DAT510 – Assignment 1

Yohannes Kassaye – 249238

Abstract

In this paper we will go through the tasks and results of the first assignment in DAT510 – Security and Vulnerability in Networks. We were tasked with decrypting different kind of messages where each message was encrypted using different types of Ciphers. The first part consisted of cracking a Poly-alphabetic cipher with little to no knowledge of information such as key length and the actual cipher that was used. The second part consisted of cracking a product cipher where we were given all the information needed do so.

1. Introduction

Cryptography is the practice and study of techniques for secure communication in the presence of third parties. The main goal of cryptography is to construct and analyze protocols that prevent third parties or the public from reading private messages, to ensure central pillars of information security such as data confidentiality, data integrity, authentication, and non-repudiation.

This assignment presents an introduction to practical ways of learning how these protocols work, trough tasks related to cracking them and thereby understanding there innerworkings.

2. Design and Implementation

The design and implementation of the tasks was done using Python and Jupyter Notebooks. This is a great combination as python is a dynamically typed language and has its strengths in being an easy and great scripting language. The use of Jupyter Notebooks was a decision that were made given the nature of the technology. It is very easy to visualize the data you are working on, and it is especially great at helping the developer keep all of the data in front of him.

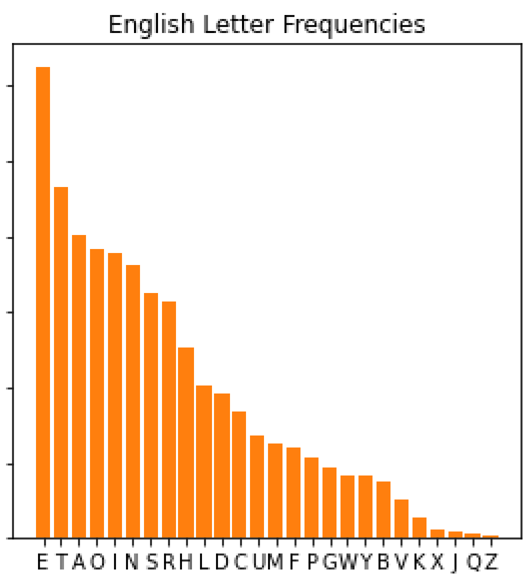
2.1 Part I – Poly-alphabetic Ciphers

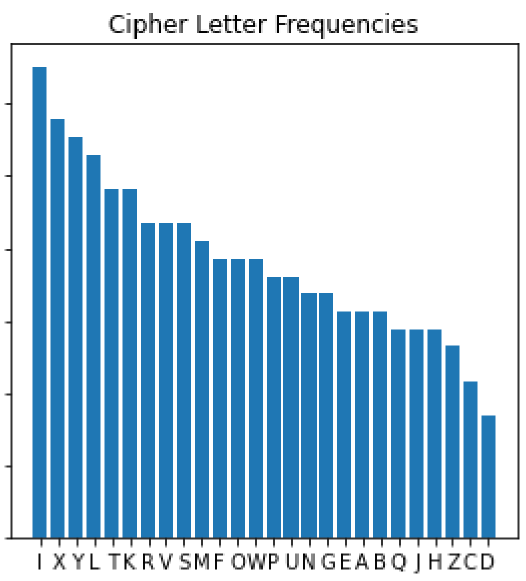
AS mentioned, the first part of the assignment consisted of deciphering a encrypted message. We were given some information on the behavior of the Cipher, but it was up to us to figure out the cipher that was used, as well as find the key used to encrypt the message.

2.1.1 Cryptanalysis

To be able to decrypt the ciphertext, we had to find out a couple of things. Firstly, we needed to figure out the encryption algorithm that were used to cipher the plaintext. Given that we only had the Ciphertext, we had to perform some analysis. We used a statistical analysis called Frequency Analysis to see if we could get some insight to the ciphering algorithm that was used.

Frequency Analysis is basically a strategy that used a weakness the English language itself has, namely that a certain combination of letters occurs with varying frequencies. This then means that performing frequency analysis on the ciphered text could be mapped back to the most frequent letters and combinations in the English language.

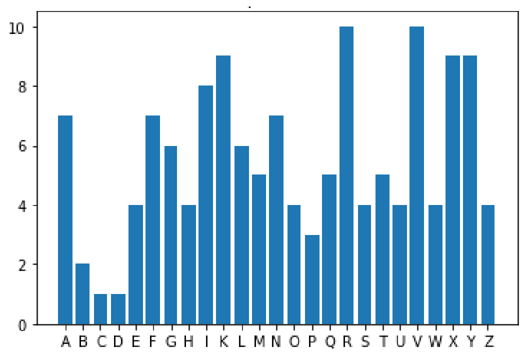




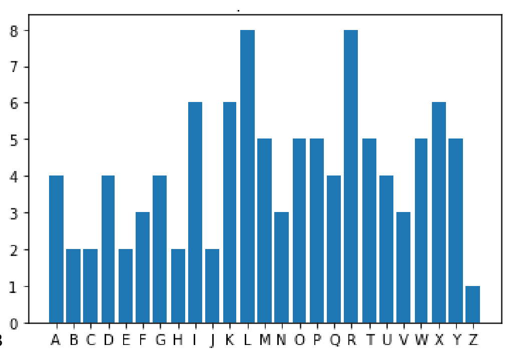
Performing frequency analysis on the ciphered text and using the relative frequencies of the letters in the English alphabet[1], yielded the figures above. Now in the book there is a plot showing the relative frequency distributions of different poly-alphabetic ciphers[2]. Compering the frequency distribution of the ciphered text, it looks very much like the distribution of the Vigenère Cipher.

The next step would be to find the key that was used to encrypt the plaintext. We were told that the key used was between 3 and 10 characters long. Now finding the key length could be done by utilizing a method called Index of Coincidence. The idea behind index of coincidence is to calculate the number of times a subset of the ciphered text appears in the text.

The more the subset appears, the more likely it is that the subset is a part of the key, given the nature of the keyword in Vigenère algorithm. Counting the number of times, a particular subset appears can give an indication of the key length.



[3]



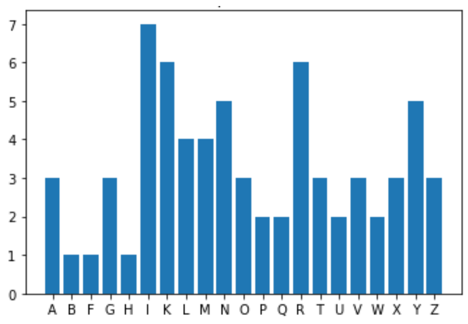
[4]

[1] https://norvig.com/mayzner.html

[2] Cryptography and Network Security 7th edition, William Stallings, Figure 3.6 - Relative Frequency of Occurrence of Letters

[3] IoC – length 3

[4] IoC – length 4



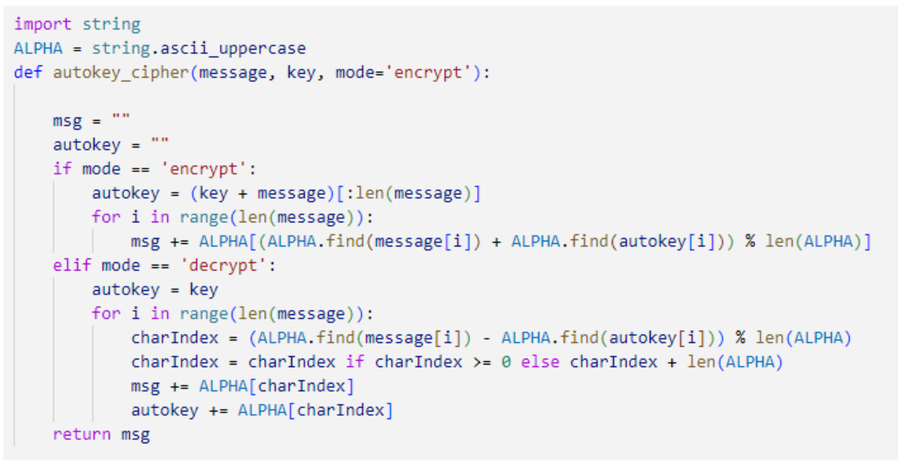
[5]

Running this method with different key-lengths of 3, 4 and 6, shown in figure 3, 4 and 5, respectively, did not yield any good results. It seems that the key used is not repeating, and this leads us to think that the ciphering algorithm used might be Vigenère Autokey.

Sadly we still haven't found the key length or the key itself. After a lot of research, trial and error I still wasn't able to find the key through pure Python code. Then we followed the hunch that the algorithm used was Autokey, and where able to use an online decoder[6] and find the key.

2.1.2 Autokey

The autokey algorithm was developed to remove the vulnerability of the old Vigenère algorithm. It differs from its predecessor by using a key which is as long as the message itself. It does this by adding the key to the start of the message and use that to encipher the plaintext. This eliminates the main vulnerability behind the Vigenère algorithm, namely the periodic nature of the keyword.



```python
import string
ALPHA = string.ascii_uppercase
def autokey_cipher(message, key, mode='encrypt'):

    msg = ""
    autokey = ""
    if mode == 'encrypt':
        autokey = (key + message)[:len(message)]
        for i in range(len(message)):
            msg += ALPHA[(ALPHA.find(message[i]) + ALPHA.find(autokey[i])) % len(ALPHA)]
    elif mode == 'decrypt':
        autokey = key
        for i in range(len(message)):
            charIndex = (ALPHA.find(message[i]) - ALPHA.find(autokey[i])) % len(ALPHA)
            charIndex = charIndex if charIndex >= 0 else charIndex + len(ALPHA)
            msg += ALPHA[charIndex]
            autokey += ALPHA[charIndex]
    return msg
```

Now running the decryption algorithm as shown above, together with the key, DATABF, we were able to decipher the following plaintext,

CRYPTOGRAPHY CAN BE STRONG OR WEAK CRYPTOGRAPHIC STRENGTH IS MEASURED IN THE TIME AND RESOURCES IT WOULD REQUIRE TO RECOVER THE PLAINTEXT THE RESULT OF STRONG CRYPTOGRAPHY IS CIPHERTEXT THAT IS VERY DIFFICULT TO DECIPHER WITHOUT POSSESSION OF THE APPROPRIATE DECODING TOOL HOW DIFFICULT… BEFORE THE END OF THE UNIVERSE

[5] IoC – length 6

[6] https://www.dcode.fr/autoclave-cipher

We were also tasked with decrypting a ciphered text that differed from the first ciphered text by one additional encryption process. Now to decrypt the message, we only needed to apply the autokey encryption twice, which resulted in the same plaintext as above.

2.2 Part II – KES

The second part of the assignment we were given a product cipher named KES. It consisted of two ciphers which together encrypted any given text. These two ciphers were Ceaser and Transposition cipher. In contrary to the first part we were given the keys that were used to encrypt the message, and was tasked with decrypting the message. Lastly we were asked to create a web server which used the KES cipher.

2.2.1 Cryptanalysis

As mentioned, the cipher consisted of two encryption algorithms. The ciphered text was encrypted first by using the transposition cipher, then by the ceasar cipher. Now in order to decrypt the message we need to reverse the encryption process, which in essence means that we need to perform decryption, firstly, with the ceasar cipher, and then with the transposition cipher.

Using the ciphering algorithms shown below, I was able to decipher the encrypted text and got,

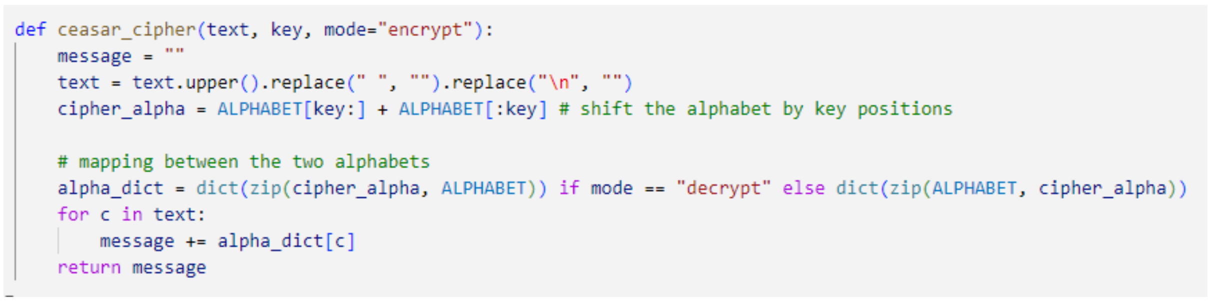
```
LEAVE YOUR PACKAGE IN THE TRAIN STATION AT SIX PM
```

2.2.2 Improvement of KES

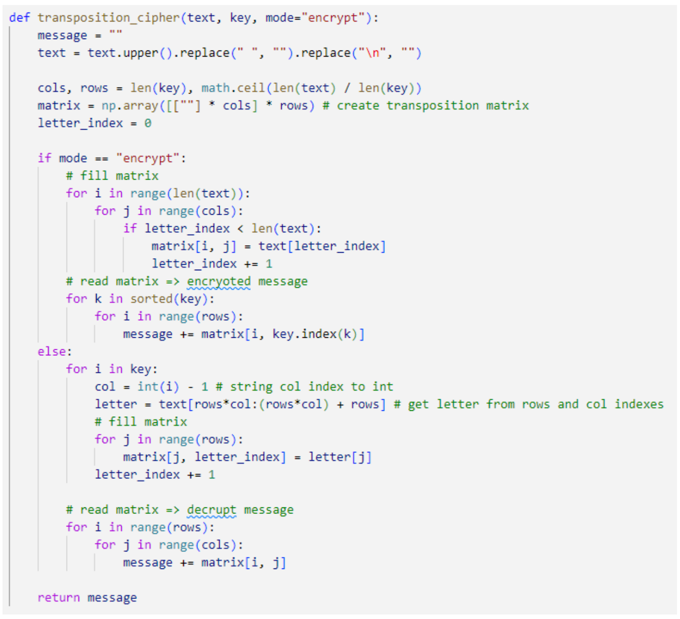
Now to improve a product cipher we could do a lot of things. We could repeat the entire KES one more time but we could also add another cipher on top of the existing once. This Is the method I choose to improve the strength of KES, namely adding another ciphering algorithm on top. This was done by adding a Vigenère Cipher on top of the existing once.

2.2.3 Implementation of KES

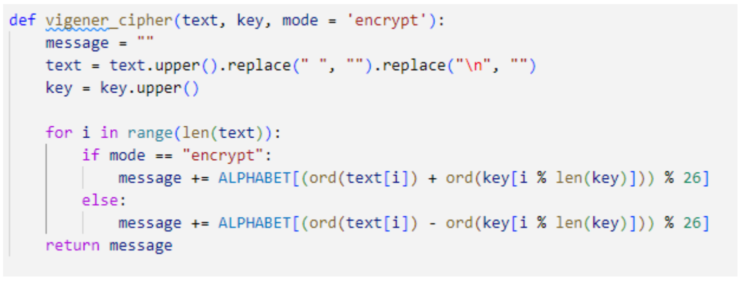
The implementation of KES was done in two steps. The first step was to create the original encryption algorithms the combined created KES. So I created a python method for the Ceasar Cipher, and then for the Transposition Cipher,



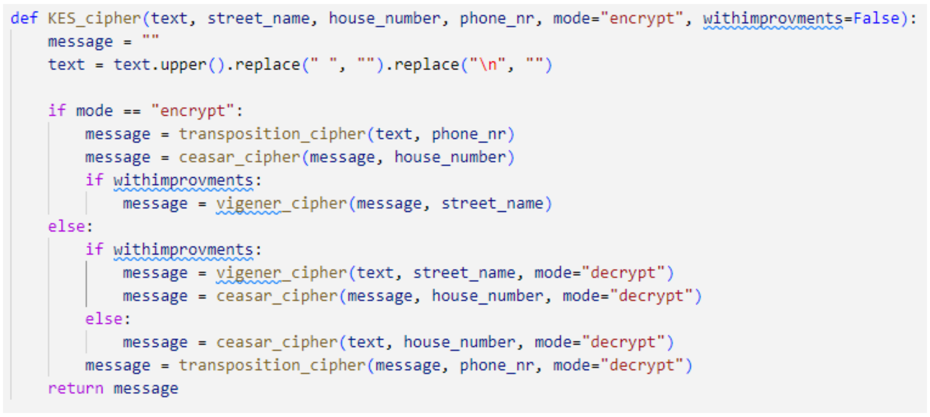
```python
def ceasar_cipher(text, key, mode="encrypt"):
    message = ""
    text = text.upper().replace(" ", "").replace("\n", "")
    cipher_alpha = ALPHABET[key:] + ALPHABET[:key] # shift the alphabet by key positions

    # mapping between the two alphabets
    alpha_dict = dict(zip(cipher_alpha, ALPHABET)) if mode == "decrypt" else dict(zip(ALPHABET, cipher_alpha))
    for c in text:
        message += alpha_dict[c]
    return message
```



```python
def transposition_cipher(text, key, mode="encrypt"):
    message = ""
    text = text.upper().replace(" ", "").replace("\n", "")

    cols, rows = len(key), math.ceil(len(text) / len(key))
    matrix = np.array([[""] * cols] * rows) # create transposition matrix
    letter_index = 0

    if mode == "encrypt":
        # fill matrix
        for i in range(len(text)):
            for j in range(cols):
                if letter_index < len(text):
                    matrix[i, j] = text[letter_index]
                    letter_index += 1
        # read matrix => encryoted message
        for k in sorted(key):
            for i in range(rows):
                message += matrix[i, key.index(k)]
    else:
        for i in key:
            col = int(i) - 1 # string col index to int
            letter = text[rows*col:(rows*col) + rows] # get letter from rows and col indexes
            # fill matrix
            for j in range(rows):
                matrix[j, letter_index] = letter[j]
            letter_index += 1

        # read matrix => decrupt message
        for i in range(rows):
            for j in range(cols):
                message += matrix[i, j]

    return message
```



```python
def vigener_cipher(text, key, mode = 'encrypt'):
    message = ""
    text = text.upper().replace(" ", "").replace("\n", "")
    key = key.upper()

    for i in range(len(text)):
        if mode == "encrypt":
            message += ALPHABET[(ord(text[i]) + ord(key[i % len(key)])) % 26]
        else:
            message += ALPHABET[(ord(text[i]) - ord(key[i % len(key)])) % 26]
    return message
```

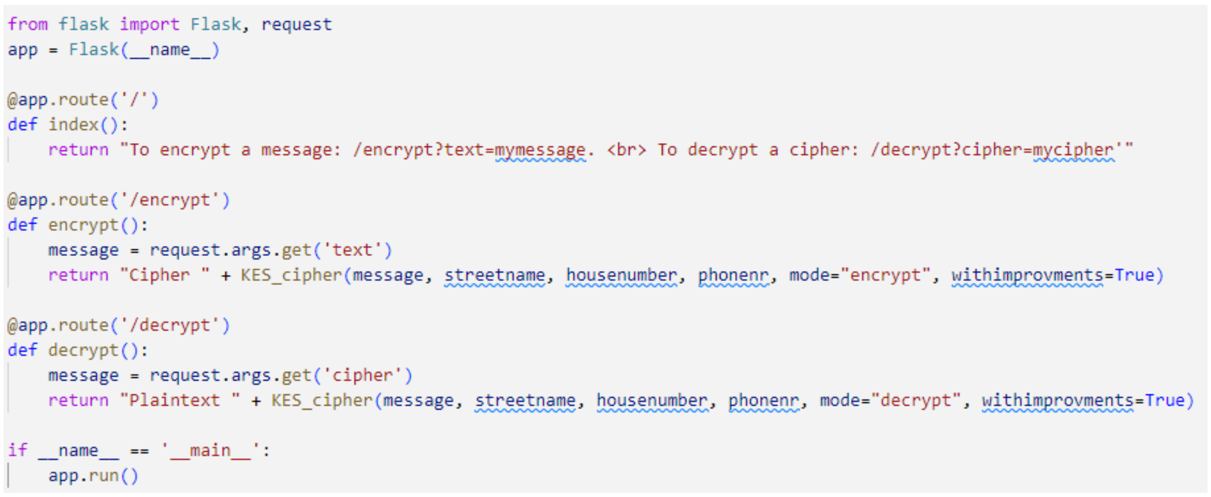


```python
def KES_cipher(text, street_name, house_number, phone_nr, mode="encrypt", withimprovments=False):
    message = ""
    text = text.upper().replace(" ", "").replace("\n", "")

    if mode == "encrypt":
        message = transposition_cipher(text, phone_nr)
        message = ceasar_cipher(message, house_number)
        if withimprovments:
            message = vigener_cipher(message, street_name)
    else:
        if withimprovments:
            message = vigener_cipher(text, street_name, mode="decrypt")
            message = ceasar_cipher(message, house_number, mode="decrypt")
        else:
            message = ceasar_cipher(text, house_number, mode="decrypt")
        message = transposition_cipher(message, phone_nr, mode="decrypt")
    return message
```

Then the remaining part was to create methods for the Vigenère Cipher, and also a method for combining all of these algorithms, which became our KES cipher. The KES cipher was able to decrypt the original ciphertext, as well as encrypt the plaintext and decrypt it again using the improved version by adding a Vigenère Cipher on top.

2.2.4 KES - HTTP Server

In the last part of the assignment were asked to create a web server that used the improved version of the KES cipher to encrypt and decrypt messages over HTTP. This was done by using a REST API framework in python called Flask, which in essence is a lightweight web server. With as little as 20 lines of code,



```python
from flask import Flask, request
app = Flask(__name__)

@app.route('/')
def index():
    return "To encrypt a message: /encrypt?text=mymessage. <br> To decrypt a cipher: /decrypt?cipher=mycipher'"

@app.route('/encrypt')
def encrypt():
    message = request.args.get('text')
    return "Cipher " + KES_cipher(message, streetname, housenumber, phonenr, mode="encrypt", withimprovments=True)

@app.route('/decrypt')
def decrypt():
    message = request.args.get('cipher')
    return "Plaintext " + KES_cipher(message, streetname, housenumber, phonenr, mode="decrypt", withimprovments=True)

if __name__ == '__main__':
    app.run()
```

we were able to create a REST API, what used the KES cipher to encrypt and decrypt text. The web server uses HTTP as protocol, and therefor the message is not encrypted while traveling from source to destination. Attackers could intercept the communication, and would have no problems with viewing the messages given that they are not encrypted.

Discussion

This assignment was as mentioned a practical introduction into the world of cryptography. We were given really interesting tasks, but also quite demanding. Sadly I wasn't able to find the keyword through python, but this also gave me a lot of valuable experience in cryptanalysis. As for part two, the tasks was a bit more manageable given that we already had the keys and also the ciphering algorithms.

Conclusion

We have seen that breaking encryption algorithms can be hard. You would need quite a bit of information about the algorithm used, the keys used and if there were multiple algorithms that where used. Taking that into consideration the state of information security is quite safe. On the contrary if a attacker was able to find a way to break even the most advanced algorithms today, then those algorithms would be completely defenseless. This shows the importance of constant development of the encryption algorithms, so that when the day comes where a algorithm is cracked, then we will be ready.

Other References

- https://www.geeksforgeeks.org/autokey-cipher-symmetric-ciphers/
- https://awwalker.com/2021/01/23/statistical-attacks-on-the-autokey-cipher/
- https://www.geeksforgeeks.org/columnar-transposition-cipher/
- https://flask.palletsprojects.com/en/2.2.x/quickstart/
- https://crypto.interactive-maths.com/frequency-analysis-breaking-the-code.html
-