ECED 4404: Computer Networks & Communication – Design Project

Final Report – Winter 2019

Mark Hooper - B00777009

Burak Ozter - B00784243

Josh Caume – B00723791

Abstract:

The following document outlines an overview into the process and design choices made while implementing a message encryption process within two images. This uses diffusion and confusion in order to mask the message from unwanted eyes. A comparison of projected and completed outcomes is included along with our validation/results. In the end, we were successful in the implementation of our desired objective with the absence of a small feature.

Technical Description:

In this project we implemented image encrypted messaging and decryption process for this project. We implemented a Public/Private image like RSA but with a different mathematical implementation.

The ASCII to image encryption was accomplished by storing the ASCII value in an X, Y coordinate of the image. The ascii value is stored in one of the three channels Red, Green, or Blue while the other two channels correspond to the next coordinate of the message. We opted for cycling what channel is used to store the ASCII value to add a layer of confusion. See Figure 1, Figure 2, Figure 3.

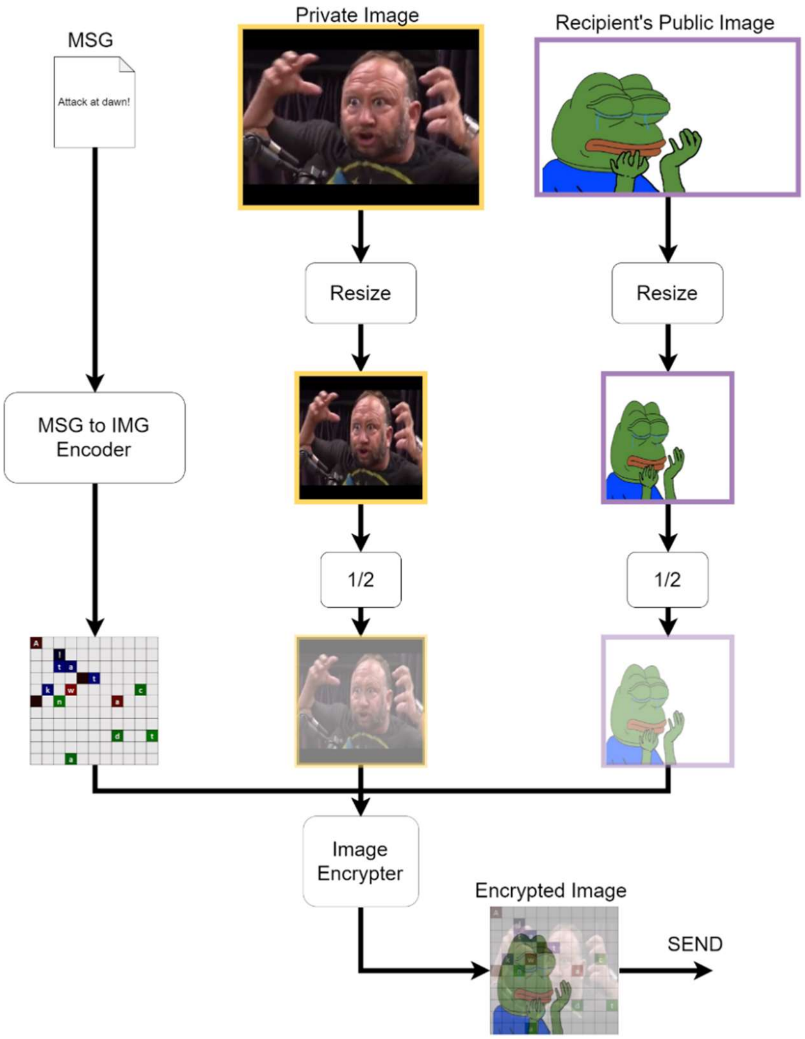


Figure 1: Image Encryption

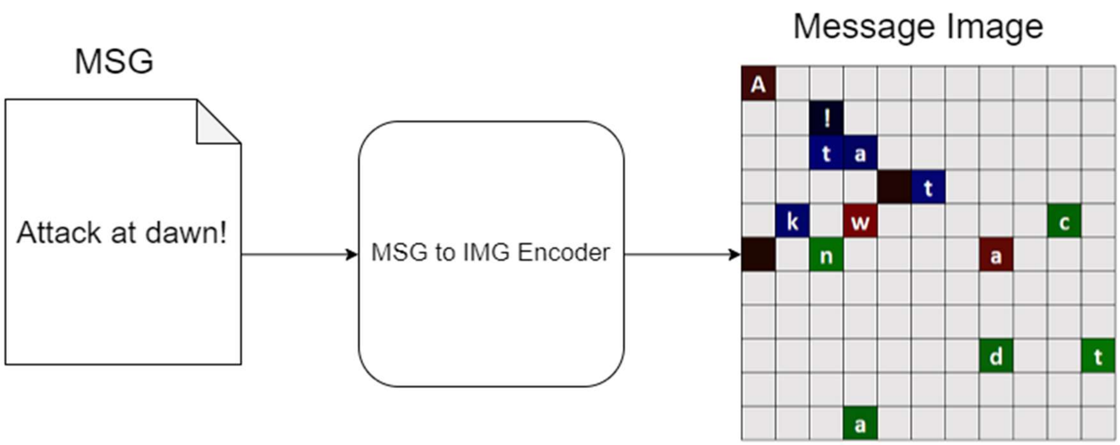
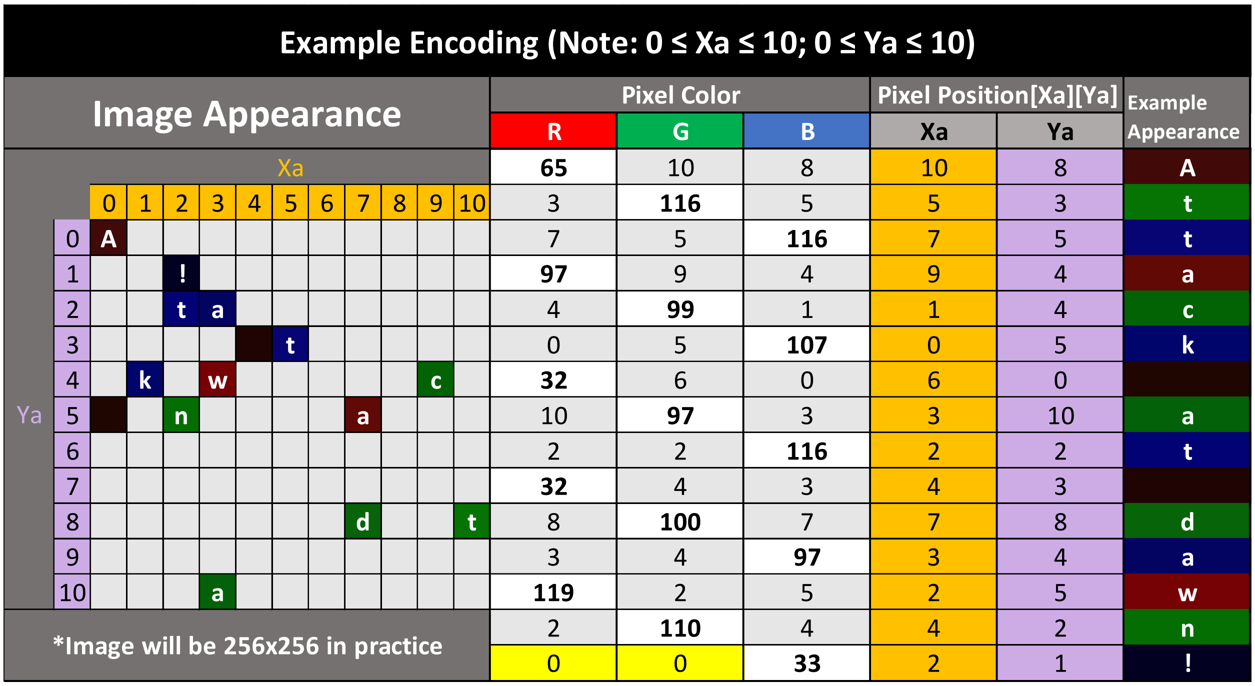


Figure 2: Message to Image Encoder



Example Encoding (Note: 0 ≤ Xa ≤ 10; 0 ≤ Ya ≤ 10)

Image Appearance

Ya \ Xa	0	1	2	3	4	5	6	7	8	9	10
0	A										
1			!								
2			t	a							
3					■	t					
4		k		w						c	
5	■		n					a			
6											
7											
8								d			t
9											
10				a							

*Image will be 256x256 in practice

Pixel Color			Pixel Position[Xa][Ya]		Example Appearance
R	G	B	Xa	Ya	
65	10	8	10	8	A
3	**116**	5	5	3	t
7	5	**116**	7	5	t
97	9	4	9	4	a
4	**99**	1	1	4	c
0	5	**107**	0	5	k
32	6	0	6	0	■
10	**97**	3	3	10	a
2	2	**116**	2	2	t
32	4	3	4	3	■
8	**100**	7	7	8	d
3	4	**97**	3	4	a
119	2	5	2	5	w
2	**110**	4	4	2	n
0	0	**33**	2	1	!

Figure 3: Encoding Example

Due to the limitations of the channel sizes any of our pictures will require a resize down from their original resolution to an array of 256x256 (0-255). We were required to limit the array to 0-254 due to the loss in data caused by the halving operation seen in Figure 1. Python Image Library (PIL/PILLOW) offered a simple function for resizing the pictures to any desired size. [1]

This encryption method is secure through obscurity but could also be combined with existing standards or additional rounds to add additional layers of security. The decryption process operates like the flow chart depicted in Figure 4.

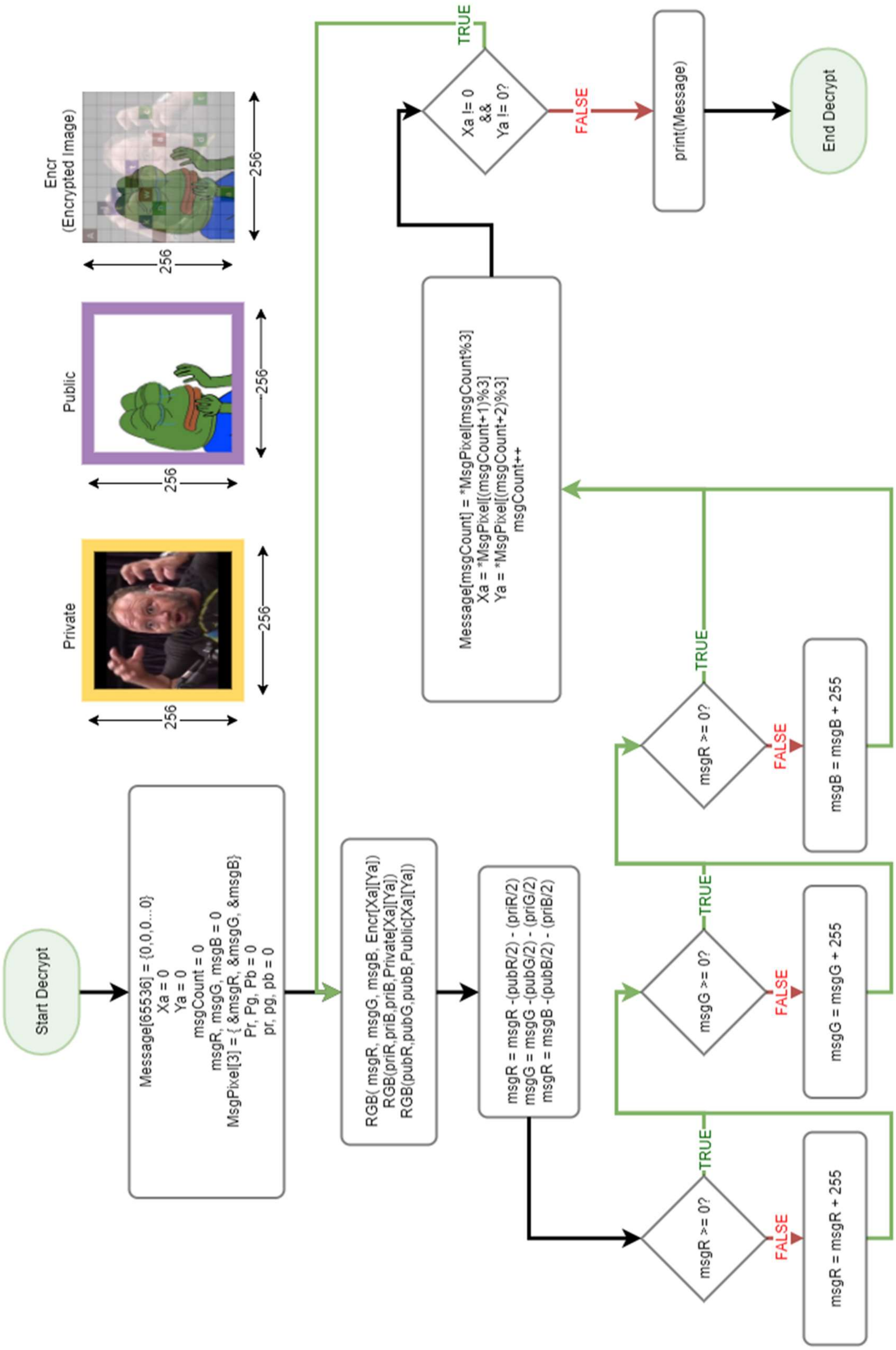

Private
256
256
Public
256
256
Encr
(Encrypted Image)
256
256
Start Decrypt
Message[65536] = {0,0,0...0}
Xa = 0
Ya = 0
msgCount = 0
msgR, msgG, msgB = 0
MsgPixel[3] = { &msgR, &msgG, &msgB}
Pr, Pg, Pb = 0
pr, pg, pb = 0
RGB(msgR, msgG, msgB, Encr[Xa][Ya])
RGB(priR,priB,priB,Private[Xa][Ya])
RGB(pubR,pubG,pubB,Public[Xa][Ya])
msgR = msgR -(pubR/2) - (priR/2)
msgG = msgG -(pubG/2) - (priG/2)
msgR = msgB -(pubB/2) - (priB/2)
msgR >= 0?
TRUE
FALSE
msgR = msgR + 255
msgG >= 0?
TRUE
FALSE
msgG = msgG + 255
msgR >= 0?
TRUE
FALSE
msgB = msgB + 255
Message[msgCount] = *MsgPixel[msgCount%3]
Xa = *MsgPixel[(msgCount+1)%3]
Ya = *MsgPixel[(msgCount+2)%3]
msgCount++
Xa != 0
&&
Ya != 0?
TRUE
FALSE
print(Message)
End Decrypt

Figure 4: Decryption Algorithm

Comparison to Relevant Material:

Steganography is the practice of concealing a file, message, image, or video within another file, message, image, or video. The word steganography combines the Greek words steganos, meaning "covered, concealed, or protected", and graphein meaning "writing". This closely relates to what we are doing in this project but is typically used for watermarking photos this day and age. [2]

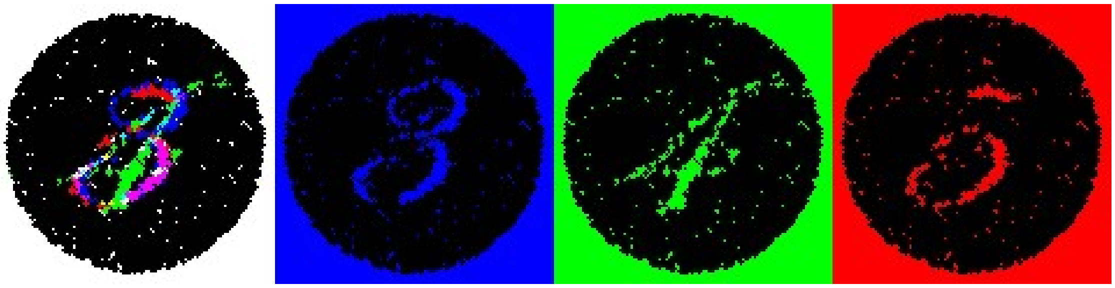


Figure 5: Steganography Example

Steganography doesn't typically give a way to the users of removing the image overlays from one another, so this is where we differ from this practice. We are making the required information accessible to the desired recipient in order to separate the pictures from one another.

Projected Versus Actual Milestones:

1. Create an image 256x256 using only ascii text. - COMPLETE
2. Implement a method of resizing images. - COMPLETE
3. Combine Message Image with Private and Public Images. - COMPLETE
4. Successfully decode an encrypted image. - COMPLETE
5. Implement the encryption scheme by having the script obtain the images from ~~Facebook automatically~~ a web hosted image catalogue. - COMPLETE
6. Complete Final Report. - COMPLETE

In comparison with our projected Milestones we are pleased with our results. We composed a script that would pull images from picsum.com in order to obtain public and private keys for use with the encoder. However, we determined that using the private key generator would make our encryption much stronger since the private key generator created a 256x256 image with random Red, Green and Blue channel values which helps obscure the ascii image pixels more reliably. Also, the image used for the private key is not a known quantity that one could attempt to find using an image search.

An issue that we encountered when decrypting caused our script to terminate earlier than expected, causing only a portion of the original message to show in the final decrypted message output. This resulted in us implementing a text file that outputted the coordinate and Red, Green, and Blue values of that point for debugging purposes. We realised that when we subtracted the public and private values from a pixel in the encrypted image, the cases where the sum was zero sometimes should have resulted in a value of 255. The condition for our decryption process to cease was when a pixel was read that indicated an X and Y coordinate pair corresponding to 0 and 0 respectively. We managed to fix this by limiting our random X and Y coordinate generator to a range of 0 to 254. Due to the compression of the

.JPEG filetype most of our testing was completed with .PNG files to ensure smooth operation between users.

Recommendations and Improvements:

One issue we have with this implementation is securely transmitting the private image to the other individual. This currently must be done via a secure channel such as physical media. In a future revision implementing some mathematical operations for the diffusion and delivery of this information would be ideal. An example of how this could be done is seen in the RSA encryption process.

Another limitation we see with this implementation is the issue we see when approaching the limit to our message size. Since we are using a random number generator for the coordinates of the used point and comparing it to a table of occupied points this begins to take much longer the more points as we approach the limit of 65,536 (actually 65,025 due to the limiting of array size to [0, 254]). This could possibly be improved through a binary check rather a linear check but may result in a much higher density of points in some areas of the picture.

Validation:

We validated the operation of this process through some simple tests. These tests will be available in the included resource and deliverable files. A demonstration of the overall functionality is available on Youtube at https://www.youtube.com/watch?v=IdLkFhk0hmE. [3]

Initial Conditions:

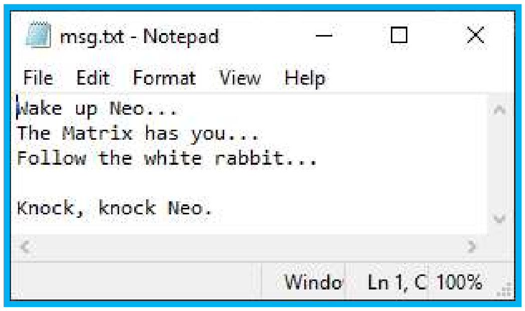
Each of the following four tests require a message contained in “msg.txt”, a public key image named “pub.png”, and the python scripts “**asciiToImg.py**”, ”**privateKeyGenerator.py**”, ”**encrypt.py**”, and “**decrypt.py**”.

The following tests were performed by running the scripts in the following order:

1. **asciiToImg.py** – The file names “msg.txt” will be converted into an image called “msgImg.png”
2. **privateKeyGenerator.py** – A private key image named “pk.png” will be created.
3. **encrypt.py** – “msgImg.png”, “pk.png”, and “pub.png” will be combined to form “enc.png”, and text files will be generated containing info about the RGB values of each image.
4. **decrypt.py** – “enc.png” will be inspected and “pk.png” and “pub.png” will be used to produce “decryptedMsg.txt” which contains the decrypted message.

Test 1 – "The Matrix"

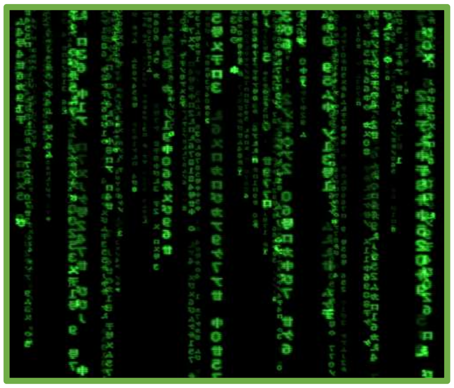
msg.txt



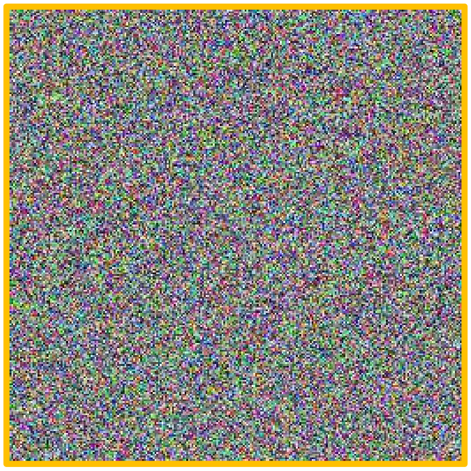
```
Wake up Neo...
The Matrix has you...
Follow the white rabbit...

Knock, knock Neo.
```

pub.png

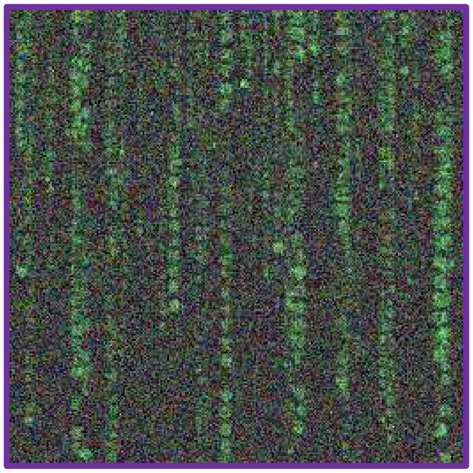


pk.png

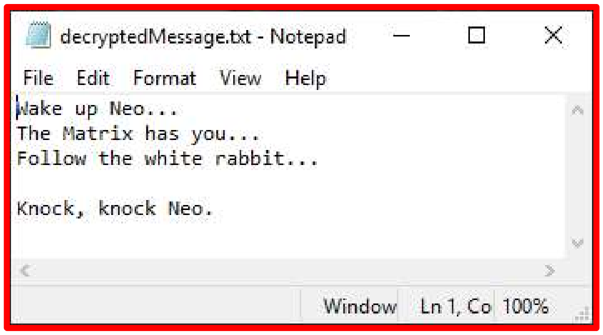


msgImg.png

enc.png



decryptedMessage.txt

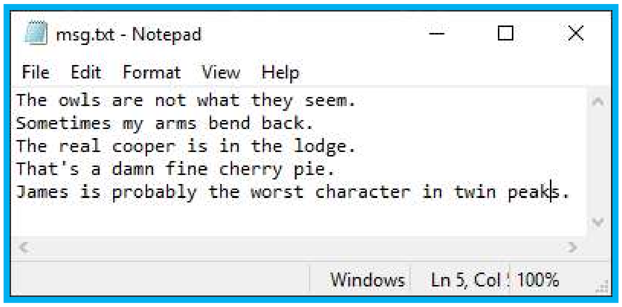


```
Wake up Neo...
The Matrix has you...
Follow the white rabbit...

Knock, knock Neo.
```

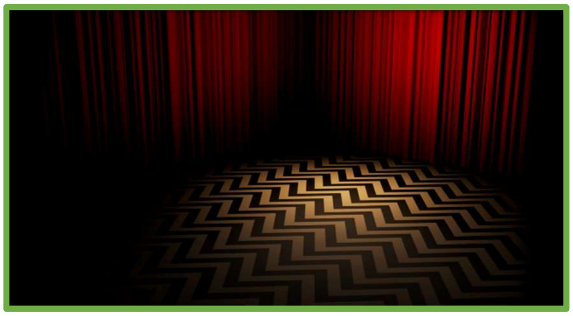
Test 2 – “Twin Peaks”

msg.txt

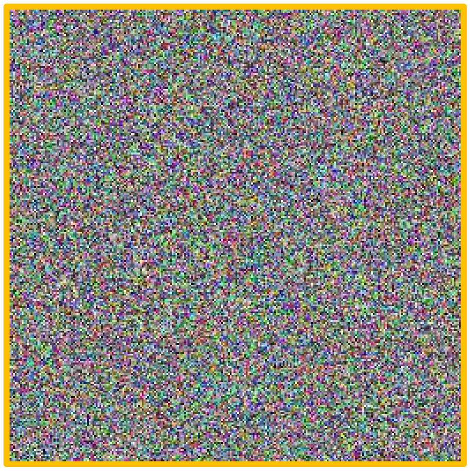


```
The owls are not what they seem.
Sometimes my arms bend back.
The real cooper is in the lodge.
That's a damn fine cherry pie.
James is probably the worst character in twin peaks.
```

pub.png

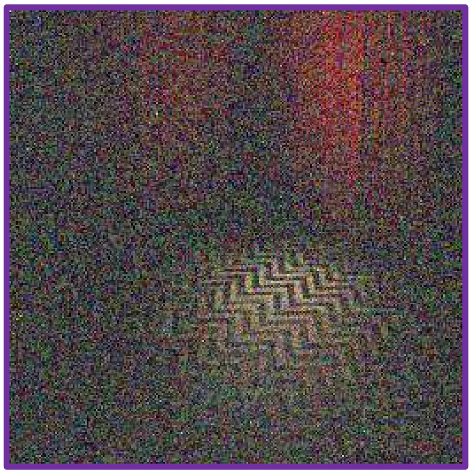


pk.png

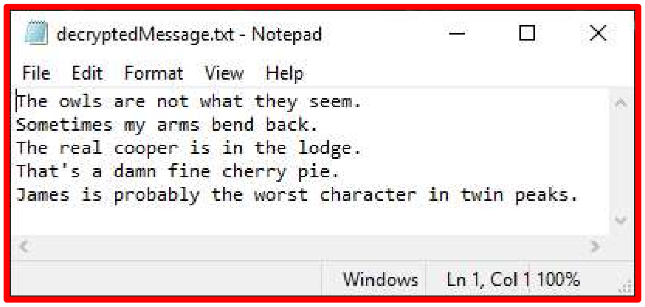


msgImg.png

enc.png



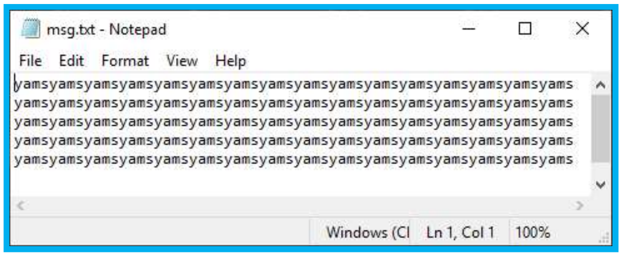
decryptedMessage.txt



```
The owls are not what they seem.
Sometimes my arms bend back.
The real cooper is in the lodge.
That's a damn fine cherry pie.
James is probably the worst character in twin peaks.
```

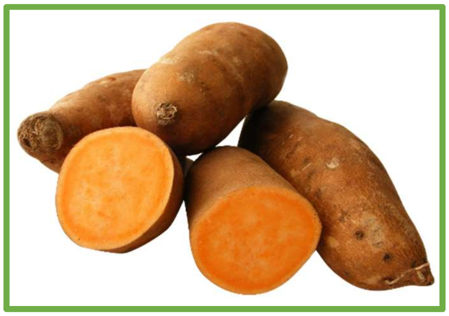
Test 3 – “Yams”

msg.txt

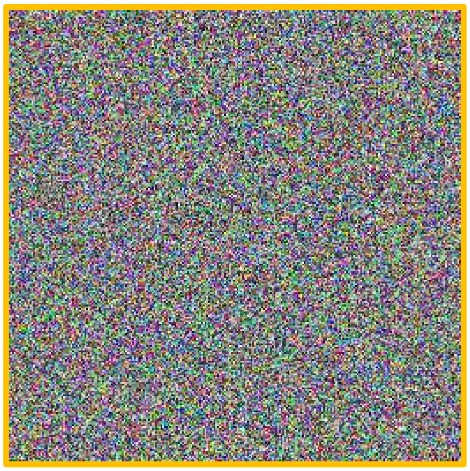


```
yamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyams
yamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyams
yamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyams
yamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyams
yamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyams
```

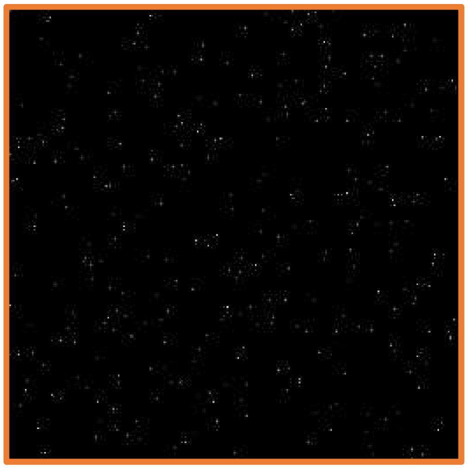
pub.png



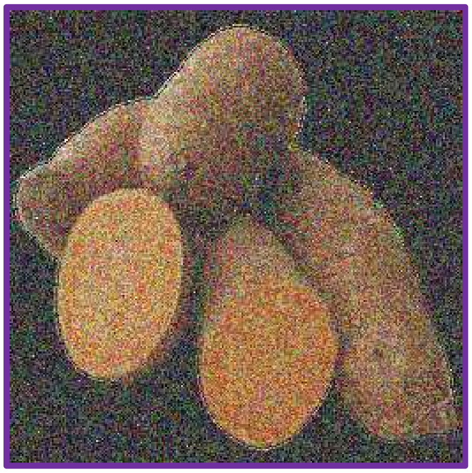
pk.png



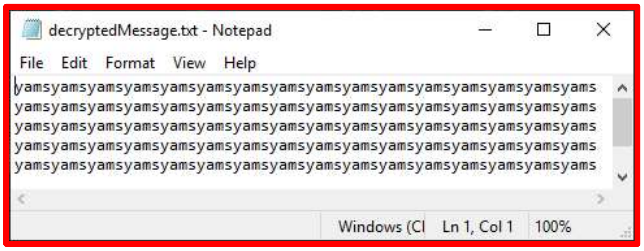
msgImg.png



enc.png



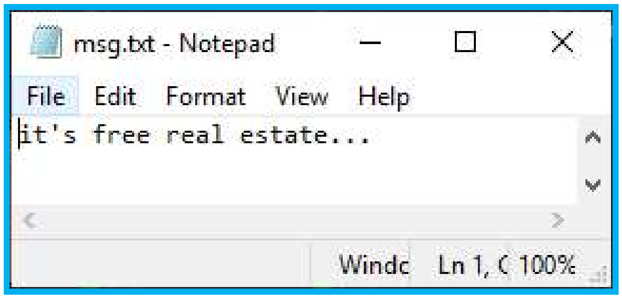
decryptedMessage.txt



```
yamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyams
yamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyams
yamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyams
yamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyams
yamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyamsyams
```

Test 4 – “Real Estate”

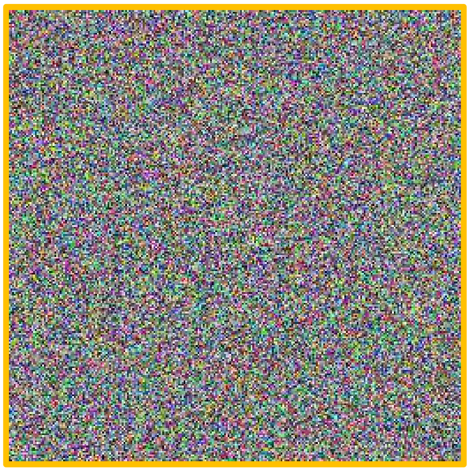
msg.txt



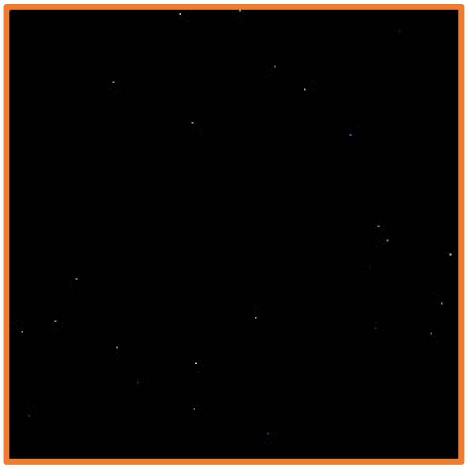
```
it's free real estate...
```

pub.png

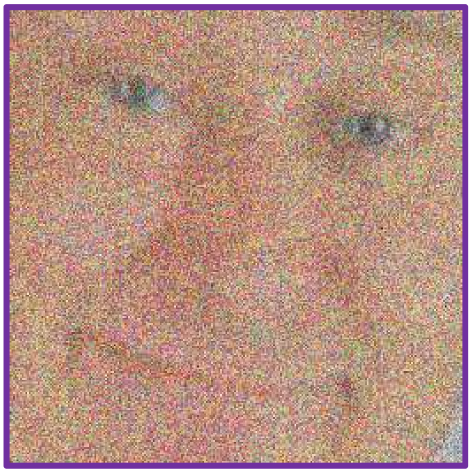
pk.png



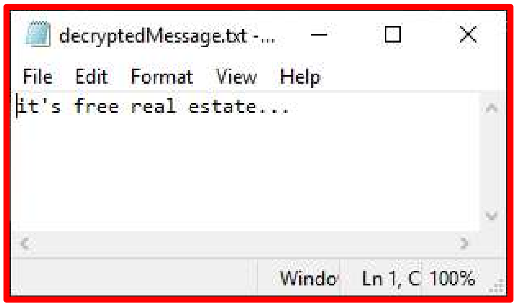
msgImg.png



enc.png



decryptedMessage.txt

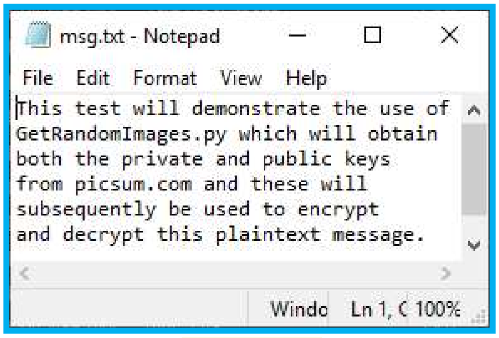


```
it's free real estate...
```

Test 5 – “Random Images”

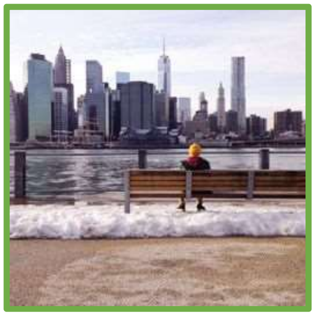
This test will use “GetRandomImages.py” to obtain images from picsum.com and save two random images as “pk.png” and “pub.png”, Otherwise the scripts are run in the order as indicated above with the exception of the “privateKeyGenerator.py” script. Just to illustrate how the encryption is compromised when using a conventional image as the private key.

msg.txt

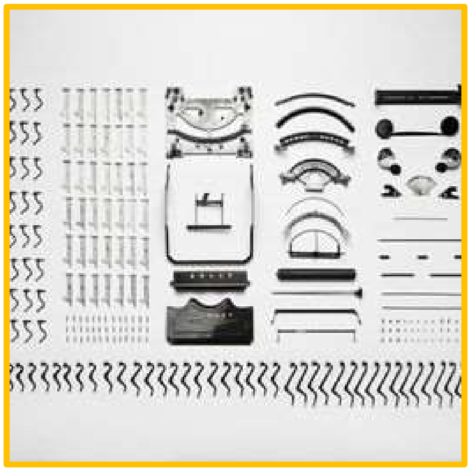


```
This test will demonstrate the use of
GetRandomImages.py which will obtain
both the private and public keys
from picsum.com and these will
subsequently be used to encrypt
and decrypt this plaintext message.
```

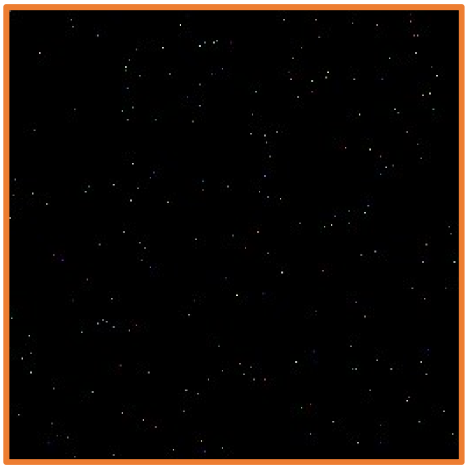
pub.png



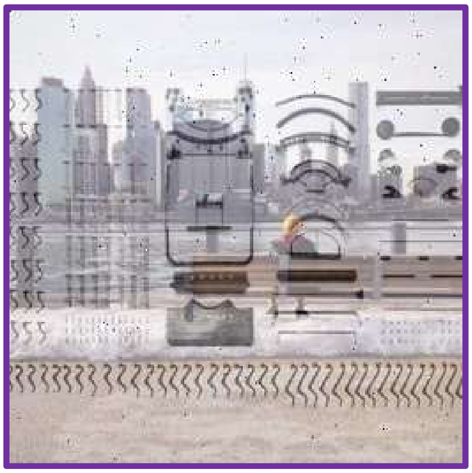
pk.png



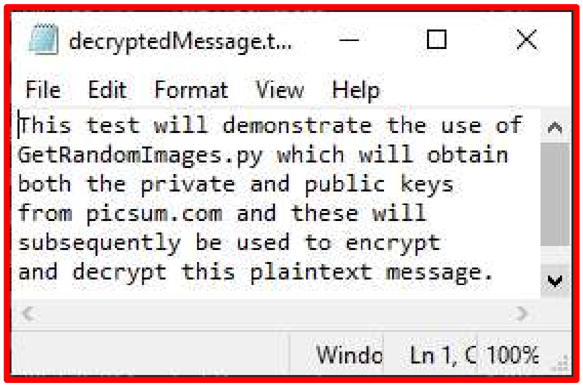
msgImg.png



enc.png



decryptedMessage.txt



```
This test will demonstrate the use of
GetRandomImages.py which will obtain
both the private and public keys
from picsum.com and these will
subsequently be used to encrypt
and decrypt this plaintext message.
```

Resources:

1. Pillows – Python image processing library (PIL)
2. Wikipedia – Free Online Encyclopedia
3. Youtube – Video Distribution Platform