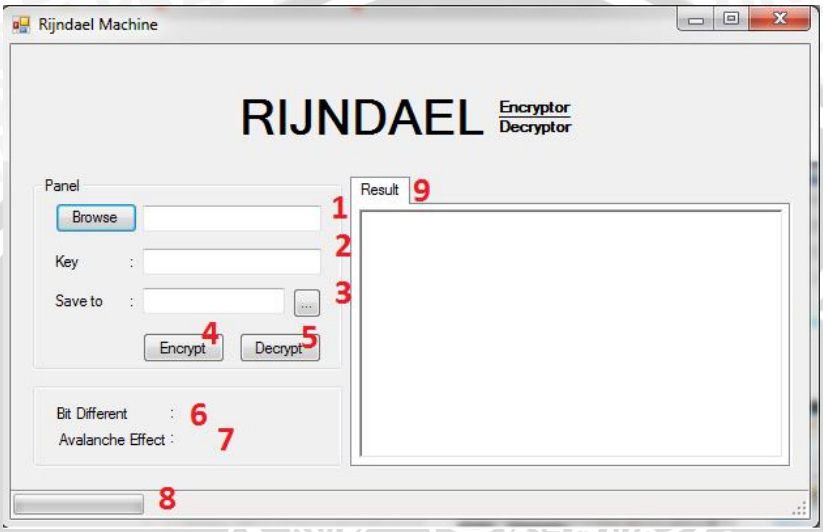
BAB 4 PERANCANGAN DAN IMPLENTASI

Perancangan Antarmuka

Pada sistem dibuat antarmuka dengan menggunakan program visual studio 2010 dengan bahasa C#.Pada antarmuka program terdapat beberapa bagian seperti yang ditunjukkan pada gambar 3.31.



Gambar 4.1 Perancangan Antarmuka

Pada gambar **4.1** terdapat 10 bagian pada antarmuka program dengan fungsi yang berbeda.Berikut adalah penjelasan berdasarkan gambar **4.1**.

1. Tombol *browse*, digunakan untuk mengambil file yang akan dienkripsi atau didekripsi.
2. *Key*, digunakan untuk memasukkan *key* untuk enkripsi maupun dekripsi.
3. *Saveto*, digunakan untuk mengarahkan kemana file hasil enkripsi atau dekripsi disimpan.
4. Tombol *encrypt*, digunakan untuk memulai proses enkripsi.
5. Tombol *decrypt*, digunakan untuk memulai proses dekripsi.
6. *Bit Different*, jumlah bit yang berbeda.
7. *Avalanche Effect,* digunakan untuk mengetahui hasil pengujian *avalanche effect*.
8. *Statusbar*, digunakan untuk menampilkan progress dari enkripsi, dekripsi, dan *bruteforce*.
9. Tab *Result*, digunakan untuk menampilkan hasil enkripsi maupun dekripsi.

Pada bab ini dilakukan implementasi algoritma rijndael untuk kriptografi sebuah teks berserta analisis perangkat lunaknya.

1.1 Lingkungan Implementasi

Pada lingkungan implementasi, terdapat dua factor yang mempengaruhi yaitu lingkungan perangkat keras dan lingkungan perangkat lunak.

1.1.1 Lingkungan Perangkat Keras

Perangkat keras yang digunakan untuk implementasi algoritma rijndael memiliki spesifikasi sebagai berikut:

1. Processor Intel Core i3 2120 CPU 3.30GHz
2. Memory 4 GB
3. Hardisk 500 GB

1.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan untuk mengembangkan kriptografi dengan menggunakan metode rijndael adalah sebagai berikut:

1. Sistem operasi Microsoft Windows 7 Ultimate 64-Bit
2. Microsoft Visual Studio C# 2010 Express Edition

1.2 Implementasi Program

Pada applikasi rijndael dibagi menjadi dua sub proses yakni enkrip, dekrip, dan .Kelas yang dipakai dalam applikasi rijndael ada satu yakni kelas Main. Penjelasan kelas-kelasnya dapat dilihat pada table **4.1**.

Tabel 4.1 Kelas pada applikasi rijndael

Nama Kelas	Penjelasan
Main (inherit Form)	Merupakan kelas utama dari applikasi Rijndael yang berisikan proses enkrip dan dekrip.

1.2.2 Implementasi Enkripsi pada Rijndael

Proses enkripsi pada applikasi rijndael terdiri dari beberapa method yang mendukung didalamnya. Method pada proses enkripsi dapat dilihat pada tabel 4.2.

Tabel 4.2 Method pada proses Enkripsi

Method	Penjelasan
privateint[,] addRoundKey(int[,] arrayState, int[,] chiperKey)	Menambahkan round key pada array state. Parameter yang digunakan: - Input : ArrayState (Array Int) dan ChiperKey (Array Int) - Return : Hasil AddRoundKey (Array Int)
privateint[,] subBytes(int[,] arrayState)	Method untuk proses subbytes Parameter yang digunakan: - Input : ArrayState (Array Int) - Return : Hasil SubBytes

	(Array Int)
privateint SubByte(int a)	Method untuk mendapatkan value berdasarkan sbox. Parameter yang digunakan: - Input : Value (Int) - Return : Hasil Sbox (Int)
privateint[,] shiftRows(int[,] arrayState)	Method untuk proses ShiftRows. Parameter yang digunakan: - Input : ArrayState (Array Int) - Return : Hasil ShiftRows (Array Int)
privateint[,] mixColumns(int[,] arrayState)	Method untuk proses MixColumns. Parameter yang digunakan: - Input : ArrayState (Array Int) - Return : Hasil MixColumns (Array Int)
privateint[,] keySchedule(int[,] chiperKey, int round)	Method yang digunakan untuk mengenerate chiperKey menjadi deretan KeySchedule. Parameter yang digunakan: - Input : ChiperKey (Array Int) dan Round (Int) - Return : Hasil

	KeySchedule (Array Int)
privateint[,] Encrypt(int[,] arrayState, int[,] chiperKey)	Method utama yang digunakan untuk proses enkripsi. Parameter yang digunakan: - Input : ArrayState (Array Int) dan ChiperKey (Array Int) - Return : Hasil Enkripsi (Array Int)

Method yang digunakan pada proses enkripsi pada rijndael merupakan step by step dari algoritma enkripsi rijndael, setiap method atau step terdapat sourcecode didalamnya. Sourcode untuk method add RoundKey dapat dilihat pada sourcecode 4.1.

```
privateint[,] addRoundKey(int[,] arrayState, int[,] chiperKey)
{
int[,] rslt_addRoundKey = newint[4,4];
for (int i = 0; i < 4; i++)
   {
for (int j = 0; j < 4; j++)
      {
         rslt_addRoundKey[i,j] = arrayState[i,j] ^
chiperKey[i,j];
      }
   }
return rslt_addRoundKey;
}
```

Sourcecode 4.1 addRoundKey

Pada sourcecode 4.1 dilakukan perulangan untuk menghitung XOR dari ArrayState dengan ChiperKey.Sourcecode untuk method subBytes dapat dilihat pada sourcecode 4.2.

```
int[,] rslt_subBytes = newint[4,4];
for (int i = 0; i < 4; i++)
{
for (int j = 0; j < 4; j++)
  {
    rslt_subBytes[i, j] = SubByte(arrayState[i, j]) -
1667457792;
  }
}
return rslt_subBytes
```

Sourcecode 4.2 subBytes

Pada sourcecode 4.2 dilakukan perulangan untuk mengambil hasil dari proses SubByte ke dalam ArrayState. Sourcecode untuk method SubByte dapat dilihat pada sourcecode 4.3.

```
Int32 value = 0xff & a;
Int32 result = sbox[value];
value = 0xff & (a >> 8);
result |= sbox[value] << 8;
value = 0xff & (a >> 16);
result |= sbox[value] << 16;
value = 0xff & (a >> 24);
return result | (sbox[value] << 24);
```

Sourcecode 4.3 SubByte

Pada sourcecode 4.3 dilakukan pengambilan nilai dari sbox.Inputan dijadikan sebagai indeks untuk mengambil nilai dari sbox sesuai dengan indeksnya.Sourcecode untuk method shiftRows dapat dilihat pada sourcecode 4.4.

```
int[,] rslt_shiftRows = newint[4,4];
for (int i = 0; i < 4; i++)
{
for (int j = 0; j < 4; j++)
   {
if ((j + i) > 3)
      {
         rslt_shiftRows[i, j] = arrayState[i, (Math.Abs(j + i)) -
4];
      }
else
      {
         rslt_shiftRows[i, j] = arrayState[i, (j + i)];
      }
   }
}
return rslt_shiftRows;
```

Sourcecode 4.4 shiftRows

Pada sourcecode 4.4 dilakukan perulangan untuk mendapatkan hasil dari pergeseran indeks pada ArrayState. ArrayState digeser sebesar i, apabila hasil indeks hasil pergeseran lebih besar 3 maka indeksnya harus dikurangi sebesar 4 blok. Sourcecode untuk method mixColumns dapat dilihat pada sourcecode 4.5.

```
byte[,] temp = newbyte[4, 4];
for (int i = 0; i < 4; ++i)
{
for (int j = 0; j < 4; ++j)
   {
```

```
        temp[i, j] = (byte)arrayState[i, j];
    }
}

for (int c = 0; c < 4; ++c)
{
    arrayState[0, c] = (byte)((int)multiplyBy2(temp[0, c]) ^
(int)multiplyBy3(temp[1, c]) ^
                    (int)multiplyBy1(temp[2, c]) ^
(int)multiplyBy1(temp[3, c]));
    arrayState[1, c] = (byte)((int)multiplyBy1(temp[0, c]) ^
(int)multiplyBy2(temp[1, c]) ^
                    (int)multiplyBy3(temp[2, c]) ^
(int)multiplyBy1(temp[3, c]));
    arrayState[2, c] = (byte)((int)multiplyBy1(temp[0, c]) ^
(int)multiplyBy1(temp[1, c]) ^
                    (int)multiplyBy2(temp[2, c]) ^
(int)multiplyBy3(temp[3, c]));
    arrayState[3, c] = (byte)((int)multiplyBy3(temp[0, c]) ^
(int)multiplyBy1(temp[1, c]) ^
                    (int)multiplyBy1(temp[2, c]) ^
(int)multiplyBy2(temp[3, c]));
}
return arrayState;
```

Sourcecode 4.5 mixColumns

Pada sourcecode 4.5 dilakukan perulangan untuk mendapatkan perhitungan multiply untuk tiap blok pada ArrayState. Sourcecode untuk method keySchedule dapat dilihat pada tabel 4.6.

```
int[,] newChiperKey = newint[4,4];
```

```
int[,] temp = newint[4,4];

for (int i = 0; i < 4; ++i)
{
if (i == 0)
   {
for (int j = 0; j < 4; j++)
      {
if ((j + 1) > 3)
         {
            temp[j, 0] = chiperKey[0, 3];
         }
else
         {
            temp[j, 0] = chiperKey[(j + 1), 3];
         }
      }
for (int k = 0; k < 4; k++)
      {
         temp[k, 0] = SubByte(temp[k, 0]) - 1667457792;
      }
for (int l = 0; l < 4; l++)
      {
         newChiperKey[l, 0] = chiperKey[l, 0] ^ temp[l, 0] ^
rcon[round, l];
      }
   }
else
   {
for (int j = 0; j < 4; j++)
      {
```

```
            newChiperKey[j, i] = chiperKey[j, i] ^ newChiperKey[j,
(i - 1)];
        }
    }
}
return newChiperKey;
```

Sourcecode 4.6 keySchedule

Pada sourcecode 4.6 dilakukan perulangan untuk menghitung key schedule selanjutnya. Terdapat dua kondisi jika blok paling atas (i=0), maka blok paling kanan dimasukkan ke dalam temp kemudian perhitungan subByte dan perhitungan rcon. Kondisi yang kedua hanya perhitungan rcon, untuk perhitungan rcon sendiri disesuaikan dengan round ke-n. Sourcecode pada method utama Encrypt dapat dilihat pada sourcecode 4.7.

```
int[,] rslt_subBytes;
int[,] rslt_shiftRows;
int[,] newKeySchedule;

int[,] rslt_addRoundKey = addRoundKey(arrayState,
chiperKey);

for (int round = 0; round < 9; round++)
{
    rslt_subBytes = subBytes(rslt_addRoundKey);
    rslt_shiftRows = shiftRows(rslt_subBytes);
int[,] rslt_mixColumns = mixColumns(rslt_shiftRows);
    newKeySchedule = chiperKey;
for (int key = 0; key <= round; key++)
    {
        newKeySchedule = keySchedule(newKeySchedule, key);
    }
```

```
    rslt_addRoundKey = addRoundKey(rslt_mixColumns,
newKeySchedule);
    ProgressBar.Value += 10;
}

rslt_subBytes = subBytes(rslt_addRoundKey);
rslt_shiftRows = shiftRows(rslt_subBytes);
newKeySchedule = chiperKey;

for (int key = 0; key < 10; key++)
{
    newKeySchedule = keySchedule(newKeySchedule, key);
}
rslt_addRoundKey = addRoundKey(rslt_shiftRows,
newKeySchedule);
ProgressBar.Value += 10;
return rslt_addRoundKey;
```

Sourcecode 4.7 Encrypt

Pada sourcecode 4.7 dilakukan beberapa pemanggilan method. Pertama kali memanggil addRoundKey, kemudian diikuti dengan perulangan untuk ronde 1sampai dengan 9. Setiap ronde memanggil method subBytes, shiftRows, mixColumns, dan addRoundKey. Langkah terakhir yakni ronde ke-10 memanggil method subBytes, shiftRows dan addRoundKey.

1.2.3 Implementasi Dekripsi pada Rijndael

Proses dekripsi pada applikasi rijndael terdiri dari beberapa method yang mendukung didalamnya. Method pada proses dekripsi dapat dilihat pada tabel 4.3.

Tabel 4.3 Method pada proses Dekripsi

Method	Penjelasan
privateint[,] addRoundKey(int[,] arrayState, int[,] chiperKey)	Menambahkan round key pada array state. Parameter yang digunakan: - Input : ArrayState (Array Int) dan ChiperKey (Array Int) - Return : Hasil AddRoundKey (Array Int)
privateint[,] InvShiftRows(int[,] arrayState)	Method yang digunakan untuk proses Inverse ShiftRows. Parameter yang digunakan: - Input : ArrayState (Array Int) - Return : Hasil Inverse Shift Rows (Array Int)
privateint[,] InvSubBytes(int[,] arrayState)	Method untuk proses inverse subbytes berdasarkan sbox. Parameter yang digunakan: - Input : ArrayState (Array Int) - Return : Hasil Inverse SubBytes (Array Int)
privateint[,] InvMixColumns(int[,] arrayState)	Method untuk proses Inverse MixColumns. Parameter yang digunakan:

	- Input : ArrayState (Array Int) - Return : Hasil Inverse MixColumns (Array Int)
privateint[,] keySchedule(int[,] chiperKey, int round)	Method yang digunakan untuk mengenerate chiperKey menjadi deretan KeySchedule. Parameter yang digunakan: - Input : ChiperKey (Array Int) dan Round (Int) - Return : Hasil KeySchedule (Array Int)
privateint[,] Decrypt (int[,] arrayState, int[,] chiperKey)	Method utama yang digunakan untuk proses dekripsi. Parameter yang digunakan: - Input : ArrayState (Array Int) dan ChiperKey (Array Int) - Return : Hasil Dekripsi (Array Int)

Method yang digunakan pada proses dekripsi pada rijndael merupakan step by step dari algoritma dekripsi rijndael, setiap method atau step terdapat sourcecode didalamnya. Sourcode untuk method InvShiftRows dapat dilihat pada sourcecode 4.8.

```
int[,] rslt_InvShiftRows = newint[4, 4];
for (int i = 0; i < 4; i++)
```

```
{
for (int j = 0; j < 4; j++)
   {
if ((j - i) < 0)
      {
         rslt_InvShiftRows[i, j] = arrayState[i, (j - i) + 4];
      }
else
      {
         rslt_InvShiftRows[i, j] = arrayState[i, (j - i)];
      }
   }
}
return rslt_InvShiftRows;
```

Sourcecode 4.8 InvShiftRows

Pada sourcecode 4.8 dilakukan perulangan untuk mendapatkan hasil dari pergeseran indeks pada ArrayState. ArrayState digeser sebesar i, apabila hasil indeks hasil pergeseran adalah minus maka indeksnya harus ditambahkan sebesar 4 blok. Sourcecode untuk method InvSubBytes dapat dilihat pada sourcecode 4.9.

```
int[,] rslt_InvSubBytes = newint[4, 4];
for (int i = 0; i < 4; i++)
{
for (int j = 0; j < 4; j++)
   {
for (int x = 0; x < 256; x++)
      {
if (arrayState[i,j]==sbox[x]){
            rslt_InvSubBytes[i, j] = x;
         }
```

```
        }

    }
}
return rslt_InvSubBytes;
```

Sourcecode 4.9 InvSubBytes

Pada sourcecode 4.9 dilakukan perulangan untuk mendapatkan indeks dari nilai sbox sesuai dengan ArrayState. Sourcecode untuk method InvMixColumns dapat diliahat pada sourcecode 4.10.

```
byte[,] temp = newbyte[4, 4];
for (int i = 0; i < 4; ++i)
{
for (int j = 0; j < 4; ++j)
    {
        temp[i, j] = (byte)arrayState[i, j];
    }
}

for (int c = 0; c < 4; ++c)
{
    arrayState[0, c] = (byte)((int)multiplyBy14(temp[0, c]) ^
(int)multiplyBy11(temp[1, c]) ^
                        (int)multiplyBy13(temp[2, c]) ^
(int)multiplyBy9(temp[3, c]));
    arrayState[1, c] = (byte)((int)multiplyBy9(temp[0, c]) ^
(int)multiplyBy14(temp[1, c]) ^
                        (int)multiplyBy11(temp[2, c]) ^
(int)multiplyBy13(temp[3, c]));
    arrayState[2, c] = (byte)((int)multiplyBy13(temp[0, c]) ^
```

```
(int)multiplyBy9(temp[1, c]) ^
                    (int)multiplyBy14(temp[2, c]) ^
(int)multiplyBy11(temp[3, c]));
   arrayState[3, c] = (byte)((int)multiplyBy11(temp[0, c]) ^
(int)multiplyBy13(temp[1, c]) ^
                    (int)multiplyBy9(temp[2, c]) ^
(int)multiplyBy14(temp[3, c]));
}
return arrayState;
```

Sourcecode 4.10 InvMixColumns

Pada sourcecode 4.10 dilakukan perulangan untuk mendapatkan hasil perhitungan multiply inverse kedalam ArrayState. Sourcecode untuk method utama Decrypt dapat dilihat pada sourcecode 4.11.

```
int[,] rslt_invShiftRows;
int[,] rslt_InvSubBytes;
int[,] newKeySchedule;

newKeySchedule = chiperKey;
for (int key = 0; key < 10; key++)
{
   newKeySchedule = keySchedule(newKeySchedule, key);
}

int[,] rslt_addRoundKey = addRoundKey(arrayState,
newKeySchedule);

for (int round = 9; round > 0; round--)
{
   rslt_invShiftRows = InvShiftRows(rslt_addRoundKey);
```

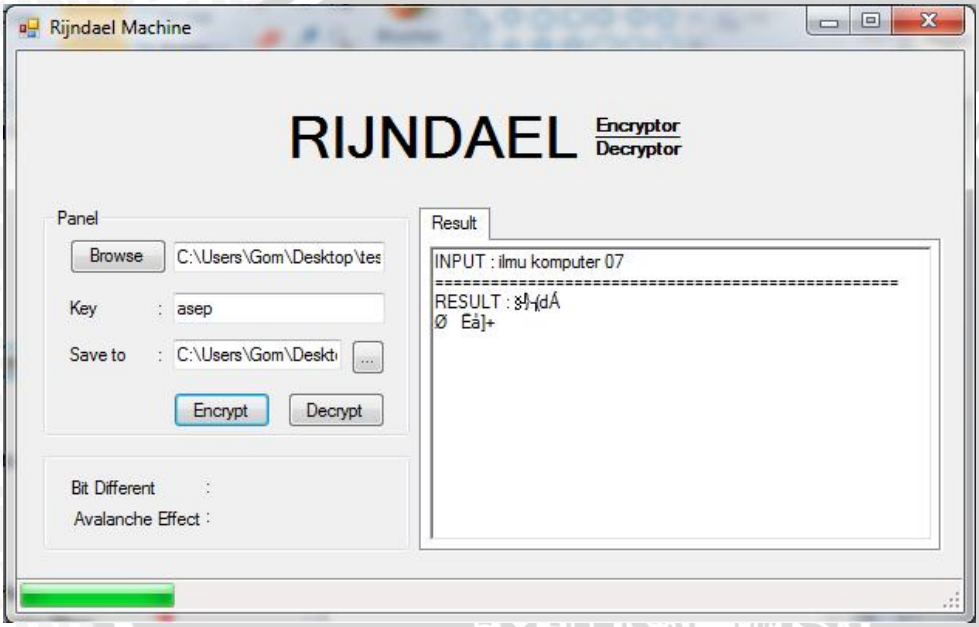
```
    rslt_InvSubBytes = InvSubBytes(rslt_invShiftRows);
    newKeySchedule = chiperKey;
for (int key = 0; key < round; key++)
    {
        newKeySchedule = keySchedule(newKeySchedule, key);
    }
    rslt_addRoundKey = addRoundKey(rslt_InvSubBytes,
newKeySchedule);
int[,] rslt_invMixColumns =
InvMixColumns(rslt_addRoundKey);
    ProgressBar.Value += 10;
}

rslt_invShiftRows = InvShiftRows(rslt_addRoundKey);
rslt_InvSubBytes = InvSubBytes(rslt_invShiftRows);
newKeySchedule = chiperKey;
for (int key = 0; key < 0; key++)
{
    newKeySchedule = keySchedule(newKeySchedule, key);
}
rslt_addRoundKey = addRoundKey(rslt_InvSubBytes,
newKeySchedule);
ProgressBar.Value += 10;
return rslt_addRoundKey;
```

Sourcecode 4.11 Decrypt

Pada sourcecode 4.11 dilakukan beberapa pemanggilan method. Pertama kali memanggil addRoundKey, kemudian diikuti dengan perulangan untuk ronde 1sampai dengan 9. Setiap *round* memanggil method InvShiftRows, InvSubBytes, addRoundKey, dan InvMixColumns. Langkah terakhir yakni *round* ke-10 memanggil method InvShiftRows, InvSubBytes dan addRoundKey.

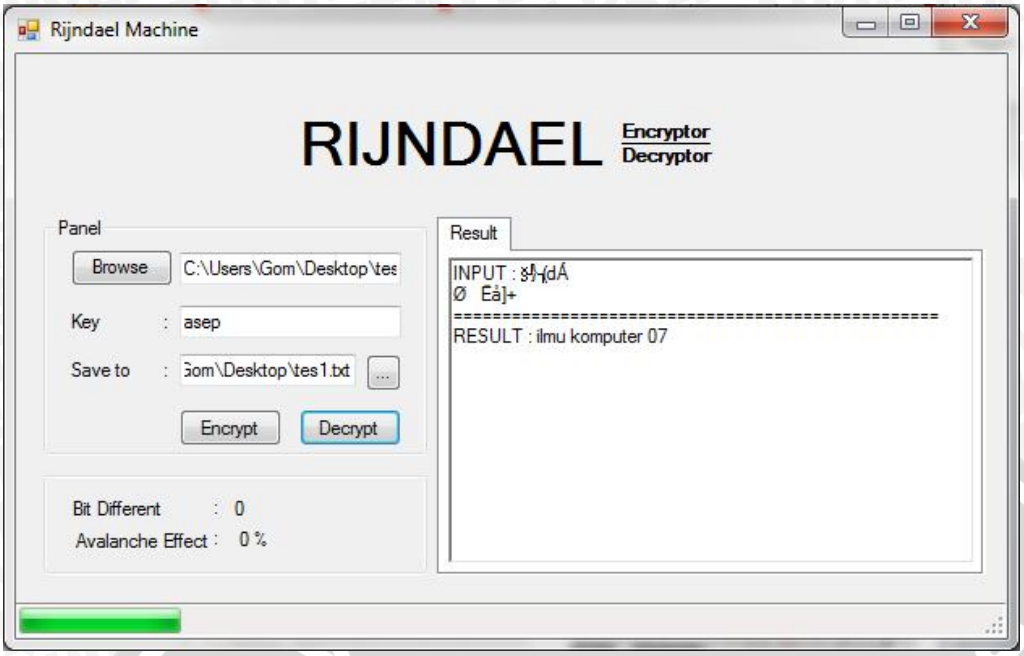
1.3 Penerapan Applikasi

Berdasarkan perancangan antarmuka pada subbab 3.4, terdapat dua subproses pada applikasi rijndael ini yakni enkrip dan dekrip yang masing-masing diaplikasikan dalam sebuah button untuk memulai proses tersebut. Untuk memulai enkripsi, hal pertama yang harus dilakukan adalah menentukan input file teks pada button browse kemudian menentukan key dan menentukan output file untuk hasil enkripsinya. Jika semuanya telah diset maka proses enkripsi bisa dijalankan dengan menekan tombol Encrypt, hasilnya akan ditampilkan pada layar result dan file teks. Hasil proses enkripsi dapat dilihat pada gambar 4.1



Gambar 4.1 Antarmuka Proses Enkripsi

Untuk memulai proses dekrip, hal pertama yang harus dilakukan adalah menentukan input file teks pada button browse kemudian menentukan key dan menentukan output file untuk hasil dekripsinya. Jika semuanya telah diset maka proses dekripsi bisa dijalankan dengan menekan tombol Decrypt, hasilnya akan ditampilkan pada layar result dan file teks. Hasil proses dekripsi dapat dilihat pada gambar 4.2.



Gambar 4.2 Antarmuka Proses Dekripsi