BAB 3

Analisis dan Perancangan Sistem

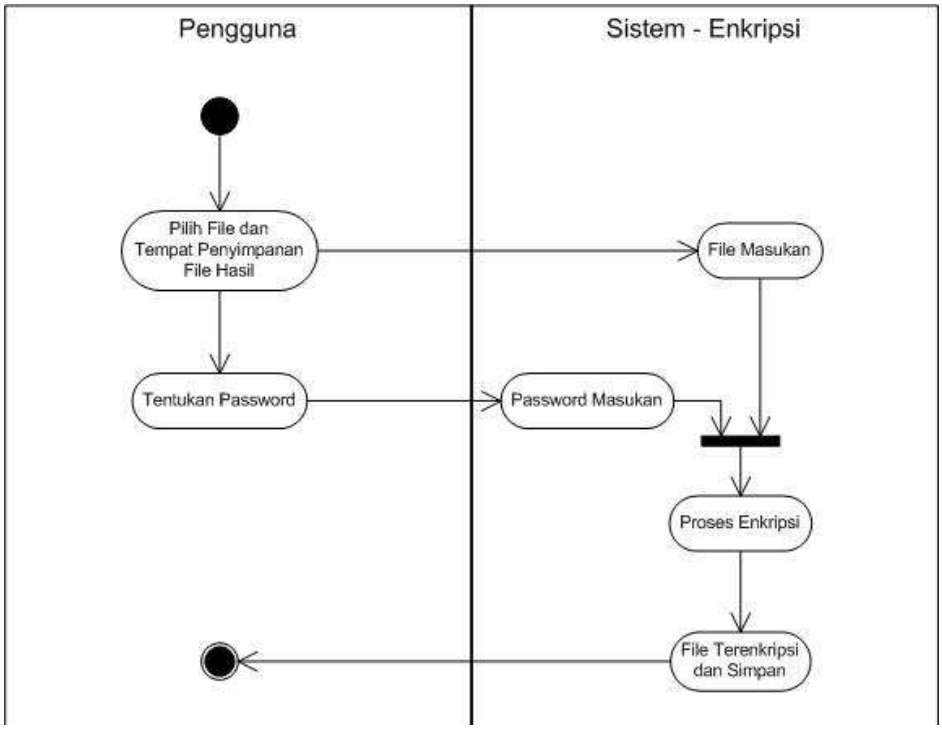
3.1 Deskripsi Umum Sistem

Deskripsi umum sistem merupakan gambaran umum tentang sistem yang dibangun. Sistem yang dibangun ini mampu mengamankan *file* atau memampatkan *file.* Terdapat empat proses utama dalam sistem ini, yaitu:

1. Proses enkripsi *file*

   Proses enkripsi *file* adalah adalah proses mengacak *file* sehingga tidak dapat dibaca oleh pihak lain dengan meyertakan kunci agar *file* dapat dikunci atau dibuka. Berikut ini adalah prosedur enkripsi *file* seperti pada Gambar 3.1:

   a. Pengguna memilih *file* yang tidak terenkripsi dan tempat penyimpanan *file* hasil enkripsi.
   b. Pengguna menentukan *password* untuk enkripsi.
   c. Sistem melakukan enkripsi.
   d. Sistem menghasilkan *file* terenkripsi dan menyimpannya.

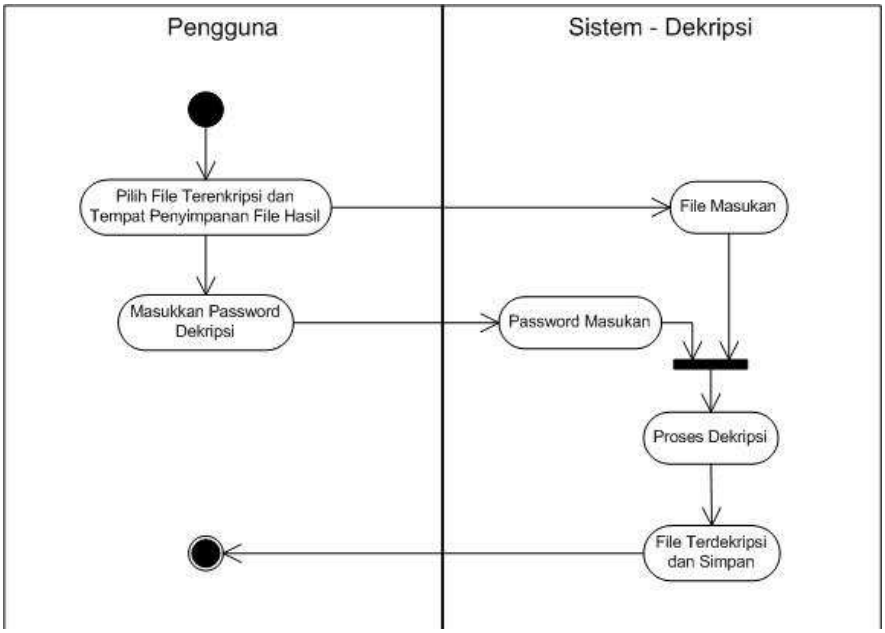


Gambar 3.1 *Activity* Diagram Proses Enkripsi

2. Proses dekripsi *file*

   Proses dekripsi *file* adalah proses membuka keamanan *file* melalui kunci yang disertakan saat proses enkripsi. Berikut ini adalah prosedur dekripsi *file* seperti pada Gambar 3.2:

   a. Pengguna memilih *file* terenkripsi dan tempat penyimpanan *file* hasil dekripsi.
   b. Pengguna memasukkan *password* untuk dekripsi.
   c. Sistem melakukan dekripsi.
   d. Sistem menghasilkan *file* terdekripsi dan menyimpannya.

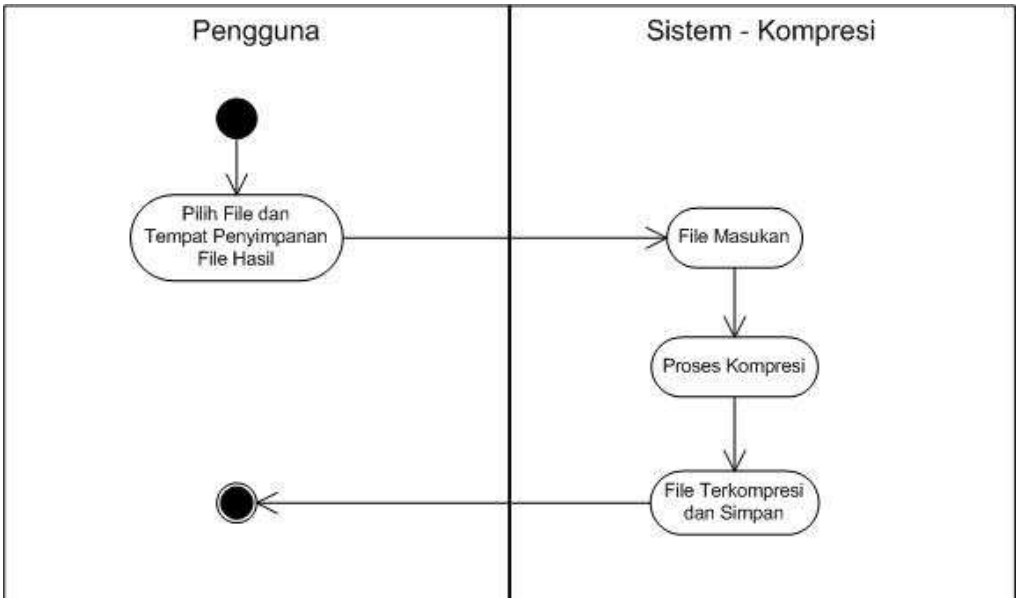


Gambar 3.2 *Activity* Diagram Proses Dekripsi

3. Proses kompresi *file*

   Proses kompresi *file* adalah proses pemampatan ukuran *file* dengan menghilangkan informasi ganda agar ukuran *file* menjadi lebih kecil dan menghemat kapasitas media penyimpanan. Berikut ini adalah prosedur kompresi *file* seperti pada Gambar 3.3:

   a. Pengguna memilih *file* yang tidak terkompresi dan tempat penyimpanan *file* hasil kompresi.
   b. Sistem melakukan kompresi *file*.
   c. Sistem menghasilkan *file* terkompresi dan menyimpannya.

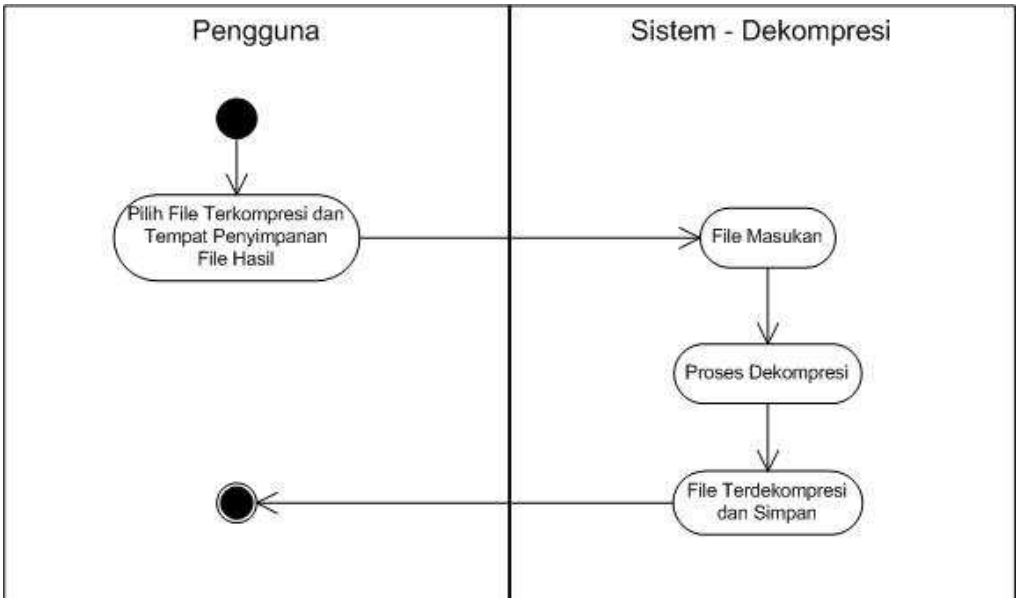


Gambar 3.3 *Activity* Diagram Proses Kompresi

4. Proses dekompresi *file.*

   Proses dekompresi *file* adalah proses mengembalikan *file* yang terkompresi dengan mengembalikan informasi-informasi yang dihilangkan pada proses kompresi menjadi seperti semula. Berikut ini adalah prosedur dekompresi *file* seperti pada Gambar 3.4:

   a. Pengguna memilih *file* terkompresi dan tempat penyimpanan *file* hasil dekompresi.
   b. Sistem melakukan dekompresi.
   c. Sistem menghasilkan *file* terdekompresi dan menyimpannya.



Gambar 3.4 *Activity* Diagram Proses Dekompresi

Pada proses-proses tersebut, sistem membutuhkan *file* sumber yang kemudian akan diproses dan menghasilkan sebuah *file* baru yang terenkripsi, terdekripsi, terkompresi, atau terdekompresi.

3.2 Analisis Sistem

Analisis sistem dapat didefinisikan sebagai penguraian dari suatu sistem yang utuh ke dalam bagian-bagian komponennnya dengan maksud untuk mengidentifikasi dan mengevaluasi permasalahan-permasalahan, hambatan-hambatan yang terjadi dan kebutuhan-kebutuhan yang diharapkan sehingga dapat diusulkan perbaikan-perbaikannya.

3.2.1 Analisis Data

Dalam sistem keamanan data ini sekumpulan *file-file* yang mempunyai berbagai jenis ekstensi yang digunakan sebagai objek penelitian. Umumnya pengenal tipe *file* tertera pada nama *file* tersebut, yaitu tiga huruf paling kanan setelah titik. Fungsinya adalah untuk mengetahui atau membedakan jenis *file*.

Untuk memperoleh hasil yang maksimal dalam enkripsi dan kompresi *file* serta mempermudah dalam pengujian kebenaran dan keakuratan aplikasi, maka data yang digunakan diambil dari beberapa *file* yang ada dalam media penyimpanan berdasarkan format dan besar kapasitas *file*. Sehingga dalam aplikasi enkripsi dan kompresi *file* ini terdapat beberapa *file* yang terdiri dari:

1. *File* teks, yang berekstensi *.doc, *.docx, *.xls, *.xlsx, *.ppt, *.pptx, *.txt, dan *.pdf
2. *File* suara, yang berekstensi *.mp3, *.wav
3. *File* citra, yang berekstensi *.jpg, *.bmp
4. *File* video, yang berekstensi *.flv, *.mkv, *.mp4

3.2.2 Analisis Algoritma *Rijndael*

Secara garis besar algoritma *Rijndael* yang beroperasi dengan panjang blok 128-bit dan panjang kunci 128-bit adalah sebagai berikut:

1. *AddRoundKey*: melakukan *XOR* antara *state* awal (plainteks) dengan *cipher key*. Tahap ini disebut juga *initial round*.
2. Putaran sebanyak *Nr* – 1 kali. Proses yang dilakukan pada setiap putaran adalah:
   a. *SubByte*: substitusi byte dengan menggunakan tabel substitusi (*S-box*).
   b. *ShiftRow:* pergeseran baris-baris *array state* secara *wrapping*.
   c. *MixColumn*: mengacak data di masing-masing kolom *array state*.
   d. *AddRoundKey:* melakukan *XOR* antara *state* sekarang dengan *round key*.
3. *Final round*: proses untuk putaran terakhir:
   a. *SubByte*.

b. *ShiftRow.*

c. *AddRoundKey*.

Algoritma *Rijndael* mempunyai 3 parameter sebagai berikut:

1. Plainteks: *array* yang berukuran 16 *byte*, yang berisi data masukan.
2. Cipherteks: *array* yang berukuran 16 *byte*, yang berisi hasil enkripsi.
3. *key* : *array* yang berukuran 16 *byte*, yang berisi kunci ciphering (disebut juga *cipher key*).

Dengan 16 *byte*, maka baik blok data dan kunci yang berukuran 128-*bit* dapat disimpan di dalam ketiga *array* tersebut (128 = 16 x 8). Selama kalkulasi plainteks menjadi cipherteks, status sekarang dari data disimpan di dalam *array state* yang berukuran `NROWS x NCOLS`. Elemen *array state* diacu sebagai

S[*r*,*c*], dengan 0 = *r* < 4 dan 0 = *c* < *Nc*

(*Nc* adalah panjang blok dibagi 32)

Pada awal enkripsi, 16-*byte* data masukan, in_0, in_1, ..., in_{15} disalin ke dalam *array state* (direalisasikan oleh fungsi `CopyPlaintextToState(state,plain text)`).

Operasi enkripsi/dekripsi dilakukan terhadap *array S*, dan keluarannya ditampung di dalam *array out.* Skema penyalinan array masukan *in* ke *array S*:

S[*r*, c] 7 *in*[*r* + 4*c*]

untuk 0 # *r* < 4 dan 0 # *c* < *Nb*

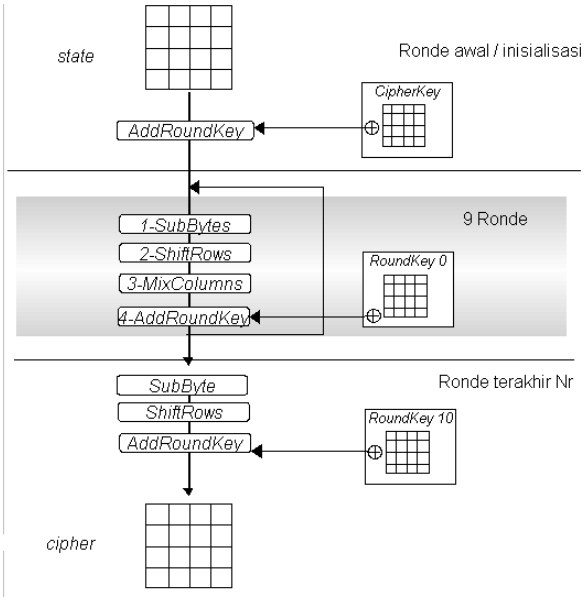
Skema penyalinan *array S* ke *array* keluaran *out*:

out[*r*+4*c*] 7 *S*[*r*,*c*]

untuk 0 # *r* < 4 dan 0 # *c* < *Nb*

3.2.2.1 Proses Enkripsi *Rijndael*

Proses enkripsi pada algoritma *Rijndael* terdiri dari 4 jenis transformasi *bytes* seperti pada Gambar 3.5, yaitu *AddRoundKey, SubBytes*, *ShiftRows*, dan *MixColumns*. Masukan yang telah disalinkan ke dalam *state* pada awal proses enkripsi akan mengalami transformasi *byte AddRoundKey*. Setelah itu, *state* akan mengalami transformasi *AddRoundKey*, *SubBytes*, *ShiftRows*, dan *MixColumns*, secara berulang-ulang sebanyak *Nr*. Proses ini dalam algoritma *Rijndael* disebut sebagai *round function*. *Round* yang terakhir agak berbeda dengan *round-round* sebelumnya dimana pada *round* terakhir, *state* tidak mengalami transformasi *MixColumns*. [8]

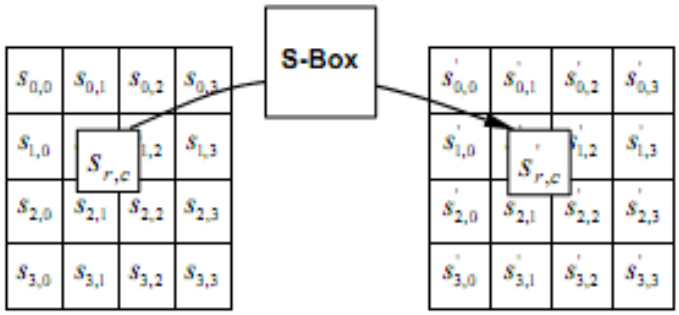


Gambar 3.5 Proses Enkripsi *Rijndael* [8]

1. Transformasi *SubBytes*

Transformasi *SubBytes* memetakan setiap *byte* dari *array state* dengan menggunakan tabel substitusi *S-Box* seperti pada Tabel 3.1. Cara pensubstitusiannya seperti pada Gambar 3.6 sebagai berikut: untuk setiap *byte*

pada *array state*, misalkan *S*[*r*,*c*] = *xy*, yang dalam hal ini *xy* adalah *digit* heksadesimal dari nilai *S*[*r*,*c*], maka nilai substitusinya, yang dinyatakan dengan *S'*[*r*,*c*], adalah elemen di dalam *S-Box* yang merupakan perpotongan baris *x* dengan kolom *y*.



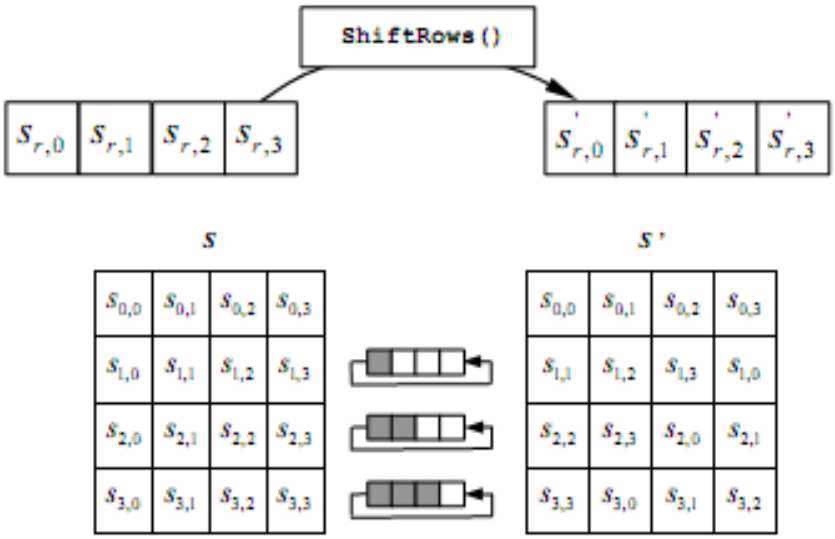
Gambar 3.6 Transformasi *SubBytes*

Tabel 3.1 S-*Box*

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	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	03	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

2. Transformasi *ShiftRow*

Transformasi *ShiftRow* melakukan pergeseran secara *wrapping* pada 3 baris terakhir dari *array state*. Jumlah pergeseran bergantung pada nilai baris (*r*). Baris *r* = 1 digeser sejauh 1 *byte*, baris *r* = 2 digeser sejauh 2 *byte*, dan baris *r* = 3 digeser sejauh 3 *byte*. Baris *r* = 0 tidak digeser seperti pada Gambar 3.7.



Gambar 3.7 Transformasi *ShiftRow*

3. Transformasi *MixColumns*

Transformasi *MixColumns* mengalikan setiap kolom dari *array state* dengan polinom $a(x)$ mod $(x^4 + 1)$. *MixColumns* memberikan difusi pada *cipher* seperti pada Gambar 3.8. Setiap kolom diperlakukan sebagai polinom 4-suku pada $GF(2^8)$. Polinom $a(x)$ yang ditetapkan adalah:

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad (3.1)$$

Transformasi ini dinyatakan sebagai perkalian matriks:

$$s'(x) = a(x) \otimes s(x)$$

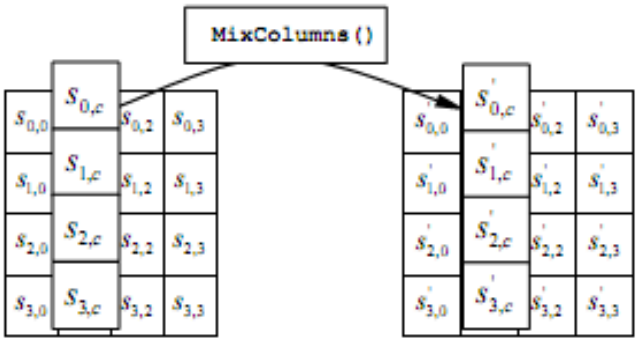
$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \text{ untuk } 0 \leq c < Nb \quad (3.2)$$

$$S'_{0,c} = (\{02\} \cdot S_{0,c}) \oplus (\{03\} \cdot S_{1,c}) \oplus S_{2,c} \oplus S_{3,c}$$

$$S'_{1,c} = S_{0,c} \oplus (\{02\} \cdot S_{1,c}) \oplus (\{03\} \cdot S_{2,c}) \oplus S_{3,c}$$

$$S'_{2,c} = S_{0,c} \oplus S_{1,c} \oplus (\{02\} \cdot S_{2,c}) \oplus (\{03\} \cdot S_{3,c})$$

$$S'_{3,c} = (\{03\} \cdot S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (\{02\} \cdot S_{3,c})$$



Gambar 3.8 Transformasi *MixColumns*

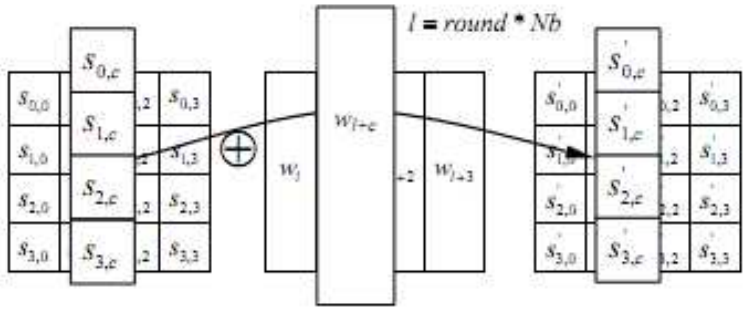
4. Transformasi *AddRoundKey*

Dalam transformasi *AddRoundKey,* kunci putaran ditambahkan ke dalam *state* melalui operasi XOR sederhana seperti pada Gambar 3.9. Setiap kunci putaran mengandung nilai *Nb* dari penjadwalan kunci. Setiap *Nb* masing-masing ditambahkan ke dalam kolom *state*, sehingga

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

untuk $0 \leq c < Nb$ (3.3)

dimana $[W_i]$ adalah penjadwalan kunci dan *round* adalah nilai di antara $0 \leq round < Nr$. Di dalam *cipher*, penambahan kunci awal putaran terjadi ketika putaran = 0 sebelum memulai perputaran. Penerapan transformasi *AddRoundKey* untuk putaran *Nr* dari *cipher* terjadi ketika $1 \leq round \leq Nr$.



Gambar 3.9 Transformasi *AddRoundKey*

5. Ekspansi Kunci

Ekspansi kunci dibutuhkan utuk memenuhi kebutuhan *subkey* yang dapat mencapai ribuan *bit* untuk melakukan enkripsi, sementara kunci enkripsi yang disediakan hanya 128 hingga 256 *bit*. Total *subkey* yang diperlukan *Rijndael* adalah *Nb(Nr* + 1*) word*. Jadi bila menggunakan *Rijndael*-128 yang berjumlah 128 *bit*, diekspan hingga menjadi 1408 *bit*, melalui proses yang disebut dengan *key schedule*.

Subkey sebanyak ini diperlukan karena setiap ronde membutuhkan *Nb word* ditambah satu *word subkey* untuk diawal. *Key schedule* menghasilkan *array* linear *word* [w_i] sebesar 4 *byte*, dimana *i* memiliki nilai di antara $0 < i < Nb\ (Nr + 1)$.

Operasi enkripsi *Rijndael* dapat dinyatakan dengan kode semu (*pseudocode*) berikut ini: [13]

Pseudocode Cipher *Rijndael*:

```
Cipher(byte in[], byte out[], word W[])   /*Nama fungsi*/
Begin
      In = 4 * Nb
      Out = 4 * Nb
      W = Nb * (Nr +1)
      Byte state[4,Nb]
      State = In /* Memasukkan Input ke state*/
```

```
      AddRoundKey(state,W)
      For round=1 step 1 to Nr-1 /*proses yang berlaku untuk
semua ronde kecuali ronde terakir*/
            SubBytes(state)
            shiftRows(state)
            MixColumns(state)
            AddRoundKey(state, w + round * Nb)
      End for
      SubBytes(state) /*proses yang berlaku khusus untuk ronde
terakhir*/
      ShiftRows(state)
      AddRoundKey(state, w + round * Nb) /* Mengirimkan keluaran
ke out */
      Out = state
End
```

Pseudocode di atas, dapat diketahui enkripsi dilakukan dengan fungsi *cipher* yang memiliki parameter masukan *in* = 16 *byte* dengan *Nb* adalah panjang blok, keluaran *out* = 16 *byte* dan array 1 dimensi w = 44 *byte* untuk *Rijndael*-128 dengan *Nr* adalah jumlah ronde.

Berikut ini adalah nilai-nilai dalam array *state* sebagai *cipher* saat berlangsung proses enkripsi untuk panjang blok dan panjang kunci *cipher* masing-masing 16 *byte* (yaitu, Nb = 4 dan Nk = 4).

```
PLAINTEXT:  00112233445566778899aabbccddeeff
KEY:        000102030405060708090a0b0c0d0e0f

CIPHER (ENCRYPT):
round[ 0].input   00112233445566778899aabbccddeeff
round[ 0].k_sch   000102030405060708090a0b0c0d0e0f
round[ 1].start   00102030405060708090a0b0c0d0e0f0
round[ 1].s_box   63cab7040953d051cd60e0e7ba70e18c
round[ 1].s_row   6353e08c0960e104cd70b751bacad0e7
round[ 1].m_col   5f72641557f5bc92f7be3b291db9f91a
round[ 1].k_sch   d6aa74fdd2af72fadaa678f1d6ab76fe
round[ 2].start   89d810e8855ace682d1843d8cb128fe4
round[ 2].s_box   a761ca9b97be8b45d8ad1a611fc97369
round[ 2].s_row   a7be1a6997ad739bd8c9ca451f618b61
round[ 2].m_col   ff87968431d86a51645151fa773ad009
round[ 2].k_sch   b692cf0b643dbdf1be9bc5006830b3fe
round[ 3].start   4915598f55e5d7a0daca94fa1f0a63f7
round[ 3].s_box   3b59cb73fcd90ee05774222dc067fb68
round[ 3].s_row   3bd92268fc74fb735767cbe0c0590e2d
round[ 3].m_col   4c9c1e66f771f0762c3f868e534df256
round[ 3].k_sch   b6ff744ed2c2c9bf6c590cbf0469bf41
```

```
round[ 4].start   fa636a2825b339c940668a3157244d17
round[ 4].s_box   2dfb02343f6d12dd09337ec75b36e3f0
round[ 4].s_row   2d6d7ef03f33e334093602dd5bfb12c7
round[ 4].m_col   6385b79ffc538df997be478e7547d691
round[ 4].k_sch   47f7f7bc95353e03f96c32bcfd058dfd
round[ 5].start   247240236966b3fa6ed2753288425b6c
round[ 5].s_box   36400926f9336d2d9fb59d23c42c3950
round[ 5].s_row   36339d50f9b539269f2c092dc4406d23
round[ 5].m_col   f4bcd45432e554d075f1d6c51dd03b3c
round[ 5].k_sch   3caaa3e8a99f9deb50f3af57adf622aa
round[ 6].start   c81677bc9b7ac93b25027992b0261996
round[ 6].s_box   e847f56514dadde23f77b64fe7f7d490
round[ 6].s_row   e8dab6901477d4653ff7f5e2e747dd4f
round[ 6].m_col   9816ee7400f87f556b2c049c8e5ad036
round[ 6].k_sch   5e390f7df7a69296a7553dc10aa31f6b
round[ 7].start   c62fe109f75eedc3cc79395d84f9cf5d
round[ 7].s_box   b415f8016858552e4bb6124c5f998a4c
round[ 7].s_row   b458124c68b68a014b99f82e5f15554c
round[ 7].m_col   c57e1c159a9bd286f05f4be098c63439
round[ 7].k_sch   14f9701ae35fe28c440adf4d4ea9c026
round[ 8].start   d1876c0f79c4300ab45594add66ff41f
round[ 8].s_box   3e175076b61c04678dfc2295f6a8bfc0
round[ 8].s_row   3e1c22c0b6fcbf768da85067f6170495
round[ 8].m_col   baa03de7a1f9b56ed5512cba5f414d23
round[ 8].k_sch   47438735a41c65b9e016baf4aebf7ad2
round[ 9].start   fde3bad205e5d0d73547964ef1fe37f1
round[ 9].s_box   5411f4b56bd9700e96a0902fa1bb9aa1
round[ 9].s_row   54d990a16ba09ab596bbf40ea111702f
round[ 9].m_col   e9f74eec023020f61bf2ccf2353c21c7
round[ 9].k_sch   549932d1f08557681093ed9cbe2c974e
round[10].start   bd6e7c3df2b5779e0b61216e8b10b689
round[10].s_box   7a9f102789d5f50b2beffd9f3dca4ea7
round[10].s_row   7ad5fda789ef4e272bca100b3d9ff59f
round[10].k_sch   13111d7fe3944a17f307a78b4d2b30c5
round[10].output  69c4e0d86a7b0430d8cdb78070b4c55a
```

Keterangan:

input: plainteks

start: *state* saat mulai *round*[*r*]

s_box: *state* sesudah *SubBytes*()

s_row: *state* sesudah *ShiftRows*()

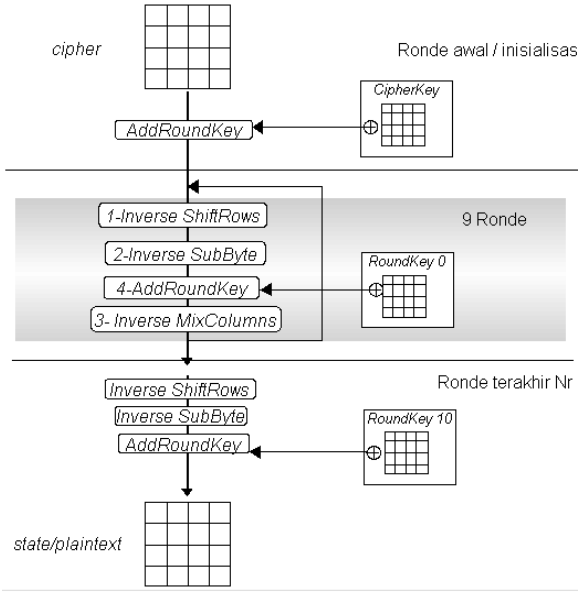
m_col: *state* sesudah *MixColumns*()

k_sch: nilai *key schedule* untuk *round*[*r*]

output: cipherteks

3.2.2.2 Proses Dekripsi *Rijndael*

Transformasi cipher dapat dibalikkan dan diimplementasikan dalam arah yang berlawanan untuk menghasilkan *inverse cipher* yang mudah dipahami untuk algoritma *Rijndael*. Transformasi *byte* yang digunakan pada *invers cipher* adalah *InvShiftRows*, *InvSubBytes*, *InvMixColumns*, dan *AddRoundKey*. Algoritma dekripsi dapat dilihat pada Gambar 3.10 berikut ini: [13]

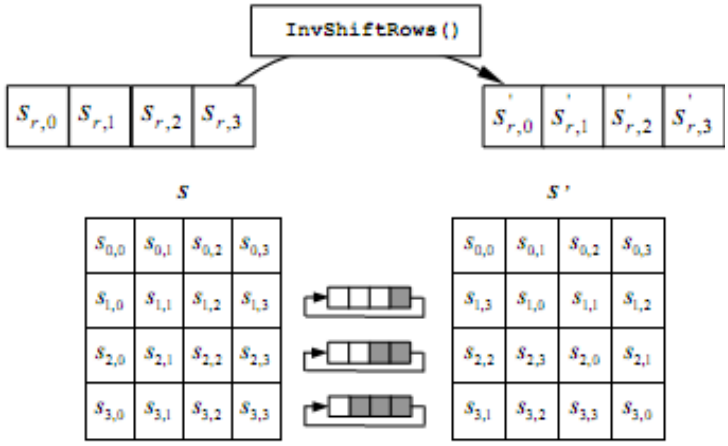


Gambar 3.10 Proses Dekripsi *Rijndael* [8]

1. Transformasi *InvShiftRows*

InvShiftRows adalah kebalikan dari transformasi *ShiftRows*. *Bytes* pada tiga baris terakhir pada *state* bergeser secara siklik dengan jumlah *byte* yang berbeda (*offsets*) seperti pada Gambar 3.11. Baris pertama, $r = 0$, tidak bergeser dan tiga baris terakhir bergeser secara siklik sebesar $Nb - shift(r,Nb)$ *bytes* dimana $shift(r,Nb)$ tergantung dari urutan baris. Secara spesifik, transformasi *InvShiftRows* adalah sebagai berikut:

$$S'_{r,c(c + shift(r,Nb)) \bmod Nb} = S_{r,c} \quad \text{untuk } 0 < r < 4 \text{ dan } 0 \leq c < Nb \qquad (3.4)$$



Gambar 3.11 Transformasi InvShiftRows

2. Transformasi *InvSubBytes*

InvSubBytes adalah kebalikan dari transformasi substitusi *byte*, dimana S-*box* terbalik diterapkan untuk setiap *byte state* seperti pada Tabel 3.2. Ini diperoleh dengan menerapkan kebalikan dari transformasi *affine* diikuti dengan mengambil invers perkalian dalam GF (2^8).

Tabel 3.2 *Inverse* S-*Box*

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

3. Transformasi *InvMixColumns*

InvMixColumns adalah kebalikan dari transformasi *MixColumns*. *InvMixColumns* mengoperasikan *state* kolom demi kolom dan menganggap setiap kolom sebagai polinomial empat-suku. Setiap kolom dianggap polinomial pada GF(2^8) dan dikalikan modulo $x^4 + 1$ dengan persamaan polinomial $a^{-1}(x)$ menjadi

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \qquad (3.5)$$

dapat dijabarkan dalam matriks menjadi

$$s'(x) = a^{-1}(x) \otimes s(x)$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \text{untuk } 0 \leq c < Nb \qquad (3.6)$$

$$S'_{0,c} = (\{0e\} \cdot S_{0,c}) \oplus (\{0b\} \cdot S_{1,c}) \oplus (\{0d\} \cdot S_{2,c}) \oplus (\{09\} \cdot S_{3,c})$$

$$S'_{1,c} = (\{09\} \cdot S_{0,c}) \oplus (\{0e\} \cdot S_{1,c}) \oplus (\{0b\} \cdot S_{2,c}) \oplus (\{0d\} \cdot S_{3,c})$$

$$S'_{2,c} = (\{0d\} \cdot S_{0,c}) \oplus (\{09\} \cdot S_{1,c}) \oplus (\{0e\} \cdot S_{2,c}) \oplus (\{0b\} \cdot S_{3,c})$$

$$S'_{3,c} = (\{0b\} \cdot S_{0,c}) \oplus (\{0d\} \cdot S_{1,c}) \oplus (\{09\} \cdot S_{2,c}) \oplus (\{0e\} \cdot S_{3,c})$$

4. Transformasi *Inverse AddRoundKey*

Pada hakikatnya transformasi *Inverse AddRoundKey* sama dengan transformasi *AddRoundKey*.

Operasi dekripsi *Rijndael* dapat dinyantakan dengan kode semu (*pseudocode*) berikut ini: [13]

Pseudocode InvCipher *Rijndael*:

```
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 Nr-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
```

Pseudocode di atas, dapat diketahui enkripsi dilakukan dengan fungsi *Invcipher* yang memiliki parameter masukan *in* – 16 *byte* dengan Nb adalah panjang blok, keluaran *out* = 16 *byte* dan array 1 dimensi w = 44 *byte* untuk *Rijndael*-128 dengan Nr adalah jumlah ronde.

Berikut ini adalah nilai-nilai dalam array *state* sebagai *cipher* saat berlangsung proses dekripsi untuk panjang blok dan panjang kunci *cipher* masing-masing 16 *byte* (yaitu, Nb = 4 dan Nk = 4).

```
CIPHERTEXT: 69c4e0d86a7b0430d8cdb78070b4c55a
KEY:        000102030405060708090a0b0c0d0e0f

INVERSE CIPHER (DECRYPT):
round[ 0].iinput  69c4e0d86a7b0430d8cdb78070b4c55a
round[ 0].ik_sch  13111d7fe3944a17f307a78b4d2b30c5
round[ 1].istart  7ad5fda789ef4e272bca100b3d9ff59f
round[ 1].is_row  7a9f102789d5f50b2beffd9f3dca4ea7
round[ 1].is_box  bd6e7c3df2b5779e0b61216e8b10b689
round[ 1].ik_sch  549932d1f08557681093ed9cbe2c974e
round[ 1].ik_add  e9f74eec023020f61bf2ccf2353c21c7
round[ 2].istart  54d990a16ba09ab596bbf40ea111702f
round[ 2].is_row  5411f4b56bd9700e96a0902fa1bb9aa1
round[ 2].is_box  fde3bad205e5d0d73547964ef1fe37f1
round[ 2].ik_sch  47438735a41c65b9e016baf4aebf7ad2
round[ 2].ik_add  baa03de7a1f9b56ed5512cba5f414d23
round[ 3].istart  3e1c22c0b6fcbf768da85067f6170495
round[ 3].is_row  3e175076b61c04678dfc2295f6a8bfc0
```

```
round[ 3].is_box  d1876c0f79c4300ab45594add66ff41f
round[ 3].ik_sch  14f9701ae35fe28c440adf4d4ea9c026
round[ 3].ik_add  c57e1c159a9bd286f05f4be098c63439
round[ 4].istart  b458124c68b68a014b99f82e5f15554c
round[ 4].is_row  b415f8016858552e4bb6124c5f998a4c
round[ 4].is_box  c62fe109f75eedc3cc79395d84f9cf5d
round[ 4].ik_sch  5e390f7df7a69296a7553dc10aa31f6b
round[ 4].ik_add  9816ee7400f87f556b2c049c8e5ad036
round[ 5].istart  e8dab6901477d4653ff7f5e2e747dd4f
round[ 5].is_row  e847f56514dadde23f77b64fe7f7d490
round[ 5].is_box  c81677bc9b7ac93b25027992b0261996
round[ 5].ik_sch  3caaa3e8a99f9deb50f3af57adf622aa
round[ 5].ik_add  f4bcd45432e554d075f1d6c51dd03b3c
round[ 6].istart  36339d50f9b539269f2c092dc4406d23
round[ 6].is_row  36400926f9336d2d9fb59d23c42c3950
round[ 6].is_box  247240236966b3fa6ed2753288425b6c
round[ 6].ik_sch  47f7f7bc95353e03f96c32bcfd058dfd
round[ 6].ik_add  6385b79ffc538df997be478e7547d691
round[ 7].istart  2d6d7ef03f33e334093602dd5bfb12c7
round[ 7].is_row  2dfb02343f6d12dd09337ec75b36e3f0
round[ 7].is_box  fa636a2825b339c940668a3157244d17
round[ 7].ik_sch  b6ff744ed2c2c9bf6c590cbf0469bf41
round[ 7].ik_add  4c9c1e66f771f0762c3f868e534df256
round[ 8].istart  3bd92268fc74fb735767cbe0c0590e2d
round[ 8].is_row  3b59cb73fcd90ee05774222dc067fb68
round[ 8].is_box  4915598f55e5d7a0daca94fa1f0a63f7
round[ 8].ik_sch  b692cf0b643dbdf1be9bc5006830b3fe
round[ 8].ik_add  ff87968431d86a51645151fa773ad009
round[ 9].istart  a7be1a6997ad739bd8c9ca451f618b61
round[ 9].is_row  a761ca9b97be8b45d8ad1a611fc97369
round[ 9].is_box  89d810e8855ace682d1843d8cb128fe4
round[ 9].ik_sch  d6aa74fdd2af72fadaa678f1d6ab76fe
round[ 9].ik_add  5f72641557f5bc92f7be3b291db9f91a
round[10].istart  6353e08c0960e104cd70b751bacad0e7
round[10].is_row  63cab7040953d051cd60e0e7ba70e18c
round[10].is_box  00102030405060708090a0b0c0d0e0f0
round[10].ik_sch  000102030405060708090a0b0c0d0e0f
round[10].ioutput 00112233445566778899aabbccddeeff
```

Keterangan:

iinput: cipherteks

istart: *state* saat mulai *round*[*r*]

is_box: *state* sesudah *InvSubBytes*()

is_row: *state* sesudah *InvShiftRows*()

ik_sch: nilai *key schedule* untuk *round*[r]

ik_add: *state* sesudah *AddRoundKey*()

ioutput: plainteks

3.2.3 Analisis Algoritma LZW

Skenario dijelaskan pada tahun 1984 dalam makalahnya, Welch mengkode urutan 8-*bit* data sama dengan 12-*bit* kode. Kode-kode dari 0 sampai 255 mewakili 1 karakter terdiri dari urutan karakter 8-*bit* yang sesuai, dan kode 256 melalui 4095 dibuat di dalam sebuah kamus untuk urutan ditemui dalam data seperti yang dikodekan. Pada setiap tahap dalam kompresi, *byte input* dikumpulkan ke urutan sampai karakter berikutnya akan membuat urutan yang ada kode belum ada dalam kamus. Kode untuk urutan (tanpa karakter yang) yang dipancarkan, dan kode baru (untuk urutan dengan karakter itu) ditambahkan ke dalam kamus.

Idenya adalah cepat beradaptasi pada situasi lain. Dalam sebuah gambar berdasarkan tabel warna, misalnya, alfabet karakter alami adalah himpunan indeks tabel warna, dan pada 1980-an, telah banyak gambar tabel warna yang kecil (di urutan 16 warna). Untuk seperti alfabet yang berkurang, 12-*bit* penuh menghasilkan kode kompresi yang buruk kecuali jika gambar itu besar, sehingga gagasan kode variabel-lebar diperkenalkan: Kode biasanya dimulai satu *bit* lebih luas daripada simbol-simbol yang dikodekan, dan setiap kode yang digunakan ukurannya meningkat sebesar 1 *bit*, sampai batas maksimum yang ditentukan (biasanya 12 *bit*).

Perbaikan lebih lanjut termasuk memesan kode untuk menunjukkan bahwa tabel kode harus dibersihkan (kode yang dibersihkan, biasanya nilai pertama segera setelah nilai untuk karakter alfabet individu), dan kode untuk menunjukkan

akhir data (kode berhenti, biasanya satu lebih besar dari kode yang jelas). Kode yang dibersihkan memungkinkan tabel untuk menginisialisasi ulang setelah mengisi, yang memungkinkan pengkodean beradaptasi dengan perubahan pola dalam data masukan. Kompresor cerdas dapat memantau efisiensi kompresi dan membersihkan tabel setiap kali tabel yang ada tidak lagi sesuai masukan dengan baik.

Karena kode ini ditambahkan dalam cara yang ditentukan oleh data, dekompresor meniru bangunan tabel seperti melihat kode yang dihasilkan. Sangat penting bahwa kompresor dan dekompresor menyepakati berbagai LZW yang digunakan: ukuran dari alfabet, kode lebar maksimum, apakah pengkodean variabel-lebar yang digunakan, ukuran kode awal, apakah akan menggunakan jelas dan berhenti kode (dan apa nilai-nilai yang mereka miliki). Kebanyakan format yang mempekerjakan LZW membangun informasi ini ke dalam spesifikasi format atau menyediakan bidang eksplisit bagi mereka di *header* kompresi untuk data.

3.2.3.1 Proses Kompresi LZW

Metode LZW yang diterapkan dalam aplikasi ini bertipe dinamik, dimana hanya dilakukan satu kali pembacaan (*one-pass*) terhadap *file* yang akan dikompresi. Pengkodean data dilakukan secara *on the fly*, bersamaan dengan proses penambahan *string* baru ke dalam *dictionary*. Pada algoritma ini, setiap *byte* hasil proses kompresi LZW dianggap sebagai 12 *bit*. Sehingga *dictionary* yang dipakai adalah sebesar 4096 (2^{12}) dan dikurangi dengan 256 karakter ASCII. Karena indeks dimulai dari nol, kisaran indeks *dictionary*-nya

dimulai dari 256 sampai dengan 4095. Berikut ini *pseudo code* dari proses kompresi:

```
Initialize table with single character strings
      P = first input character
            WHILE not end of input stream
                  C = next input character
                  IF P + C is in the string table
                        P = P + C
                  ELSE
                        output the code for P
                  END IF
                  add P + C to the string table
                  P = C
            END WHILE
      output code for P
```

Sebuah kamus diinisialisasi mengandung karakter *string* tunggal sesuai dengan semua karakter masukan yang mungkin (dan tidak ada yang lain kecuali kode yang jelas dan berhenti jika mereka sedang digunakan). Algoritma ini bekerja dengan memindai melalui *string input* untuk sub*string* berturut lagi sampai menemukan satu yang tidak ada dalam kamus. Ketika seperti *string* ditemukan, indeks untuk *string* yang kurang karakter terakhir (yaitu, sub*string* terpanjang yang ada di kamus) diambil dari kamus dan dikirim ke *output*, dan *string* baru (termasuk karakter terakhir) ditambahkan ke kamus dengan kode yang tersedia berikutnya. Karakter terakhir masukan ini kemudian digunakan sebagai titik awal berikutnya untuk memindai sub*string*.

Sebagai contoh, string “MAMA MASAK SAGU” akan dikompresi dengan LZW. Isi *dictionary* pada awal proses diset dengan tujuh karakter dasar yang ada: “M”, “A”, “_”, “S”, “K”, “G”, “U”. *Bit dictionary* yang dipakai adalah 8 *bit*. Tahapan proses kompresi ditunjukkan pada Tabel 3.3.

Input String: ‘M’,’A’,’M’,’A’,’_’,’M’,’A’,’S’,’A’,’K’,’_’,’S’,’A’,’G’,’U’

bit dictionary: 8 *bit*

Output String: ‘M’,‘A’,‘256’,‘_’,‘256’,‘S’,‘A’,‘K’,‘_’,‘261’,‘G’,‘U’

Tabel 3.3 Proses Kompresi LZW

No	*Input*	*String + Char (UCode)*	*in Dictionary?*	*Temporary*	*Dictionary*		*Output*
					code	*char*	
1	M	M	-	M	-	-	none
2	A	MA	N	A	256	MA	M
3	M	AM	N	M	257	AM	A
4	A	MA	Y (256)	MA	-	-	none
5	_	MA_	N	_	258	MA_	MA
6	M	_M	N	M	259	_M	_
7	A	MA	Y (256)	MA	-	-	none
8	S	MAS	N	S	260	MAS	MA
9	A	SA	N	A	261	SA	S
10	K	AK	N	K	262	AK	A
11	_	K_	N	_	263	K_	K
12	S	_S	N	S	264	_S	_
13	A	SA	Y (261)	SA	-	-	none
14	G	SAG	N	G	265	SAG	SA
15	U	GU	N	U	266	GU	G
16							U

Total *Input* sebelum kompresi = Total *Input* * *bit dictionary*

= 15 * 8 *bit*

= 120 *bit*

Total *Output* sesudah kompresi = Total *Output* * *bit dictionary*

= 12 * 8 *bit*

= 96 *bit*

Rasio kompresi = Total *Output*/Total *Input* * 100%

= 96/120 * 100%

= 80%

3.2.3.2 Proses Dekompresi LZW

Proses dekompresi pada LZW dilakukan dengan prinsip yang sama seperti proses kompresi. Pada awalnya, *dictionary* diinisialisasi dengan semua karakter dasar yang ada. Lalu pada setiap langkah, kode dibaca satu per satu dari stream kode, dikeluarkan string dari *dictionary* yang berkorespondensi dengan kode tersebut, dan ditambahkan string baru ke dalam dictionary. Metode LZW yang diterapkan dalam penelitian ini bertipe dinamik, dimana hanya dilakukan satu kali pembacaan (*one-pass*) terhadap *file* yang akan dikompresi. Pengkodean data dilakukan secara *on the fly*, bersamaan dengan proses penambahan string baru ke dalam *dictionary*. Algoritma dekompresinya sebagai berikut:

```
Initialize table with single character strings
      OLD = first input code output translation of OLD
      WHILE not end of input stream
            NEW = next input code
            IF NEW is not in the string table
                  S = translation of OLD
                  S = S + C
            ELSE
                  S = translation of NEW
            END IF
            output S
            C = first character of S
            OLD + C to the string table
            OLD = NEW
      END WHILE
```

Algoritma dekompresi bekerja dengan membaca nilai dari *input* dikodekan dan menghasilkan *string* yang sesuai dari kamus diinisialisasi. Pada saat yang sama memperoleh nilai berikutnya dari *input*, dan menambah kamus gabungan dari *string* dan hanya *output* karakter pertama dari *string* yang diperoleh oleh dekompresi nilai *input* berikutnya. Dekompresor kemudian hasil untuk nilai *input* berikutnya (yang sudah dibaca sebagai "nilai berikutnya" dalam pass sebelumnya)

dan mengulangi proses tersebut sampai tidak ada *input* lebih, di mana titik nilai *input* akhir diterjemahkan tanpa tambahan lagi ke dalam kamus.

Dengan cara ini dekompresor membangun sebuah kamus yang identik dengan yang digunakan oleh kompresor, dan menggunakannya untuk memecahkan kode nilai *input* berikutnya. Jadi kamus penuh tidak perlu dikirim dengan data dikodekan; hanya kamus awal berisi karakter tunggal-*string* adalah cukup (dan biasanya didefinisikan terlebih dahulu dalam kompresor dan dekompresor bukannya eksplisit dikirim dengan data dikodekan). Algoritma dekompresi bekerja dengan membaca nilai dari *input* dikodekan dan menghasilkan *string* yang sesuai dari kamus diinisialisasi. Pada saat yang sama memperoleh nilai berikutnya dari *input*, dan menambah kamus gabungan dari *string* dan hanya *output* karakter pertama dari *string* yang diperoleh oleh dekompresi nilai *input* berikutnya. Dekompresor kemudian hasil untuk nilai *input* berikutnya (yang sudah dibaca sebagai "nilai berikutnya" dalam pass sebelumnya) dan mengulangi proses tersebut sampai tidak ada *input* lebih, di mana titik nilai *input* akhir diterjemahkan tanpa tambahan lagi ke dalam kamus.

Dengan cara ini dekompresor membangun sebuah kamus yang identik dengan yang digunakan oleh kompresor, dan menggunakannya untuk memecahkan kode nilai *input* berikutnya. Jadi kamus penuh tidak perlu dikirim dengan data dikodekan; hanya kamus awal berisi karakter tunggal-*string* adalah cukup (dan biasanya didefinisikan terlebih dahulu dalam kompresor dan dekompresor bukannya eksplisit dikirim dengan data dikodekan).

Sebagai contoh, output kode ‘M’,‘A’,‘256’,‘_’,‘256’,‘S’,‘A’,‘K’,‘_’,‘261’,‘G’,‘U’ didekompresi dengan algoritma LZW. *Bit dictionary* yang dipakai adalah 8 *bit*. Tahapan proses dekompresi ditunjukkan pada Tabel 3.4.

Input String: ‘M’,‘A’,‘256’,‘_’,‘256’,‘S’,‘A’,‘K’,‘_’,‘261’,‘G’,‘U’

bit dictionary: 8 *bit*

Tabel 3.4 Proses Dekompresi LZW

No	*Input*	*String*	*Char*	*String + Char*	*Dictionary*		*Output*
					code	*char*	
1	M	M	-	-	-		M
2	A	M	A	MA	256	MA	A
3	256	A	MA	AM	257	AM	MA
4	_	MA	_	MA_	258	MA_	_
5	256	_	MA	_M	259	_M	MA
6	S	MA	S	MAS	260	MAS	S
7	A	S	A	SA	261	SA	A
8	K	A	K	AK	262	AK	K
9	_	K	_	K_	263	K_	_
10	261	_	SA	_S	264	_S	SA
11	G	SA	G	SAG	265	SAG	G
12	U	G	U	GU	266	GU	U

3.2.4 Analisis Kebutuhan Non-Fungsional

Analisis kebutuhan non-fungsional merupakan analisis yang dibutuhkan untuk menentukan spesifikasi kebutuhan sistem. Spesifikasi ini juga meliputi elemen atau komponen-komponen apa saja yang dibutuhkan untuk sistem yang akan dibangun sampai dengan sistem tersebut diimplementasikan. Analisis kebutuhan ini juga menentukan spesifikasi masukan yang diperlukan sistem, keluaran yang akan dihasilkan sistem dan proses yang dibutuhkan untuk mengolah masukan sehingga menghasilkan suatu keluaran yang diinginkan.

3.2.4.1 Analisis Kebutuhan Perangkat Keras

Analisis kebutuhan perangkat keras merupakan analisis penggunaan perangkat keras pada sistem keamanan data di PT. Prospera Consulting Engineers yang akan mendukung pembangunan sistem dari awal pembuatan hingga pengujian sistem. Adapun perangkat keras yang dibutuhkan yaitu dengan spesifikasi yang dapat dilihat di Tabel 3.5.

Tabel 3.5 Spesifikasi Min Perangkat Keras Kebutuhan Sistem

No	Perangkat Keras	Spesifikasi
1	Prosessor	Intel Pentium 4 (1,6 GHz)
2	Monitor	Monitor CRT 17 *inch*
3	VGA	VGA Card On-Board 128 Mb
4	Memori	Memori DDR 512 MB
5	Media Penyimpanan	HDD kapasitas bebas minimal 1 Gb
6	Media Masukan	*Keyboard* dan *Mouse*

Sedangkan spesifikasi perangkat keras yang terdapat di PT. Prospera Consulting Engineers (PCE) dapat dilihat pada Tabel 3.6 sebagai berikut:

Tabel 3.6 Spesifikasi Perangkat Keras di PT. Prospera Consulting Engineers

No	Perangkat Keras	Spesifikasi
1	Prosessor	Intel Pentium Core 2 Duo (2,8 GHz)
2	Monitor	Monitor LCD 17 *inch*
3	VGA	VGA Card On-Board 128 Mb
4	Memori	Memori DDR3 2 GB
5	Media Penyimpanan	HDD 320 GB (kapasitas bebas 70 Gb)
6	Media Masukan	*Keyboard* dan *Mouse*

Setelah analisis spesifikasi perangkat keras di PT. Prospera Consulting Engineers dengan membandingkan kebutuhan spesifikasi minimal untuk pembuatan sistem, dapat disimpulkan spesifikasi perangkat keras sudah memenuhi kebutuhan perangkat keras dari sistem keamanan data.

3.2.4.2 Analisis Kebutuhan Perangkat Lunak

Pembangunan sistem kemanan data di PT. Prospera Consulting Engineers, selain kebutuhan perangkat keras yang harus diperhatikan juga harus mengevaluasi mengenai pemilihan perangkat lunak untuk mendukung sistem yang dibangun. Adapun spesifikasi perangkat lunak yang dibutuhkan dalam pembangunan sistem keamanan data dapat dilihat pada Tabel 3.7:

Tabel 3.7 Spesifikasi Kebutuhan Perangkat Lunak

No	Perangkat Lunak	Spesifikasi
1	Sistem Operasi	Microsoft Windows XP
2	*Client* .NET *Framework*	.NET Framework 3.5

Sedangkan spesifikasi perangkat lunak yang terdapat di PT. Prospera Consulting Engineers dapat dilihat pada Tabel 3.8 sebagai berikut:

Tabel 3.8 Spesifikasi Perangkat Lunak Di PT. Prospera Consulting Engineers

No	Perangkat Lunak	Spesifikasi
1	Sistem Operasi	Microsoft Windows 7
2	*Client* .NET *Framework*	.NET Framework 3.5 SP 1

Setelah evaluasi mengenai spesifikasi perangkat lunak di PT. Prospera Consulting Engineers dengan membandingkan kebutuhan spesifikasi minimal

untuk pembangunan sistem, dapat disimpulkan spesifikasi perangkat lunak sudah memenuhi kebutuhan perangkat lunak dari sistem keamanan data.

3.2.5 Analisis Kebutuhan Perangkat Pikir

Suatu sistem keamanan data akan berjalan optimal apabila ditunjang oleh perangkat pikir yang memiliki kebutuhan dalam menjaga kerahasiaan suatu data atau informasi. Adapun perangkat pikir (pengguna) yang dibutuhkan untuk menjalankan sistem keamanan data adalah sebagai berikut:

1. Pengguna

Umur	:	22-50 tahun
Pendidikan terakhir	:	D3/S1
Lingkup pekerjaan	:	bagian keuangan, bagian administrasi atau bagian pusat data/informasi

Sedangkan perangkat pikir yang berada di lingkungan PT. Prospera Consulting Engineers adalah sebagai berikut:

1. Pengguna

Umur	:	33-47 tahun
Pendidikan terakhir	:	S1
Lingkup pekerjaan	:	bagian keuangan, bagian administrasi, atau bagian pusat data/informasi

Setelah evaluasi mengenai perangkat pikir di PT. Prospera Consulting Engineers dengan membandingkan kebutuhan perangkat pikir yang dibutuhkanal untuk pembangunan sistem, dapat disimpulkan perangkat pikir sudah memenuhi kebutuhan perangkat pikir dari sistem keamanan data.

3.2.6 Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional menggambarkan proses kegiatan yang akan diterapkan dalam sistem dan menjelaskan kebutuhan yang diperlukan agar sistem dapat berjalan dengan baik serta sesuai dengan kebutuhan sistem keamanan data.

Analisis yang dilakukan dimodelkan dengan menggunakan *Unified Modeling Language* (UML). Tahapan pemodelan dalam analisis tersebut antara lain mengidentifikasi aktor, pembuatan *use case diagram, Use Case Scenario, activity diagram, Sequence Diagram, class diagram, dan state diagram*.

3.2.6.1 Identifikasi Aktor

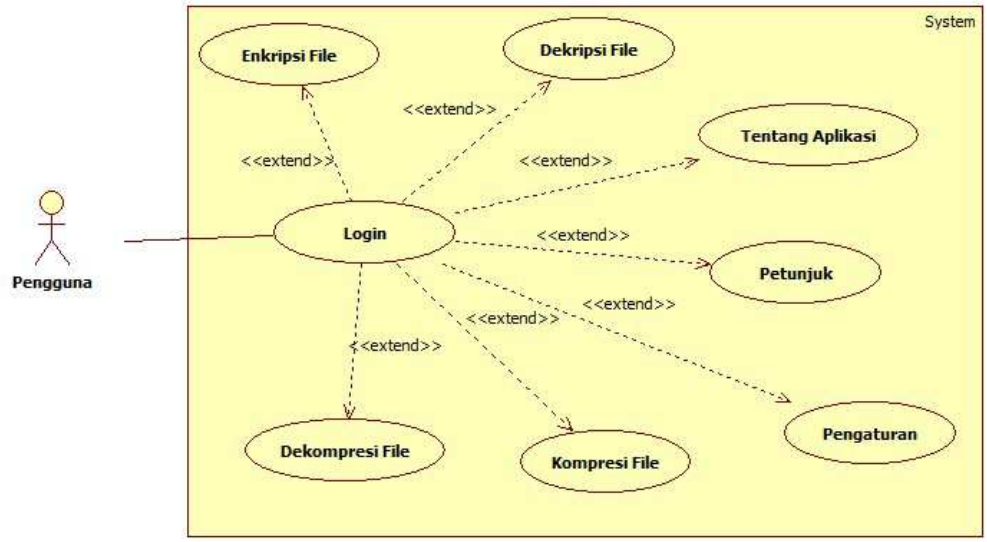
Aktor yang telibat pada sistem keamanan data ada satu aktor yaitu pengguna. Deskripsi untuk setiap aktornya akan dijelaskan pada Tabel 3.9 sebagai berikut:

Tabel 3.9 Identifikasi Aktor

No	Aktor	Deskripsi
1	Pengguna	Sebagai pengatur atau pegawai yang mempunyai hak akses mengelola data penting perusahaan dan akses masuk ke dalam sistem.

3.2.6.2 *Use Case Diagram*

Use case diagram adalah diagram yang menunjukkan fungsionalitas suatu sistem atau kelas dan bagaimana sistem tersebut berinteraksi dengan dunia luar dan menjelaskan sistem secara fungsional yang terlihat oleh pengguna. Dari identifikasi aktor yang terlibat di atas maka *use case* diagram dapat dilihat pada Gambar 3.12.



Gambar 3.12 ***Use Case Diagram***

3.2.6.3 *Use Case Scenario*

Use Case Scenario bertujuan untuk mendeskripsikan urutan langkah-langkah dalam proses bisnis baik yang dilakukan aktor terhadap sistem maupun yang dilikakukan oleh sistem terhadap aktor. Berdasarkan *use case diagram* pada Gambar 3.7., maka *Use Case Scenario* untuk sistem yang dibangun terdiri dari *scenario* Login, *scenario* Enkripsi File, *scenario* Dekripsi File, *scenario* Kompresi File, *scenario* Dekompresi File, *scenario* Pengaturan, *scenario* Tentang Aplikasi, dan *scenario* Petunjuk.

1. *Use Case* Login

Interaksi antara aktor pengguna dengan *use case* Login dijelaskan dalam Tabel 3.10 sebagai berikut:

Tabel 3.10 ***Use Case Scenario*** **Login**

Identifikasi Masalah	
Nama	Login
Tujuan	Masuk ke dalam menu utama

Identifikasi Masalah	
Deskripsi	Proses ini melakukan autentikasi pengguna sistem
Aktor	Pengguna
Usecase Yang Berkaitan	-
Skenario Utama	
Kondisi Awal	*Form* Login ditampilkan
Aksi Aktor	**Reaksi Sistem**
1) Mengisi *password* login.	
2) Menekan tombol Login.	3) Menampilkan menu utama
Skenario Alternatif (Jika Gagal)	
Aksi Aktor	**Reaksi Sistem**
	1. a. Menampilkan pesan kesalahan *password* login
	b. Menampilkan *form* Login
c. Kembali ke langkah nomor 1	
Kondisi Akhir	Pengguna dapat melakukan kegiatan di dalam sistem

2. *Use Case Scenario* Enkripsi File

Interaksi antara aktor pengguna dengan *use case* Enkripsi File dijelaskan dalam Tabel 3.11 sebagai berikut:

Tabel 3.11 *Use Case Scenario* Enkripsi File

Identifikasi Masalah	
Nama	Enkripsi File
Tujuan	Melakukan proses enkripsi *file*
Deskripsi	Proses ini melakukan enkripsi *file* yang ditentukan
Aktor	Pengguna
Usecase Yang Berkaitan	-
Skenario Utama	
Kondisi Awal	*Form* Enkripsi File ditampilkan
Aksi Aktor	**Reaksi Sistem**
1) Menekan tombol Browse.	
2) Memilih *file* yang tidak terenkripsi.	3) Menampilkan alamat *file* yang akan dienkripsi dan alamat *file* hasil enkripsi. Melakukan proses enkripsi.
4) Mengisi *password* dan konfirmasi *password*.	

Identifikasi Masalah	
5) Menekan tombol Enkrip.	6) Melakukan proses enkripsi *file*.
	7) Hapus *file* masukan.
	8) Simpan *file* terenkripsi.
	9) Menampilkan pesan Enkripsi File Berhasil.
Skenario Alternatif (Jika Gagal)	
Aksi Aktor	**Reaksi Sistem**
	6) a. 1. Menampilkan pesan kesalahan *password* kosong
2. Kembali ke langkah nomor 4.	
	6) b. 1. Menampilkan pesan kesalahan *password* enkripsi *file* tidak sama.
	2. Menampilkan *form* Enkripsi File yang sudah terisi *file* yang akan dienkripsi dan mengosongkan kolom *password* dan konfirmasi *password*
3. Kembali ke langkah nomor 4	
Kondisi Akhir	*File* masukan terenkripsi

3. *Use Case Scenario* Dekripsi File

Interaksi antara aktor pengguna dengan *use case* Dekripsi File dijelaskan dalam Tabel 3.12 sebagai berikut:

Tabel 3.12 *Use Case Scenario* Dekripsi File

Identifikasi Masalah	
Nama	Dekripsi File
Tujuan	Melakukan proses dekripsi *file*
Deskripsi	Proses ini melakukan dekripsi *file* yang terenkripsi oleh sistem
Aktor	Pengguna
Usecase Yang Berkaitan	-
Skenario Utama	
Kondisi Awal	*Form* Dekripsi File ditampilkan
Aksi Aktor	**Reaksi Sistem**
1) Menekan tombol Browse.	
2) Memilih *file* yang terenkripsi.	3) Menampilkan alamat *file* yang akan didekripsi dan alamat *file* hasil dekripsi. Melakukan proses

Identifikasi Masalah	
	dekripsi
4) Mengisi *password* untuk dekripsi *file*.	
5) Menekan tombol Dekrip.	6) Melakukan proses dekripsi *file*.
	7) Hapus *file* masukan.
	8) Simpan *file* terdekripsi.
	9) Menampilkan pesan Dekripsi File Berhasil.
Skenario Alternatif (Jika Gagal)	
Aksi Aktor	**Reaksi Sistem**
	6) a. Menampilkan pesan kesalahan *password* dekripsi *file*.
	b. Menampilkan *form* Dekripsi File yang sudah terisi *file* yang akan dienkripsi dan mengosongkan kolom *password.*
c. Kembali ke langkah nomor 4.	
Kondisi Akhir	*File* masukan terdekripsi.

4. *Use Case Scenario* Kompresi File

Interaksi antara aktor pengguna dengan *use case* Kompresi File dijelaskan dalam Tabel 3.13 sebagai berikut:

Tabel 3.13 *Use Case Scenario* Compress File

Identifikasi Masalah	
Nama	Kompresi File
Tujuan	Melakukan kompresi *file*
Deskripsi	Proses ini melakukan kompresi *file* yang ditentukan
Aktor	Pengguna
Usecase Yang Berkaitan	-
Skenario Utama	
Kondisi Awal	*Form* Kompresi File ditampilkan
Aksi Aktor	**Reaksi Sistem**
1) Menekan tombol Browse.	
2) Memilih *File* yang tidak terkompresi.	3) Menampilkan alamat *file* yang akan dikompresi dan alamat *file* hasil kompresi.
4) Menekan tombol Kompres.	5) Melakukan proses kompresi dan menghasilkan *file* terkompresi.

Identifikasi Masalah	
	6) Menampilkan pesan Kompresi File Berhasil.
Kondisi Akhir	*File* masukan terkompresi.

5. *Use Case Scenario* Dekompresi File

Interaksi antara aktor pengguna dengan *use case* Dekompresi File dijelaskan dalam Tabel 3.14 sebagai berikut:

Tabel 3.14 *Use Case Scenario* Dekompresi File

Identifikasi Masalah	
Nama	Dekompresi File
Tujuan	Melakukan dekompresi *file.*
Deskripsi	Proses ini melakukan dekompresi *file* yang terkompresi oleh sistem.
Aktor	Pengguna
Usecase Yang Berkaitan	-
Skenario Utama	
Kondisi Awal	*Form* Dekompresi File ditampilkan
Aksi Aktor	**Reaksi Sistem**
1) Menekan tombol Browse.	
2) Memilih *file* yang terkompresi.	3) Menampilkan alamat *file* yang akan didekompresi dan alamat *file* hasil dekompresi.
4) Menekan tombol Dekompres	5) Melakukan proses dekompresi dan menghasilkan *file* terdekompresi.
	6) Menampilkan pesan Dekompresi File Berhasil
Kondisi Akhir	*File* masukan terdekompresi

6. *Use Case Scenario* Pengaturan

Interaksi antara aktor pengguna dengan *use case* Pengaturan dijelaskan dalam Tabel 3.15 sebagai berikut:

Tabel 3.15 *Use Case Scenario* Pengaturan

Identifikasi Masalah	
Nama	Pengaturan
Tujuan	Memperbaharui *password* login

Identifikasi Masalah	
Deskripsi	Proses ini mengganti *password* untuk login sistem
Aktor	Pengguna
Usecase Yang Berkaitan	-
Skenario Utama	
Kondisi Awal	*Form* Pengaturan ditampilkan
Aksi Aktor	**Reaksi Sistem**
1) Mengisi *password* lama.	
2) Mengisi *password* baru dan konfirmasi *password*.	
3) Menekan tombol Simpan.	4) Menyimpan *password* login baru di dalam sistem.
	5) Menampilkan pesan Password Login Berhasil Diperbaharui.
Skenario Alternatif (Jika Gagal)	
Aksi Aktor	**Reaksi Sistem**
	4) a. 1. Menampilkan pesan kesalahan *password* lama.
	2. Menampilkan *form* Pengaturan dalam keadaan kosong.
3. Kembali ke langkah nomor 1.	
	4) b. 1. Menampilkan pesan kesalahan *password*.
	2. Menampilkan *form* Pengaturan yang sudah terisi kode keamanan serta mengosongkan kolom *password* dan konfirmasi *password*.
3. Kembali ke langkah nomor 2.	
Kondisi Akhir	*Password* login sistem terbaharui.

7. *Use Case Scenario* Tentang Aplikasi

Interaksi antara aktor pengguna dengan *use case* Tentang Aplikasi dijelaskan dalam Tabel 3.16 sebagai berikut:

Tabel 3.16 *Use Case Scenario* Tentang Aplikasi

Identifikasi Masalah	
Nama	Tentang Aplikasi
Tujuan	Melihat informasi aplikasi

Identifikasi Masalah	
Deskripsi	Proses ini menampilkan informasi tentang aplikasi
Aktor	Pengguna
Usecase Yang Berkaitan	-
Skenario Utama	
Kondisi Awal	*Form* Tentang Aplikasi ditampilkan
Aksi Aktor	**Reaksi Sistem**
1) Menekan tombol OK	2) Kembali ke menu utama
Kondisi Akhir	Pengguna hanya dapat membaca informasi mengenai aplikasi.

8. *Use Case Scenario* Petunjuk

Interaksi antara aktor pengguna dengan *use case* Petunjuk dijelaskan dalam Tabel 3.17 sebagai berikut:

Tabel 3.17 *Use Case Scenario* Petunjuk

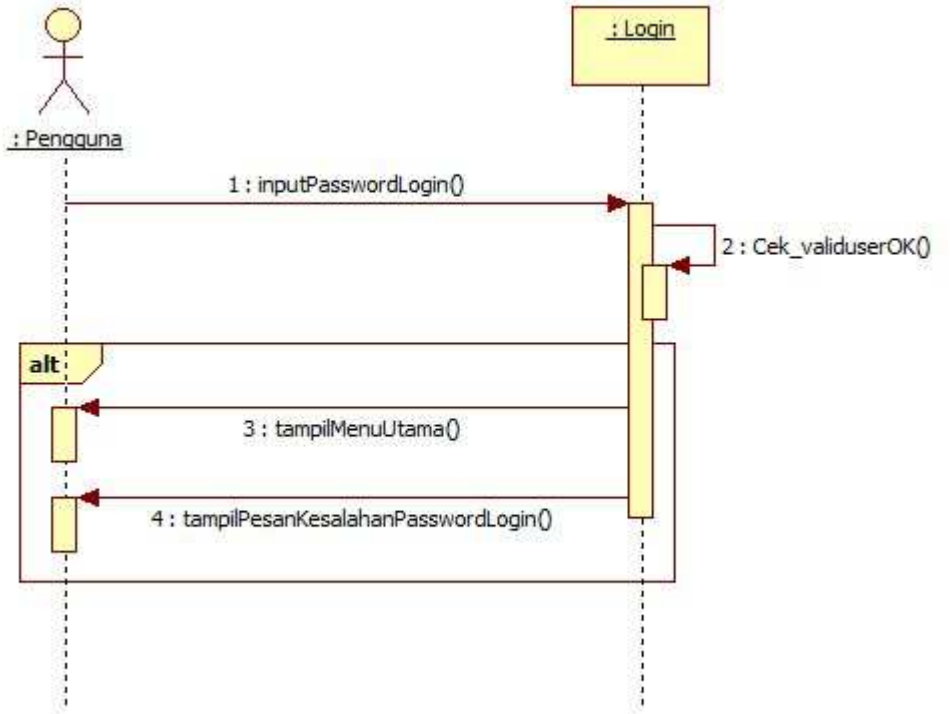
Identifikasi Masalah	
Nama	Petunjuk
Tujuan	Membaca petunjuk penggunaan aplikasi
Deskripsi	Proses ini informasi petunjuk penggunaan aplikasi
Aktor	Pengguna
Usecase Yang Berkaitan	-
Skenario Utama	
Kondisi Awal	*Form* Petunjuk ditampilkan
Aksi Aktor	**Reaksi Sistem**
1) Menekan tombol OK	2) Kembali ke halaman Utama
Kondisi Akhir	Pengguna hanya dapat membaca informasi mengenai petunjuk penggunaan aplikasi.

3.2.6.4 *Sequence Diagram*

Sequence diagram menggambarkan interaksi antar masing-masig objek pada setiap *Use Case* dalam urutan waktu. Interaksi ini berupa pengiriman serangkaian data antar objek-objek yang saling berinteraksi. Berikut ini penjelasan dari masing-masing *Sequence Diagram*:

1. *Sequence Diagram* Login

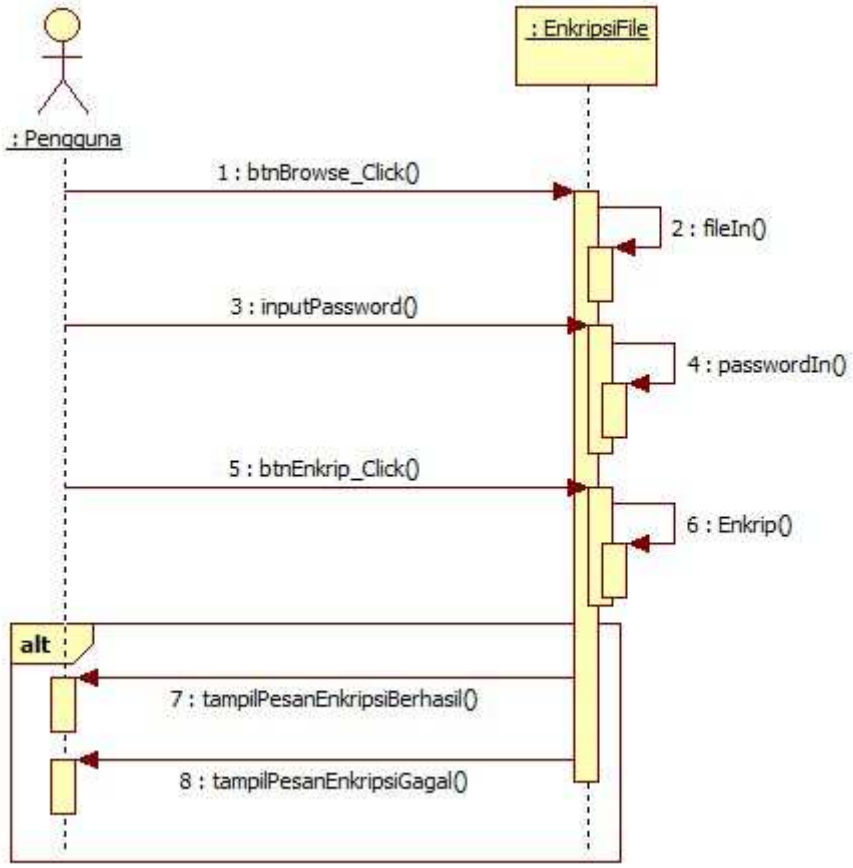
Interaksi antara aktor pengguna dengan *use case* Login dijelaskan dalam *Sequence Diagram* pada Gambar 3.13 sebagai berikut:



Gambar 3.13 *Sequence Diagram* Login

2. *Sequence Diagram* Enkripsi File

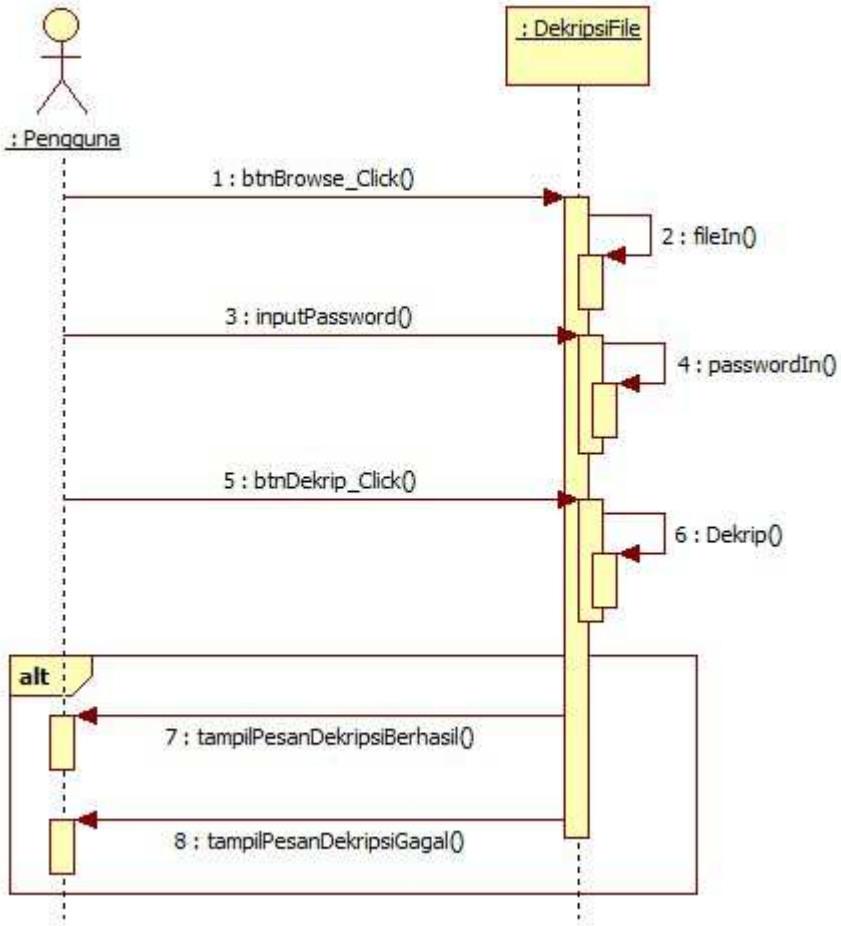
Interaksi antara aktor pengguna dengan *use case* Enkripsi File dijelaskan dalam *Sequence Diagram* pada Gambar 3.14 sebagai berikut:



Gambar 3.14 *Sequence Diagram* Enkripsi File

3. *Sequence Diagram* Dekripsi File

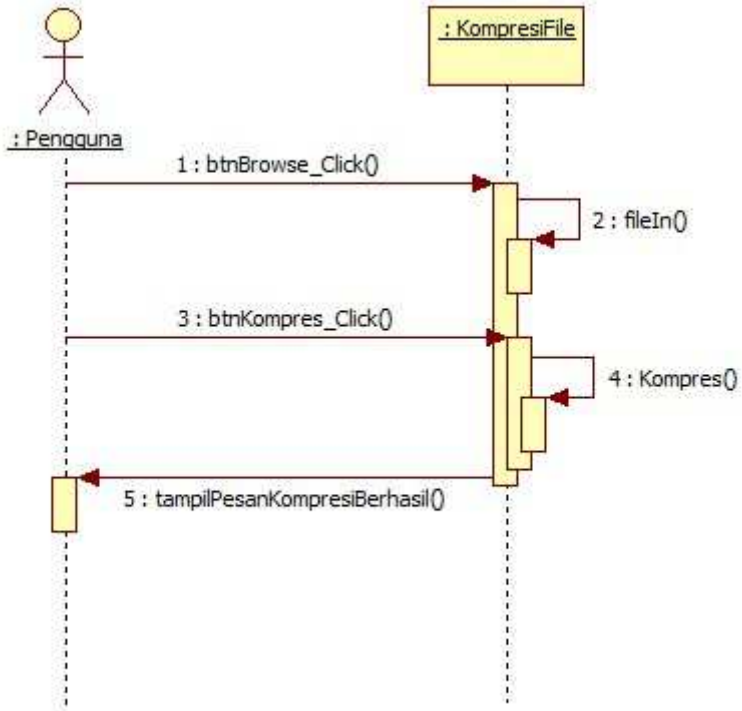
Interaksi antara aktor pengguna dengan *use case* Dekripsi File dijelaskan dalam *Sequence Diagram* pada Gambar 3.15 sebagai berikut:



Gambar 3.15 *Sequence Diagram* Dekripsi File

4. *Sequence Diagram* Kompresi File

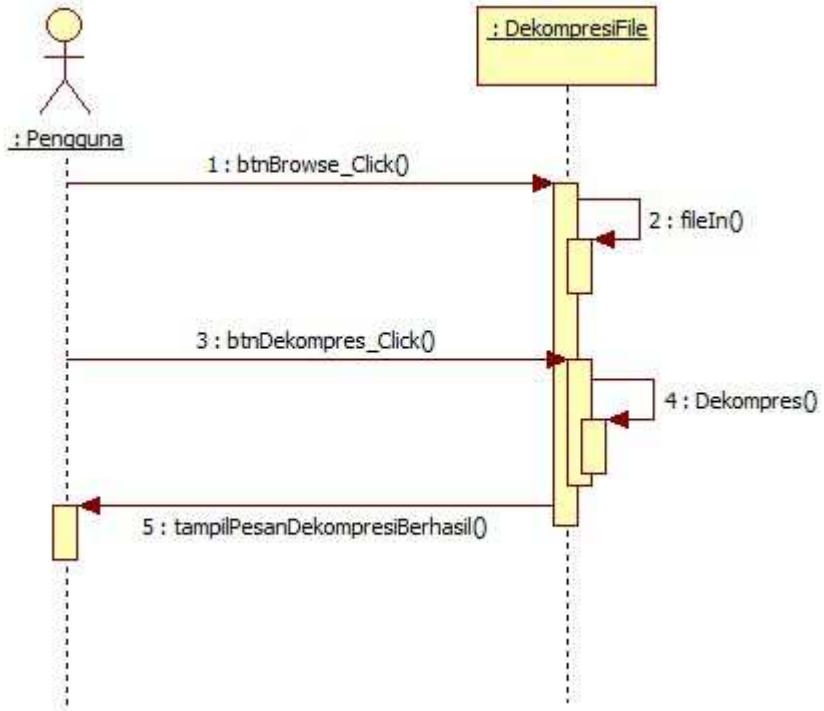
Interaksi antara aktor pengguna dengan *use case* Kompresi File dijelaskan dalam *Sequence Diagram* pada Gambar 3.16 sebagai berikut:



Gambar 3.16 *Sequence Diagram* Kompresi File

5. *Sequence Diagram* Dekompresi File

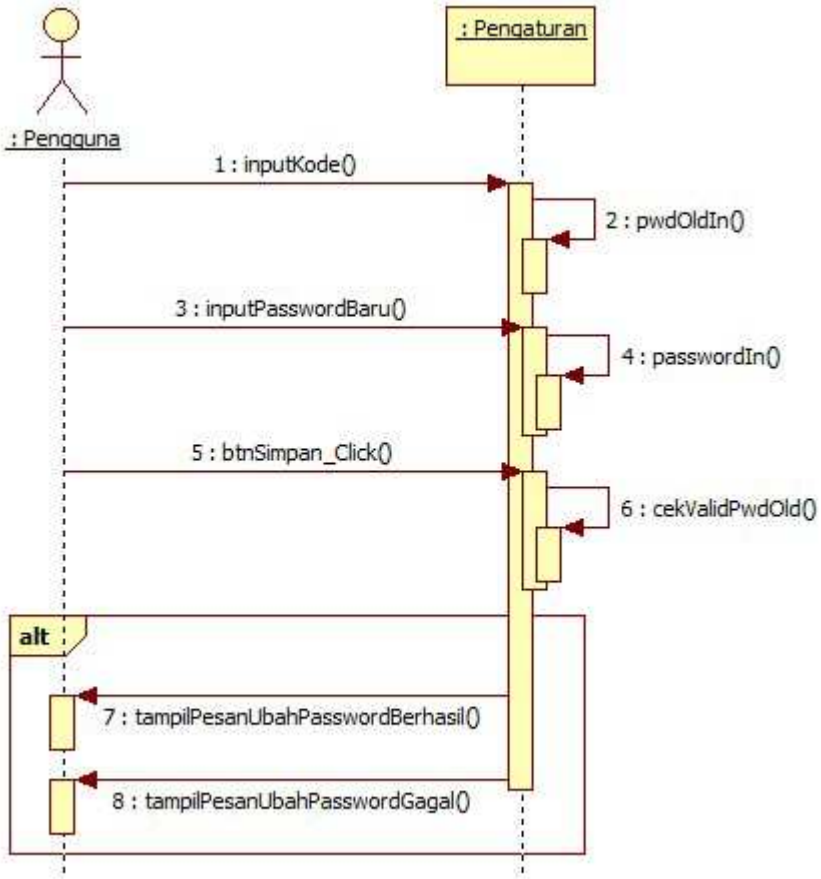
Interaksi antara aktor pengguna dengan *use case* Dekompresi File dijelaskan dalam *Sequence Diagram* pada Gambar 3.17 sebagai berikut:



Gambar 3.17 *Sequence Diagram* Dekompresi File

6. *Sequence Diagram* Pengaturan

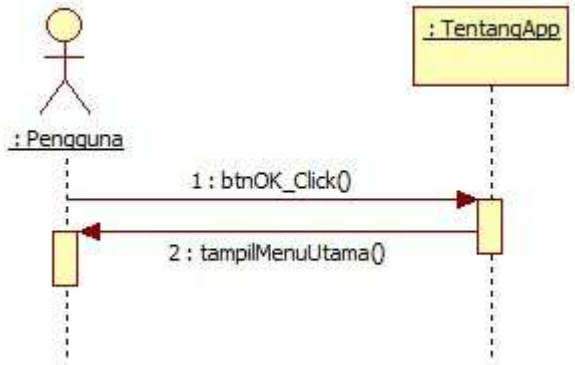
Interaksi antara aktor pengguna dengan *use case* Pengaturan dijelaskan dalam *Sequence Diagram* pada Gambar 3.18 sebagai berikut:



Gambar 3.18 *Sequence Diagram* Pengaturan

7. *Sequence Diagram* Tentang Aplikasi

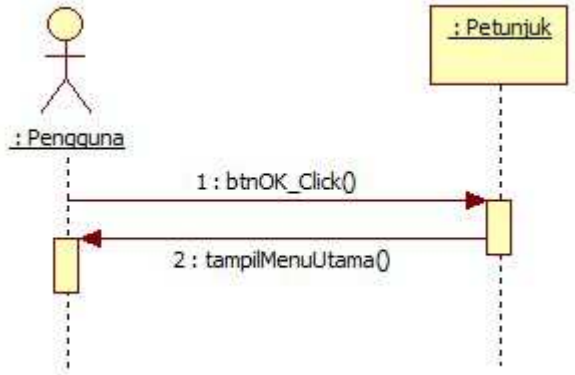
Interaksi antara aktor pengguna dengan *use case* Tentang Aplikasi dijelaskan dalam *Sequence Diagram* pada Gambar 3.19 sebagai berikut:



Gambar 3.19 *Sequence Diagram* Tentang Aplikasi

8. *Sequence Diagram* Petunjuk

Interaksi antara aktor pengguna dengan *use case* Petunjuk dijelaskan dalam *Sequence Diagram* pada Gambar 3.20 sebagai berikut:



Gambar 3.20 *Sequence Diagram* Petunjuk

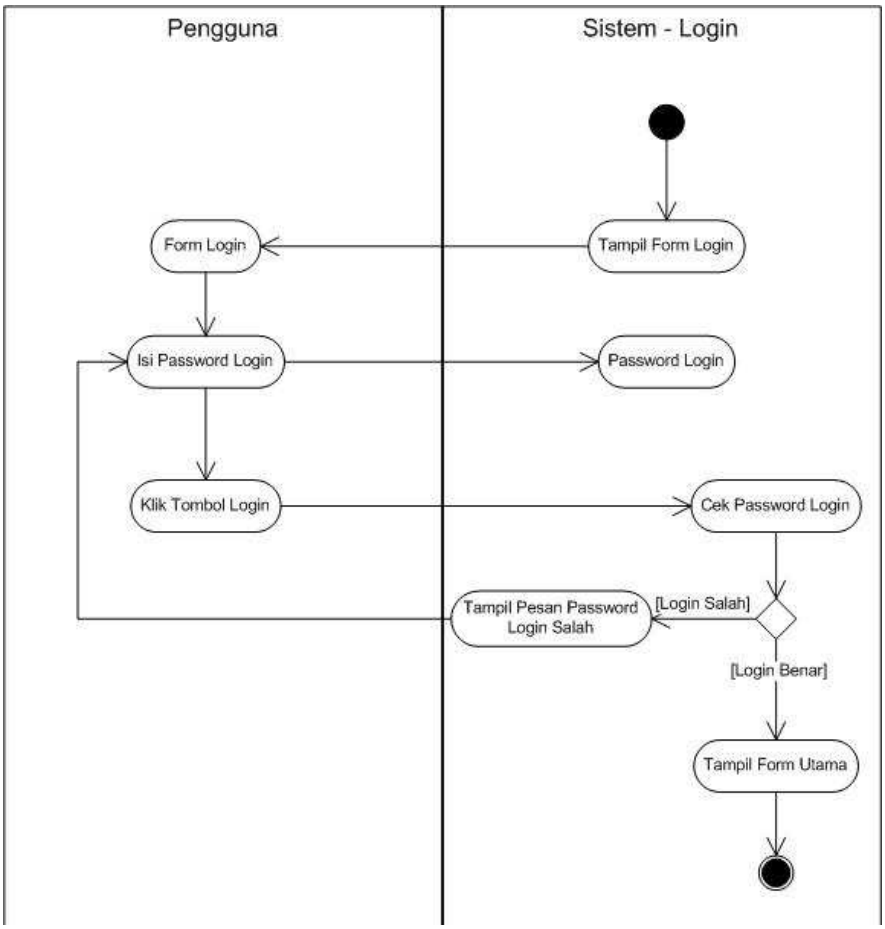
3.2.6.5 *Activity Diagram*

Activity diagram memodelkan aliran kerja atau *workflow* dari urutan aktifitas dalam suatu proses yang mengacu pada *use case diagram* yang ada. *Activity diagram* pada aplikasi keamanan data terdiri dari *activity diagram* Login, *activity diagram* Enkripsi File, *activity diagram* Dekripsi File, *activity diagram*

Kompresi File, *activity diagram* Dekompresi File, *activity diagram* Tentang Aplikasi, dan *activity diagram* Petunjuk. Berikut ini penjelasan dengan *activity diagram* untuk penjelasan dari masing-masing *use case scenario*.

1. *Activity Diagram* Login

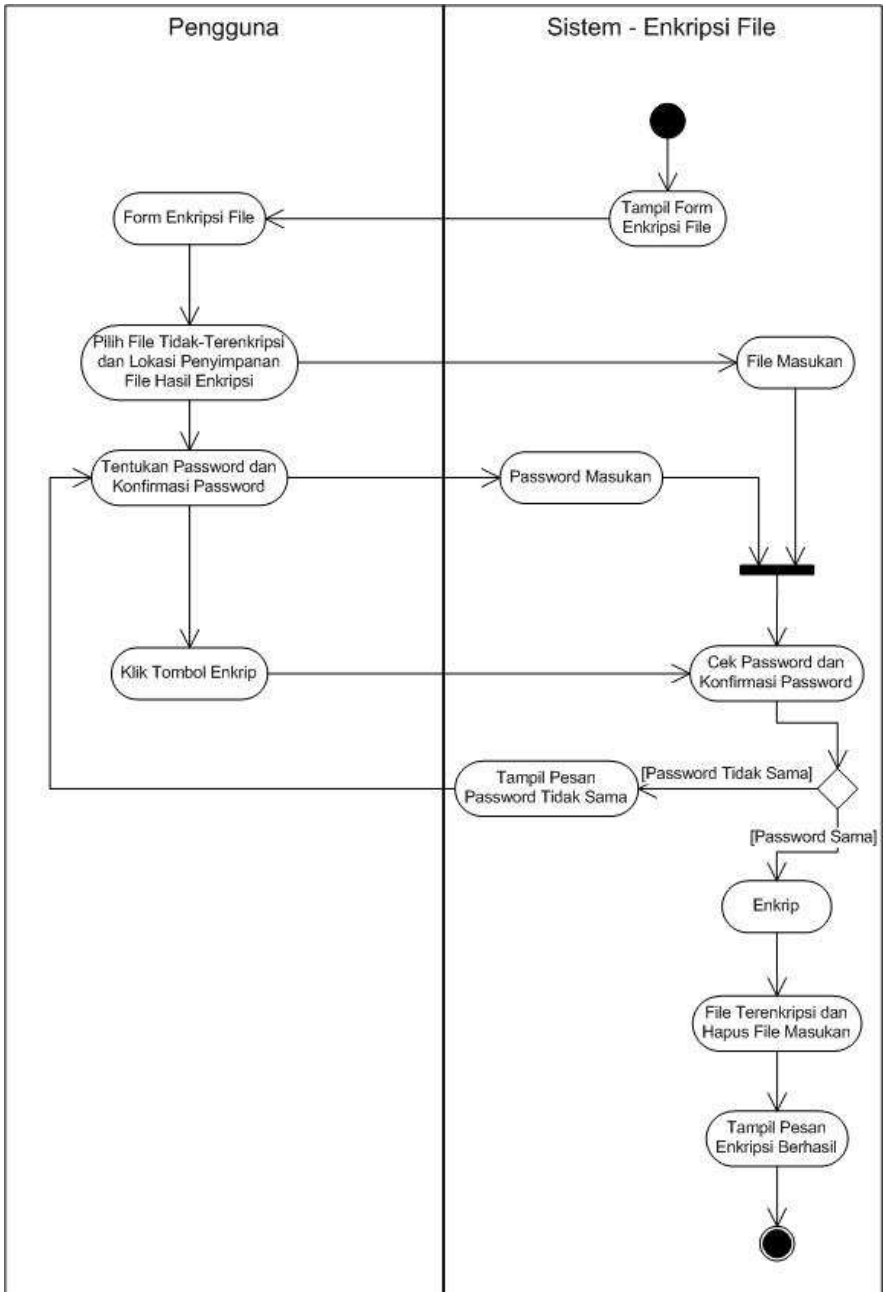
Interaksi antara aktor pengguna dengan *use case* Login dijelaskan dalam *activity diagram* pada Gambar 3.21 sebagai berikut:



Gambar 3.21 *Activity Diagram* Login

2. *Activity Diagram* Enkripsi File

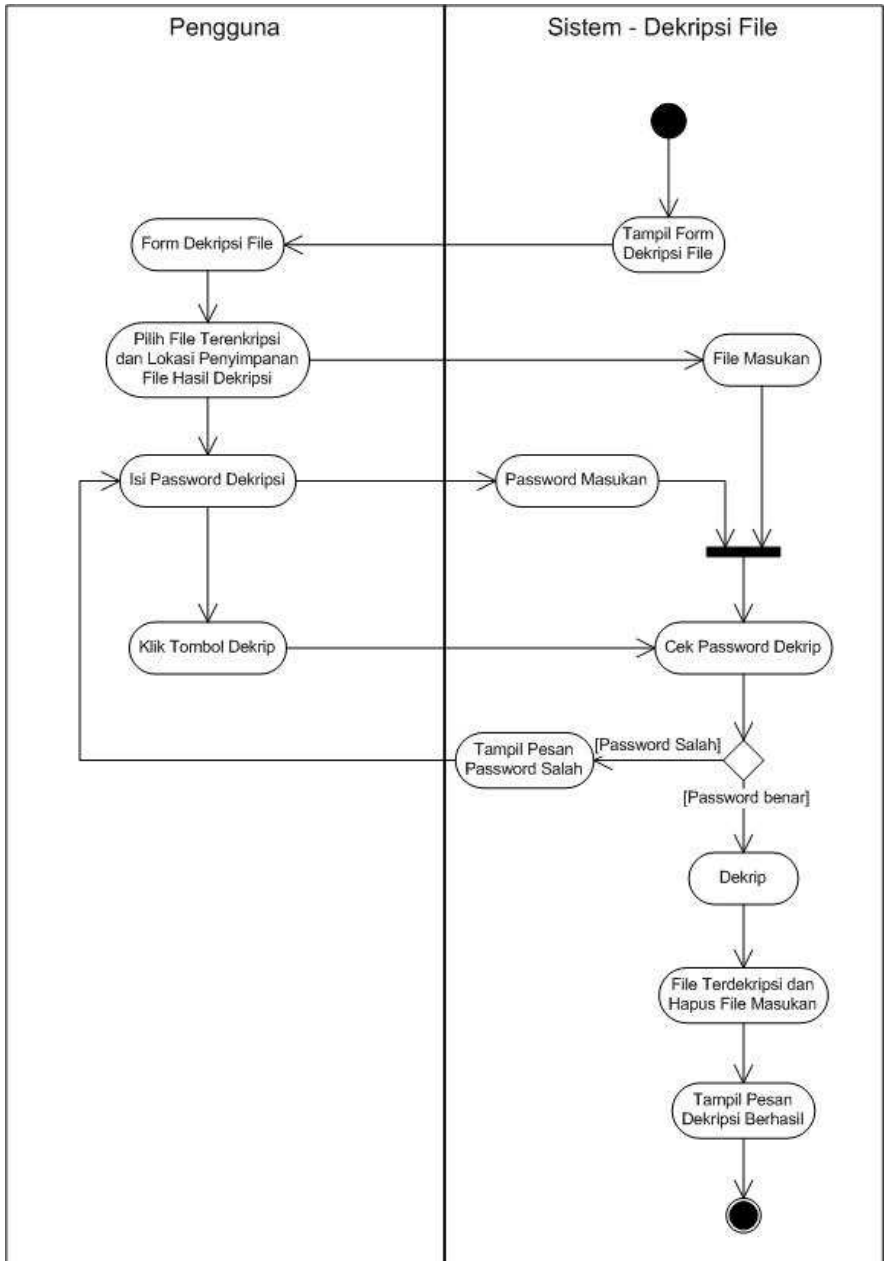
Interaksi antara aktor pengguna dengan *use case* Enkripsi File dijelaskan dalam *activity diagram* pada Gambar 3.22 sebagai berikut:



Gambar 3.22 *Activity Diagram* Enkripsi File

3. *Activity Diagram* Dekripsi File

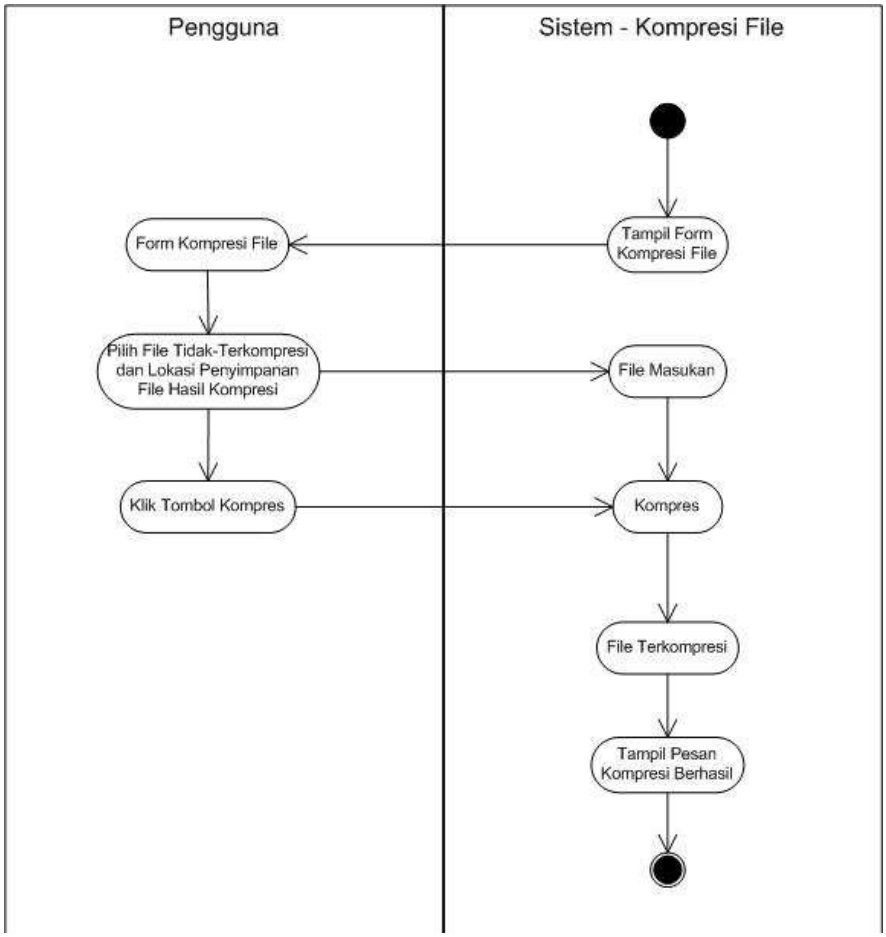
Interaksi antara aktor pengguna dengan *use case* dekripsi file dijelaskan dalam *activity diagram* pada Gambar 3.23 sebagai berikut:



Gambar 3.23 *Activity Diagram* Dekripsi File

4. *Activity Diagram* Kompresi File

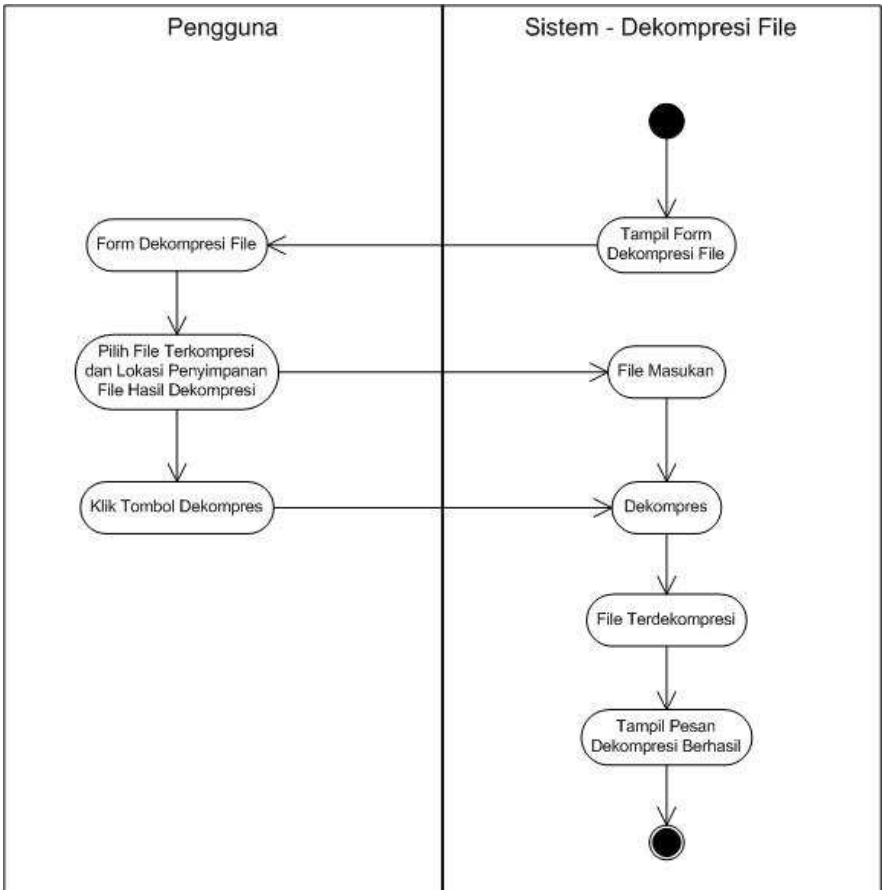
Interaksi antara aktor pengguna dengan *use case* Kompresi File dijelaskan dalam *activity diagram* pada Gambar 3.24 sebagai berikut:



Gambar 3.24 *Activity Diagram* Kompresi File

5. *Activity Diagram* Dekompresi File

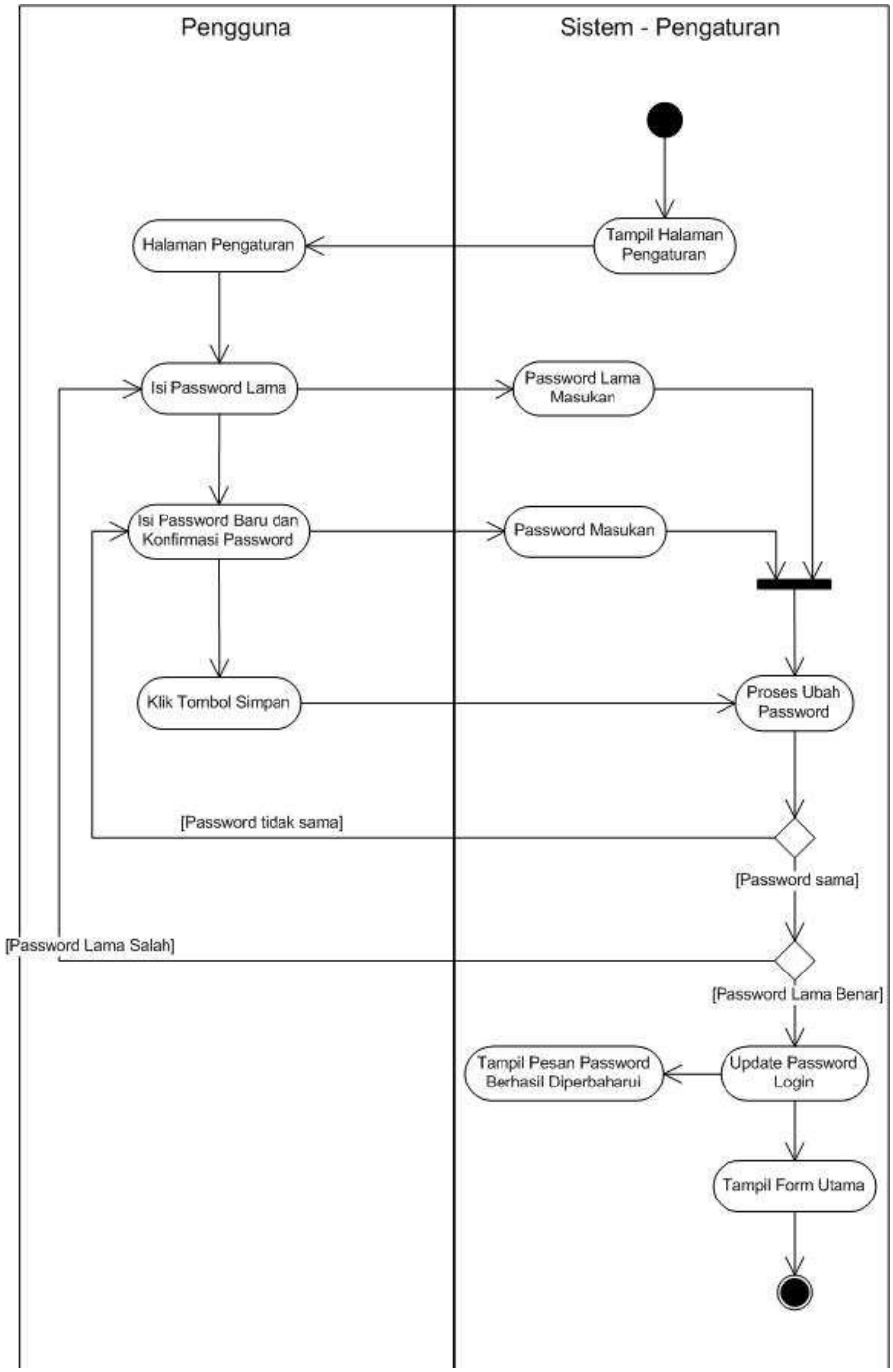
Interaksi antara aktor pengguna dengan *use case* Dekompresi File dijelaskan dalam *activity diagram* pada Gambar 3.25 sebagai berikut:



Gambar 3.25 *Activity Diagram* Dekompresi File

6. *Activity Diagram* Pengaturan

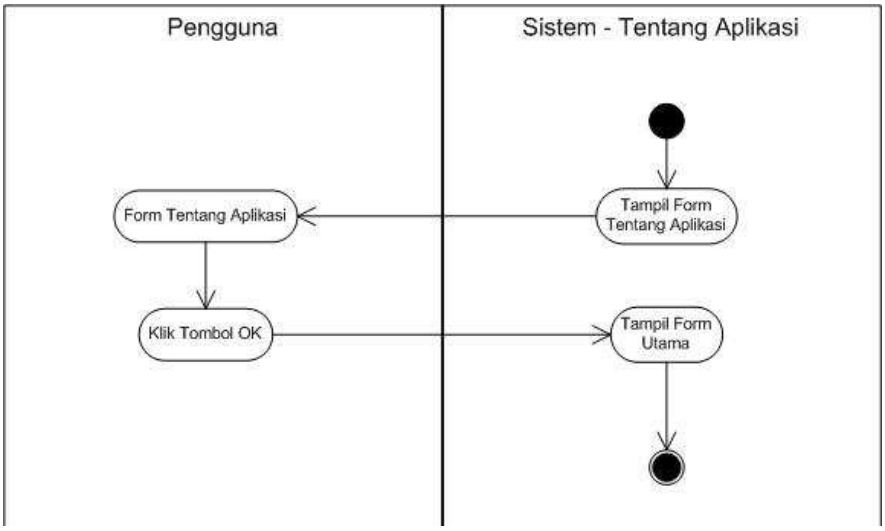
Interaksi antara aktor pengguna dengan *use case* Pengaturan dijelaskan dalam *activity diagram* pada Gambar 3.26 sebagai berikut:



Gambar 3.26 *Activity Diagram* Pengaturan

7. *Activity Diagram* Tentang Aplikasi

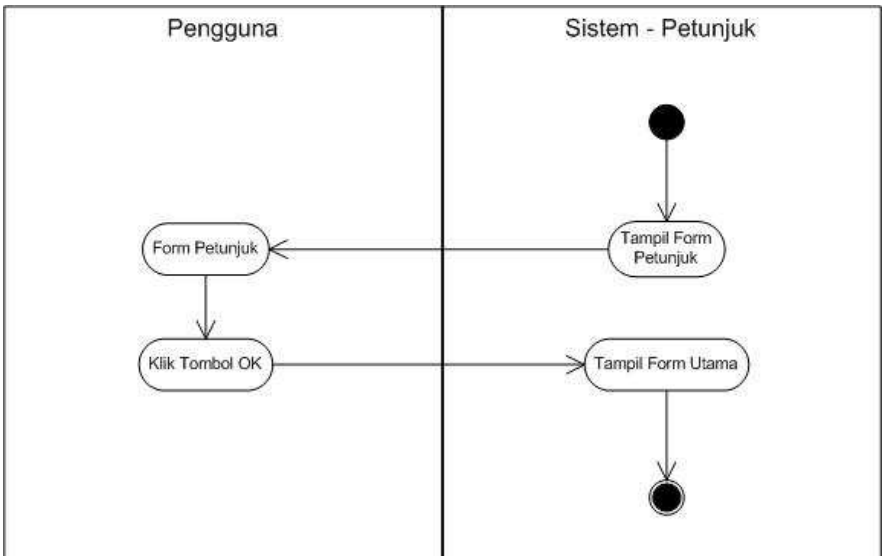
Interaksi antara aktor pengguna dengan *use case* Tentang Aplikasi dijelaskan dalam *activity diagram* pada Gambar 3.27 sebagai berikut:



Gambar 3.27 *Activity Diagram* Tentang Aplikasi

8. *Activity Diagram* Petunjuk

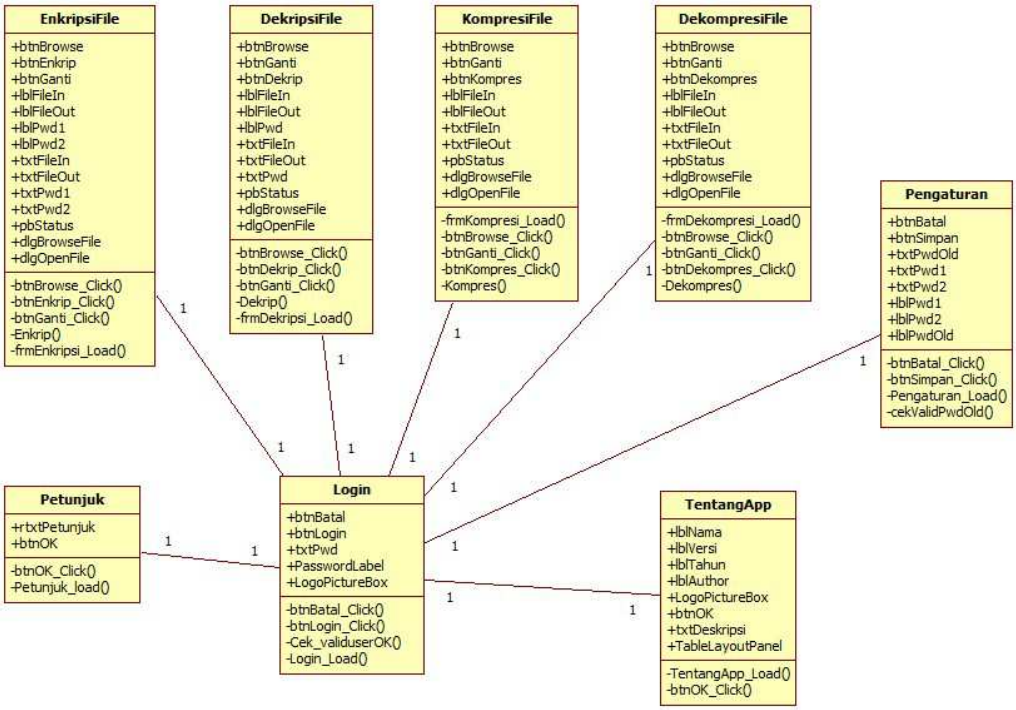
Interaksi antara aktor pengguna dengan *use case* Petunjuk dijelaskan dalam *activity diagram* pada Gambar 3.28 sebagai berikut:



Gambar 3.28. *Activity Diagram* Petunjuk

3.2.6.6 Class Diagram

Class Diagram menggambarkan struktur dan hubungan antar objek-objek yang ada pada sistem. Struktur itu meliputi atribut-atribut dan *method-method* yang ada pada masing-masing kelas, sedangkan hubungnnya meliputi pewarisan asosiasi, generalilasi, dan lain sebagainya. *Class diagram* dalam sistem keamanan data di PT. Prospera Consulting Engineers dapat dilihat pada Gambar 3.29 sebagai berikut:



Gambar 3.29 *Class Diagram*

3.3 Perancangan Sistem

Perancangan sistem merupakan penggambaran, perencanaan, dan pembuatan sketsa atau pengaturan dari beberapa elemen yang terpisah ke dalam suatu kesatuan yang utuh. Tahapan ini meliputi mengkonfigurasi komponen-

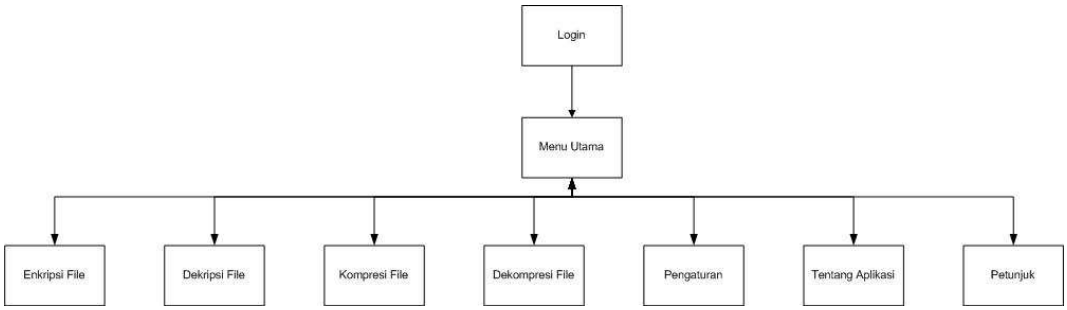
komponen perangkat lunak dan perangkat keras dari suatu sistem. Adapun perancangan sistem dari sistem keamanan data yang dibuat dijelaskan sebagai berikut.

3.3.1 Perancangan Arsitektur

Perancagan arsitektur adalah perancangan yang menggambarkan suatu desain dari alur suatu sistem yang nantinya akan diimplementasikan. Berikut adalah perancangan arsitektur pada sistem keamanan data.

3.3.1.1 Struktur Menu

Pada Gambar 3.30 di bawah ini merupakan struktur menu dari sistem keamanan data:



Gambar 3.30 Struktur Menu

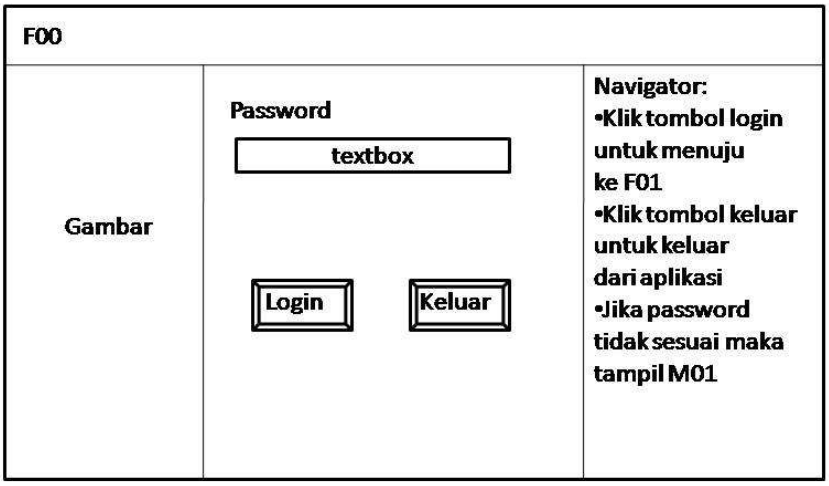
3.3.2 Perancangan Antarmuka

Perancangan antarmuka dibuat untuk memberi konsep pada implementasi pembuatan suatu antarmuka nantinya pada sistem. Berikut adalah gambaran dari perancangan antarmuka di sistem keamanan data:

3.3.2.1 Perancangan *Form*

Perancangan *form* merupakan desain antarmuka dari setiap proses yang terdapat di dalam sistem. Berikut ini adalah perancangan antarmuka setiap proses yang terdiri dari:

1. *Form* Login



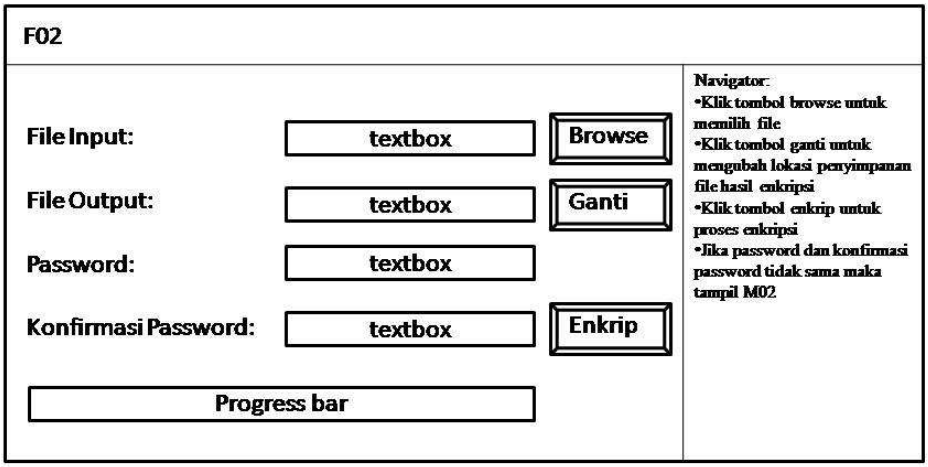
Gambar 3.31 Perancangan Antarmuka *Form* Login

2. *Form* Utama



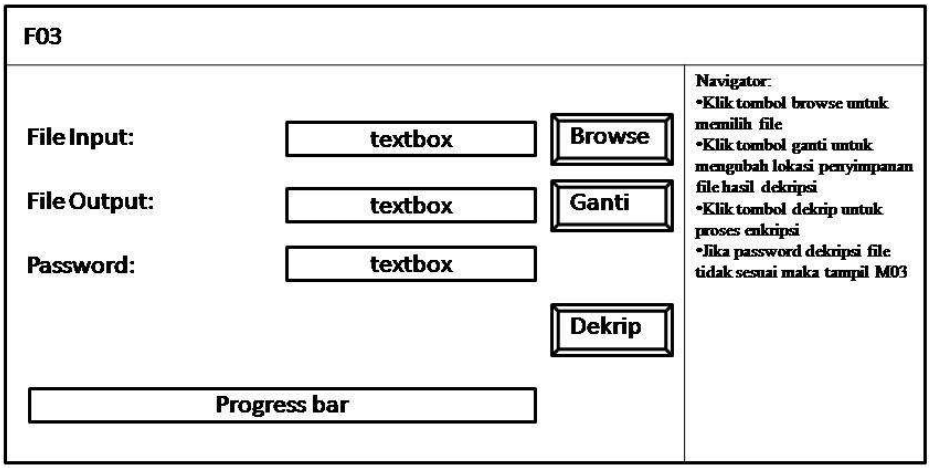
Gambar 3.32 Perancangan Antarmuka *Form* Utama

3. *Form* Enkripsi File



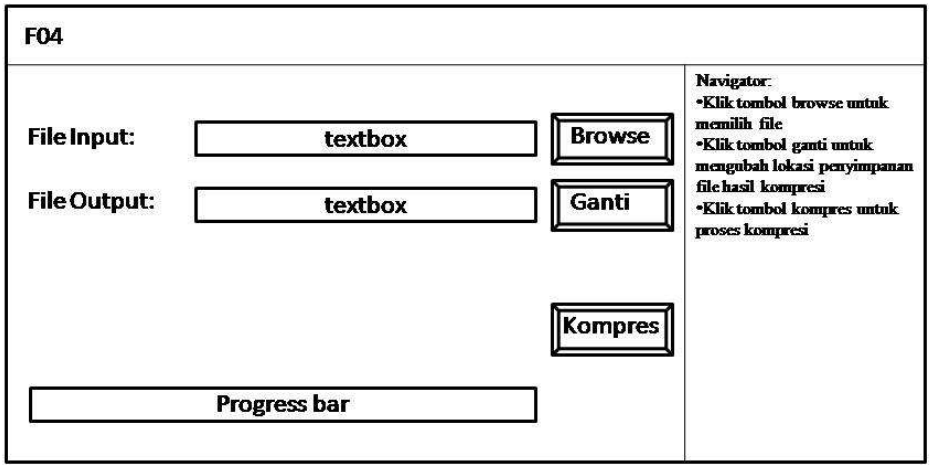
Gambar 3.33 Perancangan Antarmuka *Form* Enkripsi File

4. *Form* Dekripsi File



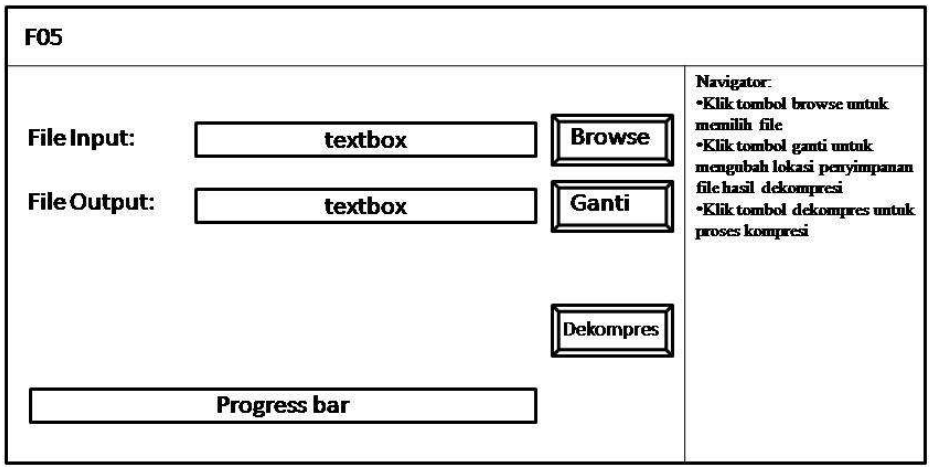
Gambar 3.34 Perancangan Antarmuka *Form* Dekripsi File

5. *Form* Kompresi File



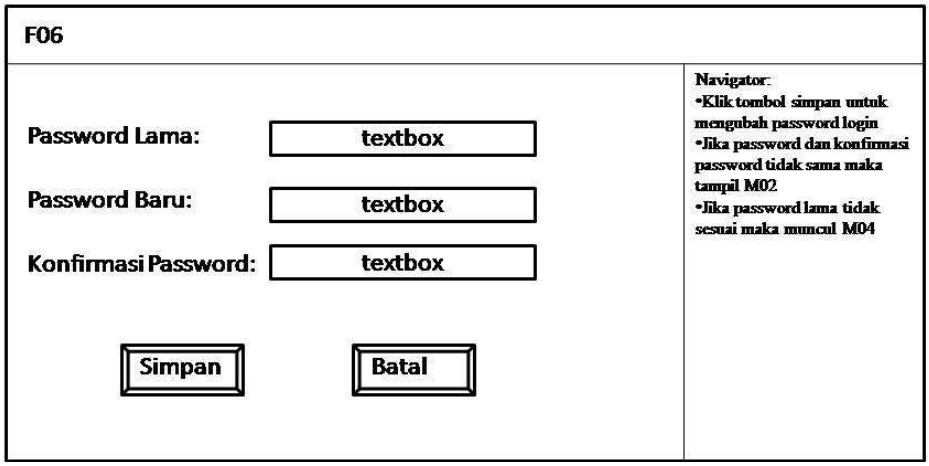
Gambar 3.35 Perancangan Antarmuka *Form* Kompresi File

6. *Form* Dekompresi File



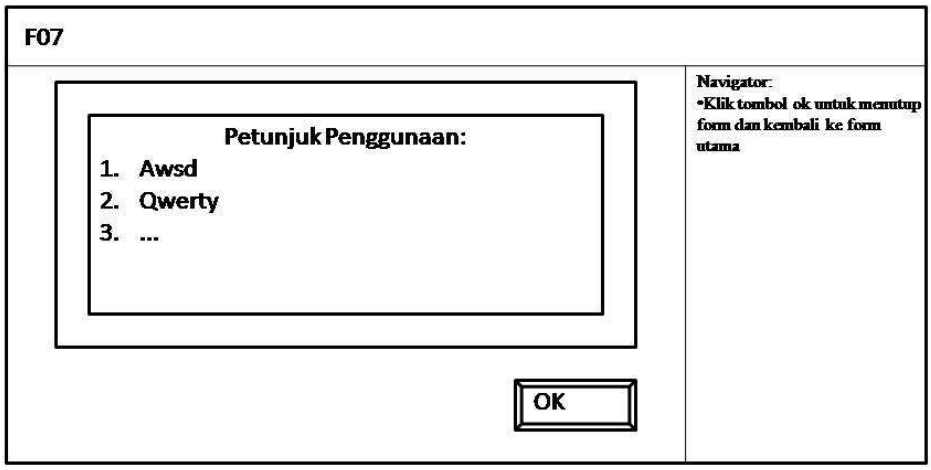
Gambar 3.36 Perancangan Antarmuka *Form* Dekompresi File

7. *Form* Ganti Password



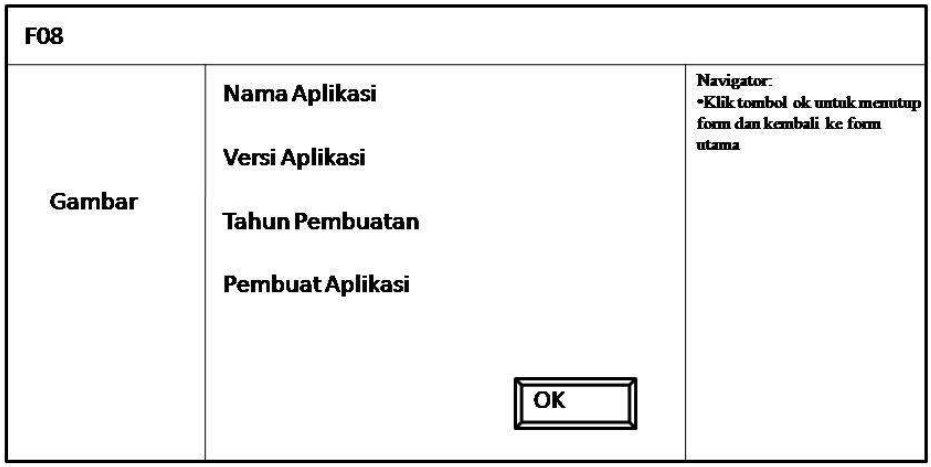
Gambar 3.37 Perancangan Antarmuka *Form* Pengaturan

8. *Form* Petunjuk



Gambar 3.38 Perancangan Antarmuka *Form* Petunjuk

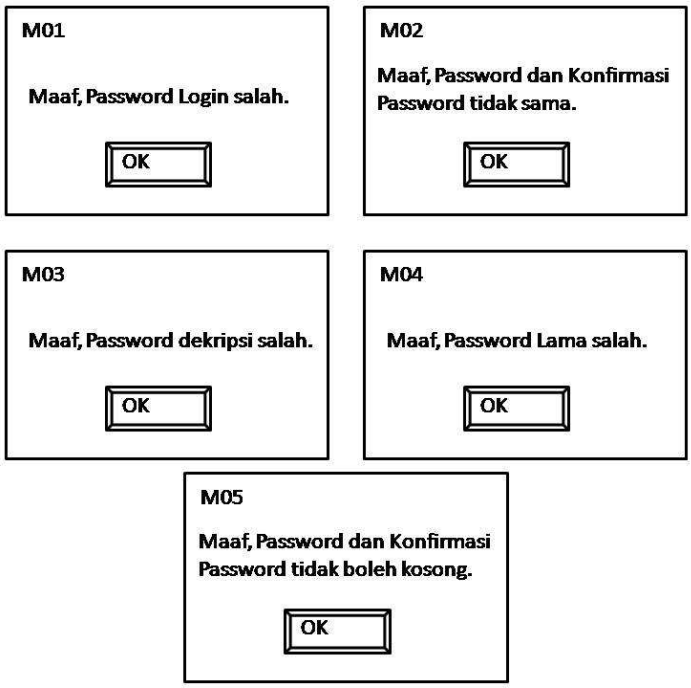
9. *Form* Tentang Aplikasi



Gambar 3.39 Perancangan Antarmuka *Form* Tentang Aplikasi

3.3.2.2 Perancangan Pesan

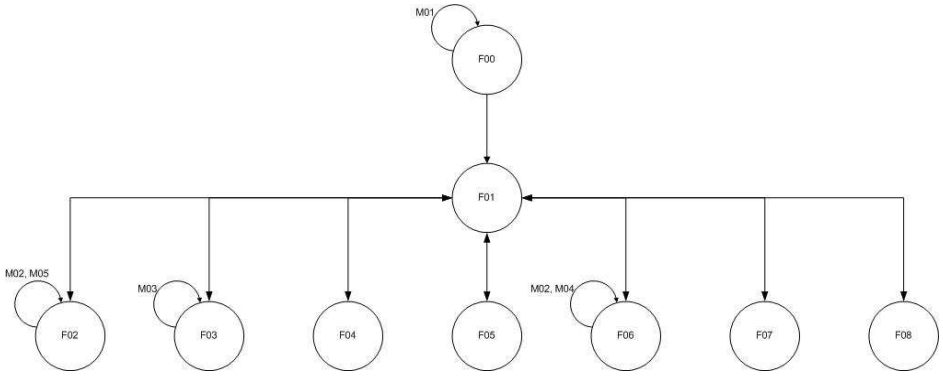
Perancangan pesan merupakan desain antarmuka yang menunjukkan hasil dari suatu proses di dalam sistem. Berikut ini adalah perancangan pesan di dalam sistem keamanan data yang dijelaskan pada Gambar 3.40:



Gambar 3.40 Perancangan Pesan

3.3.2.3 Jaringan Semantik Menu

Jaringan semantik adalah gambaran pengetahuan grafis yang menunjukkan hubungan antar berbagai objek, terdiri dari lingkaran-lingkaran yang dihubungkan dengan anak panah yang menunjukkan objek dan informasi tentang objek-objek tersebut. Berikut ini adalah Gambar 3.41 yang menjelaskan tentang jaringan semantik di dalam sistem keamanan data:



Gambar 3.41 Jaringan Semantik