**York College of Pennsylvania**

CS 497

Program Assignment: Breaking a Caesar Cipher

**Introduction**

When you hear the word "encryption," you might think about modern computers and things like email and online bank accounts. But did you know that encryption has been around for thousands of years? In this project you will learn about the Caesar cipher, a simple type of encryption that replaces each letter of the alphabet with another letter, and demonstrate how a modern computer can crack this ancient code in just a few seconds.

**Objective**

Write a Python program to decrypt text that has been encrypted with a Caesar cipher.

**General**

Have you ever wanted to send a secret message to a friend? What if someone, like a parent or a teacher, intercepts the message and reads it? In order to make sure that only your friend could read the message, even if it was intercepted, first you would need to encrypt it. **Encryption** is the process of encoding a message so only the intended recipient can read it. Encryption is used to protect many of our daily online activities, like emails and credit card transactions, from unauthorized access.

Modern encryption algorithms are very complicated and (ideally) difficult to break. However, encryption has been around for thousands of years—long before computers existed. Leaders throughout history have used various types of encryption to send messages to allied countries and military leaders during wartime. One famous example is the **Caesar cipher**, used by Julius Caesar in ancient Rome. The Caesar cipher is an example of a **substitution cipher**, where each letter of the alphabet (in English, 26 letters) is replaced by another letter of the alphabet. This is done by "shifting" the entire alphabet by a certain number of spaces. This number is called the **key**. For example, here is a shift of 3 (note how the alphabet "wraps around" from the end):

|  |  |
| --- | --- |
| Original alphabet: | ABCDEFGHIJKLMNOPQRSTUVWXYZ |
| Shifted alphabet: | DEFGHIJKLMNOPQRSTUVWXYZABC |

To encode a message, each letter in the original message (called the **plaintext**) is replaced with the letter directly below it in the shifted alphabet (A becomes D, B becomes E, and so on). The result is called the **ciphertext**. Here is a plaintext message encrypted using a shift of 3:

|  |  |
| --- | --- |
| Plaintext: | THIS IS A SECRET MESSAGE |
| Ciphertext: | WKLV LV D VHFUHW PHVVDJH |

In order to share secret messages, you and your friend need to agree on a key in advance. Then, you can use the key to encrypt messages, and your friend can use the same key (shifting the alphabet in the opposite direction) to decrypt them. Anyone who intercepts the messages will be unable to read them if they do not know the key.

But, what if a very determined person wants to crack your code? How could they do it? One major weakness of the Caesar cipher is that it is vulnerable to a **brute-force attack**, an attack that tries all possible keys to decrypt a message. Since there are only 25 possible keys in English (using a key of 26 gets you back to the original alphabet), for very short encrypted messages it would not take you long to manually try all the keys. For example, here is a short encrypted message (note that this simple version of the Caesar cipher only changes letters; punctuation remains unchanged).

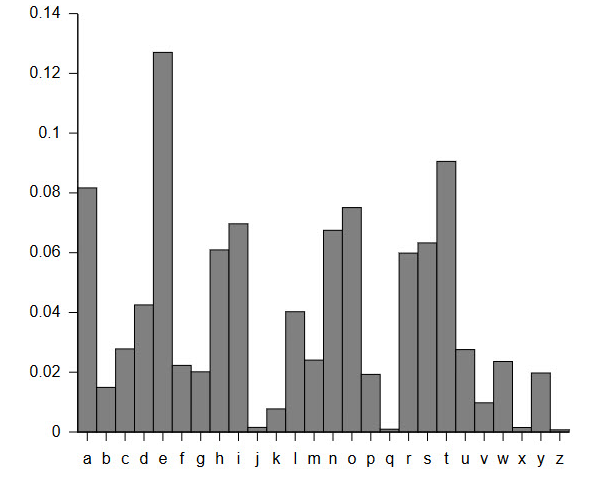
RPC NDJ RGPRZ IWT RDST?

What happens if we try to decrypt this message using a shift of 1? That would mean that during encryption, A became B, B became C, and so on. To decrypt the message, we work backwards (B becomes, A, C becomes B, and so on). If we try this on the entire message, we get this result:

QOB MCI QFOQY HVS QCRS?

The message is still gibberish, so we know that 1 is not the key (assuming the original message was actually in English!). Can you try to decrypt the message using the other 24 possible keys? Keep trying different keys until you get a sentence that makes sense in English. How long does it take you to do it by hand?

Another method that can be used to crack a Caesar cipher (or any other type of substitution cipher) is **frequency analysis**. Frequency analysis is based on the fact that certain letters appear with different frequencies in English writing—for example, E usually occurs the most often, followed by T and A; whereas Q and Z appear the least often (Figure 1).



**Figure 1.** Letter frequencies in the English language.

For example, look at this encrypted text:

|  |
| --- |
| L PZ AOL TVZA MYLXBLUA SLAALY PU AOPZ ZLUALUJL |

If you count the letters, you will notice that L appears more often than any other letter (9 times). It is therefore a safe guess that L stands for E if this is a substitution cipher and the original message was in English. L is 7 spaces away from E in the alphabet. What happens if you work backwards to decrypt this message using a key of 7 (L becomes E, M becomes F, and so on)?

Doing a brute-force attack or frequency analysis by hand can be easy for very short messages, but can become time-consuming for entire paragraphs or pages of text. This is where writing a computer program to do the work for you comes in handy. In the procedure of this project, you will write your own programs that can first encrypt plaintext using a Caesar cipher, and then attempt to decrypt the text using both a brute-force attack and frequency analysis.

**Tasks**

You might want to make sure you know how to use the following features of Python before you start.

• Lists

• Strings

• IF statements

• FOR loops

• The **modulo** operator (%)

If you get stuck when writing your program, an online search for something general (like "python if statement") or specific (like "how to read a string from a text file in python") will typically give helpful results.

1. On your computer, write a sentence or short paragraph (or copy one from this page) and save it as a text file.

2. Write a program that:

a. Reads a plaintext string from a text file.

b. Encrypts the string using a Caesar cipher with a randomly generated key. You can make your program only change the letters A-Z and leave other characters (numbers, punctuation, spaces) unchanged.

c. Saves the ciphertext to a new text file.

3. Write a program to perform a brute-force attack on the ciphertext. Refer to the **General** section if you need a reminder about how a brute-force attack works. The program should:

a. Load the encrypted string from the text file.

b. Try all 25 possible keys to decrypt the ciphertext, saving each result in a new string.

c. Look at all 25 resulting strings. Most of them should be gibberish. Do any of them make sense? Can you figure out which one was the correct key?

4. Write a program to perform frequency analysis on the ciphertext. Refer to the **General** section if you need a reminder about how frequency analysis works. The program should:

a. Load the encrypted string from the text file.

b. Count how many times each letter occurs in the ciphertext, and find the letter that occurs most often.

c. Use this information to calculate the key (assuming the most common letter corresponds to the letter E in plaintext).

d. Decrypt the text using the key you calculated. Does the resulting plaintext make sense? If not, what do you think went wrong? (hint: be careful with frequency analysis, E might not be the most common letter in individual sentences or short paragraphs)

5. Test your program. Test your program on the following blocks of text. Which approach works better for each message, brute force or frequency analysis?

Example 1: BNMFQZSTKZSHNMR! XNT GZUD BQZBJDC SGD BNCD!

Example 2: UZU PFL KYZEB KYZJ GIFAVTK NRJ WLE? TYVTB FLK KYV

CVRIE DFIV JVTKZFE KF CVRIE RSFLK TRIVVIJ ZE TPSVIJVTLIZKP.

**Deliverable(s)**

Prepare a report that contains:

* + Python program
  + Plaintext and ciphertext messages you used
  + Screenshots showing your results
  + Discussion on your results
  + Explain what you learned from doing this program

Use a cover sheet with your name, class number and date.

**Grading**

Post your report in Moodle by the scheduled due date in the syllabus. Your grade for this lab will be composed of:

* 33% - Design
* 33% - Observations
* 34% - Explanation

**References**

• Learn Cryptography (n.d.). *Caesar Cipher*. Retrieved July 11, 2017 from <https://learncryptography.com/classical-encryption/caesar-cipher>

• Rodriguez-Clark, D. (n.d.). *Frequency Analysis: Breaking the Code*. Crypto Corner. Retrieved July 11, 2017 from <http://crypto.interactive-maths.com/frequency-analysis-breaking-the-code.html>

• Shaw, Z. (n.d.). *Learn Python the Hard Way*. Retrieved July 11, 2017 from <https://learnpythonthehardway.org/python3/>