CSC140-2 Project 3

Keep your secret safe – Multi-Alphabet Substitution Cipher

If you watched the movie "Imitation Game", you would know about the Enigma Machine, which was used by the Nazi Germany during World War II to encrypt secret messages, and how Turing was trying to crack it. Or probably you are a fan of Sherlock Holmes and know about him breaking the dancing man code. We will try to do a similar thing in this project: encrypt and decrypt messages. **Do not worry. The method we will be using is quite straight forward and easy to code.** This project is worth 15 pts. of your projects grade. The rubric for this project will be specified at the end of the project description.

We often want to send information to another person secretly, like a love letter, a military secret, or simply an email. The desire for secrecy has not decreased over time, and is valued by individuals as well as governments. Given the amount of sensitive information being transmitted online (like your online banking information, login credentials, e-mail messages and chat/text messages) are encrypted so that anyone who gathers this information cannot read or understand it's meaning. You might hear about the hackers or attackers, or read about a leak of user information. If the text being sent out is not encrypted, the attackers can easily grab this information from the network, read it and use it.

In this project, we will not be carrying out the complex algorithms that modern companies are using to protect your online information. Instead, we will try to implement **a simple historical encrypting method**, which is the root of the modern methods and the basic idea remains the same. It is not strong enough to protect your secret from modern super computer and FBI agents, but at least, if you write your letter encrypted using the historical methods, a hacker who accesses your data will not be able to (easily) understand it.

The method we will be using is called Multi-Alphabet Substitution Cipher. Before explaining how it works, let us look at the simplest method in the history to have **a feel of encryption**, which is called **Caesar Cipher** or Shift Cipher related to a Roman general called Gaius Julius Caesar (100-44 B.C.).

The sender and the receiver will have a shared key, which they both know and keep secret from the others. For example, the key might be the number 3 (this is a ***secret, shared key***). When sender is writing a message, every letter will be shifted to the letter that is 3 positions ahead in the alphabet (this is the ***encryption process***). Using the key, the word "apple" (this is the ***plain text***) will become the word "dssoh" (this is the ***cipher text***), which hopefully does not make sense to the attackers. When the intended

recipient receives the message “dssoh”, he will then move every letter 3 positions behind and recover the message “apple” (this is the ***decryption process***).

This approach is very easy to implement, but it is also very easy to break. Cetain English letters have identifying characteristics: for example, a frequency analysis can be done to help detect letters like 'e' or 's'. Also, there are certain one-letter words that help identify 'a' and 'I'. It is easy to determine the letters that appear the most frequently in a cipher text message and then discover the secret key (statistical breaking)-- **It is how Sherlock Holmes broke the dancing man code!** It would not take much time to work out a brute force attack, given the small number of possible keys (just try out all the possible keys).

Now we are finally ready to see how **Multi-Alphabet Substitution Cipher** works. It is also simple, but more difficult to break than a Caesar Cipher.

1 We have 3 random permutations of the letters in the English alphabet as the secret key, no one outside of this class should know about. They can be stored as string: *key0, key1, key2*. These random permutations are easy to generate using python (random.shuffle()).

Plain text alphabet: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Cipher text alphabet 0: J D M K U F L N R Y A C G S V T B I P Q O W E H Z X
Cipher text alphabet 1: S Q W H D U Z L B V E C A O Y K P J R F N G M I T X
Cipher text alphabet 2: A Y G K O I D M X J U V S L Q R E Z N H P B T F C W

2 Every letter in the plain text can be substituted by a letter from the key. The first letter in the plain text can be replaced by the first letter in the key0; the second letter in the plain text can be replaced by the second letter in the key1; the third letter using the third letter in the key2; the forth letter, going back, using the forth letter in the key0; and so on.

For example, we want to encrypt “HELLO”, from plain text to cipher text using our key-the 3 randomized alphabet:

A	B	C	D	***E***	F	G	***H***	I	J	K	***L***	M	N	***O***	P	Q	R	S	T	U	V	W	X	Y	Z
J	D	M	K	U	F	L	***N***	R	Y	A	***C***	G	S	V	T	B	I	P	Q	O	W	E	H	Z	X
S	Q	W	H	***D***	U	Z	L	B	V	E	C	A	O	***Y***	K	P	J	R	F	N	G	M	I	T	X
A	Y	G	K	O	I	D	M	X	J	U	***V***	S	L	Q	R	E	Z	N	H	P	B	T	F	C	W

H is replaced by the letter at the same position in the key0, N; E by the letter at the same position in the key1, D; L by the letter at the same position in key2, V; L by the letter at the same position in key0, C; O by the letter at the same position in key1, O. The final cipher text is “NDVCY”.

3 Ok, sounds easy to human, but…How to code this?

1) Download and save the file called "SamplePlainText.txt" which is available on Moodle. Save this file in the python folder that you save your .py files to.
2) Create a helper function that creates a file containing the shared keys. This function should randomly permute, or shuffle, the letters of the alphabet (all capitals) three times, and each result should be written to a text file. This text file should only contain 3 lines. Each line should contain a string with 26 characters (no spaces, all capital letters). For example, one line in the file might be the string JDMKUFLNRYACGSVTBIPQOWEHZX. Call this text file "SharedKeys.txt" and save it in the same directory as the "SamplePlainText.txt" file that you downloaded from Moodle. Call this function *CipherKeyGenerator()*.
3) Write a python function that carries out this encryption process. Use the naming scheme: *MultiAlphaCipher()*. Your function should do the following:
    a. Read the contents of the input file: "SamplePlainText.txt" that you downloaded. Also, read the contents of the "SharedKeys.txt" file that your helper function created.
    b. For each letter in "SamplePlainText.txt", use the appropriate shared key to find its replacement letter and generate the corresponding cipher text (the first line should be used for the first letter, the second line for the second letter, the third line for the third letter, and so on.). Do not change the case of the letter! Any other symbols contained in the plain text file should not be changed.
    c. Write the ciphter text to a file called "MyCiphterText.txt"

- Hint 1: changing the case of the letter as part of the encryption process might be helpful
- Hint 2: a helper function that gets the index that a letter exists in a list might be useful
- Hint 3: try writing a function that works when a message is in ALL CAPS first, and then adjusts for lower case letters and other symbols

How long would it take to try to decrypt this message? If an attacker (for example, someone from another class who wanted to know the contents of your message…) is trying to determine the key using brute force, then they have to try all the possible keys, which are all possible combinations of 3 alphabets. There would be 26!=26*25*24*…*2*1 possible letter permutations to form one alphabet. And for 3 alphabets, that would be 26!*26!*26!. In real life, it is reasonable to use 5 alphabet instead of 3, which is (26!)^5≈2^441 possible keys to try. Also, every letter was changed to 3 different letters based on its position. If the attacker does not know how many alphabets you are using, it is hard to do a statistical break. Also, we weakened the bond between a given letter in the plain text to its corresponding letter in the cipher text…this helps to undermine frequency analysis on the cipher text. It is just as simple as the Caesar Cipher, but much safer. This is because we are using a much longer secret key.

We have a pretty safe way to encrypt our plain text now. But the receiver need to know how to decipher it. Since the receiver knows the key, it is also easy. The receiver just needs to:

1) Write a python function named: *MultiAlphaDecipher()* that reads in the cipher text stored in the file "MyCiphterText.txt" and the shared key file "SharedKeys.txt", and decrypts the cipher text:
    a. Open the file containing the cipher text that was created in the previous section. This file was named: "MyCipherText.txt"
    b. Build the plain text message from the cipher text message:

i. For every letter in the cipher text message, find its index in the appropriate key alphabet string.
ii. Then find the corresponding plain text letter at the index

c. Write the decrypted message to a file called "MyDecryptedText.txt"

Your project submission should include as least these three functions. And they should call small helper functions to finish the job.

Important: The more reasonable small helper functions that you have, the easier for you to debug. Making small ones work and link them together is the ultimate way to code efficiently. It would be a headache and sometimes impossible to debug a large piece of messy code (which requires advanced logical thinking and coding skill). A common result for a large piece of messy buggy code without good structure (little function links)--- rewrite from scratch and restructure it better. We do that from time to time as programmers when not able to fix old code that other people or even ourselves wrote. Patching bugs in messy code often makes it worse to maintain, so next time using the code or add a new functionality will be even harder.

- CipherKeyGenerator()
- MultiAlphaCipher()
- MultiAlphaDecipher()

Your code now implements the Multi-Alphabet Permutation Cipher. This is a website to cipher your message to dancing man code, have fun:

- http://www.geocachingtoolbox.com/index.php?lang=en&page=dancingMen

Grading rubric: 15 pts total

Code with clear comments

input/output larger comments on top of every function like we have in exercises that clearly define what this function is doing –1 pt

comments of important steps, telling what your code is doing each part of the code, like before loops and the conditions for if-else –1 pt

Good structure

consists of reasonably separated small helper functions that piece up the bigger function (reference project 1) –1 pt

CipherKeyGenerator()

Creates file with correct name –1pt

The key file has correct content –1 pt

MultiAlphaCipher()

Opens the two file indicated and reads the contents –1pt

→Able to encode All UPPER CASE letter plain text—1pt

→Able to encode upper case letter with other symbols –1pt

→Able to encode normal sentence with mixture of upper/lower letter and common symbols—1pt

Writes the ciphter text to an output file, with correct name—1pt

MultiAlphaDecipher()

Opens the two files indicated and reads the contents –1pt

→Able to decode All UPPER CASE letter plain text—1pt

→Able to decode upper case letter with other symbols –1pt

→Able to decode normal sentence with mixture of upper/lower letter and common symbols—1pt

Writes the plain text to an output file, with correct name –1pt

Overall:

The contents of "SamplePlainText.txt" and "MyDecryptedText.txt" should be the same: after the encryption and decryption of a file, the original text is reproduced.