CMPUT 299, Winter 2018, Assignment 4

Acknowledge all resources consulted (discussions, texts, urls, etc.) in working on this assignment. Without this acknowledgement, your assignment will not be graded. Non-detailed oral discussion with others (including non-students) is permitted as long as any such discussion is summarized and acknowledged by all parties; the viewing or exchanging of any written work, even in rough or preliminary form, is expressly forbidden, as is any detailed discussion. Late assignments will not be accepted.

One problem in this assignment takes the form of a short-answer question. Your answer should be submitted as a PDF named “a4.pdf” or a plain text file named “a4.txt”.

You will produce four files for this assignment: caesarHackerAuto.py (for problem 1), autokeyHacker.py (for problem 2), “a4.pdf” or “a4.txt” (for problem 3). and a README file (also in either PDF or plain text), indicating who you worked with, which resources you consulted, the name of your collaborator (if yours is a joint submission), and any other relevant information as indicated in the first paragraph. For your submission, put all of these files in a directory called 299-as-4. Zip this directory so that you have a new file called 299-as-4.zip. To submit the assignment, upload your zipped assignment to the assignment page in eClass.

Problem 1

In chapter 7, you saw a program to break the Caesar cipher through brute-force. We say that this program is a *partially automated* cipher breaker, since it requires some manual effort: a human still needs to look at the 26 possible decipherments, and choose the best one.

In general, it is preferable to have a *fully automated* cipher breaker, one that requires no manual effort at all. The user needs only to provide the ciphertext, and the program will output the plaintext, and the correct key, with no other human action being required.

Use the “detectEnglish.py” module and code from the “caesarHacker.py” program to produce a module called “caesarHackerAuto.py”. It must contain a function named “autoBruteForce”, which takes a single argument, a string containing a ciphertext. It should return a dictionary containing decipherments for which “detectEnglish.isEnglish” returns true. Each value should be such a decipherment (as a string), and it’s key should be the key (a number between 0 and 25, inclusive) which generates it.

Here are some sample function calls which your code should be able to reproduce:

```
>>> import caesarHackerAuto
>>> caesarHackerAuto.autoBruteForce('GUVF VF ZL FRPERG ZRFFNTR')
{13: 'THIS IS MY SECRET MESSAGE'}
>>> caesarHackerAuto.autoBruteForce('NKXK OY G YGSVRK OTVAZ LUX EUAX VXUMXGS')
{6: 'HERE IS A SAMPLE INPUT FOR YOUR PROGRAM'}
>>> caesarHackerAuto.autoBruteForce('ESP ZYWJ HZCO DAPWWPO TYNZCCPNEWJ TY L
RZZO OTNETZYLCJ TD TYNZCCPNEWJ')
{11: 'THE ONLY WORD SPELLED INCORRECTLY IN A GOOD DICTIONARY IS INCORRECTLY'}
```

Problem 2

Included with this assignment is a Python module named "autokeyCipher.py". As its name suggests, it implements the *autokey cipher*, which is the cipher you implemented in assignment 2 problem 4. (You may wish to take a moment to review the assignment 2 specification to familiarize yourself with this cipher.) It has two functions, "encrypt" and "decrypt", each taking two arguments, a keystring and a plaintext or ciphertext (respectively). For example, the following all evaluate to true:

```
autokeyCipher.encrypt('cdgd', 'HELLO, WORLD') == 'JHROV, AZCZZ'
autokeyCipher.decrypt('cdgd', 'JHROV, AZCZZ') == 'HELLO, WORLD'
autokeyCipher.encrypt('thirteen', 'ENCRYPTED WITH A SHIFT OF THIRTEEN')
  == 'XUKIRTXRH JKKF P LLLBB HM TZPZYXSS'
autokeyCipher.decrypt('thirteen', 'XUKIRTXRH JKKF P LLLBB HM TZPZYXSS')
  == 'ENCRYPTED WITH A SHIFT OF THIRTEEN'
```

Since the number of possible keystrings is exponential in the length of the keystring (i.e. there are 26^n keystrings of length n), brute-force attacks are unlikely to succeed. However, improper usage of the cipher can lead to an exploitable weakness: if the keyword the user chooses is an actual English word, the cipher can be quickly broken by simply trying every word in an English dictionary. The number of keystrings may be too large for a brute-force attack to be effective, but the number of English words is small enough to make this kind of cryptanalysis feasible. This is called a *dictionary attack*, and it is closely related to the *brute-force attack*.

Create a Python module "autokeyHacker.py", which uses the "detectEnglish" and "autokeyCipher" modules. It must contain a function "dictAttack", which takes a single argument, a string containing a ciphertext, and attempts to break the cipher by using every word in dictionary.txt, one at a time, as the key for the autokey cipher. This function should return a dictionary containing decipherments for which "detectEnglish.isEnglish" returns true. Each value should be such a decipherment (as a string), and it's key should be the key (a word from "dictionary.txt") which generates it. For example:

```
>>> import autokeyHacker
>>> autokeyHacker.dictAttack('XUKIRTXRH JKKF P LLLBB HM TZPZYXSS')
{'THIRTEEN': 'ENCRYPTED WITH A SHIFT OF THIRTEEN'}
```

Problem 3

The included file "ciphers.txt" contains ten ciphertexts[1], one per line, each enciphered independently with one of the Caesar cipher, the autokey cipher (with the keystring being a word in "dictionary.txt"), or the transposition cipher. Using the code from the textbook, as well as the code you wrote for problems 1 and 2, decrypt all ten ciphertexts.

Your a4.pdf or a4.txt should contain the ten plaintexts *in the same order in which their respective ciphertexts are presented* (e.g. the seventh plaintext in your file should be your solution to the cipher given on the seventh line of "ciphers.txt").

IMPORTANT NOTE: in order to get useful results, you may need to adjust the parameters of the "isEnglish" function.

[1]Credit: plaintexts two through nine were copied or adapted from Wikipedia.