D0029E - Symmetric Encryption (Lab 2)

Martin Askolin*

Luleå tekniska universitet
971 87 Luleå, Sverige

12 september 2021

*email: marsak-8@student.ltu.se

1 Frequency Analysis

The frequency analysis of the text was made by a website. Started with three letter words like 'the', 'and', and 'too'. After that 'i' and 's' was established from two lettered words like 'is', and 'it'. The other letters could be guesses from context and further look at the frequency analysis.

Results

Code	Count	Frequency
YTN	79	2.011 %
VUP	30	0.764 %
NQY	22	0.560 %
PYT	20	0.509 %
MUR	20	0.509 %
YNH	18	0.458 %
XZY	16	0.407 %
NHN	16	0.407 %
NUY	14	0.356 %
MXU	14	0.356 %
YTV	14	0.356 %

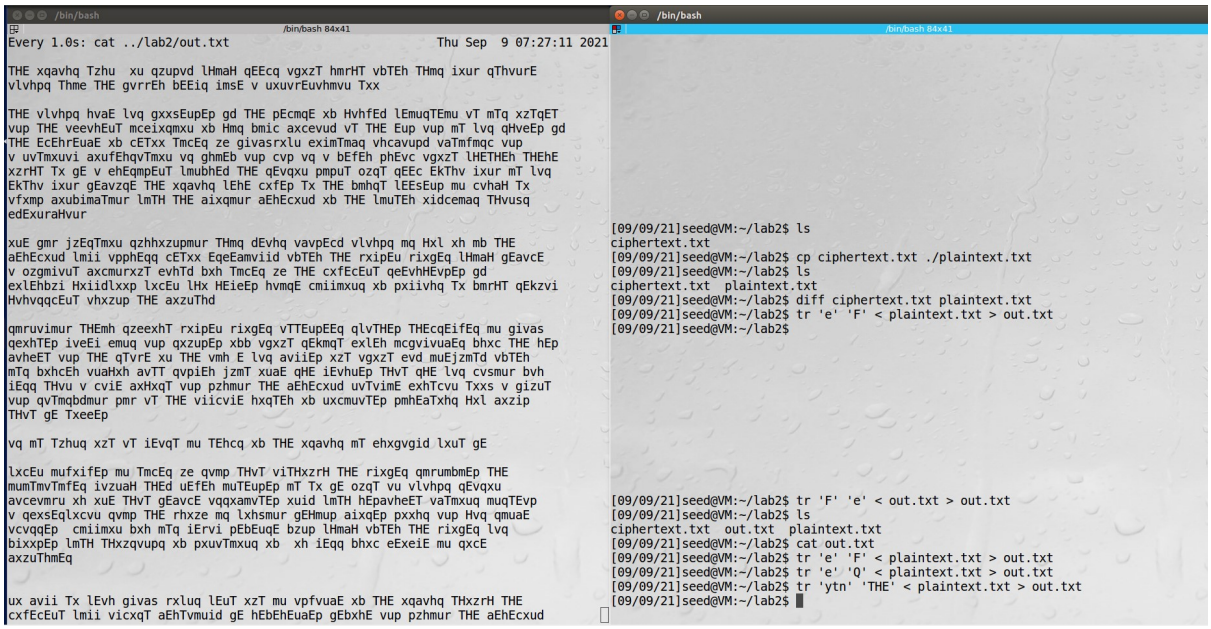
Comparison

Language

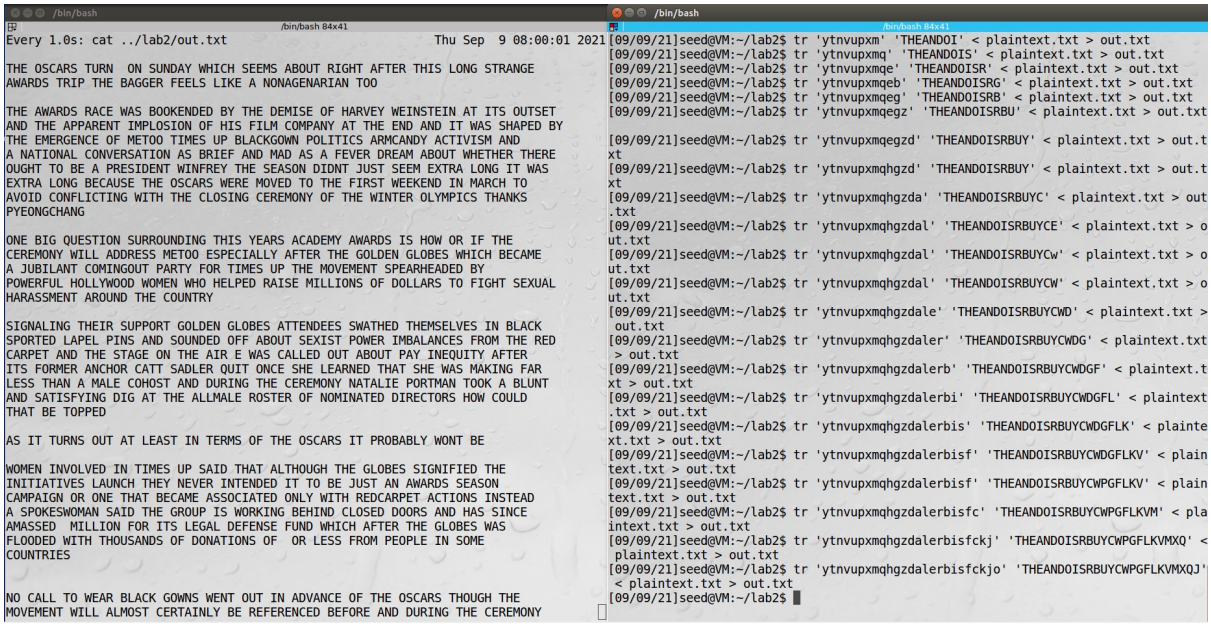
English

Code	Frequency
THE	1.897 %
AND	0.944 %
ING	0.762 %
HER	0.584 %
THA	0.470 %
HAT	0.445 %
ERE	0.405 %
HIS	0.392 %
ENT	0.335 %
YOU	0.328 %

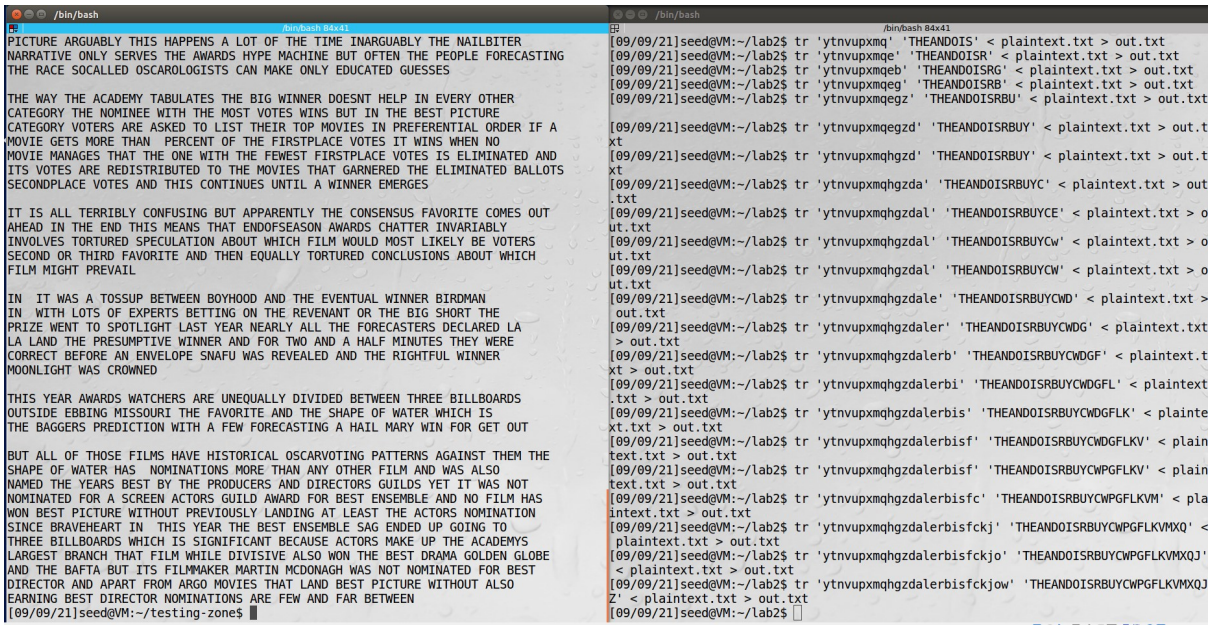
Figur 1: The Frequency Analysis.



Figur 2: Bash set up, initial testing, and establishing 'THE'



Figur 3: Top portion of text completed and 25 out of 26 characters established

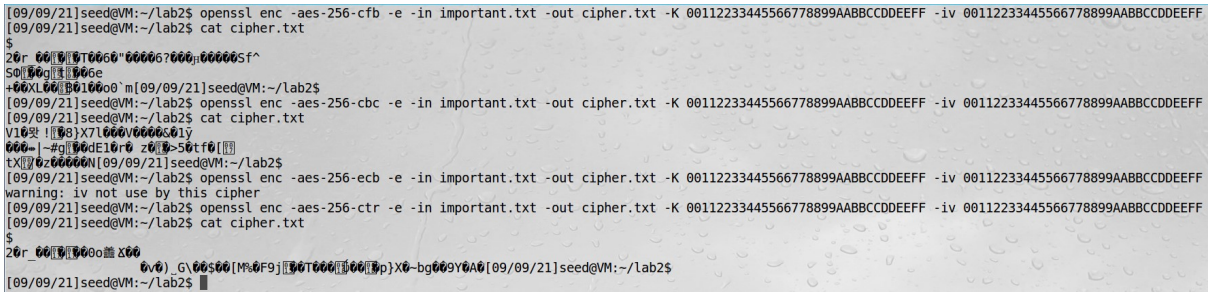


```
/bin/bash
/bin/bash 84x41
PICTURE ARGUABLY THIS HAPPENS A LOT OF THE TIME INARGUABLY THE NAILBITER
NARRATIVE ONLY SERVES THE AWARDS HYPE MACHINE BUT OFTEN THE PEOPLE FORECASTING
THE RACE SOCALLED OSCAROLOGISTS CAN MAKE ONLY EDUCATED GUESSES

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
PRIZE 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

THIS YEAR AWARDS WATCHERS ARE UNEQUALLY DIVIDED BETWEEN THREE BILLBOARDS
OUTSIDE EBBING MISSOURI THE FAVORITE AND THE SHAPE OF WATER WHICH IS
THE BAGGERS PREDICTION WITH A FEW FORECASTING A HAIL MARY WIN FOR GET OUT

BUT ALL OF THOSE FILMS HAVE HISTORICAL OSCARVOTING PATTERNS AGAINST THEM THE
SHAPE OF WATER HAS  NOMINATIONS MORE THAN ANY OTHER FILM AND WAS ALSO
NAMED THE YEARS BEST BY THE PRODUCERS AND DIRECTORS GUILDS YET IT WAS NOT
NOMINATED FOR A SCREEN ACTORS GUILD AWARD FOR BEST ENSEMBLE AND NO FILM HAS
WON BEST PICTURE WITHOUT PREVIOUSLY LANDING AT LEAST THE ACTORS NOMINATION
SINCE BRAVEHEART IN  THIS YEAR THE BEST ENSEMBLE SAG ENDED UP GOING TO
THREE BILLBOARDS WHICH IS SIGNIFICANT BECAUSE ACTORS MAKE UP THE ACADEMYS
LARGEST BRANCH THAT FILM WHILE DIVISIVE ALSO WON THE BEST DRAMA GOLDEN GLOBE
AND THE BAFTA BUT ITS FILMMAKER MARTIN MCDONAGH WAS NOT NOMINATED FOR BEST
DIRECTOR AND APART FROM ARGO MOVIES THAT LAND BEST PICTURE WITHOUT ALSO
EARNING BEST DIRECTOR NOMINATIONS ARE FEW AND FAR BETWEEN
[09/09/21]seed@VM:~/testing-zone$

/bin/bash
/bin/bash 84x41
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmq' 'THEANDOIS' < plaintext.txt > out.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqe' 'THEANDOISR' < plaintext.txt > out.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqeb' 'THEANDOISRG' < plaintext.txt > out.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqeg' 'THEANDOISRB' < plaintext.txt > out.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqegz' 'THEANDOISRBU' < plaintext.txt > out.txt

[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqegzd' 'THEANDOISRBUY' < plaintext.txt > out.t
xt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzd' 'THEANDOISRBUY' < plaintext.txt > out.t
xt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzda' 'THEANDOISRBUYC' < plaintext.txt > out
.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzdal' 'THEANDOISRBUYCE' < plaintext.txt > o
ut.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzdal' 'THEANDOISRBUYCw' < plaintext.txt > o
ut.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzdal' 'THEANDOISRBUYCW' < plaintext.txt > o
ut.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzdale' 'THEANDOISRBUYCWD' < plaintext.txt >
 out.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzdaler' 'THEANDOISRBUYCWDG' < plaintext.txt
 > out.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzdalerb' 'THEANDOISRBUYCWDGF' < plaintext.t
xt > out.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzdalerbi' 'THEANDOISRBUYCWDGFL' < plaintext
.txt > out.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzdalerbis' 'THEANDOISRBUYCWDGFLK' < plainte
xt.txt > out.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzdalerbisf' 'THEANDOISRBUYCWDGFLKV' < plain
text.txt > out.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzdalerbisf' 'THEANDOISRBUYCWPGFLKV' < plain
text.txt > out.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzdalerbisfc' 'THEANDOISRBUYCWPGFLKVM' < pla
intext.txt > out.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzdalerbisfckj' 'THEANDOISRBUYCWPGFLKVMXQ' <
 plaintext.txt > out.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzdalerbisfckjo' 'THEANDOISRBUYCWPGFLKVMXQJ'
 < plaintext.txt > out.txt
[09/09/21]seed@VM:~/lab2$ tr 'ytnvupxmqhgzdalerbisfckjow' 'THEANDOISRBUYCWPGFLKVMXQJ
Z' < plaintext.txt > out.txt
[09/09/21]seed@VM:~/lab2$
```

Figur 4: Bottom portion of text completed and all characters established

2 Encryption using Different Cipher and Modes

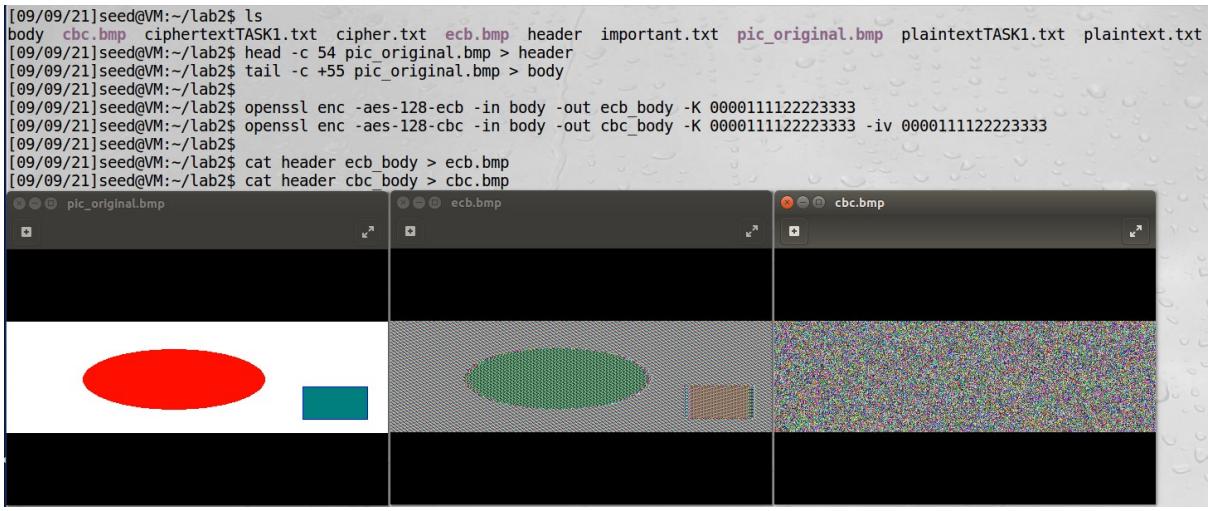
During this task we wondered why they (in the example) had used a 32 byte (256 bit) string of characters as the key and iv when it should have been a 128 bit key and iv. We later realized that the input should be in hex and not ascii. However this did not stop us from encrypting with AES-256 using a 128 bit key before later realizing our mistake.



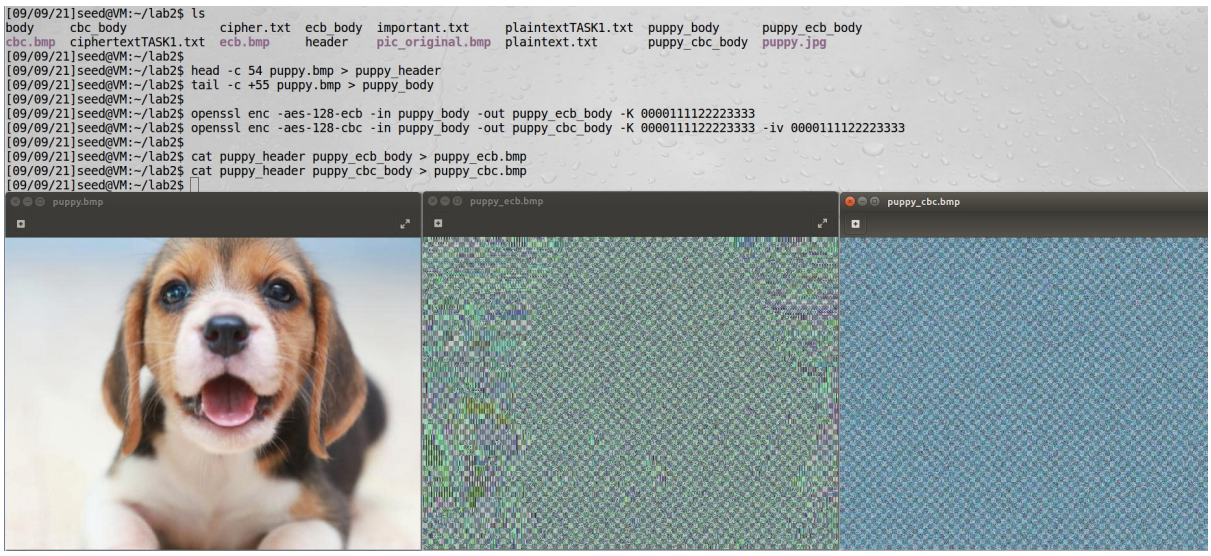
```
[09/09/21]seed@VM:~/lab2$ openssl enc -aes-256-cfb -e -in important.txt -out cipher.txt -K 00112233445566778899AABBCCDDEEFF -iv 00112233445566778899AABBCCDDEEFF
[09/09/21]seed@VM:~/lab2$ cat cipher.txt
$
2r ⍰⍰⍰⍰T⍰⍰6⍰"⍰⍰⍰⍰6?⍰⍰⍰µ⍰⍰⍰⍰⍰Sf^
S⍰⍰⍰g⍰⍰⍰⍰6e
+⍰⍰XL⍰⍰⍰⍰⍰1⍰⍰o0`m[09/09/21]seed@VM:~/lab2$
[09/09/21]seed@VM:~/lab2$ openssl enc -aes-256-cbc -e -in important.txt -out cipher.txt -K 00112233445566778899AABBCCDDEEFF -iv 00112233445566778899AABBCCDDEEFF
[09/09/21]seed@VM:~/lab2$ cat cipher.txt
V1⍰º!⍰⍰8}X7l⍰⍰⍰V⍰⍰⍰⍰&⍰1ÿ
⍰⍰⍰⍰|~#g⍰⍰⍰⍰dE1⍰r⍰ z⍰⍰⍰⍰>5⍰tf⍰[⍰⍰
tX⍰⍰⍰z⍰⍰⍰⍰⍰N[09/09/21]seed@VM:~/lab2$
[09/09/21]seed@VM:~/lab2$ openssl enc -aes-256-ecb -e -in important.txt -out cipher.txt -K 00112233445566778899AABBCCDDEEFF -iv 00112233445566778899AABBCCDDEEFF
warning: iv not use by this cipher
[09/09/21]seed@VM:~/lab2$ openssl enc -aes-256-ctr -e -in important.txt -out cipher.txt -K 00112233445566778899AABBCCDDEEFF -iv 00112233445566778899AABBCCDDEEFF
[09/09/21]seed@VM:~/lab2$ cat cipher.txt
$
2⍰r_⍰⍰⍰⍰⍰⍰⍰⍰0o⍰⍰⍰⍰⍰⍰⍰⍰
```

3 Encryption mode - ECB vs. CBC

A lot of the key characteristics from pic_original.bmp can still be viewed from the ECB encrypted version while the CBC version has almost turned into an indistinguishable white noise / TV noise. The explanation for this is that CBC's cipher blocks depends on all earlier plaintext blocks (requiring the use of an initialization vector) while ECB's cipher blocks do not. This removes any type of pattern that might have been visible in the encrypted image.



Figur 6: Display of code for encrypting the image and the result.



Figur 7: Repeated experiment with puppy image

4 Padding

4.1 Block ciphers and Stream ciphers

ECB and CBC requires padding while CFB and OFB does not. This is because the latter are block ciphers and the former are stream ciphers. The reason why CFB and OFB are stream ciphers is because the plaintext to encrypt is XORed with it's corresponding block cipher encryption (16 bytes) and since XOR is a bit-wise operation the plaintext's length is inconsequential.

```
[09/09/21]seed@VM:~/lab2$ echo '0' > one.txt
[09/09/21]seed@VM:~/lab2$
[09/09/21]seed@VM:~/lab2$ openssl enc -aes-128-ecb -e -in one.txt -out ecb -K 0000111122223333
[09/09/21]seed@VM:~/lab2$ openssl enc -aes-128-cbc -e -in one.txt -out cbc -K 0000111122223333 -iv 0000111122223333
[09/09/21]seed@VM:~/lab2$ openssl enc -aes-128-cfb -e -in one.txt -out cfb -K 0000111122223333 -iv 0000111122223333
[09/09/21]seed@VM:~/lab2$ openssl enc -aes-128-ofb -e -in one.txt -out ofb -K 0000111122223333 -iv 0000111122223333
[09/09/21]seed@VM:~/lab2$
[09/09/21]seed@VM:~/lab2$ wc -c < ecb
16
[09/09/21]seed@VM:~/lab2$ wc -c < cbc
16
[09/09/21]seed@VM:~/lab2$ wc -c < cfb
2
[09/09/21]seed@VM:~/lab2$ wc -c < ofb
2
```

Figur 8: Cipher text length for different modes.

4.2 Inner working of padding

```
[09/09/21]seed@VM:~/lab2$ echo -n "12345" > five.txt
[09/09/21]seed@VM:~/lab2$ echo -n "1234567890" > ten.txt
[09/09/21]seed@VM:~/lab2$ echo -n "1234567890123456" > sixteen.txt
[09/09/21]seed@VM:~/lab2$
[09/09/21]seed@VM:~/lab2$ openssl enc -aes-128-cbc -e -in five.txt -out c_five.txt -K 0000111122223333 -iv 0000111122223333
[09/09/21]seed@VM:~/lab2$ openssl enc -aes-128-cbc -e -in ten.txt -out c_ten.txt -K 0000111122223333 -iv 0000111122223333
[09/09/21]seed@VM:~/lab2$ openssl enc -aes-128-cbc -e -in sixteen.txt -out c_sixteen.txt -K 0000111122223333 -iv 0000111122223333
[09/09/21]seed@VM:~/lab2$
[09/09/21]seed@VM:~/lab2$ ls -l five.txt
-rw-rw-r-- 1 seed seed 5 Sep  9 09:23 five.txt
[09/09/21]seed@VM:~/lab2$ wc -c < five.txt
5
[09/09/21]seed@VM:~/lab2$ wc -c < c_five.txt
16
[09/09/21]seed@VM:~/lab2$ wc -c < ten.txt
10
[09/09/21]seed@VM:~/lab2$ wc -c < c_ten.txt
16
[09/09/21]seed@VM:~/lab2$ wc -c < sixteen.txt
16
[09/09/21]seed@VM:~/lab2$ wc -c < c_sixteen.txt
32
```

Figur 9: Creation, encryption and display of length for 5, 10, and 16 byte files

```
[09/09/21]seed@VM:~/lab2$ openssl enc -aes-128-cbc -d -nopad -in c_five.txt -out r_five.txt -K 0000111122223333 -iv 0000111122223333
[09/09/21]seed@VM:~/lab2$ openssl enc -aes-128-cbc -d -nopad -in c_ten.txt -out r_ten.txt -K 0000111122223333 -iv 0000111122223333
[09/09/21]seed@VM:~/lab2$ openssl enc -aes-128-cbc -d -nopad -in c_sixteen.txt -out r_sixteen.txt -K 0000111122223333 -iv 0000111122223333
[09/09/21]seed@VM:~/lab2$
[09/09/21]seed@VM:~/lab2$ hexdump -C r_five.txt
00000000  31 32 33 34 35 0b 0b 0b  0b 0b 0b 0b 0b 0b 0b 0b  |12345...........|
00000010
[09/09/21]seed@VM:~/lab2$ hexdump -C r_ten.txt
00000000  31 32 33 34 35 36 37 38  39 30 06 06 06 06 06 06  |1234567890......|
00000010
[09/09/21]seed@VM:~/lab2$ hexdump -C r_sixteen.txt
00000000  31 32 33 34 35 36 37 38  39 30 31 32 33 34 35 36  |1234567890123456|
00000010  10 10 10 10 10 10 10 10  10 10 10 10 10 10 10 10  |................|
00000020
[09/09/21]seed@VM:~/lab2$
```

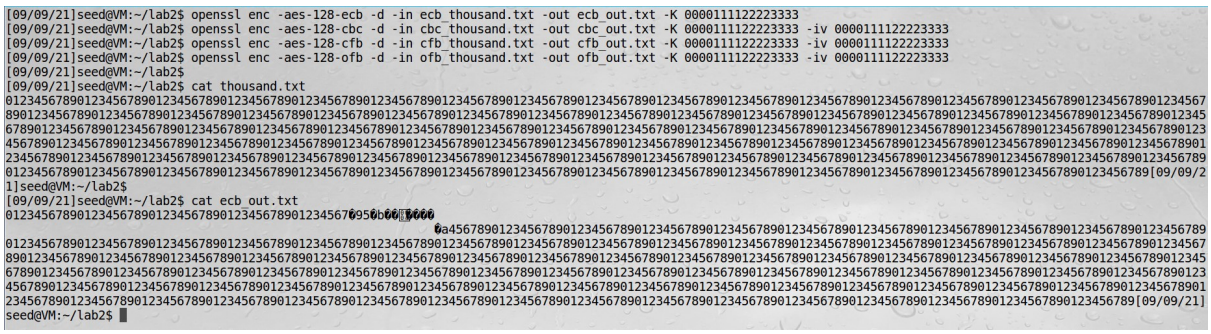
Figur 10: Decryption of files without removing the padding.

As can be seen from Figure 10 the padding displayed is the hex value for the size (in bytes) of padding added to the file. This is the product of PKCS#5. It can also be noted from the picture that padding is always added. If the file is a multiple of 16 bytes it will still append a whole 16 byte block of padding.

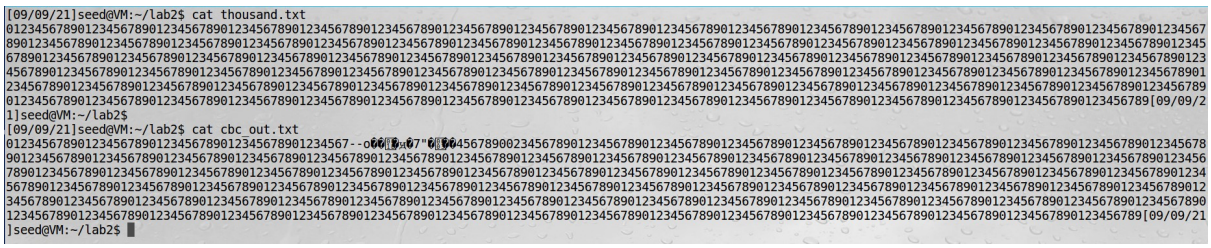
5 Error propagation - Corrupted cipher text

The damage will depend on if the bit is consequential or inconsequential to decrypting it's own cipher block or other cipher blocks. My guess is that for

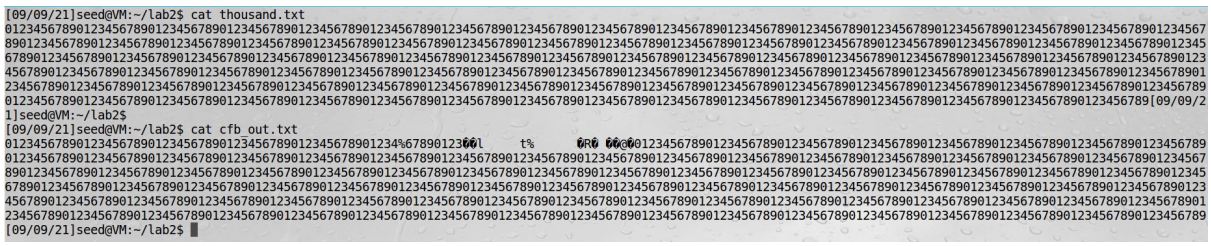
- ECB it will destroy it's own cipher block.
- CBC it will destroy it's own cipher block and XOR one bit wrong in the next block.
- CFB it will XOR one bit wrong in the prior block and destroy it's own.
- OFB it will only XOR one bit wrong in it's own block.



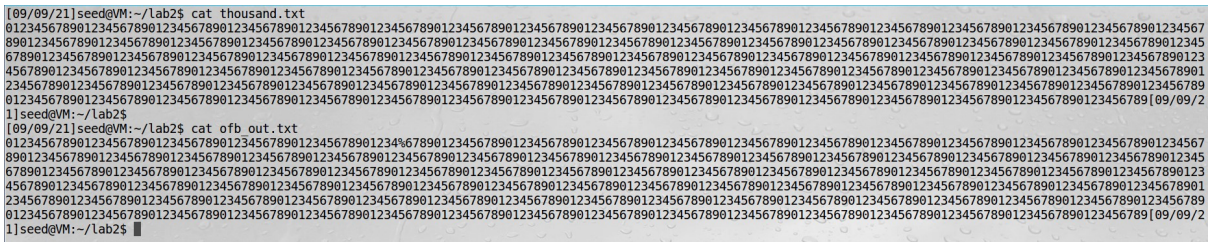
Figur 11: Display of plaintext file and ECB one bit error decrypted file



Figur 12: Display of plaintext file and CBC one bit error decrypted file



Figur 13: Display of plaintext file and CFB one bit error decrypted file

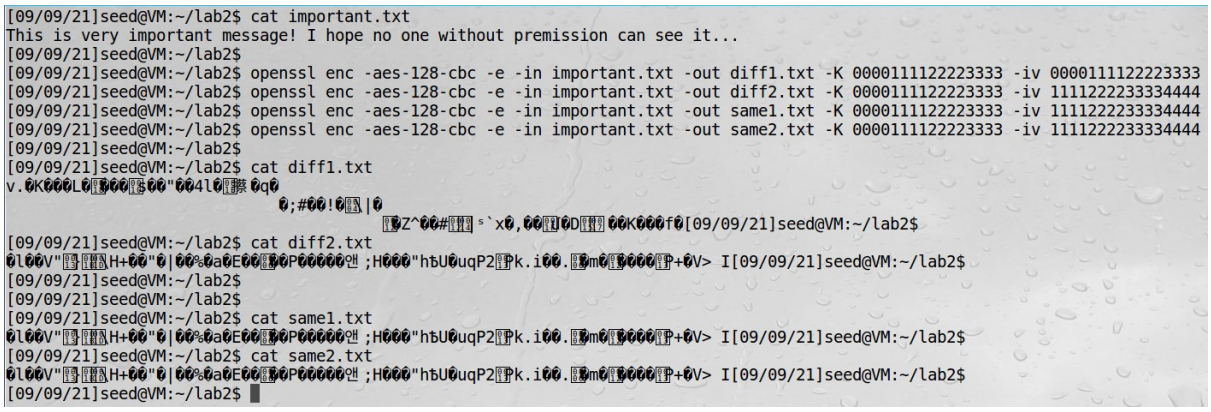


Figur 14: Display of plaintext file and OFB one bit error decrypted file

6 Initial Vector (IV) - Common Mistakes

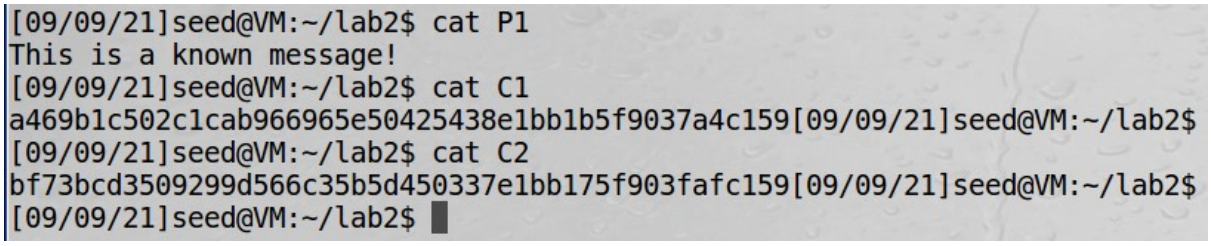
6.1 Identical Initial Vectors

If IVs are not unique then two instances of an identical message will be mapped to the same cipher leading to potential data leaks. This problem is even worse for some modes such as OFB where the known-plaintext attack is possible rendering all plaintexts ecrypted using that key and iv fully exposed.



Figur 15: Display of one message encrypted in four ways; two of them with the same IV and two with different IVs.

6.2 The known-plaintext attack (Same IVs)



Figur 16: Set up for attack

```
[09/09/21]seed@VM:~/lab2$ xxd -p P1
546869732069732061206b6e6f776e206d657373616765210a
[09/09/21]seed@VM:~/lab2$ cat C1
a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159[09/09/21]seed@VM:~/lab2$
[09/09/21]seed@VM:~/lab2$
[09/09/21]seed@VM:~/lab2$ xor.py 546869732069732061206b6e6f776e206d657373616765210a a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159
bash: ./xor.py: Permission denied
[09/09/21]seed@VM:~/lab2$ chmod u+x xor.py
[09/09/21]seed@VM:~/lab2$ xor.py 546869732069732061206b6e6f776e206d657373616765210a a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159
f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478
[09/09/21]seed@VM:~/lab2$
[09/09/21]seed@VM:~/lab2$ cat C2
bf73bcd3509299d566c35b5d450337e1bb175f903fafc159[09/09/21]seed@VM:~/lab2$
[09/09/21]seed@VM:~/lab2$
[09/09/21]seed@VM:~/lab2$ xor.py f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478 bf73bcd3509299d566c35b5d450337e1bb175f903fafc159
4f726465723a204c61756e63682061206d697373696c6521
[09/09/21]seed@VM:~/lab2$
[09/09/21]seed@VM:~/lab2$ echo 4f726465723a204c61756e63682061206d697373696c6521 | xxd -r -p > P2
[09/09/21]seed@VM:~/lab2$
[09/09/21]seed@VM:~/lab2$ cat P2
Order: Launch a missile![09/09/21]seed@VM:~/lab2$
[09/09/21]seed@VM:~/lab2$
```

Figur 17: Attack + display of unknown message P2

6.3 Chosen-plaintext attack (Predictable IVs)

The operations required to do a chosen-plaintext attack is pretty straight forward when it comes to CBC. We XOR our guess with the IV of bobs message, after that we XOR the result with our own IV. We then turn it into ascii and give it to bob for encryption. Since the mode is CBC the plaintext blocks will be XORed with the IV (undoing our initial XOR with the same IV) making the block cipher encryption identical to bobs and as a result gives us a identical matching cipher to bobs (if the message is the same). However.. during this task we ran into a wall. We knew that the aes-128-cbc would append PKCS#5 padding but failed to think of that before constructing our own guesses. This costed us hours and images of these attempts are appended after the successful ones.

```python
#!/usr/bin/python3
from sys import argv
script, first, second = argv
aa = bytearray.fromhex(first)
bb = bytearray.fromhex(second)
xord = bytearray(x^y for x,y in zip(aa, bb))
print(xord.hex())
```

Figur 18: code for xor.py

```
[09/10/21]seed@VM:~/.../hackingbob$ echo '5965730d0d0d0d0d0d0d0d0d0d0d0d0d'
5965730d0d0d0d0d0d0d0d0d0d0d0d0d
[09/10/21]seed@VM:~/.../hackingbob$ xor.py 5965730d0d0d0d0d0d0d0d0d0d0d0d0d "$(cat iv1)"
68574039383b3a35343d3c3f3e39383b
[09/10/21]seed@VM:~/.../hackingbob$ xor.py 68574039383b3a35343d3c3f3e39383b "$(cat iv2)" | xxd -p -r > P2_yes
[09/10/21]seed@VM:~/.../hackingbob$
[09/10/21]seed@VM:~/.../hackingbob$ openssl enc -aes-128-cbc -e -in P2_yes -out C2_yes -K "$(cat key)" -iv "$(cat iv2)"
[09/10/21]seed@VM:~/.../hackingbob$
[09/10/21]seed@VM:~/.../hackingbob$ cat C1
bef65565572ccee2a9f9553154ed9498
[09/10/21]seed@VM:~/.../hackingbob$ xxd -p C2_out
xxd: C2_out: No such file or directory
[09/10/21]seed@VM:~/.../hackingbob$ xxd -p C2_yes
bef65565572ccee2a9f9553154ed94983402de3f0dd16ce789e5475779ac
a405
```

Figur 19: Successful attack

```
[09/09/21]seed@VM:~/.../hackingbob$ echo 'C1, key, IV1, IV2 are all stored as hex'
C1, key, IV1, IV2 are all stored as hex
[09/09/21]seed@VM:~/.../hackingbob$ ls
C1  IV1  IV2  key  no  xor.py  yes
[09/09/21]seed@VM:~/.../hackingbob$
[09/09/21]seed@VM:~/.../hackingbob$
[09/09/21]seed@VM:~/.../hackingbob$ xor.py "$(xxd -p yes)" "$(cat IV1)" > yes_temp
[09/09/21]seed@VM:~/.../hackingbob$ xor.py "$(xxd -p no)" "$(cat IV1)" > no_temp
[09/09/21]seed@VM:~/.../hackingbob$
[09/09/21]seed@VM:~/.../hackingbob$ xor.py "$(cat IV2)" "$(cat yes_temp)" | xxd -p -r > P2_yes
[09/09/21]seed@VM:~/.../hackingbob$ xor.py "$(cat IV2)" "$(cat no_temp)" | xxd -p -r > P2_no
[09/09/21]seed@VM:~/.../hackingbob$
[09/09/21]seed@VM:~/.../hackingbob$ openssl enc -aes-128-cbc -e -in P2_yes -out C2_yes -K "$(cat key)" -iv "$(cat IV2)"
[09/09/21]seed@VM:~/.../hackingbob$ openssl enc -aes-128-cbc -e -in P2_no -out C2_no -K "$(cat key)" -iv "$(cat IV2)"
[09/09/21]seed@VM:~/.../hackingbob$
[09/09/21]seed@VM:~/.../hackingbob$ cat C1
bef65565572ccee2a9f9553154ed9498[09/09/21]seed@VM:~/.../hackingbob$
[09/09/21]seed@VM:~/.../hackingbob$
[09/09/21]seed@VM:~/.../hackingbob$ xxd -p C2_yes
a9defedd16efeb764472e1d34da88d4f
[09/09/21]seed@VM:~/.../hackingbob$ xxd -p C2_no
fce0813b1039af4eabc0a7ab728cb9df
[09/09/21]seed@VM:~/.../hackingbob$
```

Figur 20: Unsuccessful attack because of padding and added new lines at end of XORs and hex transformations

```
[09/09/21]seed@VM:~/.../hackingbob$ xor.py "$(xxd -p yes)" "$(cat iv1)" | head -c -1 > yes_temp
[09/09/21]seed@VM:~/.../hackingbob$ xor.py "$(xxd -p no)" "$(cat iv1)" | head -c -1 > no_temp
[09/09/21]seed@VM:~/.../hackingbob$
[09/09/21]seed@VM:~/.../hackingbob$ xor.py "$(cat yes_temp)" "$(cat iv2)" | head -c -1 | xxd -p -r > P2_yes
[09/09/21]seed@VM:~/.../hackingbob$ xor.py "$(cat no_temp)" "$(cat iv2)" | head -c -1 | xxd -p -r > P2_no
[09/09/21]seed@VM:~/.../hackingbob$
[09/09/21]seed@VM:~/.../hackingbob$ openssl enc -aes-128-cbc -e -in P2_yes -out C2_yes -K "$(cat key)" -iv "$(cat iv2)"
[09/09/21]seed@VM:~/.../hackingbob$ openssl enc -aes-128-cbc -e -in P2_no -out C2_no -K "$(cat key)" -iv "$(cat iv2)"
[09/09/21]seed@VM:~/.../hackingbob$
[09/09/21]seed@VM:~/.../hackingbob$ cat C1
bef65565572ccee2a9f9553154ed9498
[09/09/21]seed@VM:~/.../hackingbob$ xxd -p C2_yes
3fd5f24f43d61f27e9539b963a3bf847
[09/09/21]seed@VM:~/.../hackingbob$ xxd -p C2_no
bb5f208ffe234c19634d069143bc9550
[09/09/21]seed@VM:~/.../hackingbob$
```

Figur 21: Unsuccessful attack because of padding

```
[09/10/21]seed@VM:~/.../hackingbob$ echo -n 'Yes............' > bobsyes
[09/10/21]seed@VM:~/.../hackingbob$
[09/10/21]seed@VM:~/.../hackingbob$ openssl enc -aes-128-cbc -e -in bobsyes -out bobsout -K "$(cat key)" -iv "$(cat iv1)"
[09/10/21]seed@VM:~/.../hackingbob$
[09/10/21]seed@VM:~/.../hackingbob$ xxd -p bobsout
cc99821bb09af95228e7763836a90791763f1ecda46d2824f8dd2086c53b
e7fa
[09/10/21]seed@VM:~/.../hackingbob$
[09/10/21]seed@VM:~/.../hackingbob$ xor.py "$(xxd -p bobsyes)" "$(cat iv1)" | head -c -1 > temp
[09/10/21]seed@VM:~/.../hackingbob$
[09/10/21]seed@VM:~/.../hackingbob$ xor.py "$(cat temp)" "$(cat iv2)"
5965732e2e2e2e2e2e2e2e2e2e2e2e2e2f
[09/10/21]seed@VM:~/.../hackingbob$
[09/10/21]seed@VM:~/.../hackingbob$ echo -n '5965732e2e2e2e2e2e2e2e2e2e2e2e2e2f' > temp2
[09/10/21]seed@VM:~/.../hackingbob$
[09/10/21]seed@VM:~/.../hackingbob$ echo -n '5965732e2e2e2e2e2e2e2e2e2e2e2e2e2f' | xxd -p -r > temp2
[09/10/21]seed@VM:~/.../hackingbob$
[09/10/21]seed@VM:~/.../hackingbob$ openssl enc -aes-128-cbc -e -in temp2  -out evesout -K "$(cat key)" -iv "$(cat iv2)"
[09/10/21]seed@VM:~/.../hackingbob$
[09/10/21]seed@VM:~/.../hackingbob$ xxd -p evesout
cc99821bb09af95228e7763836a90791763f1ecda46d2824f8dd2086c53b
e7fa
[09/10/21]seed@VM:~/.../hackingbob$
```

Figur 22: The attack that led us on the right track

7 Programming with Crypto Library

After playing around for about an hour in C (not making any progress) I decided to try and implement a solution using python instead. It took me about three hours of intense google:ing to get it working. The reason why the color scheme and appearance changed between prior tasks and this one is because I opted to program it on my laptop running Ubuntu (since yesterday (That's why the code is written in the text editor)) instead of the SEED virtual box.

```python
#!/usr/bin/python3

from sys import argv
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
from urllib import request

script, first, second, third = argv

plaintext = bytearray(first, encoding = 'utf-8')
ciphertext = bytearray.fromhex(second)
iv = bytearray.fromhex(third)

url = "https://seedsecuritylabs.org/Labs_16.04/Crypto/Crypto_Encryption/files/words.txt"
words = request.urlopen(url)
keys = words.read().splitlines()

for k in keys:
        if len(k) <= 16:
                key = k + bytes('#', 'utf-8') * (16 - len(k))
                cipher = AES.new(key, mode = AES.MODE_CBC, iv = iv)
                guess = cipher.encrypt(pad(plaintext, 16))

                if guess == ciphertext:
                        print('The key is:', key.decode('utf-8'))
                        exit(0)
```

Figur 23: Code

```
martin@lap-ma: ~/Develop/Cryptography/D0029E
martin@lap-ma:~/Develop/Cryptography/D0029E$ ls
ciphertext.txt  findkey.c  iv.txt  key.txt  plaintext.txt  task7.py
martin@lap-ma:~/Develop/Cryptography/D0029E$ python3 task7.py "$(cat plaintext.txt)" "$(cat ciphertext.txt)" "$(cat iv.txt)"
The key is: Syracuse########
martin@lap-ma:~/Develop/Cryptography/D0029E$
```

Figur 24: Running the python code