|  |
| --- |
|  |
| ASSIGNMENT 3 |
| Breaking Transposition Ciphers |

|  |
| --- |
| Sukh Atwal  10-11-2020 |

Contents

[Introduction 2](#_Toc53590403)

[How to Use 3](#_Toc53590404)

[How it works - Walkthrough **Error! Bookmark not defined.**](#_Toc53590405)

[Testing 5](#_Toc53590406)

[Test 1 6](#_Toc53590407)

[Test 2 7](#_Toc53590408)

[Test 3 10](#_Toc53590409)

[Test 4 14](#_Toc53590410)

# Introduction

The objective of this assignment was to design and implement an application that goal was to break a transposition cipher. This would be done through brute forcing using every possible key length.

The language I used was Python due to ease of use and experience with using it along with the sys and math modules. I also used the scripts that were provided which included the encryption, decryption, English detection, and dictionary text file.

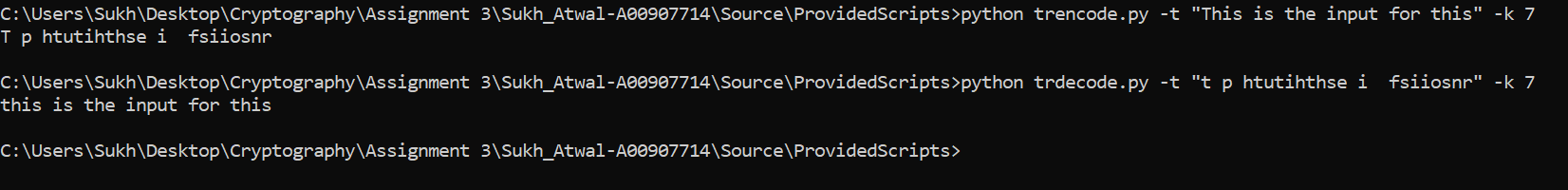
# How to Use

If you want to encrypt a message, go to the ‘ProvidedScripts’ in the Source folder and run the trencode.py through the terminal. The arguments are:

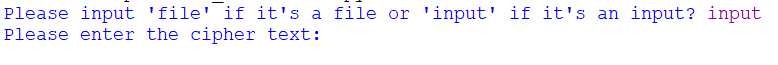
-t “Input whatever you want to encrypt here” -k “Enter the key size”

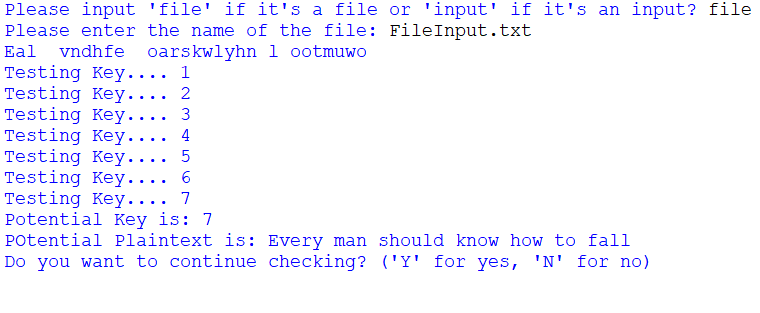
In the same folder, there is also a way to decrypt the encrypted message if you know the key. You would run the trdecode.py through the terminal and use the following arguments:

-t “The encrypted text” -k “The key size”



Finally, in the ‘Source’ folder there is a script to break a transposition cipher if you don’t know the key size. You would run the script through the terminal where it would ask if it’s a file or input you want to use. 

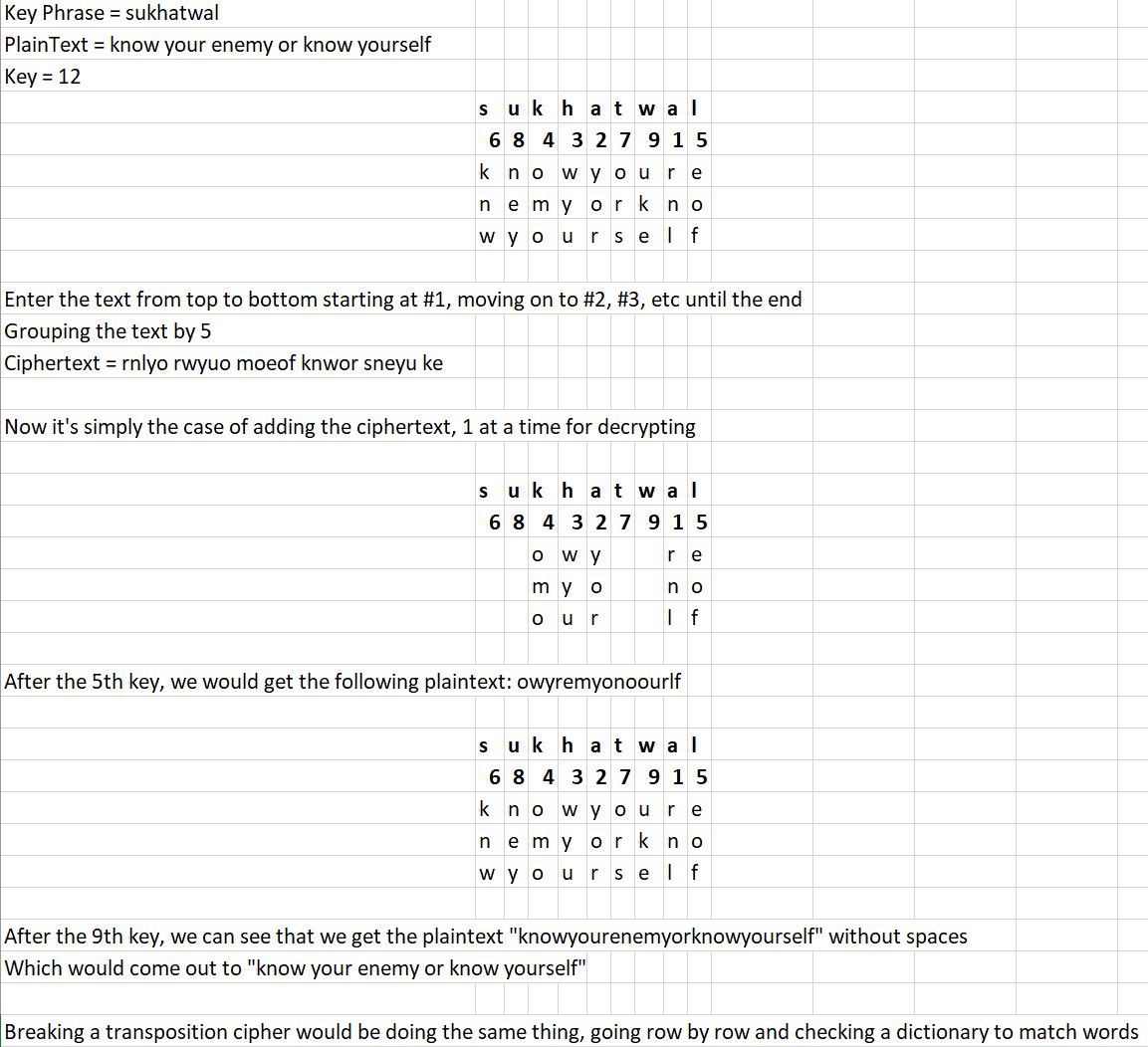
If we choose input, we would then manually enter or paste the cipher text

If we choose file, we would enter the file name with the encrypted text. In this case, I already have a file called FileInput.txt which I entered, and the following is the output. 

At the end, it’ll ask if you want to continue checking (Y) or stop checking (N).

# How it Works - Diagram

The following diagram is an explanation of how the encryption and decryption of transposition ciphers can work.



Essentially, breaking the transposition cipher would just revolve around an automatic process that tries every key until it finds a combination of letters that matches another combination of letters in a dictionary script that your script would check with.

# Testing

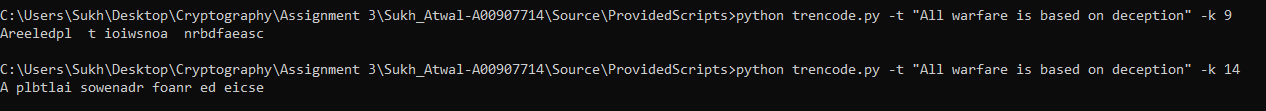
My testing setup is a Windows computer running Windows 10 using IDLE as my Python development platform of choice. You can also view all the tests in the ‘Videos’ folder.

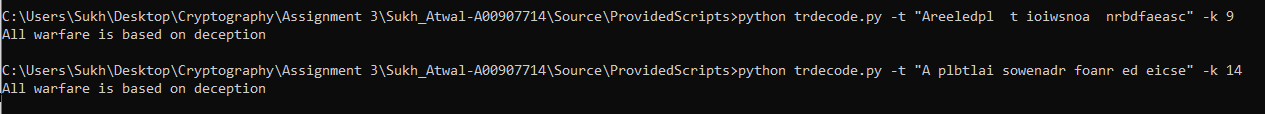
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test #** | **Tools Used** | **Expectation** | **Actuality** | **Pass or Fail** |
| 1 | IDLE, CMD | The text should be able to be encrypted and return the same text when decrypted. | The encrypted text returned the same text when decrypted | Pass |
| 2 | IDLE | The ‘input’ option should allow me to input encrypted text and should output the correct decrypted output back. | When we input the cipher, it was successfully decrypted with the same keys. | Pass |
| 3 | IDLE | The ‘file option should allow me to enter a file with encrypted text in it and should output the correct decrypted output back. | When we had a file with the cipher, it was successfully decrypted with the same keys. | Pass |
| 4 | IDLE | The script should not work if the user does not enter ‘file’ or ‘input’ and output a message saying to enter the correct inputs | The script asked for the correct inputs when the user input ones that weren’t listed |  |

## Test 1

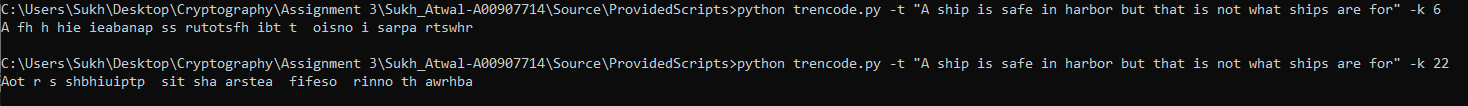
Throughout the rest of the tests, I’ll be using the following two texts for breaking the encryption. Here I’ll show the encryption and decryption process to ensure the accuracy of it.

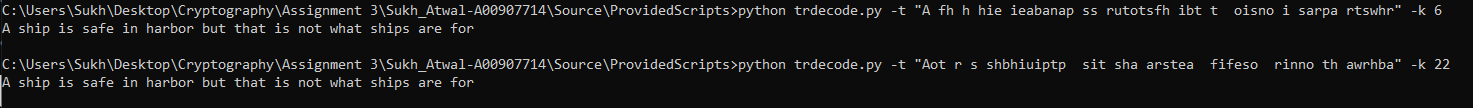
I’ll be using two text statements to ensure accuracy. The first will be “All warfare is based on deception” with a key size of 9 and 14.

For the first text statement “All warfare is based on deception”, it outputs the following two cipher text: 

Now if I run the decryption script, using the ciphertext corresponding with the key should output the text statement, “All warfare is based on deception”. 

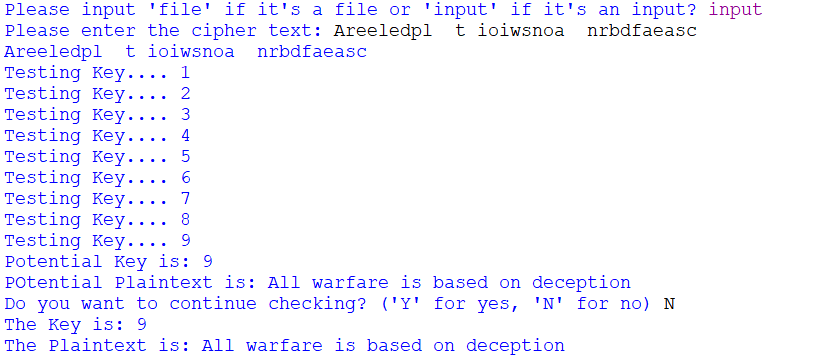
I’ll repeat what I did above but for the second text statement “A ship is safe in harbor but that is not what ships are for” with a key size of 6 and 22.



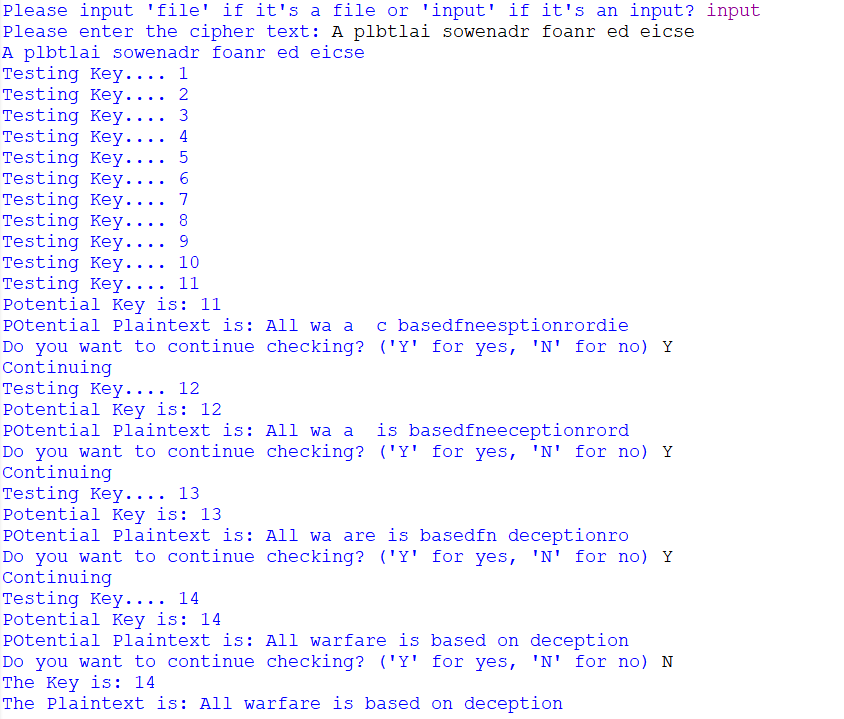
And if I decrypt it, it should output the same text. 

## Test 2

I’ll start the script to break the cipher text. Here I’ll be using the ‘input’ option to manually input the 4 encrypted texts.

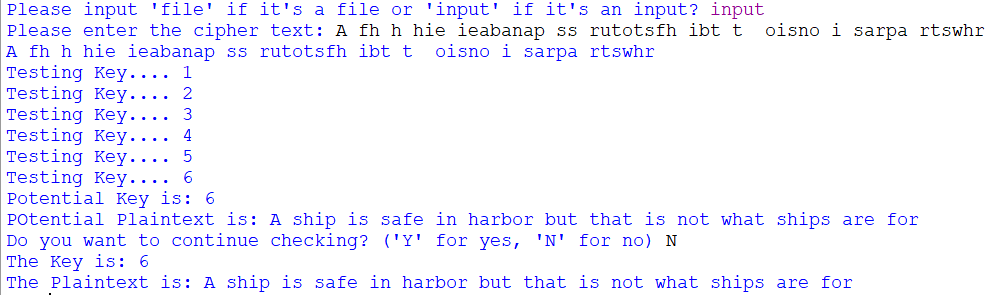


Here is the results of the first cipher text which we created with a key size of 9. Seeing as the plaintext was correct, we didn’t continue checking.



Here, we can see that the plaintext was random letters and wasn’t outputting any sensible words, so we continued checking. Eventually, we got the correct output with a key of 14 which matches.

Now we’ll move to the second statement.



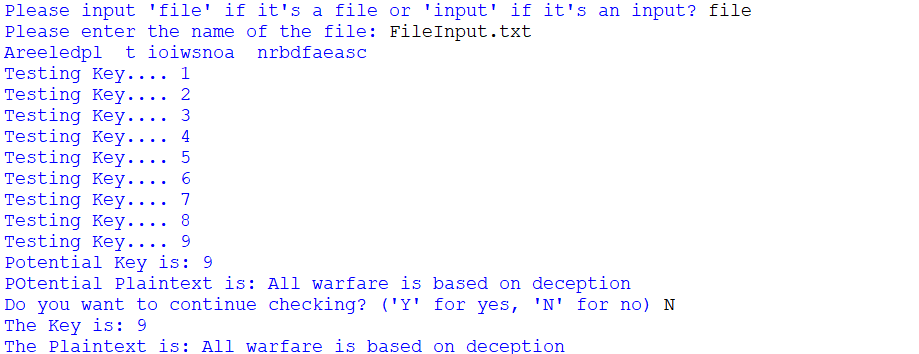
The plaintext is correct, and it got it with a key of 6 which matches what we did when creating it.



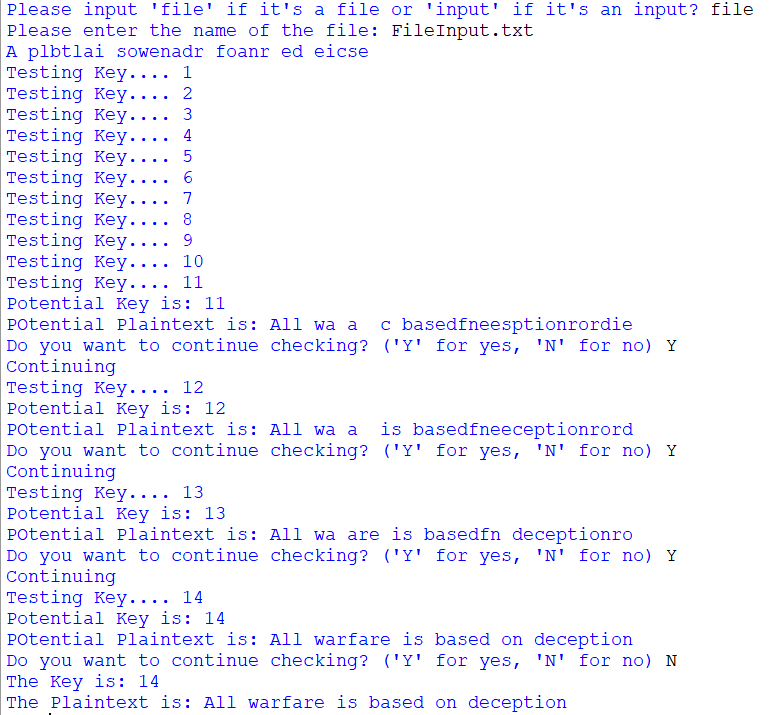
Here, we see a similar thing that happened before where the plaintext doesn’t initially match what we put in however it’s also still random letters and doesn’t create any legible sentences until key 22.

## Test 3

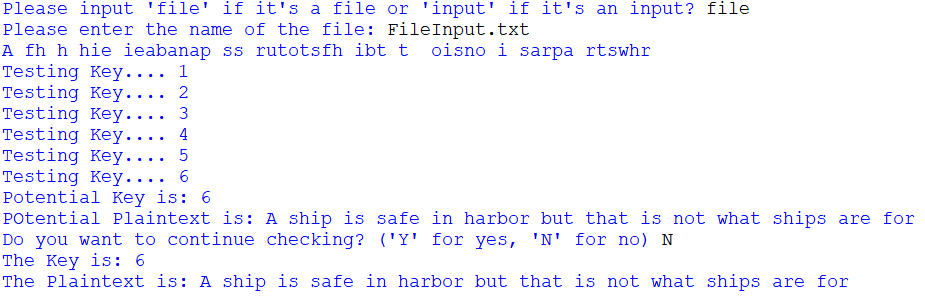
Here I’ll be doing the same idea as above however I’ll be using the ‘file’ option which allows you to break a transposition cipher that’s in the file. I’ll be using a file called FileInput.txt in this scenario.



We can see that it matches the rest of the tests, where it correctly gets the plaintext after 9 attempts.



Similarly, we can see it some of the initial plaintexts are just random letters and words strung together up until Key 14, where it outputs the correct plaintext.



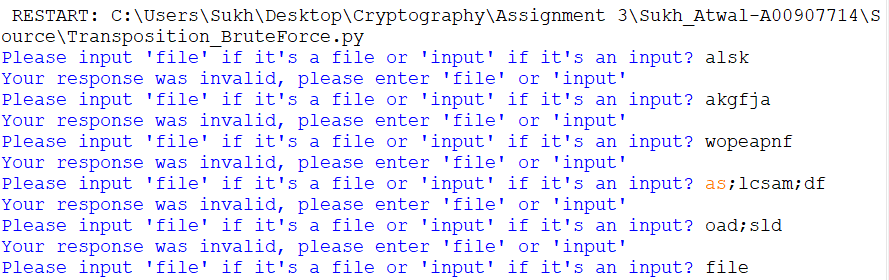
For the second text statement we encrypted, the results are as expected with the correct plaintext for Key 6.



Once again, random letters and words initially for the plaintext but eventually gives the correct plaintext with a Key of 22.

## Test 4

The script should repeatedly ask the user to correctly enter the commands, either ‘input’ if it’s a user input or ‘file’ if it’s a file.



As you can see, if the user enters random gibberish, the script asks if the user can input the correct commands. This continues until the user enters them.