



**Faculty of Engineering & Technology
Electrical & Computer Engineering Department**

APPLIED CRYPTOGRAPHY

ENCS4320

Pseudo Random Number Generation Lab

Prepared by:

ShahEd Jamhour 1180654

Instructor: Dr. Ahmad Alsadeh

Section:1

April 2022

Table of Contents

Abstract:	3
Task 1: Generate Encryption Key in a Wrong Way.....	4
Task 2: Guessing the Key	6
Task 3: Measure the Entropy of Kernel	8
Task 4: Get Pseudo Random Numbers from /dev/random	9
Task5: Get Random Numbers from /dev/urandom	10

Figure 1: Task 1 code -" Generating a 128-bit encryption key.....	4
Figure 2:Executing the code.....	4
Figure 3:Executing the code after commenting SRAND	5
Figure 4:get the epoch of 2018-04-15	6
Figure 5:Generate all possible keys.....	6
Figure 6:Redirect list of possible keys to txt file	6
Figure 7:Guessing Key.....	7
Figure 8:Measuring the Entropy of Kernel.....	8
Figure 9:Get Pseudo Random Numbers from /dev/random	9

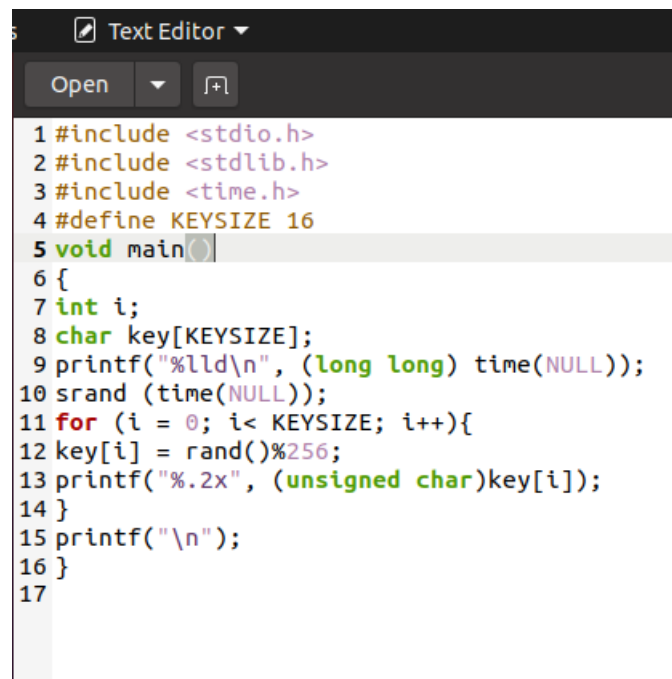
Abstract:

Our goal in this lab is to learn why random number generation is not appropriate for generating secrets such as encryption keys. In addition, the lab will provide us with a standard way for generating pseudo-random numbers good for security reasons.

Task 1: Generate Encryption Key in a Wrong Way

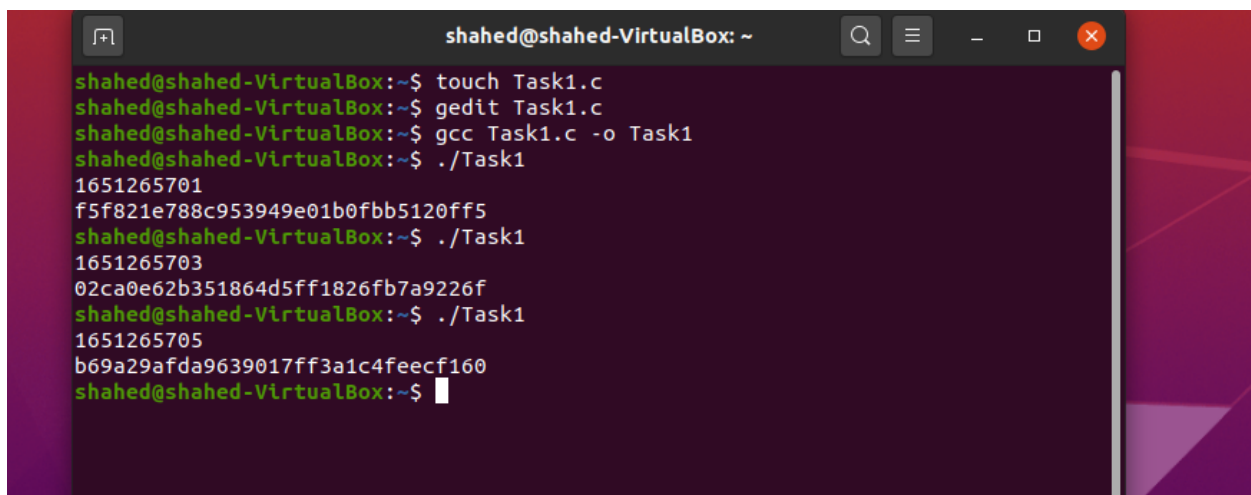
As a starting point, we must begin with something that is random; otherwise, the outcome would be quite predictable.

With the **current time** seeded into the pseudo random number generator, the following program is run.



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #define KEYSIZE 16
5 void main()
6 {
7     int i;
8     char key[KEYSIZE];
9     printf("%lld\n", (long long) time(NULL));
10    srand (time(NULL));
11    for (i = 0; i < KEYSIZE; i++){
12        key[i] = rand()%256;
13        printf("%.2x", (unsigned char)key[i]);
14    }
15    printf("\n");
16 }
17
```

Figure 1: Task 1 code -" Generating a 128-bit encryption key



```
shahed@shahed-VirtualBox: ~
shahed@shahed-VirtualBox:~$ touch Task1.c
shahed@shahed-VirtualBox:~$ gedit Task1.c
shahed@shahed-VirtualBox:~$ gcc Task1.c -o Task1
shahed@shahed-VirtualBox:~$ ./Task1
1651265701
f5f821e788c953949e01b0fbb5120ff5
shahed@shahed-VirtualBox:~$ ./Task1
1651265703
02ca0e62b351864d5ff1826fb7a9226f
shahed@shahed-VirtualBox:~$ ./Task1
1651265705
b69a29afda9639017ff3a1c4feecf160
shahed@shahed-VirtualBox:~$
```

Figure 2:Executing the code

As can be seen, executing “RandomTime” several times always gives a different result

Since the **current time** is used as a random seed, the seed is always different every time the program runs.

Now after commenting line 10, it can be noticed that the generated random number remains unchanged in every run.

A terminal window with a dark purple background and green text. The prompt is 'shahed@shahed-VirtualBox:~\$'. The user enters 'gedit Task1.c', then './Task1'. The output shows a decimal number '1651265885' and a hexadecimal string '53bccf1b4b9a89797c5af9eaefc4fda0'. This sequence is repeated three times, with the decimal number changing slightly from 1651265885 to 1651265887 in the third run, while the hexadecimal string remains identical. The prompt is '~\$'.

Figure 3:Executing the code after commenting SRAND

At first, we noticed that the random number generated and the number of seconds were different each time the program executed. This is because the function SRAND uses time (NULL) to set a different seed.

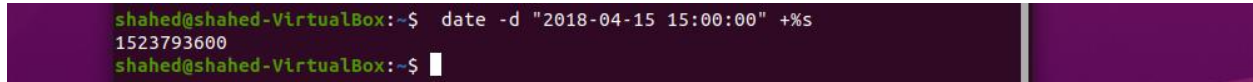
"Time (NULL)" returns the number (after conversion) of seconds since about midnight 1970-01-01, so that number changes every second.

But then, when rand (time (NULL)) is commented out, the default is a random number seed 0 because time is not seeded, so every time you run the program, the resulting random number is the same.

Task 2: Guessing the Key

First, we need to get the epoch of 2018-04-17 23:08:49 by:

```
date -d "2018-04-15 23:08:49" +%s
```



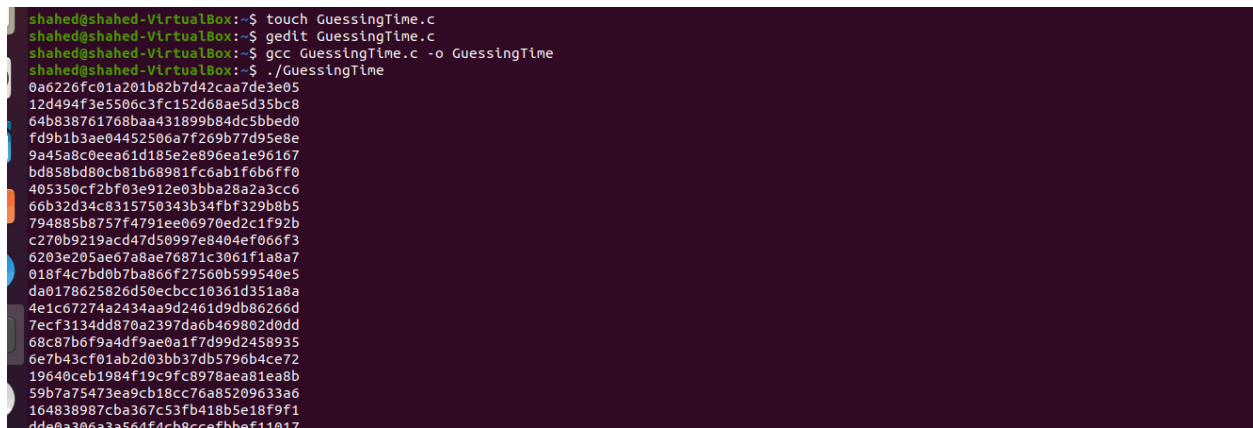
```
shahed@shahed-VirtualBox:~$ date -d "2018-04-15 15:00:00" +%s
1523793600
shahed@shahed-VirtualBox:~$
```

Figure 4: get the epoch of 2018-04-15

it returns 1523793600.

Then we list all possible random numbers generated by Task1.c within the two hours, This was don't by adding a loop before line 12 .

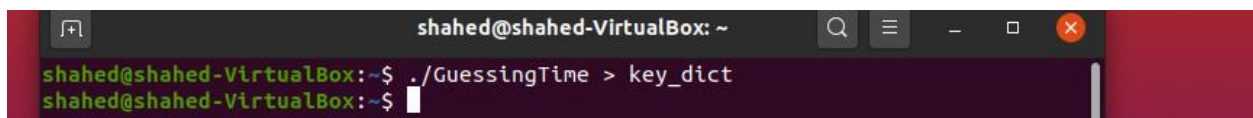
The file was named GuessingTime.c:



```
shahed@shahed-VirtualBox:~$ touch GuessingTime.c
shahed@shahed-VirtualBox:~$ gcc GuessingTime.c -o GuessingTime
shahed@shahed-VirtualBox:~$ ./GuessingTime
0a6226fc01a201b82b7d42caa7de3e05
12d494f3e5506c3fc152d68ae5d35bc8
64b838761768baa431899b84dc5bbcd0
fd9b1b3ae04452506a7f269b77d95e8e
9a45a8c0eea61d185e2e896ea1e96167
bd858bd80cb81b68981fc6ab1f6b6ff0
405350cf2bf03e912e03bba28a2a3cc6
66b32d34c8315750343b34fbf329b8b5
794885b0757f4791ee06970ed2c1f92b
c270b9219acd47d50997e8404ef066f3
6203e205ae07a8ae76871c3061f1a8a7
018f4c7bd0b7ba866f27560b599540e5
da0178625826d50ecbcc10361d351a8a
4e1c67274a2434aa9d2461d9db86266d
7ecf3134dd870a2397da6b469802d0dd
68c87b6f9a4df9ae0a1f7d99d2458935
6e7b43cf01ab2d03bb37db5796b4ce72
19640ceb1984f19c9fc8978aea81ea8b
59b7a75473ea9cb18cc76a85209633a6
164838987cba367c53fb418b5e18f9f1
dde0a386a3a564f4cb8c6fbbef11017
```

Figure 5: Generate all possible keys

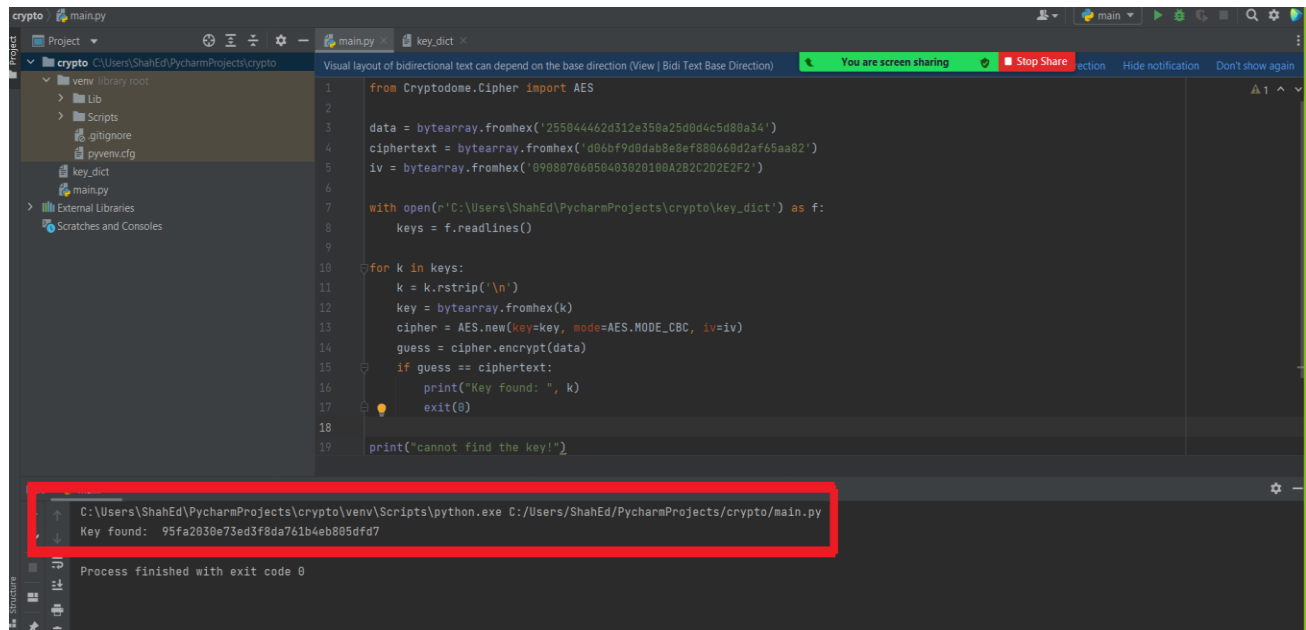
The list of keys was obtained then redirect it to a txt file:



```
shahed@shahed-VirtualBox: ~
shahed@shahed-VirtualBox:~$ ./GuessingTime > key_dict
shahed@shahed-VirtualBox:~$
```

Figure 6: Redirect list of possible keys to txt file

Then brute-force method was used to crack the key from key_dict.txt:



```
1 from Cryptodome.Cipher import AES
2
3 data = bytearray.fromhex('255044462d312e350a25d0d4c5d80a34')
4 ciphertext = bytearray.fromhex('d06bf9d0dab8e8ef880640d2af65aa82')
5 iv = bytearray.fromhex('09080706050403020100A2B2C2D2E2F2')
6
7 with open(r'C:\Users\ShahEd\PycharmProjects\crypto\key_dict.txt') as f:
8     keys = f.readlines()
9
10 for k in keys:
11     k = k.rstrip('\n')
12     key = bytearray.fromhex(k)
13     cipher = AES.new(key=key, mode=AES.MODE_CBC, iv=iv)
14     guess = cipher.decrypt(data)
15     if guess == ciphertext:
16         print("Key found: ", k)
17         exit(0)
18
19 print("cannot find the key!")
```

Key found: 95fa2030e73ed3f8da761bb4eb805dfd7

Process finished with exit code 0

Figure 7: Guessing Key

Key was found = 95fa2030e73ed3f8da761bb4eb805dfd7

- Time as a seed value is not a true random number, and it is not advisable to generate random numbers with time.

Task 3: Measure the Entropy of Kernel

An entropy measure is used to determine randomness. It indicates how many random bits the system currently possesses. The following command tells us how much entropy the kernel currently possesses:

```
cat /proc/sys/kernel/random/entropy_avail
```

A terminal window with a dark purple background and green text. It shows a series of commands and their outputs. The commands are: 'cat /proc/sys/kernel/random/entropy_avail' (repeated four times), 'touch hello', 'cat /proc/sys/kernel/random/entropy_avail' (repeated two times), and 'watch -n .1 cat /proc/sys/kernel/random/entropy_avail'. The outputs are: 759, 791, 820, 1040, 1118, and the start of a watch command output.

```
shahed@shahed-VirtualBox:~$ cat /proc/sys/kernel/random/entropy_avail
759
shahed@shahed-VirtualBox:~$ cat /proc/sys/kernel/random/entropy_avail
791
shahed@shahed-VirtualBox:~$ cat /proc/sys/kernel/random/entropy_avail
820
shahed@shahed-VirtualBox:~$ touch hello
shahed@shahed-VirtualBox:~$ cat /proc/sys/kernel/random/entropy_avail
1040
shahed@shahed-VirtualBox:~$ cat /proc/sys/kernel/random/entropy_avail
1118
shahed@shahed-VirtualBox:~$ watch -n .1 cat /proc/sys/kernel/random/entropy_avail
```

Figure 8: Measuring the Entropy of Kernel

A terminal window showing the output of the 'watch' command. The text 'Every 0.1s: cat /proc/sys/kernel/random/entropy_avail' is followed by the value '1468'. The top right corner of the terminal shows the date and time: 'shahed-VirtualBox: Sat Apr 30 17:35:29 2022'.

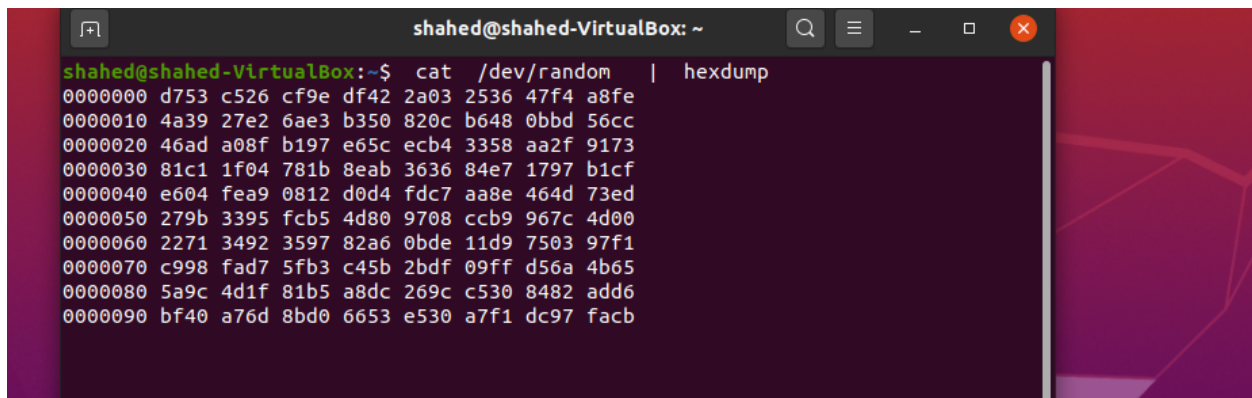
```
Every 0.1s: cat /proc/sys/kernel/random/entropy_avail
1468
shahed-VirtualBox: Sat Apr 30 17:35:29 2022
```

It was found that every time you move the mouse, tap the keyboard, etc., it will cause a change in entropy.

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

Random data collected from the physical resources are stored in Linux's random pool and turned into pseudo-random numbers by using two devices. These two devices are /dev/random and /dev/urandom.

The main difference between /dev/random, /dev/urandom is that /dev/random blocks if the entropy is not indicating sufficient randomness, /dev/urandom does not block ever, even when the pseudo-random number generator is not fully seeded.

A terminal window titled 'shahed@shahed-VirtualBox: ~' with a search icon, menu icon, and window controls. The command 'cat /dev/random | hexdump' is entered. The output shows a series of 16 lines of hexadecimal data, each starting with an offset from 00000000 to 0000000f. The data consists of 16 groups of 8 hexadecimal bytes each, representing random data from /dev/random.

```
shahed@shahed-VirtualBox:~$ cat /dev/random | hexdump
00000000 d753 c526 cf9e df42 2a03 2536 47f4 a8fe
00000010 4a39 27e2 6ae3 b350 820c b648 0bbd 56cc
00000020 46ad a08f b197 e65c ecb4 3358 aa2f 9173
00000030 81c1 1f04 781b 8eab 3636 84e7 1797 b1cf
00000040 e604 fea9 0812 d0d4 fdc7 aa8e 464d 73ed
00000050 279b 3395 fcb5 4d80 9708 ccb9 967c 4d00
00000060 2271 3492 3597 82a6 0bde 11d9 7503 97f1
00000070 c998 fad7 5fb3 c45b 2bdf 09ff d56a 4b65
00000080 5a9c 4d1f 81b5 a8dc 269c c530 8482 add6
00000090 bf40 a76d 8bd0 6653 e530 a7f1 dc97 facb
```

Figure 9: Get Pseudo Random Numbers from /dev/random

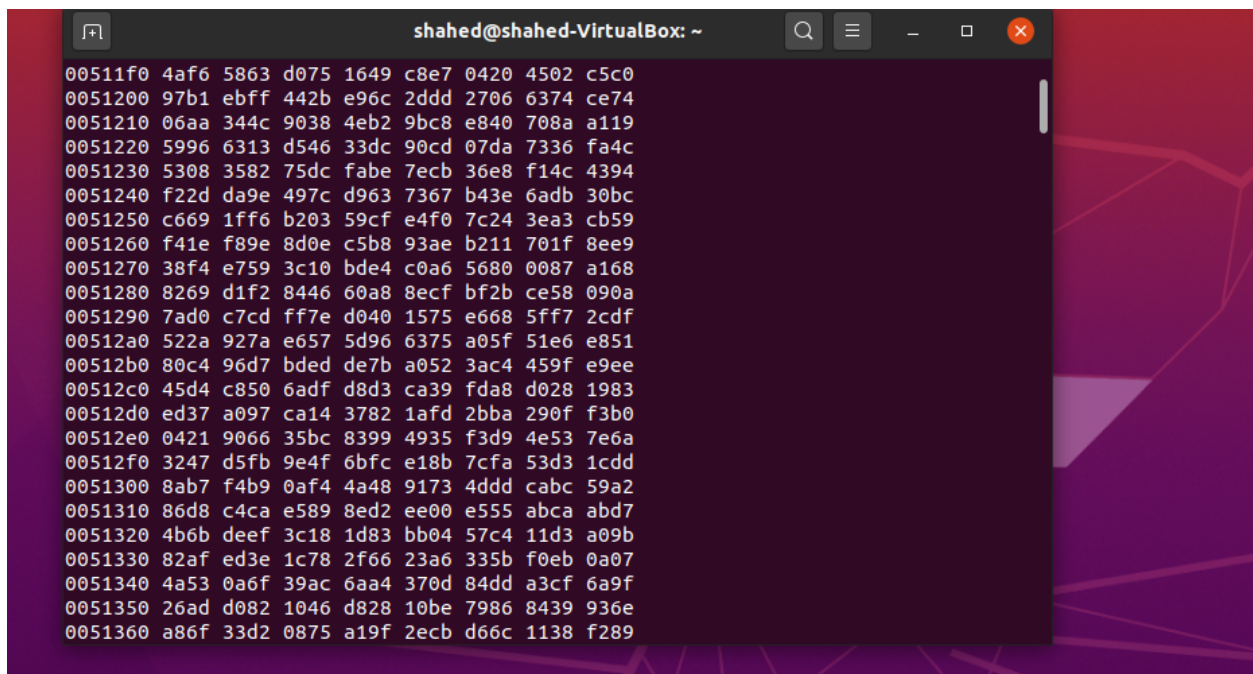
- Question: If a server uses /dev/random to generate the random session key with a client. Please describe how you can launch a Denial-Of-Service (DOS) attack on such a server?

An attacker keeps requesting connection establishments, making /dev/random run out of entropy. At that point, random number generation stops working.

Task5: Get Random Numbers from /dev/urandom

urandom is a PRNG that's periodically re-seeded from the system's entropy pools when they contain enough estimated entropy.

Let's look at the behavior of /dev/urandom. again, we use cat to get pseudo-random numbers from this device.

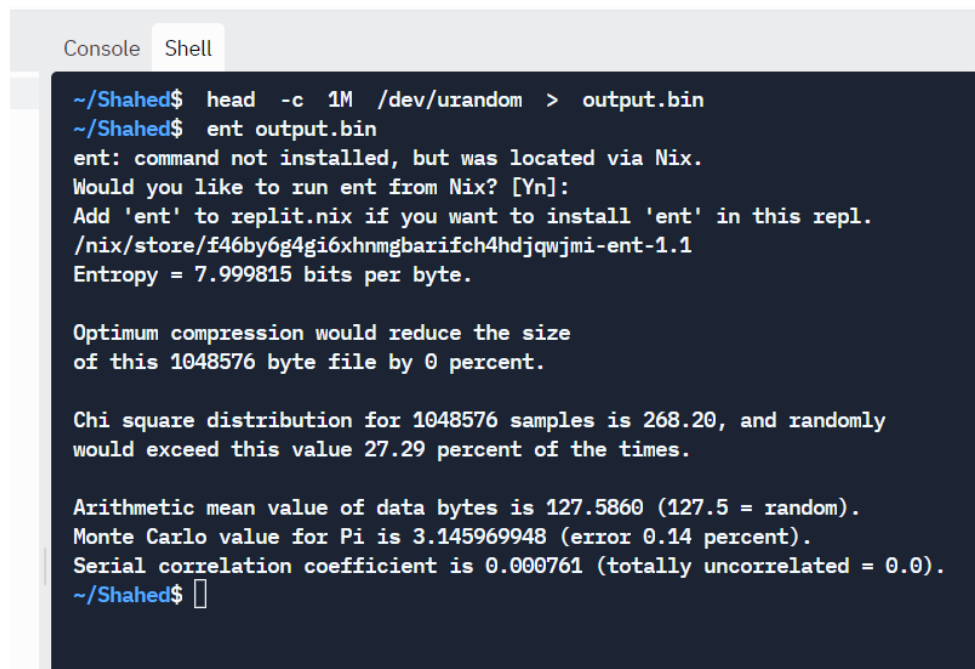
A terminal window titled 'shahed@shahed-VirtualBox: ~' displays the output of the command 'cat /dev/urandom'. The output consists of 32 lines of hexadecimal strings, each 32 characters long, representing random data. The strings are displayed in a monospaced font on a dark background. The window has standard Linux terminal window controls (minimize, maximize, close) and a search icon in the title bar.

```
shahed@shahed-VirtualBox: ~  
00511f0 4af6 5863 d075 1649 c8e7 0420 4502 c5c0  
0051200 97b1 ebff 442b e96c 2ddd 2706 6374 ce74  
0051210 06aa 344c 9038 4eb2 9bc8 e840 708a a119  
0051220 5996 6313 d546 33dc 90cd 07da 7336 fa4c  
0051230 5308 3582 75dc fabe 7ecb 36e8 f14c 4394  
0051240 f22d da9e 497c d963 7367 b43e 6adb 30bc  
0051250 c669 1ff6 b203 59cf e4f0 7c24 3ea3 cb59  
0051260 f41e f89e 8d0e c5b8 93ae b211 701f 8ee9  
0051270 38f4 e759 3c10 bde4 c0a6 5680 0087 a168  
0051280 8269 d1f2 8446 60a8 8ecf bf2b ce58 090a  
0051290 7ad0 c7cd ff7e d040 1575 e668 5ff7 2cdf  
00512a0 522a 927a e657 5d96 6375 a05f 51e6 e851  
00512b0 80c4 96d7 bded de7b a052 3ac4 459f e9ee  
00512c0 45d4 c850 6adf d8d3 ca39 fda8 d028 1983  
00512d0 ed37 a097 ca14 3782 1afd 2bba 290f f3b0  
00512e0 0421 9066 35bc 8399 4935 f3d9 4e53 7e6a  
00512f0 3247 d5fb 9e4f 6bfc e18b 7cfa 53d3 1cdd  
0051300 8ab7 f4b9 0af4 4a48 9173 4ddd cabc 59a2  
0051310 86d8 c4ca e589 8ed2 ee00 e555 abca abd7  
0051320 4b6b deef 3c18 1d83 bb04 57c4 11d3 a09b  
0051330 82af ed3e 1c78 2f66 23a6 335b f0eb 0a07  
0051340 4a53 0a6f 39ac 6aa4 370d 84dd a3cf 6a9f  
0051350 26ad d082 1046 d828 10be 7986 8439 936e  
0051360 a86f 33d2 0875 a19f 2ecb d66c 1138 f289
```

The console will frantically print data., so we truncate the first 1 MB outputs into a file named output.bin.

Then we used **ent** to evaluate its information density:

➤ ent output.bin

A screenshot of a terminal window with a dark background and light-colored text. The window has two tabs at the top: 'Console' and 'Shell'. The terminal shows the following commands and output:

```
~/Shahed$ head -c 1M /dev/urandom > output.bin
~/Shahed$ ent output.bin
ent: command not installed, but was located via Nix.
Would you like to run ent from Nix? [Yn]:
Add 'ent' to repl.it.nix if you want to install 'ent' in this repl.
/nix/store/f46by6g4gi6xhnmgbarifch4hdjqwjmi-ent-1.1
Entropy = 7.999815 bits per byte.

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

Chi square distribution for 1048576 samples is 268.20, and randomly
would exceed this value 27.29 percent of the times.

Arithmetic mean value of data bytes is 127.5860 (127.5 = random).
Monte Carlo value for Pi is 3.145969948 (error 0.14 percent).
Serial correlation coefficient is 0.000761 (totally uncorrelated = 0.0).
~/Shahed$
```

Figure 10:evaluate information density

The given code was modified to generate a 256-bit encryption key.

```
Open  [icon]
1 /*  task5.c  */
2 #include <stdio.h>
3 #include <stdlib.h>
4 #define LEN 32 // 256 bits
5
6 void main()
7 {
8
9     int i;
10    unsigned char *key = (unsigned char *)malloc(sizeof(unsigned char) * LEN);
11    FILE *random = fopen("/dev/urandom", "r");
12    for (i = 0; i < LEN; i++)
13    {
14        fread(key, sizeof(unsigned char) * LEN, 1, random);
15        printf("%.2x", *key);
16    }
17    printf("\n");
18    fclose(random);
19 }
```

```
shahed@shahed-VirtualBox:~$ touch task5.c
shahed@shahed-VirtualBox:~$ gedit task5.c
shahed@shahed-VirtualBox:~$ gcc task5.c -o task5
shahed@shahed-VirtualBox:~$ ./task5
d17ceff3430c3be494fc600185e75eaefb084a828cec58ab5c539bc545766569
shahed@shahed-VirtualBox:~$
```

Figure 11:Task5 execution

This is a true random number because it is read from /dev/urandom