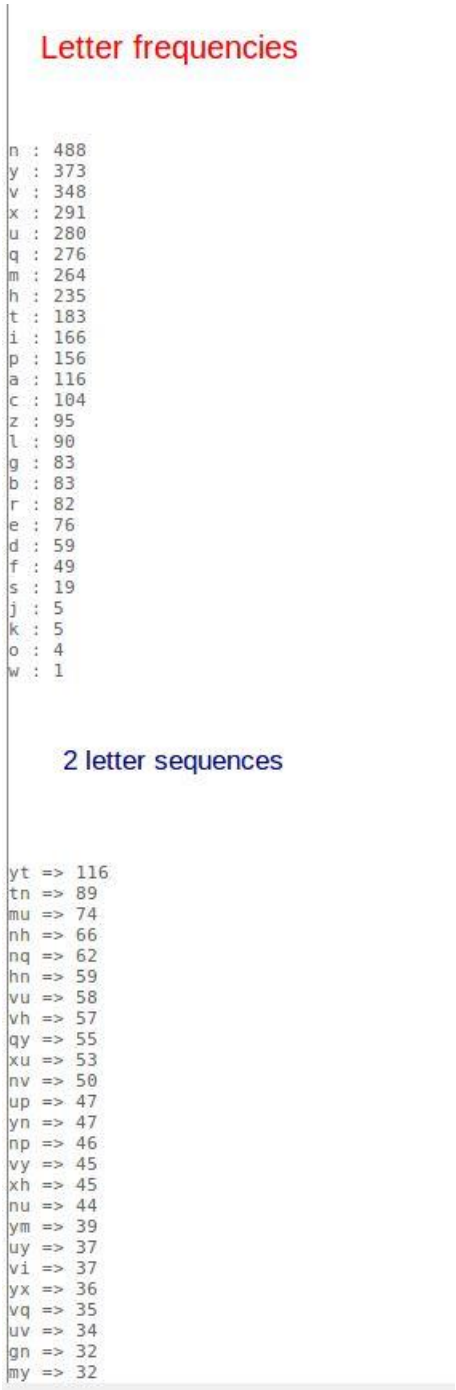
Justin Guerrero

Lab07

Task 1: Frequency Analysis Against Monoalphabetic Substitution Cipher

In this first task we are asked to decipher a text given to us by Travis. We first navigate to the website suggested by the seed book “http:www.richkni.co.uk/php/crypta/freq.php” which is a fancy website that will give us the frequency of the letters used in the encoded text. We enter the text and get the frequencies for each letter.



After reading about frequencies we decide to try and decode this text. The most commonly used letter is e in the English language, and in our code it's n. With some further investigation we see some commonly used 3 letter words that would transfer into words we see in our frequency check.

After some serious time doing educated guessing with the frequencies and investigating the code we find the following string works to decrypt the message

```
cat plaintext.txt | tr 'abcdefghijklmnopqrstuvwxyz' 'cfmypvbrlqxwieodsgkhnawotu' > ciphertext.txt
```

Which gives us the following.

```
the way the academy tabulates the big winner doesnt help in every other
category the nominee with the most votes wins but in the best picture
category voters are asked to list their top movies in preferential order if a
movie gets more than  percent of the firstplace votes it wins when no
movie manages that the one with the fewest firstplace votes is eliminated and
its votes are redistributed to the movies that garnered the eliminated ballots
secondplace votes and this continues until a winner emerges

it is all terribly confusing but apparently the consensus favorite comes out
ahead in the end this means that endofseason awards chatter invariably
involves tortured speculation about which film would most likely be voters
second or third favorite and then equally tortured conclusions about which
film might prevail

in  it was a tossup between boyhood and the eventual winner birdman
in  with lots of experts betting on the revenant or the big short the
oriwe went to spotlight last year nearly all the forecasters declared la
la land the presumptive winner and for two and a half minutes they were
correct before an envelope snafu was revealed and the rightful winner
moonlight was crowned
```

Task 2: Encryption using different ciphers and modes

In this task we are asked to experiment with different types of ciphers and modes. The three we chose to work with are listed below.

```
[04/05/20]seed@VM:~/.../sslStuff$ openssl enc -aes-128-cbc -e -in plaintext.txt -out cipher.bin -K 00112233445566778889aabb
ccddeeff -iv 0102030405060708
[04/05/20]seed@VM:~/.../sslStuff$ openssl enc -camellia-128-cbc -e -in plaintext.txt -out cipher2.bin -K 001122334455667788
89aabbccddeeff -iv 0102030405060708
[04/05/20]seed@VM:~/.../sslStuff$ openssl enc -rc4-hmac-md5 -e -in plaintext.txt -out cipher3.bin -K 00112233445566778889aa
bbccddeeff -iv 0102030405060708
AEAD ciphers not supported by the enc utility
[04/05/20]seed@VM:~/.../sslStuff$ openssl enc -des-ede3-cfb8 -e -in plaintext.txt -out cipher3.bin -K 0011223344556677889a
abbccddeeff -iv 0102030405060708
[04/05/20]seed@VM:~/.../sslStuff$ ls
cipher2.bin  cipher3.bin  cipher.bin  plaintext.txt
[04/05/20]seed@VM:~/.../sslStuff$
```

We encrypted song lyrics into unreadable cipher code that is fully encrypted in three different ways. The only way to decrypt will be by using the same cipher chosen and using the dec method.

Task 3: Encryption modes – ECB vs. CBC

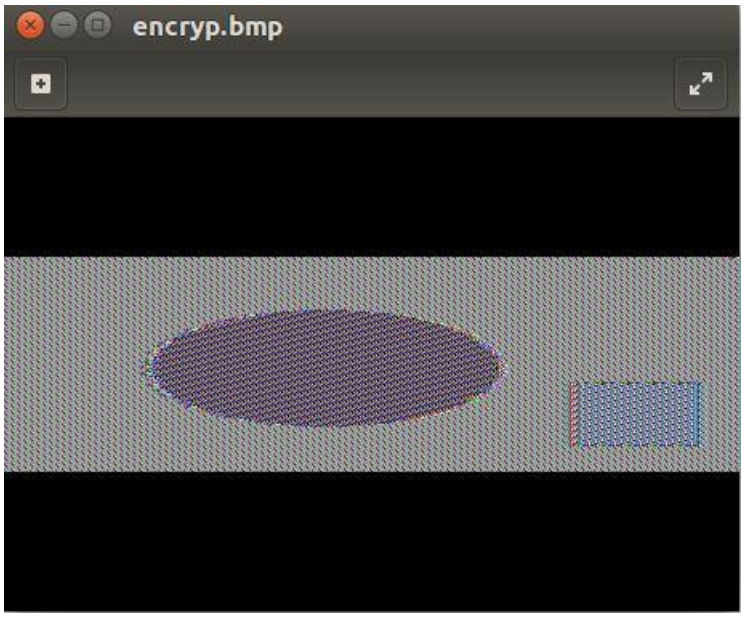
Task 3.1

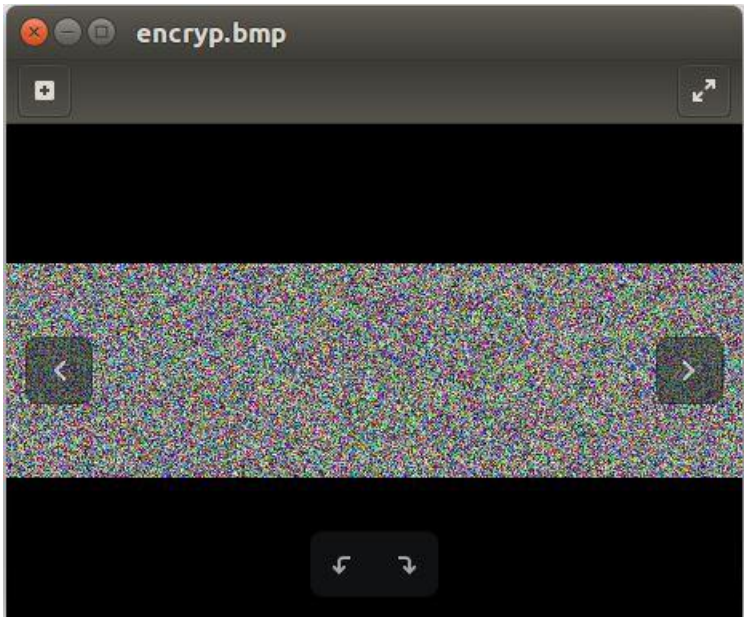
Our task here is to encrypt a photo that was given to us. The first thing we do is we look up the cypher algorithms for openssl and pick one ECB and one CBC, we choose -aes-256-ecb and -aes-256-cbc.

```
[04/05/20]seed@VM:~/.../lab09$ openssl enc -aes-256-ecb -e -in new.bmp -out p1.bmp -K 00112233445566778889aabbccddeeff -iv
0102030405060708
warning: iv not use by this cipher
[04/05/20]seed@VM:~/.../lab09$ ls
body  ciphertext.txt  header  new.bmp  non.py  p1.bmp  pic_original.bmp  plaintext.txt  sslStuff
[04/05/20]seed@VM:~/.../lab09$ openssl enc -aes-256-cbc -e -in new.bmp -out p2.bmp -K 00112233445566778889aabbccddeeff -iv
0102030405060708
```

After running those commands, we will now see two new bitmap files created in the our directory we are working in. (*Lab09). However, if you try to open the files, an error will show telling that the file is in an unrecognized format. So, we need to retrieve the original 54-byte header in order to actually open the file as a bitmap. Here we use the commands given to us in the lab to copy the 54-byte header information from the original file into the encrypted files.

And finally we have our encrypted photos, EBC followed by CBC.





What our takeaways from the two types of encryptions are the following:
EBC encryption is better than none, but we still see the majority of the photo and can tell what it is. While in this case it's easy to tell, and maybe in other photos this would be okay.
However, CBC we can see that you cannot see anything besides static. This allows the image to be completely hidden and transformed into something unreadable. CBC is the recommended encryption service for photos.

Task 3.2

This task we went out and found our own BMP file from Deltas website, and we encrypt it using CBC.

So as long as we use the proper header length and file concatenation we can encrypt our own photos using this method!

Task 4: Experimenting with padding

We want to see what the computer has to pad in order for the encryption to be completed. We start by compiling the 5,10, and 16 byte files into 4 different encryption types. See below.
(insert encryption photos task 4)

They all appear to have padding for various encryption methods and for all in the 5 and 16 byte files. But we note that the 10 byte file did not need padding in some runs.

```
[04/05/20]seed@VM:~/.../lab09$ echo -n "01234" > f1.txt
[04/05/20]seed@VM:~/.../lab09$ echo -n "0123456789" > f2.txt
[04/05/20]seed@VM:~/.../lab09$ echo -n "0123456789987654" > f3.txt
[04/05/20]seed@VM:~/.../lab09$ du -b f1.txt
5       f1.txt
[04/05/20]seed@VM:~/.../lab09$ du -b f2.txt
10      f2.txt
[04/05/20]seed@VM:~/.../lab09$ du -b f3.txt
16      f3.txt
[04/05/20]seed@VM:~/.../lab09$
```

```
[04/05/20]seed@VM:~/.../lab09$ xxd decf1.txt
00000000: 3031 3233 340b 0b0b 0b0b 0b0b 0b0b 0b0b  01234...........
[04/05/20]seed@VM:~/.../lab09$ openssl enc -aes-128-cbc -e -in f2.txt -out newf2.txt -K 00112233445566778889aabbccddeeff -i
v 0102030405060708
[04/05/20]seed@VM:~/.../lab09$ openssl enc -aes-128-cbc -d -nopad -in newf2.txt -out decf2.txt -K 00112233445566778889aabbc
cddeeff -iv 0102030405060708
[04/05/20]seed@VM:~/.../lab09$ xxd decf2.txt
00000000: 3031 3233 3435 3637 3839 0606 0606 0606  0123456789......
[04/05/20]seed@VM:~/.../lab09$ openssl enc -aes-128-cbc -e -in f3.txt -out newf3.txt -K 00112233445566778889aabbccddeeff -i
v 0102030405060708
[04/05/20]seed@VM:~/.../lab09$ openssl enc -aes-128-cbc -d -nopad -in newf3.txt -out decf3.txt -K 00112233445566778889aabbc
cddeeff -iv 0102030405060708
[04/05/20]seed@VM:~/.../lab09$ xxd decf3.txt
00000000: 3031 3233 3435 3637 3839 3938 3736 3534  0123456789987654
00000010: 1010 1010 1010 1010 1010 1010 1010 1010  ................
```

```
[04/08/20]seed@VM:~/.../lab09$ openssl enc -aes-128-ecb -e -in f1.txt -out newf1.txt -K 0011223344556
v 0102030405060708
warning: iv not use by this cipher
[04/08/20]seed@VM:~/.../lab09$ openssl enc -aes-128-ecb -d -nopad -in newf1.txt -out decf1.txt -K 001
cddeeff -iv 0102030405060708
warning: iv not use by this cipher
[04/08/20]seed@VM:~/.../lab09$ xxd decf1.txt
00000000: 3031 3233 340b 0b0b 0b0b 0b0b 0b0b 0b0b  01234...........
[04/08/20]seed@VM:~/.../lab09$ openssl enc -aes-128-ecb -e -in f2.txt -out newf2.txt -K 0011223344556
v 0102030405060708
warning: iv not use by this cipher
[04/08/20]seed@VM:~/.../lab09$ openssl enc -aes-128-ecb -d -nopad -in newf2.txt -out decf2.txt -K 001
cddeeff -iv 0102030405060708
warning: iv not use by this cipher
[04/08/20]seed@VM:~/.../lab09$ xxd decf2.txt
00000000: 3031 3233 3435 3637 3839 0606 0606 0606  0123456789......
[04/08/20]seed@VM:~/.../lab09$ openssl enc -aes-128-ecb -e -in f3.txt -out newf3.txt -K 0011223344556
v 0102030405060708
warning: iv not use by this cipher
[04/08/20]seed@VM:~/.../lab09$ openssl enc -aes-128-ecb -d -nopad -in newf3.txt -out decf3.txt -K 001
cddeeff -iv 0102030405060708
warning: iv not use by this cipher
[04/08/20]seed@VM:~/.../lab09$ xxd decf3.txt
00000000: 3031 3233 3435 3637 3839 3938 3736 3534  0123456789987654
00000010: 1010 1010 1010 1010 1010 1010 1010 1010  ................
[04/08/20]seed@VM:~/.../lab09$
```

```
[04/08/20]seed@VM:~/.../lab09$ openssl enc -aes-128-cfb -e -in f2.txt -out newf2.txt -K 00112233445566778889a
v 0102030405060708
[04/08/20]seed@VM:~/.../lab09$ openssl enc -aes-128-cfb -d -nopad -in newf2.txt -out decf2.txt -K 00112233445
cddeeff -iv 0102030405060708
[04/08/20]seed@VM:~/.../lab09$ xxd decf2.txt
00000000: 3031 3233 3435 3637 3839                0123456789
```

```
[04/08/20]seed@VM:~/.../lab09$ openssl enc -aes-128-ofb -e -in f2.txt -out newf2
v 0102030405060708
[04/08/20]seed@VM:~/.../lab09$ openssl enc -aes-128-ofb -d -nopad -in newf2.txt
cddeeff -iv 0102030405060708
[04/08/20]seed@VM:~/.../lab09$ xxd decf2.txt
00000000: 3031 3233 3435 3637 3839                0123456789
```

Task 5: Experimenting with error propagation and corrupted ciphertext

Task 5.1

- In ECB mode, only one block is affected when any problem in a cipher text happens; furthermore; moreover, each block is decrypted independently. However, the corrupted bit of the 55th byte in cipher text block 8 bytes might spread to all n bits in plaintext block 8 bytes since we do the decryption one block at a time.
- In CBC mode, there was affect in two blocks.
- In CFB mode, there is problem in n / r number of blocks. Therefore, the error propagation criterion is poorer (Modes of Operation of Block Ciphers).

- In OFB mode, the feedback is only in the key-generation system. If the single digit of the 55th byte corrupted, then in plain text that only that byte or character is corrupted. Thus, only OFB mode shows the most promising result and almost all the texts are recovered.

Task 5.2

```
[04/08/20]seed@VM:~/.../task$ openssl enc -aes-128-ofb -d -in plaintextofb.txt -out plaintextofb1.txt
89aabbccddeeff -iv 010203
[04/08/20]seed@VM:~/.../task$ openssl enc -aes-128-cfb -d -in plaintextcfb.txt -out plaintextcfb1.txt
89aabbccddeeff -iv 010203
[04/08/20]seed@VM:~/.../task$ openssl enc -aes-128-ecb -d -in plaintextecb.txt -out plaintextecb1.txt
89aabbccddeeff -iv 010203
warning: iv not use by this cipher
[04/08/20]seed@VM:~/.../task$ openssl enc -aes-128-cbc -d -in plaintextcbc.txt -out plaintextcbc1.txt
89aabbccddeeff -iv 010203
```

```
[04/08/20]seed@VM:~/.../task$ cat plaintextcbc1.txt
Parents need to know that although this underdog sports movie is based on the Ja@w▯b@@▯;@@@%@@@eam's@appearance ir
```

```
[04/08/20]seed@VM:~/.../task$ cat plaintextecb1.txt
Parents need to know that although this underdog sports movie is based on the JaSsC@@@▯{&@@W@>o+eam's appearance in the
```

```
[04/08/20]seed@VM:~/.../task$ cat plaintextcfb1.txt
Parents need to know that although this underdog sports movie is based on the Jamaica@ bobsled t@@@z@@~@{@▯@
                                                                                                L- in th
```

```
[04/08/20]seed@VM:~/.../task$ cat plaintextofb1.txt
Parents need to know that although this underdog sports movie is based on the Jamaica@ bobsled team's appearance in
```

I believe my hypothesis was backwards.

Task 6: Experimenting with the initialization vector (IV)

Task 6.1:

We begin this task by checking the uniqueness of the IV importance. If we use the same key for many things, you will be able to crack into many things that the user may have encrypted with the same code.
My observation Is that the encoding with a different encryption IV will change the look of the file within.

```
[04/08/20]seed@VM:~/.../lab09$ openssl enc -aes-128-ofb -e -in plaintext.txt -out plaintext2.txt -K 00112233445566778889aab
bccddeeff -iv 0102030405060708
[04/08/20]seed@VM:~/.../lab09$ openssl enc -aes-128-ofb -e -in plaintext.txt -out plaintext3.txt -K 00112233445566778889aab
bccddeeff -iv 010203
[04/08/20]seed@VM:~/.../lab09$ openssl enc -aes-128-ecb -e -in plaintext.txt -out plaintext3.txt -K 00112233445566778889aab
bccddeeff -iv 010203
warning: iv not use by this cipher
[04/08/20]seed@VM:~/.../lab09$ openssl enc -aes-128-ofb -e -in plaintext.txt -out plaintext4.txt -K 00112233445566778889aab
bccddeeff -iv 010203
[04/08/20]seed@VM:~/.../lab09$ cat plaintext4.txt
@@~Cm-@@@/▯2r▯@;@@4-KO@@Uj@Y▯▯@@@@▯@@L@Ye@e|▯@ @T@@3@@@@N@gB@LAQ@@@▯@mkrş@▯@▯▯▯_@{
                                                                                @@e▯h@G@@@
6@-1p,▯@5O@I=M@▯N@:@@9@@@jz@~@Y@@@U'|@@0@@@@@@@@@@9ģgn@o@@vfS▯@@p▫▯▯▯q@qBhPE0"▯@@+▯@6@UJ▯@@▯@@@@@

@@/7@@@@:|-@@OG@@@c@0y~@v@R▯h@]@<ȷ@UY\@@▯@vk
```

Task 6.2

In task 6.2 we are asked to do what's known as a known-text attack. We are given the secret message and key for that particular message, but we are given a hexed message and an educated guess that our own Bob has used the same encryption IV. We want to decrypt using those pieces of knowledge we have in order to find the missing information.

```
[04/08/20]seed@VM:~/.../task6$ xxd -p p1
546869732069732061206b6e6f776e206d65737361676521
[04/08/20]seed@VM:~/.../task6$ xxd -p p2
xxd: p2: No such file or directory
[04/08/20]seed@VM:~/.../task6$ xxd -p c1
6134363962316335303263316361623936363936356535303432353433338
653162623162356639303337613463313539313 3
[04/08/20]seed@VM:~/.../task6$ man xxd
[04/08/20]seed@VM:~/.../task6$ xxd -p p1
546869732069732061206b6e6f776e206d65737361676521
[04/08/20]seed@VM:~/.../task6$ sudo python3 xor.py 546869732069732061206b6e6f776e206d65737361676521\a469b1c502c1cab966965e5
0425438e1bb1b5f9037a4c15913
Traceback (most recent call last):
  File "xor.py", line 5, in <module>
    script, first, second = argv
ValueError: not enough values to unpack (expected 3, got 2)
[04/08/20]seed@VM:~/.../task6$ sudo python3 xor.py 546869732069732061206b6e6f776e206d65737361676521 \ a469b1c502c1cab966965
e50425438e1bb1b5f9037a4c15913
hi
f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478[04/08/20]seed@VM:~/.../task6$ sudo python3 xor.py bf73bcd3509299d566c35b5d
6353e2d2356c1d67e2ce356c3a478f001d8b622a8b99907b
hi
4f726465723a204c61756e63682061206d697373696c6521[04/08/20]seed@VM:~/.../task6$ echo -n "4f726465723a204c61756e63682061206d6
97373696c6521" | xxd -r -p
Order: Launch a missile![04/08/20]seed@VM:~/.../task6$ .
```

Lets launch the missile, after all, it is an order!

Task 6.3

In this task we are asked to perform a known-text attack that takes an educated guess, and turns it into a possibly known answer.
We start by guessing to see a vote that bob wants, so we pad a message and begin the experiment. Our goal is to try and have the same code length outcome from a hexdump which gives us the educated guess that things are what we originally think.
Here are the steps.

```
-id-aes192-wrap           -id-aes256-CCM            -id-aes256-GCM
-id-aes256-wrap           -id-smime-alg-CMS3DESwrap  -rc2
-rc2-40-cbc               -rc2-64-cbc               -rc2-cbc
-rc2-cfb                  -rc2-ecb                  -rc2-ofb
-rc4                      -rc4-40                   -rc4-hmac-md5
-seed                     -seed-cbc                 -seed-cfb
-seed-ecb                 -seed-ofb
[04/12/20]seed@VM:~/.../task6$ openssl enc -aes-128-cfb -d -in plaintextcfb.txt -out plaintextcfb1.txt -K 00112233445566778
889aabbccddeeff -iv 010203
plaintextcfb.txt: No such file or directory
3070756544:error:02001002:system library:fopen:No such file or directory:bss_file.c:398:fopen('plaintextcfb.txt','r')
3070756544:error:20074002:BIO routines:FILE_CTRL:system lib:bss_file.c:400:
[04/12/20]seed@VM:~/.../task6$ echo -n "John smith......" > bob
[04/12/20]seed@VM:~/.../task6$ openssl enc -aes-128-cfb -d -in bob -out bobout -K 00112233445566778889aabbccddeeff -iv 1234
567890123456
[04/12/20]seed@VM:~/.../task6$ echno -n 1234567890123456 | xxd -r -p > ivBob
No command 'echno' found, did you mean:
 Command 'echo' from package 'coreutils' (main)
echno: command not found
[04/12/20]seed@VM:~/.../task6$ echo -n 1234567890123456 | xxd -r -p > ivBob
[04/12/20]seed@VM:~/.../task6$ md5sum ivBob
f0dd73b2b97e8856c67e50184121d763  ivBob
[04/12/20]seed@VM:~/.../task6$ echo -n "John smith......" > p1Guessed
[04/12/20]seed@VM:~/.../task6$ xxd -p p1Guessed
4a6f686e20736d6974682e2e2e2e2e2e
[04/12/20]seed@VM:~/.../task6$ sudo python3 xor.py 4a6f686e20736d6974682e2e2e2e2e2e \ 1234567890123456
hi
585b3e16b061593f[04/12/20]seed@VM:~/.../task6$ sudo python3 xor.py f0dd73b2b97e8856c67e50184121d763 \ 313233343536373839303
13233343537
hi
c1ef40868c48bf6eff4e612a7215e254[04/12/20]seed@VM:~/.../task6$ echo "c1ef40868c48bf6eff4e612a7215e254" | xxd -r -p > p2
[04/12/20]seed@VM:~/.../task6$ openssl enc -aes-128-cbc -e -in p2 -out c2 -K 00112233445566778899aabbcceeddff -iv f0dd73b2b
97e8856c67e50184121d763
[04/12/20]seed@VM:~/.../task6$ xxd -p c1
6134363962316335303263316361623936363936356535303432353433338
653162623162356639303337613463313539313 3
[04/12/20]seed@VM:~/.../task6$ xxd -p c2
b37bb0df118a0bdfddb675b52d12089c9a0c4b44b20792356c988bde3792
```

As you can see we end up with same length outputs, thus confirming what we think. Bob voted for "john smith"

Justin