Crack Ciphers

Due date: September 27, 2024 (Friday, at 11:59:59PM)

Problem Description

Your job is to crack encrypted messages. On the course website there are four files containing ciphertexts (cipher1.txt, cipher2, txt, cipher3.txt, and cipher4.txt). The messages are quite long, which should ease the difficulty in cracking the messages. For each message you should find out the type of encryption that was used, as well as the key and the plaintext to show you decrypted it properly (if it can be decrypted). If you cannot decrypt a message for whatever reason, explain why. Possible encryption schemes are:

- Shift Cipher
- Substitution Cipher
- Vigenère Cipher
- Permutation Cipher (Columnar Transposition)
- One-Time Pad

What To Do

The cracking process should be as automated as possible. Ideally, the program takes as input the ciphertext, and outputs the type of the encryption scheme, as well as the key and the plaintext (if it can be decrypted). However, it is not easy to fully automate this process since human effort is usually needed to verify the decrypted plaintext is a valid English text. Thus, in this project, your program should be able to take user input (where necessary) and decide the next step based on the user input.

You may use a variety of statistical analysis methods to determine the cipher type. Some of the methods have been covered in the lectures (see the below list). Therefore, you should at least implement these functions in the computer to help you carry out the mundane tasks. But before you start to crack the ciphers, you should first develop the logic and algorithm to determine the cipher types and decrypt messages, and then start to write your cipher-cracking program. Think about an overall strategy to reduce the time complexity of your cipher-cracking algorithms, as well as the human effort by making this whole process as automated as possible. Avoid brute-force attacks whenever possible, as your cipher-cracking algorithm should not use pure brute force.

Your cipher-cracking program should be written in a popular programming language that you prefer and are familiar with (e.g., C/C++, Matlab, Java, etc). It should compile and run without errors on the cse server. The program shall achieve our goals in this project with as little human effort as possible (e.g., manual inspection or interaction with the program). There is no need to

develop a graphical user interface (GUI), but if you use C/C++, outputting intermediate results and taking user inputs in a console/terminal is recommended.

How to use the CSE server: The School's server, nuros.unl.edu (we'll just call it CSE). You access CSE via simple terminals. In Windows, we recommend that you use PowerShell. For Mac, just use "Terminal" under "Utilities". Once you have your terminal open, you connect to CSE using the following command:
`ssh nuros.unl.edu -l your_UNL_username` Note that the flag, -l, is -(lowercase L).
OR
`ssh your_UNL_username@nuros.unl.edu`
Once you log in to CSE, you are in you home directory. Hit a few "return" keys and you should notice that the line keep scrolling up each time. That last line that the cursor is on is the command prompt.

The following is a minimum set of required functions that you should implement in this project:

- Frequency analysis: given a ciphertext string as input, outputs the frequencies of monograms and digrams of the ciphertext.
- Index of Coincidence (IC): given a ciphertext string as input, outputs the IC value.
- Decryption function: given a ciphertext string and a possible key, outputs the decrypted "plaintext" of a cipher (for each type of cipher in the provided ciphertexts).

You can use these and any additional functions you deem necessary to crack the ciphers. You may use known frequency distributions of English text (from lecture as well as the Internet).

Some optional functions:

- Shifted version of IC: given a ciphertext and a shift value as input, outputs the shifted IC value.
- Kasiski Test: given a ciphertext as input, outputs the five most frequent trigrams in the ciphertext, along with their appearing locations in the ciphertext, and the GCDs of the distance between consecutive trigrams for each frequent trigram.

In addition, you will need to write a separate function to identify the type of cipher, as well as functions to crack each type of cipher that you have determined to exist. (Hint: user input may be needed to verify the correctness of the decrypted plaintexts).

Note for an exception: for the substitution cipher, an automatic function of cracking this type of cipher is not necessary, which means it can be done manually (based on frequency analysis).

It is recommended that each of the above mentioned functions be written in a separate file with the specified input and output, to facilitate testing and grading. There should be a main file as the entry point of your whole program, which may call other functions.

Also note: certain online tools may help you explore the process of cracking ciphers, some may help you compute statistical functions. But you should still write your own program without

relying on those online tools in this project. All the above required functions should be implemented by yourself.

Report

You will need to submit a report, including an overview of your idea and method to crack the ciphers (logic flow charts and any other diagrams or charts are welcome to help describe your approach). Include not only the final results of the type of encryption, associated keys and plaintexts, but also the detailed actual intermediate steps you (and your program) have taken to deduce them (For example, statistical analysis results, the steps of determining each key, screenshots, etc). Also include any other interesting observations (how much manual validation/input was used, how many keys were tried before the correct one was discovered, etc.), and conclude with what you have learnt from this project (especially, anything unexpected?). Finally, attach a documentation in the appendix part that explains how to compile, run and use your program, and what each file is for. There are no page number limits nor specific format guidelines (except that the font should not exceed 12 points). Your report should be clear, concise, and complete. This should be a well-presented report with headings and other organizational methods to make it easy to read and understand. Generally, the report should be in PDF or DOC format.

Grading

This project is worth 100 points. The grade breakdown is as follows: report (60 points) and program (40 points). The report grading will be based on the correctness of your approach (40%), correctness of intermediate and final results (50%), and the quality of writing (10%). If you use pure brute force to crack all the ciphers, you will get a zero point for the approach part. The grading of your program will be based on whether it compiles and runs without error (30%), the inclusion and correctness of the required functions (30%), the correctness of the program logic and whether the program matches with what your report describes (40%). If things do not seem to work, you may be asked to meet with the grader to process the test files.

You can earn up to 10 bonus points. Points will be given based on the "coolness" of your program as compared to others. For example: Is it easy to use? The degree of automation (or does it require a small amount of user effort)? Does it have any extra features or options (that relate to the assignment or general purpose feature)? In your documentation you should state what you did that is deserving of these points.

Submission

Submit a zip file including your source code, and the report through Canvas. The file should be named "<Lastname>_hw1.zip".