# Applied Cryptography (UE20CS314)

## Lab 3

Name: Vishwa Mehul Mehta

SRN: PES2UG20CS389

Section: F

### Task 1: Generate Encryption Key in a Wrong Way

Step 1:-

Code:

seed@VM: ~/.../lab3

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define KEYSIZE 16
void main() {
        int i;
        char key[KEYSIZE];
        printf("%lld\n", (long long) time(NULL));
        srand (time(NULL));
        for (i = 0; i< KEYSIZE; i++) {
                key[i] = rand()%128;
                printf("%.2x", (unsigned char)key[i]);
        }
        printf("\n");
}
~
~
~
~
~
~
~
~
"task1.c" 15L, 298C written
```

Output:

```
[09/27/22]seed@VM:~/.../lab3$ gcc task1.c -o task1
[09/27/22]seed@VM:~/.../lab3$ ./task1
1664250219
30146a371a257e2305557c2971695126
[09/27/22]seed@VM:~/.../lab3$ ./task1
1664250219
30146a371a257e2305557c2971695126
[09/27/22]seed@VM:~/.../lab3$ ./task1
1664250220
6146736b2d79791866414803474f1600
[09/27/22]seed@VM:~/.../lab3$
```

Observation:

A new key is generated every second as the seed value we pass is the time elapsed in seconds till that point.

Step 2:-

Code:

seed@VM: ~/.../lab3

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define KEYSIZE 16
void main() {
        int i;
        char key[KEYSIZE];
        printf("%lld\n", (long long) time(NULL));
        //srand (time(NULL));
        for (i = 0; i< KEYSIZE; i++) {
                key[i] = rand()%128;
                printf("%.2x", (unsigned char)key[i]);
        }
        printf("\n");
}
~
~
~
~
~
~
~
~
"task1.c" 15L, 300C written
```

Output:

```
[09/27/22]seed@VM:~/.../lab3$ gcc task1.c -o task1
[09/27/22]seed@VM:~/.../lab3$ ./task1
1664250261
67466973517f4a6c294d3a2b727b6346
[09/27/22]seed@VM:~/.../lab3$ ./task1
1664250262
67466973517f4a6c294d3a2b727b6346
[09/27/22]seed@VM:~/.../lab3$ ./task1
1664250262
67466973517f4a6c294d3a2b727b6346
[09/27/22]seed@VM:~/.../lab3$
```

Observation:

Here the same key is generated as there's no new seed value being passed to the rand function.

Task2: Guessing the key

Step 1:-

Output:

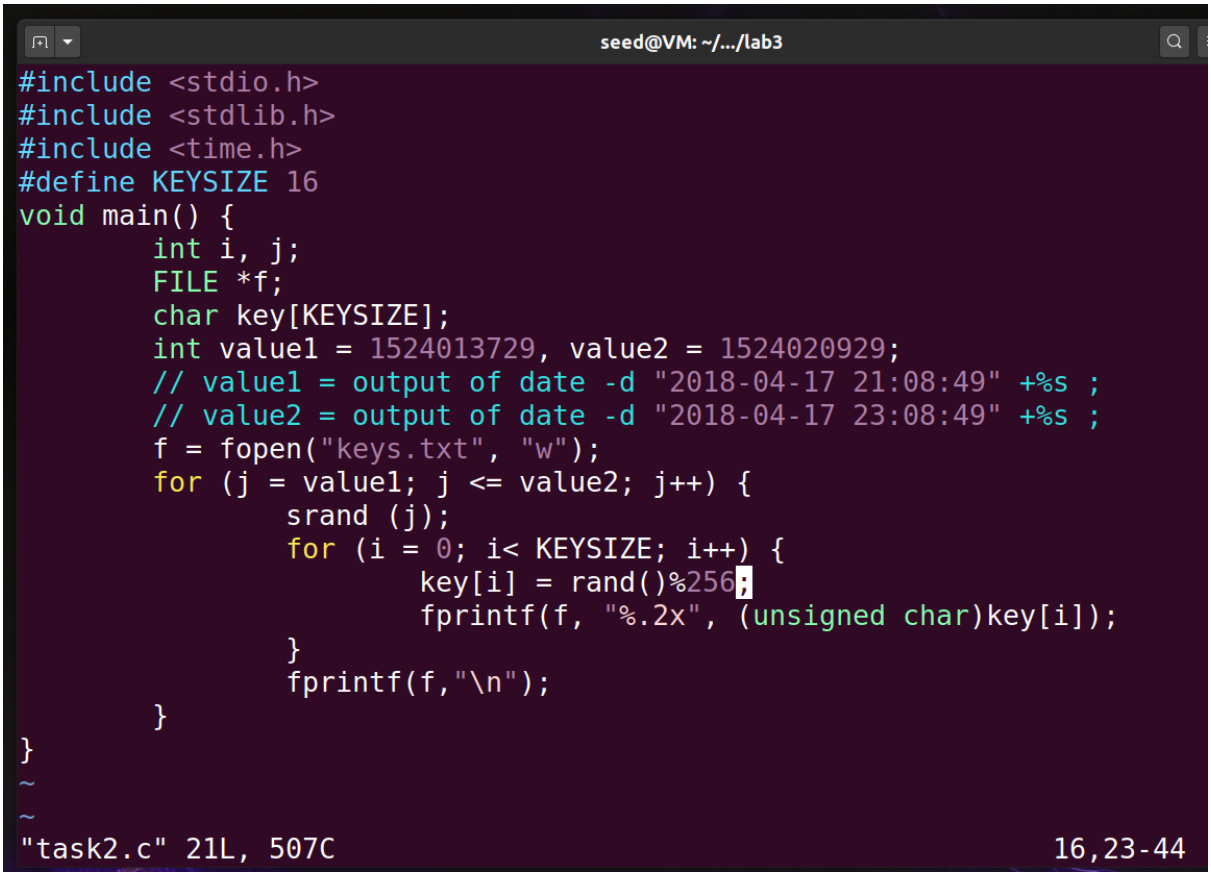
```
[09/27/22]seed@VM:~/.../lab3$ date -d "2018-04-17 21:08:49" +%s
1523979529
[09/27/22]seed@VM:~/.../lab3$ date -d "2018-04-17 23:08:49" +%s
1523986729
[09/27/22]seed@VM:~/.../lab3$
```

Observation:

The output is the time elapsed in seconds between the current time and the time specified in the argument.

Step 2:-

Code:


seed@VM: ~/.../lab3
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define KEYSIZE 16
void main() {
int i, j;
FILE *f;
char key[KEYSIZE];
int value1 = 1524013729, value2 = 1524020929;
// value1 = output of date -d "2018-04-17 21:08:49" +%s ;
// value2 = output of date -d "2018-04-17 23:08:49" +%s ;
f = fopen("keys.txt", "w");
for (j = value1; j <= value2; j++) {
srand (j);
for (i = 0; i< KEYSIZE; i++) {
key[i] = rand()%256;
fprintf(f, "%.2x", (unsigned char)key[i]);
}
fprintf(f,"\n");
}
}
"task2.c" 21L, 507C
16,23-44

Output:

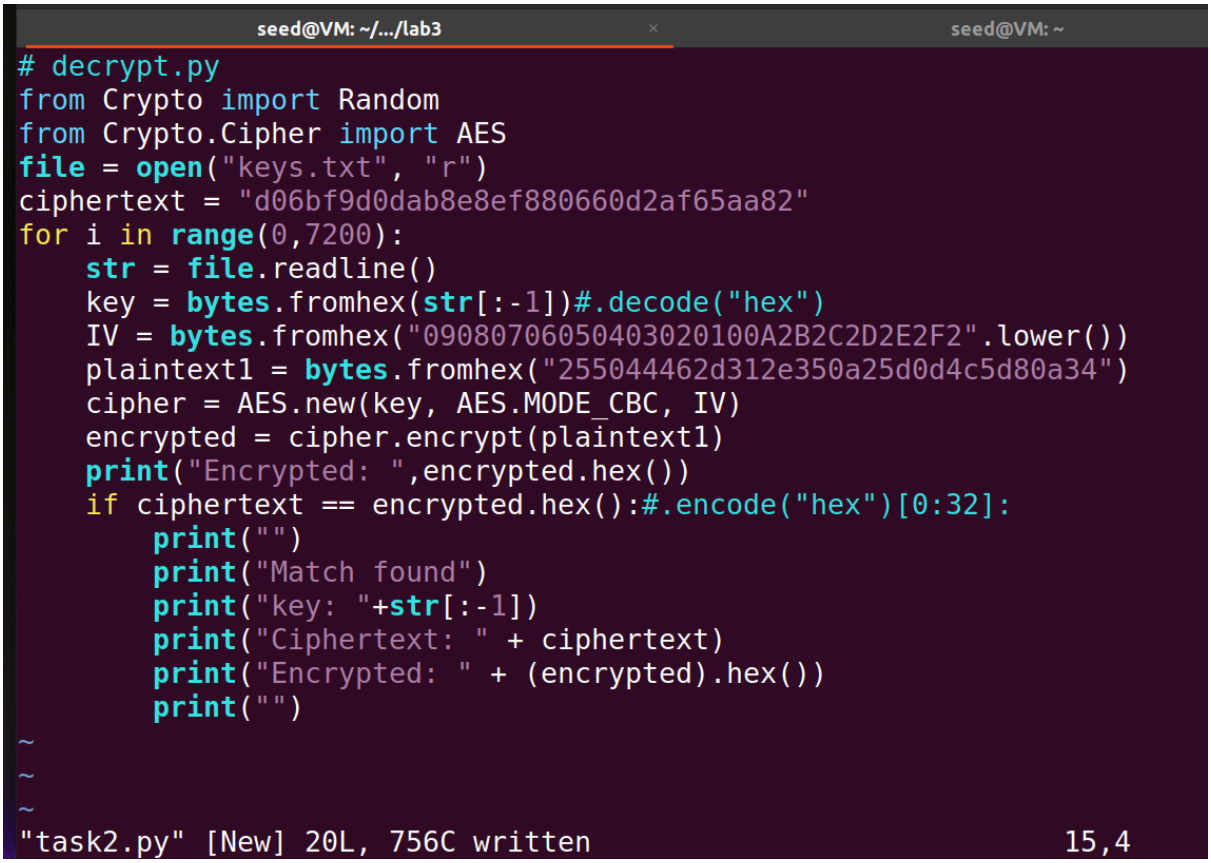

[09/27/22]seed@VM:~/.../lab3$ gcc task2.c -o task2
[09/27/22]seed@VM:~/.../lab3$ ./task2
[09/27/22]seed@VM:~/.../lab3$ head keys.txt
0a6226fc01a201b82b7d42caa7de3e05
12d494f3e5506c3fc152d68ae5d35bc8
64b838761768baa431899b84dc5bbed0
fd9b1b3ae04452506a7f269b77d95e8e
9a45a8c0eea61d185e2e896ea1e96167
bd858bd80cb81b68981fc6ab1f6b6ff0
405350cf2bf03e912e03bba28a2a3cc6
66b32d34c8315750343b34fbf329b8b5
794885b8757f4791ee06970ed2c1f92b
c270b9219acd47d50997e8404ef066f3
[09/27/22]seed@VM:~/.../lab3$ wc -l keys.txt
7201 keys.txt

## Observation:

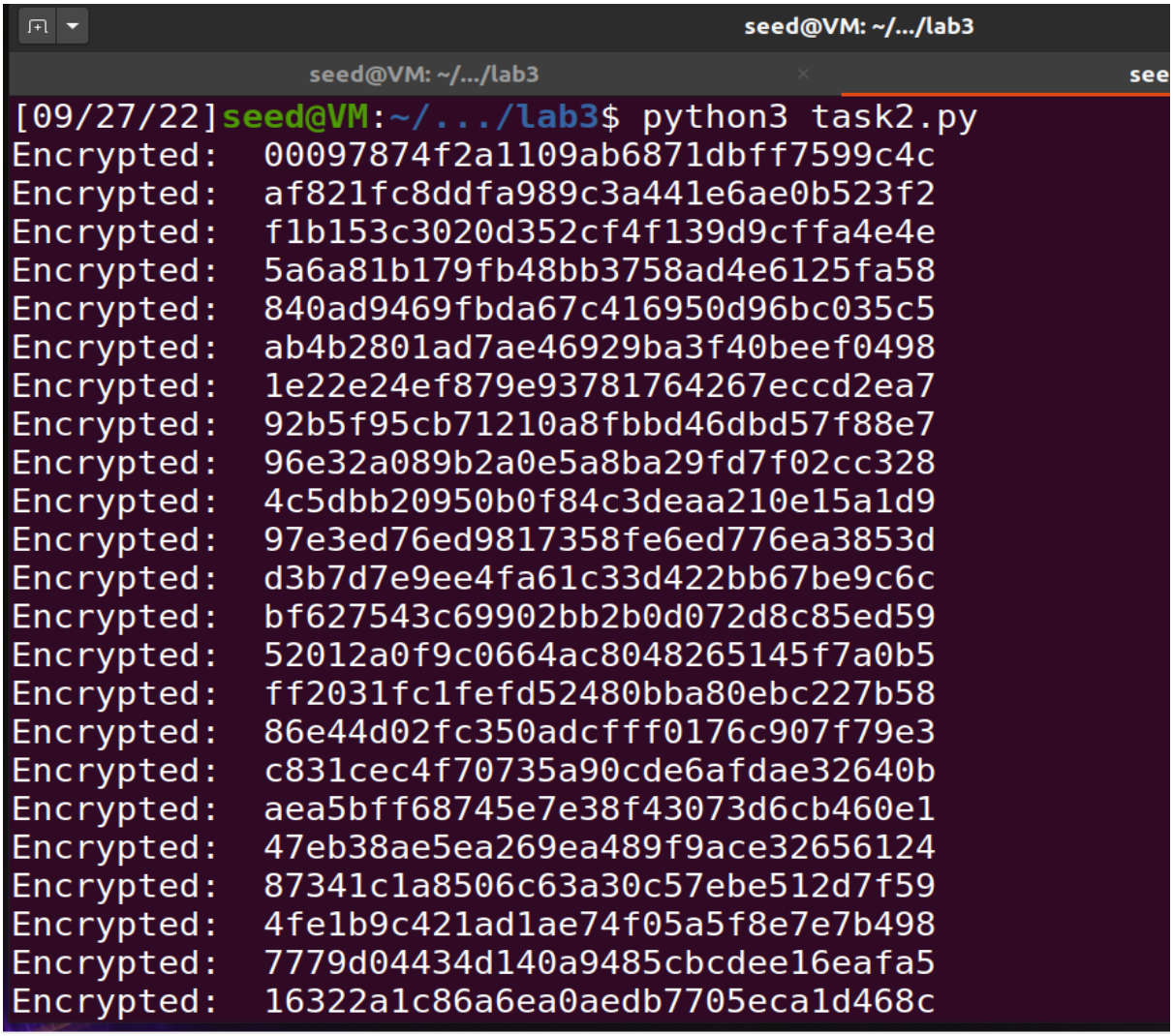
The output is all the possible keys that could have been generated between the time of value1 and value2.

Step 3:-

## Code:




## Output:




Observation:

Gives the output of encrypted plaint text over all the stored keys in keys.txt file.

Code:

```
seed@VM: ~/.../lab3    seed@VM: ~/.../lab3    seed@VM: ~
# decrypt.py
from Crypto import Random
from Crypto.Cipher import AES
file = open("keys.txt", "r")
ciphertext = "d06bf9d0dab8e8ef880660d2af65aa82"
for i in range(0,7200):
    str = file.readline()
    key = bytes.fromhex(str[:-1])#.decode("hex")
    IV = bytes.fromhex("09080706050403020100A2B2C2D2E2F2".lower())
    plaintext1 = bytes.fromhex("255044462d312e350a25d0d4c5d80a34")
    cipher = AES.new(key, AES.MODE_CBC, IV)
    encrypted = cipher.encrypt(plaintext1)
    #print("Encrypted: ",encrypted.hex())
    if ciphertext == encrypted.hex():#.encode("hex")[0:32]:
        print("")
        print("Match found")
        print("key: "+str[:-1])
        print("Ciphertext: " + ciphertext)
        print("Encrypted: " + (encrypted).hex())
        print("")
~
~
~
"task2.py" 20L, 757C written                                13,5
```

Output:

```
[09/27/22]seed@VM:~/.../lab3$ python3 task2.py

Match found
key: 95fa2030e73ed3f8da761b4eb805dfd7
Ciphertext: d06bf9d0dab8e8ef880660d2af65aa82
Encrypted: d06bf9d0dab8e8ef880660d2af65aa82

[09/27/22]seed@VM:~/.../lab3$
```

Observation:

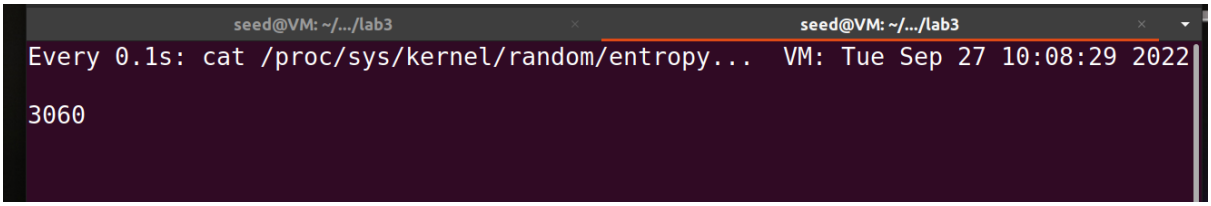
Commenting out the unwanted output we now check if the cipher text we need is present in the list of encrypted values.

## Task 3: Measure the Entropy of Kernel

### Code:

```
[09/27/22]seed@VM:~/.../lab3$  watch -n .1 cat /proc/sys/kernel/random/entropy_a
vail
[09/27/22]seed@VM:~/.../lab3$
```

### Output:




### Observation:

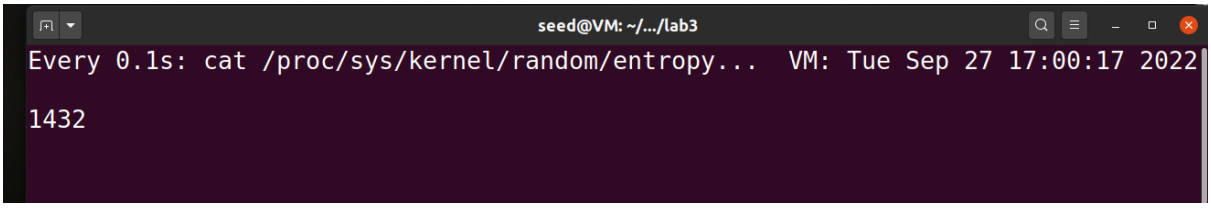
The entropy changes quickly when the mouse is moved faster and slower if it is not moved around.

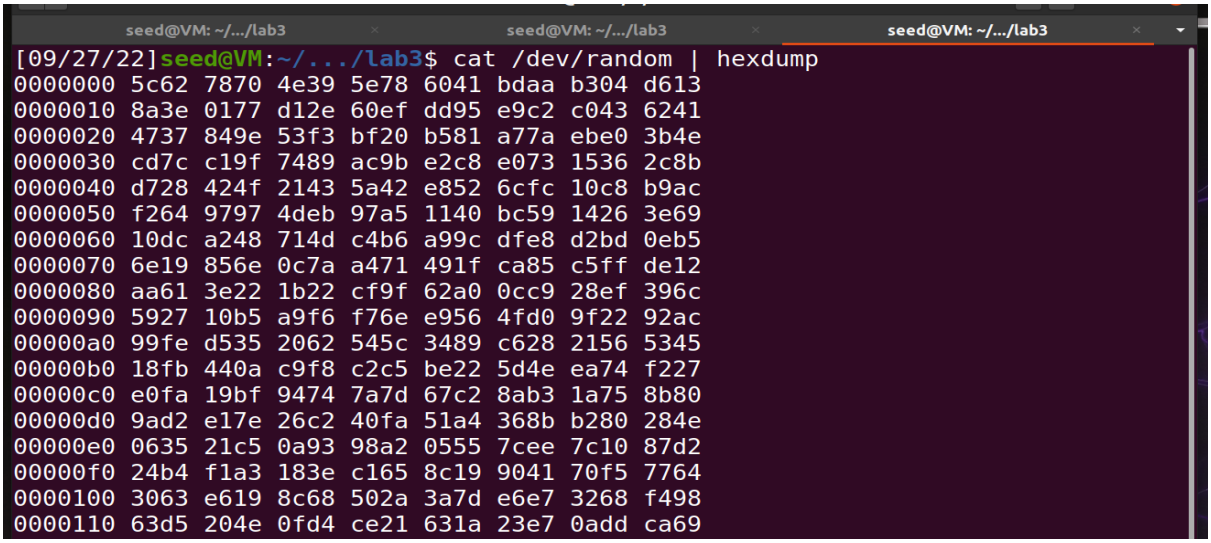
## Task 4: Get Pseudo Random Numbers from /dev/random

Step 1:-

### Output:

```
[09/27/22]seed@VM:~/.../lab3$ watch -n .1 cat /proc/sys/kernel/random/entropy_av
ail
```





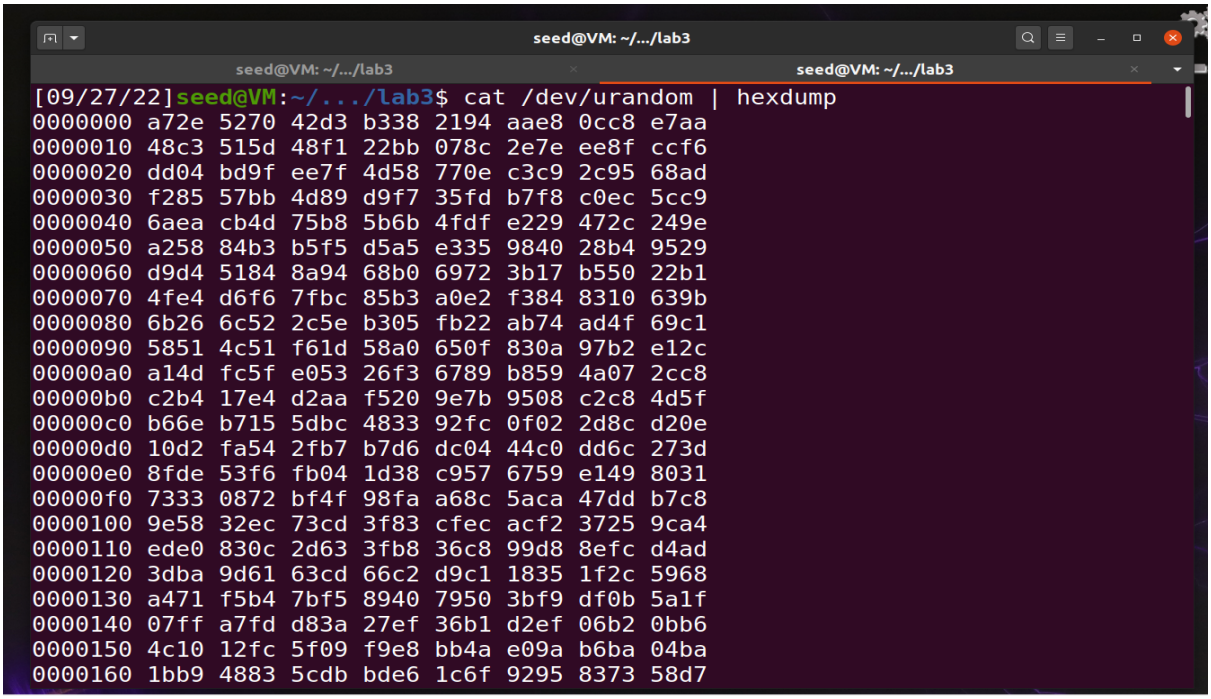

## Observation:

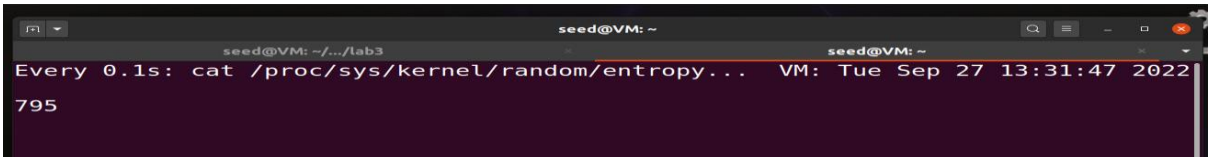
Movement of mouse and keyboard increases the entropy and low activity reduces the change rate of entropy.

## Task 5: Get Pseudo Random Numbers from /dev/urandom

Step 1:-

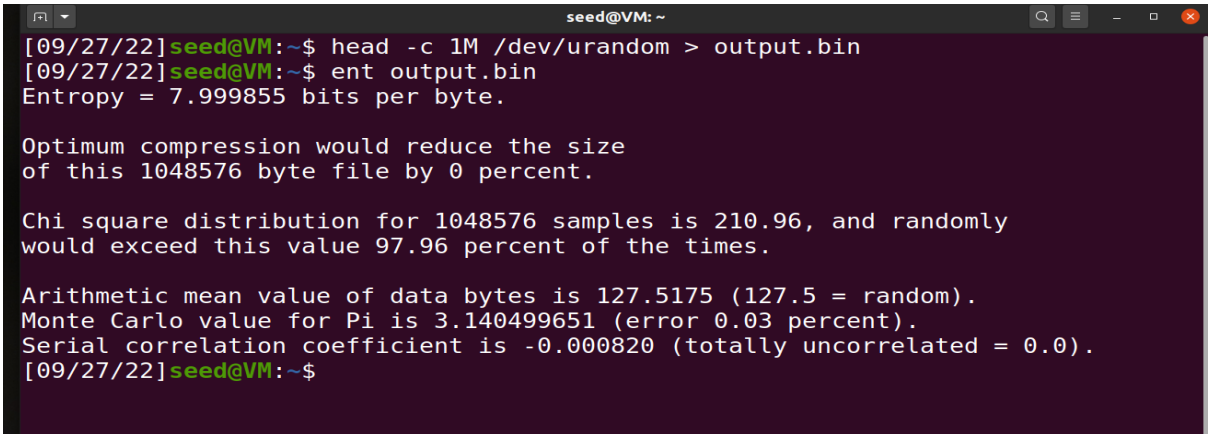
## Output:





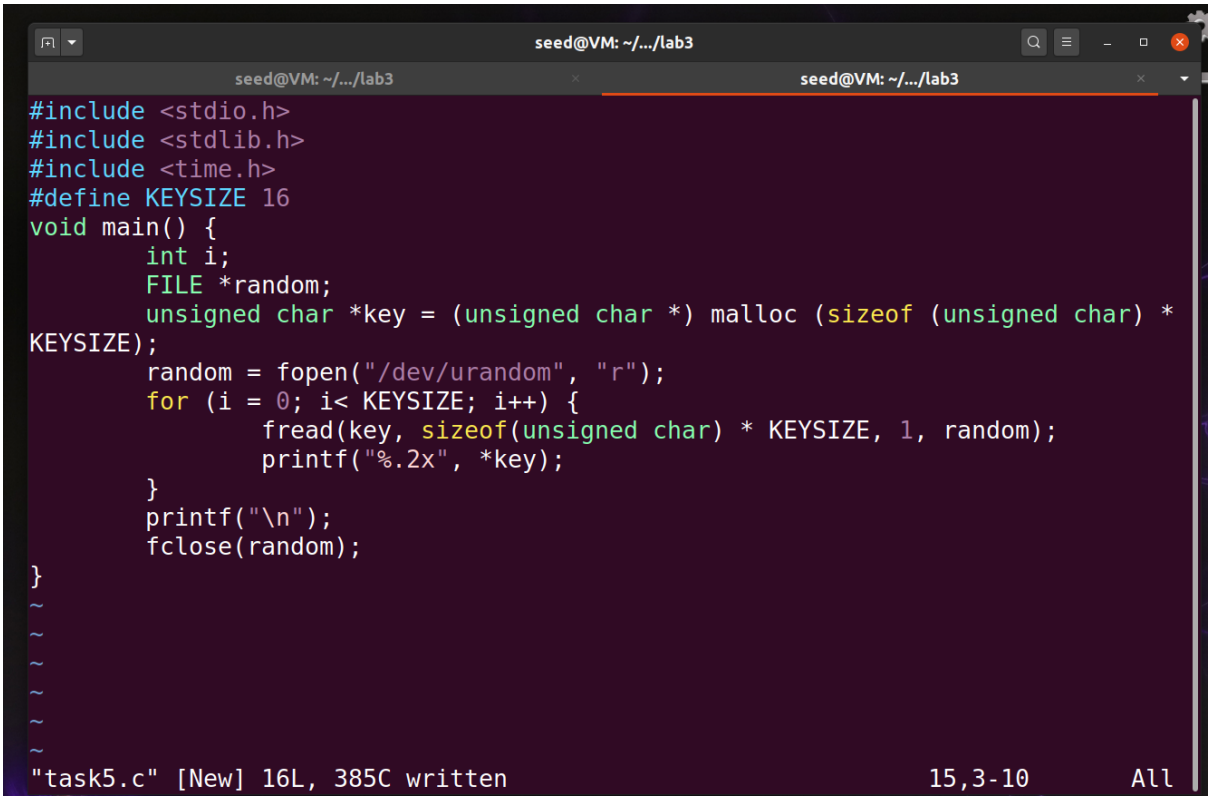

Step 2:-

Output:

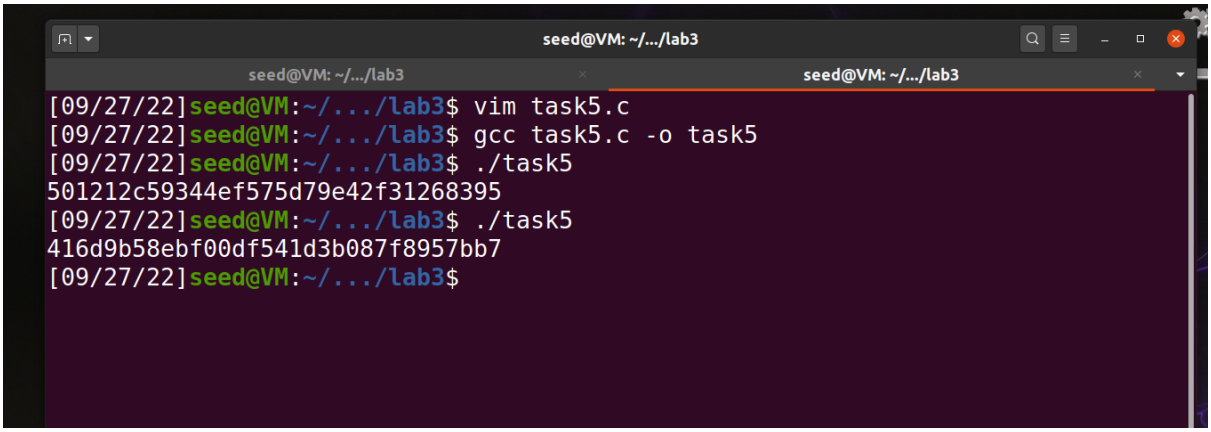



Step 3:-

Code:



## Output:




## Observation:

We use a better way of generating key using PRNG rather than relying on seed and time values.