Programming Assignment for CS 419— Computer Security

1 Breaking Monoalphabetic Substitution Cipher

I have an implementation of monoalphabetic substitution encryption. Here is an example:

```
import sys
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def main():
    myMessage = 'To be, or not to be, that is the question.'
    myKey = 'QWERTYUIOPASDFGHJKLZXCVBNM'
    checkValidKey(myKey)
    translated = encryptMessage(myKey, myMessage)
    print(translated)

def checkValidKey(key):
    keyList = list(key)
    lettersList = list(LETTERS)
    keyList.sort()
    lettersList.sort()
    if keyList != lettersList:
        sys.exit('This is not a valid monoalphabetic substitution cipher key!')

def encryptMessage(key, message):
    translated = ''
    charsA = LETTERS
    charsB = key
    for symbol in message:
        if symbol.upper() in charsA:
            symIndex = charsA.find(symbol.upper())
            if symbol.isupper():
                translated += charsB[symIndex].upper()
            else:
                translated += charsB[symIndex].lower()
        else:
            # symbol is not in LETTERS, just add it
            translated += symbol
    return translated

if __name__ == "__main__":
    main()
```

Your job in this programming assignment is to break it. Specifically, you need to implement a program that takes an encrypted FILE as the input, and output the key. You are also required to submit a typed PDF document explaining your design.

- Programming language: Python 3 is recommended. You can use others as well. Be sure to inform the TAs if you are not using Python. For whatever programming language you use, please make sure it can run on iLab machines without extra configuration.
- Submissions: 1) one typed PDF with no hand-written texts/images; 2) and source code with necessary data files. If you are using Python, please submit one or multiple `*.py` files. You can use Juypter Notebook for development and documentation, but please export it to Python scripts when submitting the coee and/or convert it to PDF when submitting the report.
- Key space and encryption: as indicated by the given example, we only substitute 26 letters, not symbols. Also, upper/lower cases are kept.
- Message length: each message will have over 2000 letters.
- Inputs for your decryption program: there is only one – the file name of the encrypted message file. Each time, you only need to decrypt one message.
- Output for your decryption program: one file `key.txt` containing the decrypted key for the given message. The output file name should be hardcoded.
- Format of output key file: a list of 26 upper case letters following the alphabetic order. An example is line 6 in the given example.
- Testing: you are encoraged to use the given encryption code to test your own program. The encryption logic will be the same.
- Document: the document should explain how you code works (the design of your decryption) and how to run the code.
- Extra helper data files: you are allowed to upload extra data files to help improve the accuracy of your attack as long as its size is allowed by Canvas.
- Grading: there will be 8 test cases/messages. Each message is encrypted with a different key. You get 1 point for each correct key. You get partial credits for getting them partially correct (scaled by percentage). All test cases are drawn from real word articles. The document explaining the design worth 2 points.

Hints:

- Frequency analysis from Wikipedia: `https://en.wikipedia.org/wiki/Frequency_analysis`
- Most common English letter(s)/word(s): `https://www.simonsingh.net/The_Black_Chamber/hintsandtips.html`
- Recall that all our messages are collected from natural English articles. For letters with similar frequencies in the ciphertext, it is perhaps good to check letters around them.