Report

Overall, I believe my code works quite well, however it is not perfect. For example, the constructor method of PermutationKey is not as efficient as it could be. The for-loops in each of the two key cases might be able to be combined into one for-loop to use up less lines of code.

In certain methods in both classes such as “encryption”, converting from ASCII values to alphabetical values is not strictly necessary and I could have left them as ASCII values for a more efficient program. This is especially prevalent as in all cases where this happens the program converts back to ASCII anyway after the operation in the method. Additionally, some algorithms use a lot of variables such as in the encryption method, some of which are likely unnecessary. These could be removed to reduce complexity by having one variable fulfil more than one role.

In the attack method of the Attack class, I could have used the invertion and composition methods from the PermutationKey class to calculate the new key instead of direct computation. This would have used less lines of code and increased efficiency by use algorithms already available to me instead of creating a new one. In the display method of Attack, the section that displays up to 300 words of the decrypted cipher text could be made more efficient by handling the space of line breaks more efficiently, and by looking at different cases of cipher text length together instead of separately. This makes this algorithm more complicated than it needs to be as it looks at both a case where the message is small, and where it is large which requires displaying a remainder part.

On the other hand, all methods work as intended in both classes, and hence are very functional. Although it has a lot of variables, the attack method in the Attack class works exactly as indeed and helped me reach the correct solution for deciphering “secret”. My code handles the swap and undo functions of the Attack class very well. Each of them is short and concise and use previous methods to decrease complexity. They also work perfectly well and even handle incorrect inputs thanks to the error check in the swap method of PermutationKey.

The lettercount method is relatively simple and perfectly functional, and I did not have to resort to built in find functions to complete it.

If I had more time, I would add a method that makes a list of common words in the cipher text, as well as recording their frequencies and a list of words that they could be the encrypted version of. This would help with deciphering words such as “the” or “is” giving a huge head start in the decryption key.

I would also implement an alternative to the Attack method which, given the letter count in the cipher text, would output a selection of likely keys, paying particular attention to single character words which will likely be an I or A when decrypted. This would also have a feature like the attack method that guesses a key based on letter counts but with the ability to “lock” certain letters in place so that the function ignores them when calculating a likely key.

To summarise, my code is not as simple and efficient as it could be but it is very functional, each algorithm does exactly what it is required to do.