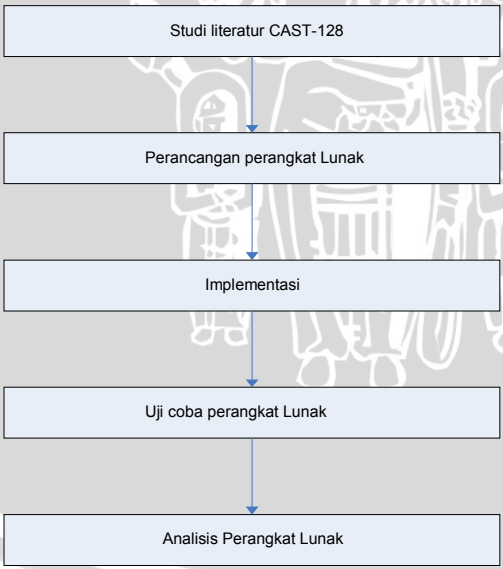
BAB III
METODOLOGI DAN PERANCANGAN

Pada bab metode dan perancangan ini akan dibahas mengenai metode yang digunakan dalam pembuatan dan analisis perangkat lunak enkripsi dan dekripsi dengan algoritma *CAST-128* antara lain:

1. Melakukan studi literatur terhadap algoritma *CAST-128*, dengan memahami secara lebih dalam konsep enkripsi dan dekripsinya.
2. Melakukan perancangan perangkat lunak, meliputi proses input, proses penghitungan *subkey*, proses enkripsi, proses dekripsi dan perancangan *interface*.
3. Implementasi hasil perancangan dengan bahasa pemrograman PHP.
4. Melakukan uji coba enkripsi dan dekripsi *file* dengan perangkat lunak yang telah dibuat.
5. Melakukan beberapa uji coba yang berkaitan dengan waktu proses dan *avalanche effect.*

Langkah-langkah tersebut dapat dilihat seperti pada gambar 3.1.



Gambar 3.1 Diagram alir pembuatan perangkat lunak

3.1 Analisis Perangkat Lunak

Pada subbab analisis ini akan dibahas dekripsi dan batasan untuk perangkat lunak.

3.1.1 Deskripsi Perangkat Lunak

Perangkat lunak enkripsi dan dekripsi *file* teks ini bermanfaat untuk *user* yang ingin mengenkripsi *file* teks yang dimilikinya agar tidak diketahui oleh orang lain. Sistem ini akan menerima *input* berupa *key* dan *file* teks. Kemudian *file* teks tersebut akan dienkripsi menggunakan *key* yang telah diinputkan *user*. Dan hanya bisa didekripsi dengan *key* yang sama.

Algoritma *CAST-128* merupakan algoritma yang beroperasi dalam *digit* biner. Untuk itu setiap *file* teks yang akan dienkripsi maupun didekripsi harus diubah dahulu ke dalam bentuk biner.

Secara umum, proses-proses yang akan dilakukan saat user menggunakan perangkat lunak adalah sebagai berikut :

1. *User* diminta untuk menginputkan *file* teks yang akan dienkripsi.
2. User diminta untuk memasukkan kata kunci (*key)*. Kemudian dilakukan penghitungan *subkey* agar *key* yang di*input*kan *user* tadi siap dipakai untuk enkripsi dan dekripsi seperti yang telah dibahas pada subbab 2.5.3 pada proses pembangkitan kunci internal.
3. Kemudian dari data biner yang didapat dari *file* input akan dilakukan enkripsi dengan menggunakan *subkey*. Proses enkripsi seperti yang telah dijelaskan pada subbab 2.5.4.
4. *Cipher file* bisa didekripsi kembali menggunakan *key* yang sama. Proses dekripsi seperti yang telah dibahas pada subbab 2.5.5. *Flowchart* untuk proses enkripsi dan dekripsi secara umum ini dapat dilihat pada gambar 3.2 dan 3.3.

3.1.2 Batasan Perangkat Lunak

1. *File* yang akan dienkripsi harus berformat *.txt*.
2. Kata kunci menggunakan karakter ASCII.

3.2 Perancangan Perangkat Lunak

Pada subbab ini akan dibahas perancangan berbagai macam perancangan dari perangkat lunak diantaranya adalah : proses *input*, proses penghitungan *subkey,* proses enkripsi dan dekripsi.

3.2.1 Perancangan Proses *Input* Perangkat Lunak

Inputan dari user terdiri dari dua bagian yaitu *inputan* saat melakukan enkripsi *file* dan inputan saat melakukan dekripsi *file*. Dimana inputan saat melakukan enkripsi *file* terdiri dari *input file* teks yang akan dienkripsi dan kunci untuk melakukan enkripsi. Sedangkan *inputan* saat melakukan dekripsi terdiri dari *input file cipher* yang akan di dekripsi dan kunci untuk melakukan dekripsi. *File* teks yang akan di*input*kan *user* berupa *file* yang berektensi *.txt* dan kunci yang digunakan berupa karakter-karakter yang ada pada *keybord* komputer sepanjang 5 sampai 16 karakter. Proses ini pertama kali diawali dengan pembukaan *file* dalam bentuk *binary*, kemudian di baca hingga akhir *file* dan terakhir *file* tersebut ditutup. *Flowchart* untuk proses *input* perangkat lunak ini dapat dilihat pada gambar 3.4.

3.2.2 Perancangan Proses Penghitungan *subkey*

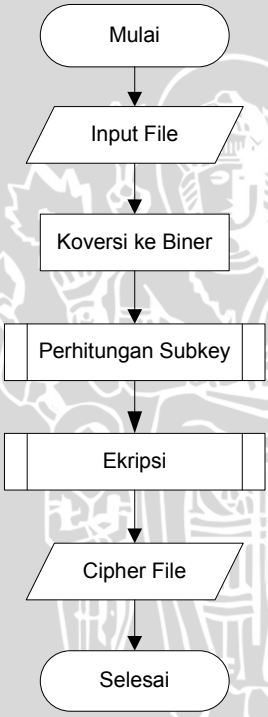
Dalam algortima *CAST-128*, kunci yang dimasukkan user tidak langsung digunakan dalam enkripsi dan dekripsi pesan. Akan tetapi diproses dahulu dengan *S-Box* yang sudah diinisialisasi sebelumnya.

Key yang bisa diterima oleh perangkat lunak ini sepanjang 40-128 bit atau 5-16 karakter ASCII. Apabila *key* melebihi 16 karakter maka karakter yang melebihi 16 itu dianggap tidak ada.

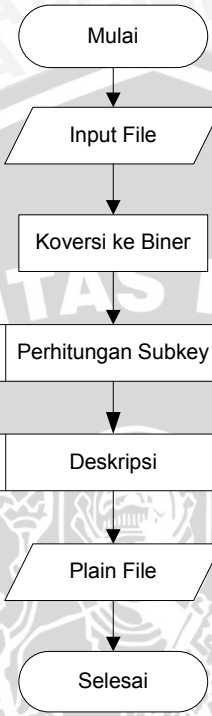
Proses penghitungan *subkey* pada perangkat lunak ini akan seperti berikut :

1. *Inputkey* yang berupa string akan di konversi ke dalam bentuk biner.
2. Perangkat lunak akan menginisialisasi *variabel array* yaitu *array* satu dimensi sebanyak 8 S-box (S1,S2,...,S8) yang sudah ditentukan.
3. *Key* akan dibagi ke dalam blok-blok berisi 8-bit atau 1 karakter dalam *array* satu dimensi X (x0, x1, x2, x3, x4, x5, x6, x7, x8, x9, xA,xB, xC, xD, xE, xF).

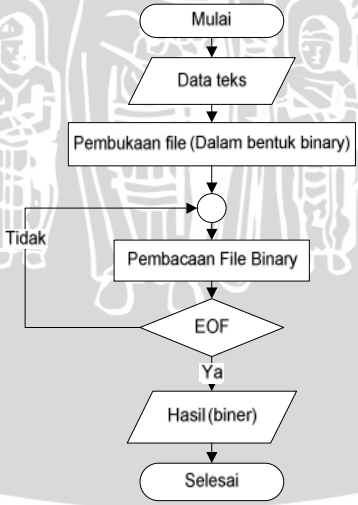
4. Perhitungan *Key Masking* dan *Key* Rotasi menggunakan Algoritma pada sub-bab 2.5.3
5. Pembentukan variabel x0x1x2x3 dengan cara menggabungkan x0 *shift left* 24, x1 *shift left* 16, x2 *shift left* 8, x3. Dengan cara yang sama bentuk x4x5x6x7, x8x9xAxB dan xCxDxExF.
6. Pembentukan variabel z0, z1, z2, z3 dengan z0z1z2z3 *shift right* 24 + binary 11111111, z0z1z2z3 *shift left* 16 + binary 11111111, z0z1z2z3 *shift left* 8 + binary 11111111, z0z1z2z3 + binary 11111111. Dengan cara yang sama bentuk z4, z5, z6, z7, z8, z9, zA, zB,zC, zD, zE dan zF.



Gambar 3.2 *Flowchart* proses enkripsi dalam perangkat lunak



Gambar 3.3 *Flowchart* proses dekripsi dalam perangkat lunak



Gambar 3.4 *Flowchart* proses *inputfile* pada perangkat lunak

3.2.3 Perancangan Proses Enkripsi

Proses enkripsi pada perangkat lunak akan akan seperti berikut:

1. *Plainfile* di-*input*-kan, apabila panjang *plainfile* bukan merupakan kelipatan 8 karakter maka dilakukan penambahan karakter *null* di kanan sampai panjang *plainfile* merupakan kelipatan 8 karakter.
2. *Input plainfile* yang berupa *string* akan di konversi ke dalam bentuk biner.
3. Enkripsi akan dilakukan pada blok yang berisi 64-bit atau 8 karakter.
4. Perangkat lunak akan membagi blok tersebut menjadi dua bagian sama besar yaitu bagian kiri (L) sebesar 32-bit dan bagian kanan (R) sebesar 32-bit. Kemudian L dan R tersebut akan dienkripsi dengan algoritma *CAST-128*. Langkah ke lima sampai delapan akan memperlihatkan proses enkripsi L dan R dengan tiga jenis fungsi *CAST*.
5. Dilakukan 16 iterasi apabila panjang *key* lebih dari 10 karakter atau 80-*byte* dan 12 iterasi apabila panjang *key* kurang atau sama dengan 10 karakter atau 80-*byte*. Dalam setiap iterasi hanya menerima 32-bit masukan. Pada iterasi ganjil menerima masukan bagian kanan (R) dan hasil dari fungsi *CAST* akan di XOR kan dengan bagian kiri (L). Dan sebaliknya untuk iterasi genap. *Flowchart* proses enkripsi perblok dapat dilihat pada gambar 3.7.
6. Penggunaan kuci *masking* dan kunci rotasi berutan sesuai urutan iterasi.
7. Didalam iterasi tersebut ada 3 jenis fungsi *CAST*, iterasi ke-1, 4, 7, 10, 13 dan 16 menggunakan fungsi tipe 1, iterasi ke-2, 5, 8, 11 dan 14 menggunakan fungsi tipe 2, iterasi ke-3, 6, 9 12 dan 15 menggunakan fungsi tipe 3. Secara garis besar fungsi CAST dibagi menjadi 4 operasi. Operasi a akan memproses 32bit *half* data dan dibagi menjadi 4 bagian. Masing-masing 8-bit dijadikan masukan untuk *S-box* 1, *S-box* 2, *S-box* 3, dan *S-box* 4. Operasi b menggabung *S-box* 1 dan *S-box* 2. Operasi c menggabung hasil operasi b dengan *S-box* 3. Dan operasi d menggabung hasil operasi c dengan *S-box* 4. *Flowchart* fungsi F ini dapat dilihat pada gambar 3.8.
8. Fungsi *CAST* tipe 1.

a. Operasi a menambahkan kunci *masking* dengan bit-bit masukan dan menggeser ke kiri sebanya kuci rotasi.
b. Operasi b adalah XOR.
c. Operasi c adalah pengurangan.
d. Operasi d adalah penjumlahan.

9. Fungsi *CAST* tipe 2.
   a. Operasi a meng XOR kan kunci *masking* dengan bit-bit masukan dan menggeser ke kiri sebanya kuci rotasi.
   b. Operasi b adalah pengurangan.
   c. Operasi c adalah penjumlahan.
   d. Operasi d adalah XOR.
10. Fungsi *CAST* tipe 3.
   a. Operasi a mengurangkan kunci *masking* dengan bit-bit masukan dan menggeser ke kiri sebanya kuci rotasi.
   b. Operasi b adalah penjumlahan.
   c. Operasi c adalah XOR.
   d. Operasi d adalah pengurangan.
11. L dan R digabungkan kembali.
12. Konversi hasil penggabungan L dan R ke dalam string sehingga menghasilkan *cipherfile*. *Flowchart* untuk proses enkripsi ini dapat dilihat pada gambar 3.6.

3.2.4 Perancangan Proses Dekripsi

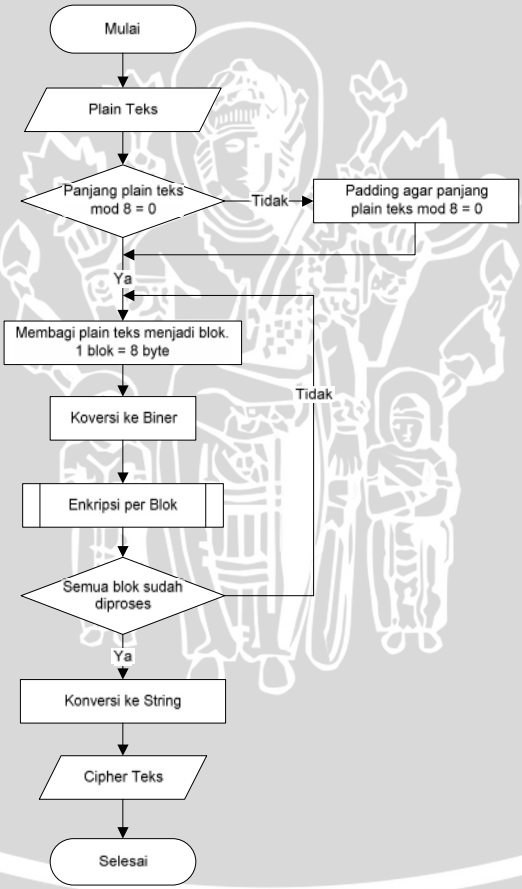
Proses dekripsi pada perangkat lunak akan akan seperti berikut:

1. *Cipherfile* di-*input*-kan, apabila panjang *cipherfile* bukan merupakan kelipatan 8 karakter maka dilakukan penambahan karakter *null* di kanan sampai panjang *cipherfile* merupakan kelipatan 8 karakter.
2. *Input cipherfile* yang berupa string akan di konversi ke dalam bentuk biner.
3. Dekripsi akan dilakukan pada blok yang berisi 64-bit atau 8 karakter.
4. Perangkat lunak akan membagi blok tersebut menjadi dua bagian sama besar yaitu bagian kiri (L) sebesar 32-bit dan bagian kanan (R) sebesar 32-bit. Kemudian L dan R tersebut

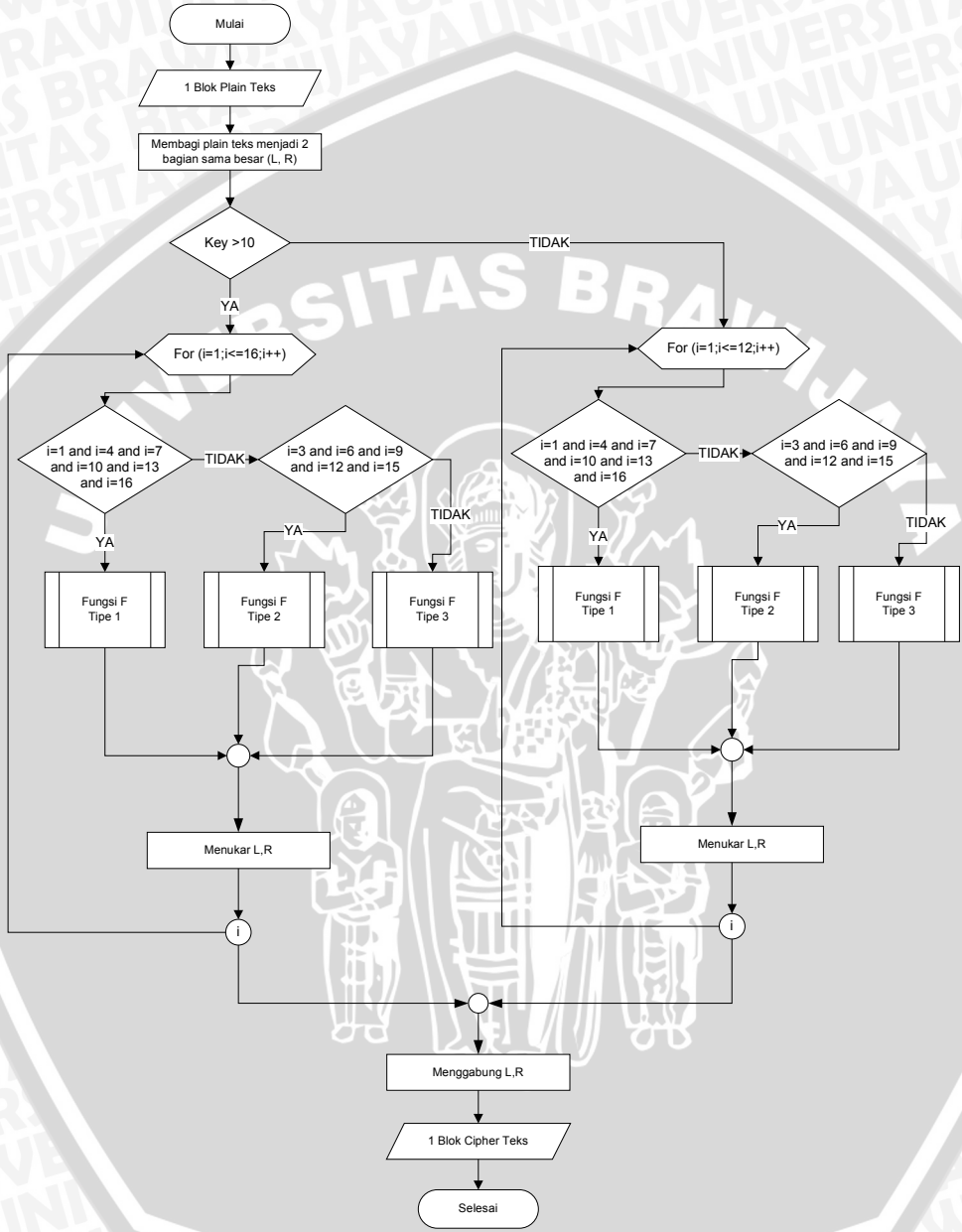
akan dienkripsi dengan algoritma *CAST-128*. Langkah ke lima sampai delapan akan memperlihatkan proses enkripsi XL dan XR dengan tiga jenis fungsi *CAST*.

5. Dilakukan 16 iterasi apabila panjang *key* lebih dari 10 karakter atau 80-*byte* dan 12 iterasi apabila panjang *key* kurang atau sama dengan 10 karakter atau 80-*byte*. Dalam setiap iterasi hanya menerima 32-bit masukan. Pada iterasi ganjil menerima masukan bagian kiri (L) dan hasil dari fungsi *CAST* akan di XOR kan dengan bagian kanan (R). Dan sebaliknya untuk iterasi genap. *Flowchart* proses dekripsi perblok dapat dilihat pada gambar 3.12.
6. Penggunaan kuci masking dan kunci rotasi dimulai dari kunci paling akhir beurut mundur.
7. Di dalam iterasi tersebut ada 3 jenis fungsi *CAST*, iterasi ke-1, 4, 7, 10, 13 dan 16 menggunakan fungsi tipe 1, iterasi ke-2, 5, 8, 11 dan 14 menggunakan fungsi tipe 2, iterasi ke-3, 6, 9 12 dan 15 menggunakan fungsi tipe 3. Secara garis besar fungsi *CAST* dibagi menjadi 4 operasi. Operasi a akan memproses 32bit *half* data dan dibagi menjadi 4 bagian. Masing-masing 8-bit dijadikan masukan untuk *S-box* 1, *S-box* 2, *S-box* 3, dan *S-box* 4. Operasi b menggabung *S-box* 1 dan *S-box* 2. Operasi c menggabung hasil operasi b dengan *S-box* 3. Dan operasi d menggabung hasil operasi c *dengan S-box* 4. *Flowchart* fungsi ini dapat dilihat pada gambar 3.8, gambar 3.9 dan gambar 3.10.
8. Fungsi *CAST* tipe 1.
    a. Operasi a menambahkan kunci *masking* dengan bit-bit masukan dan menggeser ke kiri sebanya kuci rotasi.
    b. Operasi b adalah XOR.
    c. Operasi c adalah pengurangan.
    d. Operasi d adalah penjumlahan.
9. Fungsi *CAST* tipe 2.
    a. Operasi a meng XOR kan kunci *masking* dengan bit-bit masukan dan menggeser ke kiri sebanya kuci rotasi.
    b. Operasi b adalah pengurangan.
    c. Operasi c adalah penjumlahan.
    d. Operasi d adalah XOR.

10. Fungsi *CAST* tipe 3.
    a. Operasi a mengurangkan kunci masking dengan bit-bit masukan dan menggeser ke kiri sebanya kuci rotasi.
    b. Operasi b adalah penjumlahan.
    c. Operasi c adalah XOR.
    d. Operasi d adalah pengurangan.
11. L dan R digabungkan kembali.
12. Konversi hasil penggabungan L dan R ke dalam string sehingga menghasilkan *plainfile. Flowchart* untuk proses dekripsi ini dapat dilihat pada gambar 3.11.

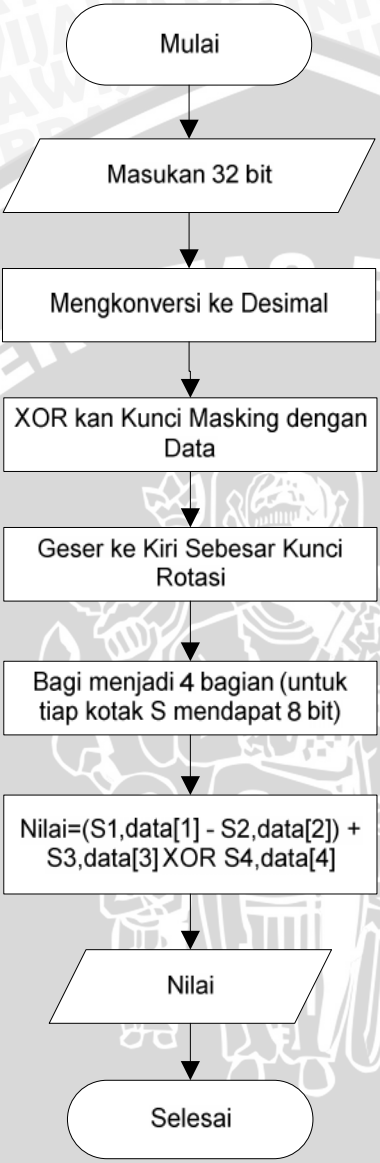


Gambar 3.5 *Flowchart* proses enkripsi

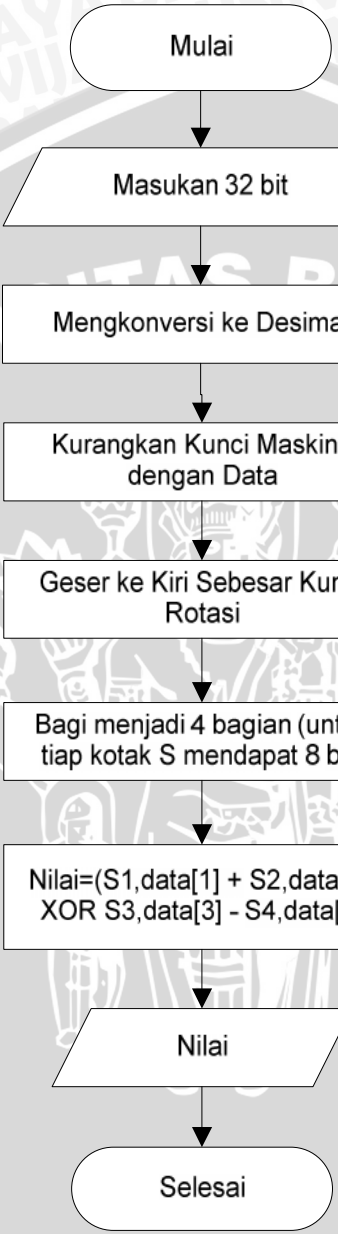


Gambar 3.6 *Flowchart* Enkripsi PerBlok

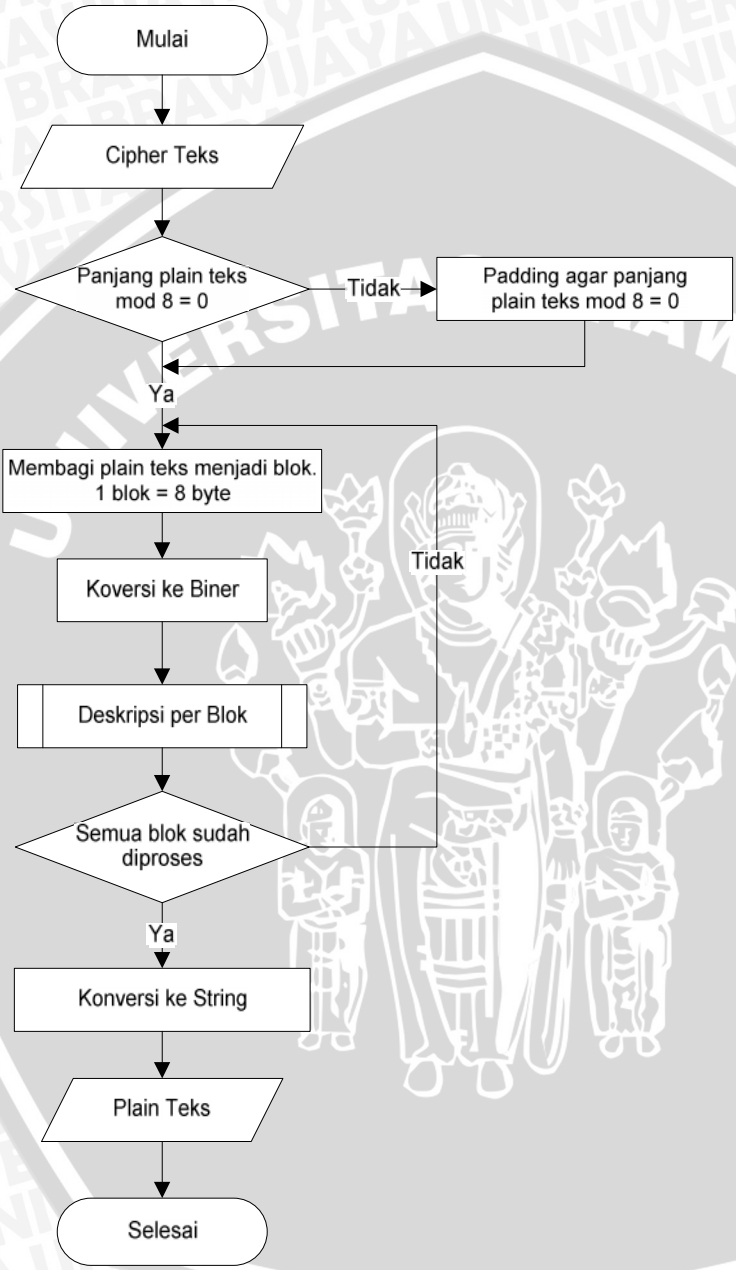
Gambar 3.7 *Flowchart* Fungsi F Tipe 1



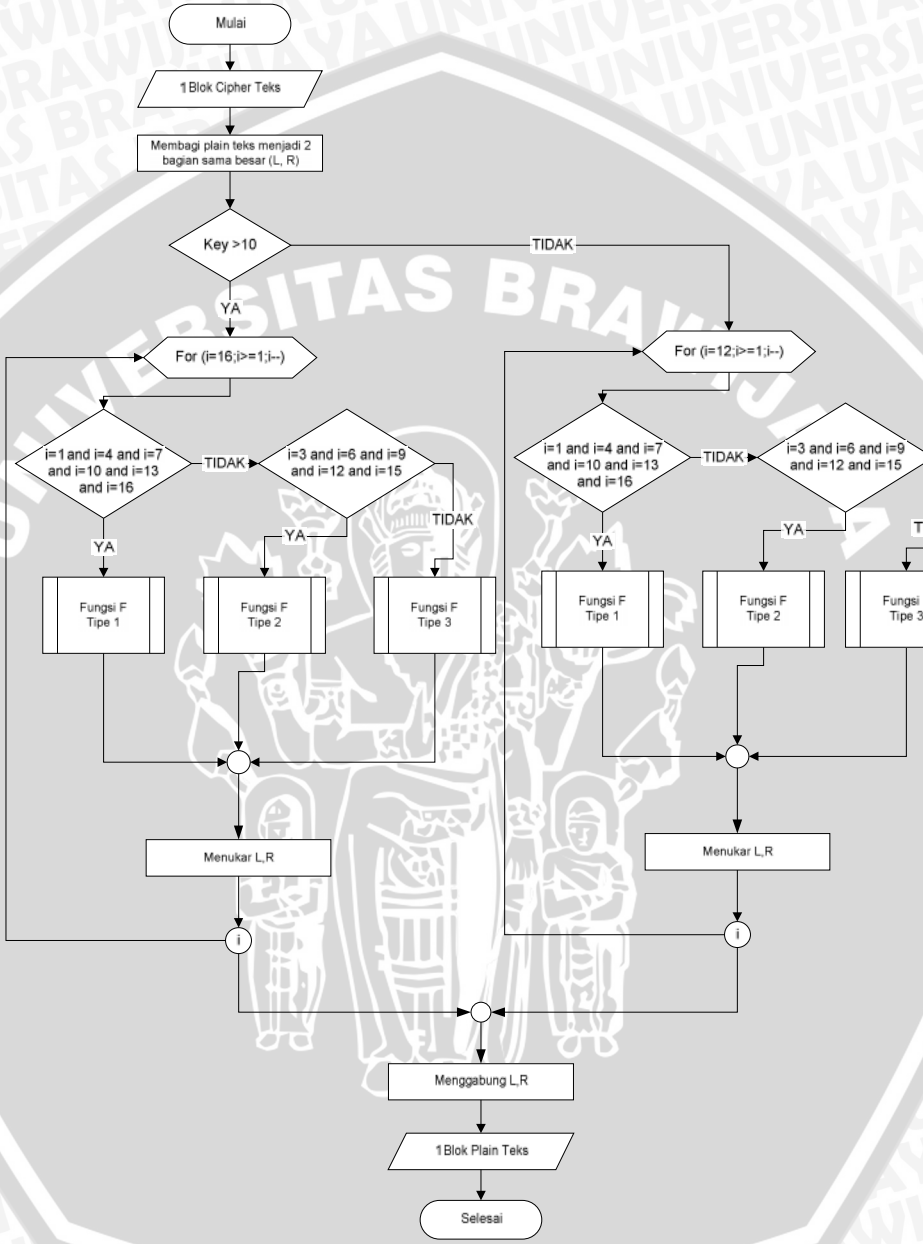
Gambar 3.8 *Flowchart* Fungsi F Tipe 2



Gambar 3.9 *Flowchart* Fungsi F Tipe 3



Gambar 3.10 *Flowchart* Proses dekripsi



Gambar 3.11 *Flowchart* Dekripsi PerBlok

3.3 Perhitungan Matematis

3.3.1 Perhitungan *Subkey*

Dalam perhitungan *subkey* pada algoritma *CAST-128* ini akan diperlukan Kunci Eksternal sepanjang 128 bit. Kemudian dibagi menjadi 16 *bytes* subkunci, yaitu: x0x1x2x3x4x5x6x7x8x9xAxBxC xDxExF, Dimana x0 menunjukkan *most significant byte*(MSB) dan xF menunjukkan *least significant byte*(LSB). Kemudian perhitungannya menggunakan algortima seperti pada subbab 2.5.3.

Kemudian sebagai contoh penghitungan *subkey* akan digunakan *key* "CAST-128". Langkah-langkah penghitungan untuk mendapatkan *subkey* adalah sebagai berikut :

1. *Key* "CAST-128" terdiri dari 8 karakter atau 64-bit. Karena kurang dari 16 karakter atau 128-bit maka perlu dilakukan *padding* menjadi "CAST-12800000000".
2. Ubah *Key* ke dalam bentuk biner. Dapat dilihat pada tabel 3.1. hasil konversi *key* ke bentuk biner adalah sebagai berikut :
   01000011010000010101001101010100001011010011000100110010001111000000000000000000000000000000000000000000000000000000000000000000000000

Tabel 3.1 Konversi *key* ke biner

string	heksadesimal	biner
C	43	01000011
A	41	01000001
S	53	01010011
T	54	01010100
-	2d	00101101
1	31	00110001
2	32	00110010
8	38	00111000
0	0	00000000
0	0	00000000
0	0	00000000
0	0	00000000
0	0	00000000
0	0	00000000
0	0	00000000
0	0	00000000

3. Perhitungan Keym(kunci *masking*) dan Keyr(Kunci Rotasi) :
   x0x1x2x3 = 01000011010000010101001101010100
   x4x5x6x7 = 00101101001100010011001000111000
   x8x9xAxB = 00000000000000000000000000000000
   xCxDxExF = 00000000000000000000000000000000

   z0z1z2z3=x0x1x2x3 XOR s5[xD] XOR s6[xF] XOR s7[xC] XOR s8[xE] XOR s7[x8]
   x0x1x2x3 = 01000011010000010101001101010100
   s5[xD] = 01111110110010010000110000000100
   s6[xF] = 11110110111110101000111110011101
   s7[xC] = 10000101111000000100000000011001
   s8[xE] = 11100010000101100011000000001101
   s7[x8] = 10000101111000000100000000011001 □
   z0z1z2z3 = 00101001011001001110000011000000

   z4z5z6z7 = x8x9xAxB XOR s5[z0] XOR s6[z2] XOR s7[z1] XOR s8[z3] XOR s8[xA]
   x8x9xAxB = 00000000000000000000000000000000
   s5[z0] = 01101000101011111000000001000000
   s6[z2] = 01100101001111010111111001101010
   s7[z1] = 11110111110111101011101110000101
   s8[z3] = 01110011000011101101111010111100
   s8[xA] = 11100010000101100011000000001101 □
   z4z5z6z7 = 01101011010101001010101100011110

   z8z9zAzB = xCxDxExF XOR s5[z7] XOR s6[z6] XOR s7[z5] XOR s8[z4] XOR s5[x9]
   xCxDxExF = 00000000000000000000000000000000
   s5[z7] = 10001101101110100001110011111110
   s6[z6] = 01100000011000101110001110010111
   s7[z5] = 10000010000111011011101010011111
   s8[z4] = 00000110011101101010001110101011
   s5[x9] = 01111110110010010000110000000100 □
   z8z9zAzB = 00010111011110101110101001011001

   zCzDzEzF = x4x5x6x7 XOR s5[zA] XOR s6[z9] XOR s7[zB] XOR s8[z8] XOR s6[xB]
   x4x5x6x7 = 00101101001100010011001000111000
   s5[zA] = 01101011101011000011000001111111

s6[z9] = 00110011001100111011000010010100
s7[zB] = 00000000100110110100111111000011
s8[z8] = 00000100100111101110110111111101
s6[xB] = 11110110111110101000111110011101 □
zCzDzEzF = 10000111010100011001111101110000

keym[1] = s5[z8] XOR s6[z9] XOR s7[z7] XOR s8[z6] XOR s5[z2]
s5[z8] = 01001100101010101101111011111111
s6[z9] = 00110011001100111011000010010100
s7[z7] = 00010010100001101011111011001111
s8[z6] = 00100010001101100001001110111101
s5[z2] = 01101100111101101110010001111001 □
keym[1] = 00100011110111110010011101100000

keym[2] = s5[zA] XOR s6[zB] XOR s7[z5] XOR s8[z4] XOR s5[z6]
s5[zA] = 01101011101011000011000001111111
s6[zB] = 11100111011011111111101111100111
s7[z5] = 10000010000111011011101010011111
s8[z4] = 00000110011101101010001110101011
s6[z6] = 01100000011000101110001110010111 □
keym[2] = 01101000110010100011000100111011

keym[3] = s5[zC] XOR s6[zD] XOR s7[z3] XOR s8[z2] XOR s7[z9]
s5[zC] = 10110000110101110000111010111010
s6[zD] = 01010011110000001000010000111010
s7[z3] = 10011000100000111111111001100110
s8[z2] = 10101010000100101110010011110010
s7[z9] = 11110100001100001100100001111101 □
keym[3] = 00100101101101100101100001101001

keym[4] = s5[zE] XOR s6[zF] XOR s7[z1] XOR s8[z0] XOR s8[zC]
s5[zE] = 01001001100100011111100001000000
s6[zF] = 01001110110001110101101110010101
s7[z1] = 11110111110111101011101110000101
s8[z0] = 11111101101010100011001101011101
s8[zC] = 00010101000101101000001011101011 □

keym[4] = 00011000001101001010100111100110

x0x1x2x3 = z8z9zAzB XOR s5[z5] XOR s6[z7] XOR s7[z4] XOR s8[z6] XOR s7[z0]
z8z9zAzB = 00010111011110101110101001011001
s5[z5] = 00001101000000011110100110000000
s6[z7] = 10101000100010000110000101001010
s7[z4] = 01010001111100001100100000000010
s8[z6] = 00100010001101100001001110111101
s7[z0] = 10110011101100101110100111001110 □
x0x1x2x3 = 01110010100001110101000011100010

x4x5x6x7 = z0z1z2z3 XOR s5[x0] XOR s6[x2] XOR s7[x1] XOR s8[x3] XOR s8[z2]
z0z1z2z3 = 00101001011001001110000011000000
s5[x0] = 11110110010101001110111111000101
s6[x2] = 11001110101100100010100101101111
s7[x1] = 01011101110110100000000000110011
s8[x3] = 11100000111101101010001001111010
s8[z2] = 10101010000100101110010011110010 □
x4x5x6x7 = 00000110101111000110000011010001

x8x9xAxB = z4z5z6z7 XOR s5[x7] XOR s6[x6] XOR s7[x5] XOR s8[x4] XOR s5[z1]
z4z5z6z7 = 01101011010101001010101100011110
s5[x7] = 00111111010010000001110110000111
s6[x6] = 10001101111001001011111110011001
s7[x5] = 00010111110111001011000011110000
s8[x4] = 00001110001001000001011000000000
s5[z1] = 01100011011001110011011110110110 □
x8x9xAxB = 10100011011001111001100001000110

xCxDxExF = zCzDzEzF XOR s5[xA] XOR s6[x9] XOR s7[xB] XOR s8[x8] XOR s6[z3]
zCzDzEzF = 10000111010100011001111101110000
s5[xA] = 01100110101101001111000010100011
s6[x9] = 10111000011110000011010010111111
s7[xB] = 11001111000110011101111101011000
s8[x8] = 11100101100000001011001111100110
s6[z3] = 00001000101010010011000011110110 □

xCxDxExF = 01111011101011010000011100100100

keym[5] = s5[x3] XOR s6[x2] XOR s7[xC] XOR s8[xD] XOR s5[x8]
s5[x3] = 11010000110011101111101001100101
s6[x2] = 11001110101100100010100101101111
s7[xC] = 11001000101001110001001100000010
s8[xD] = 00110111110111111001001100101011
s5[x8] = 01000100010010001001010000000110 □
keym[5] = 10100101010011001100011100100101

keym[6] = s5[x1] XOR s6[x0] XOR s7[xE] XOR s8[xF] XOR s6[xD]
s5[x1] = 10110000110101110000111010111010
s6[x0] = 01000010110100010101110110011001
s7[xE] = 00100000001010001101101000011111
s8[xF] = 10110011000000011101010000001010
s6[xD] = 10110110110010000101001010000011 □
keym[6] = 11010111111001110000111110110101

keym[7] = s5[x7] XOR s6[x6] XOR s7[x8] XOR s8[x9] XOR s7[x3]
s5[x7] = 00111111010010000001110110000111
s6[x6] = 10001101111001001011111110011001
s7[x8] = 00010101111000001101011111111001
s8[x9] = 10000101100111000001010110100101
s7[x3] = 00111000101000001100000001111101 □
keym[7] = 00011010011100001010000000111111

keym[8] = s5[x5] XOR s6[x4] XOR s7[xA] XOR s8[xB] XOR s8[x7]
s5[x5] = 10110011110011011100111101110010
s6[x4] = 11101100111011010101110010111100
s7[xA] = 00101000111001110100111001000001
s8[xB] = 01011000110010110111111000000111
s8[x7] = 00000000110110100110110101110111 □
keym[8] = 00101111110101101100111011111111

z0z1z2z3=x0x1x2x3 XOR s5[xD] XOR s6[xF] XOR s7[xC] XOR s8[xE] XOR s7[x8]

x0x1x2x3 = 01110010100001110101000011100010
s5[xD] = 01111101000101100001101110111010
s6[xF] = 11010000110101010001100100110010
s7[xC] = 11001000101001110001001100000010
s8[xE] = 00000101001011001110100010110101
s7[x8] = 00010101111000001101011111111001 □
z0z1z2z3 = 00000111001011110111111000100100

z4z5z6z7 = x8x9xAxB XOR s5[z0] XOR s6[z2] XOR s7[z1] XOR s8[z3] XOR s8[xA]
x8x9xAxB = 10100011011001111001100001000110
s5[z0] = 00010111001100010001011001111111
s6[z2] = 11101001101010011101100001001000
s7[z1] = 11010011010011010111010100010110
s8[z3] = 10110011000000011101010000001010
s8[xA] = 11001101011111101101011100000101 0 □
z4z5z6z7 = 11110000110011100101100101100111

z8z9zAzB = xCxDxExF XOR s5[z7] XOR s6[z6] XOR s7[z5] XOR s8[z4] XOR s5[x9]
xCxDxExF = 01111011101011010000011100100100
s5[z7] = 10001110110010100011011011000001
s6[z6] = 11100111011011111111011111100111
s7[z5] = 11000011111101011110000110010100
s8[z4] = 11101001011101100010010110100101
s5[x9] = 10001110110010100011011011000001 □
z8z9zAzB = 10110110010000010011100011110010

zCzDzEzF = x4x5x6x7 XOR s5[zA] XOR s6[z9] XOR s7[zB] XOR s8[z8] XOR s6[xB]
x4x5x6x7 = 00000110101111000110000011010001
s5[zA] = 11011111000100111010001010000000
s6[z9] = 10101010100100101000001000100011
s7[zB] = 11010001100110001000111100110101
s8[z8] = 10110100100010100010010001100101
s6[xB] = 11101111111010001100001101101110 □
zCzDzEzF = 11111001110001110010100001001100

keym[9] = s5[z3] XOR s6[z2] XOR s7[zC] XOR s8[zD] XOR s5[z9]

s5[z3] = 11000001000001101110110011010111
s6[z2] = 11101001101010011101100001001000
s7[zC] = 10000100101100011101001101110000
s8[zD] = 11111001110000010111101010001111
s5[z9] = 11111110001110001101001110011001 □
keym[9] = 10101011111001110100111011111001

keym[10] = s5[z1] XOR s6[z0] XOR s7[zE] XOR s8[zF] XOR s6[zC]
s5[z1] = 01001110001010011111111010100111
s6[z0] = 00110010010101010101001110101100
s7[zE] = 00010000011101111000100110111110
s8[zF] = 00111100111100011101001011100010
s6[zC] = 01001001110010010010111101010100 □
keym[10] = 00011001001100111101100100000011

keym[11] = s5[z7] XOR s6[z6] XOR s7[z8] XOR s8[z9] XOR s7[z2]
s5[z7] = 10001110110010100011011011000001
s6[z6] = 11100111011011111111101111100111
s7[z8] = 00000101001111100101011000011010
s8[z9] = 00101001100110001101111100000100
s7[z2] = 10111110100010111001110100101101 □
keym[11] = 11111011100010001101100100010101

keym[12] = s5[z5] XOR s6[z4] XOR s7[zA] XOR s8[zB] XOR s8[z6]
s5[z5] = 01111000010011010110101100010111
s6[z4] = 00111011010011001011111110011111
s7[zA] = 01000111101111000010100000101001
s8[zB] = 00001110001001010010010001001011
s8[z6] = 00101101010111100011001101010100 □
keym[12] = 00100111110001101110101110111110

x0x1x2x3 = z8z9zAzB XOR s5[z5] XOR s6[z7] XOR s7[z4] XOR s8[z6] XOR s7[z0]
z8z9zAzB = 10110110010000010011100011110010
s5[z5] = 01111000010011010110101100010111
s6[z7] = 10111000011110000011010010111111
s7[z4] = 10010001110110100101010111110100

s8[z6] = 00101101010111100011001101010100
s7[z0] = 00100000001010001101101000011111 □
x0x1x2x3 = 11101010110110001101101111100101

x4x5x6x7 = z0z1z2z3 XOR s5[x0] XOR s6[x2] XOR s7[x1] XOR s8[x3] XOR s8[z2]
z0z1z2z3 = 00000111001011110111111000100100
s5[x0] = 01101011101011000011000001111111
s6[x2] = 01111111100101111100010110101011
s7[x1] = 11010011101101011010101100110100
s8[x3] = 00001101011101110001110000101011
s8[z2] = 01111100110100010110111011111100 □
x4x5x6x7 = 10110001000001110101001000010011

x8x9xAxB = z4z5z6z7 XOR s5[x7] XOR s6[x6] XOR s7[x5] XOR s8[x4] XOR s5[z1]
z4z5z6z7 = 11110000110011100101100101100111
s5[x7] = 11001100111101011100000110000000
s6[x6] = 11111110100010010011011001010101
s7[x5] = 00100000001010001101101000011111
s8[x4] = 11101111001101000111100011011101
s5[z1] = 01001110001010011111111010100111 □
x8x9xAxB = 01000011100001111111001011010111

xCxDxExF = zCzDzEzF XOR s5[xA] XOR s6[x9] XOR s7[xB] XOR s8[x8] XOR s6[z3]
zCzDzEzF = 11111001110001110010100001001100
s5[xA] = 01110101010101010100010000101100
s6[x9] = 01011001001010101111100101010000
s7[xB] = 11000110000111100100010110111110
s8[x8] = 10011011011011011111010010010001
s6[z3] = 11010000110101010001100100110010 □
xCxDxExF = 01011000000111100011110100101101

keym[13] = s5[x8] XOR s6[x9] XOR s7[x7] XOR s8[x6] XOR s5[x2]
s5[x8] = 00000110001001000000011111101010
s6[x9] = 01011001001010101111100101010000
s7[x7] = 11100010010011101111000110111101
s8[x6] = 11001001101011111111011001111011

s5[x3] = 00001111111101101111100011110011 □
keym[13] = 01111011000110010000000110001111

keym[14] = s5[xA] XOR s6[xB] XOR s7[x5] XOR s8[x4] XOR s5[x6]
s5[xA] = 01110101010101010100010000101100
s6[xB] = 01010100111101001010000010000100
s7[x5] = 00100000001010001101101000011111
s8[x4] = 11101111001101000111100011011101
s6[x7] = 11100100011000100101111001111110 □
keym[14] = 00001010110111110001100000010100

keym[15] = s5[xC] XOR s6[xD] XOR s7[x3] XOR s8[x2] XOR s7[x8]
s5[xC] = 10111100111100111111000010101010
s6[xD] = 10101000100010000110000101001010
s7[x3] = 01011000100111011101001110010000
s8[x2] = 11000111111010111000111100110111
s7[x8] = 01110010111111000000100000011010 □
keym[15] = 11111001111100011100010101011101

keym[16] = s5[xE] XOR s6[xF] XOR s7[x1] XOR s8[x0] XOR s8[xD]
s5[xE] = 11001000101011011110110110110011
s6[xF] = 01001100011111110100010001001000
s7[x1] = 11010011101101011010101100110100
s8[x0] = 00110110100110010111101100000111
s8[xD] = 01110010110111110001100100011011 □
keym[16] = 00010011001000010110000011010011

z0z1z2z3=x0x1x2x3 XOR s5[xD] XOR s6[xF] XOR s7[xC] XOR s8[xE] XOR s7[x8]
x0x1x2x3 = 11101010110110001101101111100101
s5[xD] = 10001101101110100001110011111110
s6[xF] = 01001100011111110100010001001000
s7[xC] = 10010010010101000100101010001011
s8[xE] = 10101110011000111010111111110010
s7[x8] = 01110010111111000000100000011010 □
z0z1z2z3 = 01100101110101100110111000110000

z4z5z6z7 = x8x9xAxB XOR s5[z0] XOR s6[z2] XOR s7[z1] XOR s8[z3] XOR s8[xA]
x8x9xAxB = 01000011100001111111001011010111
s5[z0] = 01010000111101011011011000010110
s6[z2] = 11100010001000100000101010111110
s7[z1] = 00111101000111111100111001101111
s8[z3] = 10000010111100111101000001010101
s8[xA] = 00001110001001010010010001001011 □
z4z5z6z7 = 01000000100110010111010000001110

z8z9zAzB = xCxDxExF XOR s5[z7] XOR s6[z6] XOR s7[z5] XOR s8[z4] XOR s5[x9]
xCxDxExF = 01011000000111100011110100101101
s5[z7] = 10101100111101100010010000111010
s6[z6] = 01111011011011100010011111111111
s7[z5] = 11000010011000010000101011001010
s8[z4] = 10111011110100110101000001001001
s5[x9] = 10110000110101110000111010111010 □
z8z9zAzB = 01000110111000110110101011010001

zCzDzEzF = x4x5x6x7 XOR s5[zA] XOR s6[z9] XOR s7[zB] XOR s8[z8] XOR s6[xB]
x4x5x6x7 = 10110001000001110101001000010011
s5[zA] = 11111011100010000111101000110111
s6[z9] = 01010000000101111101010101011011
s7[zB] = 00111100100110010111111001111110
s8[z8] = 01011000110010110111111000000111
s6[xB] = 01010100111101001010000010000100 □
zCzDzEzF = 00101010001111100101110110000010

keyr[1] = 0x1F && (s5[z8] XOR s6[z9] XOR s7[z7] XOR s8[z6] XOR s5[z2])
s5[z8] = 10010000110001111001010100000101
s6[z9] = 01010000000101111101010101011011
s7[z7] = 01001011001101101001010111110010
s8[z6] = 10110000111111100001001101001100
s5[z2] = 10110110111101011000100111011110 □
0x1F = 00000000000000000000000000011111&&
keyr[1] = 00000000000000000000000000011110

keyr[2] = s5[zA] XOR s6[zB] XOR s7[z5] XOR s8[z4] XOR s5[z6]
s5[zA] = 11111011100010000111101000110111
s6[zB] = 00111000000101001111001000000000
s7[z5] = 11000010011000010000101011001010
s8[z4] = 10111011110100110101000001001001
s6[z6] = 01111011011011100010011111111111 ▫
0x1F = 00000000000000000000000000011111&&
keyr[2] = 00000000000000000000000000001011

keyr[3] = s5[zC] XOR s6[zD] XOR s7[z3] XOR s8[z2] XOR s7[z9]
s5[zC] = 11101101000011001001111001010110
s6[zD] = 00001000101000011001100001100110
s7[z3] = 01001110011110110011101011111111
s8[z2] = 00100100001001011001111111010111
s7[z9] = 11111101000101100000011011110010 ▫
0x1F = 00000000000000000000000000011111&&
keyr[3] = 00000000000000000000000000001010

keyr[4] = s5[zE] XOR s6[zF] XOR s7[z1] XOR s8[z0] XOR s8[zC]
s5[zE] = 01100100111011100010110101111110
s6[zF] = 11110011101001011111011001110110
s7[z1] = 00111101000111111100111001101111
s8[z0] = 00111110001101111000000101100000
s8[zC] = 00010111011011110100001111101000 ▫
0x1F = 00000000000000000000000000011111&&
keyr[4] = 00000000000000000000000000001111

x0x1x2x3 = z8z9zAzB XOR s5[z5] XOR s6[z7] XOR s7[z4] XOR s8[z6] XOR s7[z0]
z8z9zAzB = 01000110111000110110101011010001
s5[z5] = 11000000111100010110010010001010
s6[z7] = 11110011100011111111011100110010
s7[z4] = 00001010100101100001001010001000
s8[z6] = 10110000111111100001001101001100
s7[z0] = 01100001111111100000001100111100 ▫
x0x1x2x3 = 10101110000010111111101110010001

x4x5x6x7 = z0z1z2z3 XOR s5[x0] XOR s6[x2] XOR s7[x1] XOR s8[x3] XOR s8[z2]
z0z1z2z3 = 01100101110101100110111000110000
s5[x0] = 10011100101011011001000000010000
s6[x2] = 01110001001010001010010001010100
s7[x1] = 01010000111100011000101110000010
s8[x3] = 10011111001100100110010001000010
s8[z2] = 00100100001001011001111111010111 ▫
x4x5x6x7 = 01100011101101010010101001100011

x8x9xAxB = z4z5z6z7 XOR s5[x7] XOR s6[x6] XOR s7[x5] XOR s8[x4] XOR s5[z1]
z4z5z6z7 = 01000000100110010111010000001110
s5[x7] = 00000100101001011100001010000100
s6[x6] = 11101000011010111110001111011010
s7[x5] = 00011001110111100111111010101110
s8[x4] = 01000110101001010010010101100100
s5[z1] = 01011111010001101011000000101010 ▫
x8x9xAxB = 10101100011010101011111010110000

xCxDxExF = zCzDzEzF XOR s5[xA] XOR s6[x9] XOR s7[xB] XOR s8[x8] XOR s6[z3]
zCzDzEzF = 00101010001111100101110110000010
s5[xA] = 00100000100100110110000001111001
s6[x9] = 11001001110001001100100000111011
s7[xB] = 11010110100110010010100101101110
s8[x8] = 11011101000001101100101010100010
s6[z3] = 00001000001110010001100110100111 ▫
xCxDxExF = 11000000110011110000111110101011

keyr[5] = s5[x3] XOR s6[x2] XOR s7[xC] XOR s8[xD] XOR s5[x8]
s5[x3] = 10010100111101110100101111000000
s6[x2] = 01110001001010001010010001010100
s7[xC] = 10011000100000111111111001100110
s8[xD] = 00011010000000000111001001101110
s5[x8] = 00111100101001011101011100010111 ▫
0x1F = 00000000000000000000000000011111&&
keyr[5] = 00000000000000000000000000001011

keyr[6] = s5[x1] XOR s6[x0] XOR s7[xE] XOR s8[xF] XOR s6[xD]
s5[x1] = 01110011010110101011101000000000
s6[x0] = 00111100110000101010110011111011
s7[xE] = 10110010100001110000011111011110
s8[xF] = 00100010001101100001001110111101
s6[xD] = 00110110000101000000000010111100 □
0x1F = 00000000000000000000000000011111&&
keyr[6] = 00000000000000000000000000000100

keyr[7] = s5[x7] XOR s6[x6] XOR s7[x8] XOR s8[x9] XOR s7[x3]
s5[x7] = 00000100101001011100001010000100
s6[x6] = 11101000011010111110001111011010
s7[x8] = 10010011110100101001101000100010
s8[x9] = 11011011000001111011101000001100
s7[x3] = 01000011100000000101000011100011 □
0x1F = 00000000000000000000000000011111&&
keyr[7] = 00000000000000000000000000010011

keyr[8] = s5[x5] XOR s6[x4] XOR s7[xA] XOR s8[xB] XOR s8[x7]
s5[x5] = 00010111011011010100100001101111
s6[x4] = 11011010010110100010011011000000
s7[xA] = 10011110101000101001010011111011
s8[xB] = 01010111000101011111011010110111
s8[x7] = 01000110101001010010010101100100 □
0x1F = 00000000000000000000000000011111&&
keyr[8] = 00000000000000000000000000000111

z0z1z2z3=x0x1x2x3 XOR s5[xD] XOR s6[xF] XOR s7[xC] XOR s8[xE] XOR s7[x8]
x0x1x2x3 = 10101110000010111111101110010001
s5[xD] = 01011000111010111011000101101110
s6[xF] = 01100000011000101110001110010111
s7[xC] = 10011000100000111111111001100110
s8[xE] = 00110111110111011101110111111100
s7[x8] = 10010011110100101001101000100010 □
z0z1z2z3 = 10101010000011100001000011010000

z4z5z6z7 = x8x9xAxB XOR s5[z0] XOR s6[z2] XOR s7[z1] XOR s8[z3] XOR s8[xA]
x8x9xAxB = 10101100011010101011111010110000
s5[z0] = 10110011000110110010101111100001
s6[z2] = 00110011111100010100100101100001
s7[z1] = 01001011001101101001010111110010
s8[z3] = 00010001010000000011000010010010
s8[xA] = 00111000110101111110010110110010 □
z4z5z6z7 = 01001110001000011001110011100010

z8z9zAzB = xCxDxExF XOR s5[z7] XOR s6[z6] XOR s7[z5] XOR s8[z4] XOR s5[x9]
xCxDxExF = 11000000110011110000111110101011
s5[z7] = 11010000110011101111101001100101
s6[z6] = 00001110111100111100100010100110
s7[z5] = 01110101011001011011110111100100
s8[z4] = 01000010010011110111011000011000
s5[x9] = 11111011100010000111101000110111 □
z8z9zAzB = 11010010010100001000110010100011

zCzDzEzF = x4x5x6x7 XOR s5[zA] XOR s6[z9] XOR s7[zB] XOR s8[z8] XOR s6[xB]
x4x5x6x7 = 01100011101101010010101001100011
s5[zA] = 10010100110010100000101101010110
s6[z9] = 11001110101100100010100101101111
s7[zB] = 00010101111000001101011111111001
s8[z8] = 01001010000011001101110101100001
s6[xB] = 01001110100011110000001001010010 □
zCzDzEzF = 00101000101011100000000010010000

keyr[9] = s5[z3] XOR s6[z2] XOR s7[zC] XOR s8[zD] XOR s5[z9]
s5[z3] = 01000100000010010100111110000101
s6[z2] = 00110011111100010100100101100001
s7[zC] = 00010000011101111000100110111110
s8[zD] = 11000100001001001000001010001001
s5[z9] = 10010001000111100111001110011010 □
0x1F = 00000000000000000000000000011111&&
keyr[9] = 00000000000000000000000000001001

keyr[10] = s5[z1] XOR s6[z0] XOR s7[zE] XOR s8[zF] XOR s6[zC]
s5[z1] = 10101100111101100010010000111010
s6[z0] = 10001000110010011000111110001000
s7[zE] = 10000101111000000100000000011001
s8[zF] = 10110110111100101100111100111011
s6[zC] = 11111101010000010001100101111110 □
0x1F = 00000000000000000000000000011111&&
keyr[10] = 00000000000000000000000000001110

keyr[11] = s5[z7] XOR s6[z6] XOR s7[z8] XOR s8[z9] XOR s7[z2]
s5[z7] = 11010000110011101111101001100101
s6[z6] = 00001110111100111100100010100110
s7[z8] = 01011110010011111001010100000100
s8[z9] = 10011101000101111101111011100111
s7[z2] = 10100000101111110111100111101110 □
0x1F = 00000000000000000000000000011111&&
keyr[11] = 00000000000000000000000000010110

keyr[12] = s5[z5] XOR s6[z4] XOR s7[zA] XOR s8[zB] XOR s8[z6]
s5[z5] = 10111010100011110110010111001011
s6[z4] = 10011010011010011010000000101111
s7[zA] = 10100011110110011101001010110000
s8[zB] = 11100101100000001011001111100110
s8[z6] = 11001110101001001101010000101000 □
0x1F = 00000000000000000000000000011111&&
keyr[12] = 00000000000000000000000000011010

x0x1x2x3 = z8z9zAzB XOR s5[z5] XOR s6[z7] XOR s7[z4] XOR s8[z6] XOR s7[z0]
z8z9zAzB = 11010010010100001000110010100011
s5[z5] = 10111010100011110110010111001011
s6[z7] = 01010001101001000111011111101010
s7[z4] = 11000110111001101111101000010100
s8[z6] = 11001110101001001101010000101000
s7[z0] = 11010111000101101110011101000000 □
x0x1x2x3 = 11100110001011110101011111111110

x4x5x6x7 = z0z1z2z3 XOR s5[x0] XOR s6[x2] XOR s7[x1] XOR s8[x3] XOR s8[z2]
z0z1z2z3 = 10101010000011100001000011010000
s5[x0] = 10100000100111000111111101110000
s6[x2] = 10101001101010011001001110000111
s7[x1] = 11010011010011010111010100010110
s8[x3] = 10001101101100101010001010000011
s8[z2] = 11011110100110101101111010110001 □
x4x5x6x7 = 00100011010111101111010100000011

x8x9xAxB = z4z5z6z7 XOR s5[x7] XOR s6[x6] XOR s7[x5] XOR s8[x4] XOR s5[z1]
z4z5z6z7 = 01001110001000011001110011100010
s5[x7] = 10100110001100110111100100010001
s6[x6] = 11110101010001001110110111101011
s7[x5] = 00011010111011000011110010101001
s8[x4] = 10101010010000000010000101100100
s5[z1] = 10101100111101100010010000111010 □
x8x9xAxB = 00000001000011000011000111101111

xCxDxExF = zCzDzEzF XOR s5[xA] XOR s6[x9] XOR s7[xB] XOR s8[x8] XOR s6[z3]
zCzDzEzF = 00101000101011100000000010010000
s5[xA] = 00000010110100011100000000000000
s6[x9] = 00011010101101101010011010111000
s7[xB] = 11110011100000010111100100010100
s8[x8] = 10111011110111011111111111111100
s6[z3] = 11101000100000010110111101001010 □
xCxDxExF = 10010000000101001000111110001010

keyr[13] = s5[x8] XOR s6[x9] XOR s7[x7] XOR s8[x6] XOR s5[x2]
s5[x8] = 00101100011011100111010010111001
s6[x9] = 00011010101101101010011010111000
s7[x7] = 11001111110001100101011010010011
s8[x6] = 00001101001000000101100111010001
s5[x3] = 01101101010001111101111000001000 □
0x1F = 00000000000000000000000000011111&&
keyr[13] = 00000000000000000000000000001011

keyr[14] = s5[xA] XOR s6[xB] XOR s7[x5] XOR s8[x4] XOR s5[x6]
s5[xA] = 00000010110100011100000000000000
s6[xB] = 10011010101101101111011011110101
s7[x5] = 00011010111011000011110010101001
s8[x4] = 10101010010000000010000101100100
s6[x7] = 11100010001100110111111101111100 ▫
0x1F = 00000000000000000000000000011111&&
keyr[14] = 00000000000000000000000000000100

keyr[15] = s5[xC] XOR s6[xD] XOR s7[x3] XOR s8[x2] XOR s7[x8]
s5[xC] = 01011000000010100010010010011111
s6[xD] = 10100011000010001110101010011001
s7[x3] = 10010000011100010110111101001011
s8[x2] = 00000110110011010001101011111000
s7[x8] = 00110011001010111111010101100111 ▫
0x1F = 00000000000000000000000000011111&&
keyr[15] = 00000000000000000000000000010010

keyr[16] = s5[xE] XOR s6[xF] XOR s7[x1] XOR s8[x0] XOR s8[xD]
s5[xE] = 01100001100001001011010110111110
s6[xF] = 01111101101001001100111011000000
s7[x1] = 11010011010011010111010100010110
s8[x0] = 01100111110011011011000101010110
s8[xD] = 10111011001001000011101000001111 ▫
0x1F = 00000000000000000000000000011111&&
keyr[16] = 00000000000000000000000000010001

3.3.2 Perhitungan Enkripsi

Setelah melakukan perhitungan *subkey* langkah selanjutnya adalah melakukan enkripsi pada sebuah *string*. Contoh *string* yang akan dienkripsi adalah ”KOMPUTER”. Karena *string* ini terdiri dari 64-bit maka tidak dilakukan *padding*.

Langkah-langkah perhitungan matematisnya adalah sebagai berikut :

1. Konversi *string* diatas kedalam bentuk biner. Hasil konversi biner *string* ”KOMPUTER” yaitu :

0100101101001111010011010101000001010101010100010
0010101010010.

Proses konversi dimulai dari konversi string ke heksadesimal kemudian heksadesimal ke biner. Proses tersebut bisa dilihat pada tabel 3.2.

Tabel 3.2 konversi plaintext kedalam biner

STRING	HEKSADESIMAL	BINER
K	4b	01001011
O	4f	01001111
M	4d	01001101
P	50	01010000
U	55	01010101
T	54	01010100
E	45	01000101
R	52	01010010

2. Bagi kedalam 2 blok masing-masing 32-bit.
L=01001011010011110100110101010000
R=01010101010101000100010101010010

3. Lakukan enkripsi *CAST-128*.

- Iterasike-1 menggunakan Fungsi *CAST* tipe 1

Tambahkan R dengan Kunci *Masking* ke-1
I = keym[1] + R
keym[1]=00100011110111110010011101100000
R=01010101010101000100010101010010 +
I=01111001001100110110110010110010

I=I <<< keyr[1]
I=10011110010011001101101100101100

F=((s1[Ia] XOR s2[Ib])- s3[Ic])- s3[Id]
s1[Ia]=10101010010100011010011110011011
s2[Ib]=01010111010100111000101011010101 □

F=00100000000101001001111100010111
s3[Ic]=00001100111011010110001111010000 -
F=00111010010010110001011110110000
s4[Id]=10000010011110110110100011010000 +
F=10111001000111111001101110110000

L= L XOR F
L=01001011010011110100110101010000
F=10111001000111111001101110110000 ▫
L=11110010010100001101011011100000

- Iterasi ke-2 Menggunakan Fungsi *CAST* tipe 2

XOR kan L dengan Kunci *Masking* ke-2
I = keym[2] XOR L
keym[2]=01101000110010100011000100111011
L=11110010010100001101011011100000 ▫
I=10011010100110101110011111011011

I=I <<<keyr[2]
I=11010111001111101101110011010100

F=((s1[Ia] - s2[Ib])+ s3[Ic]) XOR s3[Id]
s1[Ia]=10111100001100000110111011011001
s2[Ib]=10100010110101001011011101101101 -
F=00011011010011001000111101100000
s3[Ic]=01111100011000111011001011001111+
F=11101010011011000111101100110010
s4[Id]=00110101101110100011111001001010 ▫
F=11001011100111000010011110001100

R= R XOR F
R=01010101010101000100010101010010
F=11001011100111000010011110001100 ▫
R=10011110110010000110001011011110

- Iterasi ke-3 Menggunakan Fungsi *CAST* tipe 3

Kurangkankan Kunci *Masking* ke-3 dengan R

I = keym[3] - R
keym[3]=00100101101101100101100001101001
R=10011110110010000110001011011110 -
I=10000110111011011111010110001011
I=I <<< keyr[3]
I=10110111110101100010111000011011

((s1[Ia] + s2[Ib])XOR s3[Ic])- s3[Id]
s1[Ia]=10101100001110010101011100001010
s2[Ib]=11000011011110110100110100001001 +
F=10011011101000010111111111000110
s3[Ic]=00100010101100001100000001010100 ▫
F=00010101110010101010110100000111
s4[Id]=00100010010101000000111100101111 -
F=11101000101100010100110100001001
L= L XOR F
L=11110010010100001101011011100000
F=11101000101100010100110100001001 ▫
L=00011010111000011001101111101001

.....
.....
.....
.....

Hasil iterasi ke-12
L = 11010110000110101111101010011001
R = 11111111111100101111010001101100

Tukar L dan R
L= 11111111111100101111010001101100
R= 11010110000110101111101010011001

Gabungkan L dan R
11111111111100101111010001101100110101100001101011111010101011001
Konversi ke *heksadesimal* dan *string*
Heksadesimal = fff2f46cd61afa99
String = ÿòôlÖú™

3.3.3 Perhitungan Dekripsi

Selanjutnya akan dilakukan dekripsi dari *stringciphertext* di atas dengan *key* yang sama saat enkripsi yaitu ”CAST-128”. Langkah-langkah dekripsi akan seperti berikut :

1. Konversi *ciphertext* kedalam bentuk biner. Hasil konversi *chipertext* ke biner adalah sebagai berikut :
   11010110000110101111101010011001111111111111110010111
   1010001101100

2. Bagi ke dalam 2 blok masing-masing 32-bit
   L=11111111111100101111010001101100
   R=11010110000110101111101010011001

   Tukar L dan R
   L= 11010110000110101111101010011001
   R= 11111111111100101111010001101100

3. Lakukan enkripsi algoritma *CAST-128* dengan menggunakan urutan iterasi dan Kunci yang terbalik dari proses enkripsi. Iterasi dimulai dari 12 dan berakhir pada iterasi ke-1 kemudian di gabung dengan L dan R.

   - Iterasi ke-12 Menggunakan Fungsi *CAST* tipe 3

     Kurangkankan Kunci *Masking* ke-12 dengan L
     I = keym[12] - L
     keym[12]=00100111110001101110101110111110
     L=11010110000110101111101010011001-
     I=01010001101010111111000100100101

     I=I geser keyr[12]
     I=10010101010001101010111111000100

     F=((s1[Ia] + s2[Ib])XOR s3[Ic])- s3[Id]
     s1[Ia]=11010101101111011001111010011000
     s2[Ib]=10111001010010011110001101010100+
     F=10101111100111100100000010111111
     s3[Ic]=10110011010001111100110010010110 ▫
     F=10000011101000011100110001111010
     s4[Id]=01000111010001001110101011010100 -

F=01011010000100010100100001100101

R= R XOR F
R=11111111111100101111010001101100
F=01011010000100010100100001100101 □
R=10100101111000111011110000001001

- Iterasi ke-11 Menggunakan Fungsi *CAST* tipe 2

XOR kan R dengan Kunci *Masking* ke-11
I = keym[11] XOR R
keym[11]=11111011100010001101100100010101
R=10100101111000111011110000001001 □
I=01011110011010110110010100011100

I=I geser keyr[11]
I=01000111000101111001101011011001

F=((s1[Ia] - s2[Ib])+ s3[Ic]) XOR s3[Id]
s1[Ia]=10110000010001100110100111111110
s2[Ib]=10100010110100101011110100101101 -
F=11001111100000101001110000000011
s3[Ic]=11011000011100010000111101101001 +
F=10011110001011111010000011110111
s4[Id]=01010110010010001111011100100101 □
F=01111011000011000001111010000000

L= L XOR F
L=11010110000110101111101010011001
F=01111011000011000001111010000000 □
L=10101101000101101110010000011001

- Iterasi ke-10 Menggunakan Fungsi *CAST* tipe 1

Tambahkan L dengan Kunci *Masking* ke-10
I = keym[10] + L
keym[10]=00011001001100111101100100000011
L=10101101000101101110010000011001+
I=11000110010010101011110100011100

I=I geser keyr[10]
I=10101111010001110011000110010010
F=((s1[Ia] XOR s2[Ib])- s3[Ic])- s3[Id]
s1[Ia]=10110011010001111100110010010110
s2[Ib]=10110000010001100110100111111110 ▫
F=11111101100000001111011010100010
s3[Ic]=01010000101110110110010010100010 -
F=01000111101001010011000001010111
s4[Id]=10011110001000000100110100000010 +
F=00110111011010000111110101110111

R= R XOR F
R=10100101111000111011110000001001
F=00110111011010000111110101110111 ▫
R=10010010100010111100000101111110
.....
.....
.....
.....

Hasil Keluaran iterasi ke-1
L=01001011010011110100110101010000
R=01010101010101000100010101010010

Gabungkan L dan R
01001011010011110100110101010000010101010101010001000101010010

Konversi ke *heksadesimal* dan *string*
Heksadesimal = 4b4f4d5055544552
String = KOMPUTER

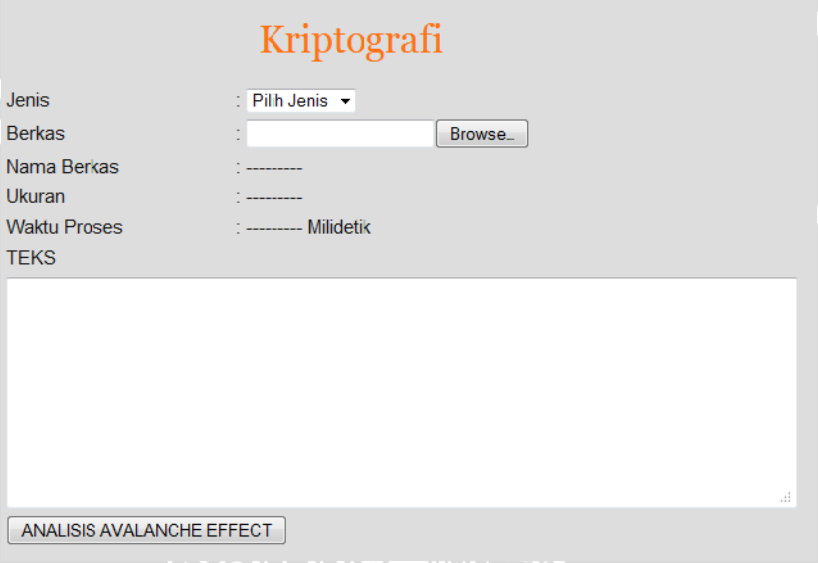
3.4 Perancangan *Interface*

Rancangan *interface* untuk aplikasi ini akan memiliki beberapa menu yaitu :

1. Jenis : Untuk memilih jenis kriptografi. Enkripsi atau dekripsi.
2. Berkas : Untuk upload file **.txt*.

3. Nama Berkas : Menampilkan nama berkas yang diupload.
4. Ukuran : Menampilkan ukuran berkas yang diupload.
5. Waktu Proses : Menampilkan lama proses enkripsi atau dekripsi
6. Teks : Berisi teks hasil enkripsi atau dekripsi
7. Analisis Avalanche Effect : Link menuju halaman analisi *Avalanche Effect*.

Rancangan *interface* untuk aplikasi enkripsi dan dekripsi *CAST-128* dapat dilihat pada gambar 3.11.



Gambar 3.12 Rancangan interface

3.5 Perancangan Analisis Waktu proses dan *Avalanche Effect*

Pada aplikasi enkripsi dan dekripsi dengan algoritma *CAST-128* ini dapat menerima masukan berupa *file* teks. Parameter yang digunakan untuk melakukan uji coba pada *file* teks tersebut diantaranya adalah waktu proses dan *avalanche effect.*

Pada analisis waktu proses akan dihitung waktu proses enkripsi dan dekripsi pada beberapa *file* dengan ukuran berbeda. *File-file* yang akan diuji diantaranya berukuran 50Kb, 100Kb, 150Kb 200Kb, 250Kb, 300Kb, 350Kb, 400Kb, 450Kb dan 500Kb. Tujuan dari uji coba waktu proses ini adalah untuk mengetahui apakah nanti

waktu proses akan berbanding lurus dengan ukuran *file* atau tidak. Bentuk tabel dari hasil uji waktu proses ini dapat dilihat pada tabel 3.3 dan tabel 3.4.

Pengujian selanjutnya adalah pengujian *avalanche effect*. Pada pengujian ini akan digunakan lima *file* dengan ukuran yang berbeda mulai ukuran 8 *bytes* hingga 40 *bytes*. Kemudian beberapa perubahan yang dilakukan dengan uji coba dengan parameter *avalance effect* adalah :

1. Perubahan bit/*byte* serta posisi pada *plaintext* terhadap *chipertext* dengan *key* enkripsi yang sama.
2. Perubahan bit/*byte* serta posisi pada *key* enkripsi terhadap *chipertext* dengan *plaintext* yang sama.
3. Perubahan bit/*byte* serta posisi pada *chipertext* terhadap *plaintext* dengan *key* enkripsi yang sama.

Tujuan dari uji coba menggunakan *avalanche effect* ini adalah untuk mengetahui ketahanan algoritma kriptografi *CAST-128*. Bentuk tabel hasil analisis *avalanche effect* ini dapat dilihat pada tabel 3.5.

Tabel 3.3 Rancangan tabel hasil uji untuk parameter waktu proses enkripsi.

Nama *File*	Ukuran *File*	Waktu Proses Enkripsi

Keterangan tabel 3.3 :

- Kolom Nama *File* berisikan nama *file* yang akan diuji
- Kolom Ukuran *File* berisikan ukuran *file* yang akan di uji (dalam *byte*)
- Waktu Proses Enkripsi berisikan waktu yang digunakan dalam mengenkripsi sebuah *file* (dalam milidetik)

Tabel 3.4 Rancangan tabel hasil uji untuk parameter waktu proses dekripsi.

Nama *File*	Ukuran *File*	Waktu Proses Dekripsi

Keterangan tabel 3.4 :
- Kolom Nama *File* berisikan nama *file* yang akan diuji
- Kolom Ukuran *File* berisikan ukuran *file* yang akan di uji (dalam *byte*)
- Waktu Proses Dekripsi berisikan waktu yang digunakan dalam mendekripsi sebuah *file* (dalam milidetik)

Tabel 3.5 Rancangan tabel uji untuk parameter *avalanche effect*

Nama *File*	Ukuran *File*	Jumlah Perubahan bit/byte	Posisi Perubahan	*Avalanche Effect* (%)

Keterangan tabel 3.3 :
- Kolom Nama *File* berisikan nama *file* yang akan diuji
- Kolom Ukuran *File* berisikan ukuran *file* yang akan di uji (dalam byte).
- Kolom Jumlah Perubahan bit/*byte* berisikan jumlah bit atau *byte* yang diubah.
- Kolom Posisi Perubahan berisikan posisi bit/*byte* yang dirubah misal di awal, tengah atau akhir.
- Kolom *Avalanche Effect* berisikan nilai dari *avalanche effectfile* yang diuji.