```python
#  KGBerryKrunch
#  Not Your Average Cereal Box Cipher
#
#  Author: Ryan Kramlich
#  I am writing this in python so that I may easily translate it into Objective-C for my iPhone project.
#  If you are fortunate enough for me to share this code with you, then I probably trust you as this is a secret algorithm of mine ;)
#
#  THIS IS THE MODULE VERSION(all interaction in command line)

#  imports and globals

import sys

letterArray = ['A','a','B','b','C','c','D','d','E','e','F','f','G','g','H','h','I','i','J','j','K','k','L','l',
 'M','m','N','n','O','o','P','p','Q','q','R','r','S','s','T','t','U','u','V','v','W','w','X','x','Y','y','Z','z',
    '0','1','2','3','4','5','6','7','8','9']
space_code = 'q34'
manual = '''
USAGE: python LibKGBerryKrunch.py [Operation] [Message] [Password]

[Operation] - E for Encrypt, D for decrypt.

[Message] - Depending on operation, message to be encoded or encoded message.
(If the message is more than 1 word, put it in quotes)

[Password] - Password
'''

#  Methods

def main():
    check_params()
    option1 = sys.argv[1]
    option1 = option1.upper()
    phrase = sys.argv[2]
    key = sys.argv[3]

    if option1 == 'E':
        phrase = cleanse(phrase)
        phrase = phrase.replace(' ',space_code)
        key2 = cleanse(key)
        if key != key2:
            print 'Invalid password. Letters and numbers only.'
            sys.exit()
        print encrypt(phrase,key)
    else:
        print decrypt(phrase,key)

def check_params():
    if len(sys.argv) != 4:
        print manual
        sys.exit()
    if sys.argv[1] != 'E' and sys.argv[1] != 'D':
        print manual
        sys.exit()
    if sys.argv[1] == 'D' and ' ' in sys.argv[2]:
        print manual
        sys.exit()

def cleanse(phrase):
    phrase = phrase.replace('&','')
    phrase = phrase.replace('/','')
    phrase = phrase.replace('*','')
    phrase = phrase.replace('_','')
    phrase = phrase.replace('?','')
    phrase = phrase.replace('!','')
    phrase = phrase.replace(',','')
    phrase = phrase.replace('.','')
    return phrase

def encrypt(phrase,key):
    finalPhrase = ''
    finalPhrase_dashes = ''
    keyLength = len(key)
    for i in range(len(phrase)):
        finalPhrase += addLetters(phrase[i], key[i%keyLength])
    for i in range(len(finalPhrase)):
        finalPhrase_dashes += finalPhrase[i]
        if i != 0 and i % 4 == 0:
            finalPhrase_dashes += '-'
    if finalPhrase_dashes[len(finalPhrase_dashes) - 1] == '-':
        finalPhrase_dashes = finalPhrase_dashes[:len(finalPhrase_dashes) - 1]
    return finalPhrase_dashes

def decrypt(phrase,key):
    phrase = phrase.replace('-','')
    finalPhrase = ''
    keyLength = len(key)
    for i in range(len(phrase)):
        finalPhrase += subtractLetters(phrase[i], key[i%keyLength])
    finalPhrase = finalPhrase.replace(space_code,' ')
    return finalPhrase

def addLetters(a,b):
    abSum = letterArray.index(a) + letterArray.index(b)
    sumLetter = letterArray[abSum % len(letterArray)]
    return sumLetter

def subtractLetters(a,b):
    if letterArray.index(a) > letterArray.index(b):
        abDiff = letterArray.index(a) - letterArray.index(b)
    else:
        abDiff = len(letterArray) - (letterArray.index(b) - letterArray.index(a))
    diffLetter = letterArray[abDiff % len(letterArray)]
    return diffLetter

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