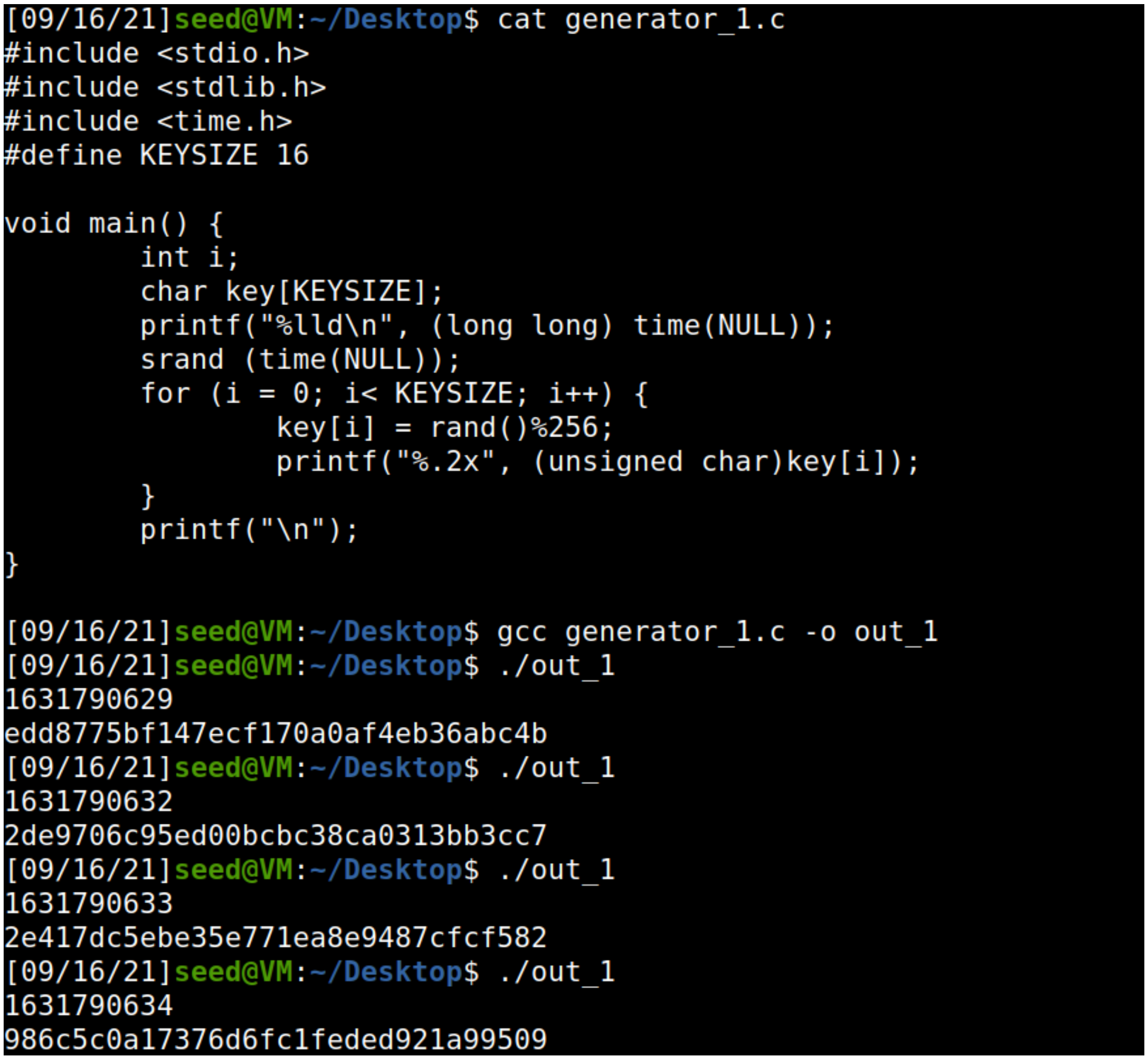
Leonardo Blas

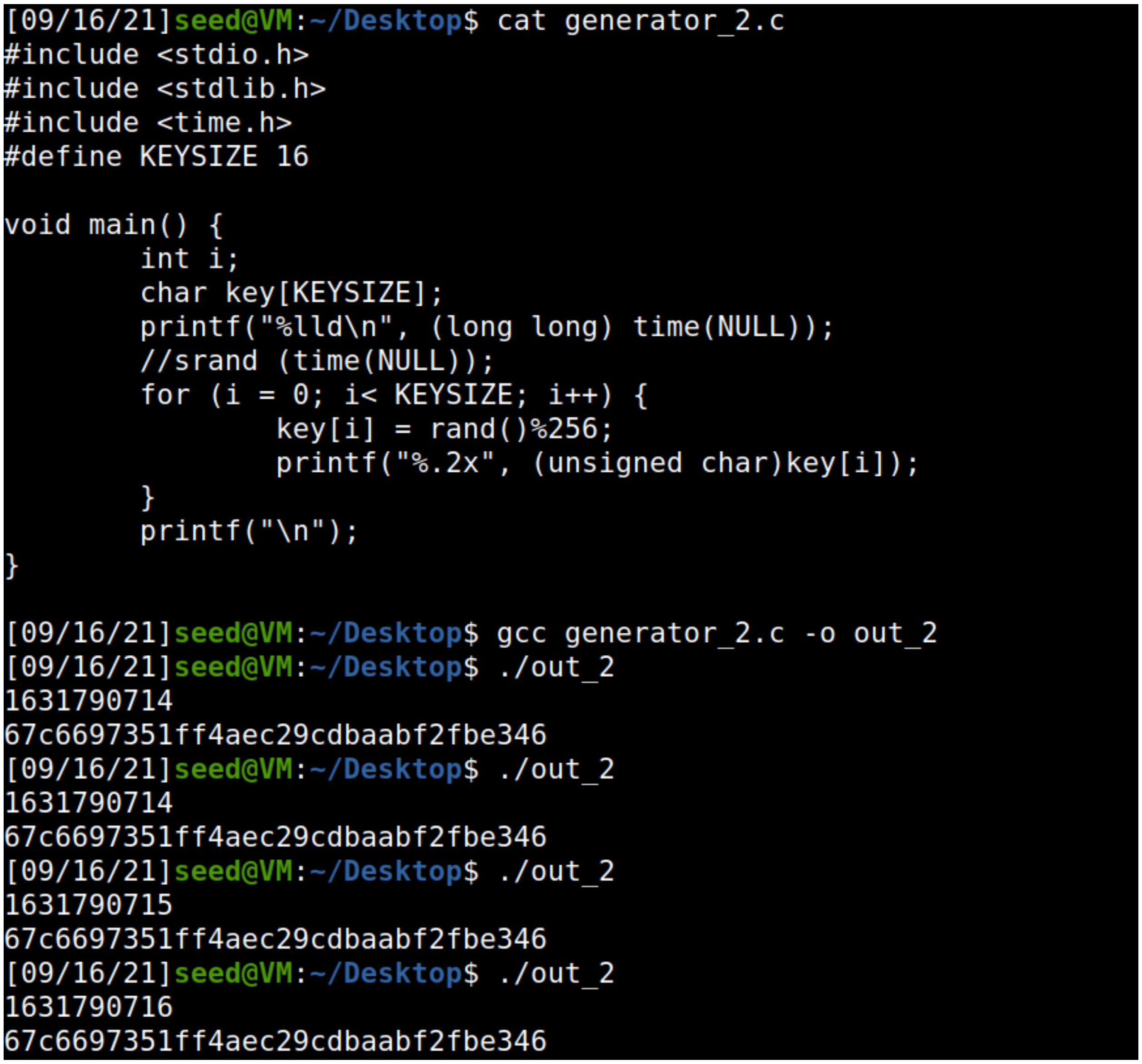
Pseudorandom Number Generation Lab

Task 1



```
[09/16/21]seed@VM:~/Desktop$ cat generator_1.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");
}

[09/16/21]seed@VM:~/Desktop$ gcc generator_1.c -o out_1
[09/16/21]seed@VM:~/Desktop$ ./out_1
1631790629
edd8775bf147ecf170a0af4eb36abc4b
[09/16/21]seed@VM:~/Desktop$ ./out_1
1631790632
2de9706c95ed00bcbc38ca0313bb3cc7
[09/16/21]seed@VM:~/Desktop$ ./out_1
1631790633
2e417dc5ebe35e771ea8e9487cfcf582
[09/16/21]seed@VM:~/Desktop$ ./out_1
1631790634
986c5c0a17376d6fc1feded921a99509
```

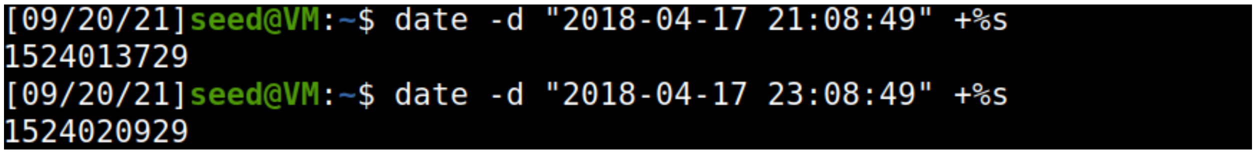
Generating a key with a time-based seed. Note the key changes with every function call.



```
[09/16/21]seed@VM:~/Desktop$ cat generator_2.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");
}

[09/16/21]seed@VM:~/Desktop$ gcc generator_2.c -o out_2
[09/16/21]seed@VM:~/Desktop$ ./out_2
1631790714
67c6697351ff4aec29cdbaabf2fbe346
[09/16/21]seed@VM:~/Desktop$ ./out_2
1631790714
67c6697351ff4aec29cdbaabf2fbe346
[09/16/21]seed@VM:~/Desktop$ ./out_2
1631790715
67c6697351ff4aec29cdbaabf2fbe346
[09/16/21]seed@VM:~/Desktop$ ./out_2
1631790716
67c6697351ff4aec29cdbaabf2fbe346
```

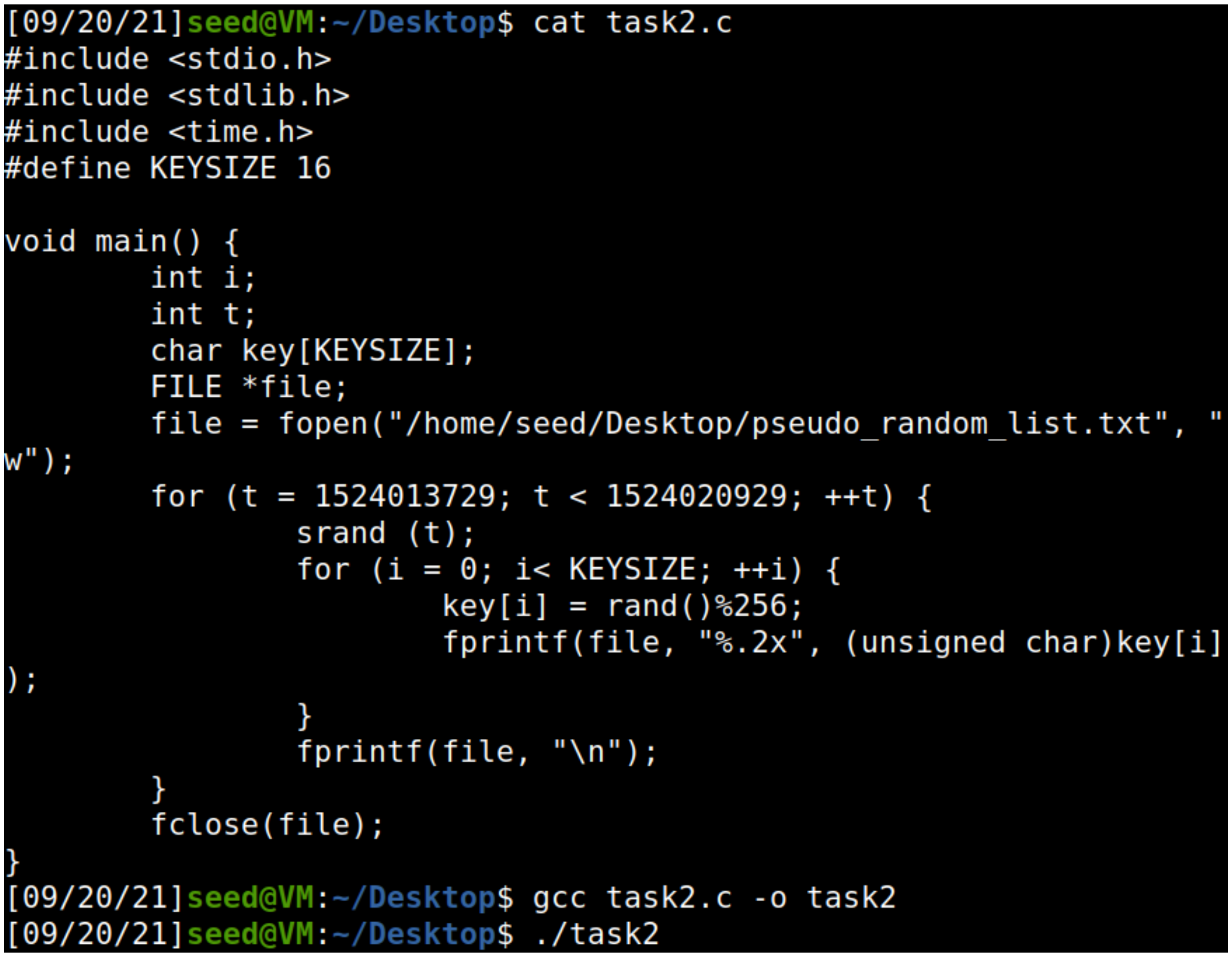
Generating a key without a time-based seed. Note the key does not change with every function call.

Task 2



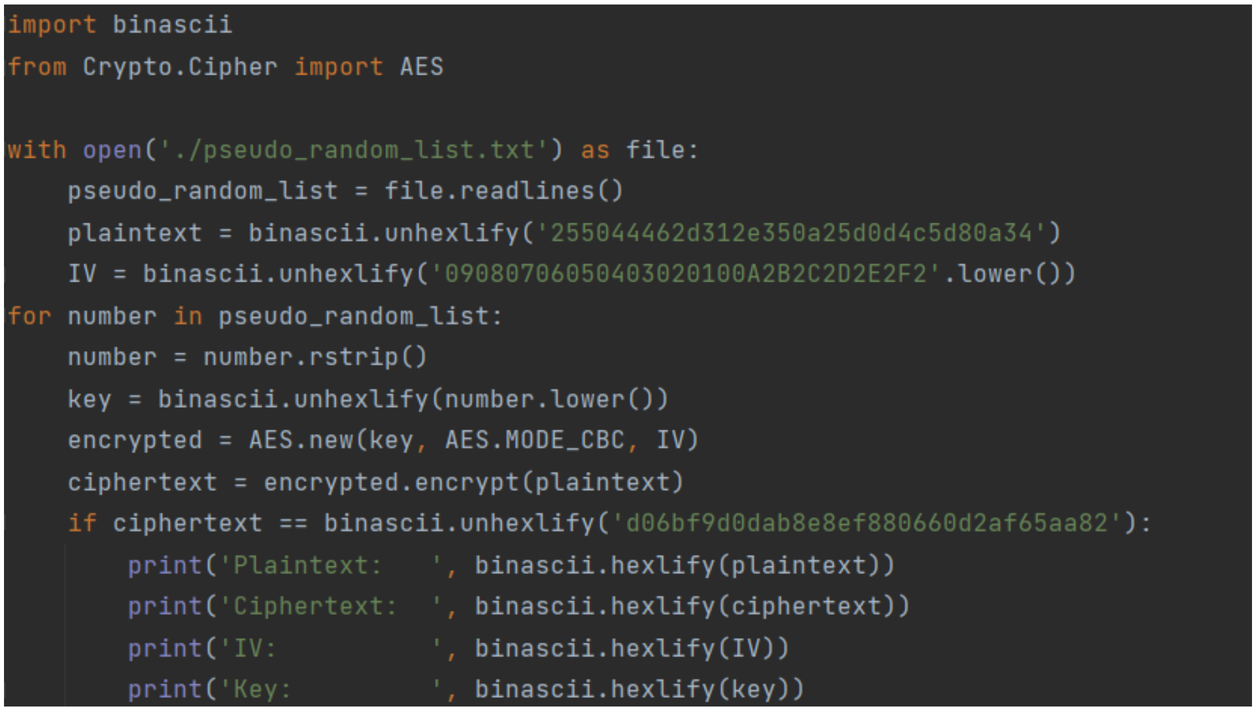
```
[09/20/21]seed@VM:~$ date -d "2018-04-17 21:08:49" +%s
1524013729
[09/20/21]seed@VM:~$ date -d "2018-04-17 23:08:49" +%s
1524020929
```

In this snippet, we compute the times between the Epoch times for the file's timestamp and 2 hours before that.



```
[09/20/21]seed@VM:~/Desktop$ cat task2.c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define KEYSIZE 16

void main() {
        int i;
        int t;
        char key[KEYSIZE];
        FILE *file;
        file = fopen("/home/seed/Desktop/pseudo_random_list.txt", "
w");
        for (t = 1524013729; t < 1524020929; ++t) {
                srand (t);
                for (i = 0; i< KEYSIZE; ++i) {
                        key[i] = rand()%256;
                        fprintf(file, "%.2x", (unsigned char)key[i]
);
                }
                fprintf(file, "\n");
        }
        fclose(file);
}
[09/20/21]seed@VM:~/Desktop$ gcc task2.c -o task2
[09/20/21]seed@VM:~/Desktop$ ./task2
```

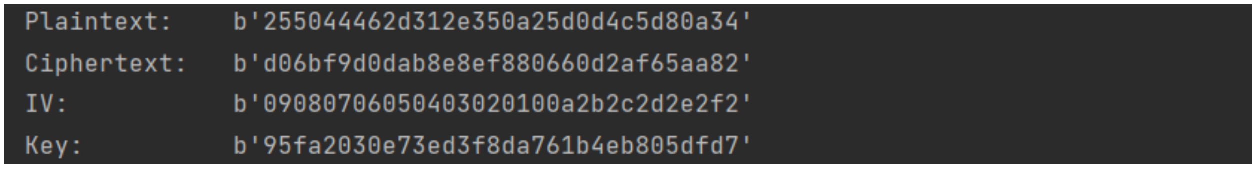
In this snippet, we create random numbers using the Epoch time interval we previously defined and store them in pseudo_random_list.txt.



```
import binascii
from Crypto.Cipher import AES

with open('./pseudo_random_list.txt') as file:
    pseudo_random_list = file.readlines()
    plaintext = binascii.unhexlify('255044462d312e350a25d0d4c5d80a34')
    IV = binascii.unhexlify('09080706050403020100A2B2C2D2E2F2'.lower())
for number in pseudo_random_list:
    number = number.rstrip()
    key = binascii.unhexlify(number.lower())
    encrypted = AES.new(key, AES.MODE_CBC, IV)
    ciphertext = encrypted.encrypt(plaintext)
    if ciphertext == binascii.unhexlify('d06bf9d0dab8e8ef880660d2af65aa82'):
        print('Plaintext:   ', binascii.hexlify(plaintext))
        print('Ciphertext:  ', binascii.hexlify(ciphertext))
        print('IV:          ', binascii.hexlify(IV))
        print('Key:         ', binascii.hexlify(key))
```

In this snippet, we use brute force to find Alice's key. We do so by using the previously generated pseudo-random numbers, the plaintext, the ciphertext, the IV, and aes-128-cbc decryption.

Finally, we compute:



```
Plaintext:    b'255044462d312e350a25d0d4c5d80a34'
Ciphertext:   b'd06bf9d0dab8e8ef880660d2af65aa82'
IV:           b'09080706050403020100a2b2c2d2e2f2'
Key:          b'95fa2030e73ed3f8da761b4eb805dfd7'
```

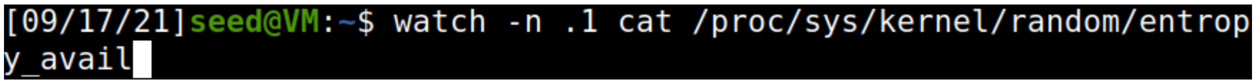
Plaintext: '255044462d312e350a25d0d4c5d80a34'

Ciphertext: 'd06bf9d0dab8e8ef880660d2af65aa82'

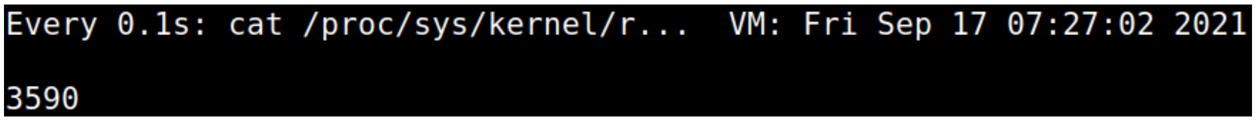
IV: '09080706050403020100a2b2c2d2e2f2'

Key: '95fa2030e73ed3f8da761b4eb805dfd7'

Task 3



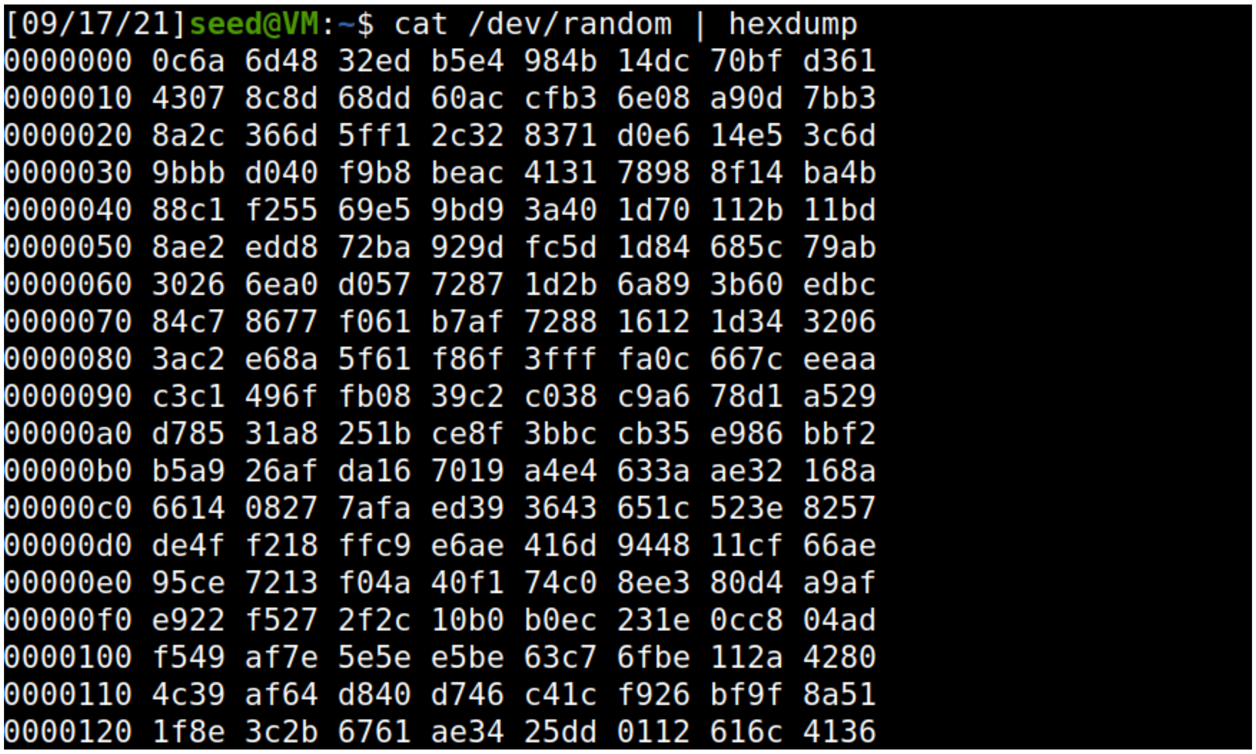
```
[09/17/21]seed@VM:~$ watch -n .1 cat /proc/sys/kernel/random/entrop
y_avail
```



```
Every 0.1s: cat /proc/sys/kernel/r...  VM: Fri Sep 17 07:27:02 2021

3590
```

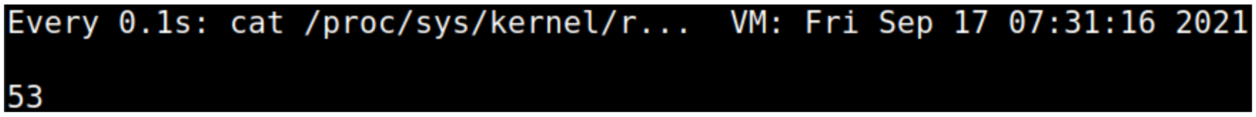
The activities that quickly increased the entropy include actions such as moving the cursor, using the mouse's scrolling wheel, clicking, and using the keyboard. Furthermore, the entropy increased even without performing any actions, at seemingly random intervals.

Task 4



```
[09/17/21]seed@VM:~$ cat /dev/random | hexdump
0000000 0c6a 6d48 32ed b5e4 984b 14dc 70bf d361
0000010 4307 8c8d 68dd 60ac cfb3 6e08 a90d 7bb3
0000020 8a2c 366d 5ff1 2c32 8371 d0e6 14e5 3c6d
0000030 9bbb d040 f9b8 beac 4131 7898 8f14 ba4b
0000040 88c1 f255 69e5 9bd9 3a40 1d70 112b 11bd
0000050 8ae2 edd8 72ba 929d fc5d 1d84 685c 79ab
0000060 3026 6ea0 d057 7287 1d2b 6a89 3b60 edbc
0000070 84c7 8677 f061 b7af 7288 1612 1d34 3206
0000080 3ac2 e68a 5f61 f86f 3fff fa0c 667c eeaa
0000090 c3c1 496f fb08 39c2 c038 c9a6 78d1 a529
00000a0 d785 31a8 251b ce8f 3bbc cb35 e986 bbf2
00000b0 b5a9 26af da16 7019 a4e4 633a ae32 168a
00000c0 6614 0827 7afa ed39 3643 651c 523e 8257
00000d0 de4f f218 ffc9 e6ae 416d 9448 11cf 66ae
00000e0 95ce 7213 f04a 40f1 74c0 8ee3 80d4 a9af
00000f0 e922 f527 2f2c 10b0 b0ec 231e 0cc8 04ad
0000100 f549 af7e 5e5e e5be 63c7 6fbe 112a 4280
0000110 4c39 af64 d840 d746 c41c f926 bf9f 8a51
0000120 1f8e 3c2b 6761 ae34 25dd 0112 616c 4136
```

The /dev/random device displays all numbers in the random pool and then blocks.



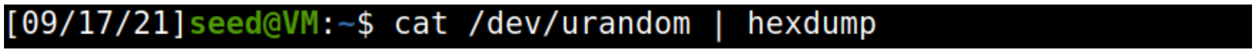
```
Every 0.1s: cat /proc/sys/kernel/r...  VM: Fri Sep 17 07:31:16 2021

53
```

The /dev/random device produces new output when the entropy reaches 64. The entropy starts at 0 and resets and inputs a number into the entropy pool once it reaches 64, non-inclusive.

Furthermore, just as previously mentioned, the entropy increases with actions such as moving the cursor, clicking, or using the keyboard. Additionally, note that the entropy will increase at random intervals even if no actions are performed.

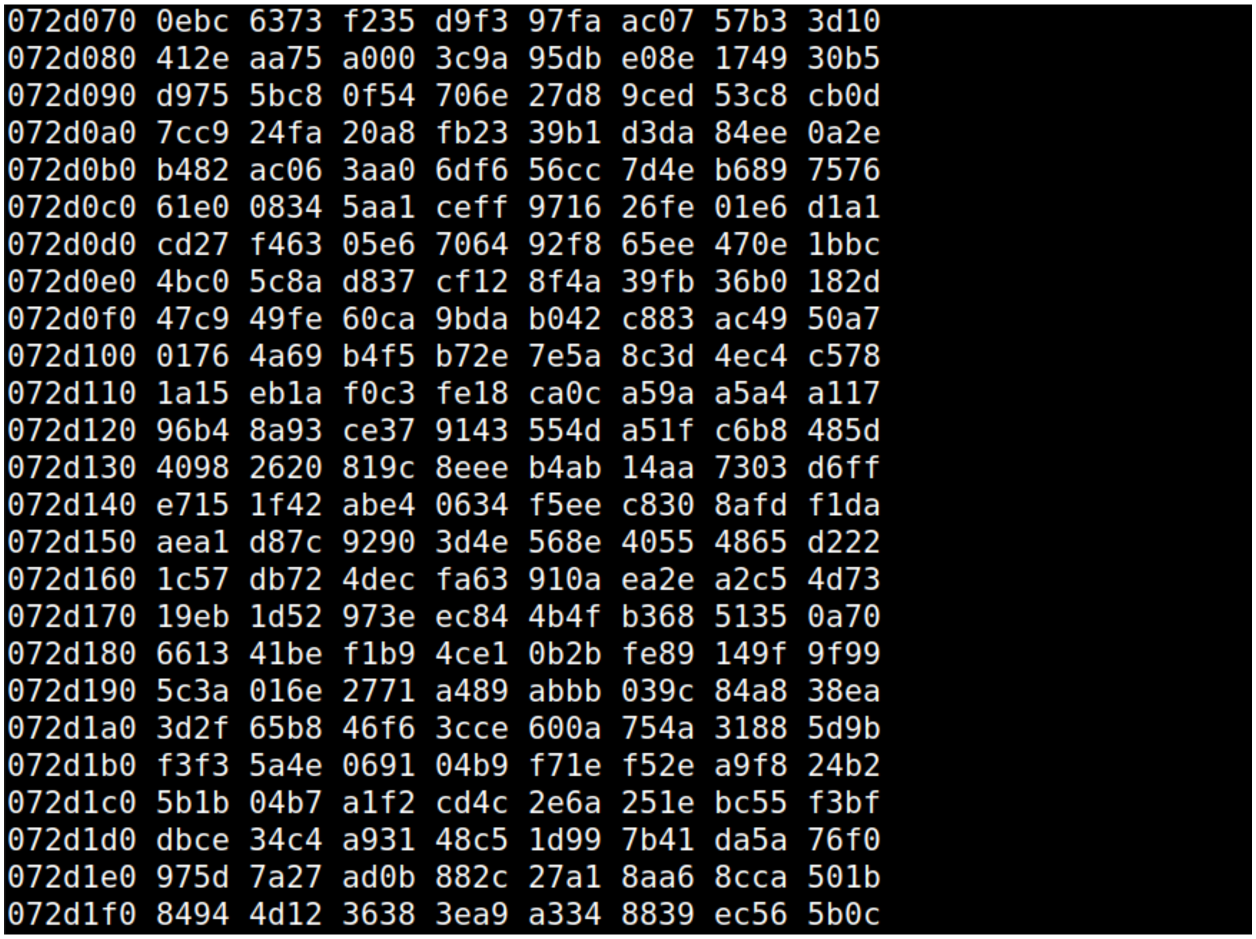
Finally, regarding how to launch a DOS attack on a server that uses /dev/random to generate random session keys with clients, we could opt for requesting many new sessions. This system's weakness is that it is entropy dependent, and it blocks when the entropy is not enough to generate a new number. If we request many sessions and deplete the entropy pool faster than it can fill, the server will deny service to users.

Task 5



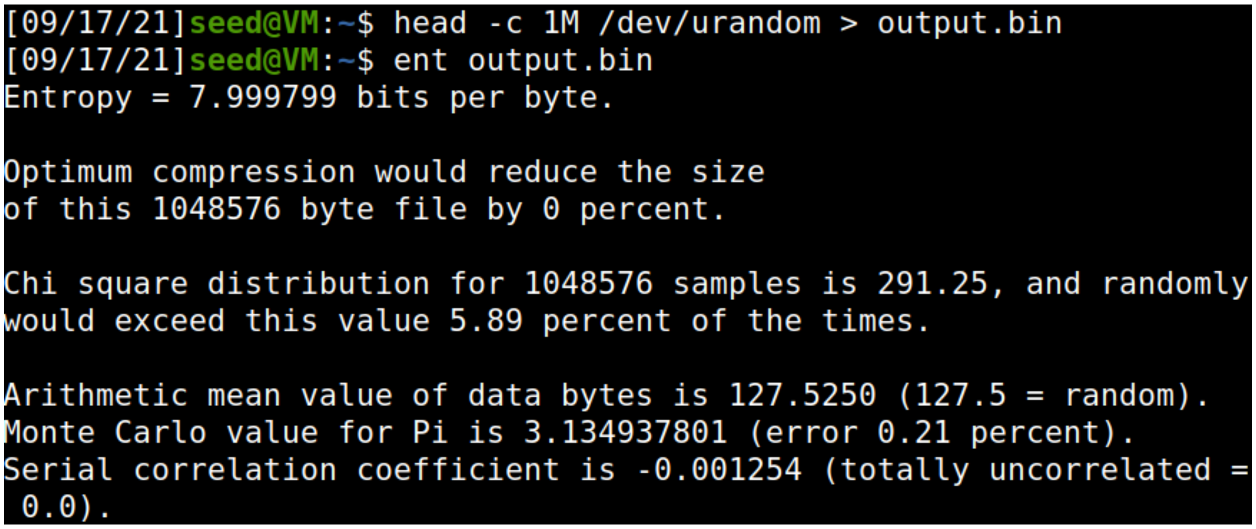
```
[09/17/21]seed@VM:~$ cat /dev/urandom | hexdump
```

Attempting to obtain pseudo-random numbers using /dev/urandom.



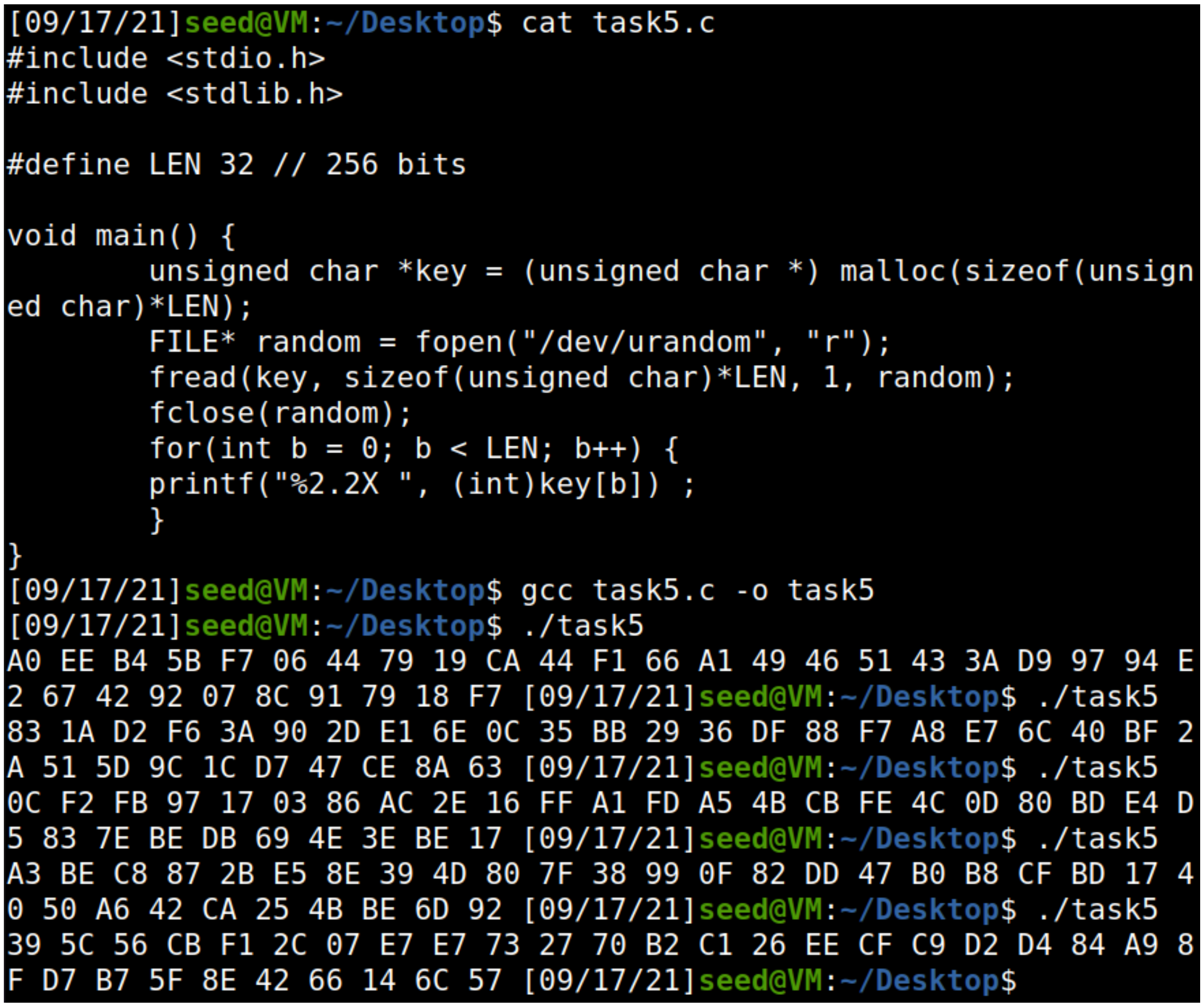
```
072d070 0ebc 6373 f235 d9f3 97fa ac07 57b3 3d10
072d080 412e aa75 a000 3c9a 95db e08e 1749 30b5
072d090 d975 5bc8 0f54 706e 27d8 9ced 53c8 cb0d
072d0a0 7cc9 24fa 20a8 fb23 39b1 d3da 84ee 0a2e
072d0b0 b482 ac06 3aa0 6df6 56cc 7d4e b689 7576
072d0c0 61e0 0834 5aa1 ceff 9716 26fe 01e6 d1a1
072d0d0 cd27 f463 05e6 7064 92f8 65ee 470e 1bbc
072d0e0 4bc0 5c8a d837 cf12 8f4a 39fb 36b0 182d
072d0f0 47c9 49fe 60ca 9bda b042 c883 ac49 50a7
072d100 0176 4a69 b4f5 b72e 7e5a 8c3d 4ec4 c578
072d110 1a15 eb1a f0c3 fe18 ca0c a59a a5a4 a117
072d120 96b4 8a93 ce37 9143 554d a51f c6b8 485d
072d130 4098 2620 819c 8eee b4ab 14aa 7303 d6ff
072d140 e715 1f42 abe4 0634 f5ee c830 8afd f1da
072d150 aea1 d87c 9290 3d4e 568e 4055 4865 d222
072d160 1c57 db72 4dec fa63 910a ea2e a2c5 4d73
072d170 19eb 1d52 973e ec84 4b4f b368 5135 0a70
072d180 6613 41be f1b9 4ce1 0b2b fe89 149f 9f99
072d190 5c3a 016e 2771 a489 abbb 039c 84a8 38ea
072d1a0 3d2f 65b8 46f6 3cce 600a 754a 3188 5d9b
072d1b0 f3f3 5a4e 0691 04b9 f71e f52e a9f8 24b2
072d1c0 5b1b 04b7 a1f2 cd4c 2e6a 251e bc55 f3bf
072d1d0 dbce 34c4 a931 48c5 1d99 7b41 da5a 76f0
072d1e0 975d 7a27 ad0b 882c 27a1 8aa6 8cca 501b
072d1f0 8494 4d12 3638 3ea9 a334 8839 ec56 5b0c
```

Pseudo-random numbers generated via /dev/urandom. Moving the mouse or using the keyboard does not appear to have any effect on this pseudo-number generation.



```
[09/17/21]seed@VM:~$ head -c 1M /dev/urandom > output.bin
[09/17/21]seed@VM:~$ ent output.bin
Entropy = 7.999799 bits per byte.

Optimum compression would reduce the size
of this 1048576 byte file by 0 percent.

Chi square distribution for 1048576 samples is 291.25, and randomly
would exceed this value 5.89 percent of the times.

Arithmetic mean value of data bytes is 127.5250 (127.5 = random).
Monte Carlo value for Pi is 3.134937801 (error 0.21 percent).
Serial correlation coefficient is -0.001254 (totally uncorrelated =
 0.0).
```

In this snippet, we use the ent tool to learn about the quality of the pseudo-random numbers generated through /dev/urandom. In this case, we learn that this pool of pseudo-random numbers is totally uncorrelated and that, by analyzing the arithmetic mean, ent considers this pool as random.



```
[09/17/21]seed@VM:~/Desktop$ cat task5.c
#include <stdio.h>
#include <stdlib.h>

#define LEN 32 // 256 bits

void main() {
        unsigned char *key = (unsigned char *) malloc(sizeof(unsign
ed char)*LEN);
        FILE* random = fopen("/dev/urandom", "r");
        fread(key, sizeof(unsigned char)*LEN, 1, random);
        fclose(random);
        for(int b = 0; b < LEN; b++) {
        printf("%2.2X ", (int)key[b]) ;
        }
}
[09/17/21]seed@VM:~/Desktop$ gcc task5.c -o task5
[09/17/21]seed@VM:~/Desktop$ ./task5
A0 EE B4 5B F7 06 44 79 19 CA 44 F1 66 A1 49 46 51 43 3A D9 97 94 E
2 67 42 92 07 8C 91 79 18 F7 [09/17/21]seed@VM:~/Desktop$ ./task5
83 1A D2 F6 3A 90 2D E1 6E 0C 35 BB 29 36 DF 88 F7 A8 E7 6C 40 BF 2
A 51 5D 9C 1C D7 47 CE 8A 63 [09/17/21]seed@VM:~/Desktop$ ./task5
0C F2 FB 97 17 03 86 AC 2E 16 FF A1 FD A5 4B CB FE 4C 0D 80 BD E4 D
5 83 7E BE DB 69 4E 3E BE 17 [09/17/21]seed@VM:~/Desktop$ ./task5
A3 BE C8 87 2B E5 8E 39 4D 80 7F 38 99 0F 82 DD 47 B0 B8 CF BD 17 4
0 50 A6 42 CA 25 4B BE 6D 92 [09/17/21]seed@VM:~/Desktop$ ./task5
39 5C 56 CB F1 2C 07 E7 E7 73 27 70 B2 C1 26 EE CF C9 D2 D4 84 A9 8
F D7 B7 5F 8E 42 66 14 6C 57 [09/17/21]seed@VM:~/Desktop$
```

Finally, in this snippet we generate 256-bit encryption keys using /dev/urandom. Note how all the keys are different.