Coursework 1 - The Viginere Cipher

Rafael Rost

***Abstract*—In order to complete a coursework of master degree cryptography classes, in this report I have presented how to decrypt a given cipher-text that was encrypted using the Vigenere Cipher. As part of the coursework, a decrypt tool was developed in Ruby language. The tool usage and also the cryptanalysis method was explained step by step across the report.**

***Keywords*—Polyalphabetic Cipher, Vigenere Cipher, Index of Coincidence IC, Ruby.**

I. Introduction

The Vigenere cipher is an encrypt method that combines a few Caesar ciphers into one. It's a very old cipher that was created by the France diplomat Blaise De Vigenere and was used during the 1500s.

Although today this cipher seems to be easily broken, during three centuries it resisted all attacks attempts. Across this paper, I presented how to break this cipher, using the index of coincidence that was first described by William Friedman. After applying Friedman's method to discover the key length, I described how to perform a simple frequency analysis to get the right key.

As part of the cryptanalysis, a tool was developed to speed up the process. The chosen language was Ruby, and the source-code is available on my GitHub account.

II. Vigenere Cipher

The Vigenere cipher basically combines a sequence of substitution ciphers, each one with a different shift value. The encryption and decryption process can be easily done by using a table that was called the Vigenere Square. The table is a composed by 26 rows and 26 columns, where each row represent a new alphabet left shift cycle, which means 26 different Caesar ciphers.

Suppose that we want to cipher the message "Hello World" with the key "QWER". We get the first plain text letter "H" and the fist key letter "Q" and look at the Viginere Square. The fist letter of the cipher-text will be the intersection between those two letters, in this case X. We keep doing the same with following plain text letters, restarting the key when needed, and we will get the "xapce ssibz" cipher-text.

III. Cryptanalysis of Vigenere Cipher

Given a cipher-text, that was encrypted with Vigenere cipher, lets now discover the key, and off course, the original plain text.

Cipher text: cpimvtcmngiheycgremhwltmvtceiiqwivwh-mumhwptawgoptfizftgitlezpcgxfspsbuchhbqwhalhdsa

A. Key Length

The first step to discover the key, is to determine the key size. There are different available methods such as Kasiski and Friedman methods. I choose Friedman's method because of the reproducibility, which means, it's easier to codify when comparing with Kasiski method.

Friedman's method is based on the index of coincidence. The main goal of this index is to determine how random a given text is. The calculation is based on a math formula and was implemented on Ruby tool as presented in the code below. Calculation steps were described in the code.

```
# Counts the occurrences of each letter
# in the text
def frequency(text)
  occurrences = Hash.new(0)
  text.each_char do |char|
    occurrences[char] += 1
  end
  occurrences
end
# Calculates f * (f - 1) for each
# frequency. Returns the sum of
# all calculated values.
def frequency_sum(text)
  occurrs = frequency(text)
  occurrs = occurrs.values.map do |item|
    item * (item - 1)
  end
  occurrs.inject(0, :+)
end
# Calculates the index of coincidence
def index_of_coincidence(text)
  sum = frequency_sum(text)
  size = (text.size * (text.size - 1))
  (sum.to_f / size.to_f)
end
```

The expected index of coincidence of an English sentence is 0.065. In other words, calculating the index for a random text will result in a value by far different then 0.065. By understanding how Vigenere cipher works, it's possible to notice that according the key size, the cipher-text will have groups of letters that was shifted with the same approach.

Since Vigenere it's all about letters transposition, it's possible to guess the key size. Lets say that the key size it's three. So, the cipher-text is chunked in three columns and we calculate the index of coincidence of each column. For the given cipher-text of this coursework, the result is showed below.

- 0.04503
- 0.04512

- 0.04535

Now, you do the same with N key sizes and try to figure out which key size will fit better with English sentences index of coincidence (0.065). In the Ruby implementation, the program will ask the max key size that you want to try, and then will print a results of each key size. For the cipher-text of current coursework the key size is five, and it's easily to see in the program report.

- 0.06652
- 0.06756
- 0.06627
- 0.06403
- 0.06669

B. Frequency analysis

Having the key size, let's now perform a simple frequency analysis in each group of letters that was encrypted (shifted) with the same part of the key. In this case, there are five blocks of letters, each one encrypted with one letter. The main goal here, is to compare the frequency of the letters in this blocks with the frequency of letters in normal English texts.

```
# Constant array with the percentual
# of occurrences of each letter in
# common english sentences
EN_PERC =
{ "a" => 8, "b" => 2, "c" => 3, "d" => 4,
  "e" => 13,"f" => 2, "g" => 2, "h" => 6,
  "i" => 7, "j" => 0, "k" => 1, "l" => 4,
  "m" => 2, "n" => 7, "o" => 8, "p" => 2,
  "q" => 0, "r" => 6, "s" => 6, "t" => 9,
  "u" => 3, "v" => 1, "w" => 2, "x" => 0,
  "y" => 2, "z" => 0 }
```

This frequency analysis was completely automated on the Ruby implementation. Basically you have to prompt the key size, and the program will show a report with the frequency analysis. The program will also try to guess the key, by calculating the shift of the most common English word "E".

To the first block, T was the most frequent letter. So, guessing that T in fact is E, we had a shift of 15 letters, which means, the first letter of the key was P (15th alphabet letter, considering that A..Z is 0..25).

On the second block, P was the most frequent letter, which means a shift of 11 letters from E. And so, the second key letter was L. On the third block, we noticed that E was the most frequent letter, and in this case, there was no shift, and the key letter must be A. Doing the same with all the blocks, we will get the entire key guess PLATO.

C. Plain Text

With a good key guess and the cipher-text, now it's time to decrypt and get access to the original plain text. This is possible by simple applying Caesar ciphers in each letter of the cipher-text, according the key.

The first cipher-text letter is "C" and was encrypted with the first key letter "P". Now we get the difference between positions of this two letters in the alphabet, in this case 13, and apply a 26 module operation. The result will be the position of the first plain text letter, which in this case is N (13). We keep doing the same, getting the second letters of cipher-text and key, applying the same steps, and we have the second plain text letter E.

In the Ruby tool there is an implementation of the described decryption steps. By running it, we finally have the entire plain-text:

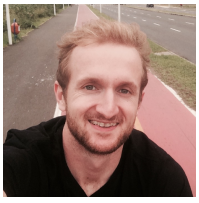
"neither must we forget that the republic is but the third part of a still larger design which was to have included an ideal history of at ..."

Having the plain-text the main goal of the coursework was achieved.

IV. Conclusion

Vigenere Cipher show us how import his to keep the cipher-text as most randomly as possible, without any kind of pattern. With a simple frequency analysis and some automation, it was possible to discover the key, and then the entire plain-text.

Off course that we are talking about a very old cipher, and at that moment this kind of encryption, with just some letters transportations, was really enough to keep secrets safe during at least three centuries.



Rafael Rost Software engineer at HP Inc. GitHub account: github.com/rafarost