IN4191 SECURITY AND CRYPTOGRAPHY 2015-2016

Assignment 1 - Report

Eric Camellini
4494164
e.camellini@student.tudelft.nl

***Abstract*—In this report, I explain how I deciphered a message encrypted with a simple substitution cipher. I focus on how I used frequency analysis to understand the substitution pattern and I also give a brief explanation of the cipher used.**

I. INTRODUCTION

The objective of this work was to decipher an encrypted message using frequency analysis or brute forcing, knowing that:

- It seemed to be encrypted using a simple cipher;
- It was probably written without spaces or punctuation;
- The language used could be English, Dutch, German or Turkish.

The ciphertext was the following:

VaqrprzoreoeratraWhyvhfraJnygreUbyynaqreqrgjrroebrefinaRqvguZnetbganneNzfgreqnzNaarjvytenntzrrxbzraznnezbrgabtrrarragvwqwrovwbznoyvwiraBznmnyurgzbrvyvwxuroorabzNaarabtrracnnejrxraqnnegrubhqrafpuevwsgRqvguSenaxvarraoevrsnnaTregehqAnhznaauhaiebrtrerohhezrvfwrvaSenaxshegnzZnva

II. METHODS

To solve the problem, I first applied the frequency analysis technique: I wrote a Python program that computes the frequency of every letter in the ciphertext, then replaces each character with the corresponding one in the letter frequency order of the selected language [1]. The program also allows to manually perform further substitutions. This approach was based on some pre-processing:

- The ciphertext was lowercased before the substitution;
- In the alphabets of the four possible languages, only the 26 traditional Latin characters were considered.

I then used the described program to analyze the ciphertext and to search for a pattern in the substitution scheme, trying to switch between the four languages and to search for existing words after the letter replacement. Later I also extended the program with a spell checker function using the PyEnchant [2] library: this function finds existing words in the text using a dictionary of the selected language.

With this approach I observed that after the frequency-based letter replacement the language with more word matches (with a minimum word length of 3 characters) was Dutch, so I focused on it trying to compare its letter frequency with the one of the ciphertext. The letters in order of frequency were the following:

Ciphertext: ”ranevzbgqhoutwyfsxjipcm”
Dutch language: ”enatirodslghvkmubpwjczfxyq”

Looking at the two lists I observed the following pattern:

- The first 3 most frequent letters in the text have (r, a, n) a 13 letters distance in the alphabet from the 3 most frequent ones in the Dutch language (e, n, a).
- Other frequent letters preserve this 13 letters distance pattern: for example the letters ’e’ and ’v’ are in the 6 most frequent letters in the ciphertex, and they are 13 letters distant from ’r’ and ’i’ respectively, that are in the 6 most frequent ones in the Dutch language.

On the base of this observation, I made a script that performs a substitution like the one shown in Figure 1, for both lower and upper case letters, and with it I successfully deciphered the text. Results are shown in Section III and the source code of the scripts can be found in the Appendices (A and B).

III. RESULTS

The output of the final script was the following text:

IndecemberbrengenJuliusenWalterHollanderdetweebroersvanEdithMargotnaarAmsterdamAnnewilgraagmeekomenmaarmoetnogeeneentijdjebijomablijvenOmazalhetmoeilijkhebbenomAnnenogeenpaarwekendaartehoudenschrijftEdithFrankineenbriefaanGertrudNaumannhunvroegerebuurmeisjeinFrankfurtamMain

It is Dutch text without spaces and punctuation, as hypothesized.

[1] http://letterfrequency.org/ - letter frequencies source.
[2] http://pythonhosted.org/pyenchant/

A. The Cipher

The cipher used to encrypt the message is the ROT13 Caesar's cipher [3], a simple substitution cipher that replaces each letter of the plaintext with the letter that is 13 positions after it in the alphabet (Figure 1). Because the alphabet is seen as a circular structure and contains 26 letters, the same function can be used for encrypting and decrypting.

IV. Conclusion

In conclusion, I successfully decrypted the given message, showing how easy is to break a simple Caesar cipher using frequency analysis and human intuition. Once discovered the substitution scheme, a simple script can easily decipher the text without a significant computational effort.

V. Discussion

In this section I propose two possible improvements in the cipher used, to make the deciphering more difficult:

- This kind of cipher could be improved by using substitution units larger than one single character, in order to increase the number of possibilities (i.e. a single letter can be replaced with the remaining 25, while a couple can be replaced with all the possible different couples of letters). In order to choose the replacement units many schemes could be invented, or there could be a fixed substitution table. Notice that frequency analysis can be done also on units larger than one single character, but it requires more effort and it is less effective.
- A further improvement could be using a substitution table where every row is a permutation of the alphabet and the letters in the plaintext address the column (e.g. the first column is the letter that will replace the A, the second one for the letter B and so forth). A shared key could then be used to address the correct row for the whole message or for every character.

Appendix A
Rot13 Python script

Source code of the final deciphering script:

[3] See https://en.wikipedia.org/wiki/ROT13 for more information.

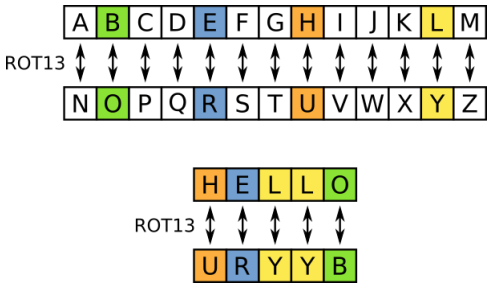

Fig. 1: Rot13 cipher table[1]

```
import sys


def substitute(text, dictionary):
    """Substitute each char in 'text'
    using 'dictionary' as a map for the
    substitution pattern."""
    output = ""
    for i in text:
        output = output + dictionary[i]
    return output


def rot13_dict():
    """Generate the rot13
    substitution dictionary"""
    d = {}
    for i in range(0, 13):
        char1 = chr(ord('a') + i)
        char2 = chr(ord(char1) + 13)
        d[char1] = char2
        d[char2] = char1
        char1 = chr(ord('A') + i)
        char2 = chr(ord(char1) + 13)
        d[char1] = char2
        d[char2] = char1
    return d

if __name__ == "__main__":
    if(len(sys.argv) != 2):
        print "Missing argument - Usage:"
        print "python rot13.py text"
        sys.exit()
    text = sys.argv[1]
    print substitute(text, rot13_dict())
```

Appendix B
Frequency analysis script

Source code of the program used to perform frequency analysis on the ciphertex (simplified version, without language selection):

```
from collections import Counter
import os
import enchant

def match_count(language_dict, c, min, max):
    count = 0
    for i in range(len(c)):
        for j in range(i + min, i + max + 1):
            if(j <= len(c)):
                word = c[i: j]
                if(language_dict.check(word)):
                    count = count + 1
                    print word, " ",
    print ""
    return count

if __name__ == "__main__":
    c = #ciphertext string
    sub_dict = {}
    dutch_freq = "enatirodslghvkmubpwjczfxyq"
```

```
d_nl = enchant.Dict("nl")
freq = Counter(c.lower()).most_common()
f = [x[0] for x in freq if (x[0] != '\n')]
sub_dict = dict(zip(f, dutch_freq))
while(1):
    os.system('clear')
    sub_c = ""
    for l in c.lower():
        if(l in sub_dict):
            sub_c = sub_c + sub_dict[l]
        else:
            sub_c = sub_c + l
    print "\nSUBSTITUTIONS: "
    print sub_dict
    print "\nTEXT WITH SUBSTITUTIONS:"
    print sub_c
    print "\nMATCHES: "
    count = match_count(d_nl, current_c)
    print count, " Words mathing."
    s1 = raw_input("\nIns. char to sub: ")
    s2 = raw_input("Ins. the new char: ")
    sub_dict[s1] = s2
```

REFERENCES

[1] Benjamin D. Esham, https://en.wikipedia.org/wiki/ROT13#/media/File: ROT13_table_with_example.svg, Wikimedia Commons, 2007.