Emily Gorelik

Software Development Writeup

Professor Arias

5/4/18

LFSR Based Image Decryption and Encryption

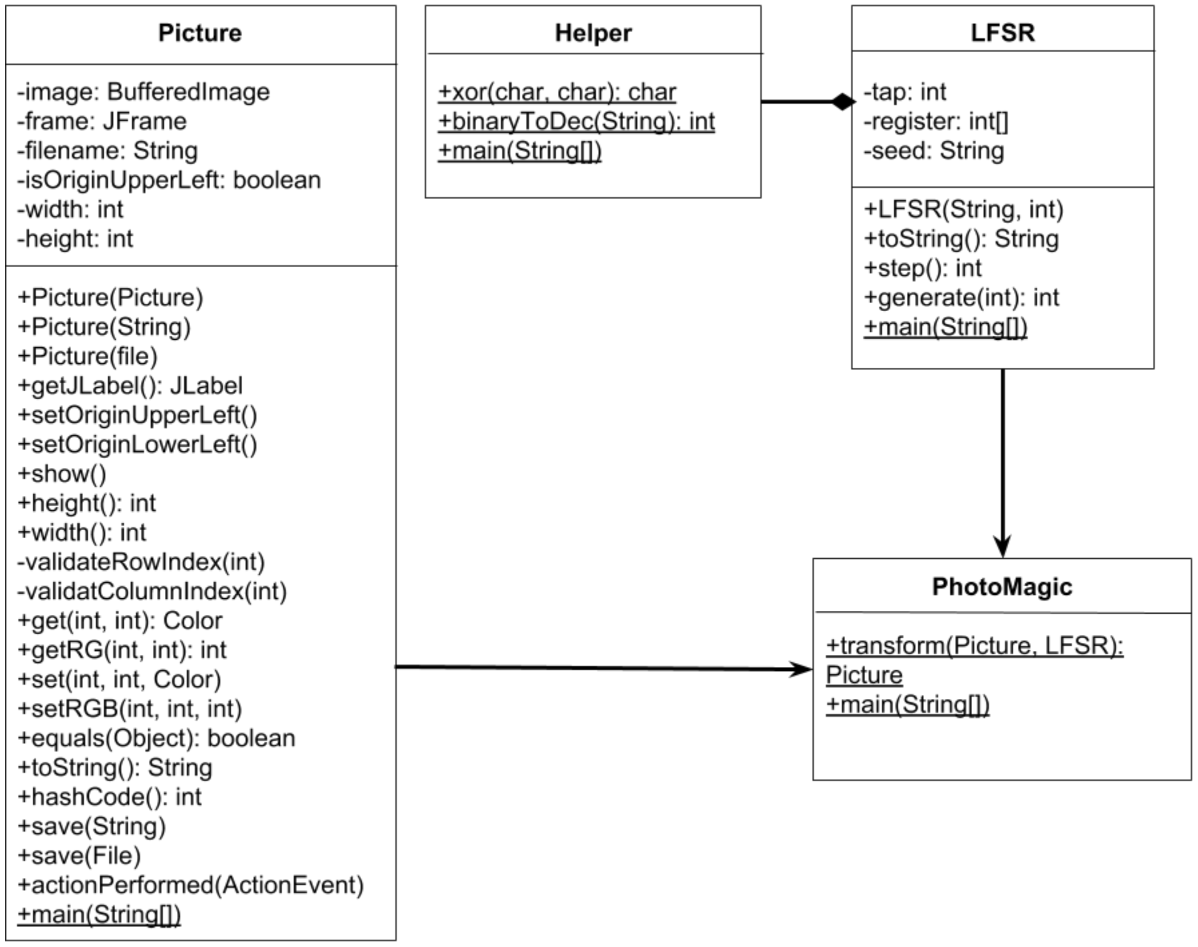
Abstract

This program is an image encryptor and decryptor. Its primary function is to encrypt an image without modifying the original file. It uses a pseudo-random integer generated by a binary number that has gone through a linear feedback shift register a specified number of times to create a modified copy of the image. PhotoMagic does not modify the original image file, which is one of its benefits. Instead, it treats the image as a two-dimensional array and iterates through each pixel and changes the color value using the integer. PhotoMagic works best on small images, and should only be used on png files. The binary number along with the number of times the user wishes the LFSR to go through becomes the key to the encryption, and can be used to encrypt as well as decrypt the image.

Introduction

Though Java includes many ways to work with images, they can be clunky and difficult to use. PhotoMagic allows for an easy way to encrypt and decrypt a small png image by using a linear feedback shift register as a cipher to modify an image. This program provides a simple course of action if all one needs is to encrypt a small image they have. This paper will include a description of how the program works, why it is needed and useful, similar systems, and instructions on how the program should be used.

System Description



This system is composed of four classes: Helper, LFSR, Picture, and PhotoMagic. Picture.java can be found in the Java standard libraries, and allows the system to create the necessary Picture object that can then be modified and encrypted. The Helper class has three static methods: xor, binaryToDec, and a main method. The xor method performs the xor function on two chars that are either a “1” or a “0.” The binaryToDec method takes in a String representation of a binary number and returns the decimal form of that number. The main method simply tests the other two. LFSR uses xor to perform the linear feedback shift register on a user inputted String representation of a binary number (i.e. the “seed”). A linear feedback shift register generates a pseudo-random integer by shifting all of the bits one position to the left and generating the new rightmost bit as the previous leftmost bit xored with a bit at a given position called the tap. The reason why an LFSR is a good cipher to use to encrypt an image is that it would be difficult to guess the newly generated number without knowing the original seed and tap, which will be generated by the user.

The LFSR class is made up of a constructor, a toString method, a main method, and the step and generate methods. The former simulates one shift of the LFSR and returns the new rightmost bit. The generate method runs the step method *k* times, and returns the newly generated integer represented by the *k* rightmost bits in the register. This integer is used in the PhotoMagic class to change the color values of the image.

The PhotoMagic class has a main method and the transform method. The transform method takes a Picture object and treats it like a two-dimensional array. It iterates through each pixel and replaces its RGB value with the xor of that color component and a generated 8-bit number from an LFSR object that the user will provide. It then returns the new encrypted image. The main method implements the transform method using the user inputs, which it will get through the console and displays the image.

Requirements

This system allows a user to easily encrypt an image without altering the original image file. This removes the risk of accidentally altering the original file with no way to get it back. For example, if the user has forgotten the decryption password, but needs the original image for something. This system also provides security for the user as the image can only be decrypted through the use of the password i.e. the seed and tap number. The password can be as long as the user desires, and is easy and convenient to use.

Literature Survey

There have been many systems made to encrypt images. They come in many different forms with various levels of security and complexity. There are a wide variety of cyphers used, such as Vigenere Ciphers, RC4, RC5, RC6, etc (Kumari et al. 2). There are three main techniques used for image encryption, pixel permutation, pixel substitution, and visual transformation (Kumar et al. 19). The difference between pixel permutation and pixel substitution is that the former scrambles the pixels while the latter modifies the pixel value.

How one goes about encrypting an image isn't the only thing to think about, however. One must also consider the decryption process. This can be approached in one of two ways, symmetric and asymmetric key encryption (19). Symmetric key encryption uses the same key for encryption and decryption, while asymmetric uses separate keys. PhotoMagic uses symmetric key encryption as well as pixel substitution. A program with a high key space, or a large key, is more secure than one with a short one, so many image encryptors try to have relatively long keys

(20). In order for an image encryptor to be effective, the distribution of continuous pixels and density distribution.

The problem that any are still trying to combat with image encryption is speed (25). Many encryption techniques are slow due to the amount of space images take up in comparison to text. It is much faster to encrypt text but an image will take more time. Because of this, people are looking to find methods of image encryption that can also compress an image, but those methods are few at the moment. It is likely that the future of image decryption will include this kind of technology as more and more people work towards this goal.

User Manual

This system is meant to be accessed through the console. First, the user should open up the console and navigate to the directory where PhotoMagic.java is located. The user should also make sure that the image they wish to encrypt is in the same directory. Then, the user must input "java Photomagic [image file name] [binary seed] [tap integer]." One example of how this would look like is "java PhotoMagic pipe.png 01101000010100010000 3". The "seed" and "tap" integer are made up by the user, but the seed must be a binary number and the tap must be an index in that string. Once this is done, the encrypted image should display after several moments. The encryption and decryption process can be a bit lengthy and may take about a minute, depending on the size of the image and the size of the seed, so it is best to use this program to encrypt smaller images. Also, the closer the tap integer is to the beginning of the seed and the longer the seed, the more effective the program is. If the image is still visible or comprehensible it might require a smaller tap. Once the image displays the user can save it under a different name so the original image remains.

In order to decrypt the image, the user must perform the same steps as before, but use the encrypted image file name instead of the original. After a few moments, the decrypted image should open. It is important that the user use a png instead of a JPEG, as data can be lost while saving a JPEG and the image will not decrypt properly.

To test out the program, there is an image called pipe.png included with the rest of the file. If the user would like to test how different seeds and taps will affect an image, they can use that one, or one of their own of similar size (about 600 by 428 px).

Conclusion

All in all, this program seeks to provide a simple method of encrypting and decrypting a small image from one's computer. PhotoMagic treats the image as if it were a two-dimensional array. Through the use of a linear feedback shift register, the system goes through each pixel, extracts the red, green, and blue components, and XORs the pseudo-random integer generated by the LFSR to get a new value. This creates an image that can later be decrypted by using the same binary seed and tap integer, which serve as a password or key. Without both aspects of the password, the image cannot be decrypted by hand, making it relatively secure.

References

Kumari, Manju, et al. "A Survey of Image Encryption Algorithms." 13 Nov. 2017, pp 1-35, *SpringerLink*, doi:10.1007/s13319-017-0148-5.

Kumar, Mohit, et al. "A Review on Various Digital Image Encryption Techniques and Security Criteria." *International Journal of Computer Applications*, vol. 96, no. 13, 2014, pp. 19-26. Print.