Engineering 133 Team Project – Image Processing in Python

ENGR 133 LC2 - Team 5

Pranesh Monda, Drew Behl, Ryan Griggs, Joe Poetz

Professor William Oakes

Purdue University

Project Motivation

To understand our project in more detail, we first started off by researching the purpose of image encryption as we wanted to know how this process is applied in the real world. When reading about this process, we found that “Image encryption has applications in corporate world, health care, military operations, and multimedia systems” (Oad 1, Yadav 1, Jain 1). This describes to us what kinds of workspaces engineers would be working in when doing image encryption. The next part about this process we wanted to learn more about was the application. Coming into this project, we understood that image encryption was used to decipher images but how does this affect the normal person? In the article, *A Review: Image Encryption Techniques and its Terminologies*, we discovered an area of study called Cryptography. After reading more on this area, we learned that Cryptography is the study of secure communications techniques, and that image encryption is a massive part of Cryptography. The article mentions “Cryptography provides a number of security goals to ensure the privacy of data, non-alteration of data and so on. Due to the great security advantages of cryptography it is widely used today” (Oad 3, Yadav 3, Jain 3). With this information, we know how engineers use image encryption and where they use it.

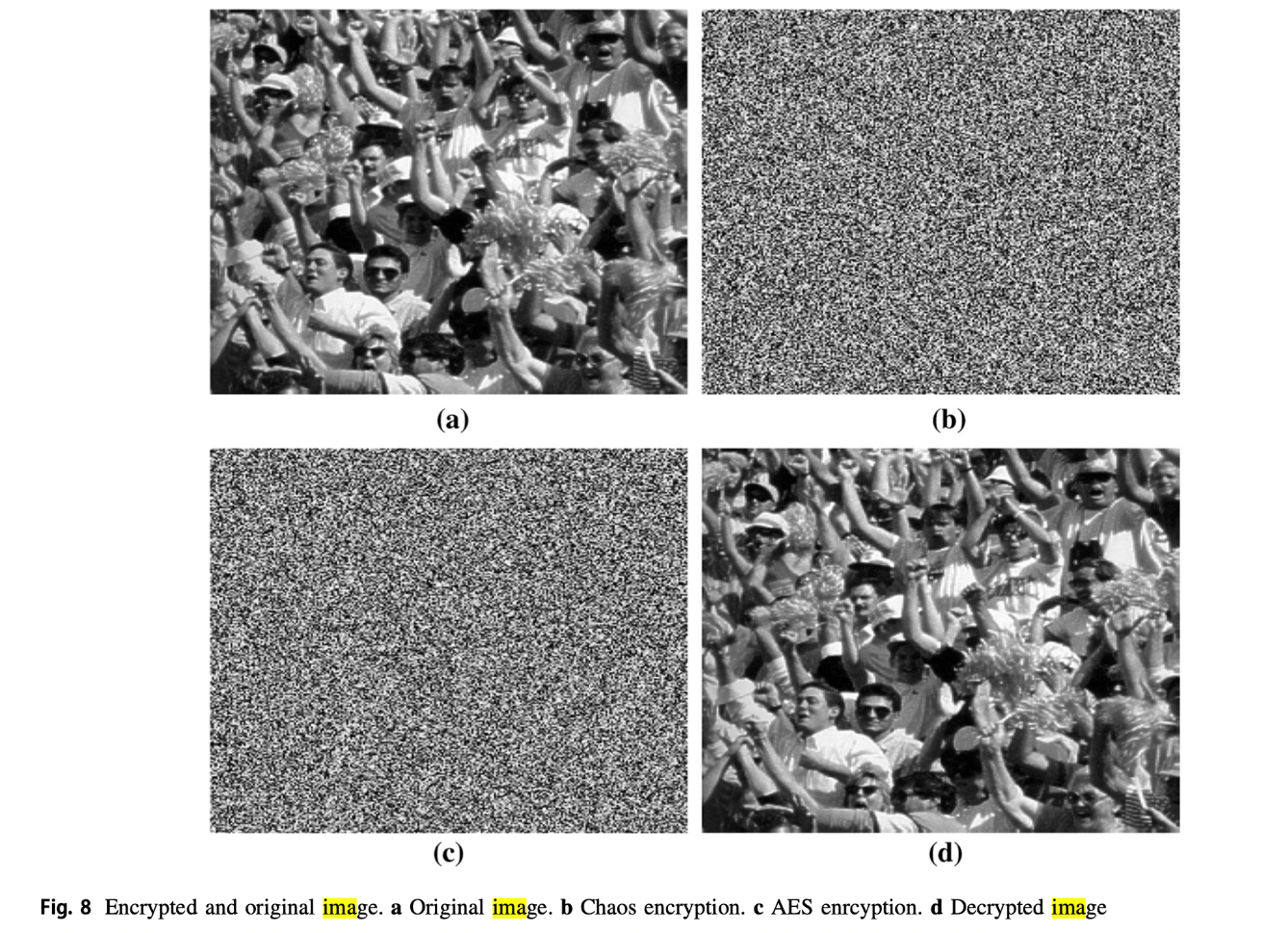
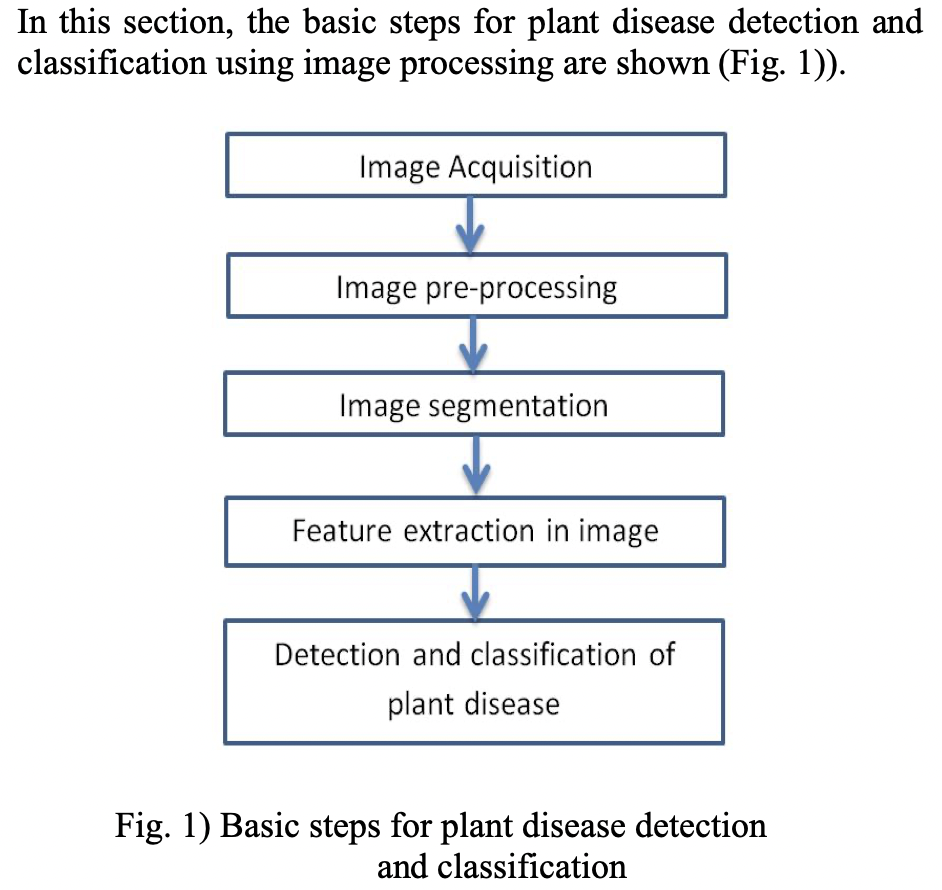


Image from *A new chaotic system with hidden attractor and its engineering applications: analog circuit realization and image encryption* by Unal Çavusoglu, Shirin Panahi, Akif Akgul, Sajad Jafari, Sezgin Kaçar.

The second part of this section is how image processing is used by engineers. We started off, once again, by understanding what image processing is. A digital image is “pictures that have been converted into a computer-readable binary format consisting of logical 0s and 1s” (Bovik 1). Engineers then take these recently converted images and use them to extract more information than the average viewer can see. Bovik describes it as, “Viewable datasets like this can be regarded as images and processed using established techniques for image processing, even if the information has not been derived from visible light sources” (Bovik 2). Engineers can take an image and derive more information from it using the process we are learning about in class. One of the most interesting ways image processing is being used is in plant disease detection. In the article, *Plant Disease Detection Using Image Processing*, the authors describe how they use the familiar software MATLAB to extract data from certain plants. They zoom in on these images and use the algorithms they have created to identify the diseases on the leaves of plants (Khriade 1, Patel 1). The process is shown in the flowchart below:



The two processes of image encryption and processing are a look forward into what some of use in Innovation of Ideas will be working on in the future. The applications for these processes are immense and will only grow over time. Working on a project like this one introduces image encryption and processing to us early so that we will be prepared for when we work with it later in life.

Project Overview and Methods

Image encryption is a key concept used in order to ensure the security of information on the internet. With the advancement of technology, it is important to create techniques in order to counteract the development of theft technology. Encryption and Decryption allows images to be protected by changing the pixel positions or changing the values of the pixels. This allows for the encrypted image to be hard to read, ensuring that the original image will be kept secured. One of the ways to encrypt an image is to manipulate image pixels in the spatial domain (Tang 1). This process mixes up the pixel positions through matrix transformation, therefore turning the original image into a chaotic image. The spatial domain technique uses a secure algorithm and it does not have any size imitations. Another type of encryption method is chaotic systems. These systems have many sensitive properties such as sensitivity to initial conditions and system parameters, but they show better performance than traditional encryption techniques. Therefore, many researchers have tried to design image encryption with chaotic maps (Tang 1). There are two main parts to chaotic systems. One of the parts will change the position of the pixels, while the second part will change the value of the pixel.

One of the more common techniques in cryptography is the XOR operation that is used in ciphers. XOR is a simple bitwise operation that allows cryptographers to create strong encryption systems, and consequently is a fundamental building block of practically all modern ciphers (Wagner 1). The XOR operator is similar to the OR operator. They both rely on binary numbers. However, the XOR operator is active if only one of the inputs is on and the other input is off. The OR operator will do the same, however, the output will also be one if both of the inputs are on. In order to use the XOR cipher by converting the numerical value into a binary number. Then, a key is created that is the same size as the binary number that is created. A XOR operator is used between the original number and the key, which creates a new, encrypted value. It is practically impossible to crack due to the possibilities of the values. The XOR cipher is used in almost all symmetric encryption algorithms and is used in the DES cipher as well (Wagner 5).

For this project specifically, the blue dot of the Earth was found first by converting the image into a grayscale image. Next, the Sobel Operator is used, which is effective at finding the discontinuities in an image using derivatives. This way, the operator will be able to find the Earth because it will create a derivative, and discontinuities correspond to maximums and minimums of a derivative. The output of the Sobel operator will be the magnitude of the gradient. Based on this output, the location of the Earth can be found, and a new picture can be outputted with the Earth as the center of the picture.

Discussion of Algorithm Design

**Background:**

The algorithm of this project is designed with three major purposes in mind: decryption, feature detection (edges/smoothing), and encryption. Several user-defined functions were created to achieve these tasks. Rather than doing everything in the main code-space in python or using one main function, the diversification of tasks into user-defined functions promotes efficiency and modularity in our algorithm’s design. Furthermore, it allows other users/clients to use this project in whatever application deemed necessary regarding image processing.

The decryption part of the algorithm splits the image into red, blue, and green channels and decrypts each channel and consolidates each of these parts into one whole array and into one whole image. In order to decrypt the image we created a key array with the same row and column dimensions as the given encrypted image so that we could apply the bitwise operation in order to decrypt each of the respective color channels. This part of our project is better known as the XOR cipher.

The feature detection of our algorithm makes use of the SciPy library’s gaussian filter function to smooth the image and remove excess noise in the image. Furthermore, the convolve2d function allows the application of a kernel array to detect edges in the x and y-directions to create a gradient map of the edges detected in our image. Next, using the gradient map, we locate the pixels of a certain object (Earth, in the case of the Pale Blue Dot) by using logic of determining where the max pixel value is in our gradient map. From there, the row and column number of this max pixel gives us the information needed to crop the image into a 100 x 101 image with the desired object (the Earth). This provides the client with the exact area of where to find the highest pixel entity in an image no matter the image, thus this algorithm is multi-purpose in nature.

After cropping Earth from our given image, the task for our team involved encrypting the image with a new key. Our team strived to keep the key generation algorithm as simple as possible by involving the use of the Random.randInt() function. In order to generate an extra level of randomness we created an encryption scalar multiple that summed the row and column numbers and found the remainder of division by the number of letter in the inputted key string. Finally, the values of our encrypted scaled random numbers were multiplied by 255 to ensure that the values input into the random numbers key array took on values from 0 to 255, which are appropriate values that an image must take on.

# **Part 1: Image IO**

**What are the data type and the range of the data types if you try to import a PNG file using matplotlib?**

The data type that we used for our PNG file was an integer and the range of the values was 0 to 255.

**How about the JPG image?**

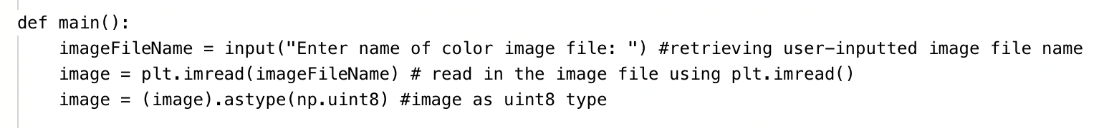
For the JPG image we used the data type uint8, the range for data type uint8 is from 0 to 255.

**How about the TIFF image?**

For the TIFF image we used the data type uint8, the range for data type uint8 is from 0 to 255.

**If the data type of the image data if was not of the unit8 type, how did you convert that?**

In order to convert, we used the astype() function which helped us reach the unit8 type. This process can be seen in the image below:



A picture containing text, whiteboard

Description automatically generated

*Part 1 Flowchart*

## **Part 2: Decryption and XOR Cipher**

**Did your XOR cipher correctly decrypt the example images given to you? How did you check that?**

Yes, our XOR cipher correctly decrypted the example images given to us. This accuracy was checked using the matplotlib’s imshow() and show() functions to see in real-time how our decryption algorithm decrypted the encrypted image.

**How did you design your XOR cipher?**

The XOR cipher was designed with a scalar multiple, A that was set equal to the product of the row and column number modded (%) by the number of letters in the key string. The mod (%) operator returns the remainder between the numerator and denominator in division. The mod can only be in the range [0, numLettersInKeyString-1] thus this multiple was not arbitrarily large at any point. Next, the A scalar was multiplied by a fraction containing 2 \*\* 8 floor division the number of letters in the keyString. The use of a base 2 exponent helps when working with image data because all the values ultimately stored in the image are just binary values that will be applied upon by the XOR cipher.

Once the key array was generated, the image array was split up into R, G, and B channels depending on the layer of the array (0,1, 2 respectively). By splitting up the RGB channels the np.bitwise\_xor() operator was utilized along with a for loop at each row and column to complete the XOR operation between image pixels and the keys stored in the key array.

**What loop did you use?**

The loop that we used for this part of the project is the for loop.

**What functions from the libraries did you use?**

The imported libraries used for this project were: random, matplotlib.pyplot, NumPy, scipy

Within SciPy the signal module was imported in order to do edge detection on our decrypted image. Matplotlib.pyplot was used to read images, plot images, and save these images into the respective directory. And NumPy, allowed us to take an image and convert it into an array that spans the values 0 to 255. NumPy was used for ALL array operations in this project.

**What is the running time of your XOR cipher to decrypt the images?**

**A picture containing company name

Description automatically generated**

*From Windows Clock App, Running Time of Decryption*

**Try to show the relationship between running time and the number of pixels.**

As the number of pixels in the image increases or if the image is color the running time increases due to the greater amount of raw computation and input the computer must deal with when processing images of such massive size with such massive amounts of data.

**What are the pros and cons of the XOR cipher?**

One of the biggest pros of the XOR cipher is the run time. It is a relatively short process compared to other ciphers, meaning it can be used in large blocks of code without highly impacting the run time. The cons of the XOR cipher can be seen when there is repetition in the key. If there is repetition in the key, the XOR cipher can be broken using frequency analysis.

A white board with writing on it

Description automatically generated with medium confidence

*Part 2 Flowchart*

# **Part 3: Feature Detection**

**What is the range of the gradient map?**

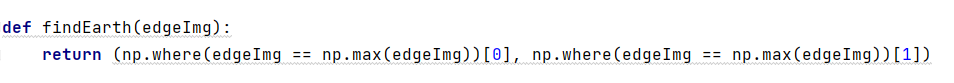
The range of the gradient map is 1152, 1304, 4. This means that there are 1152 rows, 1304 columns, and 4 layers in the gradient map.

Diagram

Description automatically generated *From Microsoft Whiteboard, Visualization of an image array in python*

**How did you find the exact location of the earth? Is there any limitation of your algorithm?**

We found the exact location of the Earth by placing the image in a grayscale image. After the image is grey, the Gaussian filter is used in order to smooth out the image. After applying the Gaussian filter is used, we use the Sobel operator. The Sobel operator is used to detect the edges using derivatives. The Sobel operator will be able to detect the Earth and it will be able to output the pixel number that corresponds to the location of the Earth.



**Did the Gaussian filter (smoothing process) affect your result?**

The Gaussian filter is important in this process because it blurs the image, and it removes the noise and detail. The filter affects our result by allowing the blue dot to be detected by removing other speckles within the image. The Gaussian filter is more effective than other filters because it can keep the edges sharp. The edges are important in this project because the image that we are asked to find it extremely small, so it is important to keep the edges as sharp as possible so that the Earth can be found. Furthermore, the image is grayscale and thus allows us to easily spot the earth which is converted into a white dot as a result of the gray scaling the image.



*Diagram

Description automatically generated Part 3 Flowchart*

*A piece of paper with writing on it

Description automatically generated with medium confidencePart 3 Flowchart (cont.)*

’

# **Part 4: New Key Generator**

**How did you design your new key generator?**

Our new key generator was designed with pseudorandom numbers in mind. Specifically, we first used a scalar value of: the row index added to the column index modded (remainder operator) by the number of letters in our key string. The previous decryption key equation part of our project inspired us to do this as this would add an interesting multiple in front of our keys. Then the encrypted scalar previously discussed was multiplied by the Random.randInt() function to generate random numbers between 1 and 100 for each. By creating two levels of varying randomness in the pixel values for the key Array this allows for the encryption to be more secure. Finally, we took our key values and multiplied them by 255 to ensure that all our integers represented color values that an image can realistically take on.

Graphical user interface, text, application

Description automatically generated

**Do you think the results are better? Why**

Based on before and after picture it could be reasonably agreed upon that no matter what kind of key string, the encryption of the image is well-done and completely masks the context of the original image.

|  |  |
| --- | --- |
| **A picture containing curtain, light, night sky  Description automatically generated**  **Before – Decrypted** | **Background pattern  Description automatically generated**  **After – Encrypted** |

As you can see the image after using our key generator/XOR cipher the image is indistinguishable from the Pale Blue Dot decrypted image. If one was given the encrypted image without context, they would have no idea what this image represents. Therefore, by viewing the image-level differences it can be concluded the encryption results are better than the decrypted image. Another important observation to make is the difference between the red, blue, and green channel histograms before and after encryption (in the inputs-outputs at end of this part). The values of these respective pixels after encryption contain a more rather uniform distribution than before encryption suggesting that our algorithm is doing its job of randomly allocating pixels to certain colors meaning our encryption has a valid level of security and is functioning correctly on an objective level.

Our team is fully aware that there are more complicated algorithms that further promote security/encryption of images and have a quantifiable level of “better” between a decrypted image and an encrypted one. Our aim, however, was to keep our key generator as simple as possible but as secure as possible. By doing this, we could reduce computational power required to process the key generator and further it would not be very challenging to explain to other engineering students/faculty or lay people. The simplicity of our key generator algorithm hinges on the power of the Random module in function and the use of the modulus/remainder operator paired with the number of letters in the key string to create an encryption scalar multiple that is bounded by the values [0, numLetters-1]. The modulus operator normally does not generate massive numbers, and this is very helpful when we multiply by random numbers so the key array can successfully complete the XOR operation with acceptable image data [0,255].

A piece of paper with writing on it

Description automatically generated

*Part 4 Flowchart*

A piece of paper with writing on it

Description automatically generated

*Part 4 Flowchart (cont.)*

References

Bovik, Alan C. *The Essential Guide to Image Processing*. Elsevier/AP, 2009.

Çavuşoğlu, Ünal, et al. “A New Chaotic System with Hidden Attractor and Its Engineering Applications: Analog Circuit Realization and Image Encryption.” *Analog Integrated Circuits and Signal Processing*, Springer US, 28 June 2018, <https://link.springer.com/article/10.1007/s10470-018-1252-z>.

Had, Ambika, et al. “A Review: Image Encryption Techniques and Its Terminologies.” *Ijeat.org*, 2014, <https://www.ijeat.org/wp-content/uploads/papers/v3i4/D2998043414.pdf>.

Khirade, Sachin D. “Plant Disease Detection Using Image Processing.” *IEEE Xplore*, <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7155951>.

Tang, Zhenjun. “Image Encryption with Double Spiral Scans and Chaotic Maps.” *Hindawi*, 15 Jan. 2019.

Wagner, Lane. “Why Is Exclusive or (XOR) Important in Cryptography?” *Qvault*, Qvault, 18 Jan. 2020, <https://qvault.io/cryptography/why-xor-in-cryptography/>.

**Appendices:**

User Manual

**Project Management Plan:**

For this project, we created two teams to accomplish the two parts of this project. We had Pranesh and Joe primarily working on the coding aspect of this project, while Drew and Ryan worked on the deliverables. When we look at the deliverables in particular, Ryan focused on Project Overview and Methods while Drew worked on Project Motivation. They then both worked on the Appendices together in order to complete it at a faster rate. After that we took screenshots of our code and explained what was going on in the section where we answered questions on the project. Pranesh has been the person working on the code and did all of parts 1,2, and 3. Joe has helped on part 4. Pranesh then helped to answer the questions on parts 2 and 4 of the code while Drew and Ryan answered the questions for parts 1 and 3 respectively. Pranesh added visual representations for parts 2, 3, and 4 as well as revising what was said by Ryan and Drew. The final code was then uploaded to the report. The flowcharts for parts 1 and 2 were completed by Pranesh, Drew, and Ryan during class on Thursday. The final two flowcharts were completed by Pranesh and Drew on Sunday before the deadline.

**Discussion of Design Process:**

The first step of the design process is to identify and research a problem or need. Our team was able to identify the project as a problem that we need to solve. We did research on how to encrypt and decrypt images and use the XOR cipher in order to complete the project. The next step of the process is to develop possible solutions. This step can be seen through the flowcharts that we created. The flowcharts gave us a sense of direction while we were making our code. The next step is to plan and to make a prototype. This is when we began to make our first code. Our first code needed improvement, and we could see our problems when we began to test our code. When testing our code, we could see what sections of our code successful and which sections were not working. The next step is to improve the solution. Therefore, we should do more research on our problems, and then implement new solutions so that our code will be more efficient.

# **Part 1: Image IO**

|  |  |
| --- | --- |
| **Inputs** | **Outputs** |
| Image File Name (Pre-encrypted image) | Image converted to numpy array as uint8 type |

## **Part 2: Decryption and XOR Cipher**

|  |  |
| --- | --- |
| **Inputs** | **Outputs** |
| Row,column,num\_layers = image.shape (1152,1304,4)  keyString = ‘COME AND GET YOUR LOVE’ | numLetters in Key String 🡪 18 (using for loop)  A = (row number \* column number) % numLetters  key = A \* (2 \*\* 8 // numLetters)  keyArr[row][col] = key |

# **Part 3: Feature Detection**

|  |  |
| --- | --- |
| **Inputs** | **Outputs** |
| Key array, encrypted image | Decrypted image |
| Decrypted image | Grayscale Image |
| Grayscale Image | Smoothed Image (Gray) |
| Smoothed Image (Gray)    kernelX = [[-1,0,1],  [-2,0,2],  [-1,0,1]]  kernelY = [[1,2,1],  [0,0,0],  [-1,-2,-1]]  g\_x = signal.convolve2d(smoothed image, kernelX)  g\_y = signal.convolve2d(smoothed image, kernel) | Edge Image (Gradient Map, Edges Detected)  *np.hypot(g\_x, g\_y)* |
| Edge Image (Gradient Map, Edges Detected)  *np.hypot(g\_x, g\_y)* | findEarth(Edge Image)  Row of Earth: 652, Column of Earth: 769 |
| findEarth(Edge Image)  Row of Earth: 652, Column of Earth: 769 | cropEarth(652, 769)  w=100  h=101  croppedImg = edges[x-(int(w/2)):x+(int(w/2)), y-(int(h/2)):y+(int((h+1)/2)), :]  creates a 100 x 101 (50/51 pixels above and below) image with earth in the center  **Cropped Image of Earth** |

# **Part 4: New Key Generator**

|  |  |
| --- | --- |
| **Inputs** | **Outputs** |
| **Decrypted Image:**    **Red Channel Histogram (before encryption)**    **Blue Channel Histogram (before encryption)**    **Green Channel Histogram (before encryption)** | **Encrypted Image:**    **Encrypted Array:**  ***Initial Key: come get love***  [[ 0 184 92 ... 55 176 166]  [204 228 205 ... 192 161 68]  [228 52 240 ... 227 252 36]  ...  [ 36 2 138 ... 240 57 154]  [ 52 137 128 ... 130 112 0]  [ 53 40 242 ... 172 0 198]]  **Red Channel Histogram (after encryption)**    **Blue Channel Histogram (after encryption)**    **Green Channel Histogram (after encryption)** |

# **Full Code for Project:**

|  |
| --- |
| *"""* *===============================================================================* *ENGR 13300 Fall 2021*  *Program Description*  *Assignment Information*  *Assignment: Team HW5 - Py4, Task 2*  *Author: Pranesh Monda, pmonda@purdue.edu*  *Team ID:*  *LC2 - 05 (e.g. LC1 - 01; for section LC1, team 01)*  *Contributor: Sijie Zhang, zhan2355@purdue.edu*  *My contributor(s) helped me:*  *[ ] understand the assignment expectations without*  *telling me how they will approach it.*  *[ ] understand different ways to think about a solution*  *without helping me plan my solution.*  *[ ] think through the meaning of a specific error or*  *bug present in my code without looking at my code.*  *Note that if you helped somebody else with their code, you*  *have to list that person as a contributor here as well.*  *ACADEMIC INTEGRITY STATEMENT* *I have not used source code obtained from any other unauthorized* *source, either modified or unmodified. Neither have I provided* *access to my code to another. The project I am submitting* *is my own original work.* *===============================================================================* *"""*  **import** random **import** matplotlib.pyplot **as** plt **import** numpy **as** np **from** scipy **import** signal   **def** decrypt\_image(image, keyArr):  decrypted = np.zeros(image.shape, dtype=np.uint8) *#1152\*1304\*4, set with dimensions of original image's shape to make appropriately sized key array*  **for** i **in** range(len(image)):  **for** j **in** range(len(image[0])):  decrypted[i][j][0] = (keyArr[i][j]) ^ (image[i][j][0]) *#decryption of the red color channel* decrypted[i][j][1] = (keyArr[i][j]) ^ (image[i][j][1]) *#decryption of the green color channel* decrypted[i][j][2] = (keyArr[i][j]) ^ (image[i][j][2]) *#decryption of the blue color channel*  **return** decrypted  **def** grayScale(image): *#Part 3* imgFile = plt.imread(image)  imArr = np.array(imgFile)  gray = np.zeros(np.shape(imArr), dtype=np.float64) *#a numpy array with zeros that will eventually store the grayscale image* **for** row **in** range(len(imArr)):  **for** col **in** range(len(imArr[0])):  gray[row][col] = int(0.2126 \* imArr[row][col][0] + 0.7152 \* imArr[row][col][1] + 0.0722 \* imArr[row][col][2]) *#this equation is applied to each gray array index to take color pixel values and turn them into grayscale*  gray = np.array(gray, dtype=np.uint8) *#now the grayscale array has type uint8* plt.imsave(**'gray.tiff'**, gray)  **return** gray  **def** sobel\_edge\_detection(decryptedGray):  gau\_filter = np.array([[1, 4, 6, 4, 1], [4, 16, 24, 16, 4], [6, 24, 36, 24, 6], [4, 16, 24, 16, 4], [1, 4, 6, 4, 1]]) *#this is the smoothing kernel* imgSmoothed = signal.convolve2d(decryptedGray, gau\_filter) *#the scipy signal function applies the gaussian filter to smooth out the image and reduce noise in the grayscale image that is passed into the function* plt.imsave(**"plainSmooth.tiff"**, imgSmoothed, cmap=**"gray"**)    kernelx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]) *#this kernel array finds edges in the x direction* kernely = np.array([[1,2,1], [0,0,0], [-1,-2,-1]]) *#this kernal array finds edges in the y direction* g\_x = signal.convolve2d(imgSmoothed, kernelx) *#by using convolve2d, the edges in x-direction are discovered* g\_y = signal.convolve2d(imgSmoothed, kernely) *#by using convolve2d, the edges in y-direction are discovered*  edgeImg = np.hypot(g\_x,g\_y) *#in order to get the full picture, taking the square root of the squares of the x and y direction deterrmines provides the average of what the image looks like as whole with edges detected* plt.imshow(edgeImg, cmap=**'gray'**)  plt.show()  plt.imsave(**"gradientMap.tiff"**, edgeImg, cmap=**"gray"**)  earthRow, earthCol = findEarth(edgeImg)  earthRow = earthRow[0] *#this variable includes only the value of the row number of Earth's pixel* earthCol = earthCol[0] *#this variable includes only the value of the column number of Earth's pixel* print(**f"Row of Earth: {**earthRow**}, Column of Earth: {**earthCol**}"**)  cropEarth(**"gradientMap.tiff"**, earthRow, earthCol)  **def** findEarth(edgeImg):  **return** (np.where(edgeImg == np.max(edgeImg))[0], np.where(edgeImg == np.max(edgeImg))[1]) *#one return statement returns exactly the earth's row and column as element one then another element*  **def** cropEarth(imgEdgeName, x, y):  edges = plt.imread(imgEdgeName) *#this is the grayscale picture that has edges detected* h=100 *#height cropping area* w=101 *#width cropping area* croppedImg = edges[x-(int(w/2)):x+(int(w/2)), y-(int(h/2)):y+(int((h+1)/2)), :] *#adding and subtracting w and h values respectively in order to capture 50 pixels/51 pixels above and below the earth to produce a cropped 100 x 101 image, values are* plt.imshow(croppedImg)  plt.show()  plt.imsave(**"cropped.tiff"**, croppedImg[:, :, 0:3])  **def** generateKey(decryptedImage):  inputKey = input(**"Enter initial key: "**)  numLetters = 0  **for** letter **in** inputKey:  **if**(letter != **" "**):  numLetters+=1 *#counts number of letters in entered keystring (exludes spaces)* rand\_nums = np.zeros((decryptedImage.shape[0],decryptedImage.shape[1]), dtype=np.uint8) *#new array that will hold keys for encryption, set with proper shape requirements* **for** i **in** range(len(rand\_nums)):  **for** j **in** range(len(rand\_nums[0])):  encryptScal= (i+j) % numLetters *#this scalar value adds the row and column number and finds the remainder when divided by the number of letters in the key string* key = (encryptScal\*(random.randint(1,100))) *#key is scalar multiple types a random integer from 1 to 100.* key \*= 255 *#multiplication by 255, ensures random values fit in the appropriate values taken on by an image array* rand\_nums[i][j] = key   print(rand\_nums)  **return** rand\_nums  **def** encryptImage(decryptedImg ,keyArr):  encrypted = np.zeros(decryptedImg.shape, dtype=np.uint8) *# 1152\*1304\*4*  **for** i **in** range(len(decryptedImg)):  **for** j **in** range(len(decryptedImg[0])):  encrypted[i][j][0] = (keyArr[i][j]) ^ (decryptedImg[i][j][0])  encrypted[i][j][1] = (keyArr[i][j]) ^ (decryptedImg[i][j][1])  encrypted[i][j][2] = (keyArr[i][j]) ^ (decryptedImg[i][j][2])   **return** encrypted   **def** main():  imageFileName = **""** invalidInput = **True**  **while**(invalidInput == **True**):  imageFileName = input(**"Enter name of color image file: "**) *# retrieving user-inputted image file name* **if** (imageFileName.\_\_contains\_\_(**".tiff"**)):  image = plt.imread(imageFileName) *# read in the image file using plt.imread()* image = (image).astype(np.uint8) *# image as uint8 type* invalidInput = **False**  **else**:  print(**"Please enter a valid file name"**)   *#Part 2* row, column, num\_layers = image.shape *# (1152,1304,4)*   **for** row\_index **in** range(len(image)):  **for** column\_index **in** range(len(image[0])):  r = image[row\_index, column\_index, 0]  g = image[row\_index, column\_index, 1]  b = image[row\_index, column\_index, 2]   keyString = **'COME AND GET YOUR LOVE'** countLetters = 0   **for** letter **in** keyString:  **if** (letter != **' '**):  countLetters += 1  dim = (image.shape[0],image.shape[1]) *#(1152, 1304)* keyArr = np.zeros(dim) *#2D array of dimensions of the image only rows and columns*  **for** i **in** range(row):  **for** j **in** range(column):  A = (i \* j) % countLetters *#here % finds the remainder of the quotient* key = A \* ((2 \*\* 8) // countLetters) *#A acts as a scalar multiple of the fraction: 2 to the power of 8 floor divided by number of letters in keyString (floor division brings down division to nearest lower integer)* keyArr[i][j] = key   keyArr = keyArr.astype(np.uint8) *#key array needs to be type uint8 in order to apply the XOR cipher* decryptedArr = decrypt\_image(image, keyArr) *#this calls the function decrypt\_image which does XOR operations on each layer/channel of the image* decryptedArrProper = decryptedArr[:,:,0:3]  plt.imsave(**'plain.tiff'**, decryptedArrProper)   *#Part 3* decryptedGray = grayScale(**'plain.tiff'**)  sobel\_edge\_detection(decryptedGray[:,:,0])   *#Part 4*  plt.hist(decryptedArrProper[:,:,0].reshape(decryptedArrProper.shape[0]\*decryptedArrProper.shape[1]), bins=np.arange((2\*\*8)+1))  plt.savefig(**'rhist.tiff'**)  plt.show()  plt.hist(decryptedArrProper[:,:,1].reshape(decryptedArrProper.shape[0]\*decryptedArrProper.shape[1]), bins=np.arange((2\*\*8)+1))  plt.savefig(**'ghist.tiff'**)  plt.show()  plt.hist(decryptedArrProper[:,:,2].reshape(decryptedArrProper.shape[0]\*decryptedArrProper.shape[1]), bins=np.arange((2\*\*8)+1))  plt.savefig(**'bhist.tiff'**)  plt.show()   keyArrEncrypt = generateKey(decryptedArrProper)  encryptedArr = encryptImage(decryptedArrProper, keyArrEncrypt)    *###################################################* *#For developing two encrypted images to be placed into Box Drive*  *# testImageForEncryption = plt.imread("purduebelltower.tiff")*  *# keyArrEncrypt = generateKey(testImageForEncryption[:, :, 0:3])*  *# encryptedArr = encryptImage(testImageForEncryption[:, :, 0:3], keyArrEncrypt)*  *# plt.imshow(encryptedArr)*  *# plt.show()*  *# plt.imsave('LC2\_05\_Image\_1.tiff', encryptedArr)*  *#*  *# testImageForEncryption2 = plt.imread("enggfountain.tiff")*  *# keyArrEncrypt = generateKey(testImageForEncryption2[:, :, 0:3])*  *# encryptedArr = encryptImage(testImageForEncryption2[:, :, 0:3], keyArrEncrypt)*  *# plt.imshow(encryptedArr)*  *# plt.show()*  *#*  *# plt.imsave('LC2\_05\_Image\_2.tiff', encryptedArr)* *##############################################################*  plt.hist(encryptedArr[:, :, 0].reshape(encryptedArr.shape[0] \* encryptedArr.shape[1]),  bins=np.arange((2 \*\* 8) + 1))  plt.savefig(**'rhistEncrypted.tiff'**)  plt.show()  plt.hist(encryptedArr[:, :, 1].reshape(encryptedArr.shape[0] \* encryptedArr.shape[1]),  bins=np.arange((2 \*\* 8) + 1))  plt.savefig(**'ghistEncrypted.tiff'**)  plt.show()   plt.hist(encryptedArr[:, :, 2].reshape(encryptedArr.shape[0] \* encryptedArr.shape[1]),  bins=np.arange((2 \*\* 8) + 1))  plt.savefig(**'bhistEncrpyted.tiff'**)  plt.show()  **if** \_\_name\_\_ == **"\_\_main\_\_"**:  main() |

# **Full Output of Code:**

Enter name of color image file: Pale\_Blue\_Dot\_Encrypted.tiff

Row of Earth: 652, Column of Earth: 769

Enter initial key: come get love

[[ 0 184 92 ... 55 176 166]

[204 228 205 ... 192 161 68]

[228 52 240 ... 227 252 36]

...

[ 36 2 138 ... 240 57 154]

[ 52 137 128 ... 130 112 0]

[ 53 40 242 ... 172 0 198]]