Beau Hodes
EECS 565 Mini Project 1 Report

First, I will shortly discuss my encryption algorithm. I begin by tracking which letters were uppercase, so that I can reconvert those indices to uppercase again at the end. Next, I make the key and the plaintext both lowercase, to make it easier on myself. Then, I encrypt the text by adding x amount of letters (where x is the current index of the letter in the key, as we scroll through the key and loop over it again if needed) to the current plaintext letter, and using mod 26 on this result so that the result can be converted to a letter and added to the encrypted text. Then, once encrypted, I recapitalize the letters (by index) that were capitalized in the original plaintext and return this encrypted text.

Now, for decryption. I began by making the dict.txt into a set, which makes it fast to check for membership. Since all of dict.txt is uppercase, I then make my ciphertext uppercase. Then, I created a variable called firstWordCT to remember what the ciphertext is for length of the first word. After this, I loop through every possible combination of letters of length keyLength, and, for each, decrypt firstWordCT to get a tempPT, which represents the plaintext first word for that key. If tempPT is in the dictionary set, then I proceed on with decoding the rest of the text and print that plaintext and the key I used. Not decoding all the text until it is needed (just decode first word before checking for membership in the dictionary) makes my algorithm faster. Here are my results (run on my Macbook Pro) with time measured in seconds:

```
1. .Plaintext candidate:  CAESARSWIFEMUSTBEABOVESUSPICION  with key:  KS
Total time of execution:  0.06647396087646484

2. Plaintext candidate:
FORTUNEWHICHHASAGREATDEALOFPOWERINOTHERMATTERSBUTESPECIALLYINWARCANBRIN
GABOUTGREATCHANGESINASITUATIONTHROUGHVERYSLIGHTFORCES  with key:  KEY
Total time of execution:  0.1012568473815918

3. Plaintext candidate:  EXPERIENCEISTHETEACHEROFALLTHINGS  with key:  IWKD
Total time of execution:  1.1968870162963867

4. Plaintext candidate:  IMAGINATIONISMOREIMPORTANTTHANKNOWLEDGE  with key:
KELCE
Total time of execution:  30.38917112350464

5. Plaintext candidate:
EDUCATIONISWHATREMAINSAFTERONEHASFORGOTTENWHATONEHASLEARNEDINSCHOOL
with key:  HACKER
Total time of execution:  680.9385621547699
```

Obviously, even by knowing the key length and first word length (and having a dictionary), it still takes quite a while to brute force this. The runtime grows exponentially depending on complexity of what you are trying to decode (complexity meaning the key length and first word length growing, but mostly the key length since you must test every combination of letters of that length).