APPLIED CRYPTOGRAPHY
Lab 3 : Random Number Generators

Name: Siri N Shetty SRN: PES2UG22CS556

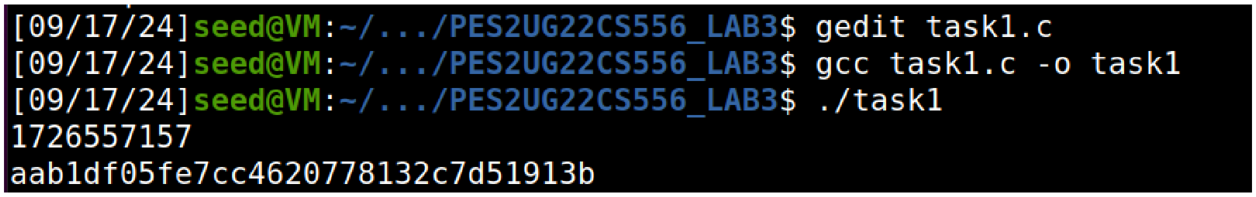
(use SEEDVM)

Task 1: Generate Encryption Key in a Wrong Way

Step 1 : Compile and Run given code.

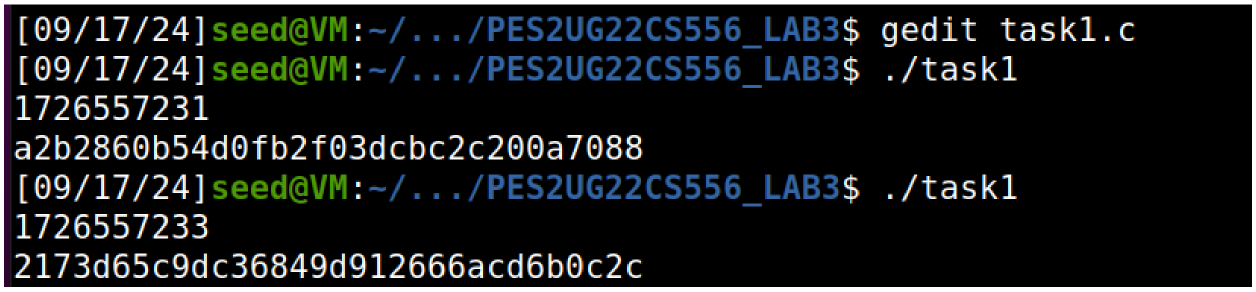
Step 2 : Comment ‘srand’ statement and run given code.

i) Output Screenshot for step 1



```
[09/17/24]seed@VM:~/.../PES2UG22CS556_LAB3$ gedit task1.c
[09/17/24]seed@VM:~/.../PES2UG22CS556_LAB3$ gcc task1.c -o task1
[09/17/24]seed@VM:~/.../PES2UG22CS556_LAB3$ ./task1
1726557157
aab1df05fe7cc4620778132c7d51913b
```

ii) Output Screenshot for step 2



```
[09/17/24]seed@VM:~/.../PES2UG22CS556_LAB3$ gedit task1.c
[09/17/24]seed@VM:~/.../PES2UG22CS556_LAB3$ ./task1
1726557231
a2b2860b54d0fb2f03dcbc2c200a7088
[09/17/24]seed@VM:~/.../PES2UG22CS556_LAB3$ ./task1
1726557233
2173d65c9dc36849d912666acd6b0c2c
```

iii) Explanation for difference between outputs of step 1 and 2

Scenario 1 (with srand(time(NULL))): Produces a different encryption key each time because the seed is based on the current system time.
If you comment out srand(time(NULL)), the RNG is not seeded with the current time, meaning it defaults to the same seed every time you run the program.
Scenario 2 (without srand(time(NULL))): Produces the same encryption key each time because the RNG seed is fixed.

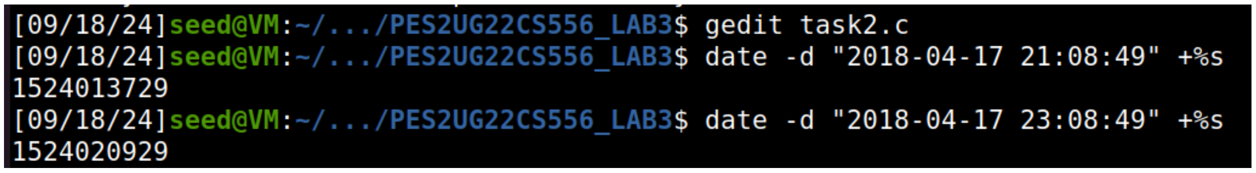
Task 2: Guessing the Key

Step 1: Run the command

Step 2: Run code to generate all keys in file

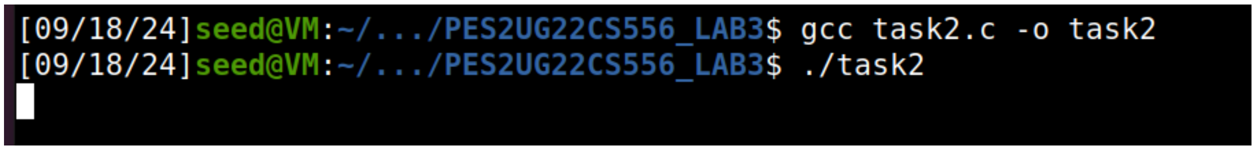
Step 3: Run code to find key match

i) Output Screenshot for step 1 : show value1,value2

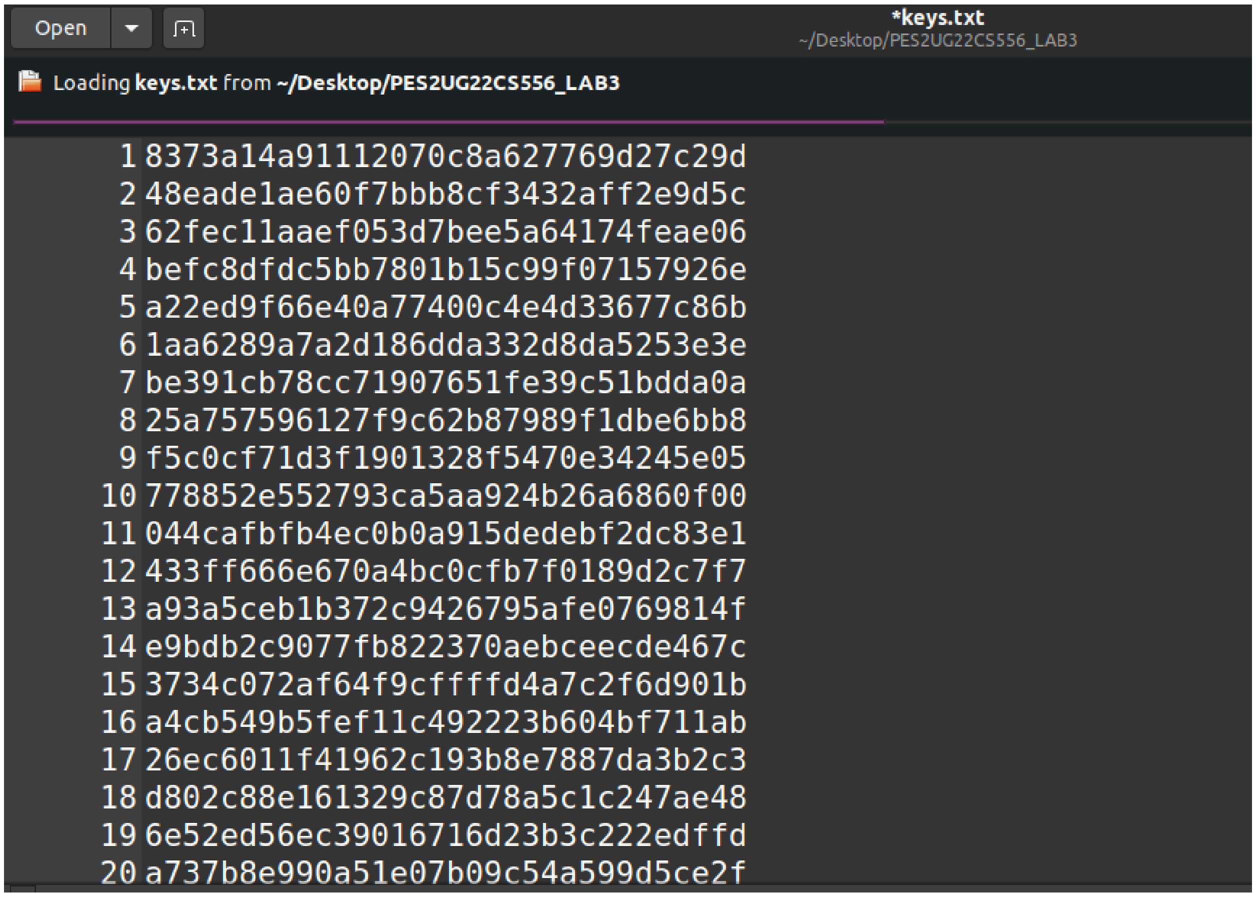


```
[09/18/24]seed@VM:~/.../PES2UG22CS556_LAB3$ gedit task2.c
[09/18/24]seed@VM:~/.../PES2UG22CS556_LAB3$ date -d "2018-04-17 21:08:49" +%s
1524013729
[09/18/24]seed@VM:~/.../PES2UG22CS556_LAB3$ date -d "2018-04-17 23:08:49" +%s
1524020929
```

ii) Output Screenshot for step 2 : show code run, contents of keys.txt



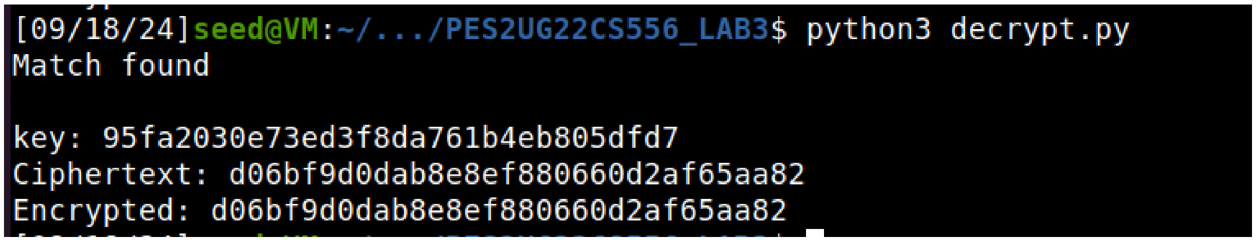
```
[09/18/24]seed@VM:~/.../PES2UG22CS556_LAB3$ gcc task2.c -o task2
[09/18/24]seed@VM:~/.../PES2UG22CS556_LAB3$ ./task2
```



Loading **keys.txt** from **~/Desktop/PES2UG22CS556_LAB3**

```
8373a14a91112070c8a627769d27c29d
48eade1ae60f7bbb8cf3432aff2e9d5c
62fec11aaef053d7bee5a64174feae06
befc8dfdc5bb7801b15c99f07157926e
a22ed9f66e40a77400c4e4d33677c86b
1aa6289a7a2d186dda332d8da5253e3e
be391cb78cc71907651fe39c51bdda0a
25a757596127f9c62b87989f1dbe6bb8
f5c0cf71d3f1901328f5470e34245e05
778852e552793ca5aa924b26a6860f00
044cafbfb4ec0b0a915dedebf2dc83e1
433ff666e670a4bc0cfb7f0189d2c7f7
a93a5ceb1b372c9426795afe0769814f
e9bdb2c9077fb822370aebceecde467c
3734c072af64f9cffffd4a7c2f6d901b
a4cb549b5fef11c492223b604bf711ab
26ec6011f41962c193b8e7887da3b2c3
d802c88e161329c87d78a5c1c247ae48
6e52ed56ec39016716d23b3c222edffd
a737b8e990a51e07b09c54a599d5ce2f
```

iii) Output Screenshot for step 3



```
[09/18/24]seed@VM:~/.../PES2UG22CS556_LAB3$ python3 decrypt.py
Match found

key: 95fa2030e73ed3f8da761b4eb805dfd7
Ciphertext: d06bf9d0dab8e8ef880660d2af65aa82
Encrypted: d06bf9d0dab8e8ef880660d2af65aa82
```

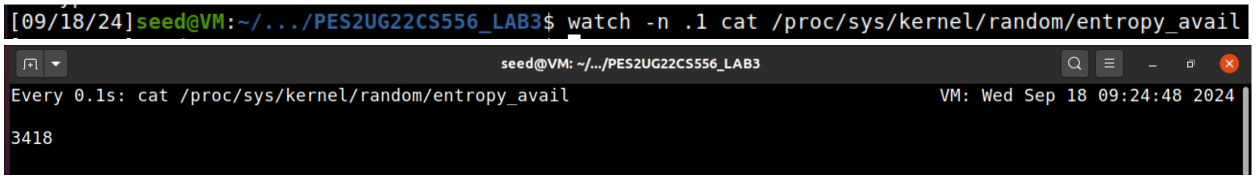
iv) Explanation of how the code finds the key

- It will discover the correct key through brute forcing all possible keys generated during the valid time frame, which happens to be 7200 seconds.
- It reads every key from keys.txt, converts to bytes and applies AES in CBC mode to encrypt known plaintext.
- The encrypted text then will be compared with the provided ciphertext.
- In case they match, it has identified the correct key, and that is printed.
- It continues with a cycle until the correct key is found.

Task 3 : Measure the Entropy of Kernel

Step 1 : Run the command

i) Output Screenshot for step 1



```
[09/18/24]seed@VM:~/.../PES2UG22CS556_LAB3$ watch -n .1 cat /proc/sys/kernel/random/entropy_avail
```

```
Every 0.1s: cat /proc/sys/kernel/random/entropy_avail                    VM: Wed Sep 18 09:24:48 2024

3418
```

ii) Explanation of the system measures entropy

The system measures entropy by collecting random events coming from hardware sources (e.g., keyboard inputs, mouse movements) for the purpose of generating random numbers. This randomness is stored in an entropy pool and its availability the only truly critical factor which determines whether or not certain cryptographic operations, and other security-sensitive tasks can be carried out.

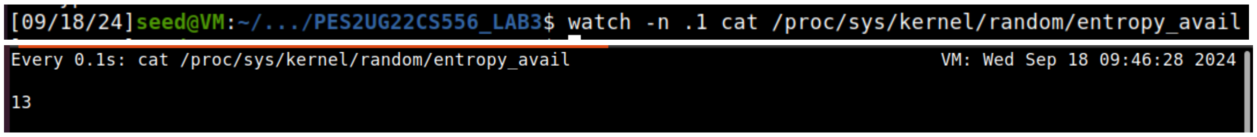
iii) Explanation of effect of cursor movement to entropy value

More entropy is introduced by any movement of the mouse or the feeling of keys under the fingers. The hard interrupts caused by such actions are unpredictable, and the kernel collects all events for the growth of the entropy pool. As we can see, the value of entropy increases with each action made with the system because it receives new unpredictable information from our actions.

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

Step 1: Run both commands simultaneously

i) Output Screenshot for step initially



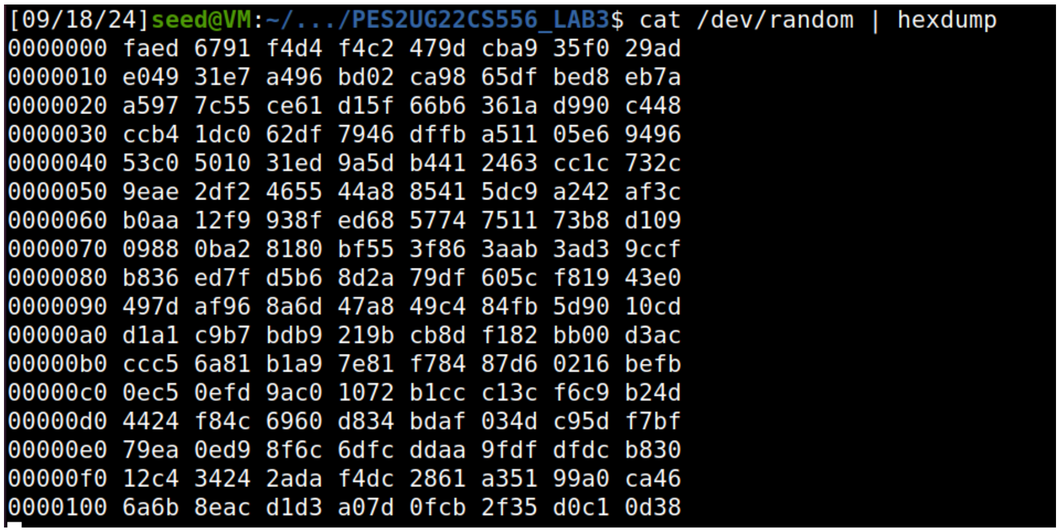
```
[09/18/24]seed@VM:~/.../PES2UG22CS556_LAB3$ watch -n .1 cat /proc/sys/kernel/random/entropy_avail
Every 0.1s: cat /proc/sys/kernel/random/entropy_avail                     VM: Wed Sep 18 09:46:28 2024

13
```

ii) What happens if you do not move your mouse or type anything. Then, randomly move your mouse and see whether you can observe any difference. Explain your observation.

Without Assuming Keystroke Activity or Pointing: If we execute both commands concurrently and leave the system entirely alone, then the entropy within the system should start collapsing because the /dev/random device is being depleted of available entropy. Since /dev/random only creates data when there is enough entropy, it will stop creating random numbers when the pool of entropy runs out. This is still increasing in my system. The cause for this maybe some system activities or maybe for Virtual Machine as the cloud infrastructure might feed entropy into guest OS which can cause entropy pool to increase without any Interaction.

With Mouse Movement or Typing: When we start moving the mouse or start typing, the entropy pool is refilled and we would see /dev/random to print random number again. It is because the system waits for unpredictable events such as mouse movement to collect more entropy.

iii) Output Screenshot for (ii)



```
[09/18/24]seed@VM:~/.../PES2UG22CS556_LAB3$ cat /dev/random | hexdump
0000000 faed 6791 f4d4 f4c2 479d cba9 35f0 29ad
0000010 e049 31e7 a496 bd02 ca98 65df bed8 eb7a
0000020 a597 7c55 ce61 d15f 66b6 361a d990 c448
0000030 ccb4 1dc0 62df 7946 dffb a511 05e6 9496
0000040 53c0 5010 31ed 9a5d b441 2463 cc1c 732c
0000050 9eae 2df2 4655 44a8 8541 5dc9 a242 af3c
0000060 b0aa 12f9 938f ed68 5774 7511 73b8 d109
0000070 0988 0ba2 8180 bf55 3f86 3aab 3ad3 9ccf
0000080 b836 ed7f d5b6 8d2a 79df 605c f819 43e0
0000090 497d af96 8a6d 47a8 49c4 84fb 5d90 10cd
00000a0 d1a1 c9b7 bdb9 219b cb8d f182 bb00 d3ac
00000b0 ccc5 6a81 b1a9 7e81 f784 87d6 0216 befb
00000c0 0ec5 0efd 9ac0 1072 b1cc c13c f6c9 b24d
00000d0 4424 f84c 6960 d834 bdaf 034d c95d f7bf
00000e0 79ea 0ed9 8f6c 6dfc ddaa 9fdf dfdc b830
00000f0 12c4 3424 2ada f4dc 2861 a351 99a0 ca46
0000100 6a6b 8eac d1d3 a07d 0fcb 2f35 d0c1 0d38
```

iii) Explain how dev/random generates a random number

- The dev/random will generate random numbers seeded from the entropy pool. It will only output random data when enough entropy has been collected, which is acquired from hardware-based events such as the movement of a computer mouse, keyboard input, and any other unpredictable system events.
- When the entropy level is low, dev/random blocks until enough entropy has accumulated so that it may be used for cryptographic applications that require randomness of good quality.

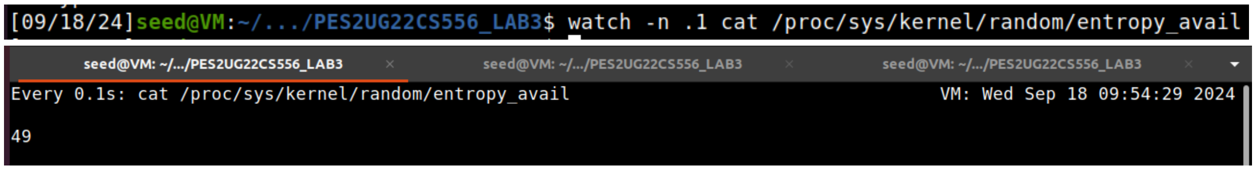
<u>Task 5 : Get Pseudo Random Numbers from /dev/urandom</u>

Step 1 : Run both commands simultaneously

Step 2 : Measure the quality of the random number.

Step 3 : Run modified code

i) Output Screenshot for step 1

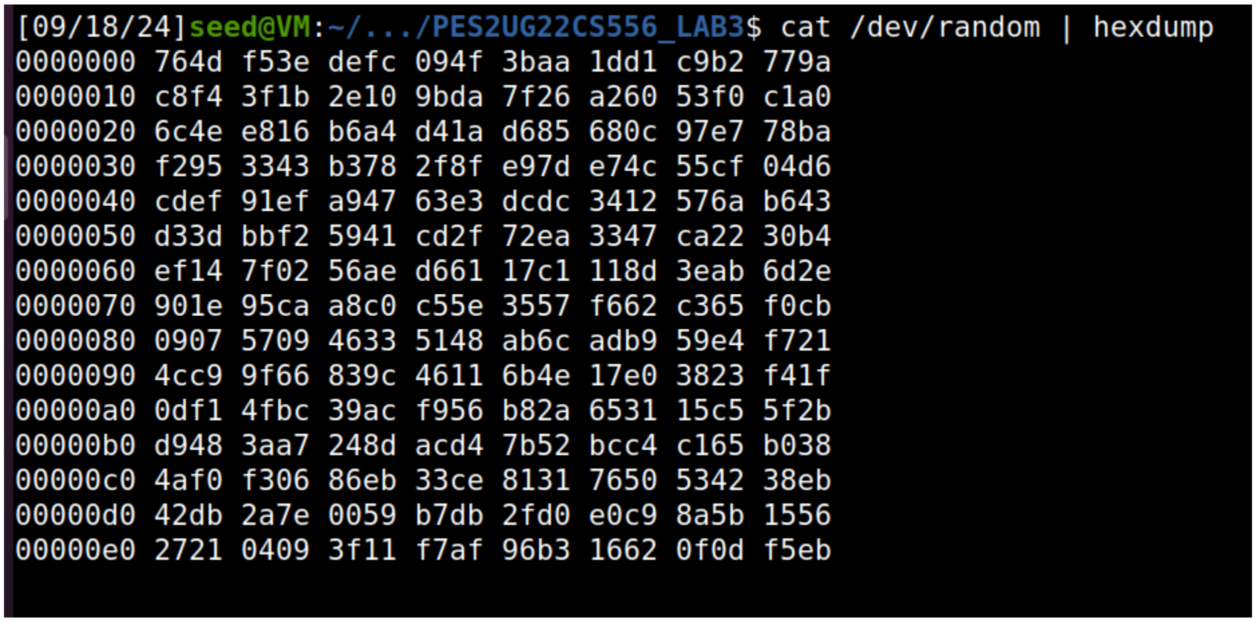


```
[09/18/24]seed@VM:~/.../PES2UG22CS556_LAB3$ watch -n .1 cat /proc/sys/kernel/random/entropy_avail
```

seed@VM: ~/.../PES2UG22CS556_LAB3
seed@VM: ~/.../PES2UG22CS556_LAB3
seed@VM: ~/.../PES2UG22CS556_LAB3

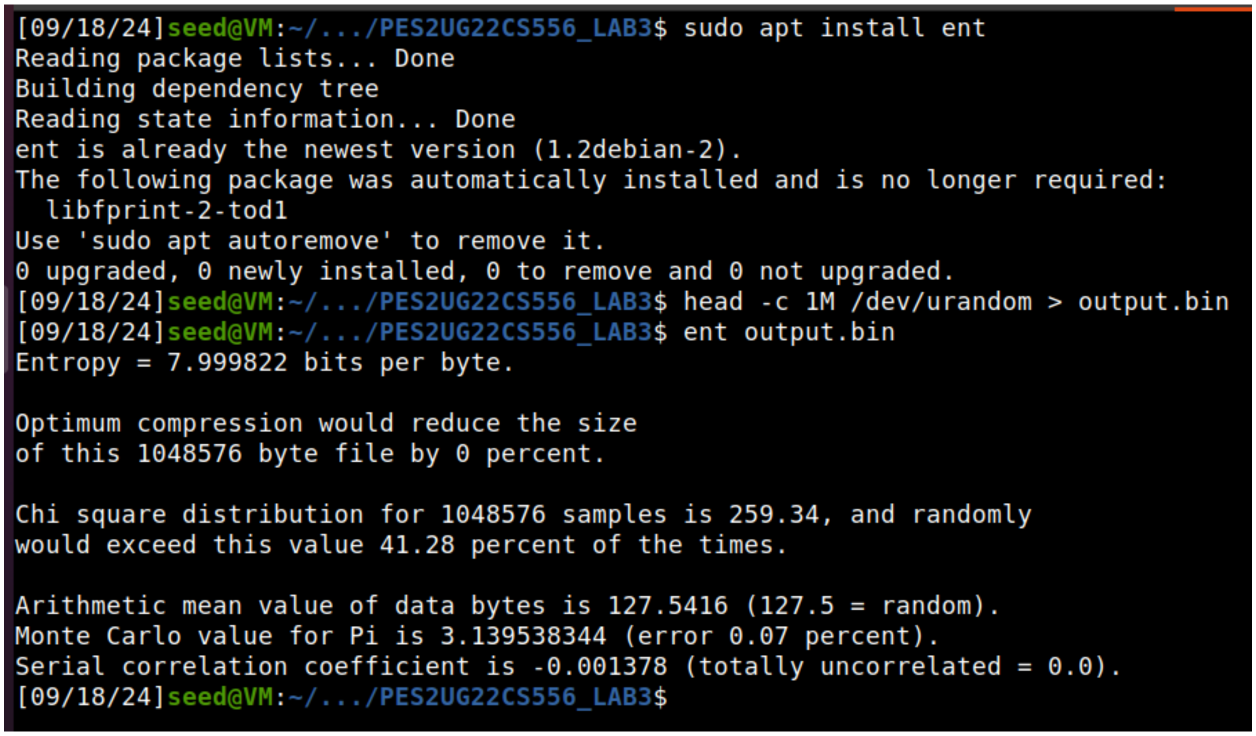
```
Every 0.1s: cat /proc/sys/kernel/random/entropy_avail                VM: Wed Sep 18 09:54:29 2024

49
```

ii) Output Screenshot for step 2



```
[09/18/24]seed@VM:~/.../PES2UG22CS556_LAB3$ cat /dev/random | hexdump
0000000 764d f53e defc 094f 3baa 1dd1 c9b2 779a
0000010 c8f4 3f1b 2e10 9bda 7f26 a260 53f0 c1a0
0000020 6c4e e816 b6a4 d41a d685 680c 97e7 78ba
0000030 f295 3343 b378 2f8f e97d e74c 55cf 04d6
0000040 cdef 91ef a947 63e3 dcdc 3412 576a b643
0000050 d33d bbf2 5941 cd2f 72ea 3347 ca22 30b4
0000060 ef14 7f02 56ae d661 17c1 118d 3eab 6d2e
0000070 901e 95ca a8c0 c55e 3557 f662 c365 f0cb
0000080 0907 5709 4633 5148 ab6c adb9 59e4 f721
0000090 4cc9 9f66 839c 4611 6b4e 17e0 3823 f41f
00000a0 0df1 4fbc 39ac f956 b82a 6531 15c5 5f2b
00000b0 d948 3aa7 248d acd4 7b52 bcc4 c165 b038
00000c0 4af0 f306 86eb 33ce 8131 7650 5342 38eb
00000d0 42db 2a7e 0059 b7db 2fd0 e0c9 8a5b 1556
00000e0 2721 0409 3f11 f7af 96b3 1662 0f0d f5eb
```

iii) Output Screenshot for step 3



```
[09/18/24]seed@VM:~/.../PES2UG22CS556_LAB3$ sudo apt install ent
Reading package lists... Done
Building dependency tree
Reading state information... Done
ent is already the newest version (1.2debian-2).
The following package was automatically installed and is no longer required:
  libfprint-2-tod1
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
[09/18/24]seed@VM:~/.../PES2UG22CS556_LAB3$ head -c 1M /dev/urandom > output.bin
[09/18/24]seed@VM:~/.../PES2UG22CS556_LAB3$ ent output.bin
Entropy = 7.999822 bits per byte.

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

Chi square distribution for 1048576 samples is 259.34, and randomly
would exceed this value 41.28 percent of the times.

Arithmetic mean value of data bytes is 127.5416 (127.5 = random).
Monte Carlo value for Pi is 3.139538344 (error 0.07 percent).
Serial correlation coefficient is -0.001378 (totally uncorrelated = 0.0).
[09/18/24]seed@VM:~/.../PES2UG22CS556_LAB3$
```

iv) Explain how dev/urandom generates a random number in contrast to dev/random.

dev/urandom:

- dev/urandom taps a finite pool of entropy accrued from sources of environmental noise such as keyboard timing, mouse movements, and hardware events.
- Whenever the system's entropy pool falls low, it will block until more entropy is accumulated to ensure that the generated random number has adequate quality of randomness. Thus, it is highly suited for cryptographic applications where unpredictability is crucial.
- dev/urandom draws from the entropy pool but does not block if the entropy pool is low.
- It seeds it off the pool of initial entropy but keeps producing the numbers even when the entropy pool is exhausted.
- As it doesn't block, /dev/urandom is faster and more suited for general-purpose random number generation, especially when large amounts of data are needed. It's still safe for most cryptographic operations, but it may not be as secure as /dev/random in low-entropy conditions.