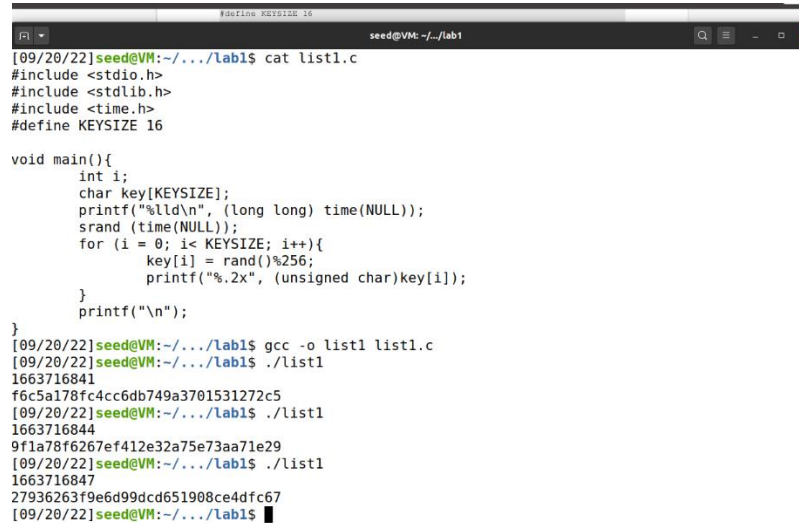


Task1

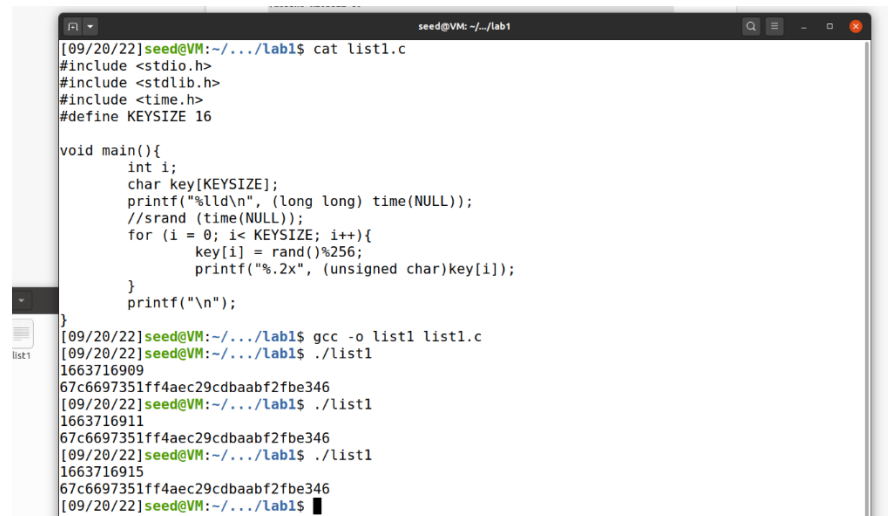


```
seed@VM: ~/.../lab1
[09/20/22]seed@VM:~/.../lab1$ cat list1.c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define KEYSIZE 16

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()%256;
        printf("%.2x", (unsigned char)key[i]);
    }
    printf("\n");
}
[09/20/22]seed@VM:~/.../lab1$ gcc -o list1 list1.c
[09/20/22]seed@VM:~/.../lab1$ ./list1
1663716841
f6c5a178fc4cc6db749a3701531272c5
[09/20/22]seed@VM:~/.../lab1$ ./list1
1663716844
9f1a78f6267ef412e32a75e73aa71e29
[09/20/22]seed@VM:~/.../lab1$ ./list1
1663716847
27936263f9e6d99dc651908ce4dfc67
[09/20/22]seed@VM:~/.../lab1$
```

Figure 1 Use time(NULL)

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



```
seed@VM: ~/.../lab1
[09/20/22]seed@VM:~/.../lab1$ cat list1.c
#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()%256;
        printf("%.2x", (unsigned char)key[i]);
    }
    printf("\n");
}
[09/20/22]seed@VM:~/.../lab1$ gcc -o list1 list1.c
[09/20/22]seed@VM:~/.../lab1$ ./list1
1663716909
67c6697351ff4aec29cdbaabf2fbe346
[09/20/22]seed@VM:~/.../lab1$ ./list1
1663716911
67c6697351ff4aec29cdbaabf2fbe346
[09/20/22]seed@VM:~/.../lab1$ ./list1
1663716915
67c6697351ff4aec29cdbaabf2fbe346
[09/20/22]seed@VM:~/.../lab1$
```

Figure 2 without time(NULL)

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

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

Task 2

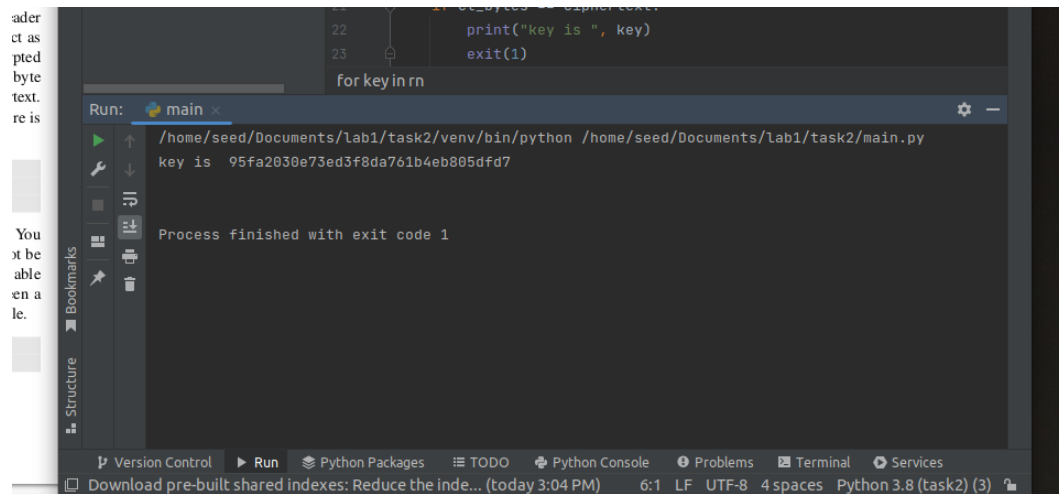


Figure 3 result

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 we let it run the random key generator as input so that we can get all possible keys in two hours starting from this time. The outputs will be noted in a txt file that can be used in the main.py.

The next step is building the AES algorithm and generating the ciphertext that keys is from txt file. We import pycryptodome 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,"%0.2x", (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()

```

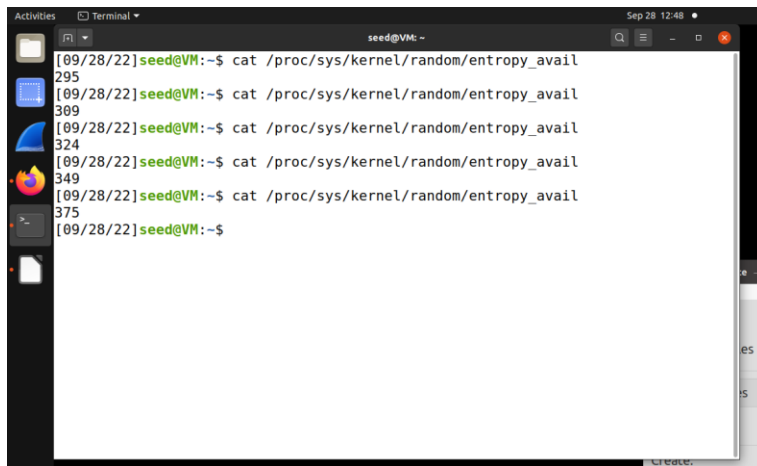
for key in rn:

```
    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
```

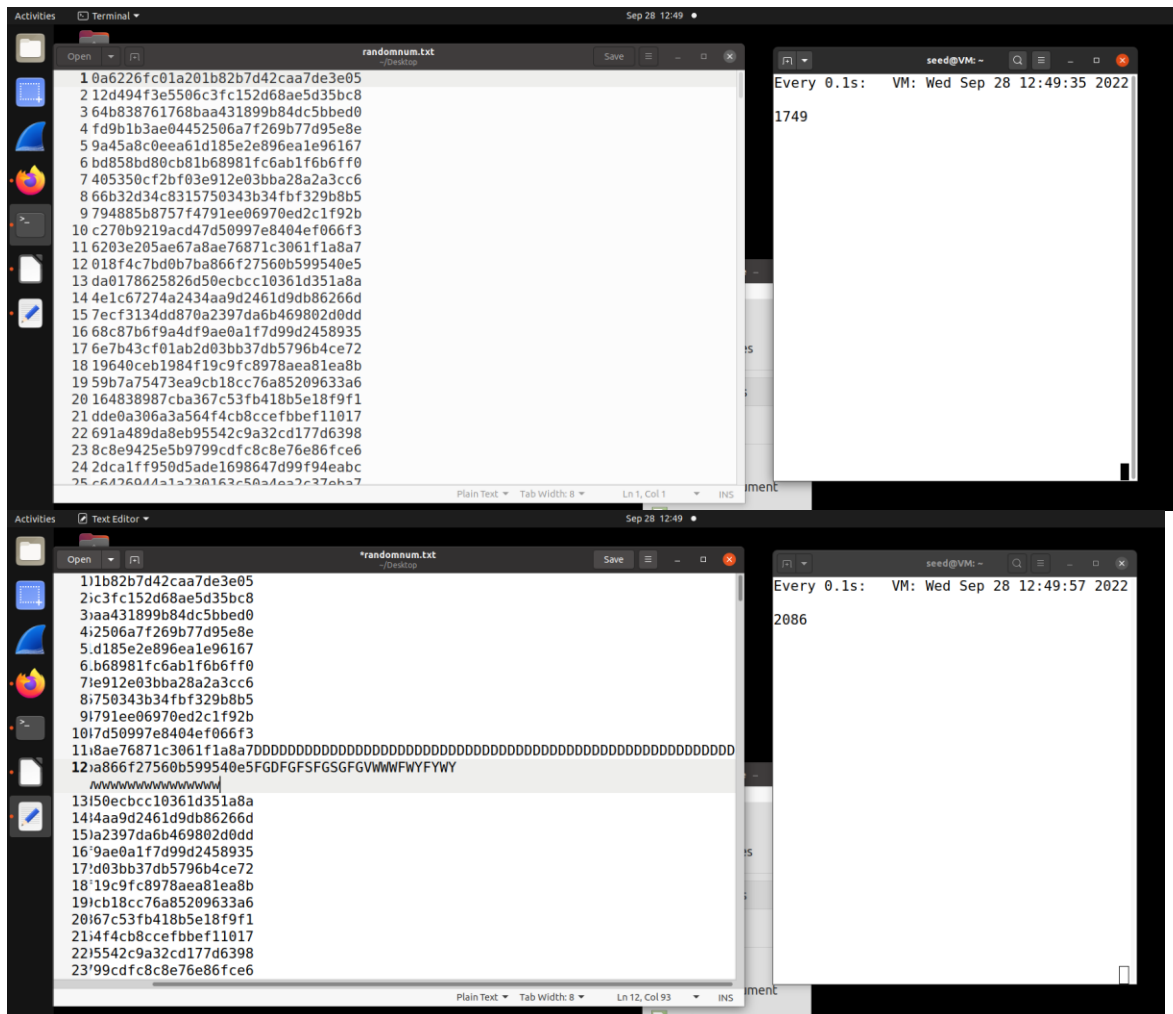
When I run this command, I can notice that the number of the outputs are increase every time,. Because every time I run this command, the keyboard, mouse are used and increase the entropy.



```
seed@VM: ~  
[09/28/22] seed@VM:~$ cat /proc/sys/kernel/random/entropy_avail  
295  
[09/28/22] seed@VM:~$ cat /proc/sys/kernel/random/entropy_avail  
309  
[09/28/22] seed@VM:~$ cat /proc/sys/kernel/random/entropy_avail  
324  
[09/28/22] seed@VM:~$ cat /proc/sys/kernel/random/entropy_avail  
349  
[09/28/22] seed@VM:~$ cat /proc/sys/kernel/random/entropy_avail  
375  
[09/28/22] seed@VM:~$
```

```
$ 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.

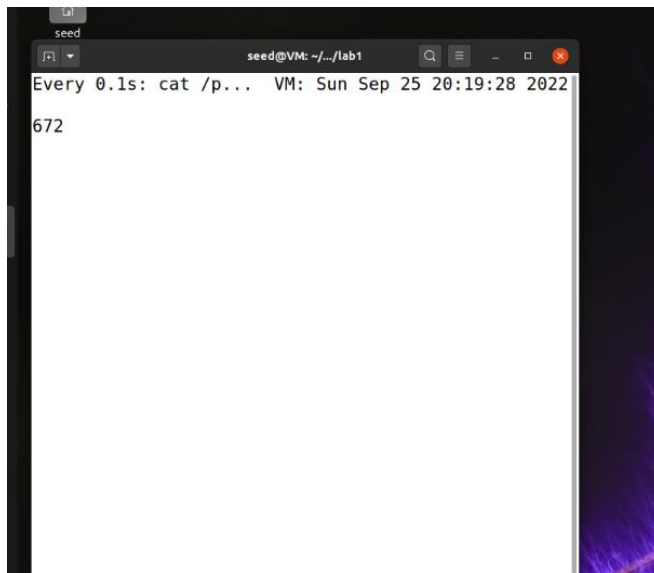


Figure 4 entropy before run hexdump

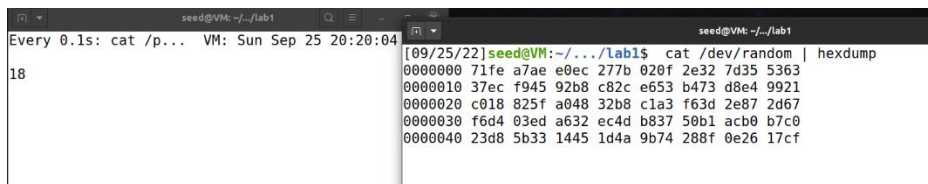


Figure 5 entropy after run hexdum

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

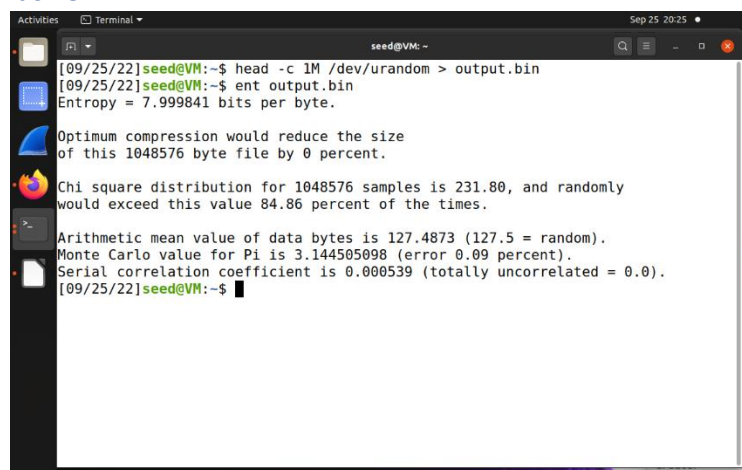
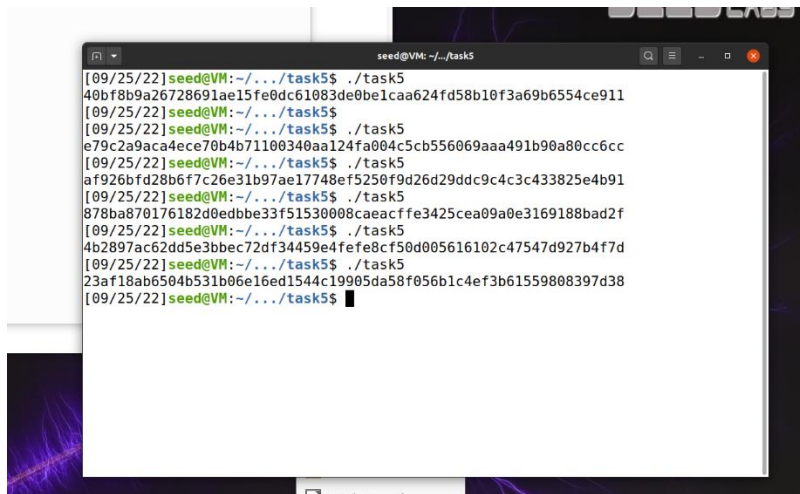


Figure 6 Analysis of random number

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.



```
seed@VM:~/.../task5$ ./task5
[09/25/22]seed@VM:~/.../task5$ ./task5
49bf8b9a26728691ae15fe0dc61083de0be1caa624fd58b10f3a69b6554ce911
[09/25/22]seed@VM:~/.../task5$ ./task5
[09/25/22]seed@VM:~/.../task5$ ./task5
e79c2a9aca4e4ce70b4b71100340aa124fa004c5cb556069aaa491b90a80cc6cc
[09/25/22]seed@VM:~/.../task5$ ./task5
af926bfd28b6f7c26e31b97ae17748ef5250f9d26d29ddc9c4c3c433825e4b91
[09/25/22]seed@VM:~/.../task5$ ./task5
878ba870176182d0edbbe33f51530008caeacffe3425cea09a0e3169188bad2f
[09/25/22]seed@VM:~/.../task5$ ./task5
4b2897ac62dd5e3bbec72df34459e4fefe8cf50d005616102c47547d927b4f7d
[09/25/22]seed@VM:~/.../task5$ ./task5
23af18ab6504b531b06e16ed1544c19905da58f056b1c4ef3b61559808397d38
[09/25/22]seed@VM:~/.../task5$
```

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");  
}
```