BAB IV
IMPLEMENTASI DAN PEMBAHASAN

Pada bab ini akan dilakukan implementasi algoritma *CAST-128* dan analisis pada perangkat lunak tersebut.

4.1 Lingkungan Implementasi

Pada lingkungan implementasi ada dua faktor yang mempengaruhi yaitu lingkungan perangkat keras dan lingkungan perangkat lunak.

4.1.2 Lingkungan Perangkat Keras

Perangkat keras yang digunakan untuk implementasi Algoritma *CAST-128* memiliki spesifikasi sebagai berikut :

1. *Processor* intel *Core i3* @2,27GHz.
2. *Memory*3 GB.
3. *Harddisk* 320 GB.

4.1.3 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan untuk implementasi Algoritma *CAST-128* adalah sebagai berikut :

1. Sistem Operasi Microsoft Windows 7 Ultimate.
2. *Tools programing* : PHP.

4.2 Implementasi Program

Berdasarkan perancangan pada subbab 3.2 maka pada bab ini akan dibahas mengenai implementasi dari perancangan tersebut.

4.2.1 Proses *Input File*

Pada perangkat lunak ini sebuah *file* **.txt* sebagai *inputan* dari *user* akan dibaca perangkat lunak per*byte* dan disimpan ke variabel *source* dan kemudian *byte-byte* tersebut akan dikonversi kembali ke *string* untuk selanjutnya dilakukan proses enkripsi.

Untuk melakukan *openfile* ini digunakan `move_uploaded_file` dan menggunakan fungsi `file_get_contents()`. Dimana pada proses open ini akan didapatkan beberapa atribut berkas yang diunggah seperti tipe, ukuran, nama dan lokasi asal berkas. Fungsi

`move_uploaded_file` untuk memindahakan atau mengunggah berkas pada server. Dan fungsi `file_get_contents` untuk mendapatkan atau membaca isi file yang diunggah. Ketiga fungsi tersebut merupakan fungsi yag sudah tersedia pada PHP. Untuk lebih jelasnya proses *input file* dapat dilihat pada *sourcecode* 4.1.

```
$tipe_file   = $_FILES['berkas']['type'];
$lokasi_file = $_FILES['berkas']['tmp_name'];
$nama_file   = $_FILES['berkas']['name'];
$ukuran_file = $_FILES['berkas']['size'];

if ($tipe_file == "text/txt")
{
$direktori ="berkas/$nama_file";
move_uploaded_file($lokasi_file,"$direktori");
$asli=file_get_contents($direktori);
$panjang=strlen($asli);
$hitung=0;
$aa=0;
$bb=80;
while ($aa<=$panjang)
{
$kata[$hitung] = substr($asli,$aa,$bb);
$aa=$aa+$bb;
$hitung++;
}
if($panjang%$bb!=0)
{
$kata[$hitung] = substr($asli,$aa);
$hitung++;
}
```

Sourcecode 4.1 Proses `Open`

4.2.2 Proses Penghitungan *Subkey*

Pada proses penghitungan *subkey* proses pertama yang dilakukan adalah mengambil *stringkey* masukan dari *user* yang kemudian di*padding* dan dikonversi ke dalam *byte* dan disimpan dalam variabel `kunci_masking` dan `kunci_rotasi` yang bertipe *byte array*. Hal tersebut terdapat pada *Sourcecode 4.2*.

```
$this->r = strlen($kunci) < 11? 12 : 16;
    $kunci = $this->formatkunci($kunci);
    $this->penjadwalan_kunci($kunci);
```

Sourcecode 4.2Proses Perhitungan Subkey

Pada baris pertama *Sourcecode 4.2* dihitung panjang karakter kunci yang di*input*kan untuk mengetahui panjang rotasi yang akan digunakan dalam proses enkripsi/deskipsi. Dan fungsi `formatkunci`

untuk memecah kunci ke dalam array dengan bentuk nilai ASCII dari masing-masing karakter dalam kunci. Untuk lebih jelasnya fungsi `formatkunci` dapat dilihat pada *Sourcecode 4.3*.

```
function formatkunci($data)
	{
		$return = array_values(unpack("C*",$data));
		for($i = count($return); $i < 16; $i++)
			$return[$i] = 0;
		return $return;
	}
```

Sourcecode 4.3 Fungsi `formatkunci`.

Fungsi `penjadwalan_kunci` berfungsi untuk menghitung ke-16 kunci masking dan ke-16 fungsi rotasi. Untuk lebih jelasnya fungsi `penjadwalan_kunci` dapat dilihat pada *Sourcecode 4.4*.

```
function penjadwalan_kunci($kunci)
{
$x0x1x2x3 = $this->buat32($kunci,0x0);
$x4x5x6x7 = $this->buat32($kunci,0x4);
$x8x9xAxB = $this->buat32($kunci,0x8);
$xCxDxExF = $this->buat32($kunci,0xC);

$b = $this->pecah32($x0x1x2x3); $x0 = $b[0]; $x1 = $b[1]; $x2 =
  $b[2]; $x3 = $b[3];
$b = $this->pecah32($x4x5x6x7); $x4 = $b[0]; $x5 = $b[1]; $x6 =
  $b[2]; $x7 = $b[3];
$b = $this->pecah32($x8x9xAxB); $x8 = $b[0]; $x9 = $b[1]; $xA =
  $b[2]; $xB = $b[3];
$b = $this->pecah32($xCxDxExF); $xC = $b[0]; $xD = $b[1]; $xE =
  $b[2]; $xF = $b[3];

$z0z1z2z3  =  $x0x1x2x3  ^  $this->s5[$xD]  ^  $this->s6[$xF]  ^
  $this->s7[$xC] ^ $this->s8[$xE] ^ $this->s7[$x8];
$b = $this->pecah32($z0z1z2z3); $z0 = $b[0]; $z1 = $b[1]; $z2 =
  $b[2]; $z3 = $b[3];
$z4z5z6z7  =  $x8x9xAxB  ^  $this->s5[$z0]  ^  $this->s6[$z2]  ^
  $this->s7[$z1] ^ $this->s8[$z3] ^ $this->s8[$xA];
$b = $this->pecah32($z4z5z6z7); $z4 = $b[0]; $z5 = $b[1]; $z6 =
  $b[2]; $z7 = $b[3];
$z8z9zAzB  =  $xCxDxExF  ^  $this->s5[$z7]  ^  $this->s6[$z6]  ^
  $this->s7[$z5] ^ $this->s8[$z4] ^ $this->s5[$x9];
$b = $this->pecah32($z8z9zAzB); $z8 = $b[0]; $z9 = $b[1]; $zA =
  $b[2]; $zB = $b[3];
$zCzDzEzF  =  $x4x5x6x7  ^  $this->s5[$zA]  ^  $this->s6[$z9]  ^
  $this->s7[$zB] ^ $this->s8[$z8] ^ $this->s6[$xB];
$b = $this->pecah32($zCzDzEzF); $zC = $b[0]; $zD = $b[1]; $zE =
  $b[2]; $zF = $b[3];

$this->kunci_masking[1]  =  $this->s5[$z8]  ^  $this->s6[$z9]  ^
  $this->s7[$z7] ^ $this->s8[$z6] ^ $this->s5[$z2];
```

```
$this->kunci_masking[2]  =  $this->s5[$zA]  ^  $this->s6[$zB]  ^
  $this->s7[$z5] ^ $this->s8[$z4] ^ $this->s6[$z6];
$this->kunci_masking[3]  =  $this->s5[$zC]  ^  $this->s6[$zD]  ^
  $this->s7[$z3] ^ $this->s8[$z2] ^ $this->s7[$z9];
$this->kunci_masking[4]  =  $this->s5[$zE]  ^  $this->s6[$zF]  ^
  $this->s7[$z1] ^ $this->s8[$z0] ^ $this->s8[$zC];

$x0x1x2x3  =  $z8z9zAzB  ^  $this->s5[$z5]  ^  $this->s6[$z7]  ^
  $this->s7[$z4] ^ $this->s8[$z6] ^ $this->s7[$z0];
$b = $this->pecah32($x0x1x2x3); $x0 = $b[0]; $x1 = $b[1]; $x2 =
  $b[2]; $x3 = $b[3];
$x4x5x6x7  =  $z0z1z2z3  ^  $this->s5[$x0]  ^  $this->s6[$x2]  ^
  $this->s7[$x1] ^ $this->s8[$x3] ^ $this->s8[$z2];
$b = $this->pecah32($x4x5x6x7); $x4 = $b[0]; $x5 = $b[1]; $x6 =
  $b[2]; $x7 = $b[3];
$x8x9xAxB  =  $z4z5z6z7  ^  $this->s5[$x7]  ^  $this->s6[$x6]  ^
  $this->s7[$x5] ^ $this->s8[$x4] ^ $this->s5[$z1];
$b = $this->pecah32($x8x9xAxB); $x8 = $b[0]; $x9 = $b[1]; $xA =
  $b[2]; $xB = $b[3];
$xCxDxExF  =  $zCzDzEzF  ^  $this->s5[$xA]  ^  $this->s6[$x9]  ^
  $this->s7[$xB] ^ $this->s8[$x8] ^ $this->s6[$z3];
$b = $this->pecah32($xCxDxExF); $xC = $b[0]; $xD = $b[1]; $xE =
  $b[2]; $xF = $b[3];

$this->kunci_masking[5]   =  $this->s5[$x3]  ^  $this->s6[$x2]  ^
  $this->s7[$xC] ^ $this->s8[$xD] ^ $this->s5[$x8];
$this->kunci_masking[6]   =  $this->s5[$x1]  ^  $this->s6[$x0]  ^
  $this->s7[$xE] ^ $this->s8[$xF] ^ $this->s6[$xD];
$this->kunci_masking[7]   =  $this->s5[$x7]  ^  $this->s6[$x6]  ^
  $this->s7[$x8] ^ $this->s8[$x9] ^ $this->s7[$x3];
$this->kunci_masking[8]   =  $this->s5[$x5]  ^  $this->s6[$x4]  ^
  $this->s7[$xA] ^ $this->s8[$xB] ^ $this->s8[$x7];

$z0z1z2z3  =  $x0x1x2x3  ^  $this->s5[$xD]  ^  $this->s6[$xF]  ^
  $this->s7[$xC] ^ $this->s8[$xE] ^ $this->s7[$x8];
$b = $this->pecah32($z0z1z2z3); $z0 = $b[0]; $z1 = $b[1]; $z2 =
  $b[2]; $z3 = $b[3];
$z4z5z6z7  =  $x8x9xAxB  ^  $this->s5[$z0]  ^  $this->s6[$z2]  ^
  $this->s7[$z1] ^ $this->s8[$z3] ^ $this->s8[$xA];
      $b  =  $this->pecah32($z4z5z6z7);  $z4  =  $b[0];  $z5  =
  $b[1]; $z6 = $b[2]; $z7 = $b[3];
$z8z9zAzB  =  $xCxDxExF  ^  $this->s5[$z7]  ^  $this->s6[$z6]  ^
  $this->s7[$z5] ^ $this->s8[$z4] ^ $this->s5[$x9];
$b = $this->pecah32($z8z9zAzB); $z8 = $b[0]; $z9 = $b[1]; $zA =
  $b[2]; $zB = $b[3];
$zCzDzEzF  =  $x4x5x6x7  ^  $this->s5[$zA]  ^  $this->s6[$z9]  ^
  $this->s7[$zB] ^ $this->s8[$z8] ^ $this->s6[$xB];
$b = $this->pecah32($zCzDzEzF); $zC = $b[0]; $zD = $b[1]; $zE =
  $b[2]; $zF = $b[3];

$this->kunci_masking[9]   =  $this->s5[$z3]  ^  $this->s6[$z2]  ^
  $this->s7[$zC] ^ $this->s8[$zD] ^ $this->s5[$z9];
$this->kunci_masking[10]  =  $this->s5[$z1]  ^  $this->s6[$z0]  ^
  $this->s7[$zE] ^ $this->s8[$zF] ^ $this->s6[$zC];
$this->kunci_masking[11]  =  $this->s5[$z7]  ^  $this->s6[$z6]  ^
  $this->s7[$z8] ^ $this->s8[$z9] ^ $this->s7[$z2];
```

```
$this->kunci_masking[12] = $this->s5[$z5] ^ $this->s6[$z4] ^
 $this->s7[$zA] ^ $this->s8[$zB] ^ $this->s8[$z6];

$x0x1x2x3 = $z8z9zAzB ^ $this->s5[$z5] ^ $this->s6[$z7] ^
 $this->s7[$z4] ^ $this->s8[$z6] ^ $this->s7[$z0];
$b = $this->pecah32($x0x1x2x3); $x0 = $b[0]; $x1 = $b[1]; $x2 =
 $b[2]; $x3 = $b[3];
$x4x5x6x7 = $z0z1z2z3 ^ $this->s5[$x0] ^ $this->s6[$x2] ^
 $this->s7[$x1] ^ $this->s8[$x3] ^ $this->s8[$z2];
$b = $this->pecah32($x4x5x6x7); $x4 = $b[0]; $x5 = $b[1]; $x6 =
 $b[2]; $x7 = $b[3];
$x8x9xAxB = $z4z5z6z7 ^ $this->s5[$x7] ^ $this->s6[$x6] ^
 $this->s7[$x5] ^ $this->s8[$x4] ^ $this->s5[$z1];
$b = $this->pecah32($x8x9xAxB); $x8 = $b[0]; $x9 = $b[1]; $xA =
 $b[2]; $xB = $b[3];
$xCxDxExF = $zCzDzEzF ^ $this->s5[$xA] ^ $this->s6[$x9] ^
 $this->s7[$xB] ^ $this->s8[$x8] ^ $this->s6[$z3];
$b = $this->pecah32($xCxDxExF); $xC = $b[0]; $xD = $b[1]; $xE =
 $b[2]; $xF = $b[3];

$this->kunci_masking[13] = $this->s5[$x8] ^ $this->s6[$x9] ^
 $this->s7[$x7] ^ $this->s8[$x6] ^ $this->s5[$x3];
$this->kunci_masking[14] = $this->s5[$xA] ^ $this->s6[$xB] ^
 $this->s7[$x5] ^ $this->s8[$x4] ^ $this->s6[$x7];
$this->kunci_masking[15] = $this->s5[$xC] ^ $this->s6[$xD] ^
 $this->s7[$x3] ^ $this->s8[$x2] ^ $this->s7[$x8];
$this->kunci_masking[16] = $this->s5[$xE] ^ $this->s6[$xF] ^
 $this->s7[$x1] ^ $this->s8[$x0] ^ $this->s8[$xD];

$z0z1z2z3 = $x0x1x2x3 ^ $this->s5[$xD] ^ $this->s6[$xF] ^
 $this->s7[$xC] ^ $this->s8[$xE] ^ $this->s7[$x8];
$b = $this->pecah32($z0z1z2z3); $z0 = $b[0]; $z1 = $b[1]; $z2 =
 $b[2]; $z3 = $b[3];
$z4z5z6z7 = $x8x9xAxB ^ $this->s5[$z0] ^ $this->s6[$z2] ^
 $this->s7[$z1] ^ $this->s8[$z3] ^ $this->s8[$xA];
$b = $this->pecah32($z4z5z6z7); $z4 = $b[0]; $z5 = $b[1]; $z6 =
 $b[2]; $z7 = $b[3];
$z8z9zAzB = $xCxDxExF ^ $this->s5[$z7] ^ $this->s6[$z6] ^
 $this->s7[$z5] ^ $this->s8[$z4] ^ $this->s5[$x9];
$b = $this->pecah32($z8z9zAzB); $z8 = $b[0]; $z9 = $b[1]; $zA =
 $b[2]; $zB = $b[3];
$zCzDzEzF = $x4x5x6x7 ^ $this->s5[$zA] ^ $this->s6[$z9] ^
 $this->s7[$zB] ^ $this->s8[$z8] ^ $this->s6[$xB];
$b = $this->pecah32($zCzDzEzF); $zC = $b[0]; $zD = $b[1]; $zE =
 $b[2]; $zF = $b[3];

$this->kunci_rotasi[1] = 0x1F & ($this->s5[$z8] ^ $this-
 >s6[$z9] ^ $this->s7[$z7] ^ $this->s8[$z6] ^ $this->s5[$z2]);
$this->kunci_rotasi[2] = 0x1F & ($this->s5[$zA] ^ $this-
 >s6[$zB] ^ $this->s7[$z5] ^ $this->s8[$z4] ^ $this->s6[$z6]);
$this->kunci_rotasi[3] = 0x1F & ($this->s5[$zC] ^ $this-
 >s6[$zD] ^ $this->s7[$z3] ^ $this->s8[$z2] ^ $this->s7[$z9]);
$this->kunci_rotasi[4] = 0x1F & ($this->s5[$zE] ^ $this-
 >s6[$zF] ^ $this->s7[$z1] ^ $this->s8[$z0] ^ $this->s8[$zC]);

$x0x1x2x3 = $z8z9zAzB ^ $this->s5[$z5] ^ $this->s6[$z7] ^
```

```
  $this->s7[$z4] ^ $this->s8[$z6] ^ $this->s7[$z0];
$b = $this->pecah32($x0x1x2x3); $x0 = $b[0]; $x1 = $b[1]; $x2 =
  $b[2]; $x3 = $b[3];
$x4x5x6x7 = $z0z1z2z3 ^ $this->s5[$x0] ^ $this->s6[$x2] ^
  $this->s7[$x1] ^ $this->s8[$x3] ^ $this->s8[$z2];
$b = $this->pecah32($x4x5x6x7); $x4 = $b[0]; $x5 = $b[1]; $x6 =
  $b[2]; $x7 = $b[3];
$x8x9xAxB = $z4z5z6z7 ^ $this->s5[$x7] ^ $this->s6[$x6] ^
  $this->s7[$x5] ^ $this->s8[$x4] ^ $this->s5[$z1];
$b = $this->pecah32($x8x9xAxB); $x8 = $b[0]; $x9 = $b[1]; $xA =
  $b[2]; $xB = $b[3];
$xCxDxExF = $zCzDzEzF ^ $this->s5[$xA] ^ $this->s6[$x9] ^
  $this->s7[$xB] ^ $this->s8[$x8] ^ $this->s6[$z3];
$b = $this->pecah32($xCxDxExF); $xC = $b[0]; $xD = $b[1]; $xE =
  $b[2]; $xF = $b[3];

$this->kunci_rotasi[5]    = 0x1F & ($this->s5[$x3] ^ $this-
  >s6[$x2] ^ $this->s7[$xC] ^ $this->s8[$xD] ^ $this->s5[$x8]);
$this->kunci_rotasi[6]    = 0x1F & ($this->s5[$x1] ^ $this-
  >s6[$x0] ^ $this->s7[$xE] ^ $this->s8[$xF] ^ $this->s6[$xD]);
$this->kunci_rotasi[7]    = 0x1F & ($this->s5[$x7] ^ $this-
  >s6[$x6] ^ $this->s7[$x8] ^ $this->s8[$x9] ^ $this->s7[$x3]);
$this->kunci_rotasi[8]    = 0x1F & ($this->s5[$x5] ^ $this-
  >s6[$x4] ^ $this->s7[$xA] ^ $this->s8[$xB] ^ $this->s8[$x7]);

$z0z1z2z3 = $x0x1x2x3 ^ $this->s5[$xD] ^ $this->s6[$xF] ^
  $this->s7[$xC] ^ $this->s8[$xE] ^ $this->s7[$x8];
$b = $this->pecah32($z0z1z2z3); $z0 = $b[0]; $z1 = $b[1]; $z2 =
  $b[2]; $z3 = $b[3];
$z4z5z6z7 = $x8x9xAxB ^ $this->s5[$z0] ^ $this->s6[$z2] ^
  $this->s7[$z1] ^ $this->s8[$z3] ^ $this->s8[$xA];
$b = $this->pecah32($z4z5z6z7); $z4 = $b[0]; $z5 = $b[1]; $z6 =
  $b[2]; $z7 = $b[3];
$z8z9zAzB = $xCxDxExF ^ $this->s5[$z7] ^ $this->s6[$z6] ^
  $this->s7[$z5] ^ $this->s8[$z4] ^ $this->s5[$x9];
$b = $this->pecah32($z8z9zAzB); $z8 = $b[0]; $z9 = $b[1]; $zA =
  $b[2]; $zB = $b[3];
$zCzDzEzF = $x4x5x6x7 ^ $this->s5[$zA] ^ $this->s6[$z9] ^
  $this->s7[$zB] ^ $this->s8[$z8] ^ $this->s6[$xB];
$b = $this->pecah32($zCzDzEzF); $zC = $b[0]; $zD = $b[1]; $zE =
  $b[2]; $zF = $b[3];

$this->kunci_rotasi[9]    = 0x1F & ($this->s5[$z3] ^ $this-
  >s6[$z2] ^ $this->s7[$zC] ^ $this->s8[$zD] ^ $this->s5[$z9]);
$this->kunci_rotasi[10]  = 0x1F & ($this->s5[$z1] ^ $this-
  >s6[$z0] ^ $this->s7[$zE] ^ $this->s8[$zF] ^ $this->s6[$zC]);
        $this->kunci_rotasi[11]  = 0x1F & ($this->s5[$z7] ^
  $this->s6[$z6] ^ $this->s7[$z8] ^ $this->s8[$z9] ^ $this-
  >s7[$z2]);
        $this->kunci_rotasi[12]  = 0x1F & ($this->s5[$z5] ^
  $this->s6[$z4] ^ $this->s7[$zA] ^ $this->s8[$zB] ^ $this-
  >s8[$z6]);

$x0x1x2x3 = $z8z9zAzB ^ $this->s5[$z5] ^ $this->s6[$z7] ^
  $this->s7[$z4] ^ $this->s8[$z6] ^ $this->s7[$z0];
$b = $this->pecah32($x0x1x2x3); $x0 = $b[0]; $x1 = $b[1]; $x2 =
```

```
  $b[2]; $x3 = $b[3];
$x4x5x6x7  =  $z0z1z2z3  ^  $this->s5[$x0]  ^  $this->s6[$x2]  ^
  $this->s7[$x1] ^ $this->s8[$x3] ^ $this->s8[$z2];
$b = $this->pecah32($x4x5x6x7); $x4 = $b[0]; $x5 = $b[1]; $x6 =
  $b[2]; $x7 = $b[3];
$x8x9xAxB  =  $z4z5z6z7  ^  $this->s5[$x7]  ^  $this->s6[$x6]  ^
  $this->s7[$x5] ^ $this->s8[$x4] ^ $this->s5[$z1];
$b = $this->pecah32($x8x9xAxB); $x8 = $b[0]; $x9 = $b[1]; $xA =
  $b[2]; $xB = $b[3];
$xCxDxExF  =  $zCzDzEzF  ^  $this->s5[$xA]  ^  $this->s6[$x9]  ^
  $this->s7[$xB] ^ $this->s8[$x8] ^ $this->s6[$z3];
$b = $this->pecah32($xCxDxExF); $xC = $b[0]; $xD = $b[1]; $xE =
  $b[2]; $xF = $b[3];

$this->kunci_rotasi[13]  =  0x1F  &  ($this->s5[$x8]  ^  $this-
  >s6[$x9] ^ $this->s7[$x7] ^ $this->s8[$x6] ^ $this->s5[$x3]);
$this->kunci_rotasi[14]  =  0x1F  &  ($this->s5[$xA]  ^  $this-
  >s6[$xB] ^ $this->s7[$x5] ^ $this->s8[$x4] ^ $this->s6[$x7]);
$this->kunci_rotasi[15]  =  0x1F  &  ($this->s5[$xC]  ^  $this-
  >s6[$xD] ^ $this->s7[$x3] ^ $this->s8[$x2] ^ $this->s7[$x8]);
$this->kunci_rotasi[16]  =  0x1F  &  ($this->s5[$xE]  ^  $this-
  >s6[$xF] ^ $this->s7[$x1] ^ $this->s8[$x0] ^ $this->s8[$xD]);
}
```

Sourcecode 4.4 fungsi penjadwalan_kunci

Fungsi `penjadwalan_kunci` pada *Sourcecode 4.4* sesuai dengan algroritma pembangkitan kunci pada subbab 2.5.3.

4.2.3 Proses Enkripsi

Pada saat *file* dibuka isi dari variabel `source` yang berupa *byte array* dikonversi ke dalam *string* dan disimpan dalam variabel `$asli` agar bisa dituliskan ke `textarea`. Kemudian saat akan melakukan enkripsi ini isi dari variabel `$asli` akan dikonversi kembali ke dalam bentuk *byte array* untuk dienkripsi. Hal tersebut tedapat pada Pada *Sourcecode 4.5*.

```
$a=0;
while($a<$hitung)
{
$hasil[$a]=$coba->enkripsi($kata[$a], $_POST[kunci]);
$a++;
}
```

Sourcecode 4.5 Proses Enkripsi

Fungsi `enkripsi` pada *Sourcecode 4.5* memanggil fungsi `enkripsi` pada *sourcecode 4.6*. Dimana pada fungsi ini data yang akan dienkripsi akan dipecah pecah kedalam blok-blok dengan ukuran 8 *byte*. Dan akan di*padding* jika panjang kurang dari 8 *byte*. Kemudian dilakukan enkripsi perblok.

```
function enkripsi($data,$kunci=null)
{
$return = '';
$this->r = strlen($kunci) < 11? 12 : 16;
$kunci = $this->formatkunci($kunci);
$this->penjadwalan_kunci($kunci);

$prevcipher = $this->iv;

for($i = 0; $i < strlen($data); $i += $this->ukuranblok)
{
$blok = substr($data,$i,$this->ukuranblok);

if(strlen($blok) < $this->ukuranblok)
       $blok = str_pad($blok,8,"\0");

$blok = $blok ^ $prevcipher;
$prevcipher = $this->enkripsi_blok($blok);
$return .= $prevcipher;
}

return $return;
}
```

Sourcecode 4.6 Fungsi *enkripsi*

Fungsi enkripsi_perblok pada *Sourcecode 4.9* dilakukan rotasi jaringan *feistel* sebanyak 12 atau 16. Sesuai dengan perhitungan jumlah rotasi pada *Sourcecode 4.2*.

```
function enkripsi_blok($data)
{
  $L = $this->formatdata(substr($data,0,4));
  $R = $this->formatdata(substr($data,4,4));

  $L ^= $this->f0($R,$this->kunci_masking[1],$this-
    >kunci_rotasi[1]);
  $R ^= $this->f1($L,$this->kunci_masking[2],$this-
    >kunci_rotasi[2]);
  $L ^= $this->f2($R,$this->kunci_masking[3],$this-
    >kunci_rotasi[3]);
  $R ^= $this->f0($L,$this->kunci_masking[4],$this-
    >kunci_rotasi[4]);
  $L ^= $this->f1($R,$this->kunci_masking[5],$this-
    >kunci_rotasi[5]);
  $R ^= $this->f2($L,$this->kunci_masking[6],$this-
    >kunci_rotasi[6]);
  $L ^= $this->f0($R,$this->kunci_masking[7],$this-
    >kunci_rotasi[7]);
  $R ^= $this->f1($L,$this->kunci_masking[8],$this-
    >kunci_rotasi[8]);
  $L ^= $this->f2($R,$this->kunci_masking[9],$this-
    >kunci_rotasi[9]);
  $R ^= $this->f0($L,$this->kunci_masking[10],$this-
    >kunci_rotasi[10]);
```

```
    $L ^= $this->f1($R,$this->kunci_masking[11],$this-
      >kunci_rotasi[11]);
    $R ^= $this->f2($L,$this->kunci_masking[12],$this-
      >kunci_rotasi[12]);
    if ($this->r > 12)
    {
      $L ^= $this->f0($R,$this->kunci_masking[13],$this-
        >kunci_rotasi[13]);
      $R ^= $this->f1($L,$this->kunci_masking[14],$this-
        >kunci_rotasi[14]);
      $L ^= $this->f2($R,$this->kunci_masking[15],$this-
        >kunci_rotasi[15]);
      $R ^= $this->f0($L,$this->kunci_masking[16],$this-
        >kunci_rotasi[16]);
    }

    $encrypted = pack("N",$R) . pack("N",$L);

    return $encrypted;
}
```

*Sourcecode 4.7*Fungsi *Enkripsi Perblok*

Fungsi `enkripsi_blok` ini terdapat tiga jenis fungsi *CAST* yang digunakan dalam proses enkripsi. Ketiga fungsi tersebut digunakan secara bergantian dengan fungsi f0 terlebih dahulu diikuti f1 dan f2 kemudian f0 kembali dan begitu seterusnya. Untuk lebih jelasnya tentang ketiga fugsi tersebut dapat dilihat pada *Sourcecode 4.8*.

```
function f0($data,$kunci_masking,$kunci_rotasi)
{
$I = $kunci_masking + $data;
$bin = str_pad(decbin($I),32,'0',STR_PAD_LEFT);
$I = bindec(substr($bin,$kunci_rotasi)
      .substr($bin,0,$kunci_rotasi));
$f = $this->s1[($I >> 24) & 0xFF] ^ $this->s2[($I >> 16) &
      0xFF];
$f = $f - $this->s3[($I >> 8) & 0xFF];
$f = $f + $this->s4[$I & 0xFF];
return $f;
}

function f1($data,$kunci_masking,$kunci_rotasi)
{
$I = $kunci_masking ^ $data;
$bin = str_pad(decbin($I),32,'0',STR_PAD_LEFT);
$I = bindec(substr($bin,$kunci_rotasi) .
      substr($bin,0,$kunci_rotasi));
$f = $this->s1[($I >> 24) & 0xFF] - $this->s2[($I >> 16) &
      0xFF];
$f = $f + $this->s3[($I >> 8) & 0xFF];
$f = $f ^ $this->s4[$I & 0xFF];
return $f;
}
```

```
function f2($data,$kunci_masking,$kunci_rotasi)
{
$I = 0xFFFFFFFF & ($kunci_masking - $data);
$bin = str_pad(decbin($I),32,'0',STR_PAD_LEFT);
$I = bindec(substr($bin,$kunci_rotasi) .
      substr($bin,0,$kunci_rotasi));
$f = ($this->s1[($I >> 24) & 0xFF] + $this->s2[($I >> 16) &
      0xFF]);
$f = $f ^ $this->s3[($I >> 8) & 0xFF];
$f = 0xFFFFFFFF & ($f - $this->s4[$I & 0xFF]);
return $f;
}
```

Sourcecode 4.8 Fungsi-fungsi *CAST*

4.2.4 Proses Dekripsi

Pada proses dekripsi sebenarnya tahap-tahapnya hampir sama dengan proses enkripsi perbedaannya hanya pada fungsi `dekripsi_blok`nya saja. Dimana putaran untuk *chipertextnya* merupakan pembalikan dari putaran fungsi `enkripsi_blok`. Fungsi `DekripsiPerBlok` dapat dilihat pada *Sourcecode 4.9.*

```
function dekripsi_blok($data)
{
$R = $this->formatdata(substr($data,0,4));
$L = $this->formatdata(substr($data,4,4));

if ($this->r > 12)
{
  $R ^= $this->f0($L,$this->kunci_masking[16],$this-
        >kunci_rotasi[16]);
  $L ^= $this->f2($R,$this->kunci_masking[15],$this-
        >kunci_rotasi[15]);
  $R ^= $this->f1($L,$this->kunci_masking[14],$this-
        >kunci_rotasi[14]);
  $L ^= $this->f0($R,$this->kunci_masking[13],$this-
        >kunci_rotasi[13]);
}
$R ^= $this->f2($L,$this->kunci_masking[12],$this-
      >kunci_rotasi[12]);
$L ^= $this->f1($R,$this->kunci_masking[11],$this-
      >kunci_rotasi[11]);
$R ^= $this->f0($L,$this->kunci_masking[10],$this-
      >kunci_rotasi[10]);
$L ^= $this->f2($R,$this->kunci_masking[9],$this-
      >kunci_rotasi[9]);
$R ^= $this->f1($L,$this->kunci_masking[8],$this-
      >kunci_rotasi[8]);
$L ^= $this->f0($R,$this->kunci_masking[7],$this-
      >kunci_rotasi[7]);
$R ^= $this->f2($L,$this->kunci_masking[6],$this-
      >kunci_rotasi[6]);
```

```
$L ^= $this->f1($R,$this->kunci_masking[5],$this-
      >kunci_rotasi[5]);
$R ^= $this->f0($L,$this->kunci_masking[4],$this-
      >kunci_rotasi[4]);
$L ^= $this->f2($R,$this->kunci_masking[3],$this-
      >kunci_rotasi[3]);
$R ^= $this->f1($L,$this->kunci_masking[2],$this-
      >kunci_rotasi[2]);
$L ^= $this->f0($R,$this->kunci_masking[1],$this-
      >kunci_rotasi[1]);

$decrypted = pack("N",$L) . pack("N",$R);

return $decrypted;
}
```

Sourcecode 4.9 Fungsi `DekripsiPerBlok`

4.2.5 Proses avalanche effect

Seperti yang sudah dijelaskan pada subbab 2.6 bahwa untuk mengetahui baik tidaknya sebuah algoritma kriptografi maka dilakukan analisis *avalanche effect*. Berikut adalah implementasi dari *avalanche effect*.

```
function av_ef($ubah, $asli)
{
        $i=0;
        $beda=0;
        while($i<strlen($asli))
        {
                if($ubah[$i]!=$asli[$i])
                {
                        $beda=$beda+1;
                }
                $i++;
        }
        $nilai=($beda/strlen($asli))*100;
        return $nilai;
}
```

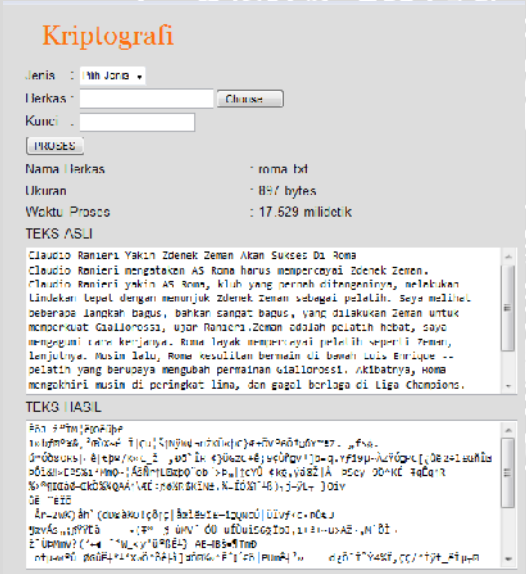
Sourcecode 4.10 Fungsi `av_ef`

Fungsi `av_ef` ini membutuhkan masukan berupa dua buah *string* yang akan dibandingkan, kemudian dibagi panjang salah satu *string* dan dikali seratus. Nilai balikan dari fungsi ini akan berupa nilai *avalanche effect*. Implementasi Fungsi `av_ef` ditunjukan pada *Sourcecode 4.10*.

4.3 Implementasi *Interface*

Berdasarkan rancangan Antarmuka pada subbab 3.4 maka dihasilkan 4 halaman , halaman pertama adalah kriptografi dimana fungsinya untuk melakukan enkripsi atau dekripsi terhadap *file *.txt* kemudian halaman kedua sampai keempat adalah uji1, uji2 dan uji 3 yang berfungsi melakukan analisis *avalanche effect* pada algoritma *CAST-128*.

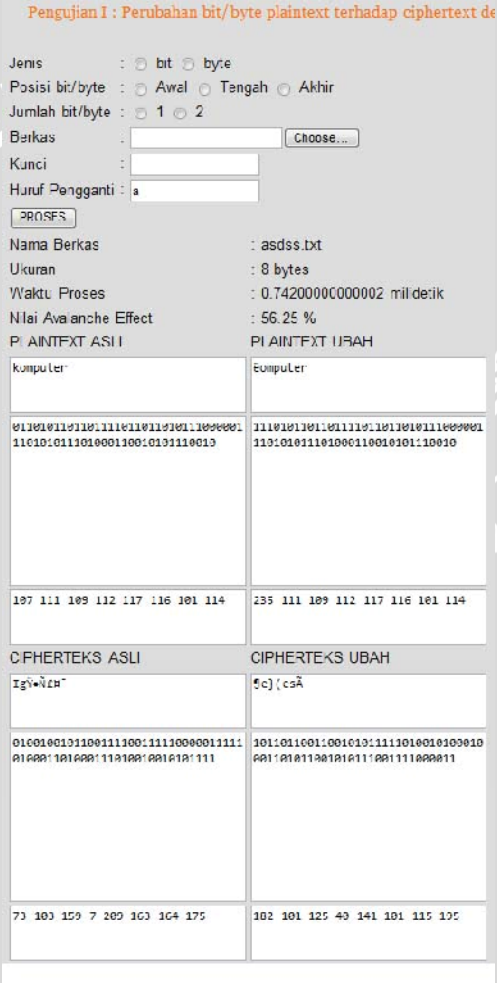
Pada halaman kriptografi ada 2 fitur utama yaitu enkripsi dan dekripsi. Proses berjalannya aplikasi adalah pertama *user* di minta untuk memilih jenis kriptografi, yaitu enkripsi dan dekripsi. Kemudian *user* diminta untuk memilih *file* yang akan diproses dan memasukkan *key*. Setelah klik tombol PROSES maka enkripsi atau dekripsi akan berlangsung dan akan ditampilkan nama berkas, ukuran berkas, waktu proses, teks asli dan teks hasil. Implementasi antarmuka halaman kriptografi dapat dilihat pada gambar 4.1.



Gambar 4.1 Halaman Kriptografi

Pada pengujian I yaitu perubahan bit/*byte plaintext* terhadap *ciphertext* dengan *key* enkripsi yang sama. Proses berjalannya *user* diminta memilih jenis perubahannya, posisi perubahan, jumlah

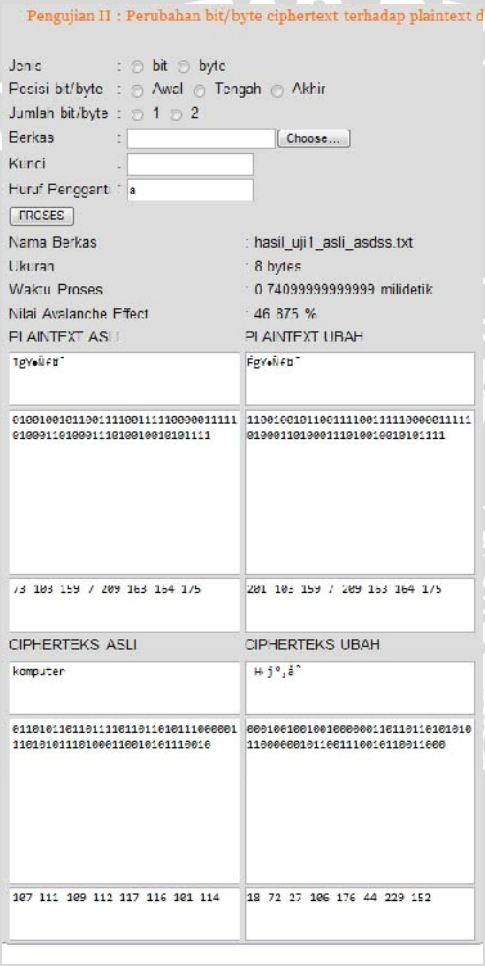
perubahan, memilih berkas, memasukkan kunci dan memasukkan huruf pengganti. Setelah klik tombol PROSES maka enkripsi akan berlangsung dan akan ditampilkan nama berkas, ukuran berkas, waktu proses, teks asli, teks hasil, nilai *avalanche effect*-nya, plaintext asli, plaintext ubah, *ciphertext* asli dan *ciphertext* ubah. Implementasi antarmuka halaman uji1 dapat dilihat pada gambar 4.2.



Gambar 4.2 Halaman Uji1

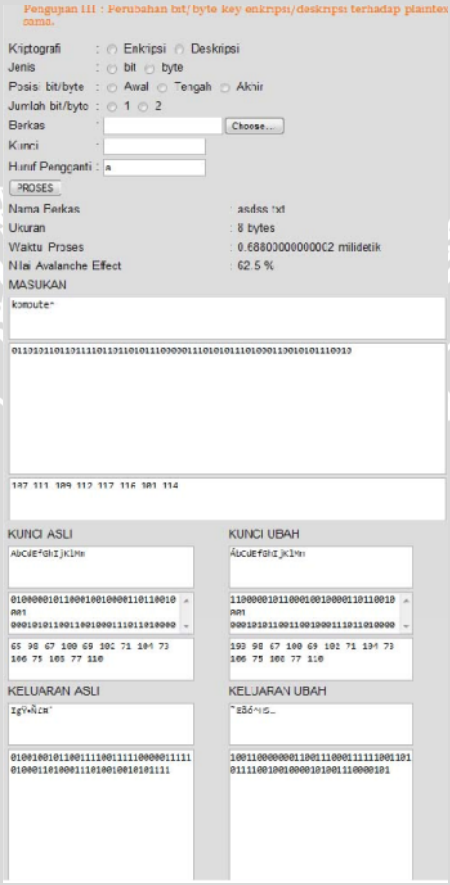
Pada pengujian II yaitu perubahan bit/*byte ciphertext* terhadap *plaintext* dengan *key* enkripsi yang sama. Proses berjalannya *user*

diminta memilih jenis perubahannya, posisi perubahan, jumlah perubahan, memilih berkas, memasukkan kunci dan memasukkan huruf pengganti. Setelah klik tombol PROSES maka dekripsi akan berlangsung dan akan ditampilkan nama berkas, ukuran berkas, waktu proses, teks asli, teks hasil, nilai *avalanche effect*-nya, *plaintext* asli, *plaintext* ubah, *ciphertext* asli dan *ciphertext* ubah. Implementasi antarmuka halaman uji2 dapat dilihat pada gambar 4.3.



Gambar 4.3 Halaman Uji2

Pada pengujian III yaitu perubahan bit/*byte key* enkripsi/dekripsi terhadap *plaintext* dan *ciphertext* yang sama. Proses berjalannya *user* diminta memilih kriptografi, jenis perubahannya, posisi perubahan, jumlah perubahan, memilih berkas, memasukkan kunci dan memasukkan huruf pengganti. Setelah klik tombol PROSES maka dekripsi akan berlangsung dan akan ditampilkan nama berkas, ukuran berkas, waktu proses, teks asli, teks hasil, nilai *avalanche effect*-nya, masukan, kunci asli, kunci ubah, *ciphertext* asli dan *ciphertext* ubah. Dan implementasi antarmuka halaman uji3 dapat dilihat pada gambar 4.4.



Gambar 4.4 Halaman Uji 3

4.4 Hasil Uji

Pada pengujian waktu proses enkripsi dan dekripsi dilakukan uji sebanyak lima kali dengan data yang sama. Pada bab ini hanya akan ditampilkan hasil rata-rata pengujian tersebut. Untuk melihat tabel pengujian secara lengkap dapat dilihat pada lampiran.

Pada tabel 4.1 dan 4.2 akan diperlihatkan hasil uji rata-rata waktu enkripsi dan dekripsi. Kemudian tabel 4.3 sampai 4.8 akan menunjukkan tabel hasil uji dengan parameter *avalanche effect*.

Dari tabel 4.1 dan 4.2 dapat dilihat waktu proses enkripsi dan dekripsi pada tiap *file* uji di atas pada ukuran sama adalah berbeda. Akan tetapi selisih nilainya relatif kecil. Seperti pada ukuran 50KB, rata-rata waktu proses untuk enkripsi sebesar 0,807detik sedangkan pada dekripsi adalah 0,794 detik. Selisihnya adalah 0,011 detik atau 11 milidetik. Detail hasil uji rata-rata waktu proses enkripsi dapat dilihat pada tabel 4.1.

Tabel 4.1 Hasil uji rata-rata waktu proses enkripsi.

Nama *File*	Ukuran *File*	Rata-rata Waktu Proses Enkripsi (s)
50KB.txt	50KB	0,807
100KB.txt	100KB	1,669
150KB.txt	150KB	2,430
200KB.txt	200KB	3,201
250KB.txt	250KB	4,273
300KB.txt	300KB	5,174
350KB.txt	350KB	5,898
400KB.txt	400KB	6,713
450KB.txt	450KB	7,662
500KB.txt	500KB	8,412

Sementara detail hasil uji rata-rata waktu proses dekripsi dapat dilihat pada tabel 4.2.

Tabel 4.2 Hasil uji rata-rata waktu proses dekripsi.

Nama *File*	Ukuran *File*	Rata-rata Waktu Proses Dekripsi (s)
50KB.txt	50KB	0,794
100KB.txt	100KB	1,637
150KB.txt	150KB	2,461
200KB.txt	200KB	3,309
250KB.txt	250KB	4,281
300KB.txt	300KB	5,036
350KB.txt	350KB	6,099
400KB.txt	400KB	6,866
450KB.txt	450KB	7,605
500KB.txt	500KB	8,451

Untuk pengujian terhadap *avalanche effect*. *File* yang akan digunakan adalah 5 buah *file* yang berukuran 8 *byte*, 16 *byte*, 24 *byte*, 32 *byte* dan 40 *byte*. Kemudian beberapa parameter yang diujikan yaitu :

1. Perubahan terhadap *plaintext*, *chipertext*, dan *key*.
2. Perubahan pada bit
3. Perubahan pada *byte*
4. Perubahan pada jumlah *byte*/bit
5. Perubahan pada posisi *byte*/bit

Berikut beberapa tabel hasil pengujian *avalanche effect* :

Yang pertama yaitu perubahan bit dan posisi pada *plaintext* terhadap *chipertext* dengan *key* enkripsi yang sama yaitu ramadhanku dapat dilihat pada tabel 4.3.

Tabel 4.3 Perubahan bit dan posisi pada *plaintext* terhadap *chipertext* dengan *key* enkripsi yang sama (*key*: ramadhanku)

Nama *file*	Ukuran *file*	Jumlah perubahan bit/*byte*	Posisi perubahan	*Avalanche effect*
8byte.txt	8 *byte*	1 bit	awal	57.81
16byte.txt	16*byte*	1 bit	awal	55.47
24byte.txt	24*byte*	1 bit	awal	54.69
32byte.txt	32*byte*	1 bit	awal	55.08
40byte.txt	40*byte*	1 bit	awal	54.38
rata-rata				55.49
8byte.txt	8 *byte*	1 bit	tengah	54.69
16byte.txt	16*byte*	1 bit	tengah	24.22
24byte.txt	24*byte*	1 bit	tengah	36.46
32byte.txt	32*byte*	1 bit	tengah	24.61
40byte.txt	40*byte*	1 bit	tengah	31.88
rata -rata				34.37
8byte.txt	8 *byte*	1 bit	akhir	40.63
16byte.txt	16*byte*	1 bit	akhir	29.69
24byte.txt	24*byte*	1 bit	akhir	16.15
32byte.txt	32*byte*	1 bit	akhir	10.55
40byte.txt	40*byte*	1 bit	akhir	9.69
rata-rata				21.34
8byte.txt	8 *byte*	2 bit	awal	54.69
16byte.txt	16*byte*	2 bit	awal	52.34
24byte.txt	24*byte*	2 bit	awal	51.56
32byte.txt	32*byte*	2 bit	awal	51.95
40byte.txt	40*byte*	2 bit	awal	53.13
rata-rata				52.57
8byte.txt	8 *byte*	2 bit	tengah	56.25
16byte.txt	16*byte*	2 bit	tengah	25
24byte.txt	24*byte*	2 bit	tengah	35.42
32byte.txt	32*byte*	2 bit	tengah	27.34
40byte.txt	40*byte*	2 bit	tengah	30

rata -rata				34,80
8byte.txt	8 *byte*	2 bit	akhir	40.63
16byte.txt	16*byte*	2 bit	akhir	19.53
24byte.txt	24*byte*	2 bit	akhir	18.75
32byte.txt	32*byte*	2 bit	akhir	11.72
40byte.txt	40*byte*	2 bit	akhir	10
rata-rata				20.13
rata-rata total				36.48

Dari perubahan 1 bit di awal dari *plaintext* terhadap *chipertext* dengan *key* yang sama diperoleh rata-rata nilai 55.49 kemudian untuk perubahan 1 bit di tengah rata-rata nilai *avalanche effect*nya 34.37, untuk perubahan 1 bit di akhir 21.34.

Kemudian untuk perubahan 2 bit di awal rata-rata nilainya adalah 52.57, di tengah adalah 34,80 dan di akhir adalah 20.13.

Nilai rata-rata total adalah sebesar 36.48, Nilai tersebut masih kurang dari nilai seharusnya yang diharapkan yaitu 45 persen yang merupakan nilai ideal untuk *avalanche effect*.

Kemudian yang kedua yaitu perubahan *byte* dan posisi pada *plaintext* terhadap *chipertext* dengan *key* enkripsi yang sama yaitu ramadhanku dengan karakter pengganti ‘a’ dapat dilihat pada tabel 4.4.

Tabel 4.4 Perubahan *byte* dan posisi pada *plaintext* terhadap *chipertext* dengan *key* enkripsi yang sama (*key*: ramadhanku, karakter pengganti 'a')

Nama *file*	Ukuran *file*	Jumlah perubahan bit/*byte*	Posisi perubahan	*Avalanche effect*
8byte.txt	8 *byte*	1 byte	awal	53.13
16byte.txt	16*byte*	1 byte	awal	51.56
24byte.txt	24*byte*	1 byte	awal	54.69
32byte.txt	32*byte*	1 byte	awal	54.3
40byte.txt	40*byte*	1 byte	awal	50.94

rata-rata				52.92
8byte.txt	8 *byte*	1 byte	tengah	54.69
16byte.txt	16*byte*	1 byte	tengah	28.13
24byte.txt	24*byte*	1 byte	tengah	33.33
32byte.txt	32*byte*	1 byte	tengah	26.95
40byte.txt	40*byte*	1 byte	tengah	31.88
rata -rata				35.00
8byte.txt	8 *byte*	1 byte	akhir	51.56
16byte.txt	16*byte*	1 byte	akhir	25.78
24byte.txt	24*byte*	1 byte	akhir	16.67
32byte.txt	32*byte*	1 byte	akhir	10.94
40byte.txt	40*byte*	1 byte	akhir	11.88
rata-rata				23.37
8byte.txt	8 *byte*	2 byte	awal	56.25
16byte.txt	16*byte*	2 byte	awal	50.78
24byte.txt	24*byte*	2 byte	awal	50
32byte.txt	32*byte*	2 byte	awal	48.05
40byte.txt	40*byte*	2 byte	awal	47.5
rata-rata				50.52
8byte.txt	8 *byte*	2 byte	tengah	50
16byte.txt	16*byte*	2 byte	tengah	25.78
24byte.txt	24*byte*	2 byte	tengah	34.38
32byte.txt	32*byte*	2 byte	tengah	22.27
40byte.txt	40*byte*	2 byte	tengah	31.56
rata -rata				32.80
8byte.txt	8 *byte*	2 byte	akhir	45.31
16byte.txt	16*byte*	2 byte	akhir	26.56
24byte.txt	24*byte*	2 byte	akhir	15.63
32byte.txt	32*byte*	2 byte	akhir	12.5
40byte.txt	40*byte*	2 byte	akhir	11.56
rata-rata				22.31
rata-rata total				36.15

Dari perubahan 1 *byte* di awal dari *plaintext* terhadap *chipertext* dengan *key* yang sama diperoleh rata-rata nilai 52.92 kemudian untuk perubahan 1 *byte* di tengah rata-rata nilai *avalanche effectnya* 35.00, untuk perubahan 1 bit di akhir 23.37.

Kemudian untuk perubahan 2 *byte* di awal rata-rata nilainya adalah 50.52, di tengah adalah 32.80 dan di akhir adalah 22.31.

Nilai rata-rata total adalah sebesar 36.15. Nilai tersebut masih kurang dari nilai seharusnya yang diharapkan yaitu 45 persen yang merupakan nilai ideal untuk *avalanche effect*.

Kemudian yang ketiga yaitu perubahan bit dan posisi pada *key* enkripsi terhadap *chipertext* dengan *plaintext* yang sama yaitu ramadhanku dapat dilihat pada tabel 4.5.

Tabel 4.5 Perubahan bit serta posisinya pada *key* enkripsi terhadap *chipertext* dengan *plaintext* yang sama. (*key*: ramadhanku)

Nama *file*	Ukuran *file*	Jumlah perubahan bit/*byte*	Posisi perubahan	*Avalanche effect*
8byte.txt	8 *byte*	1 bit	awal	46.88
16byte.txt	16*byte*	1 bit	awal	50.78
24byte.txt	24*byte*	1 bit	awal	46.88
32byte.txt	32*byte*	1 bit	awal	47.66
40byte.txt	40*byte*	1 bit	awal	46.56
rata-rata				47.75
8byte.txt	8 *byte*	1 bit	tengah	50
16byte.txt	16*byte*	1 bit	tengah	55.47
24byte.txt	24*byte*	1 bit	tengah	50.52
32byte.txt	32*byte*	1 bit	tengah	53.52
40byte.txt	40*byte*	1 bit	tengah	51.56
rata -rata				52.21
8byte.txt	8 *byte*	1 bit	akhir	56.25
16byte.txt	16*byte*	1 bit	akhir	51.56
24byte.txt	24*byte*	1 bit	akhir	52.08

32byte.txt	32*byte*	1 bit	akhir	50.78
40byte.txt	40*byte*	1 bit	akhir	51.25
rata-rata				52.38
8byte.txt	8 *byte*	2 bit	awal	42.19
16byte.txt	16*byte*	2 bit	awal	53.13
24byte.txt	24*byte*	2 bit	awal	52.08
32byte.txt	32*byte*	2 bit	awal	51.56
40byte.txt	40*byte*	2 bit	awal	50.63
rata-rata				49.92
8byte.txt	8 *byte*	2 bit	tengah	56.25
16byte.txt	16*byte*	2 bit	tengah	52.34
24byte.txt	24*byte*	2 bit	tengah	51.56
32byte.txt	32*byte*	2 bit	tengah	47.27
40byte.txt	40*byte*	2 bit	tengah	47.19
rata -rata				50.92
8byte.txt	8 *byte*	2 bit	akhir	51.56
16byte.txt	16*byte*	2 bit	akhir	46.09
24byte.txt	24*byte*	2 bit	akhir	47.4
32byte.txt	32*byte*	2 bit	akhir	45.7
40byte.txt	40*byte*	2 bit	akhir	46.88
rata-rata				47.53
rata-rata total				50.12

Dari perubahan 1 bit di awal dari *key* terhadap *chipertext* dengan *plaintext* yang sama diperoleh rata-rata nilai 47.75 kemudian untuk perubahan 1 bit di tengah rata-rata nilai *avalanche effectnya* 52.21, untuk perubahan 1 bit di akhir adalah 52.38.

Kemudian untuk perubahan 2 bit di awal rata-rata nilainya adalah 49.92, di tengah adalah 50.92 dan di akhir adalah 47.53.

Nilai rata-rata total adalah sebesar 50.12. Nilai tersebut merupakan nilai ideal untuk *avalanche effect* karena lebih dari 45%.

Kemudian yang keempat yaitu perubahan *byte* dan posisi pada *key* enkripsi terhadap *chipertext* dengan *plaintext* yang sama yaitu

ramadhanku dengan karakter pengganti ‘a’ dapat dilihat pada tabel 4.6.

Tabel 4.6 Perubahan *byte* serta posisinya pada *key* enkripsi terhadap *chipertext* dengan *plaintext* yang sama. (*key:* ramadhanku, karakter pengganti = 'a')

Nama *file*	Ukuran *file*	Jumlah perubahan bit/*byte*	Posisi perubahan	*Avalanche effect*
8byte.txt	8 *byte*	1 byte	awal	46.88
16byte.txt	16*byte*	1 byte	awal	51.56
24byte.txt	24*byte*	1 byte	awal	56.25
32byte.txt	32*byte*	1 byte	awal	55.08
40byte.txt	40*byte*	1 byte	awal	52.81
rata-rata				52.52
8byte.txt	8 *byte*	1 byte	tengah	50
16byte.txt	16*byte*	1 byte	tengah	53.91
24byte.txt	24*byte*	1 byte	tengah	52.6
32byte.txt	32*byte*	1 byte	tengah	51.95
40byte.txt	40*byte*	1 byte	tengah	51.88
rata -rata				52.07
8byte.txt	8 *byte*	1 byte	akhir	46.88
16byte.txt	16*byte*	1 byte	akhir	50.78
24byte.txt	24*byte*	1 byte	akhir	50
32byte.txt	32*byte*	1 byte	akhir	49.61
40byte.txt	40*byte*	1 byte	akhir	48.75
rata-rata				49.20
8byte.txt	8 *byte*	2 byte	awal	46.88
16byte.txt	16*byte*	2 byte	awal	51.56
24byte.txt	24*byte*	2 byte	awal	56.25
32byte.txt	32*byte*	2 byte	awal	55.08
40byte.txt	40*byte*	2 byte	awal	52.81
rata-rata				52.52

8byte.txt	8 *byte*	2 byte	tengah	50
16byte.txt	16*byte*	2 byte	tengah	53.91
24byte.txt	24*byte*	2 byte	tengah	52.6
32byte.txt	32*byte*	2 byte	tengah	51.95
40byte.txt	40*byte*	2 byte	tengah	51.88
rata -rata				52.07
8byte.txt	8 *byte*	2 byte	akhir	34.38
16byte.txt	16*byte*	2 byte	akhir	37.5
24byte.txt	24*byte*	2 byte	akhir	41.15
32byte.txt	32*byte*	2 byte	akhir	44.53
40byte.txt	40*byte*	2 byte	akhir	44.38
rata-rata				40.39
rata-rata total				49.79

Dari perubahan 1 *byte* di awal dari *key* terhadap *chipertext* dengan *plaintext* yang sama diperoleh rata-rata nilai 52.52, kemudian untuk perubahan 1 *byte* di tengah rata-rata nilai *avalanche effectnya* 52.07, untuk perubahan 1 *byte* di akhir adalah 49.20.

Kemudian untuk perubahan 2 *byte* di awal rata-rata nilainya adalah 52.52, di tengah adalah 52.07 dan di akhir adalah 40.39.

Nilai total adalah sebesar 49.79. Nilai tersebut merupakan nilai ideal untuk *avalanche effect* karena lebih dari 45.

Kemudian yang keempat yaitu perubahan bit dan posisi pada *chipertext* terhadap *plainteks* dengan *key* yang sama yaitu ramadhanku dapat dilihat pada tabel 4.7.

Tabel 4.7 Perubahan bit dan posisi pada *chipertext* terhadap *plaintext* dengan *key* dekripsi yang sama (*key*: ramadhanku)

Nama *file*	Ukuran *file*	Jumlah perubahan bit/*byte*	Posisi perubahan	*Avalanche effect*
8byte.txt	8 *byte*	1 bit	awal	54.69
16byte.txt	16*byte*	1 bit	awal	28.13

24byte.txt	24*byte*	1 bit	awal	18.75
32byte.txt	32*byte*	1 bit	awal	14.06
40byte.txt	40*byte*	1 bit	awal	11.25
rata-rata				25.38
8byte.txt	8 *byte*	1 bit	tengah	60.94
16byte.txt	16*byte*	1 bit	tengah	25
24byte.txt	24*byte*	1 bit	tengah	18.23
32byte.txt	32*byte*	1 bit	tengah	12.11
40byte.txt	40*byte*	1 bit	tengah	10
rata -rata				25.26
8byte.txt	8 *byte*	1 bit	akhir	51.56
16byte.txt	16*byte*	1 bit	akhir	21.09
24byte.txt	24*byte*	1 bit	akhir	17.19
32byte.txt	32*byte*	1 bit	akhir	12.89
40byte.txt	40*byte*	1 bit	akhir	10.94
rata-rata				22.73
8byte.txt	8 *byte*	2 bit	awal	42.19
16byte.txt	16*byte*	2 bit	awal	22.66
24byte.txt	24*byte*	2 bit	awal	15.1
32byte.txt	32*byte*	2 bit	awal	11.33
40byte.txt	40*byte*	2 bit	awal	9.06
rata-rata				20.07
8byte.txt	8 *byte*	2 bit	tengah	60.94
16byte.txt	16*byte*	2 bit	tengah	23.44
24byte.txt	24*byte*	2 bit	tengah	13.02
32byte.txt	32*byte*	2 bit	tengah	12.5
40byte.txt	40*byte*	2 bit	tengah	10
rata -rata				23.98
8byte.txt	8 *byte*	2 bit	akhir	51.56
16byte.txt	16*byte*	2 bit	akhir	24.22
24byte.txt	24*byte*	2 bit	akhir	15.1
32byte.txt	32*byte*	2 bit	akhir	11.72

40byte.txt	40*byte*	2 bit	akhir	11.88
rata-rata				22.90
rata-rata total				23.39

Dari perubahan 1 bit di awal dari *chipertext* terhadap *plaintext* dengan *key* yang sama diperoleh rata-rata nilai 25.38 kemudian untuk perubahan 1 bit di tengah rata-rata nilai *avalanche effectnya* 25.26, untuk perubahan 1 bit di akhir 22.73.

Kemudian untuk perubahan 2 bit di awal rata-rata nilainya adalah 20.07, di tengah adalah 23.98 dan di akhir adalah 22.90.

Nilai rata-rata total adalah sebesar 23.39. Nilai tersebut masih kurang dari nilai seharusnya yang diharapkan yaitu 45 persen yang merupakan nilai ideal untuk *avalanche effect.*

Kemudian yang keempat yaitu perubahan *byte* dan posisi pada *chipertext* terhadap *plainteks* dengan *key* yang sama yaitu ramadhanku dengan karakter pengganti ‘a’ dapat dilihat pada tabel 4.8.

Tabel 4.8 Perubahan *byte* dan posisi pada *chipertext* terhadap *plainteks* dengan *key* dekripsi yang sama (*key*: ramadhanku, karakter pengganti 'a')

Nama *file*	Ukuran *file*	Jumlah perubahan bit/*byte*	Posisi perubahan	*Avalanche effect*
8byte.txt	8 *byte*	1 byte	awal	39.06
16byte.txt	16*byte*	1 byte	awal	22.66
24byte.txt	24*byte*	1 byte	awal	15.1
32byte.txt	32*byte*	1 byte	awal	11.33
40byte.txt	40*byte*	1 byte	awal	9.06
rata-rata				19.44
8byte.txt	8 *byte*	1 byte	tengah	54.69
16byte.txt	16*byte*	1 byte	tengah	21.88
24byte.txt	24*byte*	1 byte	tengah	20.83
32byte.txt	32*byte*	1 byte	tengah	12.11
40byte.txt	40*byte*	1 byte	tengah	8.44

rata -rata				23.59
8byte.txt	8 *byte*	1 byte	akhir	56.25
16byte.txt	16*byte*	1 byte	akhir	22.66
24byte.txt	24*byte*	1 byte	akhir	15.1
32byte.txt	32*byte*	1 byte	akhir	12.5
40byte.txt	40*byte*	1 byte	akhir	10.94
rata-rata				23.49
8byte.txt	8 *byte*	2 byte	awal	57.81
16byte.txt	16*byte*	2 byte	awal	33.59
24byte.txt	24*byte*	2 byte	awal	22.4
32byte.txt	32*byte*	2 byte	awal	16.8
40byte.txt	40*byte*	2 byte	awal	13.44
rata-rata				28.81
8byte.txt	8 *byte*	2 byte	tengah	53.13
16byte.txt	16*byte*	2 byte	tengah	21.09
24byte.txt	24*byte*	2 byte	tengah	18.75
32byte.txt	32*byte*	2 byte	tengah	13.67
40byte.txt	40*byte*	2 byte	tengah	10.31
rata -rata				23.39
8byte.txt	8 *byte*	2 byte	akhir	42.19
16byte.txt	16*byte*	2 byte	akhir	27.34
24byte.txt	24*byte*	2 byte	akhir	15.63
32byte.txt	32*byte*	2 byte	akhir	15.23
40byte.txt	40*byte*	2 byte	akhir	9.06
rata-rata				21.89
rata-rata total				23.44

Dari perubahan 1 *byte* di awal dari *chipertext* terhadap *plaintext* dengan *key* yang sama diperoleh rata-rata nilai 19.44, kemudian untuk perubahan 1 *byte* di tengah rata-rata nilai *avalanche effect*nya 23.59, untuk perubahan 1 *byte* di akhir 23.49.

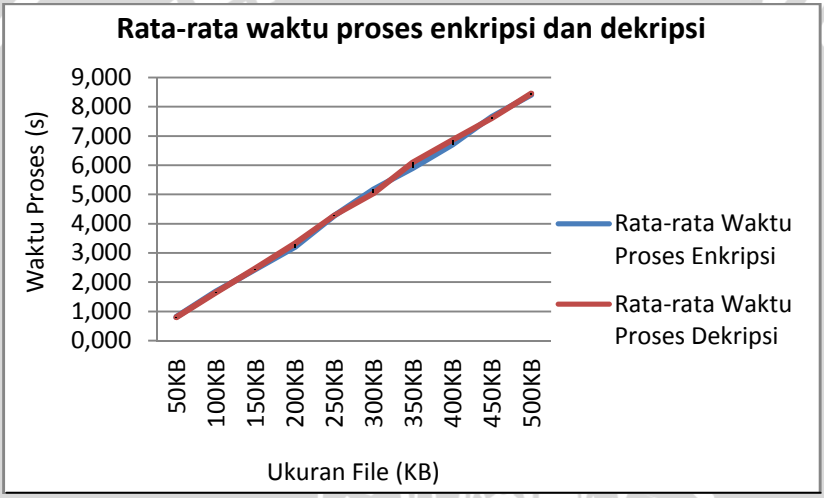
Kemudian untuk perubahan 2 *byte* di awal rata-rata nilainya adalah 28.81, di tengah adalah 23.39 dan di akhir adalah 21.89.

Nilai rata-rata total adalah sebesar 23.44. Nilai tersebut masih kurang dari nilai seharusnya yang diharapkan yaitu 45 persen yang merupakan nilai ideal untuk *avalanche effect.*

4.5 Analisis hasil

Pada subbab ini akan dilakukan analisis terhadap hasil uji yang didapatkan pada subbab 4.4.

Pertama akan dilakukan analisis pada hasil uji coba waktu proses, dapat dilihat pada grafik 4.1.

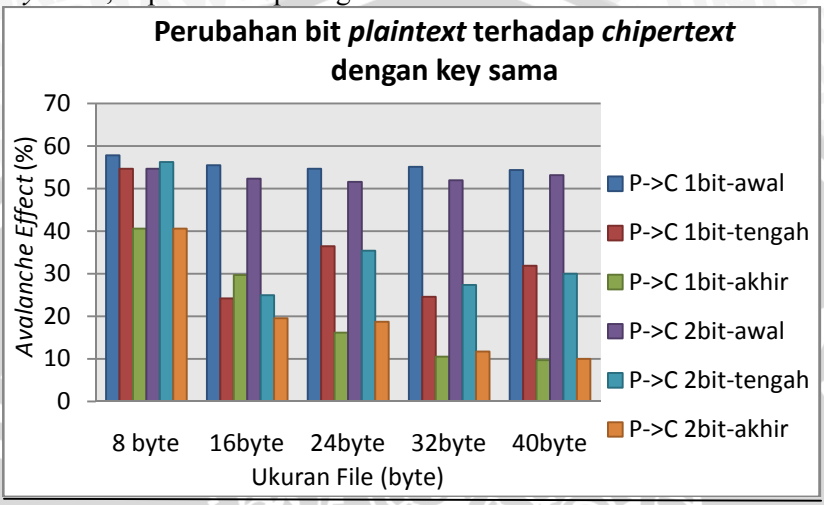


Grafik 4.1 Rata-rata waktu proses enkripsi dan dekripsi

Grafik rata-rata waktu dekripsi berada agak sejajar dengan grafik rata-rata waktu enkripsi. Hal ini mengindikasikan bahwa rata-rata waktu dekripsi hampir sama dengan rata-rata waktu enkripsi.

Kemudian yang kedua akan dilakukan analisis pada hasil uji *avalanche effect*, pada pembuatan grafik dari hasil uji *avalanche effect* didapatkan 6 buah grafik (dapat dilihat pada lampiran) dimana terdapat 2 bentuk grafik yang unik. Yaitu bentuk yang cenderung menurun dan bentuk yang cenderung naik turun di atas angka 40. Bentuk yang cenderung menurun dimiliki oleh grafik perubahan *plaintext* terhadap *chipertext* dan perubahan *chipertext* terhadap *plaintext* sedangkan bentuk yang kedua dimiliki oleh grafik perubahan *key* terhadap *chipertext.*

Kemudian akan dilakukan analisis pada hasil perhitungan nilai *avalanche effect* perubahan bit plaintext terhadap chipertext dengan *key* sama, dapat dilihat pada grafik 4.2.



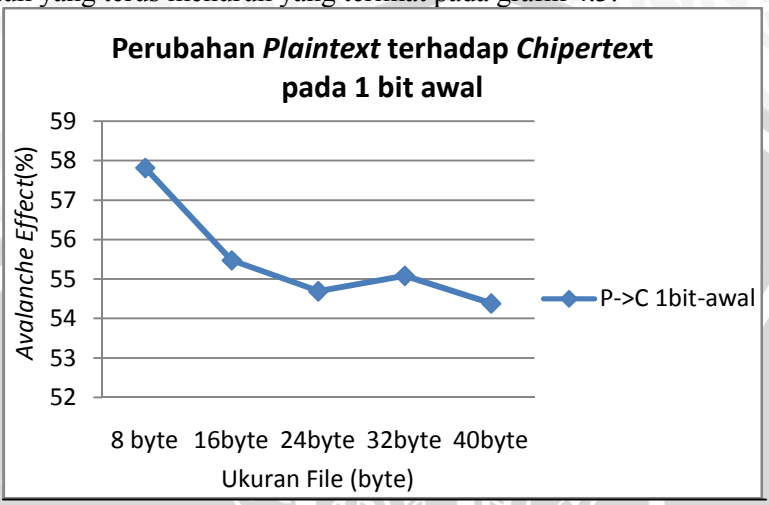
Grafik 4.2 perubahan bit *plaintext* terhadap *chipertext*

Pada grafik perubahan bit *plaintext* terhadap *chipertext* (dengan pengecualian pada grafik kelima pada tiap ukuran *file*). dapat dilihat bentuk grafik pada umumnya adalah menurun dan nilai maksimum adalah pada ukuran 8 *byte*, hal ini dikarenakan algoritma *CAST-128* mengolah *plaintext* per blok yang berisi 64 bit atau 8 *byte*. Sehingga apabila dilakukan perubahan pada satu bit pada blok tertentu maka hanya isi blok tersebut yang berubah, sedangkan blok lainnya tidak berubah sehingga nilai *avalanche effect* pada ukuran *file* yang besar cenderung lebih kecil.

Contohnya saja pada perubahan *plaintext* terhadap *chipertext* pada satu bit awal. Perubahan pada *chipertext* akan diilustrasikan sebagai berikut(yang dicetak tebal menunjukkan blok yang berubah):

8 *byte* = **blok1**
16 *byte* = **blok1**-blok2
24 *byte* = **blok1**-blok2-blok3
32 *byte* = **blok1**-blok2-blok3-blok4
40 *byte* = **blok1**-blok2-blok3-blok4-blok5

Dari ilustrasi tersebut tentu saja dapat diperoleh jawaban dari bentuk yang terus menurun yang terlihat pada grafik 4.3.



Grafik 4.3 Perubahan *Plaintext* terhadap *Chipertex*t pada 1 bit awal

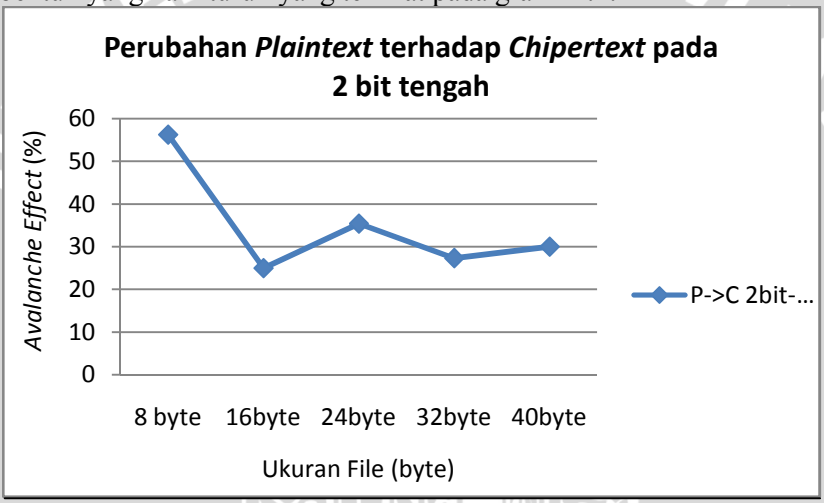
Akan tetapi hal itu tidak terjadi pada perubahan 2 bit pada posisi tengah (grafik perubahan bit *plaintext* terhadap *chipertext* grafik kelima pada tiap ukuran *file*), bentuk grafiknya cenderung naik turun, bisa dilihat pada grafik di atas pada ukuran 8 dan 16 *byte* grafiknya akan sama, kemudian pada 24 *byte* turun lalu pada ukuran 32 *byte* naik lagi dan pada 40 *byte* turun lagi. Hal ini disebabkan pada perubahan 2 bit pada posisi tengah pada *file* yang mempunyai blok genap akan melibatkan 2 blok dan otomatis 2 blok tadi akan berubah. Sedangkan pada *file* yang memiliki blok ganjil misalnya 24 *bytes* yang memiliki 3 blok hanya akan melibatkan 1 blok saja yang berubah.

Misalnya saja pada grafik perubahan bit *plaintext* terhadap *chipertext* (grafik kelima pada tiap ukuran *file*), pada ukuran 16 *byte* 2 bit tengah yang akan diubah yaitu bit terakhir dari blok pertama dan bit pertama dari blok ke dua, maka blok delapan dan sembilan ini akan berubah saat dilakukan enkripsi, beda halnya dengan *file* yang berukuran 24 *byte*. Karena terdiri dari 3 blok, tentu saja 2 bit yang akan diubah akan berada pada blok yang di tengah yaitu blok kedua, sehingga perubahan hanya terjadi pada blok ke dua saja.

Contohnya saja pada perubahan *plaintext* terhadap *chipertext* pada dua bit tengah. Perubahan pada *chipertext* akan diilustrasikan sebagai berikut(yang dicetak tebal menunjukkan blok yang berubah):

8 *byte* = **blok1**
16*byte* = **blok1-blok2**
24*byte* = blok1-**blok2**-blok3
32*byte* = blok1-**blok2-blok3**-blok4
40*byte* = blok1-blok2-**blok3**-blok4-blok5

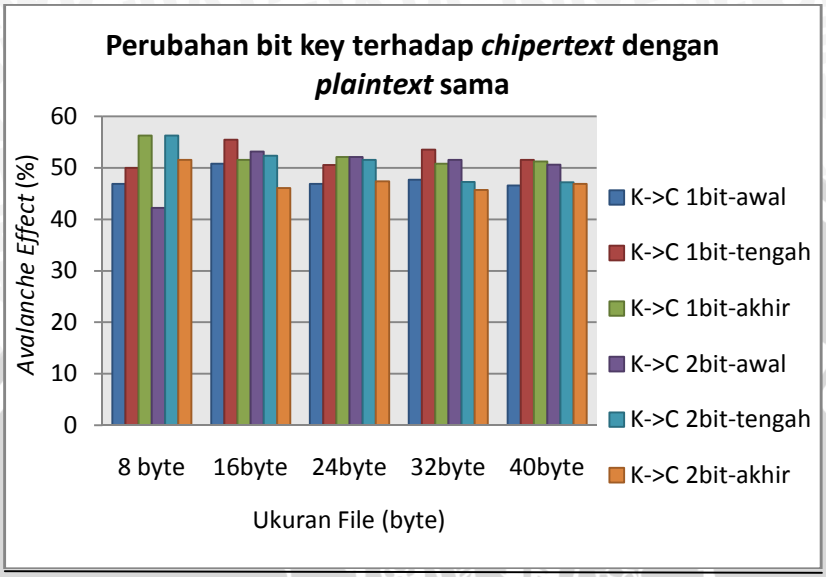
Dari ilustrasi tersebut tentu saja dapat diperoleh jawaban dari bentuk yang naik turun yang terlihat pada grafik 4.4.



Grafik 4.4 Perubahan *Plaintext* terhadap *Chipertext* pada 2 bit tengah

Bentuk grafik selanjutnya adalah bentuk grafik yang cenderung naik turun. Grafik ini dimiliki oleh perubahan *key* terhadap *chipertext* baik itu perubahan bit maupun perubahan *byte*.

Hal ini terjadi karena perubahan *key* terhadap *chipertext* tidak langsung berpengaruh pada blok sebuah *plaintext*. Akan tetapi berpengaruh pada kunci *masking* dan kunci rotasi pada jaringan *feistel*. Sehingga pada ukuran besar nilai *avalanche effect* tidak semakin kecil seperti grafik grafik perubahan bit *plaintext* terhadap *chipertext*. Hal ini dapat dilihat pada perubahan bit *key* terhadap *chipertext*. Dan hal tersebut dapat dilihat pada grafik 4.5.



Grafik 4.5 Perubahan bit *key* terhadap *chipertext*