Task1

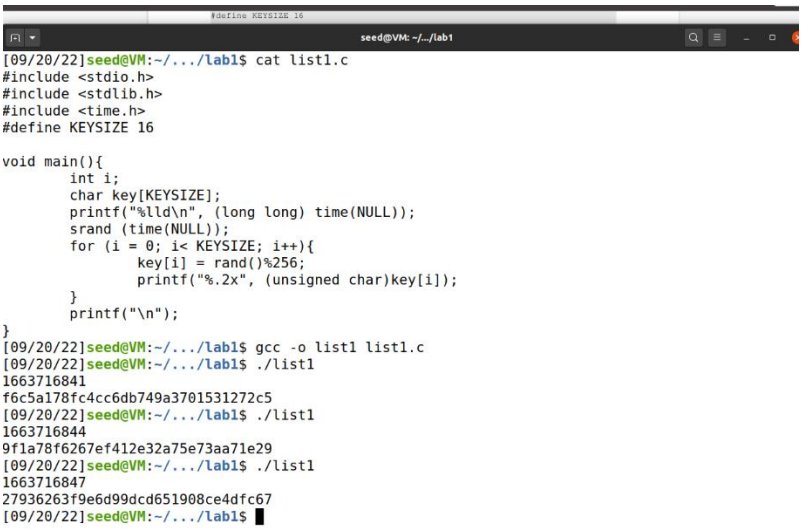


Figure 1Use time(NULL)

If we use srand(time(NULL)), we can get different result in each time.

Figure 2without time(NULL)

If we comment scrand(time(NULL)), we get the same result in each time.

The reason is time(NULL) generates the current time and scrand() use it as a seed to generate the ramdom number, which will get the different seed every time when we want to get a new key.

Task 2

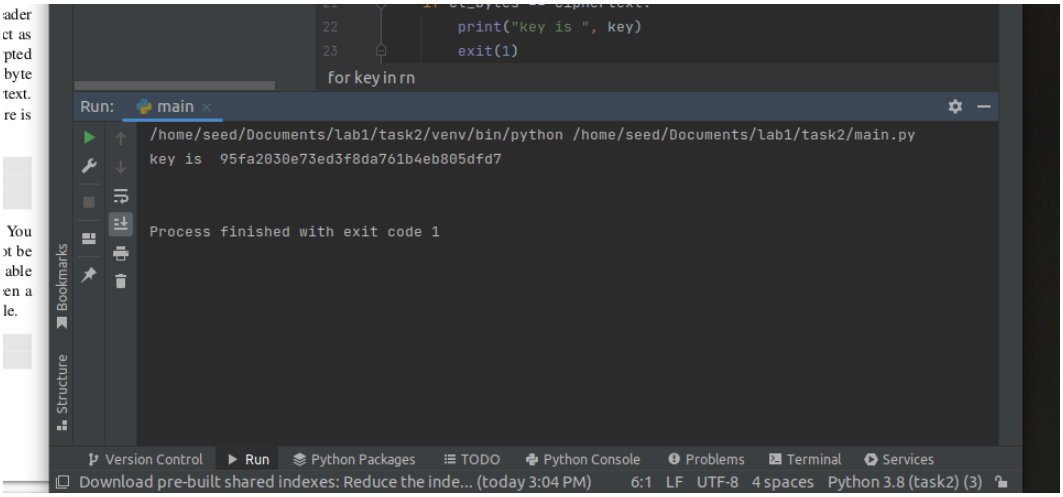


Figure 3 result

First, we use # date -d "2018-04-17 23:08:49" +%s to get the result “1524020929” which is the second time of 2018-04-17 23:08:49. Then wlet it run the random key generator as input so that we can get the all possible keys in two hours starting from this time. The outpts will be noted in a txt file that can be used in the main.py.

The next step is building the AES algorism and generating the ciphertext that keys is from txt file. We imports pycyptodome to build the AES.CBC encryption process. The calculated ciphertexts will be compared with the real ciphertext and output the key who fits the ciphertext..

Generate all possible key

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define KEYSIZE 16

void main(){

        FILE * fp;

        fp = fopen("randomnum.txt", "w");

        if(fp == NULL){

                printf("Create file fail.\n");

                exit(0);

        }
```

```
        int i;
        int d;
        unsigned long second = 1524020929;
        char key[KEYSIZE];
        for (unsigned long d=second; d > second-(2*60*60); d-- ){
                srand (d);
                for (i = 0;i<KEYSIZE;i++){
                        key[i] = rand()%256;
                        fprintf(fp,"%.2x", (unsigned char)key[i]);
                }
                fprintf(fp,"\n");
        }
        fclose(fp);
}
```

Main.py

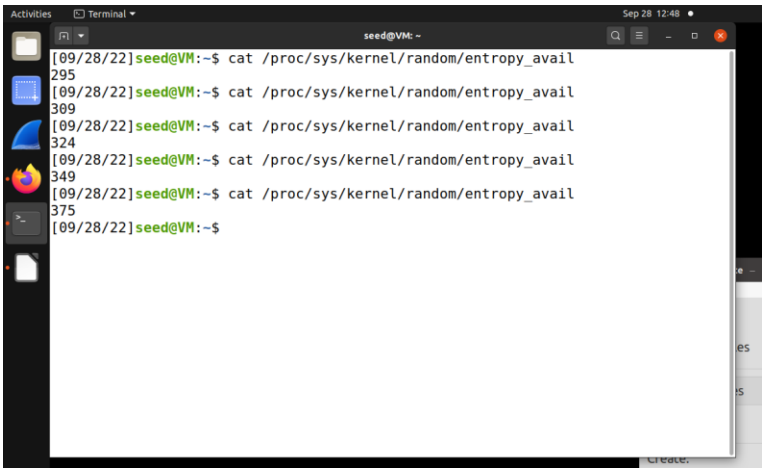
```
# Plaintext:  255044462d312e350a25d0d4c5d80a34
# Ciphertext: d06bf9d0dab8e8ef880660d2af65aa82
# IV:       09080706050403020100A2B2C2D2E2F2
# date -d "2018-04-17 23:08:49" +%s
# 1524020929

from Crypto.Cipher import AES

plaintext = bytearray.fromhex("255044462d312e350a25d0d4c5d80a34")
iv = bytearray.fromhex("09080706050403020100A2B2C2D2E2F2")
ciphertext = bytearray.fromhex("d06bf9d0dab8e8ef880660d2af65aa82")
f = open("../task1/randomnum.txt", "r")
rn = f.readlines()
```

```
for key in rn:
        key = key.strip("/n")
        keyb = bytearray.fromhex(key)
        cipher = AES.new(keyb, AES.MODE_CBC, iv)
        ct_bytes = cipher.encrypt(plaintext)
        if ct_bytes == ciphertext:
                print("key is ", key)
                exit(1)
```

Task 3

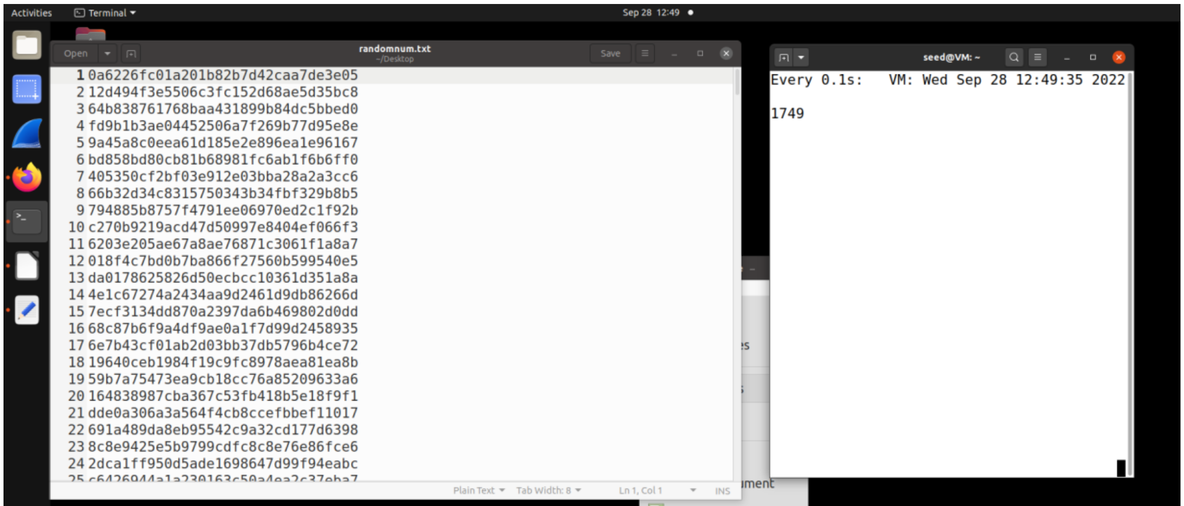
$ cat /proc/sys/kernel/random/entropy_avail

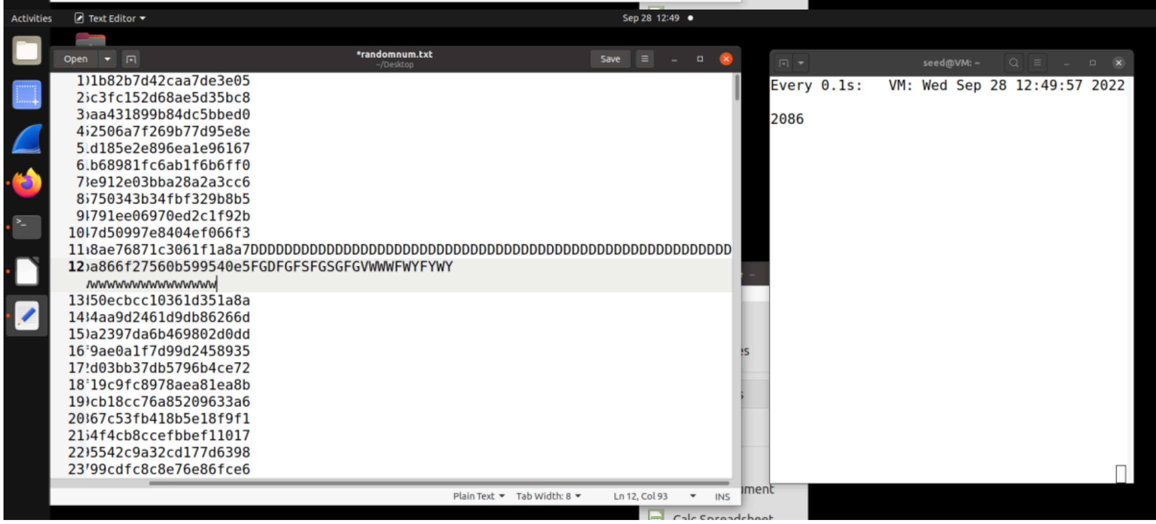
When I run this command, I can notice that the number of the outputs are increase every time,. Because every time I run this command, the keyboard, mouse are used and increase the entropy.



$ watch -n .1 cat /proc/sys/kernel/random/entropy_avail

When I run this command, I can notice that it generated the output per 0.1 second, and it will output faster if I move my mouse or type something. It will generate faster when I input faster or move mouse faster.





Task 4

$ cat /dev/random | hexdump

When I run this command, it printed out random numbers. When I use “ $ watch -n .1 cat /proc/sys/kernel/random/entropy_avail ” to monitor the entropy, I can noticed that the number of output in the monitor page will decrease suddenly, and when I move the mouse, type something ex., the number of output will increase again, then the hex number will be output, the number of the entropy will decrease again. It can proof that the entropy from user’s behaviour is used by generating the random number.

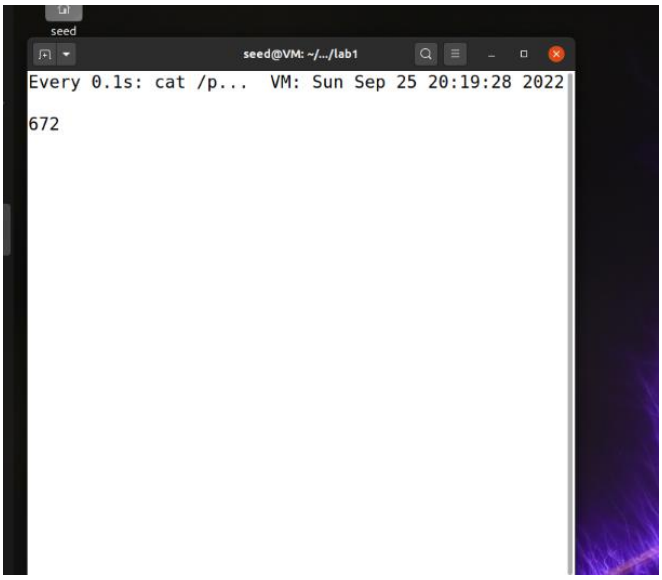


Figure 4 entropy before run hexdump

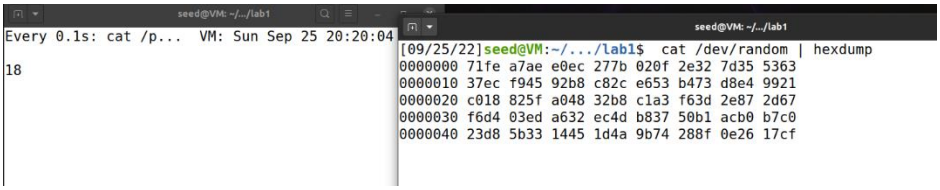


Figure 5 entropy after run hexdum

Question: We can attack the server using this way, which can consume the entropy and let the entropy of the random pool almost depleted. Therefore, it is hard to generate the random number and server is difficult to generate the random session key with the client.

Task 5

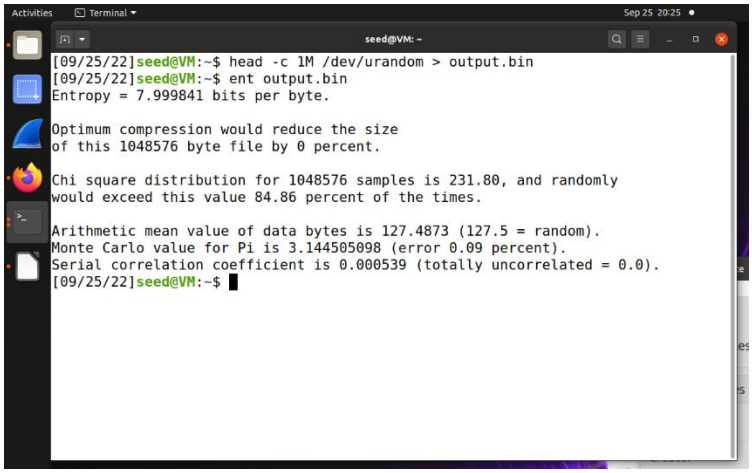


Figure 6 Analysis of random number

We generate 1M random number into the file and analysis the random number. It notes that the entropy of the random number is 9.99841 bits per bytes, and the chi square distribution is 231.80, the percentage of exceeding these value is 84.86%, which is also high. At the same time, the error of the Monte Carlo is only 0.09%, which is very small. Finally, totally uncorrelated is 0.0. In conclusion, the sequence is fairly random. The generated numbers are fairly random.

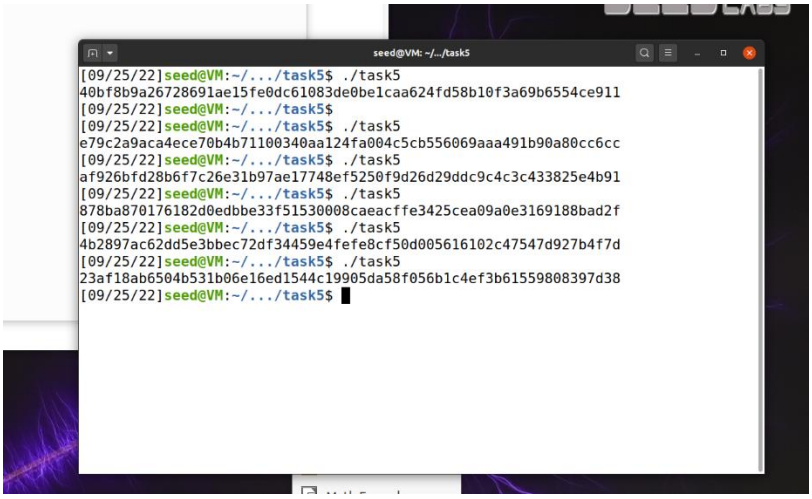


Figure 7 Output of random number generation

It outputted different random number in each time, so that the real random number can be generated.

Key generate.c

```
#include <stdio.h>

#include <stdlib.h>

#define LEN 32

int main(){

        int i;

        unsigned char *key = (unsigned char *) malloc(sizeof(unsigned char)*LEN);

        FILE* random = fopen("/dev/urandom", "r");

        fread(key, sizeof(unsigned char)*LEN, 1, random);

        fclose(random);

        for (i = 0;i<LEN;i++){

                printf("%.2x", (unsigned char)key[i]);

        }
```

```
        printf("\n");
}
```