Masukan (*Input)*	Data Teks, Data IP , Data Algoritma
	Keluaran (*output*)	Info Data Text, Info Data IP, Info Data Algoritma, Info Data Pengiriman
	Tujuan (*destination*)	Penerima
	Logika Proses	Begin {Pengirim memasukan Data Teks, Data IP, Data Algoritma ke sistem} End
11.	No. Proses	3.1
	Nama Proses	Terima Koneksi
	Sumber (*source*)	Penerima
	Masukan (*Input)*	Data Text, Data IP Pengirim
	Keluaran (*output*)	Info Data Text, Info Data IP Pengirim
	Tujuan (*destination*)	Pengirim
	Logika Proses	Begin { Pengirim memasukan Data Text } End
12.	No. Proses	3.2
	Nama Proses	Tolak Koneksi
	Sumber (*source*)	Penerima
	Masukan (*Input)*	Data IP
	Keluaran (*output*)	Info Data IP
	Tujuan (*destination*)	Penerima
	Logika Proses	Begin {Penerima memasukan Data IP ke sistem } End
13.	No. Proses	3.3
	Nama Proses	Dekripsi
	Sumber (*source*)	Penerima
	Masukan (*Input)*	Data Text
	Keluaran (*output*)	Info Data Text

	Tujuan (*destination*)	Penerima
	Logika Proses	Begin {Penerima memasukan Data Text ke sistem } End
13.	No. Proses	4.1
	Nama Proses	Mulai Koneksi
	Sumber (*source*)	Pengirim
	Masukan (*Input)*	Data IP Pengirim, Data Port Pengirim
	Keluaran (*output*)	Info Data IP Pengirim, Info Data Port Pengirim
	Tujuan (*destination*)	Penerima
	Logika Proses	Begin {Pengirim memasukan Data IP dan Data Port Pengirim ke sistem } End
14.	No. Proses	4.2
	Nama Proses	Berhenti Koneksi
	Sumber (*source*)	Pengirim
	Masukan (*Input)*	Data IP Pengirim, Data Port Pengirim
	Keluaran (*output*)	Info Data IP Pengirim, Info Data Port Pengirim
	Tujuan (*destination*)	Penerima
	Logika Proses	Begin {Pengirim memasukan Data IP dan Data Port Pengirim ke sistem } End
15.	No. Proses	5.1
	Nama Proses	Start IP
	Sumber (*source*)	Penerima
	Masukan (*Input)*	Data IP Penerima, Data Port Penerima
	Keluaran (*output*)	Info Data IP Penerima, Info Data Port Penerima
	Tujuan (*destination*)	Pengirim
	Logika Proses	Begin {Penerima memasukan Data IP dan Data Port Pengirim ke sistem } End
16.	No. Proses	5.2
	Nama Proses	Stop IP
	Sumber (*source*)	Penerima
	Masukan (*Input)*	Data IP Penerima, Data Port Penerima
	Keluaran (*output*)	Info Data IP Penerima, Info Data Port Penerima
	Tujuan (*destination*)	Pengirim

	Logika Proses	Begin {Penerima memasukan Data IP dan Data Port Pengirim ke sistem } End

3.1.5.4 Kamus Data

Kamus data adalah kumpulan dari elemen – elemen atau simbol yangdigunakan untuk membantu dalam penggambaran atau pengidentifikasian setiap*field* atau *file* didalam sistem.Kamus data merupakan deskripsi formal mengenai seluruh elemen yang mencakup DFD.

Kamus data dapat mendefinisikan dengan lengkap data yang mengalir diantara proses, penyimpanan data, dan entitas. Data yang mengalir tersebut dapat berupa masukan untuk sistem atau hasil di proses sistem. Kamus data dibuat berdasarkan arus data yang mengalir pada konteks diagram dan DFD. Kamus data dapat dilihat pada Tabel 3.5.

Tabel 3.5 Kamus Data

Nama	**Data Login**
Where used/ how Used	Proses 2 Login (*Input*)
Deskripsi	Berisi data login yang akan digunakan untuk masuk ke dalam sistem
Struktur Data	*txtUser & txtPass*
txtUser	[0..9] {30}
txtPass	[0..9] {12}
Nama	**Data Text**
Where used/ how Used	Proses 1 Kirim File (*Input*) Proses 3 Terima File (*Input*) Proses 1.1 Pilih File (*Input*) Proses 1.5 Enkripsi (*Input*) Proses 3.3 Dekripsi (*Input*)
Deskripsi	Berisi data teks yang akan di enkripsi dan di dekripsi
Struktur Data	*Txt*

txt	[a..z | A..Z | 0..9] {30}
Nama	**Data IP Pengirim**
Where used/ how Used	Proses 1 Kirim File (*Input*) Proses 4 Send IP (*Input*) Proses 1.3 Sambung Koneksi (*Input*) Proses 3.1 Start Koneksi Penerima (*Input*) Proses 4.1 Start (*Input*) Proses 4.2 Stop (*Input*)
Deskripsi	Berisi data IP pengirimyang akan digunakan untuk koneksi jaringan
Struktur Data	txtIP
txtIP	[0..9] {12}
Nama	**Data IP Penerima**
Where used/ how Used	Proses 3 Dekripsi (*Input*) Proses 5 Receiver (*Input*) Proses 1.2 Start Koneksi (*Input*) Proses 3.1 Terima Koneksi (*Input*) Proses 5.1 Start IP(*Input*) Proses 5.2 Stop IP(*Input*)
Deskripsi	Berisi data IP penerima yang akan digunakan untuk koneksi jaringan
Struktur Data	txtIP
txtIP	[0..9] {12}

3.2 Perancangan Sistem

Perancangan akan dimulai setelah tahap analisis terhadap sistem selesai dilakukan. Perancangan dapat didenifisikan sebagai penggambaran, perencanaan danpembuatan sketsa atau pengaturan dari beberapa elemen yang terpisah ke dalam satu kesatuan yang utuh dan berfungsi. Tahapan ini menyangkut mengkonfigurasi dari komponen-komponen perangkat lunak dan perangkat keras dari suatu sistem sehingga setelah instalasi dari sistem akan benar-benar memuaskan dari rancang bangun yang telah ditetapkan pada akhir tahap analisis sistem.

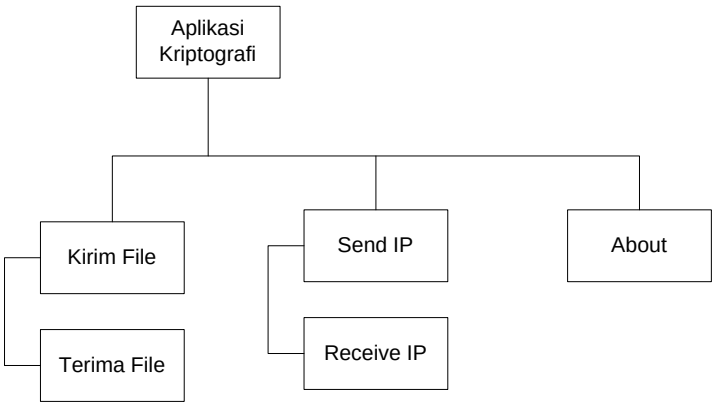
Alat bantu yang digunakan untuk menggambarkan perancangan sistem secara umum yang akan dibangun, yaitu diagram konteks dan *data flow diagram*. Untuk menjelaskan bagaimana suatu masukan diproses pada sistem maka digunakan spesifikasi proses dan kamus data untuk mengetahui aliran data yang mengalir pada sistem.

3.2.1 Perancangan Arsitektur

Perancangan arsitektur merupakan perancangan yang dibuat sebelum program aplikasi dibangun. Dengan perancangan arsitektur akan mempermudah proses pembangunan Sistem Kriptografi.

3.2.1.1 Perancangan Struktur Menu

Perancangan struktur menu berisikan menu dan submenu yang berfungsi memudahkan *user* didalam menggunakan sistem. Menu – menu tersebut dibagi berdasarkan hak akses masing-masing seperti pada Gambar 3.53.



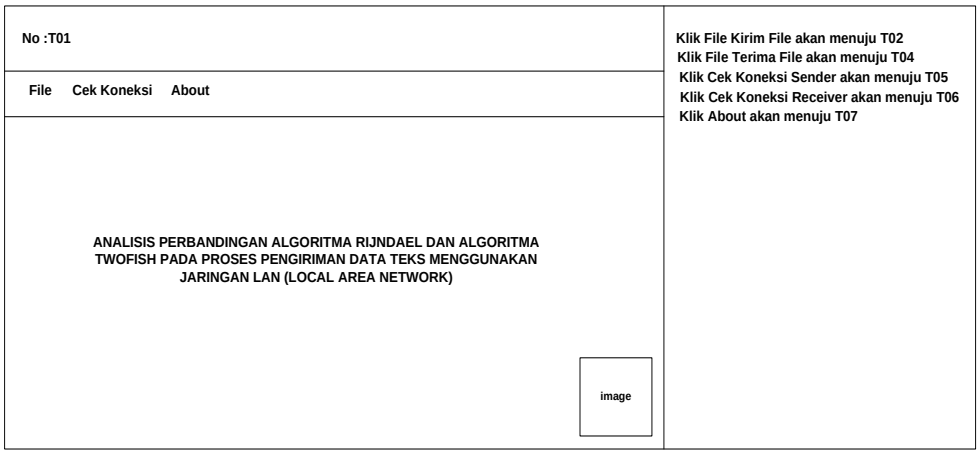
Gambar 3.53 Struktur Menu

3.2.1.2 Perancangan Antarmuka

Spesifikasi antarmuka merupakan suatu bentuk tampilan dari program yang akan dibuat untuk kebutuhan *interface* dengan *user*. Spesifikasi antarmuka terdiri dari perancangan tampilan program, tampilan pesan, dan jaringan semantik.

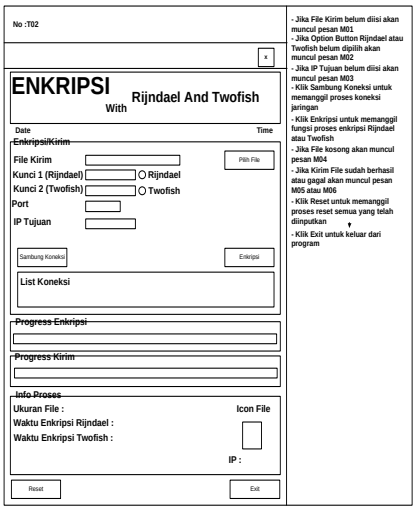
3.2.1.2.1 Perancangan Tampilan Program

1. Tampilan *Menu* Utama Aplikasi Kriptografi



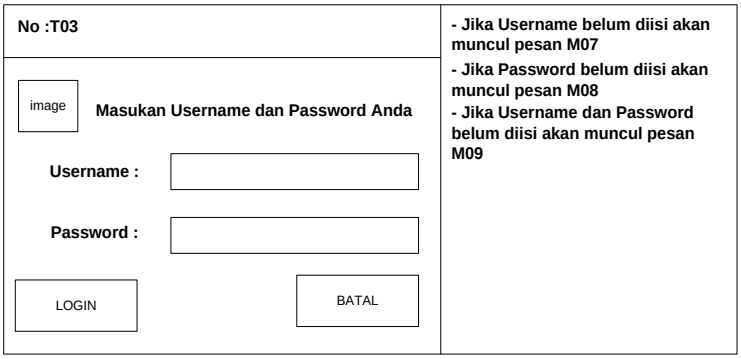
Gambar 3.54 Tampilan *Menu* Utama Aplikasi Kriptografi

2. Tampilan *Menu* Kirim *File*



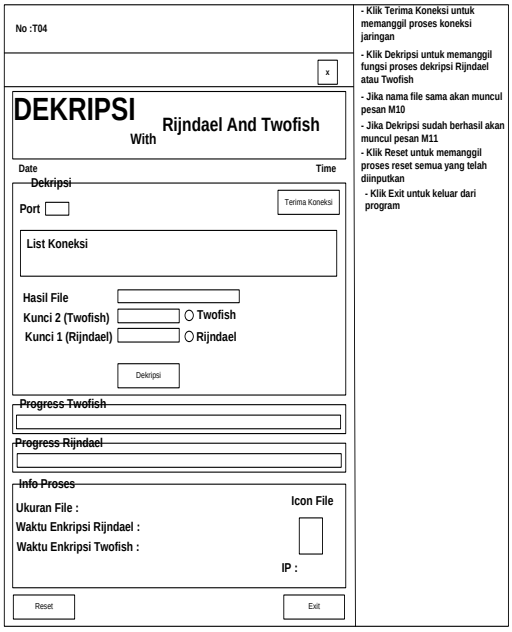
Gambar 3.55 Tampilan *Menu* Kirim *File*

3. Tampilan *Menu Login*



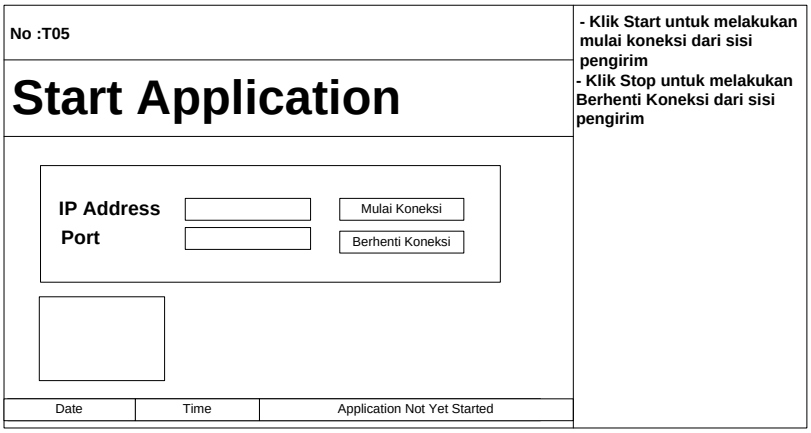
Gambar 3.56 Tampilan *Menu Login*

4. Tampilan *Menu* Terima *File*



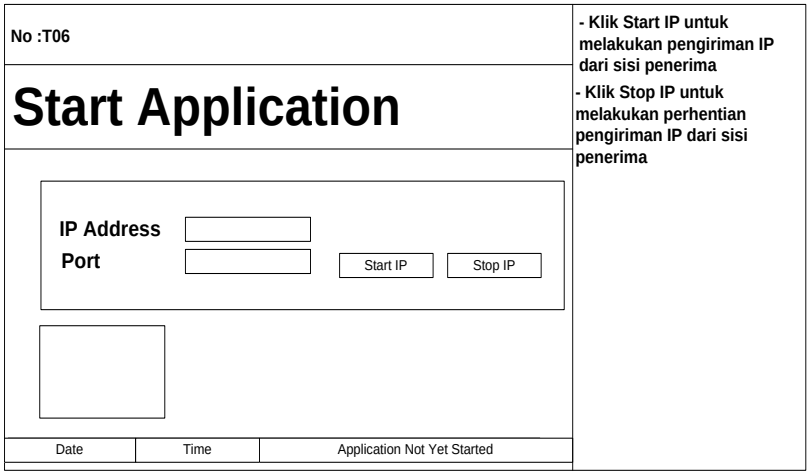
Gambar 3.57 Tampilan *Menu* Terima *File*

5. Tampilan *Menu Send IP*



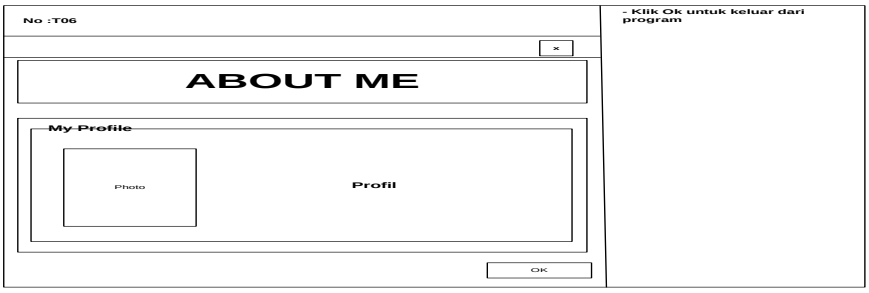
Gambar 3.58 Tampilan *Menu Send IP*

6. Tampilan *Menu Receive IP*



Gambar 3.59 Tampilan *Menu Receive IP*

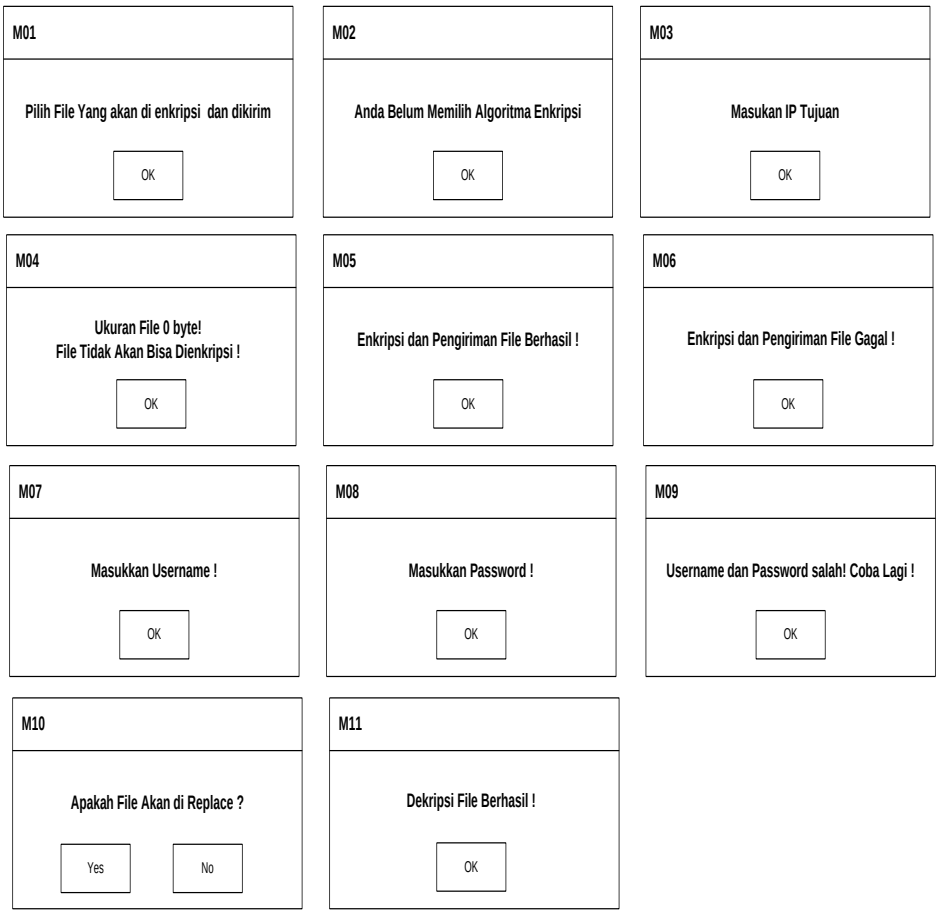
7. Tampilan *Menu About*



Gambar 3.60 Tampilan *Menu About*

3.2.1.3 Perancangan Pesan

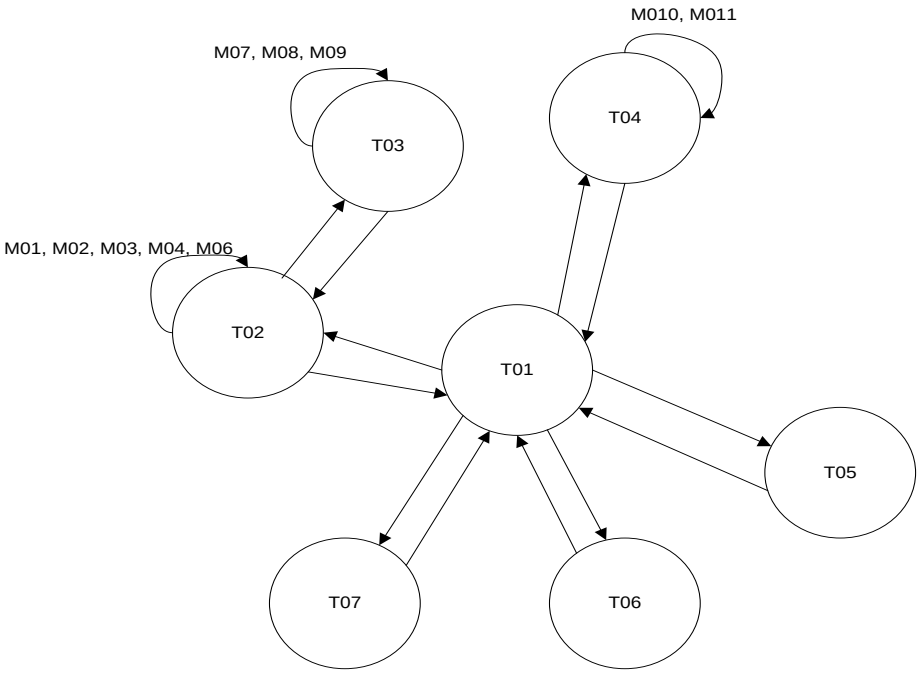
Berikut ini adalah perancangan pesan yang ada pada aplikasi kriptografi.



Gambar 3.61 Perancangan Pesan

3.2.1.4 Jaringan Semantik

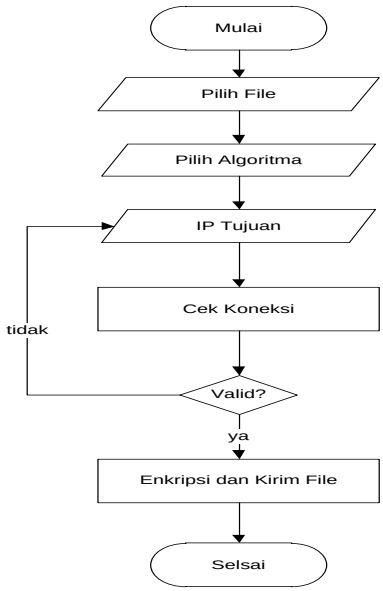
Jaringan semantik merupakan alat efektif untuk mempresentasikan pemetaan data, yang bertujuan mencegah terjadinya duplikasi data. Untuk penjelasan dari jaringan semantik yang ada dibawah ini akan diuraikan sebagai berikut.



Gambar 3.62 Jaringan Semantik

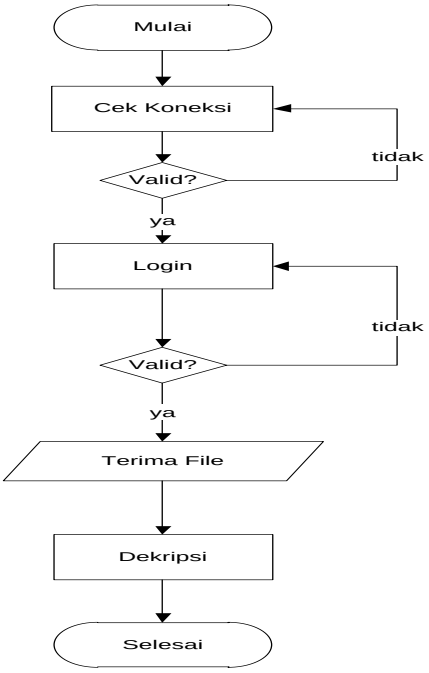
3.2.1.5 Perancangan Prosedural

Perancangan program akan menjelaskan tentang bagaimana program aplikasi berjalan. Perancangan prosedural digambarkan dalam bentuk *flowchart.*



Gambar 3.63 Perancangan Prosedural Enkripsi dan Pengiriman *File*

Untuk melakukan proses enkripsi dan pengiriman *file* langkah petama yang dilakukan adalah melakukan pemilihan *file* yang akan di enkripsi lalu pilih algoritmanya untuk algoritma *Rijndael* atau algoritma *Twofish*. Lalu masukkan *IP* tujuan untuk koneksi ke komputer penerima. Jika *IP* tujuan yang diisi *valid* maka lanjut ke proses enkripsi dan kirim *file*, jika salah maka kembali ke proses pengisian *IP* tujuan kembali. Langkah terakhir setelah terkoneksi dengan komputer penerima, lakukan proses kirim *file* dan selesai.



Gambar 3.64 Perancangan Prosedural Dekripsi

Untuk melakukan proses dekripsi langkah pertama yang dilakukan adalah melakukan cek koneksi terlebih dahulu agar dapat mengambil *file* yang telah dikirim. Jika tidak file yang telah dikirim tidak akan masuk. Pada saat proses penerimaan file, si penerima harus melakukan proses login. Jika berhasil dalam proses login maka akan melanjutkan ke proses penerimaan file. Jika gagal akan tetap dalam proses login. Setelah berhasil dalam proses login maka akan dilanjutkan kembali untuk ke proses selanjutnya yaitu penerimaan file. Langkah terakhir jika sudah diambil filenya maka proses dekripsi sudah bisa dilakukan dan selesai.