Nicolas Mays
Juan Arias
Software Development 1
April 3, 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 I like to call dynamic encryption, which uses the binary values of characters to turn plain text into sets of numbers that are hard to crack. The difficulty in cracking lies the front half of the cypher, the encrypted message, usually contains ones and zeroes and the back half is just a slew of numbers. This implies that the cryptanalysis will have a hard time knowing what exactly to look for when cracking.

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.

System Description:

The application allows the user to enter the message, at the moment, a hard-keyed string and a hard-keyed string. A longer key of unique characters is the best potential way to evade crackable messages, but the key cannot exceed the length of the message in dynamic encryption. The driver method main() then employs the encrypt function. The first step of encrypt is parsing the String into a byte array with the String method getBytes(). An integer array is created of the same length, then the key is transformed using the local method getNewKey() which returns the key as a positive summation of the values of each individual character. Then the encryption cycles through each value of bytes and turns it into a binary number. The binary number is then subject to the XOR operation against the key. The new value is stored in the respective part of the integer array and returned as an integer array. The driver then prints the integer array and it can be shown that the numbers resemble binary for the front half and random numbers on the back half of each token. The driver also prints the time it took to encrypt. To decrypt, the driver employs decrypt(), taking in an integer array and key as parameters, once again getNewKey() returns the integer summation of the key’s values. The decryptor uses the XOR operation to decrypt the values in the integer array back to binary. Then the binary integer is converted into String and then parsed with a radix of 2 to bring it back to a base-10 value. After being casted as a character it is stored in a string as plain text and returned. The driver then prints the decrypted message with a completion time. The message should be lossless, meaning no text or line break is missing from the final string.

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.

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 and with basic XOR encryption a passage as long as a bible verse can be decrypted in a matter of seconds using a basic key. However, the dynamic encryption with binary values allow for a much harder cypher.

Conclusion:

In conclusion, Folsom as an application is well rounded and will become even more so with the implementation of the File Read/Write module and potentially being able to encrypt .png images alongside text. The large file size capability and efficiency factors play a huge role in the performance of the application and makes it something to potentially use to protect data with little threat of it becoming compromised.

Works Cited

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