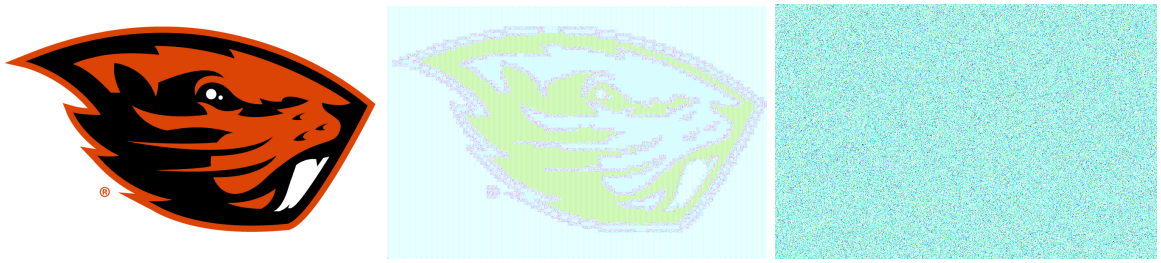
3.1

I used DES to encrypt the images in ECB and CBC modes on the flip server. I used the bless hex editor on my local machine to edit the relevant header information. I used the following commands for the beaver head:

```
> openssl des-ecb -in beaver-head.bmp -out beaver-head-ecb.bmp
> openssl des-cbc -in beaver-head.bmp -out beaver-head-cbc.bmp
```

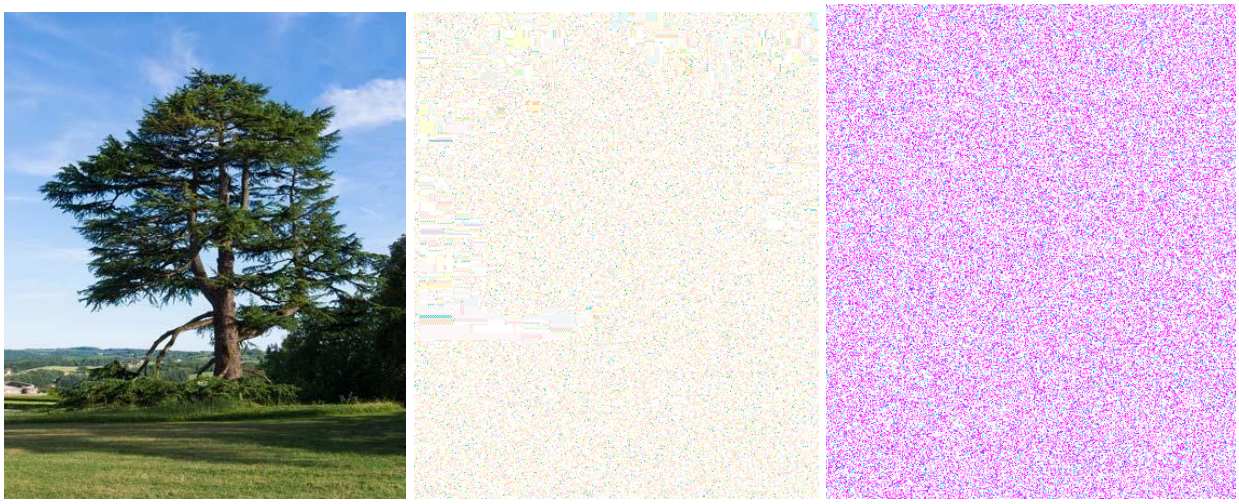
This resulted in the following images: original image on the left, ECB mode encryption center and CBC mode encryptionon the right.



The ECB mode encryption shows similarity to the original image. The beaver head can clearly be made out. CBC mode appears to be indistinguishable from randomly colored pixels. For the second image, I chose a picture of a cedar tree. I again used DES to encrypt in both ECB and CBC modes with the following commands:

```
> openssl des-ecb -in cedar.bmp -out cedar-ecb.bmp
> openssl des-cbc -in cedar.bmp -out cedar-cbc.bmp
```

This is the result: original image on the left, ECB mode encryption center, and CBC mode encryption on the right.



Unlike the beaver head, it is not possible to make out the cedar tree in ECB mode. In CBC mode, the image once again appears indistinguishable from randomly colored pixels.

This may be due to the fact that ECB mode will always have a predictable relationship from “plaintext” to “cipher”. Since the beaver head has flat colors, the colors will be transformed but are still distinguishable from each other. There are also few colors in beaver head image.This is not the case with the cedar tree. Since the cedar tree is a “natural” image, it has a wider range of colors that make it hard to see the relationship between the original image and the ECB encryption. However, there are some artifacts in the image. There are patches of pixels that are square and/or straight. This is unlikely to occur in an image of random noise, so we can’t say that it is indistinguishable from random. Both images appear random when encrypted with CBC mode. This is likely due to the avalanche/cascading properties of CBC mode. Since each block is enciphered with data from the previous block, changes propagate across the whole “message space”--or image in our case.

3.2

I used the words list provided. For the python script I also used the library provided. I checked every word in the words list as a key against the plaintext and checked if the resulting cipher text matched the provided ciphertext. This was the result:

```
[gallegon@flip1 (master) ~/f2021/cs370/project1 ] ./findkey.py plaintext
8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9 words.txt
FOUND! key: median
```

3.3

1. Due to the long processing time, I only attempted 10 trials for weak collision resistance. These were the results:

```
[nich@nich-pc CS370-Project-1]$ ./findcollision.py w 10 20000000 5
Looking for 10 weak collisions
b'ku;'
Found weak collision: b'ku;'
Trials: 16147843
b' \x0bz'
b'6\x838'
Found weak collision: b'6\x838'
Trials: 11813206
b'\xa2\xc3\xd8'
b'\xdb#R'
b'\xf1\xf2,'
Found weak collision: b'\xf1\xf2,'
Trials: 16517802
b'\xc2\xe8\x8c'
Found weak collision: b'\xc2\xe8\x8c'
Trials: 7359241
```

```
b'M\xc7\x9e'
b'\xa2\xfc\xe4'
Found weak collision: b'\xa2\xfc\xe4'
Trials: 19277706
b'k\xc5\xa5'
Found 5/10 collisions
[16147843, 11813206, 16517802, 7359241, 19277706]
Average number of trials: 14223159.6
```

It took ~14,000,000 trials on average to find a collision. Notice how only 5 collisions were found. We are able to attempt to find more collisions by adjusting the “guesses” parameter in the find_collision function. For this experiment, I set “guesses” to 20,000,000 guesses per trial. I used this because we would expect 2^24 (16,777,216) hashes with 24 bits. However, a hash function does not necessarily guarantee that 2^24 inputs would uniquely map to 2^24 hashes, so I used a slightly higher number that would hopefully capture a weak collision without using too many computing resources.

2. Due to the long processing time on my local machine, once again I did 10 trials for this experiment, this is the output:

```
[nich@nich-pc CS370-Project-1]$ ./findcollision.py s 10 20000000 5
Looking for 10 strong collisions
b'\xfa\x92Q'
Found strong collision: b'\xfa\x92Q'
Trials: 1990243
Original string: b'U2B'
Collision string: b'c^\x1e'
b'\x81\xd7\x9d'
Found strong collision: b'\x81\xd7\x9d'
Trials: 8137766
Original string: b'FZY'
Collision string: b'&,|'
b'C:\xf6'
Found strong collision: b'C:\xf6'
Trials: 5515556
Original string: b'XGE'
Collision string: b'$)T'
b'\xf5O\xb6'
Found strong collision: b'\xf5O\xb6'
Trials: 5411381
Original string: b'RTE'
Collision string: b'5\x92R'
b'\x90;\xf7'
Found strong collision: b'\x90;\xf7'
Trials: 1130691
Original string: b'CSY'
Collision string: b'\xc3@\x11'
b'\x06:\xad'
b'\xd1\xf3\xdb'
Found strong collision: b'\xd1\xf3\xdb'
```

```
Trials: 12049220
Original string: b'QE4'
Collision string: b'D\xdb\xb7'
b'^\xc1m'
b' \x00\xb3'
Found strong collision: b' \x00\xb3'
Trials: 532254
Original string: b'NLL'
Collision string: b'\x1e\x1f\x08'
b"\xa1'2"
Found strong collision: b"\xa1'2"
Trials: 3284935
Original string: b'AKY'
Collision string: b'\xc7\x1f2'
Found 8/10 collisions
[1990243, 8137766, 5515556, 5411381, 1130691, 12049220, 532254, 3284935]
Average number of trials: 4756505.75
```

This took significantly less time on average to find each collision. Requiring ~4.7 million trials per strong collision. There were also fewer fails on finding a collision, we found 8 or 10 collisions.

3. It was slightly easier to break the strong-collision property.
4. I believe this is similar to the birthday problem. Weak collision is the probability of finding someone with a specific birthday (mapping an input to a hash). Strong collision is the probability of finding two people in a room with the same birthday. The probability of finding a strong collision is higher. The “one-wayness” of a hash may be more difficult to solve than the probability that two distinct inputs may map to the same hash output, as N inputs does not necessarily map to N distinct hash outputs.