Project Motivation

The main purpose of image encryption is to send and receive sensitive material. *Computational Image Encryption Techniques* states "Images contain very sensitive and confidential information. Because images play a significant role in many applications such as military communication, remote-sensing, and medical-imaging, therefore, it is necessary to protect sensitive and confidential information from unauthorized use and modification." UAVs are widely used in military operations and their video streams need to be encrypted to protect sensitive materials. This means when engineers are designing a drone system that they must create efficient video encryption algorithms so the drone pilot can get a seamless video stream that won't be delayed. *Embedded Real-Time Video Encryption Module on UAV Surveillance Systems* states "Now, there are new mission for UAV, like combat, intelligence, and civil applications [6]. Because any transmission from a military UAV must be secured it's a must to use an encryption method to protect de confidentiality of the transmitted data." Drones have a lot more advantages in the military than regular aircraft: they are cheaper to build, easier to maintain, harder to detect, and don't put human life at risk. Turkey has been a world leader in small drones and exports this technology all over the world. A research paper from Gazi University talks about the benefits of putting the image encryption locally on a drone, to increase the reliability of the information being transmitted. "The UAVs/drones' low cost and easy obtainability have also increased the questions about their reliabilities. As the UAVs themselves can be used as an attack tool, also the attacks made to them have become a situation that is often encountered. It is very important that air vehicles are physically reliably that software and data transfer are reliable. To achieve this goal, the first concern to consider and improve is to increase the reliability of data link usage in UAVs. In both the world and our country, although UAVs are produced by many companies, the data link/communication security problem continues. As well as transmitting data to long distances, it is very important to send the right information to the right spot."

Another application of image encryption is in Biomedical Engineering where patients' sensitive data must be encrypted and stored. Since most medical images are extremely detailed, big, and sometimes 3D; engineers need to create secure and efficient algorithms to make sure patient data is secure and accessible. As more medical technology becomes digital these encryption algorithms are key to progress. *Simultaneous encryption and compression of medical images based on optimized tensor compressed sensing with 3D Lorenz* states "In recent years, numerous studies on encryption of medical images, such as computed tomography (CT) and magnetic resonance imaging (MRI), have been reported [1–6], although most of them did not consider compression during encryption. The storage, transmission, and retrieval of massive bio-information should meet several compulsory requirements [7]: (1) high efficiency for rapid transmission and prompt retrieval; (2) strict information security to guarantee users' privacy; and (3) high data fidelity to preserving the pathological information. It requires decreasing the quantity of data to be transmitted (compression) and protecting such data against unauthorized access (encryption). Therefore, simultaneous compression and encryption technology of medical images that are represented as three-dimensional (3D) volumes has additional meanings."

References:

CIPRIAN RĂCUCIU, NICOLAE JULA, CONSTANTIN BĂLAN & COSMIN ADOMNICĂI. (2008, April 21). *Embedded Real-Time Video Encryption Module on UAV Surveillance Systems*. Communications Department Military Technical Academy. Retrieved October 21, 2021 from https://www.researchgate.net/publication/228418297_Embedded_real-time_video_encryption_module_on_UAV_surveillance_systems

Dursun, M. (2017, April 20). *Secure Video Streaming Implementation for Unmanned Air Vehicle (UAV) Data Link with Raspberry Pi 3 over https*. Retrieved October 21, 2021, from https://dergipark.org.tr/tr/download/article-file/416610.

Kaur, M., Singh, S., & Kaur, M. (2021, July 19). *Computational Image Encryption Techniques: A Comprehensive Review*. Mathematical Problems in Engineering. Retrieved October 20, 2021, from https://www.hindawi.com/journals/mpe/2021/5012496/.

Wang, Q., Chen, X., Wei, M. *et al.* Simultaneous encryption and compression of medical images based on optimized tensor compressed sensing with 3D Lorenz. *BioMed Eng OnLine* **15,** 118 (2016). https://doi.org/10.1186/s12938-016-0239-1

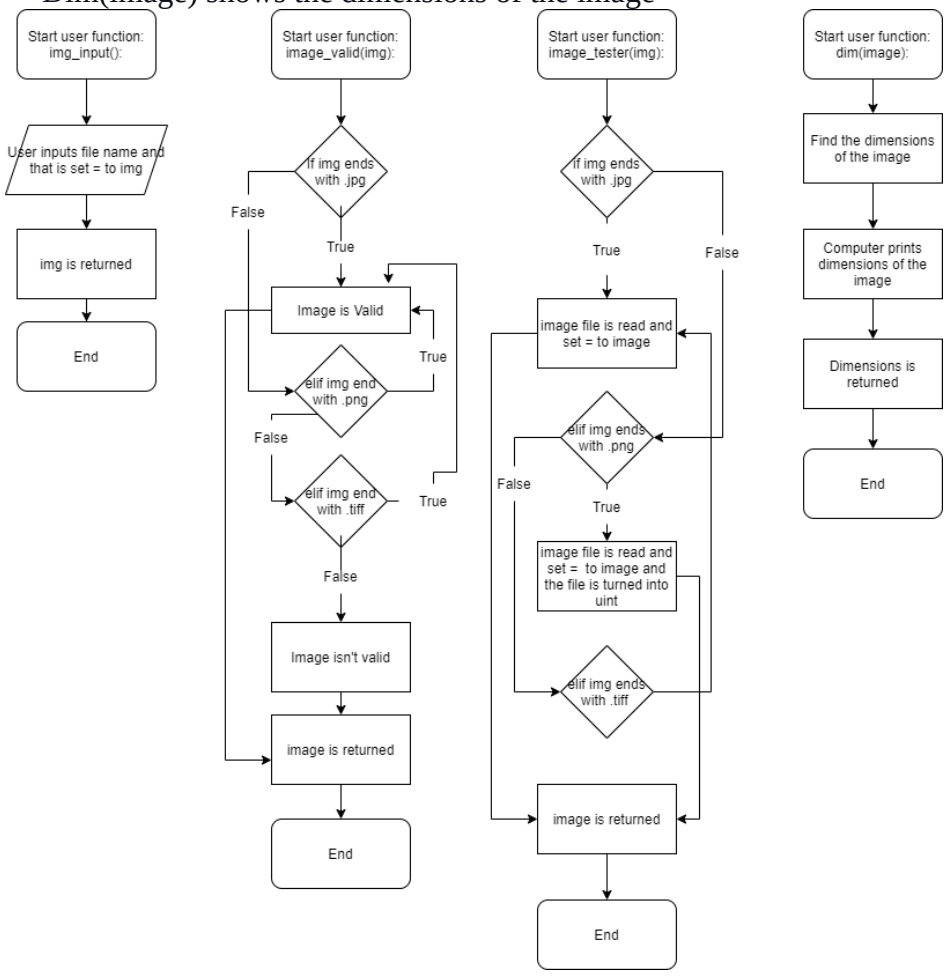
Project Overview and Methods

- Image encryption and decryption is a method of hiding the information in an image by scrambling (encryption) the information using a key or password of some sort so that it can be unscrambled (decryption) by anyone else who has the key.
- The XOR cipher compares each binary bit of the image and the key using the exclusive or logic operation to alter each bit of the image according to the key. The upside to this method of encryption is that the XOR operator acts a lot like a Palindrome, as comparing the key with either the encrypted or decrypted image will decrypt or encrypt the image respectively.
- To get the gradient map, a few steps must be taken:
    - The image must first be smoothed to remove noise. Noise in the image may be interpreted as an edge later on in image processing
    - Next, the gradient of the image must be taken across each row of the image, treating each row as a function of brightness in terms of the position and taking the first derivative of that function with respect to position.
    - Next, the gradient of the image must be taken across each column of the image, treating each row as a function of brightness in terms of the position and taking the first derivative of that function with respect to position.
    - The row-wise and column-wise gradients must then be combined by finding $\sqrt{(row\ gradient)^2 + (column\ gradient)^2}$ for each pixel of the two gradients to obtain the gradient map

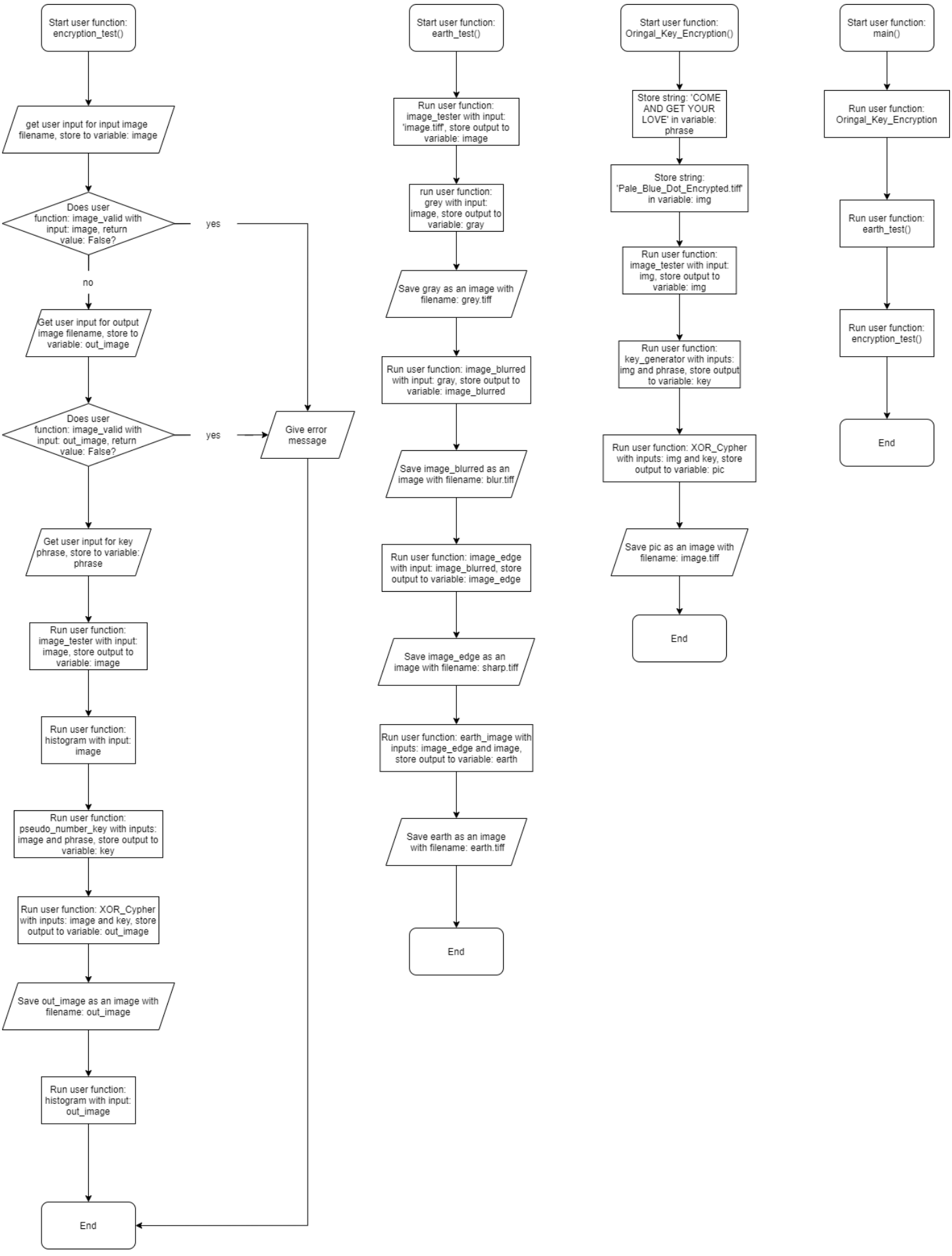
- T The edges in the original image are where the brightest spots in the gradient map are, so edge detection looks at these parts of the gradient map to find edges.
- We used an XOR cipher and key to encrypt our images because an XOR statement can be applied once to encrypt it and applying the XOR cipher again can decrypt it.

Discussion of Algorithm Design

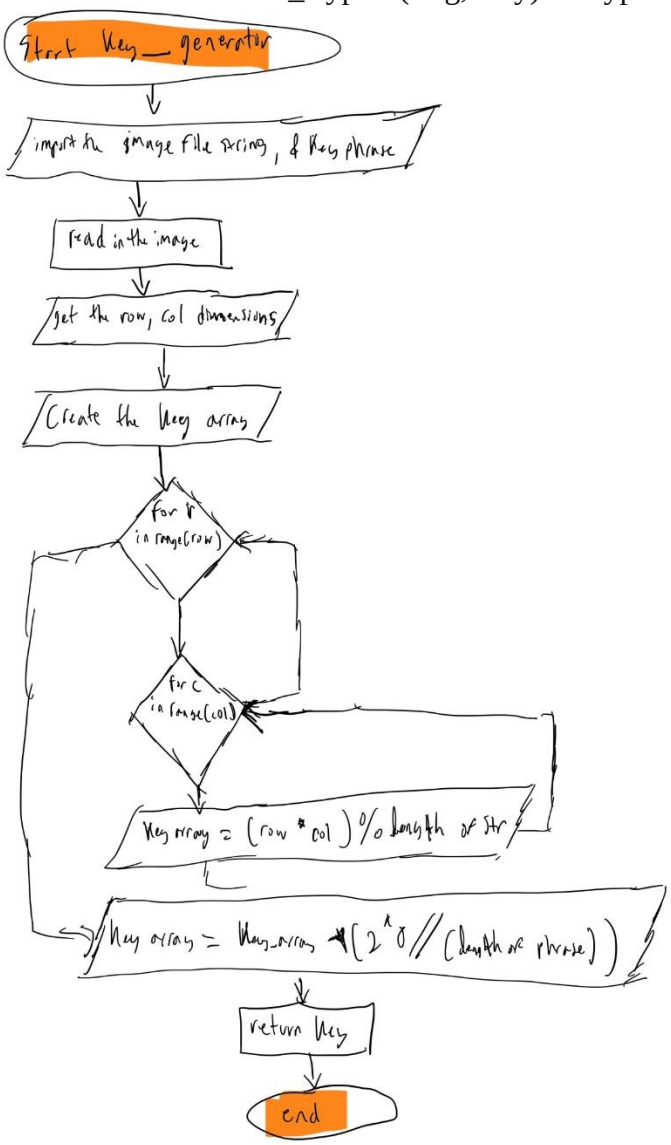
- We split the code into 4 separate python files: Image_analysis, grey_search, key_generator, histo_key based on the 4 different parts of the project.
    - **Image_analysis** contains 4 user defined functions
        - Img_input asks the user to input a file name
        - Image valid checks to see whether the image is either a .jpg, .png, or .tiff file
        - Image tester test the ending of the input file and then reads the file correctly depending on the ending
        - Dim(image) shows the dimensions of the image

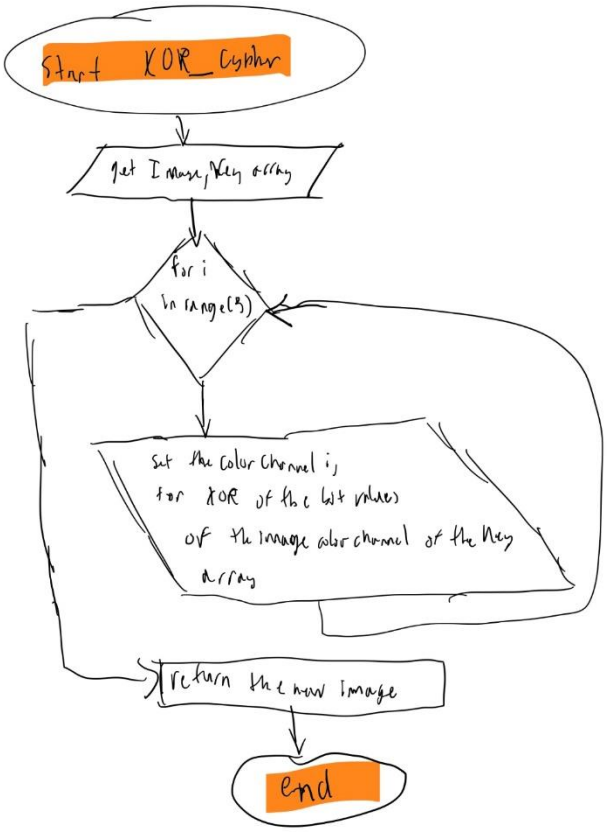


- **Grey_search** contains 7 user-defined functions
    - Grey(image) gets the RGB values & dimension of the image and inserts the grey RGB values into the grey photo
    - Earth_finder finds the location of the brightest dot in the mage
    - Earth image teas the location from the earth finder and gets the array from the original image and outputs 50 in each direction
    - Image_blur(image) blurs the image
    - edge_dectector(blurred_image) gets the gradient of the blurred image in both the x and y direction and then combines the horizontal and vertical edges
    - Float64_uint8(img) gets the max number the data could be and then times the RGB values by 255 to get uint8
    - Uint8_float64(img) divides the uint8 valeus to get the 0-1 float64 values

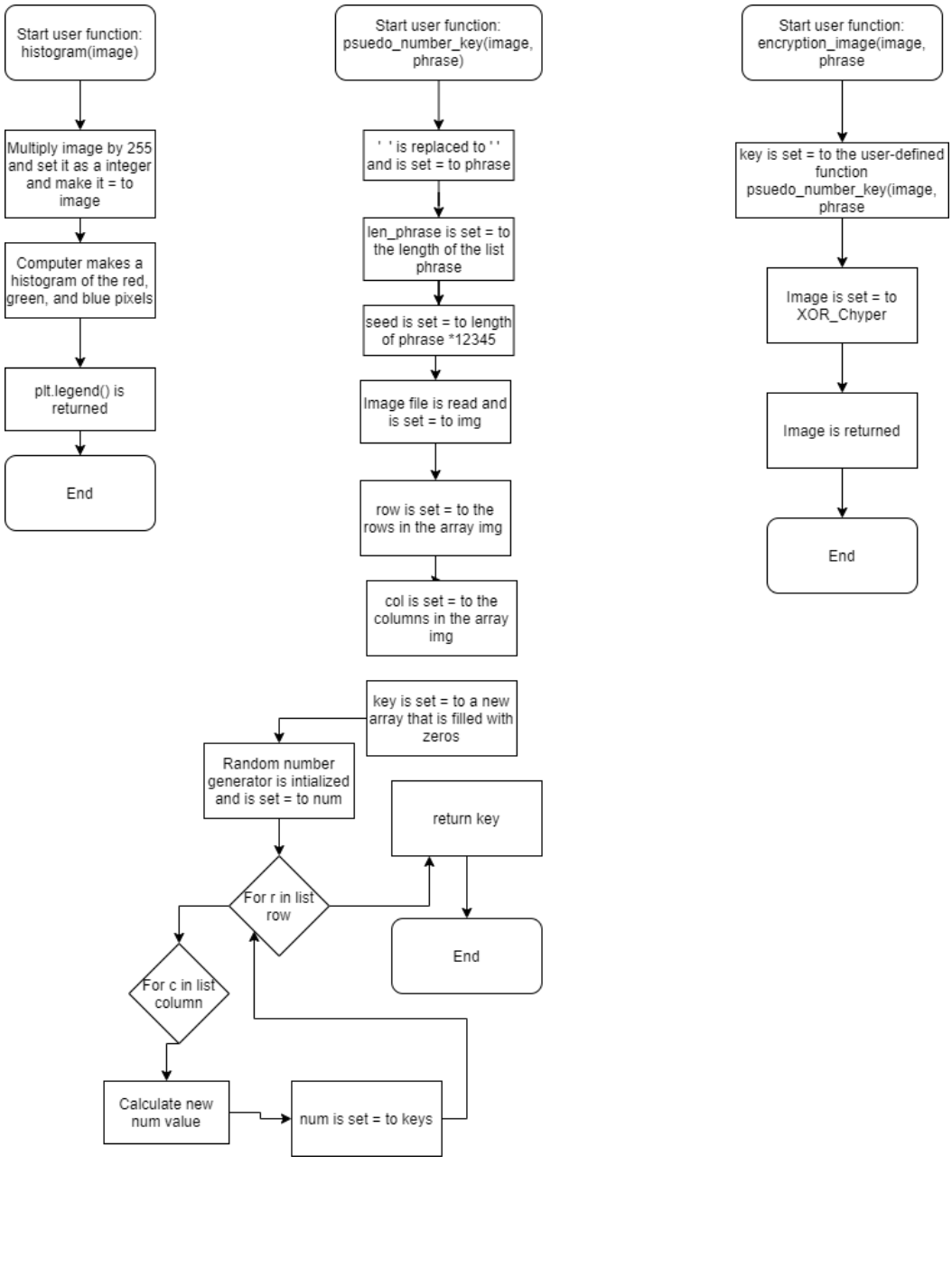

Start user function: encryption_test()
get user input for input image filename, store to variable: image
Does user function: image_valid with input: image, return value: False?
yes
no
Get user input for output image filename, store to variable: out_image
Does user function: image_valid with input: out_image, return value: False?
yes
Give error message
Get user input for key phrase, store to variable: phrase
Run user function: image_tester with input: image, store output to variable: image
Run user function: histogram with input: image
Run user function: pseudo_number_key with inputs: image and phrase, store output to variable: key
Run user function: XOR_Cypher with inputs: image and key, store output to variable: out_image
Save out_image as an image with filename: out_image
Run user function: histogram with input: out_image
End
Start user function: earth_test()
Run user function: image_tester with input: 'image.tiff', store output to variable: image
run user function: grey with input: image, store output to variable: gray
Save gray as an image with filename: grey.tiff
Run user function: image_blurred with input: gray, store output to variable: image_blurred
Save image_blurred as an image with filename: blur.tiff
Run user function: image_edge with input: image_blurred, store output to variable: image_edge
Save image_edge as an image with filename: sharp.tiff
Run user function: earth_image with inputs: image_edge and image, store output to variable: earth
Save earth as an image with filename: earth.tiff
End
Start user function: Oringal_Key_Encryption()
Store string: 'COME AND GET YOUR LOVE' in variable: phrase
Store string: 'Pale_Blue_Dot_Encrypted.tiff' in variable: img
Run user function: image_tester with input: img, store output to variable: img
Run user function: key_generator with inputs: img and phrase, store output to variable: key
Run user function: XOR_Cypher with inputs: img and key, store output to variable: pic
Save pic as an image with filename: image.tiff
End
Start user function: main()
Run user function: Oringal_Key_Encryption
Run user function: earth_test()
Run user function: encryption_test()
End

- **Key_generator** contains 2 user-defined functions
  - Key_generator(img, key_str) gets the dimensions of the image, creates a new array, and then loads the key values into the key array
  - XOR_Cypher(Img, Key) encrypts the image



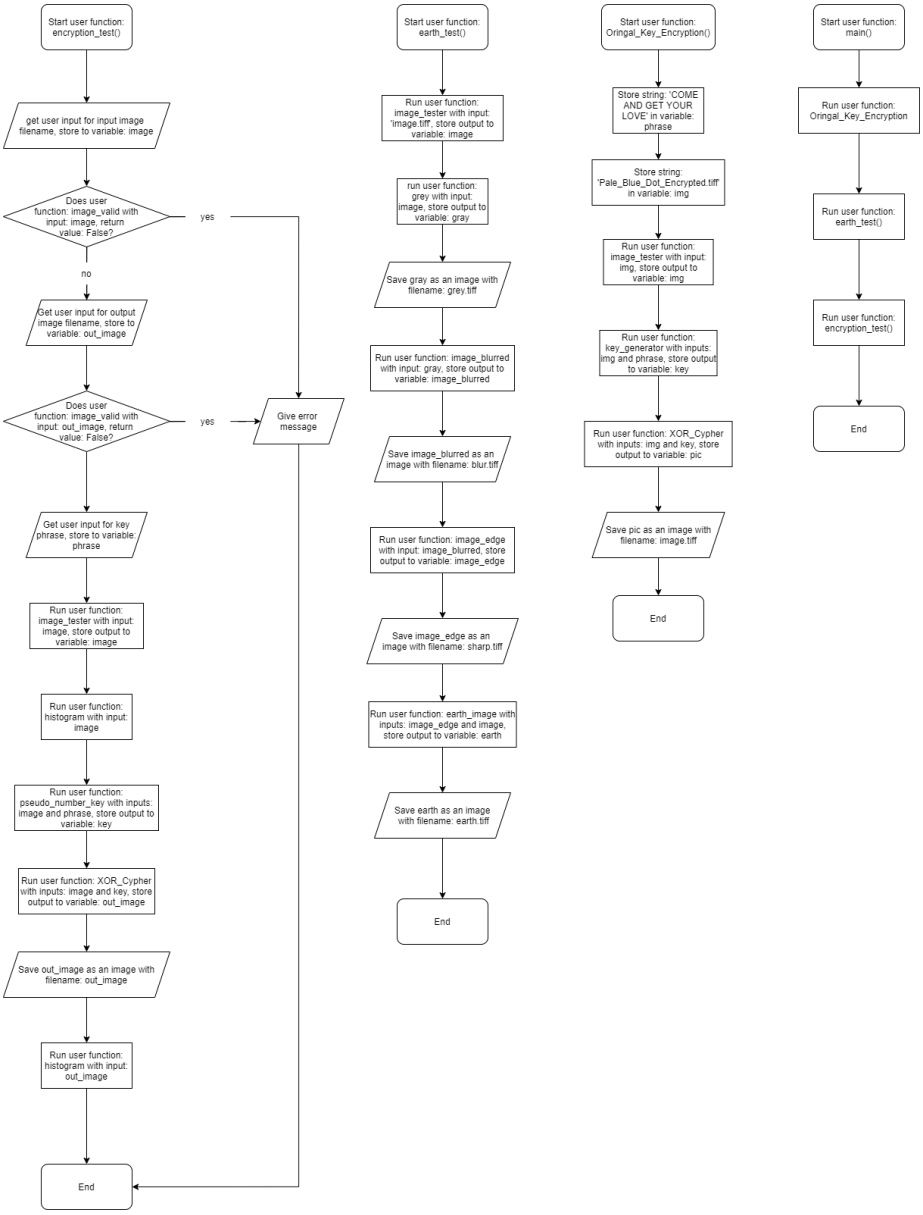


- **Histo_key** has two user-defined functions
  - Histogram(key) creates a histogram from the image inputted back in image_analysis
  - Pseudo_number_key(images, phrase) creates the seed by the length of the phrase. It then loads the dimensions of the image and creates a new key array. Finally, it leads the seed values into the array


Start user function: histogram(image)
Multiply image by 255 and set it as a integer and make it = to image
Computer makes a histogram of the red, green, and blue pixels
plt.legend() is returned
End
Start user function: psuedo_number_key(image, phrase)
' ' is replaced to ' ' and is set = to phrase
len_phrase is set = to the length of the list phrase
seed is set = to length of phrase *12345
Image file is read and is set = to img
row is set = to the rows in the array img
col is set = to the columns in the array img
key is set = to a new array that is filled with zeros
Random number generator is intialized and is set = to num
For r in list row
return key
End
For c in list column
Calculate new num value
num is set = to keys
Start user function: encryption_image(image, phrase
key is set = to the user-defined function psuedo_number_key(image, phrase
Image is set = to XOR_Chyper
Image is returned
End

main0.py

- Demonstrates that the code works for the TA's and anyone who wants to test the code



Questions

Part 1

1. For a PNG file, the data type is float32. For a JPG file and a TIFF image, the data type is uint8.
2. We used the .astype(np.uint8) to convert the image data to a uint8 type.

Part 2

1. Yes, the XOR cipher worked correctly. We were able to verify that since the cipher decrypted the image on the first run of the code and encrypted the image on the second run of the code.
2. Originally, we use a nested loop that would cover the rows and columns. In the for loop, we converted the integers to bits and then used an XOR stamen by using the '^' operator. We then found a better way to use the NumPy library. Using NumPy.bitwise_xor we used and for loop in range(3) for the color channels, we used that XOR the two matrices. 0.005871514854 seconds

3. It took 0.005871514854 seconds for the smaller 6 MB image and 0.1974872185480 seconds for the larger 150 MB image. The number of rows and columns does make a difference in the run time of the program.
4. The cons of the XOR cipher is that if one of the subkeys is known then the image can be partially decrypted and the person can brute force the encryption for the master key. Compared to other logical operators the XOR is equally likely to be a 0 or a 1. The XOR cipher will also not leak information about the original value and the same XOR function can be used to both decrypt and encrypt the information.

Part 3

1. The range of the gradient map is the size of the original image
2. The exact location of the earth was found by looking at the brightest point on the gradient map. If there is more than one point or edge in the image, only the sharpest (and therefore brightest on the gradient map) edge will be looked at even though all the edges of the image are found
3. The Gaussian filter for smoothing the image removed noise from areas, thus preventing edge detection in places where edges were not present.

Part 4

1. First, the computer creates the seed by the length of the phrase. It then loads the dimensions of the image and creates a new key array. Finally, it leads the seed values into the array. Key is then returned.
2. The results are better. In the histogram, the color values are scrambled much better which indicated better encryption.

References

Gouillart, Emmanuelle, and Gaël Varoquaux. "2.6. Image Manipulation and Processing Using Numpy and Scipy¶." *2.6. Image Manipulation and Processing Using Numpy and Scipy - Scipy Lecture Notes*, scipy-lectures.org/advanced/image_processing/.

NumPy, numpy.org/.

"Numpy.hypot." *Numpy.hypot - NumPy v1.15 Manual*, docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.hypot.html.

"Scipy.ndimage.sobel." *Scipy.ndimage.sobel - SciPy v1.7.1 Manual*, docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.sobel.html.

"Scipy.org." *SciPy.org - SciPy.org*, www.scipy.org/.

"Visualization with Python." *Matplotlib*, matplotlib.org/.

"numpy.bitwise_xor() in Python", *GeeksforGeeks*, https://www.geeksforgeeks.org/numpy-bitwise_xor-in-python/

"numpy.histogram" *numpy.histogram - SciPy v1.7.1, NumPy v1.15 Manual*, https://numpy.org/doc/stable/reference/generated/numpy.histogram.html

Arihant Kr Bhopal, and Namita Tiwari, "Image Encryption using Pseudo Random Number Generators", *International Journal of Computer Applications, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.404.1890&rep=rep1&type=pdf, April 2013*

"Linear congruential generator", *Wikipedia*, https://en.m.wikipedia.org/wiki/Linear_congruential_generator

"numpy.where() in Python", *GeeksforGeeks*, https://www.geeksforgeeks.org/numpy-where-in-python/, 12/03/2020

"numpy.hypot() in Python", *GeeksforGeeks*, https://www.geeksforgeeks.org/numpy-hypot-python/, 11/18/2020

"numpy.amax() in Python", *GeeksforGeeks*, https://www.geeksforgeeks.org/numpy-amax-python/, 11/18/2020

"Python zip() Function", *w3schools*, *https://www.w3schools.com/python/ref_func_zip.asp*

"Data Types", *Data Types*, *NumPy v1.15 Manual*, https://numpy.org/doc/stable/user/basics.types.html

"scipy.ndimage.gaussian_filter", scipy.ndimage.gaussian_filter - *SciPy v1.7.1 Manual*, https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.gaussian_filter.html

"Finding edges with Sobel filters", *scipy-lectures*, https://scipy-lectures.org/advanced/image_processing/auto_examples/plot_find_edges.html

*"*Grayscale", *Wikipedia*, *https://en.wikipedia.org/wiki/Grayscale*

Appendicies

a)

Image_analysis.py

Img_input()

- Will prompt the user for the image file name and return the string.

Image_valid(img)

- Given an image file name, will output true or false to tell if it's a .tiff, .jpg, .png or not
- Returns a bool.

image_tester(img)

- Make sure the type(img) == string and the image file is in your directory.
- This will read the image file input and read the file into an array.

dim(image)

- Make sure type(image) == <class 'numpy.dtype'> .
- This will print the dimensions of the array and return the dimensions.

```
image = image_tester('image.tiff')
dim(image)
>>> Dimensions of the image is (1152, 1304, 3)
```

```
print(image_valid(image))
>>> True
```

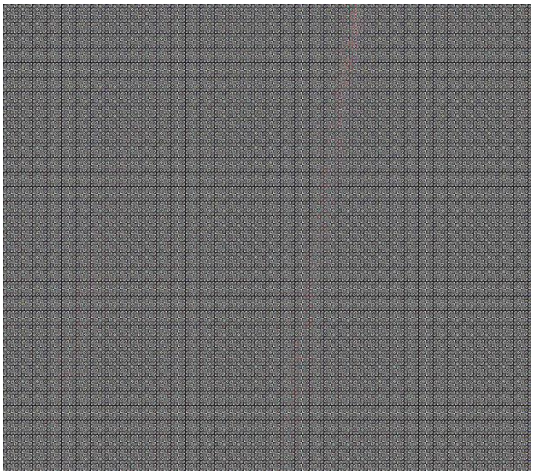
key_generator.py

key_generator(img, key_str)

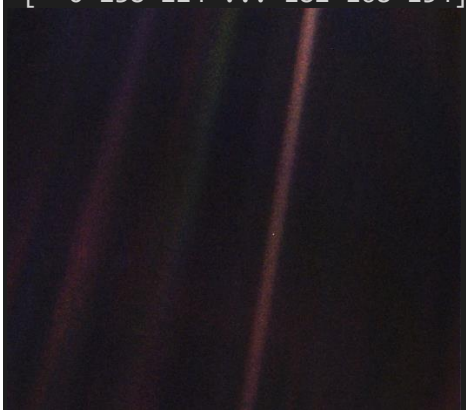
- Make sure type(image) == <class 'numpy.dtype'> with dtype == uint8, type(key_str) == string.
- Will return a key, that is the same dimensions as the image inputed.

XOR_Cypher(Img, Key)

- Make sure type(image) == <class 'numpy.dtype'> with dtype == uint8 and type(Key) == np.array.
- Will do an XOR operation on the Img with the Key inputed.
- Then it will return the image after the XOR operations are complete.



```
phrase = 'COME AND GET YOUR LOVE'
img = 'Pale_Blue_Dot_Encrypted.tiff'
img = plt.imread(img)[:,:,:3]
key = key_generator(img, phrase)
print(key)

pic = XOR_Cypher(img, key)
plt.imshow(pic)
>>> [[  0   0   0 ...   0   0   0]
 [  0  14  28 ...  70  84  98]
 [  0  28  56 ... 140 168 196]
 ...
 [  0 210 168 ...  42   0 210]
 [  0 224 196 ... 112  84  56]
 [  0 238 224 ... 182 168 154]]
```



grey_search.py

grey(image)

- Make sure type(image) == <class 'numpy.dtype'> with dtype == uint8.
- Will read the image color values and set the color to greyscale using the ITU-R Recommendation BT.601 method.
- Will return a greyscale image of the input string file.

earth_finder(grey_image)

- Make sure type(image) == <class 'numpy.dtype'> with dtype == uint8.
- Will find the brightest spot on the picture then return the coordinates of that picture.

earth_image(sharpened_image, original_image)

- Make sure type(sharpened_image, original_image) == <class 'numpy.dtype'> with dtype == uint8 and the images file are in your directory.
- Find the earth location form the earth_finder fuction.
- Will outputs a 101x101 picture with specified location coordinates.

Image_blur(image):

- Make sure type(image) == <class 'numpy.dtype'> with dtype == uint8.
- Will use the gaussian filter to blur the image and return the blurred image.

edge_dectector(blurred_image)

- Make sure type(image) == <class 'numpy.dtype'> with dtype == uint8 .
- The program will then blur your image, then find the partial derivatives of the x,y values, then use the gradient to sharpen your image, then it will combine the two values to get the new sharpened image.
- This will allow your detection program to be more accurate in it's search.

float64_uint8(img)

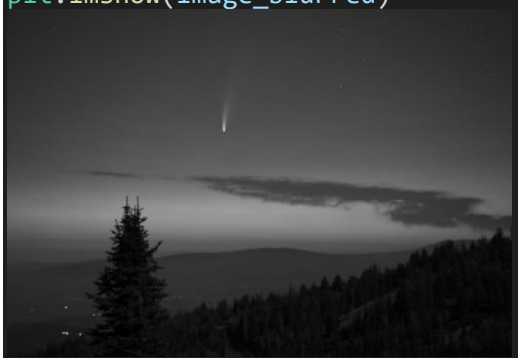
- Make sure type(image) == <class 'numpy.dtype'> with dtype == uint8 .
- Will convert the float64(0-1) to unit8 (0-255).

uint8_float64(img)

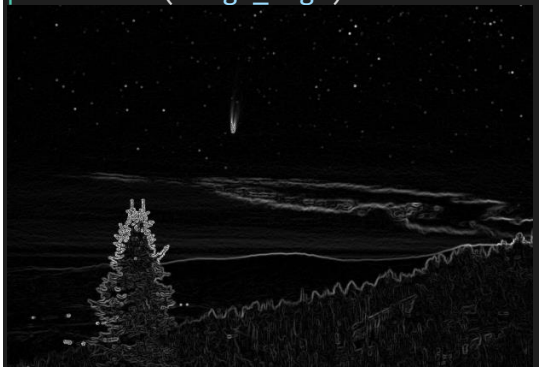
- Make sure type(image) == <class 'numpy.dtype'> with dtype == uint8.
- Will convert the unit8 (0-255) to float64(0-1).

```
import numpy as np
import matplotlib.pyplot as plt
import scipy
from scipy import ndimage
image = plt.imread('img_plain9.tiff')[:,:,:3]
gray = grey(image)
plt.imshow(gray)
```

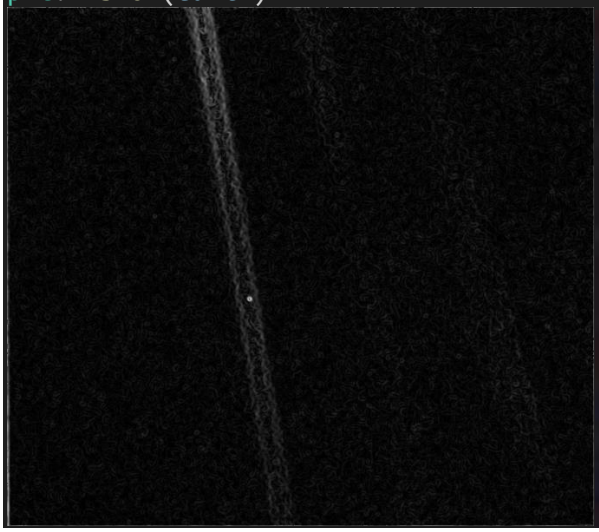
```
image_blurred = image_blur(gray)
plt.imshow(image_blurred)
```

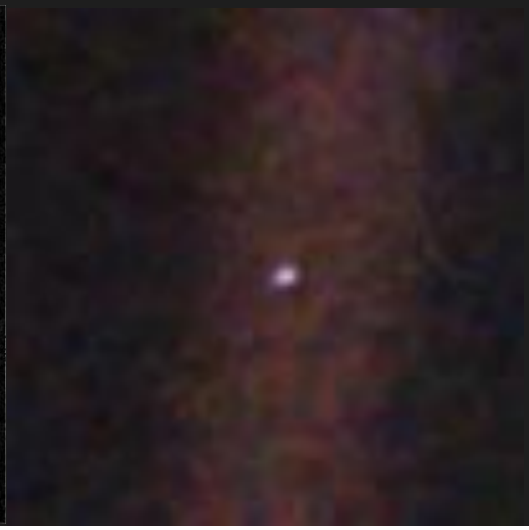


```
image_edge = edge_dectector(image_blurred)
plt.imshow(image_edge)
```



```
image = ia.image_tester('image.tiff')
gray = grey(image)

image_blurred = image_blur(gray)

image_edge = edge_dectector(image_blurred)
plt.imshow(image_edge)



earth = earth_image(image_edge, image)
plt.imshow(earth)
```





Histo_key.py

histogram(image)

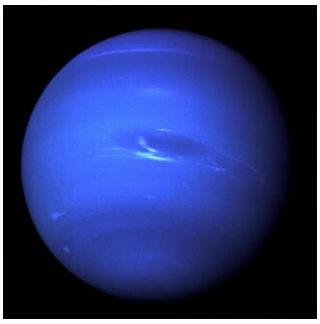
- Make sure type(image) == <class 'numpy.dtype'> with dtype == uint8.
- Will return a histogram of the color values of the inputted image.

pseudo_number_key(image, phrase)

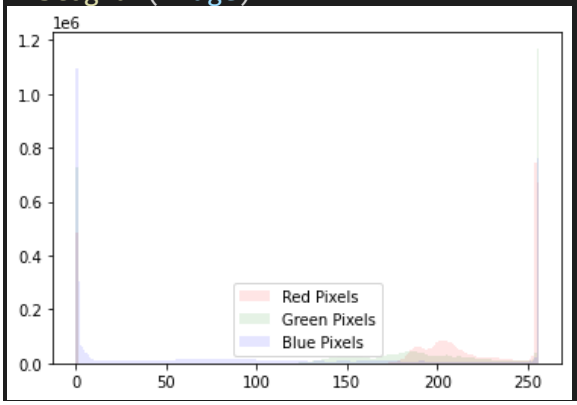
- Make sure type(image) == <class 'numpy.dtype'> with dtype == uint8 and type(phrase) == <class ‘string’>.
- Will return an array of the numbers with the same dimensions as the image, using pseudo-number generator.

encryption_image(image, phrase)

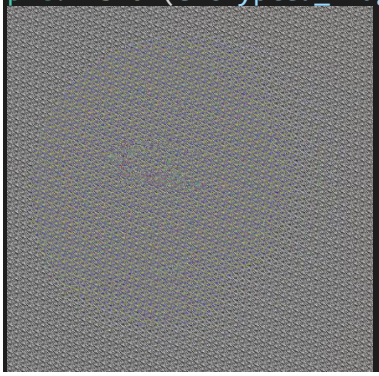
- Does a full encryption/decryption on the phrase and image .



```
image = plt.imread('img_plain3.tiff')[:,:,:3]
histogram(image)
```




```
image = plt.imread('img_plain3.tiff')[:,:,:3]
nKey = pseudo_number_key(image, 'Test')
encrypted_image =  encrytion_image(image, 'Test')

plt.imshow(encrypted_image)
```



main0.py

encryption_test()

- Shows the functionality of our new encryption key.

earth_test()

- Shows the functionality of the image filter, image sharpening, earth search algorithm, and gray scale function.

Oringal_Key_Encryption()

- Shows that the old encryption key works and returns the proper image.

main()

- This will show the functionality of the Team 07 Project 1.

b.

As a team, we worked primarily in overlapping smaller groups of two, and every week we had a group meeting time to discuss the progress and algorithms. The flowcharts were written in the group meetings so everyone could knowledgeably learn about the algorithms. All parts of the project were worked on by two people to make sure that no one would be overwhelmed since team members could help innovate algorithms and debug together. We used GitHub to centralize all our work since all participants could have immediate access to each other's code and created a centralized folder. We used discord to communicate meeting times, but most of the actual communication was in person. For future projects, we will continue to use GitHub as the advantages have been huge.

c.

Our team process involved first finding out what the question was asking us. From there we would create tasks to try and solve the questions. All the tasks would then be delegated to different members to create an algorithm for. We would then start to design the algorithms to answer the questions as a group during the team meetings. After the meetings were over, everyone was expected to meet up with their task partner and code up their algorithms. During these meeting times, the programs would be debugged and analyzed for functionality. Improvements would be discussed after the code was uploaded to GitHub and everyone could test the code. An example of this improvement was when the numpy.bitwise function was implemented because the code was taking too long to run for the bigger images. Overall, our team process involved a lot of delegating tasks and partnership work. This process minimizes the feeling of being overwhelmed on a big task/project since everyone felt they coded more efficiently when there was someone there to brainstorm their ideas with and help them.

d.

```
import matplotlib.pyplot as plt

import scipy

from scipy import ndimage


def img_input():

   #gets file name

   img = input('Enter image file name: ')

   return img
```

```
def image_valid(img):
    if img.endswith('.jpg'):
        image = True
    elif img.endswith('.png'):
        image = True
    elif img.endswith('.tiff'):
        image = True
    else:
        image = False

    return image

def image_tester(img):
    #test the ending of the image file input
    #then reads the file in correctly depending on the ending
    if img.endswith('.jpg'):
        image = plt.imread(img)
    elif img.endswith('.png'):
        image = plt.imread(img)
        image = image.astype(np.uint8)
    elif img.endswith('.tiff'):
        image = plt.imread(img)[:,:,:3]

    return image



def dim(image):
    #shows the dimension
    #image = image_tester(image)
    dim = image.shape
    print(f'Dimensions of the image is {image.shape}')
```

```
    return dim
def key_generator(img, key_str):

    key_str = key_str.replace(' ', '')
    len_key_str = len(key_str)

    #img = ia.image_tester(img) #plt.imread(img)[:,:,:3]
    #gets the dimensions of the image
    row = img.shape[0]
    col = img.shape[1]

    #creates hte new array
    key_array = np.zeros([row, col], dtype=np.uint8)

    #loads the key values into the key array
    for r in range(row):
        for c in range(col):
            key_array[r][c] = ((r*c)%len_key_str)

    Key = key_array*(2**8//(len_key_str))

    return Key
def XOR_Cypher(Img, Key):
    #reads the image
    #Img = ia.image_tester(Img) #plt.imread(Img)[:,:,:3]



    #is a quicker way to encrypt the iage
    for i in range(3):
        #the i is the color channels to encrypt
        #the np.bitwise converts to bit then takes the xor operator by dot properties
        Img[:,:,i] = np.bitwise_xor(Img[:,:,i], Key)
```

```
    return Img


def grey(image):

    #image = ia.image_tester(image)
    #uses matrix multiplication on the rgb values of the photo
    grey_values = (image[:,:,0]*.299 + image[:,:,1]*.587 + image[:,:,2]*.114)
    #this automatically gets the dim from the image, by just coppying
    grey = image.copy()

    #this inserts the grey rgb values in to the grey photo, since the rgb values should all the be same for the same row, col
    for i in range(3):
        grey[:,:,i] = grey_values

    return grey



def earth_finder(grey_image):
    #reads the string
    #grey_image = ia.image_tester(grey_image)

    #makes sure the border exception is taken out
    grey_image = grey_image[1:,1:,:]

    #gets the brighest spot on the image
    brightest_spot = np.amax(grey_image)

    #finds the index where the maximum value is
    index_max = np.where(grey_image == brightest_spot)

    #takes where first items(ie earth row, earth col) of each list return, the turns the tupple to a list
```

```
    cordinates = list(zip(index_max[0], index_max[1]))

    #since all rgb values are the same, we only need the first item of the list
    return cordinates[0]



def earth_image(sharpened_image, orginal_image):
    #gets the location
    location = earth_finder(sharpened_image)

    #it take the location and get the row-50 to row+50 and the same for the column with the color values the same

    #gets the 101x101 image by take the place 0,0 as the location of the earth from the last fuction earth_finder
    #then is takes the 50 pixels, in front, above, behind and below it
    earth = orginal_image[location[0]-50:location[0]+51, location[1]-50:location[1]+51,:]

    return earth

def image_blur(image):

    image = uint8_float64(image)
    #image = ia.image_tester(image)
    #using the guassian filter, blurres the image
    blurred1 = scipy.ndimage.gaussian_filter(image, sigma=2.5)

    return blurred1

def edge_dectector(blurred_image):

    #no need for th absolute value since the both the x and y compnenet gonna be squared
    #gets the gradient of the blurred image in the x-direction
    sx = ndimage.sobel(blurred_image, axis=0, output=None, mode ='constant', cval=0.0)
```

```
    #gets the gradient of the blurred image in the y-direction
    sy = ndimage.sobel(blurred_image, axis=1, output=None, mode ='constant', cval=0.0)
    #hypot get the hypotenuse, or ie sqrt(a^2 + b^2) = C, this is get the gradient edge decetection
    #ie this combine the vertical and horizontal edges
    sob = np.hypot(sx, sy)

    sob = float64_uint8(sob)
    sob = grey(sob)

    return sob

def float64_uint8(img):
    #gets the max number the data could be
    #img_data_max = np.iinfo(img.dtype).max()
    #print(img_data_max)
    #img = img.astype(np.float64)/img_data_max
    #img = ia.image_tester(img)
    #times the image rgb value by 255 to get the unit8
    img = 255*img
    img = img.astype(np.uint8)

    return img



def uint8_float64(img):
    #img_data_max = np.iinfo(img.dtype).max
    #img = ia.image_tester(img)
    #dives the unint8 values to the get the 0-1 float64 values
    img = img/255
    img = img.astype(np.float64)

    return img
```

```
def histogram(image):
    #image = ia.image_tester(image)
    image=(image*255).astype(np.uint8)



    plt.hist(image[:,:,0].reshape(image.shape[0]*image.shape[1]),bins=np.arange(2**8+1), color='red', alpha=.1, label='Red Pixels')
    plt.hist(image[:,:,1].reshape(image.shape[0]*image.shape[1]),bins=np.arange(2**8+1), color='green', alpha=.1, label='Green Pixels')
    plt.hist(image[:,:,2].reshape(image.shape[0]*image.shape[1]),bins=np.arange(2**8+1), color='blue', alpha=.1, label='Blue Pixels')
    plt.legend()
    plt.show()



def pseudo_number_key(image, phrase):
    phrase = phrase.replace(' ', '')
    len_phrase = len(phrase)

    #creates the seed, by the length of the phrase
    seed = len_phrase*12345

    #loads the image dim
    #img = ia.image_tester(image)
    row = image.shape[0]
    col = image.shape[1]

    #Creates a new key array
    key = np.zeros([row,col], dtype=np.uint8)
    #creates the n0, n1, ...nx to make the series work
```

```
    num = seed

    for r in range(row):
        for c in range(col):
            num = ((1103515245*num) + seed)%(2**31)
            #loads the seed values into the array
            key[r][c] = num

    return key

def encryption_test():
    image = input('Enter your image: ')
    if image_valid(image) == False:
        raise ValueError('This is not a valid image, please use a .tiff, .jpg, .png')

    out_image = input('Enter your output image file (as a .tiff): ')
    if image_valid(out_image) == False:
        raise ValueError('This is not a valid image, please use a .tiff, .jpg, .png')

    phrase = input('Enter your phrase: ')

    image = image_tester(image)
    histogram(image)
    key = pseudo_number_key(image, phrase)
    out_image = XOR_Cypher(image, key)
    plt.imsave(out_image, out_image)
    histogram(out_image)


def earth_test():
    image = image_tester('image.tiff')
    gray = grey(image)
```

```
    plt.imsave('grey.tiff', gray)

    image_blurred = image_blur(gray)
    plt.imsave('blur.tiff', image_blurred)


    image_edge = edge_dectector(image_blurred)
    plt.imsave('sharp.tiff', image_edge)


    earth = earth_image(image_edge, image)
    plt.imsave('earth.tiff', earth)

def Oringal_Key_Encryption():
    phrase = 'COME AND GET YOUR LOVE'
    img = 'Pale_Blue_Dot_Encrypted.tiff'
    img = image_tester(img)


    key = key_generator(img, phrase)
    pic = XOR_Cypher(img, key)
    plt.imsave("image.tiff", pic)

def main():
    Oringal_Key_Encryption()
    earth_test()
    encryption_test()


if __name__ == '__main__':
    main()
```

Code Appendix