Jason Wilson

Professor Arias

Friday, November 16, 2018

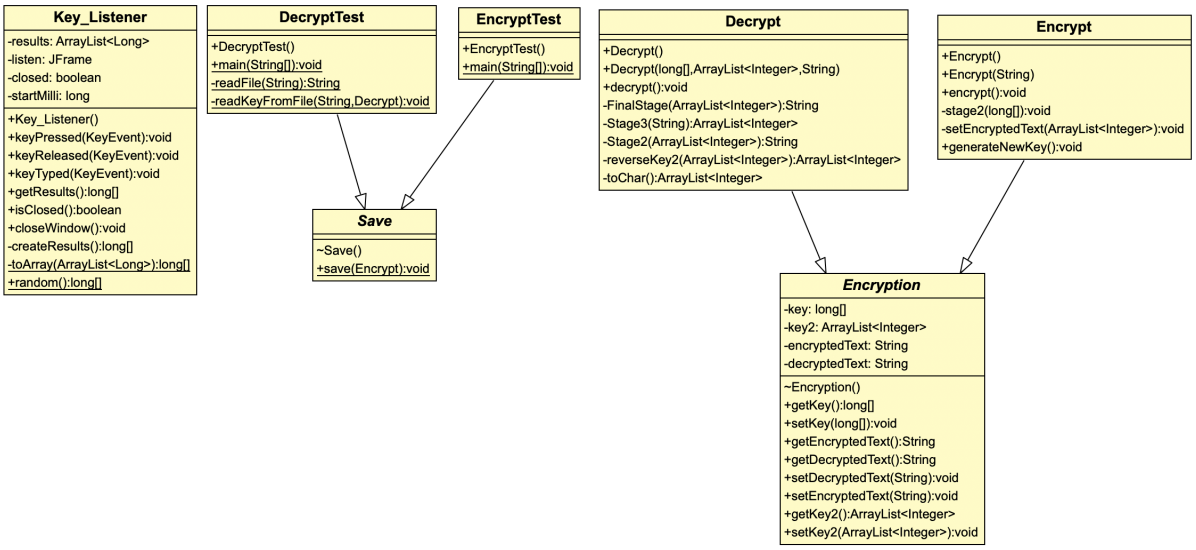
Abstract:

Creating a symmetric key encryption algorithm that uses truly randomly generated numbers to scramble the data. The program is designed to take an input of text, generate truly random numbers, encrypt, and output unintelligible characters and an key file to a data folder.

The motivation behind creating this encryption program was primarily to gain a better understanding of encryption and how it works. Homemade encryption programs shouldn't be used over any of the current encryption standards when encryption important data. It was found that random number generators are not completely secure. If the generator is an simply algorithm, that "randomness" can be predicted and reversed. This is a problem when writing an encryption program that relies on randomly generated keys. So, this projects overall goal is to make a custom encryption program that generates truly random keys based on user input for a harder to break encryption. This paper will go over how the system works, how users will currently interact with it, and how the classes interact with each other through UML diagrams.

The encryption program first accepts user input into the console, it then comes up with a window with a key listener active where it records random user keyboard input based on the millisecond the key was pushed, released, and what key was push and released. The program then turns all of the characters in to their integer values and encrypts them based on ((randomNumber1 * char value) + randomNumber2). For each character in the string to be encrypted, a different set of random number from the generated random numbers is used. This means the more time a user spends creating a key based on the length of the text to be encrypted, the more secure it will be. Then those encrypted values are split up even further by taking a random number from the Math library and adding by a value between 32 and 230, and

then added 48 onto it in order to get the encrypted text to a printable string. Once that is finished the encryption is complete and the encrypted text and two keys are saved into their own text files. The encrypted text is completely unintelligible and can only be read again by running the decryption program. The decryption program imports the encrypted text and two keys and reverses the encryption program until it produces the message that was encrypted. Here are the UML diagrams describing how the classes interact with each other, currently:



Problems that this system is addressing is the everlasting effort to secure the worlds data, or at least gain a better understanding of it and to create truly random numbers based on user input and use those numbers to encrypt data. Requirements that this system has are to completely encrypt that data that is given to it and be able to successfully decrypt that data.

As far as other works that people have done to address a similar problem. There are all of the previous encryption standards that are implemented heavily by the government and companies across the world in order to secure their data from data breaches. There are also a lot of small projects where people simply wish to encrypt data in a non serious way, but these encryptions, including mine, are only for a learning experience and should not be trusted unless heavily tested.

To use the system, the first thing a user does is fire up the EncryptionTest class. This will prompt the user for a text input and wait to proceed. Once the text is input, a window will pop up where the user must type randomly into, changing the letters, duration between key press and releases, and hit enter once finished. Once this is complete, the encryption will have been executed and saved to the data folder. To decrypt, the user must start the DecryptionTest class, where it will decrypt and output the decrypted text, no steps required by the user to complete the decryption.

The system successfully encrypts data and decrypts data using truly random numbers and accomplishes most early goals. What still needs to be done is fix a bug where the decryption program fails upon reading a character and cannot complete the decryption. The system also needs an interface to make using it as easy as possible allowing for specified file destinations and other features. The save class will also be rearranged so it can handle the saving as well as importing encrypted/decrypted data. Also, many small big fixes and review of the encryption and decryption algorithms are required which will be a lengthy task. There is no shortage of work that needs to be completed for this system.

The references used so far (links only):

https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf
https://www.howtogeek.com/183051/htg-explains-how-computers-generate-random-numbers/
https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide