Cryptography Final - Terry Wade

I have neither given or received, nor have I tolerated other’s use of unauthorized aid.

For my final project, I decided to go back on the Random Encoding assignment and write a more efficient algorithm to encode messages in this fashion. My first iteration of this was written just to speed up the process of random encoding, but this time I wrote a program with a method to encode. I chose this because I think that random encoding is one of the best of the simple encryption algorithms for effectively hiding a message. It also seemed a bit more complex than a regular substitution algorithm programming-wise, since you have to account for the randomness that comes with it. In my experience, it was not very hard to write the script and make it work, so it was more about expanding on this idea covered in Assignment 5.

My first go at this program used multiple nested for-loops, which is simply bad practice. I read in the text as a whole, and then had to loop through each paragraph, then nest a loop that goes through each sentence, and so on. I avoided this by using more of Python’s built in string manipulation methods to convert the read data to a string with no spaces. This allows me to convert only the text and split the message up at the end. The other most important thing I needed to work around was having the key in plaintext in the script. I decided to make the key dynamically generated while the encoding happens. This made my program a lot more secure, since the actual key won’t just be sitting in the source code for anybody to see. Other changes included moving my code into a method so it can be reused later, and returning both the key and the cipher text for the person encoding the message. This may pose a security risk if it were not for the fact that the key is dynamically generated with the program’s execution.

I chose Python to do this simply because it has a very rich array of features that make it easy to quickly do what needed to be done. I was easily able to read in file input and manipulate that data thanks to its built in functionality. For something as lightweight as this algorithm, it was really necessary to use a more intense language like Java or C. I feel like performance would have definitely suffered if I used one like that since there’s a lot more that happens at compile time. Python also allowed me to run the program very quickly on my command line. There is not any compiling necessary for a Python program except when it is ran, so that made it a lot quicker to test out.

Something I could probably improve on when I look back on this assignment is the dictionary of characters. I think this also poses a security risk since people can probably use the key-value pairs to get the key of a message. In a future iteration, I may like to use the ASCII values of each character to encode like in Assignment 9 and Assignment 10, but for the scope of this question I didn’t feel like it was necessary. I’ve read online that lowercase numbers can also be used strategically to make the ciphertext have more variety and therefore more security, but by design I decided to not include them. Using only uppercase letters gives each individual letter the ability to be converted to any other letter, whereas including both cases would not. For example, if I only use uppercase, A can be converted into any letter. But if I include lowercase and continue to mod by 26, then A can’t be converted to b for example, but C can. This would make it easier to decode a cipher by hand simply because you would know which letters cannot be encoded by each letter in the ciphertext.

I think I learned a lot from this assignment about the security risks and tradeoffs you take when designing an algorithm. I had not ever thought about these things when writing encryption algorithms throughout the semester, but this has really emphasized the importance of them. This assignment also always reminded me to use good code practice and not just to do what needs to be done for the assignment. Random encoding is something I’m particularly interested in, and I’m going to use this as a basis to create some more random encoding algorithms in the future.

Here are some screenshots of my program:





