**ENCRYPTION ALGORITHM**

Algorithm DNAEncrypt (P,Key)

P is the plaintext which was read from textfile and Key is the Keyword

**Step 1:** Processing the Keyword. That is removing the duplicates from the Keyword and then map the characters into DNA ASCII Table.

**Step 2:** Arrange the above output into DNA Playfair Keymatrix.

**Step 3:** Convert the plaintext characters into DNA Bases.

**Step 4:** Apply playfair technique on these bases.

**Step 5:** After applying the resultant bases are Cipher text.

**DECRYPTION ALGORITHM**

**Step 1:** Read Encrypt data from the Text File and Key data.

**Step 2:** Split the data into 4 string.

**Step 3:** Mapped to the Key DNA Matrix and Apply “Play Fair”.

**Step 4:** Get Decrypted Data after applying play fair.

**Step 5:** Save the decrypted data as Text File.

**CODING**

1. **Generate Codan Matrix**

private void generate\_dna\_sequence\_data\_upto\_255()

{

char[] data = { 'A', 'C', 'G', 'T' };

int index = 0;

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

for (int k = 0; k < 4; k++)

for (int m = 0; m < 4; m++)

{

String a = data[i].ToString() + data[j].ToString() + data[k].ToString() + data[m].ToString();

ascii\_dna\_code.Add(a);

index++;

}

for (int f = 0; f < ascii\_dna\_code.Count; f++)

codan.Add(Convert.ToChar(f).ToString(), ascii\_dna\_code[f].ToString());

int indexx = 0;

for (int i = 0; i < 16; i++)

{

for (int j = 0; j < 16; j++)

{

VAR\_BASE\_DNA\_MATRIX[i, j]=codan[codan.Keys.ElementAt(indexx)];

indexx++;

}}}

1. **Generate Key Matrix**

public void generate\_key\_matrix(string UserEnteredkeyy)

{

key\_remove\_duplicate(UserEnteredkeyy);//"ArAj");

char[] key = keyList\_without\_duplicates.ToArray<char>();

List<string> DNA\_match\_Key\_list = generate\_char\_array\_to\_matching\_dna\_sequence(key);

Generate\_Key\_Full\_List(DNA\_match\_Key\_list);

Display\_matrix(KeyList\_FUll);

}

private void Display\_matrix(List<string> List\_into\_matrix)

{

int index = 0;

for (int i = 0; i < 16; i++)

{

for (int j = 0; j < 16; j++)

{

Console.Write(KEY\_MATRIX[i, j] = List\_into\_matrix[index] + ",");

Console.Write("\t");

index++;

}

Console.WriteLine();

}

}

private void Generate\_Key\_Full\_List(List<string> DNA\_match\_Key\_list)

{

int count = 0;

for (int f = 0; f < ascii\_dna\_code.Count+DNA\_match\_Key\_list.Count; f++)

{

if (f < DNA\_match\_Key\_list.Count)

{

KeyList\_FUll.Add(DNA\_match\_Key\_list[f]);

}

else if (!KeyList\_FUll.Contains(ascii\_dna\_code[count]))

{

KeyList\_FUll.Add(ascii\_dna\_code[count]);

count++;

}

else

{

count++;

}}}//Generate Key Matrix

1. **Encryption Code**

private string getEncryptText(string splittext)

{

string encrypttext = null;

int index = 0;

string nextdata1 = null, nextdata2 = null,nextdata3 = null, nextdata4=null;List<string> get\_codanto\_encryptlist = new List<string>();

foreach (char c in splittext.ToCharArray())

{

string value;

codan.TryGetValue(c.ToString(), out value);

get\_codanto\_encryptlist.Add(value);

}

int[] row = new int[2]; int[] col = new int[2];

for (int i = 0; i < 16; i++)

{

for (int j = 0; j < 16; j++)

{

if (index < get\_codanto\_encryptlist.Count)

{

string compare\_var = KEY\_MATRIX[i, j];

if (compare\_var.Equals(get\_codanto\_encryptlist[index]+","))

{

row[index] = i;

col[index] = j;

i = 0; j = 0;

index++;

}}}}

if (row[0] == row[1])

{

if (col[0] >= lastindex)

{

nextdata1 = KEY\_MATRIX[row[0], col[0] - lastindex];

}

if (col[1] >= lastindex)

{

nextdata2 = KEY\_MATRIX[row[1], col[1] - lastindex];

}

if (col[0] < lastindex)

{

nextdata1 = KEY\_MATRIX[row[0], col[0] + 1];

//both are in same row

}

if (col[1] < lastindex)

{

nextdata2 = KEY\_MATRIX[row[1], col[1] + 1]; }}

else if (col[0] == col[1]){

if (row[0] >= lastindex){

nextdata1 = KEY\_MATRIX[row[0] - lastindex, col[0]];

}if (row[1] >= lastindex){

nextdata2 = KEY\_MATRIX[row[1] - lastindex, col[1]];}

if (row[0] < lastindex)

{

nextdata1 = KEY\_MATRIX[row[0] + 1, col[0]];

//both are in same row}

if (row[1] < lastindex){

nextdata2 = KEY\_MATRIX[row[1] + 1, col[1]]; }

//both are in same Column}

else{

nextdata3 = KEY\_MATRIX[row[0], col[1]];

nextdata4 = KEY\_MATRIX[row[1], col[0]];

}

if (nextdata1 != null)

encrypttext += nextdata1;

if (nextdata2 != null)

encrypttext += nextdata2;

if(nextdata3!=null)

encrypttext += nextdata3;

if(nextdata4!=null)

encrypttext += nextdata4;

return encrypttext;

}////////Completed Encryption///////

1. **Decryption Code**

private void Decryption\_with\_Playfair(string encrypttext)

{

string[] dcryptext = new string[10];

List<string> userdata\_split\_into\_2letters = Get\_user\_encrypt\_text\_data(encrypttext);

for (int i = 0; i < userdata\_split\_into\_2letters.Count; i++)

{

string[] text = getDecryptText(userdata\_split\_into\_2letters[i]).Split(',');

for (int j = 0; j < text.Length; j++)

{

if (text[j] != "")

{

var keys = from entry in codan where entry.Value == text[j] select entry.Key;

foreach (var key in keys)

DecryptList\_Data.Add(key);

}}}}

public string getDecryptText(string encryptSplit)

{

string decrypttext = null;

int index = 0;

string decry\_nextdata1 = null, decry\_nextdata2 = null, decry\_nextdata3 = null, decry\_nextdata4 = null;

List<string> get\_codanto\_decrylist = new List<string>();

for (int i = 0; i < encryptSplit.Length; i += 4)

{

string g = encryptSplit[i].ToString() + encryptSplit[i+1].ToString() + encryptSplit[i+2].ToString() + encryptSplit[i+3].ToString();

get\_codanto\_decrylist.Add(g);//value);

}

int[] decry\_row = new int[2]; int[] decry\_col = new int[2];

for (int i = 0; i < 16; i++)

{for (int j = 0; j < 16; j++)

{

if (index < get\_codanto\_decrylist.Count)

{

string compare\_var = KEY\_MATRIX[i, j];

if (compare\_var.Equals(get\_codanto\_decrylist[index] + ","))

{decry\_row[index] = i;

decry\_col[index] = j;

i = 0; j = 0;

index++;

}}}}

if (decry\_row[0] == decry\_row[1])

{

if (decry\_col[0] > lastindex)

{

decry\_nextdata1 = KEY\_MATRIX[decry\_row[0], decry\_col[0] + lastindex];

}

if (decry\_col[1] > lastindex)

{

decry\_nextdata2 = KEY\_MATRIX[decry\_row[1], decry\_col[1] + lastindex];

}

if (decry\_col[0] <= lastindex)

{

if(decry\_col[0]==0)

decry\_nextdata1 = KEY\_MATRIX[decry\_row[0], decry\_col[0] + lastindex];

else

decry\_nextdata1 = KEY\_MATRIX[decry\_row[0], decry\_col[0] - 1];

}

if (decry\_col[1] <= lastindex)

{

if (decry\_col[1]==0)

decry\_nextdata2 = KEY\_MATRIX[decry\_row[1], decry\_col[1] + lastindex];

else

decry\_nextdata2 = KEY\_MATRIX[decry\_row[1], decry\_col[1] - 1];

}

}

else if (decry\_col[0] == decry\_col[1])

{

if (decry\_row[0] > lastindex)

{

decry\_nextdata1 = KEY\_MATRIX[decry\_row[0] + lastindex, decry\_col[0]];

}

if (decry\_row[1] > lastindex)

{

decry\_nextdata2 = KEY\_MATRIX[decry\_row[1] + lastindex, decry\_col[1]];

}

if (decry\_row[0] <= lastindex)

{

if (decry\_row[0] == 0)

decry\_nextdata1 = KEY\_MATRIX[decry\_row[0] + lastindex, decry\_col[0]];

if (decry\_row[0] == lastindex)

decry\_nextdata1 = KEY\_MATRIX[decry\_row[0] - lastindex, decry\_col[0]];

else if (decry\_row[0] != 0)

{

decry\_nextdata1 = KEY\_MATRIX[decry\_row[0] - 1, decry\_col[0]];

}

//both are in same row

}

if (decry\_row[1] <= lastindex)

{

if (decry\_row[1] == 0)

decry\_nextdata2 = KEY\_MATRIX[decry\_row[1] + lastindex, decry\_col[1]];

if (decry\_row[1] == lastindex)

decry\_nextdata2 = KEY\_MATRIX[decry\_row[1] - lastindex, decry\_col[1]];

else if (decry\_row[1] != 0)

{

decry\_nextdata2 = KEY\_MATRIX[decry\_row[1] - 1, decry\_col[1]];

}

}

//both are in same Column

}

else

{

decry\_nextdata3 = KEY\_MATRIX[decry\_row[0], decry\_col[1]];

decry\_nextdata4 = KEY\_MATRIX[decry\_row[1], decry\_col[0]];

}

if (decry\_nextdata1 != null)

decrypttext += decry\_nextdata1;

if (decry\_nextdata2 != null)

decrypttext += decry\_nextdata2;

if(decry\_nextdata3!=null)

decrypttext += decry\_nextdata3;

if(decry\_nextdata4!=null)

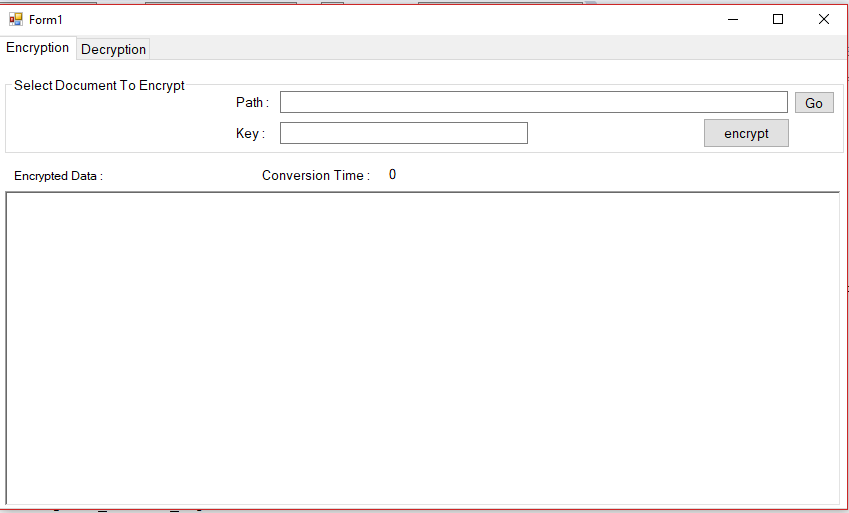
decrypttext += decry\_nextdata4;

return decrypttext;

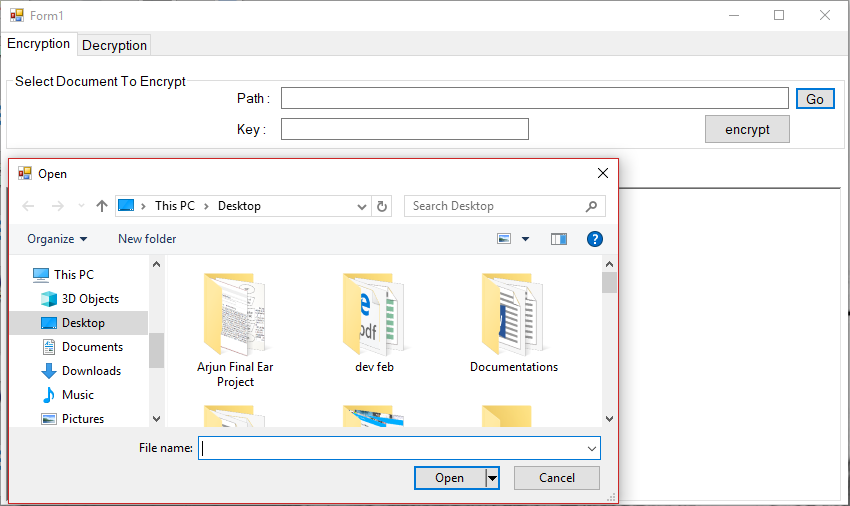
}///////Completed Decryption

**4.4 SCREENSHOTS**

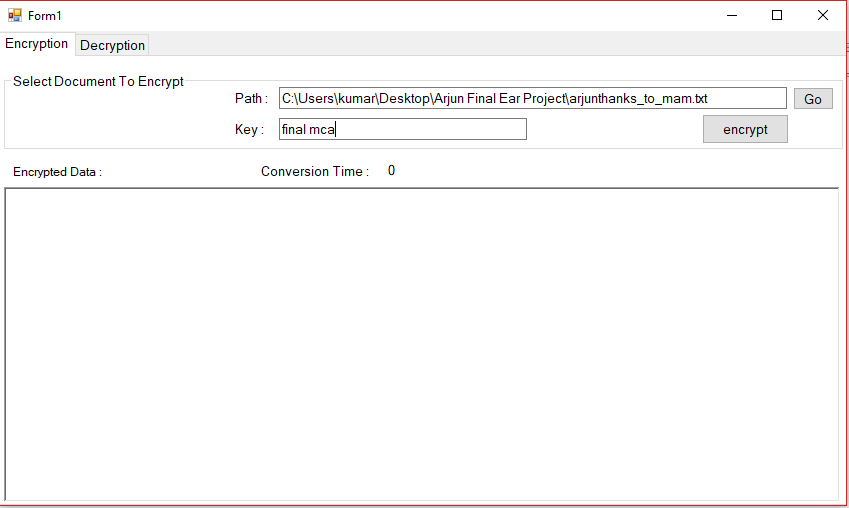
1. **UI**

****

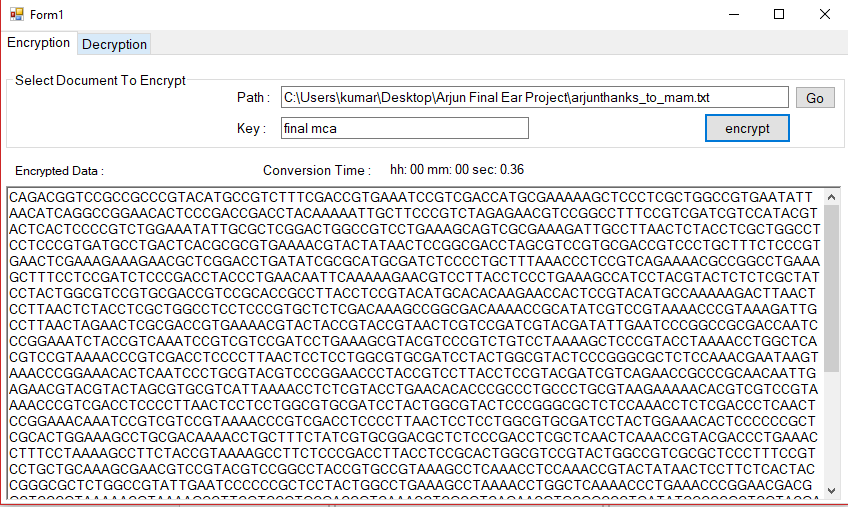
1. **Go Button Click**

****

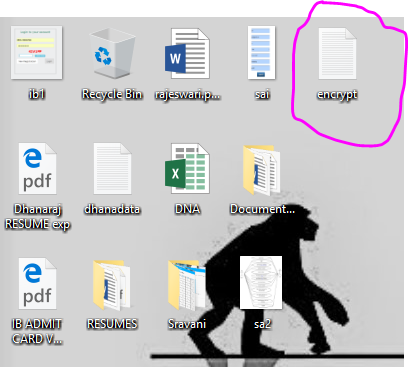
1. **Entering Encrypt Key**

****

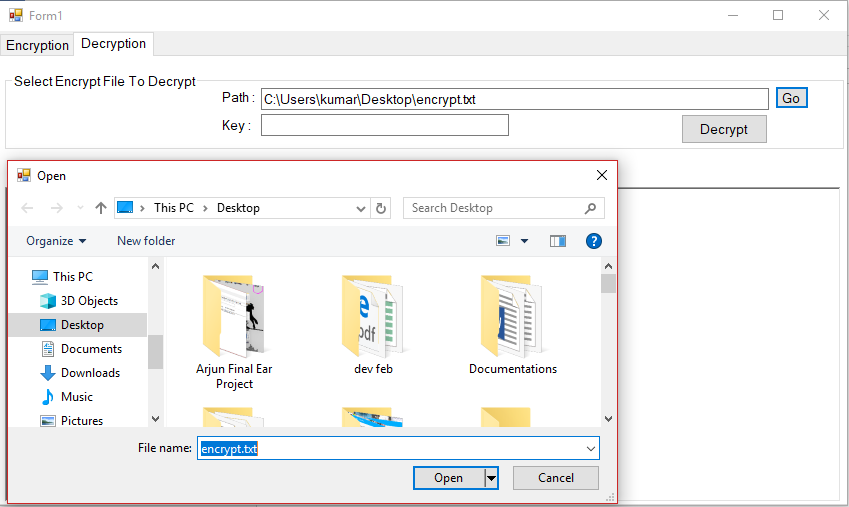
1. **Encryption Button Click**

****

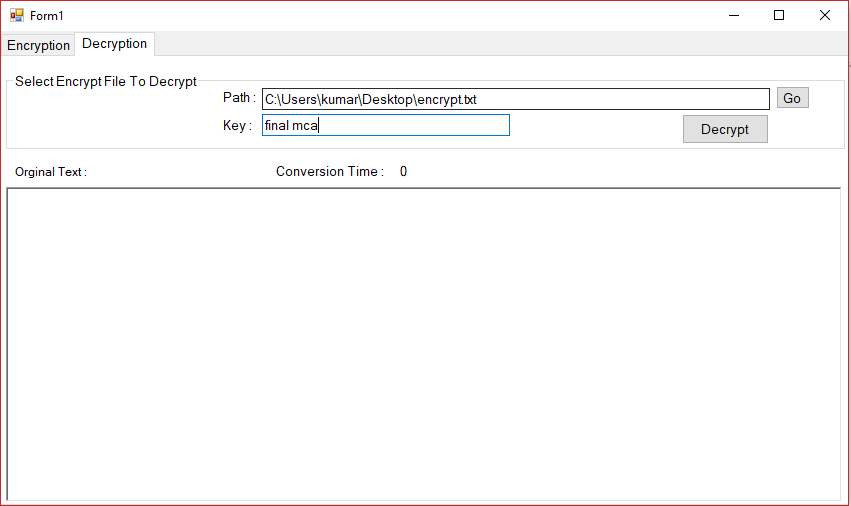
1. **Encrypt File Created**

****

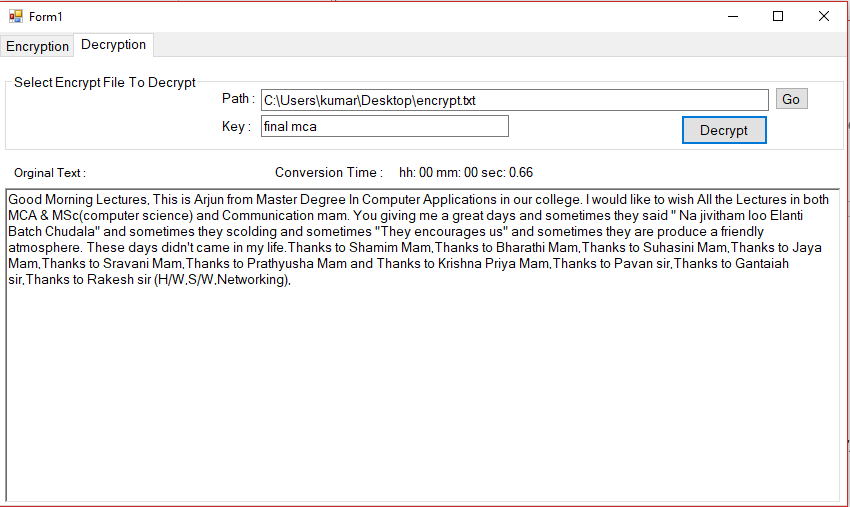
1. **Decryption => Go Button Click**

****

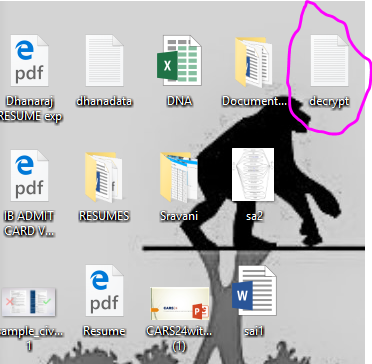
1. **Entering Decrypt Key**

****

1. **Decryption Button Click**

****

1. **Decrypt File Created**

****

**RESULT ANALYSIS**

|  |  |  |
| --- | --- | --- |
| **SIZE** | **Encryption Time** | **Decryption Time** |
| **1 KB** | **0.011 sec** | **0.10 sec** |
| **5 KB** | **3.45 sec** | **7.32 sec** |
| **10 KB** | **5.23 sec** | **10.23 sec** |
| **50 KB** | **10.84 sec** | **26.54 sec** |