KRYPTO FUN

MAKENZIE MANNA – MSCS-630

Abstract

This paper seeks to explain the process of creating a software application that uses encryption. Specifically, it will utilize the popular Caesar Cipher. Not many people realize the multitude of ways in which you can encrypt messages, images, etc. The application should be simple enough for the average person to utilize and be able to decipher what exactly is going on in the back end of the application. In other words, the user should be able to figure out why their message was encrypted to the cipher and how the shift key is applied. It seeks to answer the questions: Do people know what a Caesar cipher is? Do they know what it does or why it is used? The goal is to introduce basic cryptography to the public and demonstrate how cryptography can be both challenging and fun. The application for which this paper is based on was developed in Android Studio using Java language.

With this project, my goal was to familiarize the “average Jane/Joe” with one of the most basic cryptography concepts. The application allows the user to input any “secret message” they would like. They are then prompted to also enter a shift key number. This aspect of allowing the user to select their own shift key was important in that it is what allowed the user to get an idea of what the backend of the program was doing. It did this by providing a starting point for the user to figure out how this key was affecting the text. Another option available is to enter an already encrypted message, the key used to encrypt the message and it would output the plaintext.

Beginning this project, I was unsure of where to begin. Having gotten used to using Visual Studio Code to develop the ciphers throughout the semester, I did not know which software to use to begin developing. I could not use VSC because there was no “easy” way to incorporate a user interface or even build one for that matter. I have some previous experience developing with Android Studio and although I have an iPhone, I have an Android powered tablet. I struggled with the decision to use AS because it ran extremely slow on my laptop. However, with some upgrades to my hard drive I was able to get it running smoothly AND was able to get the emulator working. The emulator in the beginning was not displaying correctly, at all. I would choose the most recent Android smartphone and it would show me an extremely old version of some sort of Android smartphone where the text was too big for the screen. To solve this issue, I had to update my ADK.

Another issue I encountered was with my lack of programming experience. Because of this, I do not know as many languages as I would like. This limited the developer software I was able to use. Specifically, because I did not want to download a bunch of different software to

"try out" and because I was unable to figure out how to create a java file in Visual Studio. Visual Studio also posed the issue of not having a straight forward design aspect, I would have had to download different plugins. Android Studio provides a great platform for designing a UI, especially for something with little to no experience developing XML files. Because of this, developing the UI was less complicated. I had to rearrange the design elements a few times after deploying the application onto both a tablet and a Google Pixel smartphone. The design is simple, but self-explanatory and visually appealing.

The backend aspect of the application is where I had the most to learn and get creative with. Originally, I had the idea of displaying dynamic output every time the user changed the shift key. I decided against this method because I really wanted it to be very straight forward for the user. Instead, I gave them the freedom to enter their own key number. There was a learning curve for me in understanding how to read input from the user in Android Studio. Normally, I would have used System.in and System.out. However, AS does not work that way. They instead use a getText() function. You declare a variable that you want to reference the input and set that equal to whatever you named the input box when declared earlier in the code then the .getText() function. This then performs the same method as the System.in and .nextLine() methods all in one line of code. The same concept applied to getting the input from the shift key input field and the cipher text input field. To display the output was also something I needed to learn. This was simpler than I had originally thought it would be. All that needed to be done was a .setText() method appended to the end of whatever variable name you declared the output field. Inside that set text method is where you call whatever you

want displayed. In this case, I wanted the plaintext (p_text) to display when running the decryption method and the ciphertext (c_text) to display when running the encryption method.

Testing and debugging the code was again, another aspect of AS I didn't really understand how to do. I wasn't sure how to read the errors the log was giving me and the documentation for AS was not helping me either. Instead of wasting too much time trying to decipher what the error messages were telling me, I decided to try my own methods. Initially I had the issue of the application not visually doing anything when the user clicked the buttons. This didn't tell me much about what the issue was. So, I started trying to print out things earlier in the code. I started with printing the user plaintext input as is, when that worked I tried printing the plaintext input and the shift key input. When I got those to work, I again tried to print after the for statement. When that still didn't work, I came to realize it must be a bug within that for loop. This helped me pinpoint the issue and try different ways to fix it. Another issue was that the program started crashing when the user clicked the decrypt button. At this point, I had the encryption working just fine. So why was it crashing? I reiterated the debugging methods I used in the encryption method and realized it was crashing because the parameters in my if statement did not make sense.

For testing I started with using the emulator. Once I was able to get the emulator up and running, I began using some simple test cases. For example, my first test case was to encrypt hi with a shift key of 1. The output for this would obviously be ij. I then decrypted ij with the same shift. I came to realize that I did not set boundaries as if the program should run in uppercase or lower or allow both. I decided that it would be run, in lowercase, so I proceeded to add the modification. Then I started testing it on other devices. I tested it on a

Google Pixel and that is when I realized the display was off. I edited the display and re-ran the application. The final device I tested the application on was an Android Fire tablet. Once I knew it ran well on all three platforms, I began exploring more creatively with test cases.

The next test was the real test, was it going to achieve the goal of educating people about the Caesar Cipher? I started out with someone who was a little more 'computer savvy', they are 24 and work in the technology field. They were able to use the application with no instruction or question. They were able to understand the logic within about twenty minutes. They did not know that it was called a Caesar Cipher. Next, the user was a 23-year-old whose occupation was in the social services field. They were also able to use the application without help. They, however, took forty-five minutes and then gave up on the logic. Once explained, they understood. There should have been a larger test pool to more accurately estimate the effectiveness of the logic.

Arguably the most important part of developing any kind of software, especially cryptographic software, is debugging. Debugging allowed me to see quite a few issues I had not thought of while developing. The first thing I noticed was that I did not have any code to tell the software what to do if encrypting and text went past z. Therefore, when that did happen, it was encrypting those characters into symbols like: ; ' " , etc. After adjusting the ASCII values to wrap back around to the beginning of the alphabet when that happens, I needed to think about how that would work with decryption. I did some research because I wasn't sure how I needed to adjust the ASCII values in this direction. This direction being, if the decryption character was less than a. After not really finding anything that directed me towards the answer, I began to tap into my mathematics background and figure it out myself. I ended up realizing I could use

the same formula that I did for the encryption method except replace the ASCII value of ‘a’ with the ASCII value of ‘z’. I tried a few more test cases and realized, I had it!

Although the Caesar Cipher is not a particularly ‘safe’ method of encryption with the level of intricacy and intelligence that hackers have today, it is a great place to begin. For me it was especially nice to start with a program like this because it wasn’t too intricate for my programming background. I learned a lot about how Android Studio works as well as how the Caesar Cipher works, since we only briefly covered it in class. My peers that tested the application also learned about the cipher, which was a goal of mine. I could improve this project by allowing capitalization of letters, expanding the values of which can be encrypted. I could have also had a larger testing pool for the educational aspect, as stated earlier. Overall, I am pleased with the work I have put forth and am excited about the fact that I was able to complete it independently.