COMP 8505 Assignment 2

Least Significant Bit Steganography

Callum Styan, A00807867

COMP 8505 Assignment 2 1
Introduction **4**
File Structure **4**
pyStego 4
imageFunctions 4
stegoFunctions 4
Instructions **4**
Example Usage 5
Design **5**
Pseudocode 6
stegoImage 6
unstegoImage 6
getLSBs - from a stego'd image 6
getSecretFileBits 6
constructHeader 7
compareSizes 7
Testing **7**
Test 1 8
Test 2 8
Test 3 9
Test 4 10
Test 5 11
Test 6 14
Test 7 15
Test 8 15

Introduction

The purpose of this assignment was to develop a basic steganography application using the least significant bit method. In this method we modify the least significant bit of each pixels RGB channel in a cover image to reflect a bit in a secret file. We do this until all bits from the secret file have been placed into the cover image.

File Structure

pyStego

This file acts as the executable portion of the application. It parses command line arguments provided by the users, and contains one function that checks to see if the cover image is large enough to store all the data required for hiding the secret file.

imageFunctions

This file contains various image related functions. Some examples are getting the least significant bits from every pixel within a stego'd image, a function to open + read the secret file and turn all it's data into a binary list, and a function that builds the header (file name + file size) that we prepend to the file data when performing a stego.

stegoFunctions

This file contains the top level stego and unstego functions, using functions from imageFunctions in order to manipulate the cover image.

Instructions

In order to use this steganography application you'll need Python 2.7, and the Pillow library. You can install Pillow via pip or easy_install, for example: pip install Pillow.

The application requires a cover image to perform the steganography on. This image can be of any format, but the application assumes the image is in RGB mode. RGBA would also work. This is because the stego function works by modifying the LSB of each RGB channel for every pixel in the cover image. The secret file you wish to hide within the cover image can be of any format, as long as the cover image has enough space to store it.

Example Usage

Store secretImage.png in coverImage.png:
`python pystego.py -m e -s path/to/secretImage.png -i path/to/coverImage.png`

This will output stegoImage.bmp, which is the pixels of coverImage.png modified with the bits from secretImage.png. As the console output states, you should renamed stegoImage.bmp before using it for real life purposes.

Store secretImage.png in coverImage.png as a png:
`python pystego.py -m e -s path/to/secretImage.png -i path/to/coverImage.png -o png`

This will produce the same output as the previous example, but as a png image instead of a bmp.

Pull secret file out of a stego'd image, save to Desktop:
`python pystego.py -m d -s ~/Desktop -i path/to/stegoImage.png`

This will pull the data for a secret file out of a stego'd image, including the file name and any header information associated with that filetype. The file will be saved under the same name as the file that was hidden in the stego'd image.

Design

As mentioned previously, this application makes use of the least significant bit method of steganography. My implementation works as follows:

To stego an image we start by determining if the cover image will be able to hold all the information for the secret file. To do this, first we get the number of bits the cover image will be able to store, which is the number of pixels in the image times three (for each channel in RGB). The amount of space the secret file will require is equal to the `filename\0` plus the `file size\0` plus `all of the data within the file`. The cover image needs to be able to store at least the number of bits required by the amount of space the secret file requires. If we have enough space to store the secret file then we can start to perform the steganography operations. First, we get `filename\0` plus the `file size\0` plus `all of the data within the file` and convert it to a list of bits. Then we loop through all of the pixels in the cover image and modifying their least significant bit to match the bits in the secret file data, ending when we run out of bits we need to store in the cover image.

The operations to unstego an image are fairly simple. Again we get a list of all the pixels, this time from the stego'd image, and then loop through all of them. In this case we can just check if each channel in the RGB representation of the pixel and we'll know the value of the least significant bit. We append all the LSB's into one list of bits, and then convert that list of bits to a string of bytes by ASCII representation. Then we split the list into three based on the first two

null characters (\0). The first string will be the name of the secret file, the second is the size of the secret file, and the third is the rest of the data. We can then create a new file with the secret files name, and write all the bytes in the third string up to the file size, that way we're not writing and LSB's that weren't actually part of the secret files data.

Pseudocode

---

stegoImage

get all the data for the secret file + the header with it's name and size
get a list of all the pixels in the cover image
for each pixel in the list of pixels
    modify each LSB in the RGB values with the bits from secret file data
    put new modified pixel back into list of pixels
    if there is no more bits left in secret file data, end loop
open a new image with the same dimensions of the cover image
save the modified pixels into the new image

---

unstegoImage

get all the least significant bits from the stego'd image
convert list of bits into a string of ascii bytes
split string on first two null characters
open a new file with the filename
save the data from 0 to filesize into the new file

---

getLSBs - from a stego'd image

get a list of all the pixels in the stego'd image
for each pixel in the list of pixels
    check each RGB channel, if value is even append 0 to a list of bits, else append a 1
return list of bits

---

getSecretFileBits

read the secret file into memory
convert data to a list of bytes
for each byte
    convert byte to binary representation, append to data container

constructHeader

get the name of the secret file from the provided path argument
get the size of the file
create a byte array of the filename + \0 + file size + \0

compareSizes

get the size of the secret file
get the number of bits we can store in the cover image (# of pixels * 3)
compare sizes, return whether or not cover image can store secret file

Testing

In all test cases images that we unstego have been stego'd with this application, except in test case 7 where we attempt to unstego the original cover image. Because of this, all test cases imply that the writing of the secret file into the cover image works as intended.

Test Case #	Test	Pass or Fail Conditions	Pass/Fail
1	cover image has enough space	pass - continue to stage fail - exits after comparison	pass
2	cover image doesn't have enough space	pass - exit after comparison fail - exits after comparison	pass
3	unstego a bitmap	pass - we get the secret file fail - secret file is bad/corrupt	pass
4	unstego a png	pass - we get the secret file fail - secret file is bad/corrupt	pass
5	unstego where secret file was an image	pass - image is viewable fail - secret file is bad/corrupt	pass
6	unstego where secret file was a pdf	pass - pdf is readable fail - secret file is bad/corrupt	pass
7	unstego an image that hasn't been stego'd	pass - program crashes or output of unstego is unreadable fail - anything else	pass
8	visual inspection, does stego'd image have visual artifacts	pass - image appears to be the same as original cover image fail - anything else	pass

Test 1

In this test case we're using a cover image that has enough space to store the secret file. The application proceeds with the steganography operation, meaning the test passes.

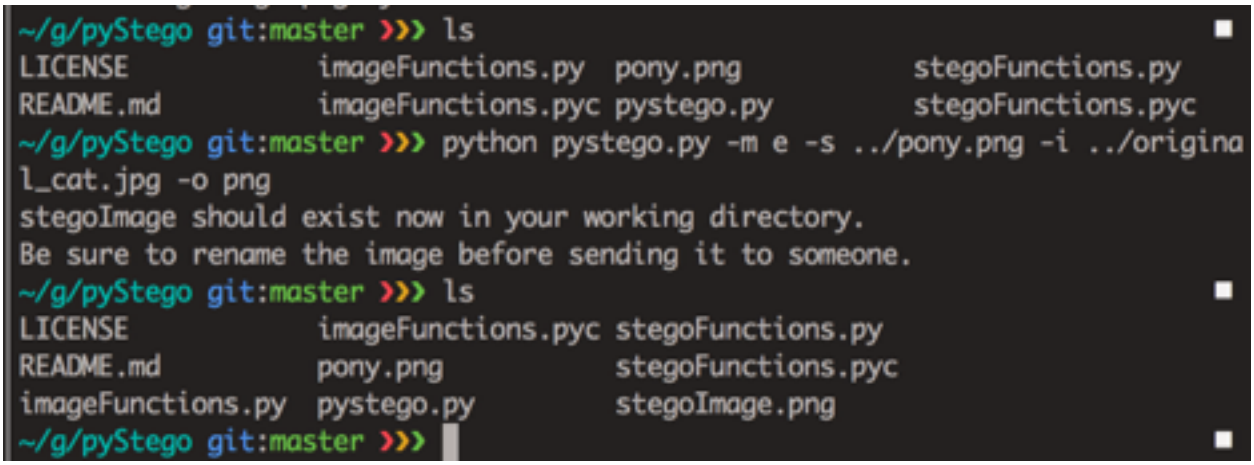


Fig 1.1: Here we can see that the steganography operation completes and the stego'd image is created.

Test 2

In this test case we're using a cover image that does not have enough space to store the secret file. The application exits after printing a message saying the cover image is not big enough, meaning the test passes.

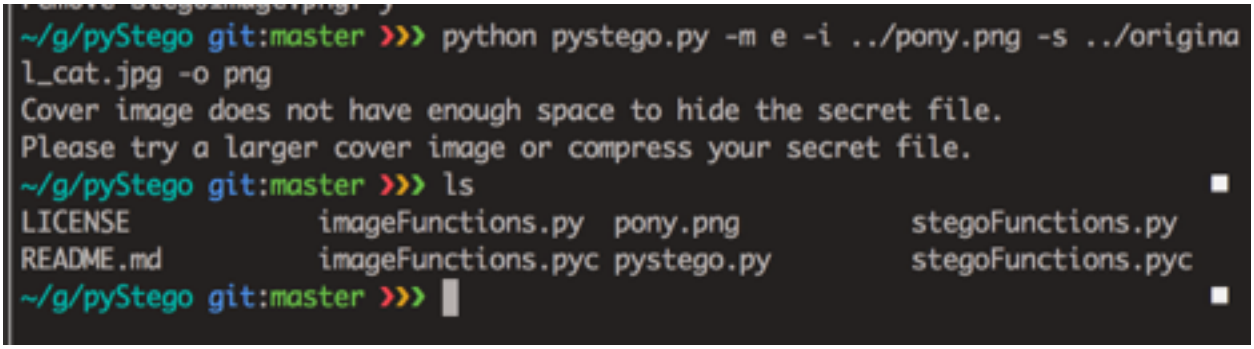


Fig 2.1: Here we can see that the application tells the user that the cover image is not large enough to store the secret file and exits before performing any steganography operations on the cover image.

Test 3

In this test we're ensuring we can unstego a bitmap image created by our application. The usntego operation completes and we get the secret file, confirming the unstego works by comparing the file sizes.

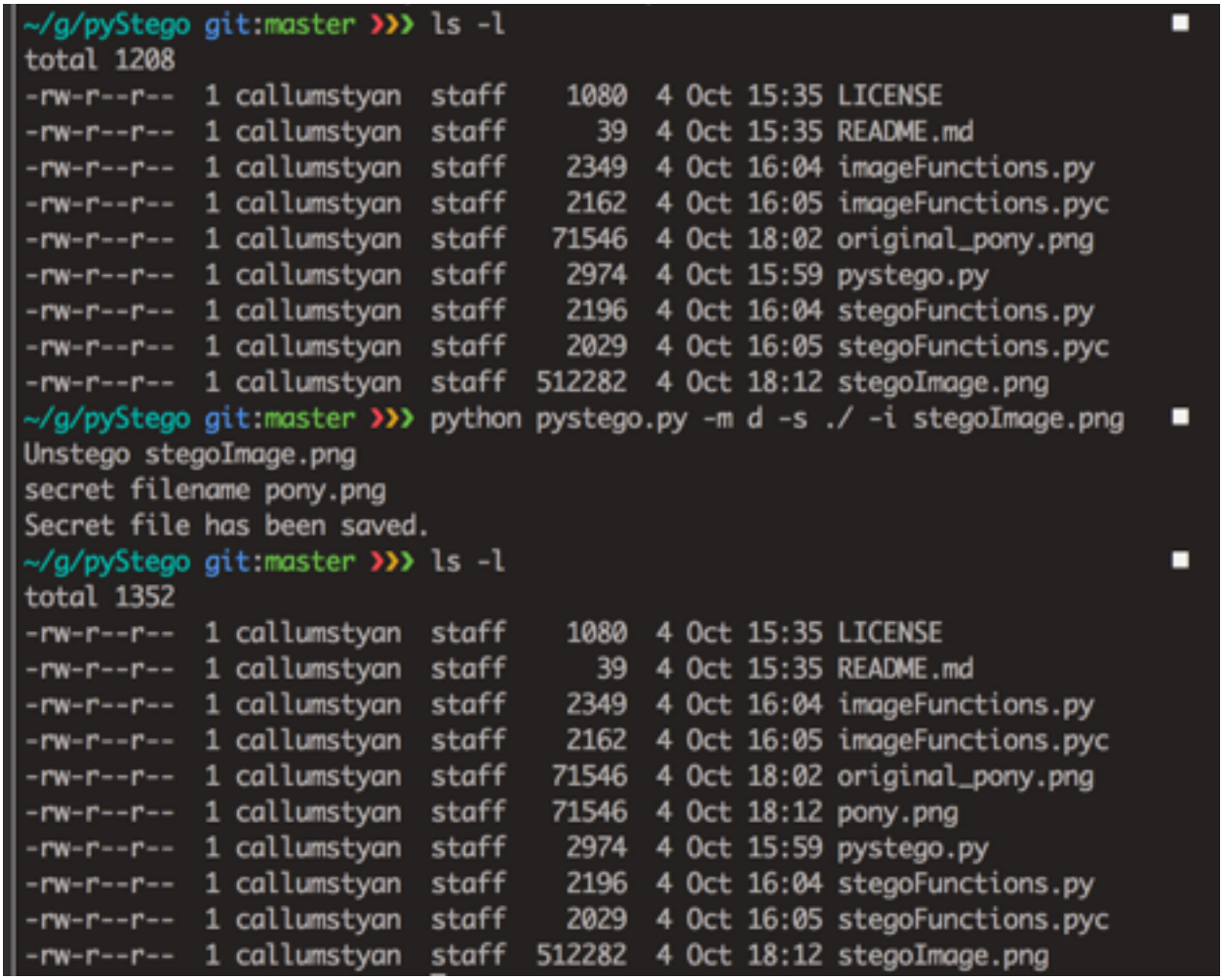
```
~/g/pyStego git:master ››› ls -l
total 12352
-rw-r--r--  1 callumstyan  staff     1080  4 Oct 15:35 LICENSE
-rw-r--r--  1 callumstyan  staff       39  4 Oct 15:35 README.md
-rw-r--r--  1 callumstyan  staff     2349  4 Oct 16:04 imageFunctions.py
-rw-r--r--  1 callumstyan  staff     2162  4 Oct 16:05 imageFunctions.pyc
-rw-r--r--  1 callumstyan  staff    71546  4 Oct 18:02 original_pony.png
-rw-r--r--  1 callumstyan  staff     2974  4 Oct 15:59 pystego.py
-rw-r--r--  1 callumstyan  staff     2196  4 Oct 16:04 stegoFunctions.py
-rw-r--r--  1 callumstyan  staff     2029  4 Oct 16:05 stegoFunctions.pyc
-rw-r--r--  1 callumstyan  staff  6220854  4 Oct 18:07 stegoImage.bmp
~/g/pyStego git:master ››› python pystego.py -m d -s ./ -i stegoImage.bmp
Unstego stegoImage.bmp
secret filename pony.png
Secret file has been saved.
~/g/pyStego git:master ››› ls -l
total 12496
-rw-r--r--  1 callumstyan  staff     1080  4 Oct 15:35 LICENSE
-rw-r--r--  1 callumstyan  staff       39  4 Oct 15:35 README.md
-rw-r--r--  1 callumstyan  staff     2349  4 Oct 16:04 imageFunctions.py
-rw-r--r--  1 callumstyan  staff     2162  4 Oct 16:05 imageFunctions.pyc
-rw-r--r--  1 callumstyan  staff    71546  4 Oct 18:02 original_pony.png
-rw-r--r--  1 callumstyan  staff    71546  4 Oct 18:07 pony.png
-rw-r--r--  1 callumstyan  staff     2974  4 Oct 15:59 pystego.py
-rw-r--r--  1 callumstyan  staff     2196  4 Oct 16:04 stegoFunctions.py
-rw-r--r--  1 callumstyan  staff     2029  4 Oct 16:05 stegoFunctions.pyc
-rw-r--r--  1 callumstyan  staff  6220854  4 Oct 18:07 stegoImage.bmp
```

Fig 3.1: Here we can see that the original image was 71546 bytes. The unstego operation finds a file in stegoImage.bmp called pony.png which is also 71546 bytes.

Test 4

```
~/g/pyStego git:master ››› python pystego.py -m e -s ../javaSucks.jpg -i ../or
iginal_cat.jpg -o png
stegoImage should exist now in your working directory.
Be sure to rename the image before sending it to someone.
~/g/pyStego git:master ››› ls -l
total 1032
-rw-r--r--  1 callumstyan  staff    1080  4 Oct 15:35 LICENSE
-rw-r--r--  1 callumstyan  staff      39  4 Oct 15:35 README.md
-rw-r--r--  1 callumstyan  staff    2349  4 Oct 16:04 imageFunctions.py
-rw-r--r--  1 callumstyan  staff    2162  4 Oct 16:05 imageFunctions.pyc
-rw-r--r--  1 callumstyan  staff   71546  4 Oct 18:02 original_pony.png
-rw-r--r--  1 callumstyan  staff    2974  4 Oct 15:59 pystego.py
-rw-r--r--  1 callumstyan  staff    2196  4 Oct 16:04 stegoFunctions.py
-rw-r--r--  1 callumstyan  staff    2029  4 Oct 16:05 stegoFunctions.pyc
-rw-r--r--  1 callumstyan  staff  425243  4 Oct 18:20 stegoImage.png
~/g/pyStego git:master ››› python pystego.py -m d -s ./ -i stegoImage.png
Unstego stegoImage.png
secret filename javaSucks.jpg
Secret file has been saved.
~/g/pyStego git:master ››› ls -l
total 1088
-rw-r--r--  1 callumstyan  staff    1080  4 Oct 15:35 LICENSE
-rw-r--r--  1 callumstyan  staff      39  4 Oct 15:35 README.md
-rw-r--r--  1 callumstyan  staff    2349  4 Oct 16:04 imageFunctions.py
-rw-r--r--  1 callumstyan  staff    2162  4 Oct 16:05 imageFunctions.pyc
-rw-r--r--  1 callumstyan  staff   24700  4 Oct 18:21 javaSucks.jpg
-rw-r--r--  1 callumstyan  staff   71546  4 Oct 18:02 original_pony.png
-rw-r--r--  1 callumstyan  staff    2974  4 Oct 15:59 pystego.py
-rw-r--r--  1 callumstyan  staff    2196  4 Oct 16:04 stegoFunctions.py
-rw-r--r--  1 callumstyan  staff    2029  4 Oct 16:05 stegoFunctions.pyc
-rw-r--r--  1 callumstyan  staff  425243  4 Oct 18:20 stegoImage.png
```

In this test we're ensuring we can unstego a bitmap image created by our application. The



usntego operation completes and we get the secret file, confirming the unstego works by comparing the file sizes.
Fig 4.1: Here we can see that the original image was 71546 bytes. The unstego operation finds a file in stegoImage.bmp called pony.png which is also 71546 bytes.

Test 5

In this test we're ensuring that a secret image is pulled out of a stego'd image and saved properly so that we can view it.
Fig 5.1: Here we can see that the unstego operation properly pulls the jpeg image out of stegoImage.png.

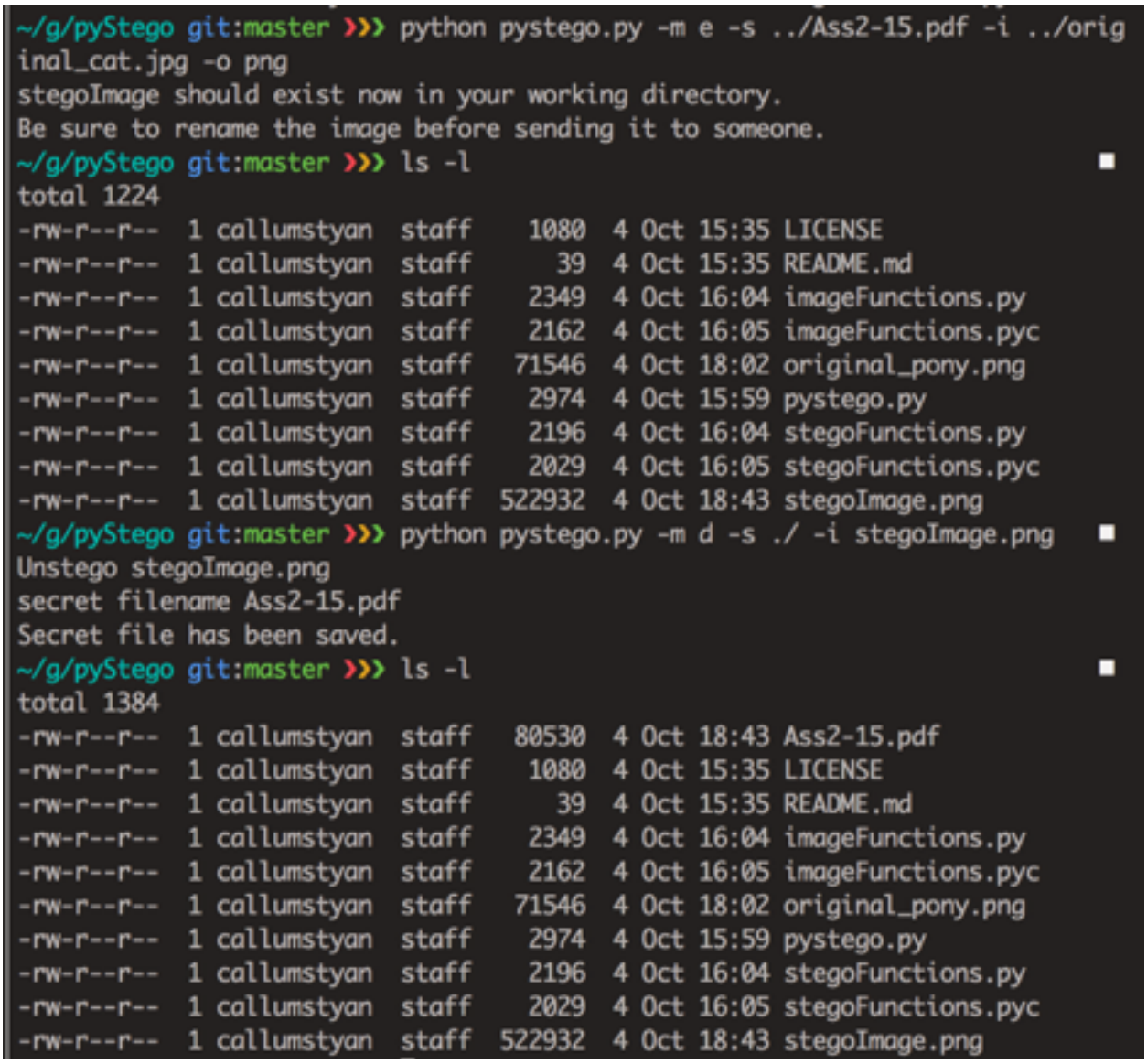


Fig 5.2: Here we can see that image pulled out of the stego'd image is viewable.

```
~/g/pyStego git:master ››› python pystego.py -m d -s ./ -i ../original_cat.jpg

Unstego ../original_cat.jpg
secret filename d[illegible]

Traceback (most recent call last):
  File "pystego.py", line 70, in <module>
    stegoFunctions.unstegoImage(args)
  File "/Users/callumstyan/git/pyStego/stegoFunctions.py", line 56, in unstego
Image
    w.write(data[: int(filesize)])
ValueError: invalid literal for int() with base 10: ''
```

Test 6

In this test we're ensuring that a secret pdf is pulled out of a stego'd image and saved properly so that we can view it.

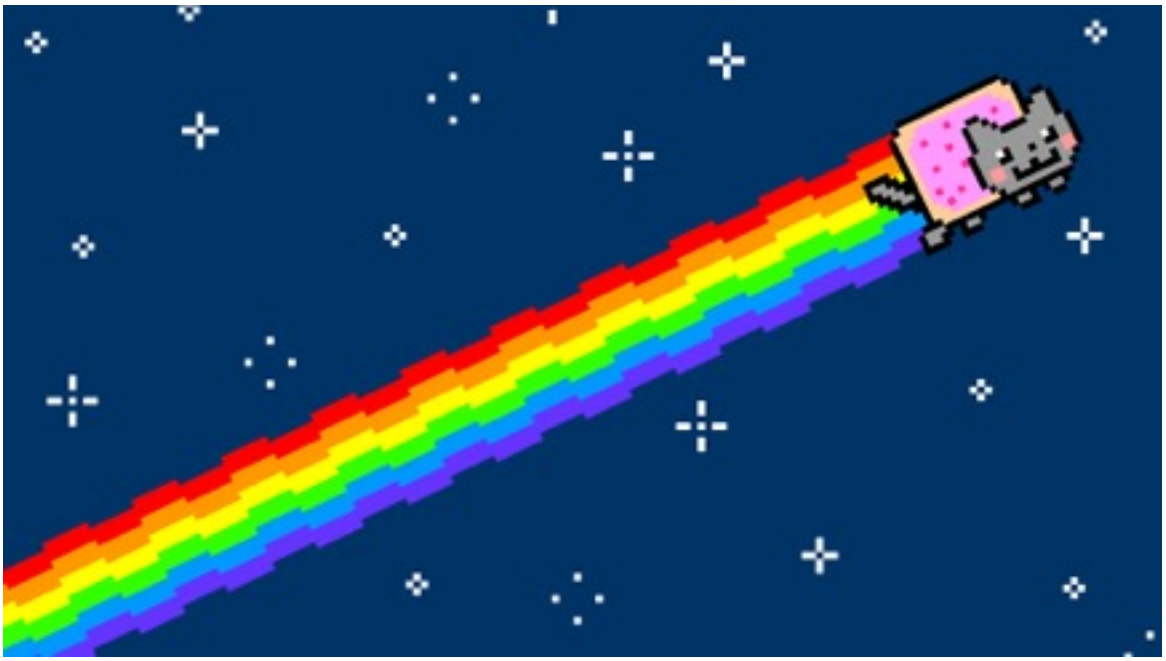
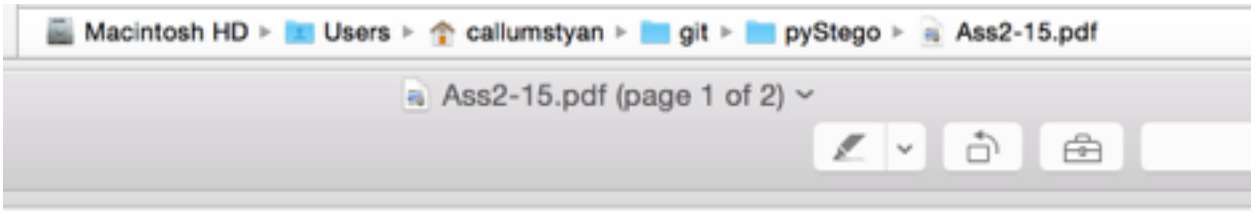


Fig 6.1: Here we can see that the unstego operation properly pulls the pdf image out of

stegoImage.png.



Comp 8505 Computer Systems Technology September 2015

Data Communication Applications

Assignment #2

5, 1100 hrs.

Fig 6.2: Here we can see that the pdf pulled out of the stego'd image is readable.

Test 7

In this test we're attempting to unstego an image we would normally use as the cover image, as in it doesn't contain any stego data.
Fig 7.1: Here we can see that the unstego operation fails and errors out because the header information with two null characters isn't present in the cover image.

Test 8

In this test we're comparing the cover image and stego'd image to check for visual artifacts in the stego'd image. The image used contains a large area with one solid colour as well as another area with many colours.
Fig 8.1: The cover image.
Fig 8.2: The stego'd image, as far as the eye can tell the images are identical.