THE UNIVERSITY OF BRITISH COLUMBIA
Electrical and Computer Engineering Department

CPEN 442 **Introduction to Cybersecurity** **Winter Term 1, 2024**

Assignment 1: Cryptanalysis

Due date: **Friday, September 20th, 2024, 11:59 pm**
Hard deadline: **Sunday, September 22nd, 2024, 11:59 pm**

Please use Piazza for questions and clarifications. Instructor and TA help will be provided only until 2pm on the same day as the *due date* above. This assignment has a 48-hour automatic deadline extension to account for unexpected issues when uploading the assignment. You are allowed to submit the assignment until the *hard deadline* date above, with no penalization. No exceptions after this date will be made!! If you have a long-term issue that will prevent you, or your group, from submitting the assignment, let the instructor know *well before the due date*.

Total marks: 100 pts
TA: no TA :(just the instructor `[simon.oya@ubc.ca]`

Doing the assignment: This assignment can be done either **individually** or in **groups of two**. Regardless of what you do, you must join a group set in Canvas (check the `People` section, then `A1 group`). The group can be just yourself. If you are not in Canvas: do not worry, you should be around Sep 16th, so you will have plenty of time to join a group and submit.

Note: for the second part of this assignment, your group number will the determine the ciphertext that you need to crack.

What to submit:

- A PDF file called `a1.pdf`, with the answers to questions below (Q2.2, Q5.1, Q5.2, Q6.1).
- A python file `a1.py`; please see the provided template and do not change the function headers.
- All Python scripts that you have used for this assignment.
- A plaintex and key that you recovered, as explained below.

1 Preamble

This assignment deals with the cryptanalysis of weak ciphers. The first part (Sections 2 to 3) focuses on the Vigenère cipher, and should be able to be completed in the lab session (although you do not need to submit anything by the end of the lab). The second part (Sec. 6) deals with the Playfair cipher; this requires more elaborate techniques, and you will probably not have time to finish this in the lab.

For simplicity, both plaintexts and ciphertexts in this assignment only contain uppercase ascii characters, i.e., from `A` to `Z` (26 characters).

1. For the first part of the assignment, you have the following files:
   - `plaintext1.txt` is a sample plaintext that, when encrypted with a Vigenere cipher with the key in `key1.txt`, yields ciphertext `ciphertext1.txt`.
   - `ciphertext2.txt` and `ciphertext3.txt` are used in the first part of the assignment. You will have to crack one of them in particular.
   - `a1.py` contains the headers of the functions you need to program for the first part of the assignment. We will import your function implementations from that file when evaluating the assignment, so do not modify the headers. You are free to include other packages, implement other functions, etc.
2. For the second part of the assignment, you have the following files:
   - `ciphertexts_playfair.txt` contains 90 ciphertexts generated with the Playfair cipher. You will need to crack the ith ciphertext, where i is your group number. The last 5 ciphertexts are given as an example.
   - `keys_playfair_last5.txt` these are the keys that have been used to generate the last five ciphertexts in `ciphertexts_playfair.txt`. These are just for you to sanitize your encryption/decryption algorithms, and your cracking script.
   - `plaintexts_playfair_last5.txt` these are the plaintexts that have been used to generate the last five ciphertexts in `ciphertexts_playfair.txt`.

Note: In order to encrypt and decrypt using the Vigenère cipher, it will be useful to "translate" characters into integers and vice-versa. A simple way to do this is with the Python functions `ord()` and `chr()`.

2 The Index of Coincidence (IC)

In the lectures, we saw the Index of Coincidence (IC), which is a metric that measures how likely is that, if you picked two letters at random from a text, both are the same letter.

If n_A denotes the number of times the character A appears in the text, n_B denotes the number of times the character B appears in the text, and so on, then the IC is computed as

$$IC = c \cdot \left(\frac{n_A}{N} \cdot \frac{n_A - 1}{N - 1} + \frac{n_B}{N} \cdot \frac{n_B - 1}{N - 1} + \dots \right) . \tag{1}$$

Here, c is the number of letters in the alphabet (in our case, for English, $c = 26$), and N is the length of the text.

For example, if the text was `ABAABC`, the IC would be

$$IC = 26 \cdot \left(\frac{3}{6} \cdot \frac{2}{5} + \frac{2}{6} \cdot \frac{1}{5} + 0 \right) . \tag{2}$$

[5 pts] Question 2.1: Implement the function `index_of_coincidence` (inside `a1.py`), which returns the index of coincidence of an input text.

Remember that different languages have difference ICs (you can check Wikipedia for a table with some examples) For example, English IC is ≈ 1.73, while French is ≈ 2.02. If you run your function over the text in `plaintext1.txt`, you should therefore obtain a number close to 1.73.

[5 pts] Question 2.2: Files `ciphertext2.txt` and `ciphertext3.txt` contain two ciphertexts. One of them has been generated by encrypting an English plaintext with a monoalphabetic substitution cipher, and the other one has been generated using a Vigenère cipher. Using *only the index of coincidence*: which ciphertext corresponds to which cipher (substitution or Vigenère) and why? Write your answer in the file `a1.pdf`.

3 Implementing the Vigenère cipher

Before cracking the Vigenère cipher, let's implement the encrypt and decrypt functions for this cipher.

[5 pts] Question 3.1: Program the functions `vigenere_encrypt` and `vigenere_decrypt`.

To test your encryption implementation, you can check that the text in `plaintext1.txt` yields the ciphertext in `ciphertext1.txt` when encrypted with the key in `key1.txt`. You can test your decryption using these files as well.

4 Cracking the key length of the Vigenère cipher

Before, you guessed that either `ciphertext2.txt` or `ciphertext3.txt` has been generated with a Vigenère cipher. From now on, use the ciphertext that corresponds to the Vigenère cipher. First, it will be helpful to guess the length of the key used to create this ciphertext.

[10 pts] Question 4.1: Implement the function `crack_key_length_vigenere`, which guesses the key length that was used to generate a target ciphertext encrypted with a Vigenère cipher. To implement this function, do not crack the key yet: you must do this by using *the index of coincidence* only! You can assume the key length is between 2 and 20 characters.

Hint: if you are stuck here, think about the example from the classroom:

```
Plaintext:  ATTACKTONIGHT
Key:        MYKEYMYKEYMYK
Ciphertext: MRDEAWRYRGSFD
```

This example shows that a Vigenère cipher with key length five can be seen as five different versions of the Caesar cipher: letters 1, 6, and 11 (`A`, `K`, and `G`), are "shifted" by the first letter of the key (`M`), letters 2, 7, and 12 (`T`, `T`, and `H`) are "shifted" by the second letter of the key (`Y`), and so on. In the previous section, you saw that you can detect such a monoalphabetic substitution cipher with the index of coincidence. How would you use the index of coincidence to figure out the key length?

Once you have programmed `crack_key_length_vigenere`, you can test it yourself with your own encrypt and decrypt functions. We will check that your function passes some randomly-generated test cases.

5 Cracking the Vigenère cipher

Finally, you will crack the Vigenère cipher. While there are different ways to do so, we will use a simple technique using a histogram of English letters. First, grab the frequencies of English letters (you can find this online, or you can compute the frequencies of a long English text).

[20 pts] Question 5.1: Using the frequencies of English letters, as well as the function that cracks the key length, implement a function that, given a Vigenère ciphertext, returns

the encryption/decryption key. In the text document (`a1.pdf`), *briefly* explain how you have cracked the key. Emphasis on *briefly* :).

You can test your script by running it on the ciphertext. We will run this particular test, but we will also test it with other ciphertexts.

[5 pts] Question 5.2: Will your script always be able to crack any ciphertext encrypted with the Vigènere cipher? If yes, explain why. If not, provide an example that it cannot crack. Write this answer in `a1.pdf`.

6 Cryptanalysis of the Playfair Cipher

While cracking the Vigènere cipher is relatively easy, the Playfair cipher requires somewhat more elaborate techniques. If you are doing this in the lab session, you will probably not have time to finish this. Grab the ith ciphertext[1] in file `ciphertexts_playfair.txt`, where i is your group number; this is the ciphertext you have to crack.

To generate the plaintexts, we replaced all `J` characters with `I`, then we added an `X` between *any* two consecutive identical characters, and we clipped the length of the resulting strings to 300 characters. Then, we encrypted each of these plaintexts using the Playfair cipher. The key for the Playfair cipher is a random permutation of `[A-Z]`, excluding `J`. Thus, the key has a length of 25 characters.

[50 pts] Question 6.1: Crack the ciphertext.

You must submit:

1. The plaintext that you have recovered, in a file called `plaintext.txt`. This should be a single line of 300 consecutive upper-case characters.
2. The key that you have recovered, in a file called `key.txt`. This should also be written as a single line of 25 consecutive characters.
3. In `a1.pdf`, include a brief explanation of how you have cracked the cipher.
4. Any code that you have used to do the assignment should also be included (if it is in files other than `a1.py`).

Using code. Using code to crack the ciphertext is optional but, if you do it, it must be included in the submission files. All the code that you use must be written in Python. You

[1] The first ciphertext is for group 1, the second for group 2, and so on. Do not get confused by the fact that Python starts indexing at 0.

can take inspiration from algorithms/approaches found on the Internet to do this cracking; if you do so, cite your sources. You can use existing code for running *English fitting* tests if needed, but you must *write any additional code yourself*. Manual approaches to cracking (e.g., by inspecting the ciphertext, not necessarily with a computer) are also valid.