Kombinasi antara *Cipher Subtitusi, Cipher Transposisi,* dan *Playfair Cipher* sebagai landasan algoritma enkripsi baru

Andzarrahim
13504013
Departemen Teknik Informatika
Institut Teknologi Bandung
E-mail : if14013@students.if.itb.ac.id

Abstraksi

Playfair Cipher pertama kali ditemukan oleh Sir Charles Wheatstone dan Baron Lyon Playfair pada tahun 1854,. Pada awalnya algoritma ini digunakan oleh tentara Inggris semasa perang dunia I. Algoritma ini menggunakan table berukuran 5 x 5 dalam proses penenkripsian pesannya. Tabel ini berperan sebagai kunci dalam proses enkripsi dan dekripsi cipher ini. Selain itu algoritma ini memiliki keunikan karena menghilangkan huruf J untuk plain textnya. Akan tetapi, ke depannya algoritma ini tidak terpakai lagi dikarenakan mudahnya proses pengenkripsian dengan menggunakan kunci yang ada. Algoritma ini merupakan salah satu algoritma *Polygram Cipher* Algoritma ini tidak mengoptimalkan dasar dari *algoritma klasik* yaitu *Cipher Subtitusi* dan *Cipher Transposisi* dalam proses pengenkripsiannya. Algoritma enkripsi baru yang akan diciptakan menggunakan prinsip *Cipher Subtitusi* dan *Cipher Transposisi* untuk menutupi kelemahan yang dimiliki oleh *Playfair Cipher* terutama pada bagian pengacakan table kunci dan hasil setelah enkripsi dengan menggunakan kunci tersebur. Untuk lebih lanjutnya akan saya bahas di makalah ini.

Dalam makalah ini akan dibahas lebih lanjut tentang *Playfair Cipher*, keunggulan serta kelemahannya. Makalah ini juga akan menjelaskan dasar dari algoritma klasik. Lalu dengan mengkombinasikan seluruh teori dasar algoritma klasik dan *Playfair Cipher* akan dibentuk algoritma Enkripsi baru lengkap dengan keunggulan dan kelemahannya beserta implementasi kasus.

Kata Kunci: *Playfair Cipher, Cipher Transposisi, Cipher Subtitusi, Polygram Cipher, Algoritma Klasik, Ciphertext, Plaintext*

1. PENDAHULUAN

Teknologi telah berkembang sebegitu pesat seiring kemajuan zaman. Teknologi seperti televisi, telepon , komputer, radio, handphone, dan masih banyak lagi telah merambah ke seluruh bidang yang ada di zaman sekarang ini. Tak terkecuali dengan bidang komunikasi, bidang ini merupakan bidang yang mengalami perkembagan dengan sangat pesat. Ini bisa dilihat dengan adanya teknologi seperti *SMS (Short Messaging Service*), *MMS (Multimedia Messaging Services*), *VOIP (Voice Over Internet Protocol)* dan masih banyak lagi.

Akan tetapi, seiring berkembangnya teknologi komunikasi, banyak sekali orang yang tidak memperhatikan segi keamanannya. Untuk itu adanya suatu algoritma dalam mengenkripsi pesan yang dikirim sangat dibutuhkan.

2. DASAR TEORI

2.1. *PLAYFAIR CIPHER*

Playfair Cipher pertama kali ditemukan oleh Charles Wheatstone pada tahun 1854, akan tetapi dipopulerkan oleh Lord Playfair beberapa tahun setelahnya. Algoritma enkripsi ini merupakan algoritma enkripsi simetris dan merupakan algortima subtitusi Digraph pertama yang pernah ada.

Algortima ini memiliki ciri khas yaitu penggunaan tabel 5 x 5 sebagai tabel kunci yang terdiri dari huruf alphabet. Tabel ini digunakan untuk mengenkripsikan masukan dengan cara mensubtitusikan setiap dua hurufmasukan dengan kunci melalui aturan tertentu (akan dibahas lebih lanjut di poin selanjutnya).

Dalam pembentukan tabel kunci itu sendiri ada aturan khusus yaitu tidak adanya huruf berulang dalam tabel dan tidak adanya huruf "J" dalam isi tabel tersebut.

Misal :

Masukan kunci = **"BANANA"**

Rumus Tabel kunci = ***Subset*** **masukan + Huruf alphabet selain *subset* dan huruf "J"**

= **"BAN" + [CDEFG….. Z]**

Sehingga tabel kunci yang terbentuk

B	A	N	C	D
E	F	G	H	I
K	L	M	O	P
Q	R	S	T	U
V	W	X	Y	Z

Selain itu dalam sebelum proses pengenkripsiannya *Plaintext* dari user haruslah melalui beberapa proses sebelum dapat dienkripsikan dengan tabel kunci yang telah terbentuk

Ada beberapa proses yang harus dipenuhi oleh *Plaintext*, yaitu :

1. Ganti huruf "J" dengan huruf "I".
2. Bentuk *Plaintext* dalam pasangan huruf.
3. Jika ada pasangan huruf yang sama, maka harus diselipkan Z di tengahnya.
4. Tambahkan huruf Z di akhir apabila jumlah huruf ganjil.

Misal :

Plaintext = **"TREE RUN JACK"**

Setelah melalui proses pertama

Plaintext = **"TREE RUN IACK"**

Setelah melalui proses kedua

Plaintext = **TR EE RU NI AC K**

Setelah melalui proses ketiga

Plaintext **= TR EZ ER UN IA CK**

Setelah melalu proses keempat

Plaintext = **TR EZ ER UN IA CK**

Proses keempat tidak dijalankan karena jumlah huruf tidaklah ganjil. Setelah kedua proses tersebut selesai barulah Algoritma *Playfair Cipher* bisa dilakukan.

2.2. PROSES ENKRIPSI

Dalam proses pengenkripsiannya, *Playfair Cipher* mengambil pasangan hurufdari *Plaintext* yang telah diproses, lalu disandingkan dengan Tabel Kunci sebelum mendapatkan *Ciphertext*.

Ada beberapa aturan yang harus dipenuhi dalam proses pengenkripsiannya, yaitu :

1. Jika pasangan huruf *Plaintext* ada di baris yang sama, maka pasangan huruf *Ciphertext* adalah satu huruf sebelah kanan dari posisi *Plaintext* sebelumnya
2. Jika pasangan huruf *Plaintext* ada di kolom yang sama, maka pasangan huruf *Ciphertext* adalah satu huruf sebelah bawah dari posisi *Plaintext* sebelumnya
3. Jika pasangan huruf *Plaintext* tidak pada kolom ataupun baris yang sama, maka pasangan huruf pada *Ciphertext* merupakan pasangan hurufpembentuk segi empat pada Tabel kunci. Huruf pertama pada *Ciphertext* memiliki baris yang sama dengan huruf pertama pada *Plaintext*

Berikut penjelasan proses enkripsi *Playfair Cipher*.

Misal :

Plaintext = "Kriptografi" ➔ **KR IP TO GR AF IZ**

Kunci = "Run"

Tabel Kunci yang terbentuk

R	U	N	A	B
C	D	E	F	G
H	I	K	L	M
O	P	Q	S	T
V	W	X	Y	Z

Untuk pasangan pertama **KR** karena memenuhi aturan ketiga maka :

R	U	N	A	B
C	D	E	F	G
H	I	K	L	M
O	P	Q	S	T
V	W	X	Y	Z

R	U	N	A	B
C	D	E	F	G
H	I	K	L	M
O	P	Q	S	T
V	W	X	Y	Z

Didapatkan pasangan pertama pada *Ciphertext* **HN.** Untuk pasangan kedua **IP** karena memenuhi aturan kedua (x1 = x2) maka :

R	U	N	A	B
C	D	E	F	G
H	I	K	L	M
O	P	Q	S	T
V	W	X	Y	Z

R	U	N	A	B
C	D	E	F	G
H	I	K	L	M
O	P	Q	S	T
V	W	X	Y	Z

Didapatkan pasangan kedua pada *Ciphertext* **PW**
Untuk pasangan ketiga **TO** karena memenuhi aturan pertama (y1 = y2) maka :

R	U	N	A	B
C	D	E	F	G
H	I	K	L	M
O	P	Q	S	T
V	W	X	Y	Z

R	U	N	A	B
C	D	E	F	G
H	I	K	L	M
O	P	Q	S	T
V	W	X	Y	Z

Didapatkan pasangan ketiga pada *Ciphertext* **OP** dan seterusnya hingga seluruh *Plaintext* diproses sehingga didapatkan :

Plaintext = **KR IP TO GR AF IZ**
Ciphertext = **HN PW OP CB FL MW**

2.3. PROSES DEKRIPSI

Untuk proses dekripsi hanya memutar balikkan aturan yang ada untuk enkripsi sedangkan kunci yang digunakan unruk proses dekripsi harus sama dengan kunci enkripsinya. Adapun aturannya menjadi

1. Jika pasangan huruf *Ciphertext* ada di baris yang sama, maka pasangan huruf *Plaintext* adalah satu huruf sebelah kanan dari posisi *Ciphertext* sebelumnya
2. Jika pasangan huruf *Ciphertext* ada di kolom yang sama, maka pasangan huruf *Plaintext* adalah satu huruf sebelah bawah dari posisi *Ciphertext* sebelumnya
3. Jika pasangan huruf *Ciphertext* tidak pada kolom ataupun baris yang sama, maka pasangan huruf pada *Plaintext* merupakan pasangan hurufpembentuk segi empat pada Tabel kunci. Huruf pertama pada *Ciphertext* memiliki baris yang sama dengan huruf pertama pada *Plaintext*

Misal :

Ciphertext = **HN PW OP CB FL MW**

Tabel Kunci yang terbentuk

R	U	N	A	B
C	D	E	F	G
H	I	K	L	M
O	P	Q	S	T
V	W	X	Y	Z

Untuk pasangan pertama pada *Ciphertext* HN

R	U	N	A	B
C	D	E	F	G
H	I	K	L	M
O	P	Q	S	T
V	W	X	Y	Z

R	U	N	A	B
C	D	E	F	G
H	I	K	L	M
O	P	Q	S	T
V	W	X	Y	Z

Didapatkan pasangan pertama *Plaintext* KR lalu untuk pasangan *Ciphertext* selanjutnya menggunakan aturan yang ada di enkripsi sehingga didapatkan

Ciphertext = **HN PW OP CB FL MW**
Plaintext = **KR IP TO GR AF IZ**

3. HASIL DAN PEMBAHASAN

3.1. ANALISA KELEMAHAN *PLAYFAIR CIPHER*

Playfair Cipher memiliki keunggulan tersendiri dibanding algoritma enkripsi klasik lainnya, karena dalam pemrosesannya algoritma ini menggunakan kombinasi dua huruf (Digraph) sehinggga Teknik Analisis Frekuensi sangat sukar digunakan untuk seorang kriptanalis menyerang algoritma ini.

Ini disebabkan oleh banyaknya kombinasi yang diberikan oleh digraph (monograph = 26 sedangkan digraph = 26 x 25). Lain halnya apabila kriptanalis menyerang dengan metode analisis pasangan huruf. Berikut contoh kasusnya.

Asumsi

- seorang kriptanalis mengetahui algoritma yang dipakai
- mengetahui bidang yang digemari pembuat *Ciphertext*
- Mengetahui cara kerja pembutatan kunci

Ciphertext = **UQ SM BV DV UB UK DQ BU SI BU CU PN SE NS TO UC SI DQ DB QT AE DW SI DQ UB UK DQ BU**

Di sini terlihat bahwa ada pengulangan pasangan kata UB, UK, DQ, BU, SI lalu adanya pembalikan pasangan seperti UB ➔ BU , CU ➔ UC .

Pertama tama seorang kriptanalis akan mencari kata yang mungkin berulang secara utuh dan tidak dipengaruhi oleh bentuk digraph dari *Playfair Cipher*.

UB UK DQ BU … UB UK DQ BU

Kriptanalis mengetahui bidang yang digeluti oleh pembuat cipher yaitu jaringan komputer . Ada beberapa kosakata yang mungkin dipakai seperti RE CE IV ER, RE PE AT ER , RE ND ER ER dll.

Kriptanalis mengasumsikan bahwa

UB UK DQ BU ➔ RE PE AT ER … (1)

Dari poin ini, dapat disimpulkan bahwa dalam tabel kunci huruf R dan E tidak terletak dalam baris dan kolom yang sama .Berikutnya kriptanalis akan mencari digraph berikutnya yang berdekatan dengan kata tersebut.

Maka didapatkan

SI BU CU ➔ … ER …

Lalu dari poin ini kriptanalis menggunakan digraph yang lain

SI DQ … SI DQ

Di baris paling bawah di *Ciphertext*, SI DQ terletak sebelum UB UK DQ BU sedangkan UB UK DQ BU telah diterjemahkan menjadi kata benda yaitu RE PE AT ER sehingga besar kemungkinan SI DQ merupakan cipher dari petunjuk kata benda

SI DQ ➔ THAT, THIS

Ini juga diperkuat dengan

SI BU CU ➔ … ER … ➔ TH ER E…

Lalu kriptanalis mengasumsikan

SI ➔ TH

Maka tabel kunci sementara yang mungkin dibentuk (untuk sementara pembentukan tabel kunci tidak dalam tabel 5 x 5)

	B			E	
	R			U	

Ini didapatkan dari persamaan (1) yang menjelaskan bahwa B dan U tidak terletak pada baris dan kolom yang sama . Lalu dari persamaan (1) juga diasumsikan

DQ ➔ AT

Kriptanalis berspekulasi dengan meletakkan huruf A di sebelah huruf B dan meletakkan pasangannya huruf T di antara R dan U. Ini semakin diperkuat dengan posisi D dan Q yang memenuhi aturan pembuatan kunci..

Sehingga

	A	B		D	E	
	Q	R		T	U	

Selajutnya dari persamaan (2), kriptanalis mengasumsikan bahwa S terletak di antara R dan T sehingga tabel kunci yang mungkin

	A	B		D	E	
			H	I		
			H	I		
	Q	R	S	T	U	
			H	I		

Sampai sini sudah banyak yang bisa dilakukan oleh kriptanalis seperti dengan *analytical attack* yaitu mencari kosakata lain dalam bahasa inggris lalu dengan digraph yang ada mencoba mencari kombinasi kunci yang mungkin dan mencocokkannya dengan tabel kunci.

Selain *analytical attack*, kriptanalis dapat juga dengan teknik *brute force*. Ini bisa dilakukan dengan cara mencoba satu satu digraph yang ada

dan mencoba mencari padanan kata yang lazim apabila dipasangkan dengan digraph sebelum dan sesudahnya.

UQ ➔ TU
UQ SM ➔ TU …
➔ TU RN
➔ TU NE , dll

Di bagian ini saya tidak akan menyelesaikan sampai tuntas proses kriptanalisnya (karena tidak akan cukup) dan dalam kasus ini kriptanalis masih belum bisa memabentuk tabel kunci yang fix sehingga membutuhkan berkas *Ciphertext* dari kunci yang sama.

Penulis menyimpulkan bahwa metode analisis frekuensi pasangan huruf dapat digunakan secara efektif untuk menemukan kunci dalam *Playfair Cipher* . Karena pengulangan pola masih saja dapat ditemukan dalam *Plaintext* untuk beberap kasus tertentu., lalu berdasarkan poin ini pula penulis membentuk algoritma enkripsi baru.

3.2. PEMBENTUKAN ALGORITMA ENKRIPSI BARU

Berdasarkan kelemahan di atas penulis berusaha menciptakan modifikasi tambahan dalam menanggulangi masalah yang dijelaskan di atas

3.2.1. SUBTITUSI POSISI

Algoritma ini bekerja setelah *Plaintext* telah dienkripsi menjadi *Ciphertext*. Algoritma ini bekerja dengan cara mensubtitusikan setiap huruf *Ciphertext* dengan rumus tertentu berdasarkan posisi yang ditempati oleh huruf tersebut.

Cara kerja Algoritma

Input = AB BA
alphabet = "ABCD… Z"

```
for (int i = 0;i < Input.Length; i++)
{
    for (int j = 0; j < alphabet.Length; j++)
     {
         if (alphabet[j] == Input[i])
         {
            temp = j + 1;
          }

          }
          acc = ((temp + i) % 25);
          output[i] = alphabet[acc];

 }
```

Dengan persamaan umum

Ciphertext = **(urutan X dalam array alphabet + posisi X dalam array Input) mod 25**

Jadi prosesnya untuk huruf pertama **A**

Urutan Alphabet = 1
Posisi Input = 1
Ciphertext = (1 + 1) mod 25
= 2
Ciphertext[i] ➔ **B**

Selanjutnya untuk huruf kedua B

Urutan Alphabet = 2
Posisi Input = 2
Ciphertext = (2 + 2) mod 25
= 4
Ciphertext[i] ➔ **D**

Lalu untuk huruf ketiga B

Urutan Alphabet = 2
Posisi Input = 3
Ciphertext = (2 + 3) mod 25
= 5
Ciphertext[i] ➔ **E**

Seterusnya hingga mendapatkan

Input = **AB BA**
Ciphertext = **BD EE**

(source lengkap dapat dilihat di lampiran)

3.2.2. TRANSPOSISI DUA POSISI

Algoritma ini bertujuan merubah susunan setiap digraph menjadi terbalik sehingga mengacaukan kriptanalis untuk mendapatkan kunci. Algoritma ini bekerja setelah algoritma subtitusi posisi.

Cara kerja algoritma

Input = BD EE FG AS KL

```
for (int i = 0; i < Input.Length; i++)
 {
   if (i % 2 != 0)
   {
    output[i-1] = Input[i];
    output[i] = Input[i - 1];
    }

 }
if (Input.Length % 2 != 0)
{
```

```
  output[Input.Length - 1] = Input[Input.Length - 1];
}
```

Sehingga hasilnya menjadi

Input = **BD EE FG AS KL**
Plaintext = **DB EE GF SA LK**

(source lengkap dapat dilihat di lampiran)

3.3. ANALISA ALGORITMA ENKRIPSI BARU

Dengan adanya dua modifikasi di atas hasil enkripsi menjadi lebih kompleks dan lebih membingungkan. Algoritma Subtitusi posisi berperan dalam memperbesar peluang tidak terbentuknya pengulangan kata dalam *Ciphertext* sehingga kriptanalis membutuhkan lebih banyak informasi ataupun berkas *Ciphertext* untuk memecahkannya.

Algoritma transposisi dua posisi berperan dalam menyembunyikan isi dari *Plaintext* itu sendiri. Algoritma ini bisa menyebabkan ambiguitas pesan yang telah di *Ciphertext*. Berdasarkan hasil tes dengan program yang penulis buat, algoritma enkripsi berhasil menciptakan kemerataan angka yang lebih signifikan dari sebelumnya sehingga mengacaukan analisis frekuensi. Selain itu, dengan adanya kombinasi keduanya pesan yang terenkripsi menjadi lebih aman dan robust.

4. KESIMPULAN

Berdasarkan pembahasan di atas, penulis menyimpulkan :

1. *Playfair Cipher* merupakan algoritma *Polygram Cipher* yang memiliki tingkat keamanan yang cukup tinggi.
2. Metode analisis frekuensi pasangan huruf merupakan metode paling ideal dalam memecahkan *Ciphertext*
3. Tambahan algoritma dari penulis berhasil mengatasi masalah sementara yang dialami *Playfair Cipher* yaitu pola kata yang berulang
4. Algoritma subtitusi posisi efektif dalam mengurangi muncul huruf terbanyak.
5. Algoritma enkripsi baru ini memliki kelemahan dari penggunaan memori, ini dikarenakan banyaknya proses yang dilakukan dalam satu tahapannya.

5. DAFTAR REFERENSI

[1]. Munir, Rinaldi, *Diktat Kuliah IF5054 Kriptografi*, Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung, 2006

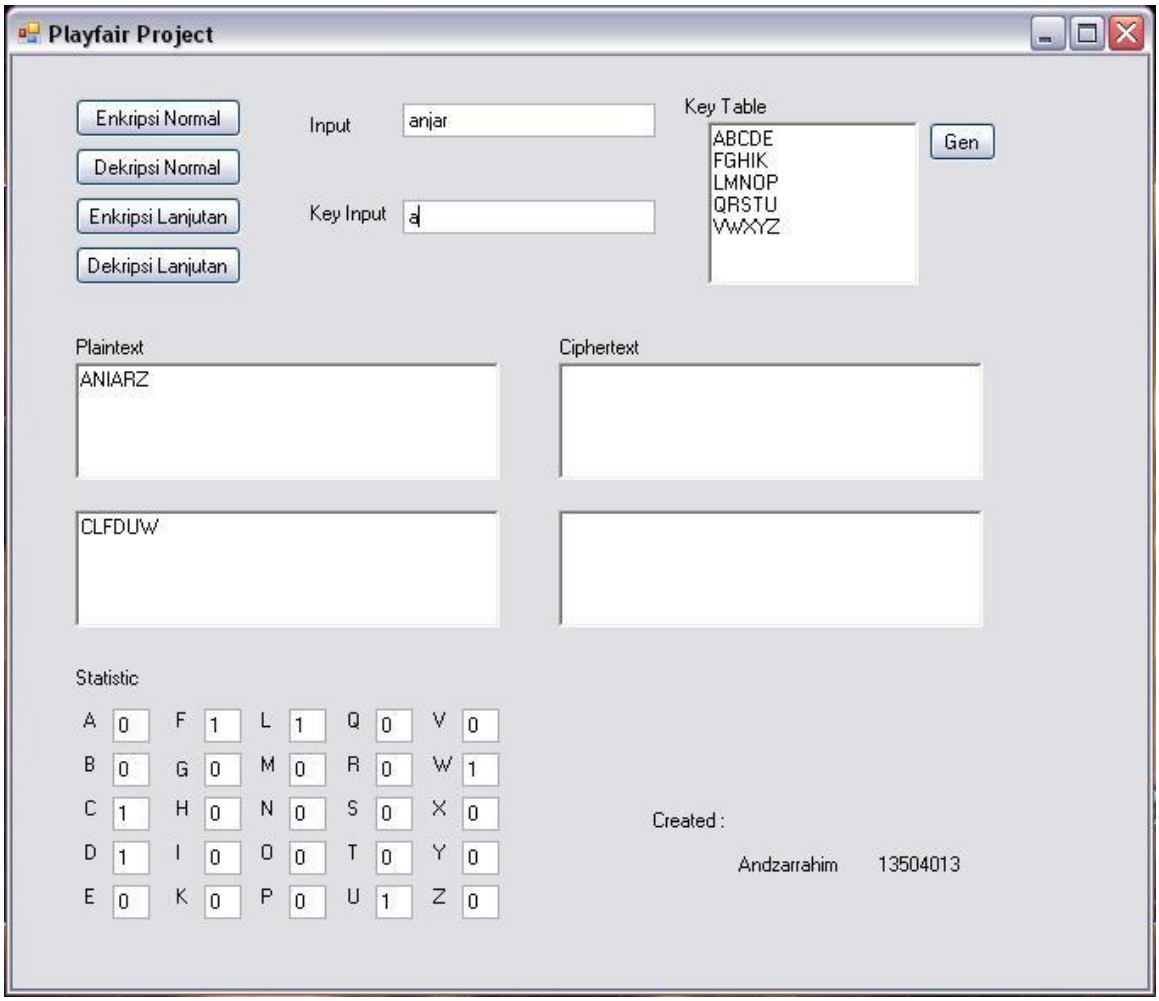
[2]. Forouzan, Behrouz, *Cryptography and Network Security*, McGraw-Hill, 2008.

[3]. http://en.wikipedia.org/wiki/Playfair_cipher

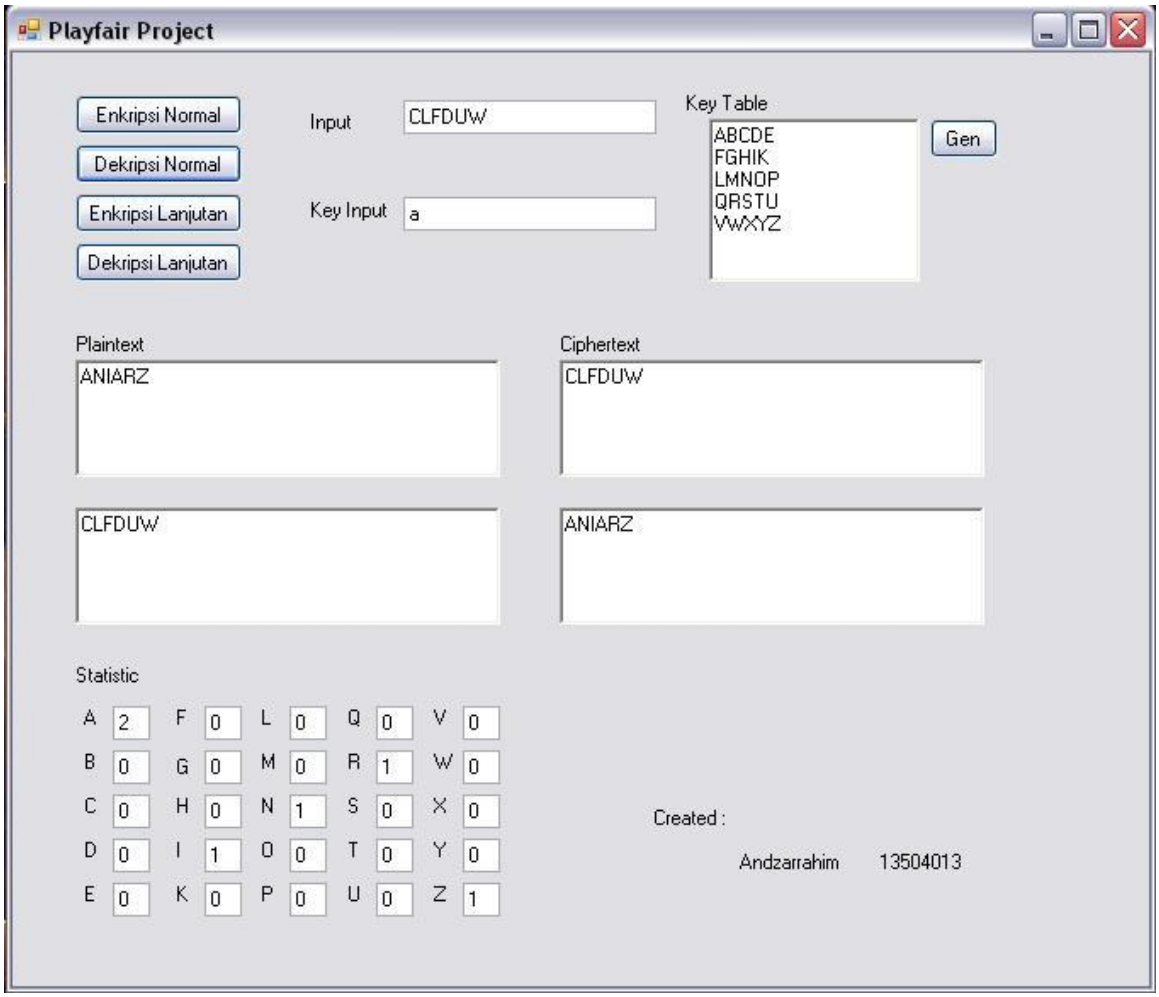
LAMPIRAN

Interface Program

Enkripsi



Dekripsi


Playfair Project
Enkripsi Normal
Dekripsi Normal
Enkripsi Lanjutan
Dekripsi Lanjutan
Input
CLFDUW
Key Input
a
Key Table
ABCDE
FGHIK
LMNOP
QRSTU
VWXYZ
Gen
Plaintext
ANIARZ
Ciphertext
CLFDUW
CLFDUW
ANIARZ
Statistic
A 2
F 0
L 0
Q 0
V 0
B 0
G 0
M 0
R 1
W 0
C 0
H 0
N 1
S 0
X 0
D 0
I 1
O 0
T 0
Y 0
E 0
K 0
P 0
U 0
Z 1
Created :
Andzarrahim 13504013

Source code program

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.IO;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Playfair_Project
{
    public partial class Form1 : Form
    {

        public Form1()
        {
            InitializeComponent();



        }

        private void button1_Click(object sender, EventArgs e) //
enkripsi normal
        {
            richTextBox1.Clear();
            richTextBox2.Clear();
            richTextBox2.AppendText(new
string(BeforeEncrypt(textBox1.Text.Trim().ToUpper().ToCharArray())));
            char[] tes =
NormalEncrypt(BeforeEncrypt(textBox1.Text.Trim().ToUpper().ToCharArray(
)), CreateKeyTable(textBox2.Text.Trim().ToCharArray()));
            GenerateStats(tes);
            richTextBox1.AppendText(new string(tes));

        }


        private void button2_Click(object sender, EventArgs e) //
dekripsi normal
        {
            richTextBox4.Clear();
            richTextBox5.Clear();
            richTextBox4.AppendText(new
string(textBox1.Text.Trim().ToUpper().ToCharArray()));
            char[] tes =
NormalDecrypt(textBox1.Text.Trim().ToUpper().ToCharArray(),
CreateKeyTable(textBox2.Text.Trim().ToCharArray()));
            GenerateStats(tes);
            richTextBox5.AppendText(new string(tes));
        }
```

```
        private void button5_Click(object sender, EventArgs e) //
Generate keytable
        {
            richTextBox3.Clear();
            char[,] zeus =
CreateKeyTable(textBox2.Text.Trim().ToCharArray());
            for (int i = 0; i < 5; i++)
            {
                for (int j = 0; j < 5; j++)
                {
                    richTextBox3.AppendText(zeus[i, j].ToString());
                }
                richTextBox3.AppendText("\n");
            }
        }


        private void button3_Click(object sender, EventArgs e)
        {
            richTextBox1.Clear();
            richTextBox2.Clear();
            richTextBox2.AppendText(new
string(BeforeEncrypt(textBox1.Text.Trim().ToUpper().ToCharArray())));
            char[] tes =
EnkripsiLanjutan(NormalEncrypt(BeforeEncrypt(textBox1.Text.Trim().ToUpp
er().ToCharArray()),
CreateKeyTable(textBox2.Text.Trim().ToCharArray())));
            GenerateStats(tes);
            richTextBox1.AppendText(new string(tes));
        }

        private void button4_Click(object sender, EventArgs e)
        {
            richTextBox4.Clear();
            richTextBox5.Clear();
            richTextBox4.AppendText(new
string(textBox1.Text.Trim().ToUpper().ToCharArray()));
            char[] tes =
NormalDecrypt(DekripsiLanjutan(textBox1.Text.Trim().ToUpper().ToCharArr
ay()), CreateKeyTable(textBox2.Text.Trim().ToCharArray()));
            GenerateStats(tes);
            richTextBox5.AppendText(new string(tes));
        }

        public char[] NormalEncrypt(char[] plaintext, char[,] key)
        {
            char[] output = new char[plaintext.Length];
            int a, b;
            int x1, y1, x2, y2;



            // loop berulang dalam prosesnya
            for (int i = 0; i < plaintext.Length; i++)
            {
```

```
                if (i % 2 == 0)  // memproses setiap 2 member
                {
                    a = i;
                    b = i + 1;

                    x1 = 0;
                    x2 = 0;
                    y1 = 0;
                    y2 = 0;
                    int j = 0;



                    for (int m = 0; m < 5; m++)
                    {
                        for (int n = 0; n < 5; n++)
                        {
                            if (plaintext[a] == key[m, n])
                            {
                                x1 = n;
                                y1 = m;
                            }
                        }
                    }

                    for (int m1 = 0; m1 < 5; m1++)
                    {
                        for (int n1 = 0; n1 < 5; n1++)
                        {
                            if (plaintext[b] == key[m1, n1])
                            {
                                x2 = n1;
                                y2 = m1;
                            }
                        }
                    }


                    // sampe di sini x dan y yang didapat bener namun
                    // ketika memasukkan ke dalam koordinat tabel kunci
harus dibalik
                    // karena tabel kunci yang terbentuk key [y,x]

                    if (x1 == x2)
                    {
                        y1 = y1 + 1;
                        y2 = y2 + 1;
                        if (y1 > 4) { y1 = 0; }
                        if (y2 > 4) { y2 = 0; }
                        output[a] = key[y1, x1];
                        output[b] = key[y2, x1];
                    }
                    if (y1 == y2)
                    {
                        x1 = x1 + 1;
                        x2 = x2 + 1;
                        if (x1 > 4) { x1 = 0; }
```

```
                if (x2 > 4) { x2 = 0; }

                output[a] = key[y1, x1];
                output[b] = key[y1, x2];
            }

            if ((x1 != x2) && (y1 != y2))
            {
                output[b] = key[y2, x1];
                output[a] = key[y1, x2];
            }
        }


    }

    return output;
}

public char[] NormalDecrypt(char[] ciphertext, char[,] key)
{
    char[] output = new char[ciphertext.Length];
    int a, b;
    int x1, y1, x2, y2;



    // loop berulang dalam prosesnya
    for (int i = 0; i < ciphertext.Length; i++)
    {

        if (i % 2 == 0)  // memproses setiap 2 member
        {
            a = i;
            b = i + 1;

            x1 = 0;
            x2 = 0;
            y1 = 0;
            y2 = 0;
            int j = 0;



            for (int m = 0; m < 5; m++)
            {
                for (int n = 0; n < 5; n++)
                {
                    if (ciphertext[a] == key[m, n])
                    {
                        x1 = n;
                        y1 = m;
                    }
                }
            }

            for (int m1 = 0; m1 < 5; m1++)
```

```
                    {
                        for (int n1 = 0; n1 < 5; n1++)
                        {
                            if (ciphertext[b] == key[m1, n1])
                            {
                                x2 = n1;
                                y2 = m1;
                            }
                        }
                    }




                    // sampe di sini x dan y yang didapat bener namun
                    // ketika memasukkan ke dalam koordinat tabel kunci
harus dibalik
                    // karena tabel kunci yang terbentuk key [y,x]
                    if (x1 == x2)
                    {
                        y1 = y1 - 1;
                        y2 = y2 - 1;
                        if (y1 < 0) { y1 = 4; }
                        if (y2 < 0) { y2 = 4; }
                        output[a] = key[y1, x1];
                        output[b] = key[y2, x1];
                    }
                    if (y1 == y2)
                    {
                        x1 = x1 - 1;
                        x2 = x2 - 1;
                        if (x1 < 0) { x1 = 4; }
                        if (x2 < 0) { x2 = 4; }

                        output[a] = key[y1, x1];
                        output[b] = key[y1, x2];
                    }

                    if ((x1 != x2) && (y1 != y2))
                    {
                        output[b] = key[y2, x1]; //satu
                        output[a] = key[y1, x2]; //dua
                    }
                }


            }

            return output;
        }

        public char[] EnkripsiLanjutan(char[] input)
        {
          //char[] output = ConvertAZ(input).ToCharArray();
           char[] output = Scramble(input);
          output = GenerateByPosition(output);
```

```
    return output;
}

public char[] DekripsiLanjutan(char[] input)
{
   char[] output = NormaledByPosition(input);
  output = Scramble(output);
   //output = DeconvertAZ(output).ToCharArray();
    return output;
}


// Fungsi buatan
public string ConvertAZ(char[] input)
{
    string output = "";
    string kumphuruf = "ABCDEFGHIKLMNOPQRSTUVWXYZ";
    char[] alphabet = kumphuruf.ToCharArray(); // 0 - 24
    int lagi;

    for (int j = 0; j < input.Length; j++)
    {
        for (int i = 0; i < alphabet.Length; i++)
        {
            if (alphabet[i] == input[j])
            {
                lagi = i + 1;
                for (int a = 0; a < lagi; a++)
                {
                    output = output + "A";
                }
            }
        }

        output = output + "Z";
    }
    return output;
} // mengkonversikan semua karakter text dengan huruf A dan Z

public string DeconvertAZ(char[] input)
{
    string output = "";
    string kumphuruf = "ABCDEFGHIKLMNOPQRSTUVWXYZ";
    char[] alphabet = kumphuruf.ToCharArray(); // 0 - 24
    int lagi = 0;

    for (int i = 0; i < input.Length; i++)
    {
        if (input[i] == alphabet[24])
        {
            lagi = lagi - 1;
            output = output + alphabet[lagi].ToString();
            lagi = 0;
        }
        else
        {
            lagi++;
```

```
            }
        }

        return output;
    }  // mengembalikan hasil konversi di atas

    public char[] Scramble(char[] input)
    {
        char[] output = new char[input.Length];

        for (int i = 0; i < input.Length; i++)
        {
            if (i % 2 != 0)
            {
                output[i-1] = input[i];
                output[i] = input[i - 1];
            }

        }
        if (input.Length % 2 != 0)
        {
            output[input.Length - 1] = input[input.Length - 1];
        }

        return output;
    } // membalik urutan setiap kelipatan 2 dalam tabel masukan

    public char[] GenerateByPosition(char[] input)
    {
        string kumphuruf = "ABCDEFGHIKLMNOPQRSTUVWXYZ";
        char[] alphabet = kumphuruf.ToCharArray(); // 0 - 24
        char[] output = new char[input.Length];
        int temp = 0;
        int acc = 0;
        for (int i = 0; i < input.Length; i++)
        {

            for (int j = 0; j < alphabet.Length; j++)
            {
                if (alphabet[j] == input[i])
                {
                    temp = j + 1;
                }

            }
            acc = ((temp + i) % 25);
            output[i] = alphabet[acc];

        }

        return output;
    }  // mengkonversikan karakter berdasarkan urutan dalam tabel

    public char[] NormaledByPosition(char[] input)
    {
        string kumphuruf = "ABCDEFGHIKLMNOPQRSTUVWXYZ";
```

```
    char[] alphabet = kumphuruf.ToCharArray(); // 0 - 24
    char[] output = new char[input.Length];
    int temp = 0;
    int acc = 0;
    for (int i = 0; i < input.Length; i++)
    {

        for (int j = 0; j < alphabet.Length; j++)
        {
            if (alphabet[j] == input[i])
            {
                temp = j + 1;
            }

        }
        acc = temp -i - 1;
        if (acc < 1)
        {
            acc = 25 -(( acc * -1) %25);
        }

        acc = acc - 1;

        output[i] = alphabet[acc];

    }

    return output;
}

public void GenerateStats(char[] input)
{
    string temp = input.ToString();
    int jum = 0;

    // A - E
    for (int i = 0; i < input.Length; i++)
    {
        if (input[i].ToString() == "A") { jum++; }
    }
    textBox3.Text = jum.ToString();
    jum = 0;
    for (int i = 0; i < input.Length; i++)
    {
        if (input[i].ToString() == "B") { jum++; }
    }
    textBox4.Text = jum.ToString();
    jum = 0;
    for (int i = 0; i < input.Length; i++)
    {
        if (input[i].ToString() == "C") { jum++; }
    }
    textBox5.Text = jum.ToString();
    jum = 0;
    for (int i = 0; i < input.Length; i++)
    {
        if (input[i].ToString() == "D") { jum++; }
```

```
}
textBox6.Text = jum.ToString();
jum = 0;
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "E") { jum++; }
}
textBox7.Text = jum.ToString();
jum = 0;



// F - K
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "F") { jum++; }
}
textBox12.Text = jum.ToString();
jum = 0;
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "G") { jum++; }
}
textBox11.Text = jum.ToString();
jum = 0;
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "H") { jum++; }
}
textBox10.Text = jum.ToString();
jum = 0;
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "I") { jum++; }
}
textBox9.Text = jum.ToString();
jum = 0;
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "K") { jum++; }
}
textBox8.Text = jum.ToString();
jum = 0;

//
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "L") { jum++; }
}
textBox17.Text = jum.ToString();
jum = 0;
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "M") { jum++; }
}
textBox16.Text = jum.ToString();
jum = 0;
```

```
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "N") { jum++; }
}
textBox15.Text = jum.ToString();
jum = 0;
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "O") { jum++; }
}
textBox14.Text = jum.ToString();
jum = 0;
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "P") { jum++; }
}
textBox13.Text = jum.ToString();
jum = 0;

// q - u
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "Q") { jum++; }
}
textBox22.Text = jum.ToString();
jum = 0;
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "R") { jum++; }
}
textBox21.Text = jum.ToString();
jum = 0;
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "S") { jum++; }
}
textBox20.Text = jum.ToString();
jum = 0;
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "T") { jum++; }
}
textBox19.Text = jum.ToString();
jum = 0;
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "U") { jum++; }
}
textBox18.Text = jum.ToString();
jum = 0;

// v - z
for (int i = 0; i < input.Length; i++)
{
    if (input[i].ToString() == "V") { jum++; }
}
textBox27.Text = jum.ToString();
```

```
        jum = 0;
        for (int i = 0; i < input.Length; i++)
        {
            if (input[i].ToString() == "W") { jum++; }
        }
        textBox26.Text = jum.ToString();
        jum = 0;
        for (int i = 0; i < input.Length; i++)
        {
            if (input[i].ToString() == "X") { jum++; }
        }
        textBox25.Text = jum.ToString();
        jum = 0;
        for (int i = 0; i < input.Length; i++)
        {
            if (input[i].ToString() == "Y") { jum++; }
        }
        textBox24.Text = jum.ToString();
        jum = 0;
        for (int i = 0; i < input.Length; i++)
        {
            if (input[i].ToString() == "Z") { jum++; }
        }
        textBox23.Text = jum.ToString();
        jum = 0;


    } // membuat statistik dari huruf yang terbentuk

    public char[,] CreateKeyTable(char[] input)
    {
        string alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ";
        char[] alpArray = alphabet.ToCharArray();
        string temp = "";
        char[] tmpArray;


        char[,] output = new char[5, 5];


        for (int i = 0; i < input.Length; i++)
        {
            if (input[i].ToString().ToUpper() == "J")
            {
                input[i] = 'I';
            }
            if (!temp.Contains(input[i].ToString().ToUpper()))
            {
                temp = temp + input[i].ToString().ToUpper();
            }
        }

        for (int j = 0; j < alpArray.Length; j++)
        {
            if (!temp.Contains(alpArray[j].ToString().ToUpper()))
            {
                temp = temp + alpArray[j].ToString().ToUpper();
```

```
                }
            }

            tmpArray = temp.ToCharArray();


            int h = 0;

            for (int x = 0; x < 5; x++)
            {
                for (int y = 0; y < 5; y++)
                {
                    output[x, y] = tmpArray[h++];
                }
            }

            return output;
        }  // membentuk tabel 5 x 5 dari kunci

        public char[] BeforeEncrypt(char[] input)
        {

            int pjgInp = input.Length;
            char[] output = new char[1024];

            int j = 0;

            for (int i = 0; i < pjgInp; i++)
            {

                if (char.IsLetter(input[i])) // dia karakter
                {
                    if (input[i] == 'J')
                    {
                        input[i] = 'I';
                    }
                    if (i == 0)
                    {
                        output[j++] = input[i];
                    }
                    else
                    {
                        if ((input[i] == output[j - 1]) && (j % 2 !=
0))
                        {
                            output[j++] = 'Z';
                            output[j++] = input[i];
                        }
                        else
                        {

                            output[j++] = input[i];
                        }
                    }

                }
```

```
            }

            if (j % 2 != 0)
            {
                output[j] = 'Z';
            }

            char[] hasil;
            if (j % 2 == 0)
            {
                hasil = new char[j];
            }
            else
            {
                hasil = new char[j + 1];
            }


            for (int asal = 0; asal < hasil.Length; asal++)
            {
                hasil[asal] = output[asal];
            }


            return hasil;

        }  // mempersiapkan input

}
```