# Studi Mengenai Optimasi Pemrosesan Vigenere Cipher pada Bahasa Pemrograman Java

Archie Anugrah / 13508001
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*if18001@students.if.itb.ac.id*

***Abstrak*—Proses Vigenere Cipher dikarakteristikkan oleh proses penggantian karakter-karakter yang bersesuaian dengan karakter yang hendak dienkripsi. Dalam makalah ini, akan diujicobakan beberapa teknik pemrosesan karakter, baik dengan menggunakan rumus, tabel, ataupun *List*. Selain itu, diujikan pula pengaruh konversi *string* dari *plain text* dan kunci. Dari hasil uji coba, diharapkan didapatkan teknik apa yang paling cocok untuk digunakan pada algoritma Vigenere.**

***Kata Kunci*—Vigenere, Tabel Substitusi, Konversi String.**

## I. PENDAHULUAN

Vigenere Cipher adalah salah satu jenis kriptografi klasik yang paling umum digunakan dalam teknik penyamaran pesan. Kepopuleran penggunaan Vigenere ini didasarkan pada kemudahan pembuatan, kesederhanaan tabel substitusi, dan kekuatan perlindungan yang cukup baik.

Ada beberapa teknik yang dapat digunakan untuk membentuk tabel substitusi, yaitu : input manual ataupun menggunakan kalkulasi. Masing-masing cara tentu memiliki kelebihan dan kekurangannya masing-masing. Tetapi, ada satu hal yang jarang ditilik mengenai hal ini, yaitu mengenai kecepatan proses masing-masing algoritma. Dalam benak saya, saya memiliki hipotesis, seharusnya untuk Vigenere Plain Text yang panjang, pembentukan tabel terlebih dahulu seharusnya lebih cepat dibanding dengan substitusi langsung dengan menggunakan rumus (dalam bahasa pemrograman modern). Tidak berhenti sampai situ saja, bahasa pemrograman modern juga sudah memiliki struktur data yang sudah dibentuk sedemikian rupa sehingga efektif digunakan yang harus dipertimbangkan pula.

Untuk pengujian bahasa pemrograman modern, digunakan bahasa Java. Java digunakan karena pertimbangan bahwa Java adalah bahasa pemrograman yang cukup modern namun cepat dalam hal eksekusi. Selain itu pula, Java sudah menyediakan berbagai fungsi yang dapat memudahkan pembuatan program uji.

Mengenai Vigenere Cipher yang digunakan sendiri, saya menggunakan Vigenere Cipher 26 karakter (algoritma klasik), dengan anggapan bahwa 26 karakter ini pun sudah cukup menggambarkan keadaan Vigenere Cipher pada umumnya.

Semoga hasil penelitian ini, walaupun sederhana dapat memberikan bantuan yang berarti bagi orang-orang yang membutuhkannya.

## II. PEMROSESAN RUMUS

Pada bagian ini, dibuat program uji yang memberikan hasil vigenere cipher dengan menggunakan rumus sederhana yaitu :

```
(PlainChar + KeyChar) mod 26
```

Dimana nilai char berkisar dari angka 0-25 (hanya alphabet).

Berikut kode dari program yang digunakan :

```
   String EncryptVigenereRumus(String plain, String key,
boolean alphabetonly) {
           //initialize variable
           String Temp = ""; //for returned String
           char TempChar; //for calculate the encrypted
character
           int KeyPositionCounter = 0; //for calculating
the position of the text

           for (int i = 0; i < plain.length(); i++)
//iterate the whole text
           {
               if (plain.charAt(i) == ' ') //if it is
space
               {
                   Temp = Temp + ' '; //Pad the temp
text with space
               } else {
                   if (alphabetonly) //if alphabet mode
                   {
                       TempChar                        =
inttoCharAlphabetMode((charToIntAlphabetMode(plain.char
At(i))                                                 +
charToIntAlphabetMode(key.charAt(KeyPositionCounter    %
key.length()))) % 26);


                   } else //if extended mode
                   {
                       TempChar         =         (char)
((plain.charAt(i)   +   key.charAt(KeyPositionCounter   %
key.length())) % 256);
                   }
                   ++KeyPositionCounter;//add        the
counter
                   Temp = Temp + TempChar; //Pad the
temp text
               }
           }
           return Temp;
    }
```

Pemrosesan dengan menggunakan rumus ini adalah

cara yang paling umum digunakan oleh para *programmer* untuk membuat Vigenere Cipher. Hal ini disebabkan karena proses pembuatan yang cepat dan mudah.

## III. PEMROSESAN TABEL

Vigenere Cipher adalah algoritma klasik yang pada awalnya merupakan tabel sederhana dari perubahan huruf-huruf berdasarkan kunci tertentu. Pada bagian ini, saya ingin menguji kecepatan program bila, tabel sudah tersedia di dalam program dan metoda penggantian karakter menggunakan metoda pencocokan tabel. Menurut hipotesa seharusnya kecepatan pembandingan *array* bisa lebih cepat dibandingkan proses kalkulasi untuk jumlah data yang besar.

Berikut kode program yang digunakan :

```
static String EncryptVigenereTabel(String plain, String
key, boolean alphabetonly, int[][] Tabel) {
          //initialize variable
          String Temp = ""; //for returned String
          char TempChar; //for calculate the encrypted
character
          int KeyPositionCounter = 0; //for calculating
the position of the text

          for (int i = 0; i < plain.length(); i++)
//iterate the whole text
          {
              if (plain.charAt(i) == ' ') //if it is
space
              {
                  Temp = Temp + ' '; //Pad the temp
text with space
              } else {
                  if (alphabetonly) //if alphabet mode
                  {
                      TempChar                        =
inttoCharAlphabetMode(Tabel[charToIntAlphabetMode(plain
.charAt(i))][charToIntAlphabetMode(key.charAt(KeyPositi
onCounter % key.length()))]);


                  }
                  ++KeyPositionCounter;//add        the
counter
                  Temp = Temp + TempChar; //Pad the
temp text
              }
          }
          return Temp;
    }
```

Dalam kasus ini, ada proses tambahan yang harus dilakukan oleh program, yaitu membuat tabel terlebih dahulu. Setelah tabel terbentuk, kita dapat mengambil tabel pada posisi ke [*plain*][key] dan mengganti *plain text* dengan *cipher text* yang bersesuaian

## IV. PEMROSESAN ARRAY LIST

Pada bahasa pemrograman modern, seperti Java terdapat berbagai tipe bentukan standar yang telah tersedia dan siap untuk digunakan. Struktur data ini telah dirancang sedemikian rupa supaya dapat bekerja dengan efisien. Oleh karena itu saya penasaran untuk menguji kecepatan pemrosesan jika array pada pengujian sebelumnya diganti dengan menggunakan *Array List* 2 dimensi.

Berikut kode program yang digunakan :

```
String EncryptVigenereList(String plain, String key,
boolean alphabetonly, List<List<Integer>> Tabel) {
          //initialize variable
          String Temp = ""; //for returned String
          char TempChar; //for calculate the encrypted
character
          int KeyPositionCounter = 0; //for calculating
the position of the text

          for (int i = 0; i < plain.length(); i++)
//iterate the whole text
          {
              if (plain.charAt(i) == ' ') //if it is
space
              {
                  Temp = Temp + ' '; //Pad the temp
text with space
              } else {
                  if (alphabetonly) //if alphabet mode
                  {

                      TempChar                        =
inttoCharAlphabetMode(Tabel.get(charToIntAlphabetMode(p
lain.charAt(i))).get(charToIntAlphabetMode(key.charAt(K
eyPositionCounter % key.length()))).intValue() );

                  } else //if extended mode
                  {
                      TempChar         =         (char)
((plain.charAt(i)   +   key.charAt(KeyPositionCounter   %
key.length())) % 256);
                  }
                  ++KeyPositionCounter;//add        the
counter
                  Temp = Temp + TempChar; //Pad the
temp text
              }
          }
          return Temp;
    }
```

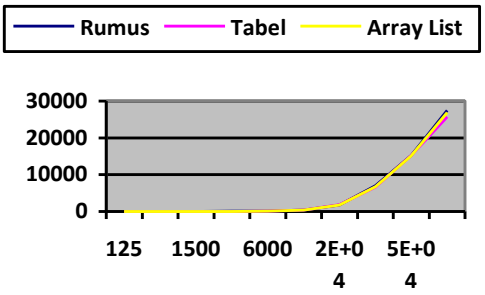
Dalam kasus ini, ada proses tambahan yang harus dilakukan oleh program, sebagaimana pada proses sebelumnya, yaitu membuat Array List terlebih dahulu. Setelah List terbentuk, kita dapat mengambil List pada posisi ke [*plain*][*key*] dan mengganti *plain text* dengan *cipher text* yang bersesuaian.

## V. PENGUJIAN

Sebagaimana hipotesa awal, bahwa pada pengujian kali ini, ingin dibuktikan bahwa untuk jumlah text yang besar, penggunaan tabel dapat memercepat jalannya proses vigenere, maka akan digunakan beberapa data uji yang berbeda-beda. Pada pengujian ini digunakan kunci sederhana yang sama, yaitu : kripto. Nilai milisekon dalam data berikut didapatkan menggunakan fungsi *currentTimeMillis* dalam Java dan diambil sampel data kedua pada setiap panjang karakter (karena setiap proses memiliki perbedaan sedikit dalam waktu proses). Pada pengukuran ini, waktu pembuatan tabel tidak dihitung karena, tabel nantinya akan dibuat pada awal program dan tidak perlu dibuat pada setiap kali proses.

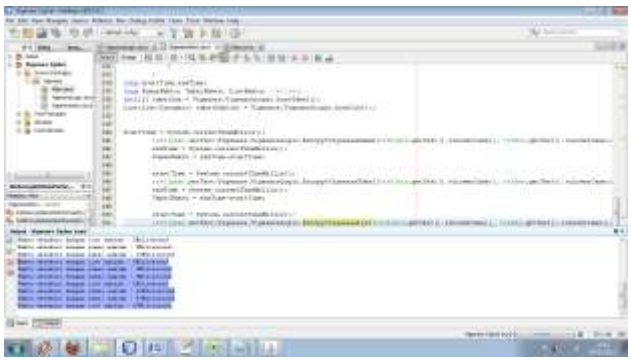
- 125 karakter :
  - Rumus : 0 Milisekon
  - Tabel : 0 Milisekon
  - Array List : 0 Milisekon
- 750 karakter :
  - Rumus : 0 Milisekon
  - Tabel : 0 Milisekon
  - Array List : 0 Milisekon
- 1500 karakter :
  - Rumus : 20 Milisekon

    - Tabel : 10 Milisekon
    - Array List : 10 Milisekon
- 3750 karakter :
    - Rumus : 62 Milisekon
    - Tabel : 47 Milisekon
    - Array List : 47 Milisekon
- 6000 karakter :
    - Rumus : 125 Milisekon
    - Tabel : 80 Milisekon
    - Array List : 90 Milisekon
- 12000 karakter :
    - Rumus : 410 Milisekon
    - Tabel : 350 Milisekon
    - Array List : 370 Milisekon
- 24000 karakter :
    - Rumus : 1792 Milisekon
    - Tabel : 1844 Milisekon
    - Array List : 1740 Milisekon
- 38354 karakter :
    - Rumus : 6984 Milisekon
    - Tabel : 6626 Milisekon
    - Array List : 6746 Milisekon
- 48000 karakter :
    - Rumus : 15216 Milisekon
    - Tabel : 15178 Milisekon
    - Array List : 15102 Milisekon
- 57600 karakter :
    - Rumus : 27498 Milisekon
    - Tabel : 25756 Milisekon
    - Array List : 26894 Milisekon




*Bagan 1. Bagan perbandingan waktu eksekusi*

*Gambar 1.Teks uji*



*Gambar 2. Hasil uji pada NetBeans*

Dari hasil perngujian, didapatkan bahwa kecepatan perbandingan tabel lebih cepat dibandingkan penggunaan rumus. Bahkan hasil yang agak mengejutkan untuk saya adalah ternyata penggunaan arraylist tetap lebih cepat dibandingkan penggunaan rumus. Dari hasil penggambaran graf terlihat bahwa gradien kurva rumus lebih besar dibandingkan kurva tabel, sehingga untuk jumlah data yang besar, **kita bisa menarik kesimpulan bahwa penggunaan konstruksi tabel lebih cepat dibandingkan penggunaan rumus**.

Walaupun hasil pengujian membuktikan bahwa penggunaan tabel bisa lebih cepat daripada rumus, kita harus memerhatikan bahwa bahkan dititik 57600 karakter saja, perbedaan kedua cara hanya sekitar 300 Milisekon. Walaupun secara angka tidak terlihat banyak, tetapi sebenarnya jumlah itu sangatlah besar. Rata-rata karakter tiap halaman dapat menampung 3818 karakter maksimal dengan menggunakan *Font Times New Roman* 12. Berarti 57600 karakter berarti sama saja dengan menggunakan 15 lembar kertas standar sampai penuh, yang pastinya tidak pernah terjadi di kondisi praktis (tetapi kita gunakan dalam asumsi untuk mempermudah). Untuk mendapatkan jeda sekitar 10 detik (1000 Milisekon), berarti kita membutuhkan : (1000/300) X 57600 = 192000 karakter atau sekitar 50 lembar penuh. Berarti hal ini mendukung hipotesa bahwa penggunaan tabel substitusi baik untuk Vigenere dengan *plain text* besar.

## VI. PEMROSESAN PLAINTEXT

Selain masalah substitusi, ada 1 hal lagi yang dapat memengaruhi kecepatan proses pada Vigenere, hal itu adalah bagaimana cara plaintext diacu oleh program. Sebagaimana pengujian hipotesa sebelumnya, pada kasus ini, saya memiliki hipotesa bahwa jika plaintext dimasukkan kedalam array terlebih dahulu, dibandingkan diacu langsung dari string, maka pemrosesan plain text akan jauh lebih cepat.

Untuk pengujian ini, digunakan Vigenere yang diproses menggunakan rumus dengan kunci “kripto”. Pada pengujian ini, sampel waktu yang diambel adalah sampel kedua dari setiap percobaan, sebagaimana pengujian sebelumnya. Hal ini dialkukan karena biasanya pada pengujian pertama, *setting up* program sering mengganggu hasil pengukuran. Pada pengukuran ini,

waktu pembuatan tabel dihitung karena tabel nantinya akan dibuat pada setiap kali proses (plain text tiap proses berbeda).

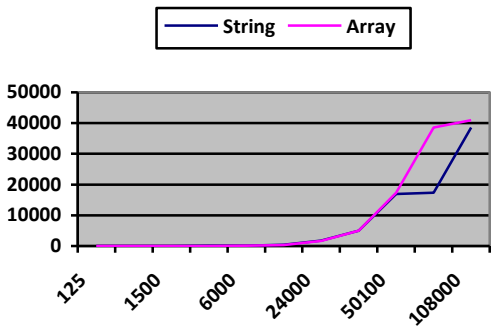
Berikut kode dari program yang digunakan (perbedaan kedua kode hanyalah apakah pada awal eksekusi, *string* diubah terlebih dahulu menjadi *array of char* atau tidak) :

```
String EncryptVigenereString(String plain, String key, boolean alphabetonly) {
          //initialize variable
          long startTime,endTime;
          long StringWaktu; //milisec

   startTime = System.currentTimeMillis();
          String Temp = ""; //for returned String
          char TempChar; //for calculate the encrypted character
          int KeyPositionCounter = 0; //for calculating the position of the text

          for (int i = 0; i < plain.length(); i++) //iterate the whole text
          {
              if (plain.charAt(i) == ' ') //if it is space
              {
                  Temp = Temp + ' '; //Pad the temp text with space
              } else {
                  if (alphabetonly) //if alphabet mode
                  {
                      TempChar                          = inttoCharAlphabetMode((charToIntAlphabetMode(plain.charAt(i))                                                    + charToIntAlphabetMode(key.charAt(KeyPositionCounter    % key.length())))) % 26);

                  } else //if extended mode
                  {
                      TempChar          =          (char) ((plain.charAt(i)   +   key.charAt(KeyPositionCounter   % key.length()))) % 256);
                  }
                  ++KeyPositionCounter;//add        the counter
                  Temp = Temp + TempChar; //Pad the temp text
              }
          }
           endTime = System.currentTimeMillis();
           StringWaktu = endTime-startTime;
           System.out.println("Waktu   eksekusi   dengan string adalah : "+ StringWaktu + " Milisecond");
          return Temp;
      }
```

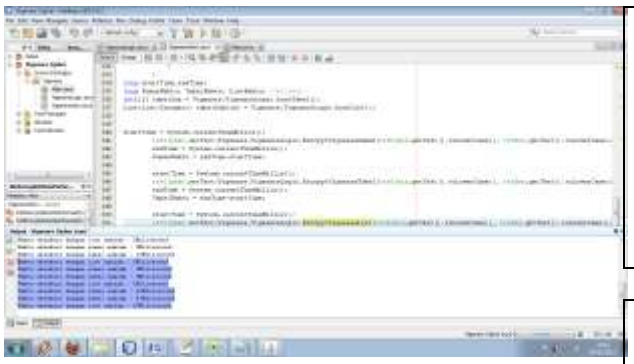
```
String EncryptVigenerechar(String plain, String key, boolean alphabetonly) {
          //initialize variable
          long startTime,endTime;
          long  CharWaktu; //milisec
          char plains[] = plain.toCharArray();
          startTime = System.currentTimeMillis();
          String Temp = ""; //for returned String
          char TempChar; //for calculate the encrypted character
          int KeyPositionCounter = 0; //for calculating the position of the text

          for (int i = 0; i < plain.length(); i++) //iterate the whole text
          {
              if (plain.charAt(i) == ' ') //if it is space
              {
                  Temp = Temp + ' '; //Pad the temp text with space
              } else {
                  if (alphabetonly) //if alphabet mode
                  {
                      TempChar                          = inttoCharAlphabetMode((charToIntAlphabetMode(plains[i]) + charToIntAlphabetMode(key.charAt(KeyPositionCounter % key.length()))) % 26);
                  } else //if extended mode
                  {
                      TempChar          =          (char) ((plain.charAt(i)   +   key.charAt(KeyPositionCounter   % key.length()))) % 256);
                  }
                  ++KeyPositionCounter;//add        the counter
                  Temp = Temp + TempChar; //Pad the temp text
              }
          }
           endTime = System.currentTimeMillis();
           CharWaktu = endTime-startTime;
           System.out.println("Waktu   eksekusi   dengan array char adalah : "+ CharWaktu + " Milisecond");
          return Temp;
      }
```

Berikut hasil pengujian yang dilakukan :

- 125 karakter :
  - String : 0 Milisekon
  - Array : 0 Milisekon
- 750 karakter :
  - String : 0 Milisekon
  - Array : 0 Milisekon
- 1500 karakter :
  - String : 15 Milisekon
  - Array : 16 Milisekon
- 3000 karakter :
  - String : 63 Milisekon
  - Array : 47 Milisekon
- 6000 karakter :
  - String : 109 Milisekon
  - Array : 110 Milisekon
- 12000 karakter :
  - String : 437 Milisekon
  - Array : 358 Milisekon
- 24000 karakter :
  - String : 1731 Milisekon
  - Array : 1701 Milisekon
- 35500 karakter :
  - String : 4976 Milisekon
  - Array : 4977 Milisekon
- 50100 karakter :
  - String : 16941 Milisekon
  - Array : 17379 Milisekon
- 64700 karakter :
  - String : 17379 Milisekon
  - Array : 38548 Milisekon
- 108000 karakter :
  - String : 38548 Milisekon
  - Array : 40919Milisekon




*Bagan 2. Bagan perbandingan*



*Gambar 3. Pengujian pada NetBeans*

Dari hasil pengujian didapatkan bahwa, pada jumlah *plain text* yang kecil, algoritma penggunaan array sedikit lebih cepat dibandingkan yang menggunakan string. Tetapi, semakin besar *plain text*, semakin lama proses konversi *string* ke *array of char* sehingga waktu proses yang menggunakan array akhirnya menjadi lebih lama dibandingkan yang langsung menggunakan *string*.

## VII. PEMROSESAN KUNCI

Pada kasus sebelumnya, kita menguji apakah penggunaan array pada cipher text dapat memercepat proses ataukah tidak dan kita mendapatkan bahwa penggunaan *array* memercepat proses tetapi proses konversi *string* menjadi *array of char* terlalu lama. Hal ini membawa saya ke suatu pertanyaan, bagaimana bila kunci yang dikonversi menjadi *array* ? Panjang kunci pada Vigenere bisa dibilang pendek dan dapat dikonversi dengan cepat. Hal ini membawa saya pada satu hipotesa bahwa konversi teks kunci menjadi *array of char* dapat memercepat waktu eksekusi program.

Pada pengujian ini, digunakan kode :

```
String EncryptVigenereString(String plain, String key,
boolean alphabetonly) {
          //initialize variable
          long startTime,endTime;
          long StringWaktu; //milisec

   startTime = System.currentTimeMillis();
          String Temp = ""; //for returned String
          char TempChar; //for calculate the encrypted
character
          int KeyPositionCounter = 0; //for calculating
the position of the text

          for (int i = 0; i < plain.length(); i++)
//iterate the whole text
          {
              if (plain.charAt(i) == ' ') //if it is
space
              {
                  Temp = Temp + ' '; //Pad the temp
text with space
              } else {
                  if (alphabetonly) //if alphabet mode
                  {
                      TempChar                        =
inttoCharAlphabetMode((charToIntAlphabetMode(plain.char
At(i))                                                +
charToIntAlphabetMode(key.charAt(KeyPositionCounter    %
key.length()))) % 26);

                  } else //if extended mode
                  {
                      TempChar         =         (char)
((plain.charAt(i)   +   key.charAt(KeyPositionCounter   %
key.length())) % 256);
                  }
                  ++KeyPositionCounter;//add        the
counter
                  Temp  =  Temp  +  TempChar;  //Pad  the
temp text
              }
          }
           endTime = System.currentTimeMillis();
           StringWaktu = endTime-startTime;
           System.out.println("Waktu   eksekusi   dengan
string adalah : "+ StringWaktu + " Milisecond");
          return Temp;
      }
```

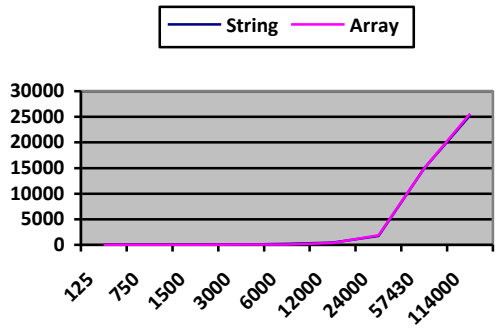
```
String  EncryptVigenerechar(String  plain,  String  key,
boolean alphabetonly) {
          //initialize variable
          long startTime,endTime;
          long  CharWaktu; //milisec
          char keys[] = key.toCharArray();
   startTime = System.currentTimeMillis();
          String Temp = ""; //for returned String
          char TempChar; //for calculate the encrypted
character
          int KeyPositionCounter = 0; //for calculating
the position of the text

          for  (int  i  =  0;  i  <  plain.length();  i++)
//iterate the whole text
          {
              if  (plain.charAt(i)  ==  ' ')  //if  it  is
space
              {
                  Temp  =  Temp  +  '  ';  //Pad  the  temp
text with space
              } else {
                  if (alphabetonly) //if alphabet mode
                  {
                      TempChar                        =
inttoCharAlphabetMode((charToIntAlphabetMode(plain.char
At(i))  +  charToIntAlphabetMode(keys[KeyPositionCounter
% key.length()])) % 26);
                  } else //if extended mode
                  {
                      TempChar         =         (char)
((plain.charAt(i)   +   key.charAt(KeyPositionCounter   %
key.length())) % 256);
                  }
                  ++KeyPositionCounter;//add        the
counter
                  Temp  =  Temp  +  TempChar;  //Pad  the
temp text
              }
          }
           endTime = System.currentTimeMillis();
           CharWaktu = endTime-startTime;
           System.out.println("Waktu   eksekusi   dengan
array char adalah : "+ CharWaktu + " Milisecond");
          return Temp;
    }
```

Berikut hasil pengujian yang dilakukan :

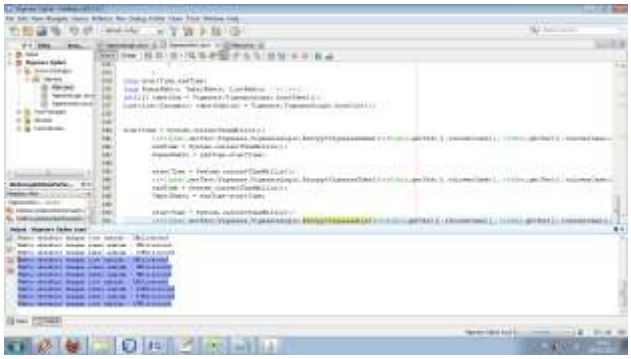
- 125 karakter :
  - String : 0 Milisekon
  - Array : 0 Milisekon
- 750 karakter :
  - String : 0 Milisekon
  - Array : 0 Milisekon
- 1500 karakter :
  - String : 15 Milisekon
  - Array : 16 Milisekon
- 3000 karakter :
  - String : 47 Milisekon
  - Array : 46 Milisekon
- 6000 karakter :
  - String : 156 Milisekon
  - Array : 93 Milisekon
- 12000 karakter :
  - String : 422 Milisekon
  - Array : 358 Milisekon
- 24000 karakter :

- String : 1763 Milisekon
- Array : 1841 Milisekon

- 57430 karakter :
  - String : 14991 Milisekon
  - Array : 14992 Milisekon
- 114000 karakter :
  - String : 25319 Milisekon
  - Array : 25631 Milisekon




*Bagan 3. Bagan Perbandingan*



*Gambar 4. Pengujian pada NetBeans*

Dari hasil pengujian didapatkan bahwa pengacuan ke *string* tidak berbeda jauh dengan *array*. Dengan demikian, lebih baik kunci dibiarkan dalam bentuk *string*.

## VIII. KESIMPULAN

Dari pengujian yang dilakukan, didapatkan kesimpulan bahwa untuk mendapatkan performa algoritma Vigenere Cipher yang baik, kita harus memerhatikan beberapa hal berikut :

- Gunakan plaintext langsung dari string menggunakan fungsi charAt.
- Gunakan kunci langsung dari string menggunakan fungsi charAt
- Buat tabel konversi pada awal program dan substitusi Vigenere berdasarkan tabel substitusi tersebut.

## REFERENCES


[1] Munir, Rinaldi. 2005. Diktat Kuliah IF5054 Kriptografi.Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.

[2] http://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher (diakses tanggal 5 Maret).

[3] http://download.oracle.com/javase/1.4.2/docs/api/overview-summary.html (diakses tanggal 5 Maret).

[4] http://wiki.answers.com/Q/How_many_characters_are_contained_in_a_single_double-spaced_typed_page (diakses tanggal 5 Maret).


## PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 5 Maret 2010

Archie Anugrah / 13508001