Attack to Vigenere Cipher

HW 1 - CNS Sapienza

Daniele Oriana 1457625

13 Oct 2016

Read about the cipher and describe it.

Answer

The Vigenére cipher represents one of the simplest polyalphabetic ciphers. It provides a text encryption based on different applications of the Caesar cipher. In a Caesar cipher each letter of the alphabet is shifted by a fixed number of positions along the alphabet, obtaining, in this way, a letter of the alphabet that represents the result of the encryption of the original letter. The Vigenére cipher uses different times the Caesar cipher, each time with a different number of shifts that depends on the letters of the so called keyword. In order to encrypt a plaintext it is possible to use the so called Vigenére table, that is a sort of matrix in which in each row we can find the application of the Caeser cipher shifting the previous row by one position on the left. This gradually application of the Caesar cipher leads to the creation of the Vigenére table representing all the 26 possible Caesar ciphers. This table shows how you can encrypt a letter of the plaintext considering the corresponding letter of the keyword that influences the shift in the table. The process provides that if the length of the plaintext is greater than the length of the keyword, then you have to reuse the keyword from the beginning.
So we can say that:

$C_i = E_k(M_i) = (M_i + K_i) \bmod 26$

$M_i = D_k(C_i) = (C_i - K_i) \bmod 26$

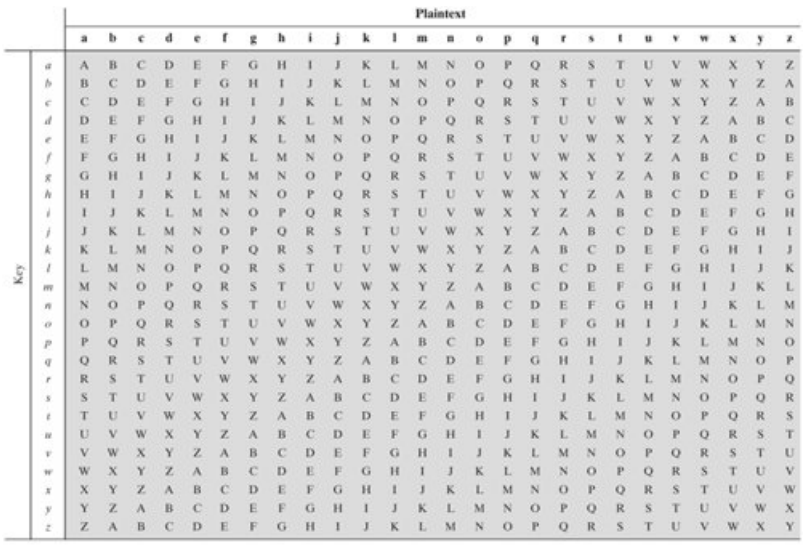

Figure 1: Vigenére table.

Implementation of the cipher.

Answer

The following java code represents my personal implementation of the Vigenere cipher.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class VigenereCipher1457625 {

    public static void main(String[] args) throws
       IOException {

        String[] options = new String[]{"I want to
           encrypt", "I want to decrypt"};
```

```
int response = JOptionPane.showOptionDialog(
   null, "", "Vigenere Chipher", JOptionPane
   .DEFAULT_OPTION, JOptionPane.
   PLAIN_MESSAGE, null, options, options[0])
   ;
if (response == 0) {
    JFrame frame = new JFrame("Vigenere
       Cipher");
    String name = JOptionPane.
       showInputDialog(null, "Insert the
       name of th file that contain the
       plaintext (without specify the .txt
       extension)");
    String plaintext = readFile(name + ".txt
       ");
    String keyword = JOptionPane.
       showInputDialog(null, "Insert the
       keyword for the cipher");
    String ciphertext = encrypt(plaintext,
       keyword);
    JTextArea textArea1 = new JTextArea(26,
       45);
    textArea1.setText("ENCRYPTED TEXT: \n" +
        ciphertext);
    textArea1.setEditable(false);
    JScrollPane scrollPane1 = new
       JScrollPane(textArea1);
    JOptionPane.showMessageDialog(frame,
       scrollPane1);
} else if (response == 1) {
    JFrame frame = new JFrame("Vigenere
       Cipher");
    String name = JOptionPane.
       showInputDialog(null, "Insert the
       name of th file that contain the
       ciphertext (without specify the .txt
       extension)");
    String ciphertext = readFile(name + ".
       txt");
```

```
            String keyword = JOptionPane.
               showInputDialog(null, "Insert the
               keyword for the cipher");
            String back = decrypt(ciphertext,
               keyword);
            JTextArea textArea2 = new JTextArea(26,
               45);
            textArea2.setText("BACK TO THE PLAINTEXT
                (DECRYPTION): \n" + back);
            textArea2.setEditable(false);
            JScrollPane scrollPane2 = new
               JScrollPane(textArea2);
            JOptionPane.showMessageDialog(frame,
               scrollPane2);
        }

        System.exit(0);
    }

    public static String encrypt(String plaintext,
       String keyword) {
        String ciphertext = "";
        plaintext = plaintext.toLowerCase();
        keyword = keyword.toLowerCase();
        for (int i = 0, j = 0; i < plaintext.length
           (); i++) {
            char c = plaintext.charAt(i);
            if (c < 'a' || c > 'z') {
                continue;
            }
            ciphertext = ciphertext + (char) (((c -
               97) + (keyword.charAt(j) - 97)) % 26
               + 97);
            j = ++j % keyword.length();
        }
        return ciphertext;
    }

    public static String decrypt(String ciphertext,
       String keyword) {
```

```
        String plaintext = "";
        ciphertext = ciphertext.toLowerCase();
        keyword = keyword.toLowerCase();
        for (int i = 0, j = 0; i < ciphertext.length
           (); i++) {
            char c = ciphertext.charAt(i);
            if (c < 'a' || c > 'z') {
                continue;
            }
            plaintext = plaintext + (char) ((c -
               keyword.charAt(j) + 26) % 26 + 97);
            j = ++j % keyword.length();
        }
        return plaintext;
    }

    public static String readFile(String fileName)
       throws IOException {
        try (BufferedReader br = new BufferedReader(
           new FileReader(fileName))) {
            StringBuilder sb = new StringBuilder();
            String line = br.readLine();

            while (line != null) {
                sb.append(line);
                sb.append("\n");
                line = br.readLine();
            }
            br.close();
            return sb.toString();
        }
    }
}
```

The implementation follows the simple features written in the previous answer, i.e.
$C_i = E_k(M_i) = (M_i + K_i) \bmod 26$
$M_i = D_k(C_i) = (C_i - K_i) \bmod 26$

Obviously i was forced to add in the code some expedients in order to

respect the conversion between decimal and char notations. These expedients are detectable, for example, in the parts of the code in which i subtract the number 97 to shift the reference in the conversion from decimal notiation to char notation or in the parts in which i add the number 26 for similar reasones.

How to run it:

Create a .txt file in which you put the text that you want encrypt or decrypt. After that run the VigenereCipher1457625.java (i have tried to send the executable jar file, but gmail blocks it for security reasons). Subsequently follow the messages on the screen.

Test the code.

Encrypt a text of Shakespeare.

Answer

The text of Shakespeare that i have choosen is "The Phoenix and the Turtle", that is the following one:

```
Let the bird of loudest lay
On the sole Arabian tree
Herald sad and trumpet be,
To whose sound chaste wings obey.

But thou shrieking harbinger,
Foul precurrer of the fiend,
Augur of the fever's end,
To this troop come thou not near.

From this session interdict
Every fowl of tyrant wing,
Save the eagle, feather'd king;
Keep the obsequy so strict.

Let the priest in surplice white,
That defunctive music can,
Be the death-divining swan,
Lest the requiem lack his right.
```

And thou treble-dated crow,
That thy sable gender mak'st
With the breath thou giv'st and tak'st,
'Mongst our mourners shalt thou go.

Here the anthem doth commence:
Love and constancy is dead;
Phoenix and the Turtle fled
In a mutual flame from hence.

So they lov'd, as love in twain
Had the essence but in one;
Two distincts, division none:
Number there in love was slain.

Hearts remote, yet not asunder;
Distance and no space was seen
'Twixt this Turtle and his queen:
But in them it were a wonder.

So between them love did shine
That the Turtle saw his right
Flaming in the Phoenix' sight:
Either was the other's mine.

Property was thus appalled
That the self was not the same;
Single nature's double name
Neither two nor one was called.

Reason, in itself confounded,
Saw division grow together,
To themselves yet either neither,
Simple were so well compounded;

That it cried, "How true a twain
Seemeth this concordant one!
Love has reason, reason none,
If what parts can so remain."

Whereupon it made this threne
To the Phoenix and the Dove,
Co–supremes and stars of love,
As chorus to their tragic scene:

threnos

Beauty, truth, and rarity,
Grace in all simplicity,
Here enclos'd, in cinders lie.

Death is now the Phoenix' nest,
And the Turtle's loyal breast
To eternity doth rest,

Leaving no posterity:
'Twas not their infirmity,
It was married chastity.

Truth may seem but cannot be;
Beauty brag but 'tis not she;
Truth and beauty buried be.

To this urn let those repair
That are either true or fair;
For these dead birds sigh a prayer.

I have choosen as keyword for the encryption the word "venice", obtaining as ciphertext the following one:

gigbjiwmelqjgshlgwopngqrolraqpzeeidmvrgzgicieinhneqiphovhuriofrbqacsf
musprqkjenxrekrbwbjgcwygbjspwuzkifmaojemfvviimjbcntmipctvzvbnvlzjvmp
hvytctsaxumhiqieagryxbbjmnxewqtxszmvljyawvrzeentshxuquwzwfqqrdrgmthd
ggmximcswypjjggteixjqpkneimvlzinoniainbjimhxqpkfirxvlzsoagupcfwuxmmp
bnioxumrvdifbkrnyexnmxijpkxzxuivhzjhvexdzruwwdgpipfzxumfivxulkzdrvvi
wreatgwoxumtilyvmopvgxpkwmmtpveihgpqyovrjniyegmfgmsjbjeoxuguewprogry
ieuconxjqvlolrjtivxubjspkvduxvrqbconxzwpknxbctqjyevgvnwuinxolbciscie
mvlzeabjihhbbjgjqzmpgzpbdgeihpwpwoeakamnhriftcsrvkbvrqbjioyebniaprlk
rvqhbwegjyioiavbujiigraqxciltqzyeftqzzmabyedruifxcirauiigrjwxdrbvgxr

sqquxdrpbuhdzvaksirbvgrpqomtxciemkrgsimyenwyikrcinzvwmizwvitigvqxvwh
vfimhvaveigriphisfxcgzanauizrgekboxuquxpvgtgeihuquupirvdyomabjihmgeg
vzejwphzvfwdioarmpxciztqzzhvluldrrbjeoxumvymxymuerlvatmblgnnehmaokro
lrxjszrvfumblgmkxcieecwolrwvlzvfukrztewrimxlecwolhactkeytgholnbvlzwr
thavwawvxcifioinmaoniiegctinhbcdpzrnugrzmgpgvoabvqvjrrecwxeytghminaq
rdrvbuigjpwpjjyalghnejlkzdwvwpkmsjbqkzxumtxjxumowzpimuczxrqvlzvamkxc
ieakqkpregvzwbegpggbursprqmfxcegqvgmmrljsrxecgeoanqpwzizmvlolvaesigb
zfeixbvgpjzrpcwminaqrminaqrisamkjrlnbremxfkcrnsemoedrjpgvzycwpmoqnlg
xcmfbjvzrrbqxcicpqiimkipholrlqzzgbawtmizmueihfbcvnsstqzzefkjsmyfbqxc
ivzvvvkvkugzrrbjvzrbadivyggvvpxuiphmeeqvcbvnkgmieytumhtyqemocumtizrp
tqwymakkryieanmzhrivldwawyxcicpqiimkvgwoealvlzxhzvpzwywaegfemcwoxbmv
imrvbahjxuzgwoprixmikawrsnxrzkxtxjiurjxgpgmmmankvhmggkxrefucvmmrlelv
wgqvcovhbjqvcfmgqwygkcrisgjgfzehbafmetjwxomfvqxnlrbtyolnvffzehbafpvv
mffzxbbjmnyevnioxuwuimicikvolnbcvzivbjimxecgsmjnqtjjvgpgwzhriffdvqau
mblnxtetie

Try to attack the ciphertext.

Answer

In order to attack the ciphertext i have choosen to use the so called Kasiski Analysis and i have used an online tool to use it (the tool is available here http://crypto.interactive-maths.com/kasiski-analysis-breaking-the-code.html#encrypt). This kind of examination allows to attack polyalphabetic substitution ciphers, such as the Vigenre cipher. We can divide the procedure provided by the attack in three main phases:

- In this first part the method tries to discover the length of the keyword used for the encryption. This is done analyzing all the ciphertext, finding all the groups of repeated letters and considering also the distance (in term of letters) between each pair of repeated group of letters. The distance between the repetition is very important, because if two groups of repeated letters are divided by a number of n letters, it means that the keyword probably has the length equal to one of the divisors of n. So, for every distance that we have found, we consider all the possible divisors. At the end of this process we can assume that the length of the keyword is equal to the divisors that occurs more times than the others.

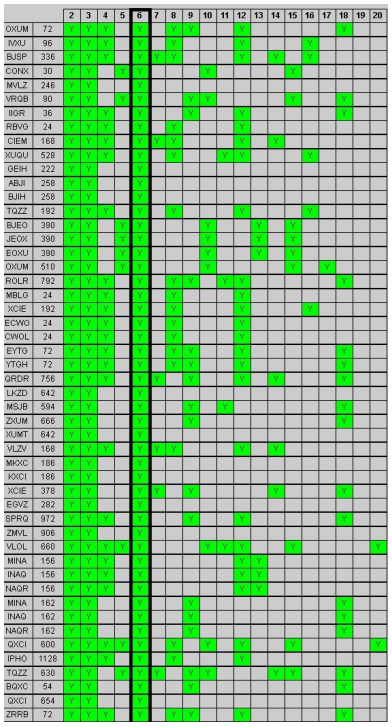

Figure 2: Application of the first phase to our case. Part I.

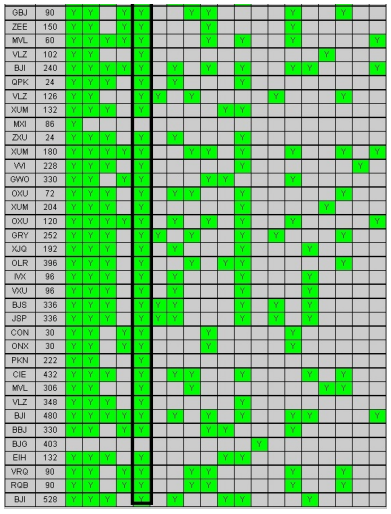

Figure 3: Application of the first phase to our case. Part II.

Applying this process to our case we obtain a lot of matches in the coloumn nuumber six. This means that probably the keyword has length equal to six (and it is true, because our keyword is venice).

- Once we have determinated the probable length of the keyword (we call it n), we can say that every n letters of the original plaintext the same shift has been used to encrypt. As a conseguence of that we can start the second phase of the process organizing the ciphertext into n coloumns; if two letters are in the same coloumn, this means that they have been encrypted using the same shift. We will label for convenience the coloumns as L1,L2,...,Ln (in our case n=6, so L1,...,L6).

- Finally we have the third phase of the process that provides the use of frequency analysis,in fact here, for each letter contained in each one

of the columns generated in the second step, we create graphically their frequency distribution in the column. Once we have created this distribution, we compare it with the distribution of each letter in the English language. A better way to compare them is the graphical one, this means that we express these two distributions through some percentage graphics and, after that, we try to overlap the graphics basing our reasoning on the similarities between them. In these way we use the frequency distribution in order to determinate the shift that has taken place for each letter of the original plaintext.

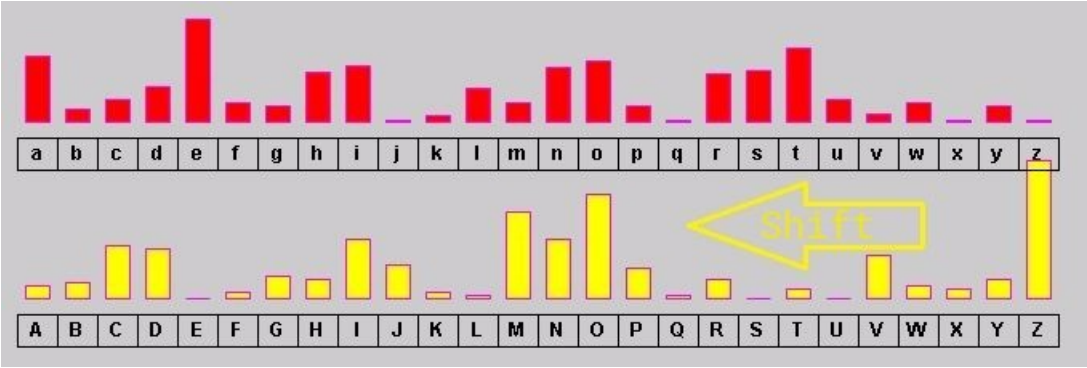

Figure 4: Initial situation, we need to shift.

Figure 5: After the shifts the two graphics are similar.

In the previous pictures the red graphics represents the percentage frequency of each letter in the English language, while the yellow one shows the percentage frequency of each letter in the first column. We shift the yellow one and when they are similar we can fix the first letter of the keyword assuming that is 'v' (because the shifts leads to the overlapping of 'a' and 'v'). In this way we can know all the shifts and therefore all the letters of the keyword, in fact continuing the method also for the others columns (L2,...,L6) we can achieve all the letters of the keyword.

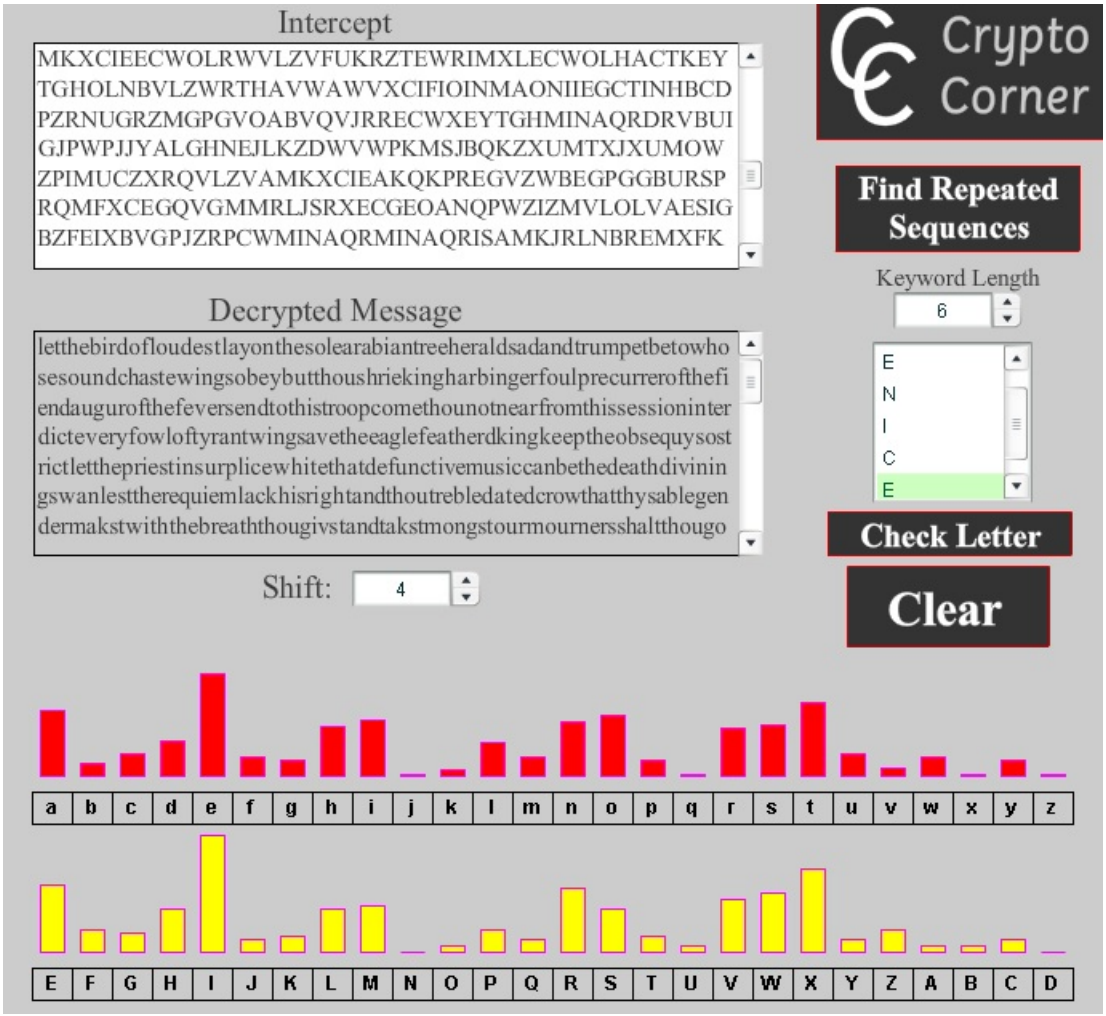

Figure 6: Result of the procedure.

So, finally we obtain the correct keyword ("venice") and the original plaintext (the text of Shakespeare).

I have tried to write a code based on the previous procedure. I have written in java code the first and the second step of the previous method, but i don't know how to create an algorithm that is able to compare different percentage distributions and, applying some shifts, modify one of the them until they can be considered similar. In particular i have found some difficulties in define the con-

cept of “similarity” for two percentage distributions. The code is in the following files: Substring1457625.java,Vigenere1457625.java and AttackVigenere1457625.java (contains main).