Steganography Exercise 1

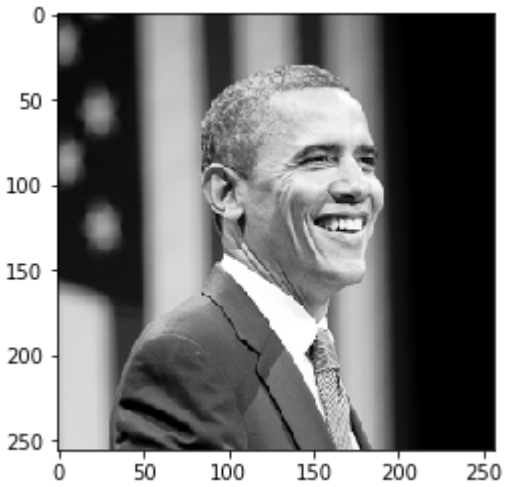
Tientso Ning

In [1]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import numpy.matlib
from skimage.util import random_noise
%matplotlib inline
```

In [2]:
```
#load the cover image as grayscale
img = cv2.imread("./obama.PNG", 0) #thanks obama
rows, col = img.shape
print(rows,col)
plt.imshow(img, cmap="Greys_r")
```

```
256 256
```

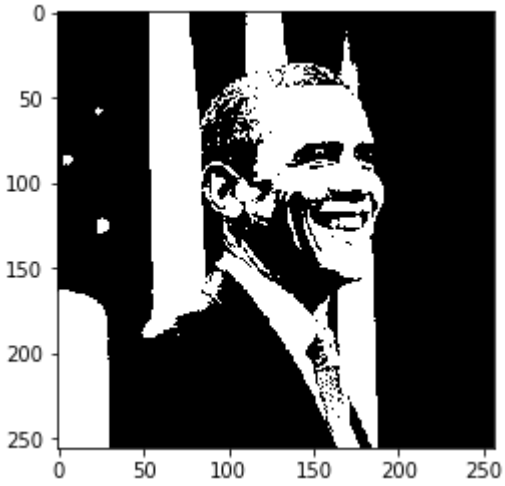
Out[2]:
```
<matplotlib.image.AxesImage at 0x7fc8aeb5aac8>
```



In [3]:
```
#separate bitplane
planes = np.zeros((8, rows, col))

for i in range(0, rows):
    for j in range(0, col):
        binaryPixel = np.binary_repr(img[i,j],8) #changes to binary rep

        k=0
        for b in range(len(binaryPixel)-1,-1,-1):
            planes[8-k-1,i,j] = int(binaryPixel[b],2) #changes to int, but sam
e as double
            k = k+1
```

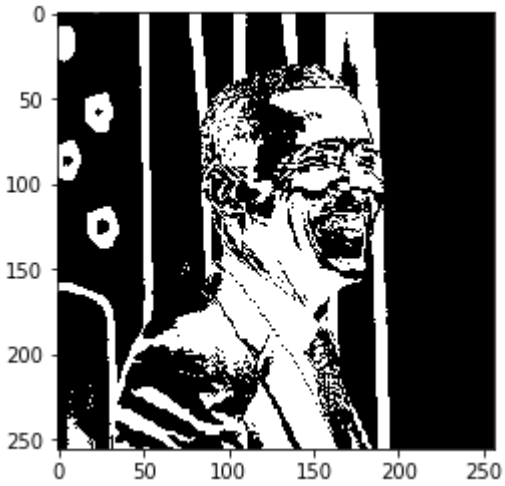
In [4]:
```
#display bitplane separately
plt.imshow(planes[0,:,:], cmap="Greys_r")
```

Out[4]: <matplotlib.image.AxesImage at 0x7fc8ae625cc0>



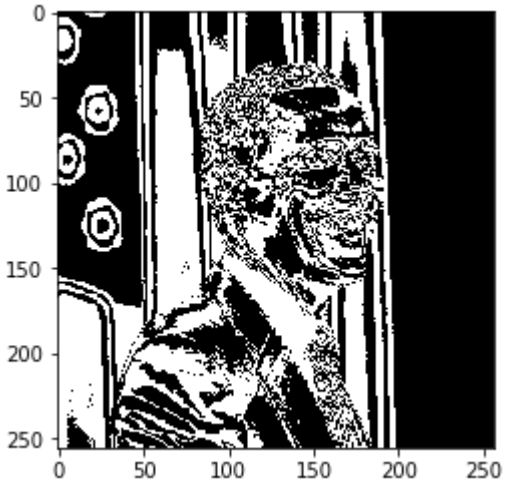
In [5]:
```
plt.imshow(planes[1,:,:], cmap="Greys_r")
```

Out[5]: <matplotlib.image.AxesImage at 0x7fc8ac959358>



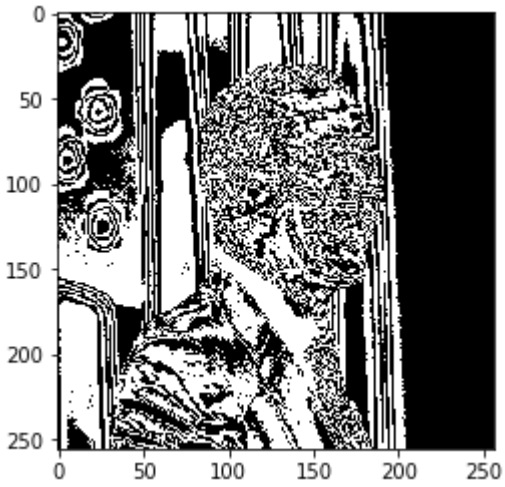
In [6]:
```
plt.imshow(planes[2,:,:], cmap="Greys_r")
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x7fc8ac9329b0>
```



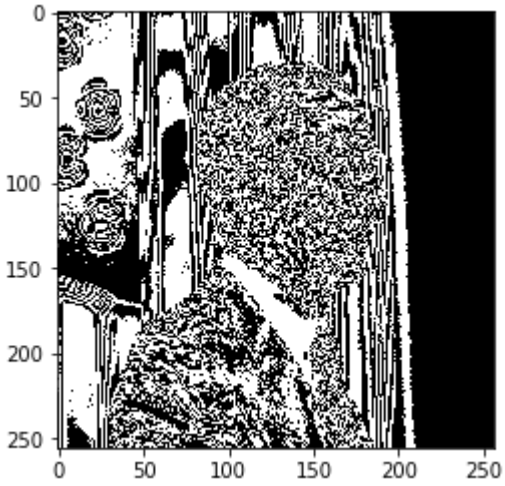
In [7]:
```
plt.imshow(planes[3,:,:], cmap="Greys_r")
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x7fc8ac890cc0>
```



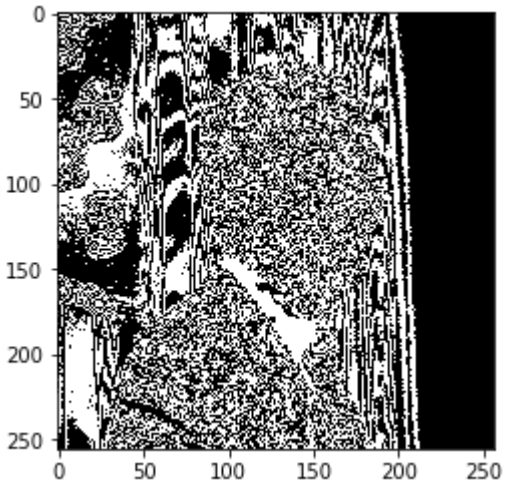
In [8]:
```
plt.imshow(planes[4,:,:], cmap="Greys_r")
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x7fc8ac870ef0>
```



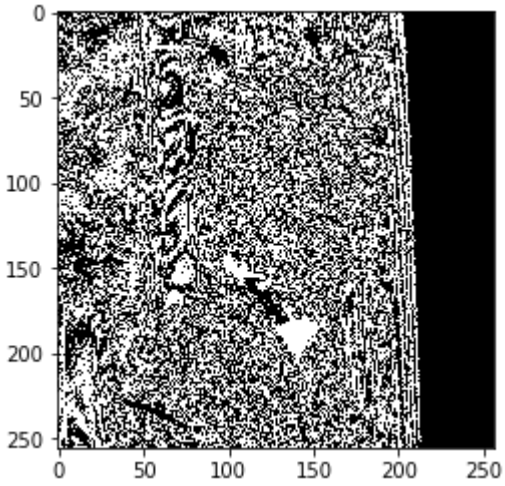
In [9]:
```
plt.imshow(planes[5,:,:], cmap="Greys_r")
```

Out[9]:
```
<matplotlib.image.AxesImage at 0x7fc8ac7db278>
```



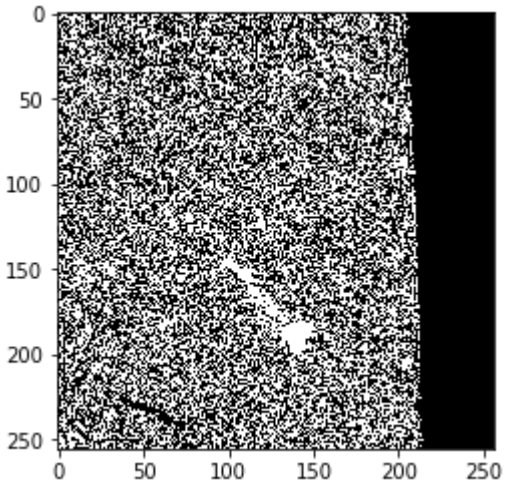
In [10]:
```
plt.imshow(planes[6,:,:], cmap="Greys_r")
```

Out[10]:
```
<matplotlib.image.AxesImage at 0x7fc8ac7b6668>
```



In [11]:
```
plt.imshow(planes[7,:,:], cmap="Greys_r")
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x7fc8ac715be0>
```

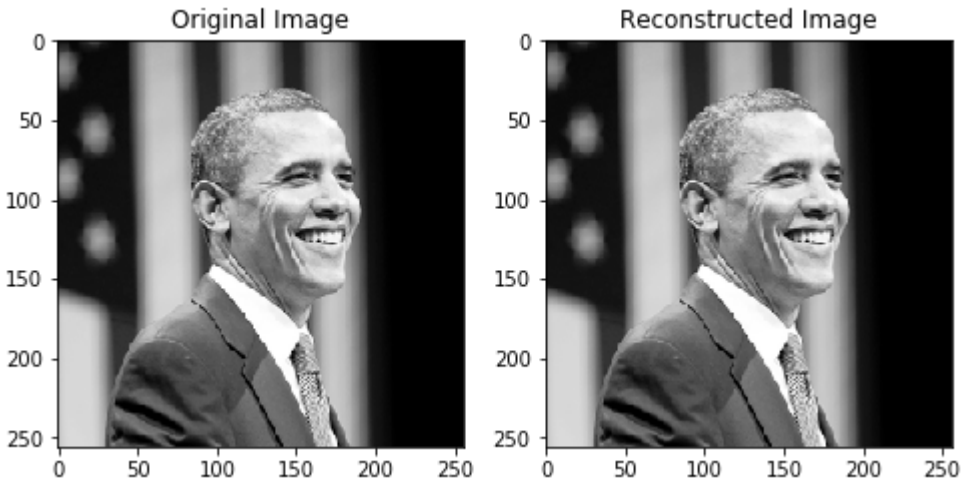


These planes are the value of the bits going from most significant (layer 1) to least significant (layer 8), and it makes sense that the top layers have most of the "features" of the image, while the last layer, the least significant bits (the ones that contribute the least information) are mostly noise.

In [12]:
```
#reconstruct the input image from planes
img_recon = np.zeros((rows,col))
for i in range(0, rows):
    for j in range(0, col):

        #select all bits of current pixel
        binaryPixel = planes[:,i,j]

        #order from LSB to MSB, but in python the endian is different so we do
n't flip
        #binaryPixel = np.flip(binaryPixel)
        binaryPixel = binaryPixel.astype(int)

        #convert the array to decimal
        img_recon[i,j] = np.packbits(binaryPixel)
```

In [13]:
```
f, ax = plt.subplots(1,2, figsize=(8,12))
ax[0].imshow(img, cmap="Greys_r")
ax[0].set_title("Original Image")
ax[1].imshow(img_recon, cmap="Greys_r")
ax[1].set_title("Reconstructed Image")
print("Thanks Obama")
```

```
Thanks Obama
```

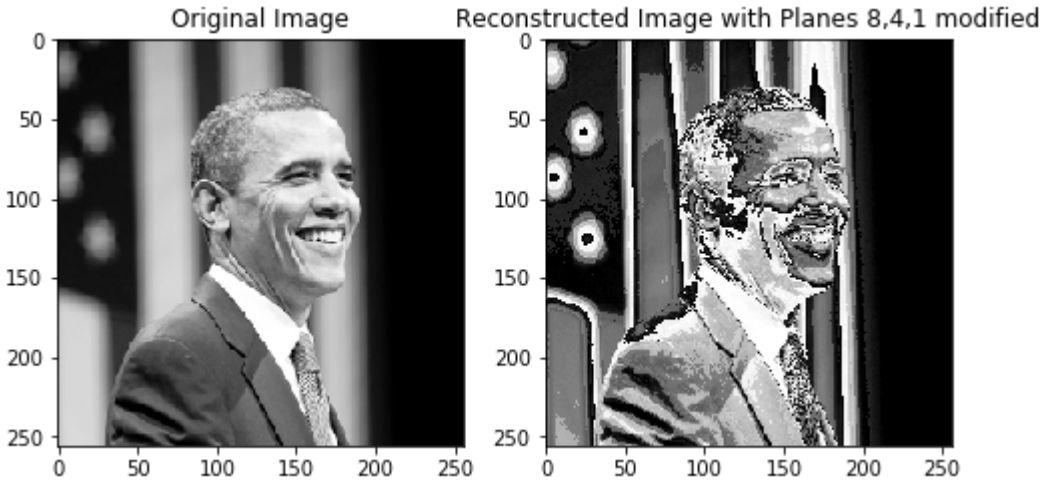


As we can see, the reconstructed image from the layers yields the same exact image as the original, nothing was altered.

In [14]:
```
#manipulate the subplanes 8, 4, 1

to_change = [7,3,0] #account for matlab differences

for p in range(0,3):
    planes[to_change[p],:,:] = np.zeros((rows,col))

    #reconstruct
    img_recon = np.zeros((rows,col))

    for i in range(0, rows):
        for j in range(0, col):

            #select all bits of current pixel
            binaryPixel = planes[:,i,j]

            #order from LSB to MSB, but in python the endian is different so w
e don't flip
            #binaryPixel = np.flip(binaryPixel)
            binaryPixel = binaryPixel.astype(int)

            #convert the array to decimal
            img_recon[i,j] = np.packbits(binaryPixel)
```

In [15]:
```
#again check the recons
f, ax = plt.subplots(1,2, figsize=(8,12))
ax[0].imshow(img, cmap="Greys_r")
ax[0].set_title("Original Image")
ax[1].imshow(img_recon, cmap="Greys_r")
ax[1].set_title("Reconstructed Image with Planes 8,4,1 modified")
print("Thanks Obama")
```

```
Thanks Obama
```



Here, messing with the planes 8, 4, and 1 will yield a significant change in the image since we messed with the top layer, the most significant bit, and that is mostly where we are seeing the loss in the image's representation. However, we can still mostly tell the two images are the same image.

Exercise 2 - Embed a message

In [16]:
```
#embed message into the LSB plane

#separate the bit planes
planes = np.zeros((8, rows, col))

for i in range(0, rows):
    for j in range(0, col):
        binaryPixel = np.binary_repr(img[i,j],8) #changes to binary rep

        k=0
        for b in range(len(binaryPixel)-1,-1,-1):
            planes[8-k-1,i,j] = int(binaryPixel[b],2) #changes to int, but sam
e as double
            k = k+1

LSBplane = np.copy(planes[7,:,:]) #should be the least sigbit plane in python
```

In [17]:
```
#create the message
txt = np.matlib.repmat(b'Help me ', 1,100)
int_txt = np.frombuffer(txt, dtype=np.uint8)
bin_txt = np.unpackbits(int_txt)
bin_txt = np.reshape(bin_txt, (800,8))
L = len(bin_txt)
```

In [18]:
```
#generate L random positions i,j
glowup = LSBplane.shape
embed_pos = np.random.permutation(glowup[0]*glowup[1])
```

In [19]:
```
#select L random positions (i,j) inside the plane
LSBplane = LSBplane.flatten()

for i in range(0, L):
    LSBplane[embed_pos[i]] = np.packbits(bin_txt[i,:])[0]
LSBplane = np.reshape(LSBplane, glowup)

#re-insert the plane
planes[7,:,:] = np.copy(LSBplane)

#reconstruct
img_recon_mod = np.zeros((rows,col))

for i in range(0, rows):
    for j in range(0, col):

        #select all bits of current pixel
        binaryPixel = planes[:,i,j]

        #order from LSB to MSB, but in python the endian is different so we do
n't flip
        #binaryPixel = np.flip(binaryPixel)
        binaryPixel = binaryPixel.astype(int)

        #convert the array to decimal
        img_recon_mod[i,j] = np.packbits(binaryPixel)
```

In [20]:
```
f, ax = plt.subplots(1,3, figsize=(12,12))
ax[0].imshow(img, cmap="Greys_r")
ax[1].imshow(img_recon_mod, cmap="Greys_r")
ax[0].set_title("Obama with a charming smile")
ax[1].set_title("Obama with a secret message")
ax[2].imshow(img-img_recon_mod, cmap="Greys_r")
ax[2].set_title("Subtraction of two images")
```

Out[20]:
```
Text(0.5, 1.0, 'Subtraction of two images')
```

Here we can see that the image has perceptual invisibility, meaning that the human eye cannot really tell that the image contains information hidden. The subtraction of the two images shows that there is indeed a secret message embedded into the second image.

In [21]:
```
img_recon_mod = img_recon_mod.astype(int)

#separate the bit planes
planes_mod = np.zeros((8, rows, col))

for i in range(0, rows):
    for j in range(0, col):
        binaryPixel = np.binary_repr(img_recon_mod[i,j],8) #changes to binary
 rep

        k=0
        for b in range(len(binaryPixel)-1,-1,-1):
            planes_mod[8-k-1,i,j] = int(binaryPixel[b],2) #changes to int, but
same as double
            k = k+1

LSBplane_mod = np.copy(planes_mod[7,:,:]) #should be the least sigbit plane in
python
```

In [22]:
```
recovered_msg = np.zeros(L)
for i in range(0, L):
    recovered_msg[i] = LSBplane.flatten()[embed_pos[i]]
```

In [23]:
```
#errors
calcd_err = recovered_msg - np.packbits(bin_txt)
np.sum(calcd_err)
```

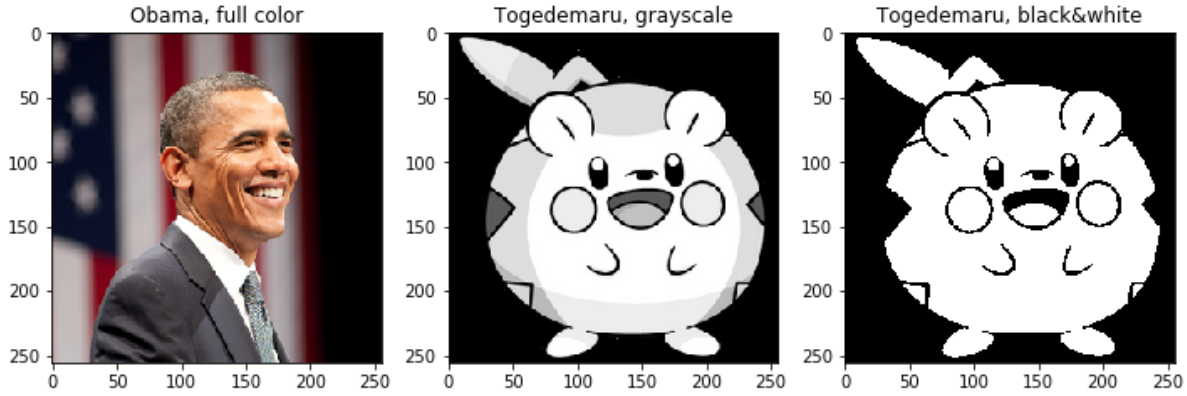
Out[23]:
```
0.0
```

As we can see, the recovered message and the original message (before we embedded it) are exactly the same! Meaning that all our processes are correct. We take the stego image and separate it to its respective planes (as we did before) and then with the permutation matrix (which we can think of as a key, since it is the "map" that shows where we placed the message information) we can identify and extract the information.

Task 3 - Color Planes

In [24]:
```
#read obama in full color, togedemaru in grayscale
img = cv2.imread("./obama.PNG", 1)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #BGR vs RGB representation fix
secret_img = cv2.imread("./togedemaru.PNG", 0)
thresh, bw_secret = cv2.threshold(secret_img, 127, 255, cv2.THRESH_BINARY)

f, ax = plt.subplots(1,3, figsize=(12,12))
ax[0].imshow(img)
ax[0].set_title("Obama, full color") #thank you obama
ax[1].imshow(secret_img, cmap="Greys_r")
ax[1].set_title("Togedemaru, grayscale")
ax[2].imshow(bw_secret, cmap="Greys_r")
ax[2].set_title("Togedemaru, black&white")
```

Out[24]:
```
Text(0.5, 1.0, 'Togedemaru, black&white')
```



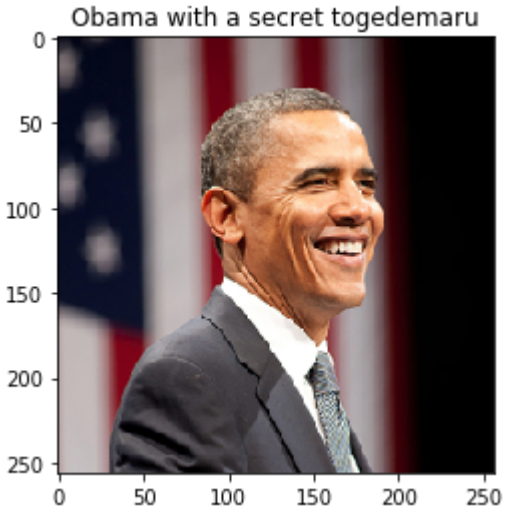
In [25]:
```
#put the colors in the right places
red_p = img[:,:,0]
green_p = img[:,:,1]
blue_p = img[:,:,2]

#green channel sizing
rows, col = green_p.shape[0], green_p.shape[1]
```

In [26]:
```
#we want to embed into the LSB of the green plane
planes = np.zeros((rows,col,8))
for i in range(0, rows):
    for j in range(0, col):
        binaryPixel = np.binary_repr(green_p[i,j],8) #changes to binary rep

        k=0
        for b in range(len(binaryPixel)-1,-1,-1):
            planes[i,j,8-k-1] = int(binaryPixel[b],2) #changes to int, but sam
e as double
            k = k+1

planes[:,:,7] = bw_secret #for python the endian should make the 8th plane the
LSB
```

In [27]:
```
#reconstruct
task3_recon = np.zeros((rows,col))

for i in range(0, rows):
    for j in range(0, col):

        #select all bits of current pixel
        binaryPixel = planes[i,j,:]

        #order from LSB to MSB, but in python the endian is different so we do
n't flip
        #binaryPixel = np.flip(binaryPixel)
        binaryPixel = binaryPixel.astype(int)

        #convert the array to decimal
        task3_recon[i,j] = np.packbits(binaryPixel)
```

In [28]:
```
task3_stego = np.stack((red_p, task3_recon.astype(int), blue_p), axis=-1)
plt.imshow(task3_stego) #obama with secret togedemaru
plt.title("Obama with a secret togedemaru")
print("Thank you Obama, very cool")
```

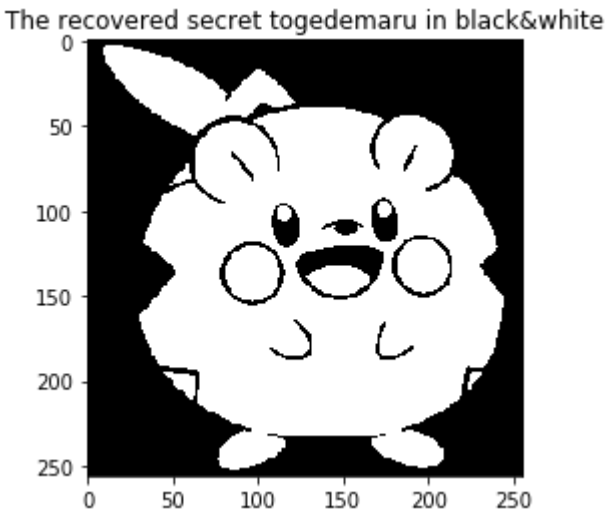
```
Thank you Obama, very cool
```



In [29]:
```
#recover the secret image
r,g,b = cv2.split(task3_stego)

#split green plane to get LSB plane
planes = np.zeros((rows,col,8))
for i in range(0, rows):
    for j in range(0, col):
        binaryPixel = np.binary_repr(g[i,j],8) #changes to binary rep

        k=0
        for b in range(len(binaryPixel)-1,-1,-1):
            planes[i,j,8-k-1] = int(binaryPixel[b],2) #changes to int, but sam
e as double
            k = k+1

LSBplane = planes[:,:,7] #LSB in python
```

In [30]:
```
task3_msg = np.copy(LSBplane)
plt.imshow(LSBplane, cmap="Greys_r")
plt.title("The recovered secret togedemaru in black&white")
```

Out[30]:
```
Text(0.5, 1.0, 'The recovered secret togedemaru in black&white')
```



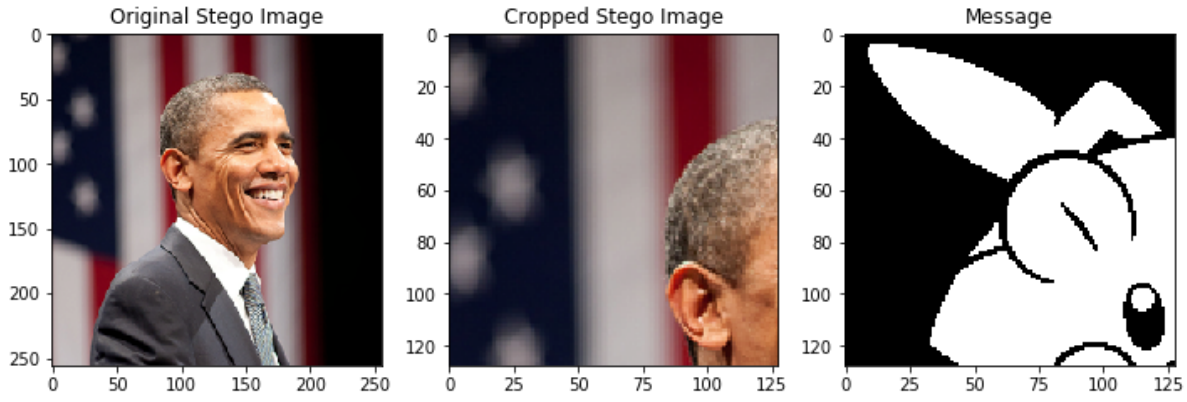
In [31]:
```
#calculate the errors
calcd_err = 255*LSBplane - bw_secret
print(np.sum(calcd_err))
```

```
0.0
```

Here we can see that we recovered the original secret image (togedemaru in black&white) that we hidden inside the cover image (obama). We also can tell this is exactly the same image since the difference between the recovered image from the LSB plane is not different from the original secret image, as we have a calculated error of zero.

In [32]:
```
task4 = np.copy(task3_stego) #we want the image with the hidden data
task4_cropped = task4[0:128,0:128,:]

#recover the secret image
r,g,b = cv2.split(task4_cropped)

#split green plane to get LSB plane
planes = np.zeros((task4_cropped.shape[0],task4_cropped.shape[1],8))
for i in range(0, task4_cropped.shape[0]):
    for j in range(0, task4_cropped.shape[1]):
        binaryPixel = np.binary_repr(g[i,j],8) #changes to binary rep

        k=0
        for b in range(len(binaryPixel)-1,-1,-1):
            planes[i,j,8-k-1] = int(binaryPixel[b],2) #changes to int, but sam
e as double
            k = k+1

LSBplane = planes[:,:,7] #LSB in python

f, ax = plt.subplots(1,3, figsize=(12,12))
ax[0].imshow(task4)
ax[0].set_title("Original Stego Image")
ax[1].imshow(task4_cropped)
ax[1].set_title("Cropped Stego Image")
ax[2].imshow(LSBplane, cmap="Greys_r")
ax[2].set_title("Message")
```

Out[32]:
```
Text(0.5, 1.0, 'Message')
```



As we can see, the message survives because we have only cropped the image, without scrambling any of the data underneath

In [33]:
```
#add noise to the image
task4_noised = random_noise(np.zeros((256,256,3)),mode="s&p", amount=0.3)
task4_noised = task4_noised.astype(int)+np.copy(task4)
```

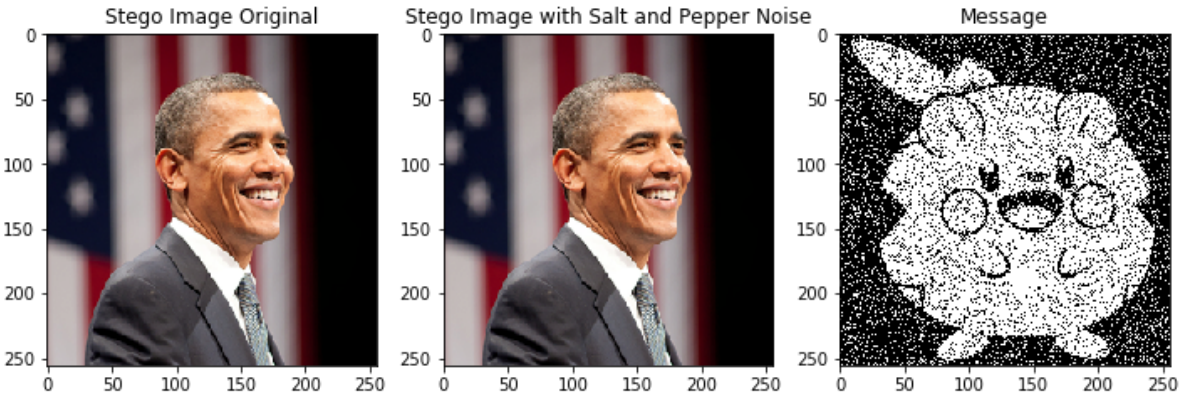
In [34]:
```
#recover the secret image
r,g,b = cv2.split(task4_noised)

#split green plane to get LSB plane
planes = np.zeros((task4_noised.shape[0],task4_noised.shape[1],8))
for i in range(0, task4_noised.shape[0]):
    for j in range(0, task4_noised.shape[1]):
        binaryPixel = np.binary_repr(g[i,j],8) #changes to binary rep

        k=0
        for b in range(len(binaryPixel)-1,-1,-1):
            planes[i,j,8-k-1] = int(binaryPixel[b],2) #changes to int, but sam
e as double
            k = k+1

LSBplane = planes[:,:,7] #LSB in python
```

```
/home/kense/.local/lib/python3.6/site-packages/ipykernel_launcher.py:8: Depre
cationWarning: Insufficient bit width provided. This behavior will raise an e
rror in the future.
```

In [35]:
```
f, ax = plt.subplots(1,3, figsize=(12,12))
ax[0].imshow(task4)
ax[0].set_title("Stego Image Original")
ax[1].imshow(task4_noised)
ax[1].set_title("Stego Image with Salt and Pepper Noise")
ax[2].imshow(LSBplane, cmap="Greys_r")
ax[2].set_title("Message")
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```

Out[35]:
```
Text(0.5, 1.0, 'Message')
```



As we can see, the perceptual visibility of the image is not damaged, but the message itself is impacted by the salt and pepper noise. Although we can still tell what the message is, with additional noise, we might not be able to!

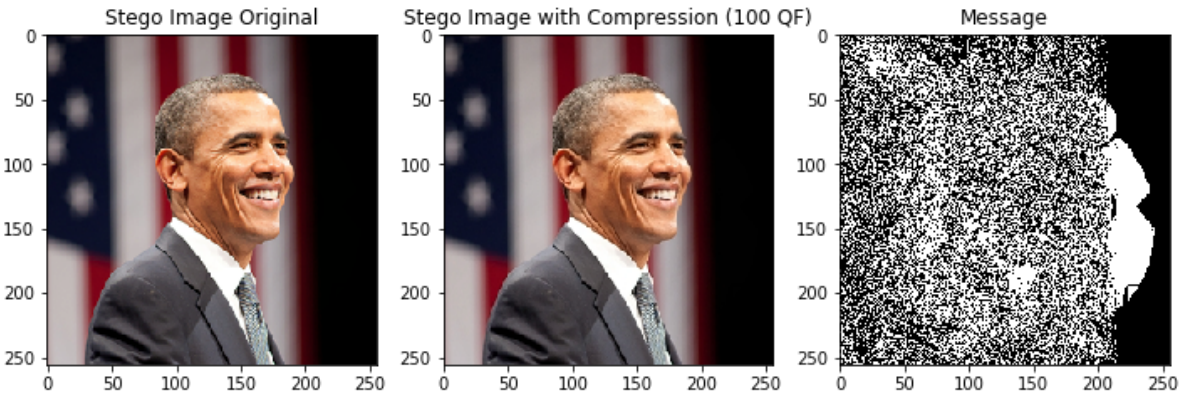
In [36]:
```
#JPEG Compression with Max Quality
cv2.imwrite("./task4_compressed.JPG", task4, [int(cv2.IMWRITE_JPEG_QUALITY), 1
00]) #100 QF compression
```

Out[36]: True

In [37]:
```
task4_compressed = cv2.imread("./task4_compressed.JPG")
#recover the secret image
r,g,b = cv2.split(task4_compressed)

#split green plane to get LSB plane
planes = np.zeros((task4_compressed.shape[0],task4_compressed.shape[1],8))
for i in range(0, task4_compressed.shape[0]):
    for j in range(0, task4_compressed.shape[1]):
        binaryPixel = np.binary_repr(g[i,j],8) #changes to binary rep

        k=0
        for b in range(len(binaryPixel)-1,-1,-1):
            planes[i,j,8-k-1] = int(binaryPixel[b],2) #changes to int, but sam
e as double
            k = k+1

LSBplane = planes[:,:,7] #LSB in python
```

In [38]:
```
f, ax = plt.subplots(1,3, figsize=(12,12))
ax[0].imshow(task4)
ax[0].set_title("Stego Image Original")
ax[1].imshow(task4_compressed)
ax[1].set_title("Stego Image with Compression (100 QF)")
ax[2].imshow(LSBplane, cmap="Greys_r")
ax[2].set_title("Message")
```

Out[38]: Text(0.5, 1.0, 'Message')

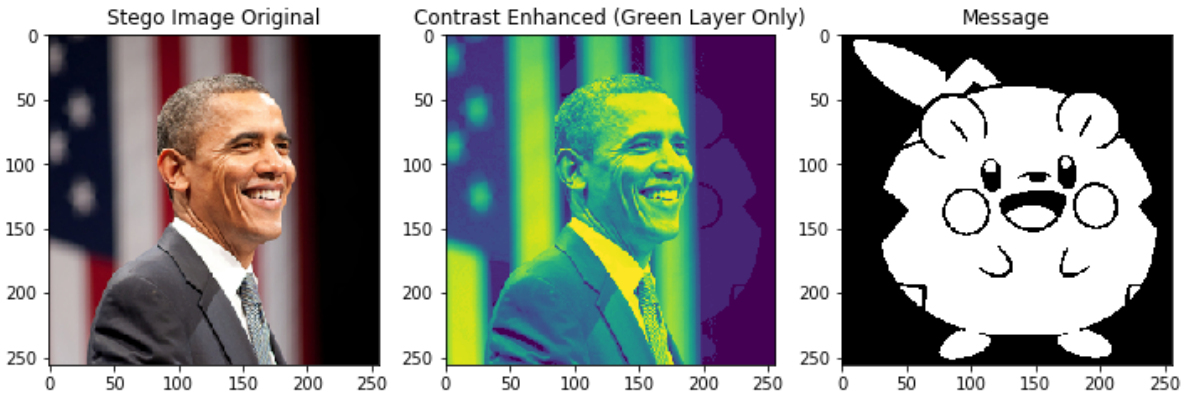


As we can see, the perceptual visibility of the stego image is uncompromised, yet our message was completely destroyed even with a QF of 100 in compression. There is no way to tell what the message is, unlike the previous example where we could at least still see through the noise. This shows that the information hidden in the LSB layer of the images are sensitive to compressions and damage.

In [39]:
```
#histogram equalization to enhance contrast of the green channel
r,g,b = cv2.split(task4)
g = np.uint8(cv2.normalize(g, None, 0, 255, cv2.NORM_MINMAX))
g_hist = cv2.equalizeHist(g)
```

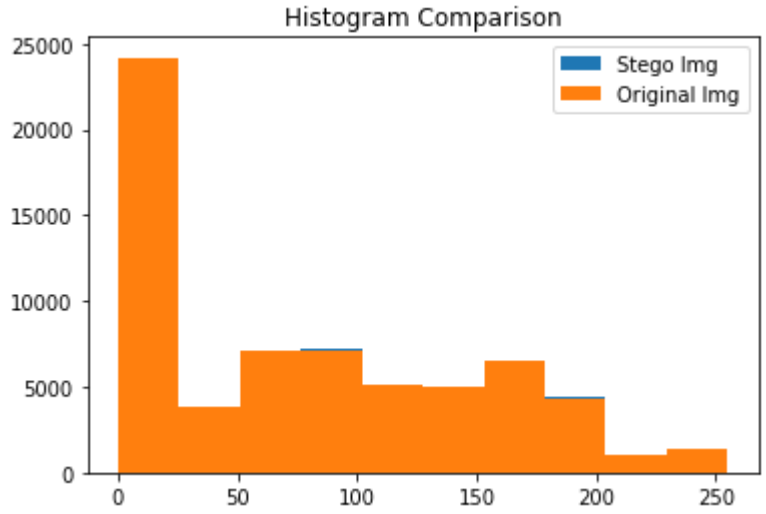
In [40]:
```
f, ax = plt.subplots(1,3, figsize=(12,12))
ax[0].imshow(task4)
ax[0].set_title("Stego Image Original")
ax[1].imshow(g_hist)
ax[1].set_title("Contrast Enhanced (Green Layer Only)")
ax[2].imshow(task3_msg, cmap="Greys_r")
ax[2].set_title("Message")
```

Out[40]:
```
Text(0.5, 1.0, 'Message')
```



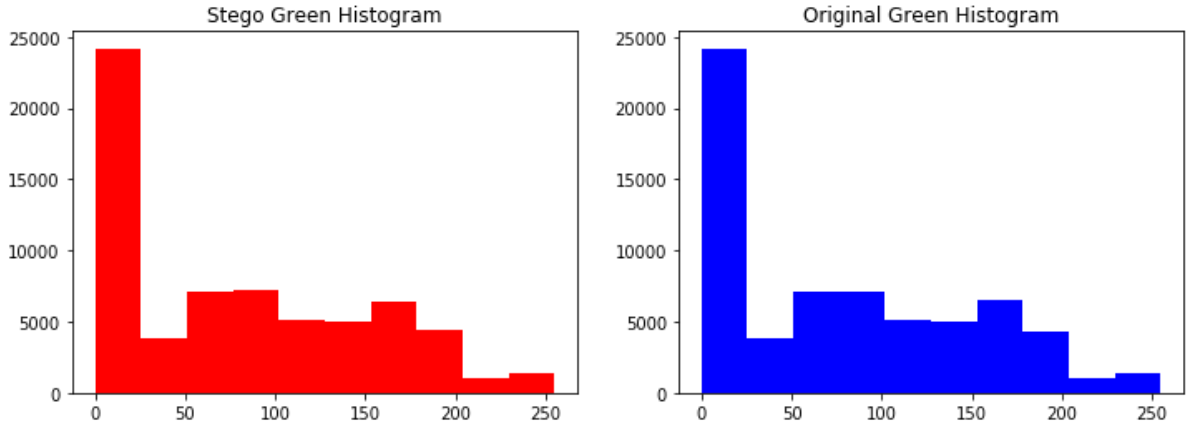
Here, we can see that the message is partly visible when we enhance the contrast, meaning that this hiding technique could easily be detected using simple analysis.

In [41]:
```
#more steganalysis
_, g2, _ = cv2.split(img)
plt.hist(g.flatten(), label="Stego Img")
plt.hist(g2.flatten(), label="Original Img")
plt.title("Histogram Comparison")
plt.legend()
plt.show()
```



In [42]:
```
f, ax = plt.subplots(1,2, figsize=(12,4))
ax[0].hist(g.flatten(), color="Red")
ax[0].set_title("Stego Green Histogram")
ax[1].hist(g2.flatten(), color="Blue")
ax[1].set_title("Original Green Histogram")
```

Out[42]:
```
Text(0.5, 1.0, 'Original Green Histogram')
```



Here we have a very very slight difference, which I believe is because of my image choice for the second image. We expect a bit more of a difference in the two channels, but the fact that there is even a difference (see the comparison image, there is slightly blue on top of the orange) tells us that if the hidden message image was perhaps different, that we could see better results.