Polygraphic Alphabetic Chiper

Satya Pratama Kadranyata [1] **13504147** [2]
1) Departemen Teknik Informatika Institut Teknologi Bandung Jl. Ganeca 10
2) if14147@students.if.itb.ac.id

Abstract – *Perkembangan arus lalu lintas pengiriman data saat ini semakin pesat. Ini disebabkan kebutuhan akan berbagai macam data dengan ukuran, jenis dan tingkat kerahasiaan tertentu. Data tersebut adakalanya harus diperoleh dari pihak lain. Untuk menjaga data, dikembangkanlah sebuah seni untuk menyembunyikan dan mempertahankan kerahasiaan data yang disebut dengan Kriptografi. Kriptografi pertama kali dikembangkan disebut Kriptografi Klasik. Algoritma-algoritma Kriptografi klasik banyak yang berbasis karakter sehingga penulis mencoba untuk memberikan sebuah algoritma baru yang bernama Polygraphic Alphabetic Chiper. Algoritma ini termasuk dalam Chiper Substitusi dan hanya dapat digunakan terhadap file dengan ekstensi txt.*

Kata Kunci*: Kriptografi klasik, Chiper Substitusi, Polygraphic Alphabetic Chiper, file .txt.*

1. PENDAHULUAN

Kriptografi awal yang dikembangkan dan digunakan disebut Kriptografi Klasik. Hampir seluruh Algoritma pada Kriptografi Klasik adalah algoritma yang berbasis karakter. Maksudnya adalah algoritma-algoritma ini memiliki fokus dalam mengolah karakter sehingga data atau pesan tetap dapat dijaga kerahasiaannya. Salah satu klasifikasi dari Algoritma Kriptografi Klasik adalah *Chiper Substitusi* [1]. Dalam Chiper Substitusi ini karakter yang ada di data maupun pesan akan digantikan oleh karakter lain. Dengan hal tersebut maka hanya pihak yang memiliki kunci saja yang dapat mengetahui arti dari data maupun pesan yang dikirimkan.

Algoritma Polygraphic Alphabetic Chiper yang akan dibahas oleh penulis dalam makalah ini termasuk dalam *Chiper Substitusi*. Akan tetapi penulis mencantumkan beberapa pemikiran dan teknik agar data maupun pesan menjadi semakin sulit untuk dipecahkan. Ide ini adalah menggunakan matriks untuk menjadi sebuah kunci dan kunci baru dapat diciptakan setelah pesan atau data yang akan dikirimkan telah berhasil dibuat. Diharapkan dengan menggunakan teknik ini maka biaya dan waktu yang digunakan untuk memecahkan kerahasiaan teknik ini akan semakin besar. Ini tentunya memenuhi salah satu syarat untuk menjadi algoritma yang kuat dan penulis berharap algoritma ini dapat digunakan untuk khalayak umum dalam bidang Kriptografi.

2. ALGORITMA

Algoritma yang akan dibahas dalam makalah ini menggunakan *pseudo code*. Algoritma yang diajukan diharapkan dapat diimplementasikan tanpa bergantung dari suatu jenis bahasa pemrograman tertentu.

Langkah pertama dari algoritma Polygraphic Alphabetic Chiper adalah mendeklarasikan 3 buah larik *global*. Larik yang pertama merupakan larik yang digunakan untuk menyimpan seluruh karakter yang akan di-*substitusi.* Larik kedua merupakan matriks yang digunakan untuk menyimpan kunci. Pada kasus ini larik pertama akan menggunakan karakter abjad A sampai dengan Z ditambah dengan <spasi> dan matriks yang digunakan adalah matriks 27x6.

```
arrChar : array[1..27] of character

key : array[1..27] of array[1..6] of
character
```

Setelah dideklasari, maka kita menginisialisasi larik yang pertama dengan false dan matriks dengan ‘ ’.

```
procedure initKey(input/output key :
array of array of character)
 deklarasi
   i : integer
   j : integer
 algoritma
   for ( i ← 1, i <= 27, i ← i + 1)
    for (j ← 1, j <= 6, j ← j + 1)
      key[i][j] ← ‘ ’
    endfor
   endfor
```

Pendeklarasian variabel yang bersifat *global* telah dilakukan maka Enkripsi dan Dekripsi siap dilakukan.

2.1. Enkripsi

Enkripsi dilakukan setelah menerima masukan *plaintext. Plaintext* ini akan diumpamakan sebagai sebuah larik dengan ukuran sama dengan ukuran .

Misalkan plaintext adalah :

Pasukan yang dipimpin Ratu Velope Flexia telah sampai di Zurich.

```
plainLength : integer{panjang plaintext}

plaintext : array [1..plainLength] of character {plaintext}
```

Dari contoh di atas maka plainLength = 64 dan *plaintext* adalah

P	a	s	u	k	a	n		y	a	n	G		d	i
p	i	m	p	i	n		R	a	t	u		V	e	l
l	o	p	e		F	l	e	x	i	a		t	e	l
l	a	h		s	a	m	p	a	i		D	i		
Z		u		R		I		c		H		.		

Setelah kita mendapatkan *plaintext*, maka kita dapat membuat kunci untuk algoritma ini.

Di awal pembuatan kunci, kita harus memastikan bahwa karakter yang dimasukan dalam kunci.

```
function isElmtKey( hrf : character, arrChar : array of character ) → boolean
  deklarasi
     i : integer

  algoritma
     i ← 1
     while (hrf != arrChar[i] and i <= 27)
       i ← i + 1
     endwhile
     if (hrf = arrChar[i])
       → true
     else
       → false
```

Penciptaan kunci di jelaskan dalam algoritma di bawah ini.

```
procedure makeKey(input plaintext : array of character, input/output key : array of array of character, input plainLength : integer, arrChar : array of character, input arrBoolChar : array of boolean)
  deklarasi
    i : integer
    brs : integer {baris}
    cBrs : integer {current baris}
    klm : integer
  algoritma
    cBrs ← 1
    for (i ← 1, i <= plainLength, i ← i + 1)
      if (isElmtKey(plaintext[i], arrChar))
        if (not isExist(plaintext[i], key))
          brs ← cBrs
          for (klm ← 1, klm <= 6, klm ← klm + 1)
            key[brs][klm] ← plaintext[i]
            if (brs = 27)
              brs ← 1
            else
              brs  ← brs + 1
            endif
          endfor
          cBrs ← cBrs + 1
        endif
      endif
    endfor
    if (cBrs != 28)
       completeKey(cBrs, arrChar, key)

function isExist( hrf : character, key : array of array of character ) → boolean
  deklarasi
     i : integer

  algoritma
     i ← 1
     while (hrf != key[i][1] and i <= 27)
       i ← i + 1
     endwhile
     if (hrf = key[i][1])
       → true
     else
       → false

procedure completeKey(input n : integer, input arrChar : array of character, input key : array of array of character)
  deklarasi
    i : integer
    cBrs : integer
    brs : integer
    klm : integer
  algoritma
    cBrs ← n
    for ( i ← 1, i<= 27, i++)
      if (not isExist(arrChar[i], key))
        brs ← cBrs
        for (klm ← 1, klm <= 6, klm ← klm + 1)
          key[brs][klm] ← plaintext[i]
          if (brs = 27)
            brs ← 1
          else
            brs ← brs + 1
        endfor
        cBrs ← cBrs + 1
```

```
    endif
  endfor
```

Hasil dari pembuatan key dari plaintext adalah

p	q	j	B	c	Z
a	p	q	J	b	C
s	a	p	Q	j	B
u	s	a	P	q	J
k	u	s	A	p	Q
n	k	u	S	a	p
<spasi>	n	k	U	s	a
y	<spasi>	n	K	u	s
g	y	<spasi>	N	k	u
d	g	y	<spasi>	n	k
i	d	g	Y	<spasi>	n
p	I	d	G	y	<spasi>
m	p	i	D	g	y
r	m	p	I	d	g
t	R	m	P	i	d
v	T	r	M	p	i
e	v	t	R	m	p
l	e	v	T	r	m
o	L	e	V	t	r
f	o	l	E	v	t
x	f	o	L	e	v
h	x	f	O	l	e
z	h	x	F	o	l
c	z	h	X	f	o
b	c	z	H	x	f
j	b	c	Z	h	x
q	j	b	C	z	h

Enkripsi akan dilakukan menggunakan kunci yang ada dan menghasilkan *chipertext* dengan panjang = 2 x panjang *plaintext*.

```
chipLength : integer

chipLength ← 2*plainLength

chipertext : array[1…chipLength] of
character
```

Algoritma Enkripsi dijelaskan di bawah ini.

```
procedure enkripsi( input
plainLength : integer, input key :
array of array of character, output
chipertext : array of character,
input plaintext : array of
character)
  deklarasi
    i : integer
    j : integer
    brsKey : integer
    klmKey : integer
  algoritma
    j ← 1
    for (i ← 1, i <= plainLength, i
←i+1)
      klmKey ← i mod 6
      if (klmKey  = 0)
       klmKey ← 6
      endif
      brsKey ←
searchBrsKey(plaintext[i], key)
      if (brsKey <= 14)

        if (klmKey <= 3)

          if (key[brsKey+1][klmKey]
= <spasi>)
            plaintext[j] ←
key[brsKey][klmKey]
          else
            plaintext[j] ←
key[brsKey+1][klmKey]
          endif

          if (key[brsKey][klmKey+1]
= <spasi>)
            plaintext[j+1] ←
key[brsKey + 1][klmKey + 1]
          else
            plaintext[j+1] ←
key[brsKey][klmKey+1]
          endif


        else
           {klmKey > 3}

          if (key[brsKey+1][klmKey]
= <spasi>)
            plaintext[j] ←
key[brsKey][klmKey]
          else
            plaintext[j] ←
key[brsKey+1][klmKey]
          endif

          if (key[brsKey][klmKey-1]
= <spasi>)
            plaintext[j+1] ←
key[brsKey - 1][klmKey - 1]
          else
            plaintext[j+1] ←
key[brsKey][klmKey-1]
          endif

        endif
      else
        {brsKey > 14}
        if (klmKey <= 3)

          if (key[brsKey-1][klmKey]
= <spasi>)
            plaintext[j] ←
```

```
key[brsKey][klmKey]
          else
            plaintext[j1] ←
key[brsKey-1][klmKey]
          endif

          if (key[brsKey][klmKey+1]
= <spasi>)
            plaintext[j+] ←
key[brsKey - 1][klmKey + 1]
          else
            plaintext[j+] ←
key[brsKey][klmKey+1]
          endif


        else
           {klmKey > 3}
          if (key[brsKey-1][klmKey]
= <spasi>)
            plaintext[j] ←
key[brsKey][klmKey]
          else
            plaintext[j] ←
key[brsKey-1][klmKey]
          endif

          if (key[brsKey][klmKey-1]
= <spasi>)
            plaintext[j+1] ←
key[brsKey - 1][klmKey - 1]
          else
            plaintext[j+1] ←
key[brsKey][klmKey-1]
          endif
        endif
      endif
      j ← j + 2
    endfor
```

Hasil enkripsi adalah :

Aqspuaksnupsnkyngyssnygggnigpdmmpdrpmipdnkyy
ttssvrksynttvveellaqvvnyvxeevvffpppsnyvrvveessxzny
vasprpmmssppynigpdyyhckktmpdzzxz.

2.2. Dekripsi

Algoritma dekripsi akan dijelaskan di bawah ini

```
procedure dekripsi(input chipLength
: integer, input chipertext : array
of character,  input key : array of
array of character, output plaintext
: array of character)
   deklarasi
      i : integer
      j : integer
      k: integer
      l : integer
      m : integer
      brsKey : integer
   algoritma
      j ← 1
      for (i←1, i<=chipLength,
i←i+2)
        l ←
searchBrsKey(chipertext[i])
        m ←
searchBrsKey(chipertext[i+1])
        chipertext[j] ← key[l][m]
      endfor
```

3. HASIL DAN PEMBAHASAN

Algoritma ini termasuk salah satu yang sulit dipecahkan sebab untuk sebuah karakter aka digantikan oleh dua buah karakter.

Untuk kasus ini kemungkinan sebuah karakter yang dapat digantikan adalah karakter yang lain adalah 2 * 27! sebab karakter tersebut masih dapat digantikan oleh sepasang karakter yang ada kemungkinan sama.

Dalam menghadapi serangan analisis statistic tak dapat dilakukan sebab kemungkinan karakter yang banyak muncul seperti A, E dll akan dapat digantikan dengan enam pasang kemungkinan.

Salah satu kelemahan dari algoritma adalah hanya dapat men-substitusi karakter-karakter abjad. Selain itu apabila diberikan plaintext yang huruf sama semua seperti aaaaaaaaaaa, maka akan terjadi pengulangan.

4. KESIMPULAN

Walaupun algoritma ini memiliki kelemahan, algoritma ini dapat dijadikan sebagai bahan masukan bagi perkembangan algoritma klasik.

DAFTAR REFERENSI

[1] Munir, Rinaldi, “Kriptografi”, 2005.