Nicolas Mays

Juan Arias

Software Development 1

May 8th, 2017

Folsom

Abstract:

Folsom is a file encryption application that allows for the encryption of the content in text files. Folsom utilizes my own version of a character by character XOR encryption which is paired with text transformation to create what I like to call dynamic encryption. Dynamic encryption uses the binary values of characters to turn plain text into sets of numbers that are extremely difficult to crack. The difficulty in cracking lies in the fact that dynamic encryption layers two different transformation and encryption types on top of each other. This creates a line of cypher numbers that only mimic binary on the front half of each number. The front half of the cypher number can usually be expected to be ones and zeroes while the back half is just a slew of numbers. This implies that the cypher is at first glance a large number, extensive cryptanalysis would have to be done to move past the point that the cypher numbers represent no decimal representation of the encrypted message, and then the XOR isn't even cracked where the key can be the entire length of a message. One full cypher could be used to encrypt an entire message.

Introduction:

Cyber security and the romanticism of "hacking" is something that most computer science students and computer scientists have interest in. The added threat of cyber-terrorism and hackers looking to maliciously exploit system data also adds the strong emphasis on data protection that is drilled in to the heads of software developers and data architects. One aspect of cyber security is encryption of files, or protecting the contents of the file from anyone who isn't allowed access. Folsom allows select information passed along to be hidden from anyone without the key, and utilizing a symmetrical cypher the key can be used for both encryption or decryption of the file. Without both the key and the message contained in a file, the only possible way to decrypt the cypher would be by an extensive cryptanalysis which is a timely and difficult project to execute that could even lead to "lossy" results. "Lossy" results being that the decrypted message still has holes in it, whether it be misplaced symbols or line breaks.

System Description:

This application works on a graphic user interface. The user interface includes two text fields respectfully labeled "filename (.txt):" and "Key:". Also included are three buttons labeled "Encrypt", "Decrypt" and "Generate Key". The user may simply have a created text file, encoded in UTF-8 or ASCII placed in the respective folder of the application workspace. The user then enters the desired to be encrypted file name with the .txt extension. Then the user can enter in a key. Then by pressing either the buttons encrypt or decrypt text button the program will encrypt or decrypt the text. The first process will of course have to be an encryption.

Folsom utilizes 3 classes, the main class Engine, the encryption class TextEncryptor and the IO class FileReadWrite. When an encryption is initiated by pressing Encrypt, Engine's event listener creates an object of the FileReadWrite class. Next a TextEncryptor object is instantiated with the String parameters message, key, and the boolean encrypted. Message is created by invoking readText(filename: String) which will read the entire text file as one large string. The key is derived from the string typed in the text field and the boolean would be set to false. Next

Engine invokes dynamicEncrypt which will do a parameter check to make sure things like the key is not a greater length than the message and the message isn't empty. If all clear the the encryption process parses the message into a byte array with the String method getBytes(). An integer array is created of the same length. Each value of bytes is turned into a binary number and then subject to the XOR operation against a specific index of the key as the bytes array is cycled through. The new value is stored in the respective part of the integer array. After encryption is completed the integers are then written to a String and that cypher overwrites the message field of the TextEncryptor object. Engine then will invoke writeCypher() of the FileReadWrite class which splits the message String at the spaces and overwrites the file with the tokens of the cypher line by line. If a successful encryption takes place a dialog box will appear with a success message and the time duration of the encryption. Lastly, a security protocol called terminate is invoked upon the TextEncryptor object which will wipe its data fields and utilize garbage cleanup.

The decryption is triggered by once again keying in a final name that's in the project directory, putting in the appropriate key and pressing the Decrypt button. The filename is read from the text field, the key is also pulled from the appropriate text field and used to create an instance of TextEncryptor. The message parameter is filled by FileReadWrite invoking readCypher on the encrypted text file which builds it as a single String. Then dynamicDecrypt is invoked which does a parameter check. If the check passes the program moves onto decrypting the messages. The message is parsed and the String is transformed to numbers and stored in array. The values have an XOR decryption operation done and lastly converted from a binary number to a string representation of the integer by parsing the integer with a radix of 2. This turns it back to decimal and then is casted as a character for readability. FileReadWrite will then write it back to a text file. After decryption, a dialog box will alert the user if it was successful or not. In the event, it is successful the time elapsed will be printed.

The last important action of the application is the generate key function. By pressing the “Generate Key” button, a key is made by a TextEncryptor object creating an appropriate sized key of a random concatenation of the characters ‘A’-‘Z’. ‘a’-‘z’, and ‘0’-‘9’. The key is returned as a string and is printed to the dialog box. The user can even copy it to their clipboard. Since a generated key only makes sense for Encryption, it would just throw an error since dynamic encryption is symmetrical. As soon as generate key is pressed, the decryption option is disabled. The user however can press the button and generate the key as many times till they find one appropriate.

Requirements

The application reads and encrypts what could be in a text-file as a single string. Meaning the max length of message the application can handle is a message containing 2,147,483,647 individual characters. A single-spaced page with 12 pt font contains about 2,300 characters, so put into context the application could in theory encrypt over 900,000 pages worth of text. A java character at large is 2 bytes or $2x10^{-9}$ Gigabytes. So, if the max length of a string is 2,147,483,647 characters then the extent of the encryption can be stretched to about 4.3 Gigabytes of encryptable and decryptable text. Since large file sizes can be encrypted then the system is able to push mass amounts of sensitive data. Also, operating time is also important and the system is able to produce an encryption of 200,000 characters in 0.003 seconds. In theory encrypting a max file would only take around 32 seconds with proper conservation techniques. That paired with the data hiding techniques the application should be safe to use and then throw away the files. The GUI form that runs the user operations also works to reduce error and increases security. The programming of the form is optimized and gives a clean simple model without much complexity so a user would have little issues using a complicated process like encryption. Also by locking out users after certain processes there is less of a chance that a user could doubly encrypt/decrypt a file and lose the entire point of encryption or decryption in a first place. Also since the TextEncryptor class has very simple parameters it has the potential to be built upon, like increasing the available file size by using multiple threads of String messages. Lastly the generate key button generates keys up to 10 characters. By using that feature the user would be increasing the risk of their information being decrypted by a non-intended source. By using the character sets and the algorithm developed with the normal case scenario of a key with a length of 10 there is ${}_{62}C_{10}$ combinations, or 107518933731 possible keys to be tested.

Survey:

Encrypting contents of text has been practiced by people since ancient Greece with very remedial methods like replacing letters of the alphabet with numbers or having secret symbols. In modern times computers have allowed linear-algebraic cyphers like Hill Cypher to be used but with limited success as matrix singularities make keys unusable and has low efficiency. With basic XOR encryption a passage as long as a bible verse can be decrypted in a matter of seconds using a basic key. A Caesar Cipher being another basic method can almost always be decrypted with cryptanalysis of the frequency of certain letter combinations. However, the dynamic encryption with binary values allow for a much harder cypher, and the cycling of the XOR values in the key create frequency analysis almost unusable. Dynamic Encryption solves the problem of both efficiency, key usability and difficulty of cypher breaking.

User Manual:

After the user creates a text file containing their information, they should place it in the project's folder and run the application. The user should enter in the appropriate filename with the .txt extension at the end. Next the user should enter the key, the key should be less than the message length or the encryption engine will throw an error. For encryption, the user has the option to generate a key. A generated key must be written down as Folsom will not regenerate the same key for decryption. After pressing the encryption button if a dialog box appears that reads "Encryption Sucessful" the user can feel free to pick up their encrypted file. Note that the user will be locked out of encryption after a successful encryption so the program must be run again to encrypt a second file.

To decrypt, follow similar steps but be sure to enter the correct key in correctly. Keys are case sensitive and non-correct keys will throw an error and wipe the file as a security measure.

Conclusion:

In conclusion, Folsom as an application is well rounded and simple allowing for secure transformation of data. The modules of FileReadWrite, Text Encryptor and the GUI form work together to create a viable security application that can be used as a tool for people interested in file security. The large file capability and efficiency factors play a huge roll in its success as an encryption system that has the ability to be improved upon. The Encryption engine being unmatched by other basic encryption techniques makes the application a potential normal use for the protection of data with little threat of it being compromised without human error.

References:

Konheim, Alan G. *Cryptography: a primer*. New York, NY: Wiley, 1982.