Codes and Ciphers Project

Ben Cohen and Tim Rollet

May 11, 2014

Introduction

Both of us are very interested in image processing. Therefore, we figured it would be natural to try to apply our cryptographical knowledge to something related to images. One of the things that makes images different from other kinds of data is their visual nature. You can hold, manipulate and view images in a way that you can't do with text. Because of this, we wanted to explore visual cryptography.

Visual Cryptography is a way of encrypting images such that decryption does not need (but can be done with) a computer! A trivial example would be to rotate the red layer of an image 90 degrees, the blue layer 270 degrees and leave the green layer alone. Someone looking at this wouldn't be able to tell exactly what the original image is, but if they had papers in front of them with each of these layers they could spin them until they all lined up. Additionally, if they knew the encryption scheme they could simply turn each image 360 minus the number of degrees it was rotated.

Ideas and Implementation

For our project, we wanted a slightly more complicated approach. We decided to implement an n-shares splitting approach. The theory behind this is as follows. We want to split an image up into n distinct shares (or pictures) that are each the size of the original image, but do not appear to have any meaning on their own. However, when they're superimposed, the original image (with some noise) forms! The amount of noise that forms is proportional to the number of shares. Though the exact amount cannot be determined due to the randomness involved in the algorithm, we have found much more noise in images divided into more shares.

The way our algorithm works is as follows. To turn one image into two shares we do the following:
First we convert the image into black and white. From here we create two blank images with the same size as the first. We then iterate through every pixel of the first image and randomly set it to black or white.

We next go through evey pixel of the 2nd image. If our original image (after being converted to black and white) was white in this pixel, we set this pixel to the random value in our first matrix. Otherwise we set it to the inverse of the random value in our first matrix.

To superimpose images, we simply perform the OR operation on each pixel in each of our N shares. This approach, however, leads to lots of noise in images divided into a non-trivial amount of shares. Therefore, we also offer the option of taking a probabalistic approach to superimposing. Instead of performing a standard OR operation, we perform the operation accounting for "error". This means if we have a lot of shares and fewer than a threshold of them are black, we throw it our and set it to white. This could lead to an "incorrect" image where few pixels in the original image could be left out, but leads to much less noise in reconstuctions with a large number of shares. This can be enabled with the "-p" flag when dividing into more than two shares.

Also included are programs to run the Caesar and Affine ciphers on images. These show how images are particularly vulnerable to pattern recognition. While finding common bigrams in a long cipher is tedious and time consuming when working with text, you can often discern the subject of a high-contrast encrypted image immediately. However, the encryption strengthens if the contrast is low, and it can sometimes be difficult to recognize a pattern when the colors are strange. Overall, the algorithm described above is much stronger encryption for images, as the majority of each share is simply noise.

Included Files

A few things are included with this submission. First, there is this report. Second, we have our implementation, n-shares.py. This requires the python imaging library (PIL) be installed. PIL can be installed via pip, the python package manager. The package is named "Pillow". To see a usage message, simply run "python nshares.py". The usage will be explained. Also included in the "/tests" directory is a couple of subfolders, each containing a test image, it's shares, the the ourput. Each subfolder also contains a "test.txt" briefly explaining the test.

The file caesar.py runs the Caesar cipher. The file affine.py encrypts images with the Affine cipher, and the file deaffine.py decrypts them. Run these files without arguments to get a usage message. The Affine cipher can only be reversed if you use a key such that $gcd(k, 256) = 1$. A list of these keys is provided.

There is one known bug involving the Affine/Caesar programs. In most cases, they work fine. However, one particular image we tested is not able to be decrypted. This may be related to image format.

Future Work

In the future we hope to enhance our security by using a password in the encryption and decryption process. One way we thought about implementing this was to use a system similar to a vigenère cipher to encode rotation data. We could modulo each letter by 4 and rotate by 90 times that value degrees. That way if Eve got ahold of our share images there would be 4^n different orientations she would have to try to "decrypt" it, where n is the number of shares. For example, if we had 4 shares and the passcode "abcd", our first share would have no rotation, our 2nd would be rotated 90 degrees, etc. To decrypt with the passcode, we would just rotate 360-n*90 degrees.

Referenced Works

O. Kafri and E. Keren, "Encryption of pictures and shapes by random grids," Opt. Lett. 12, 377-379 (1987) http://www.opticsinfobase.org/ol/abstract.cfm?URI=ol-12-6-377

S. Patil and J. Rao, "Extended Visual Cryptography for Color Shares using Random Number Generators" International Journal of Advanced Research in Computer and Communication Engineering Vol. 1, Issue 6, August 2012