

CRYPTOGRAPHY LAB- WEEK 1

Pseudo Random Number Generation

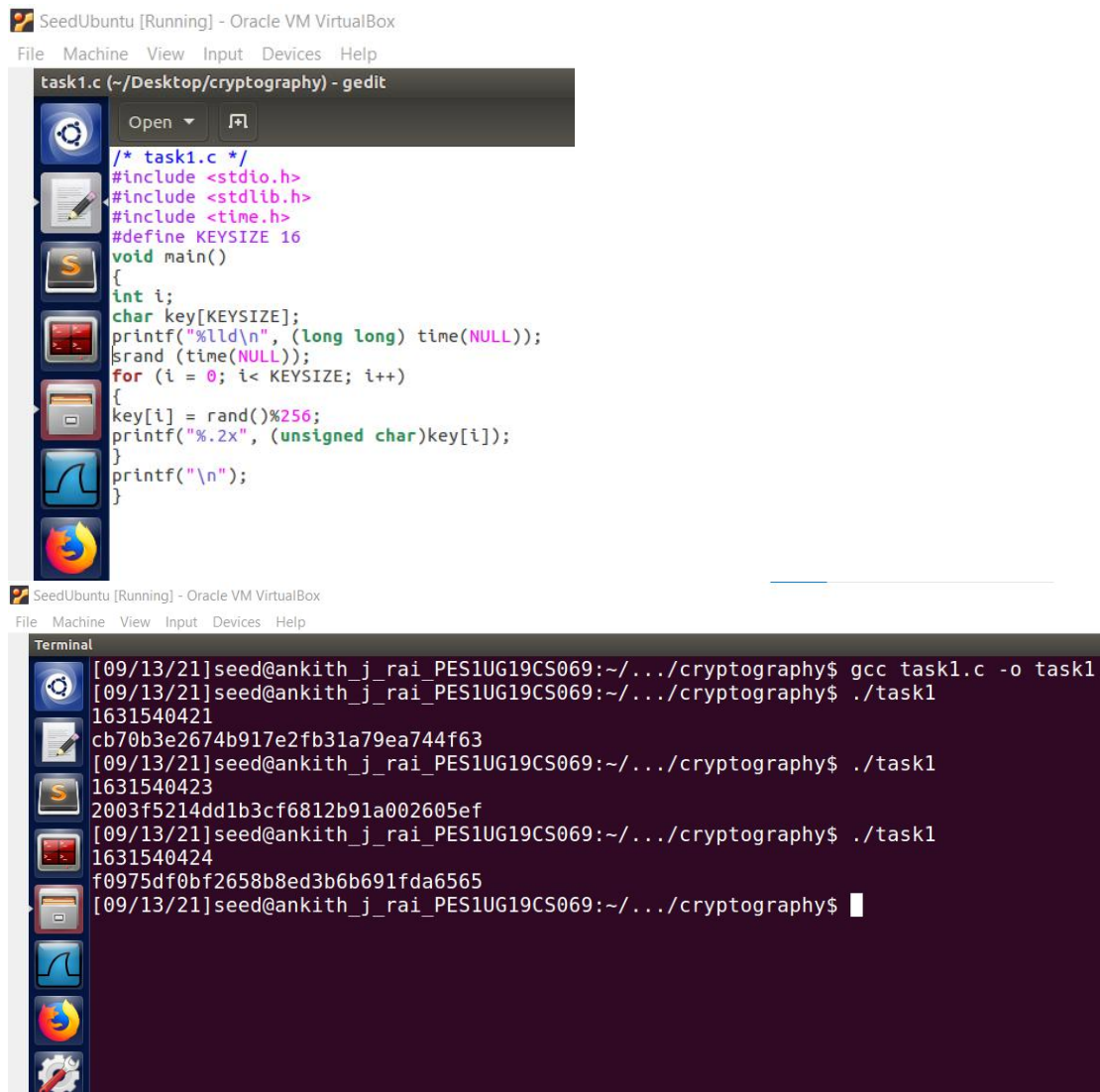
NAME : Ankith J Rai

SRN : PES1UG19CS069

SEC : B

Task 1: Generate Encryption Key in a Wrong Way

Step 1: In this step a 128 bit encryption key is generated using the computer time as parameter. Hence each time the program is run a different key is generated.



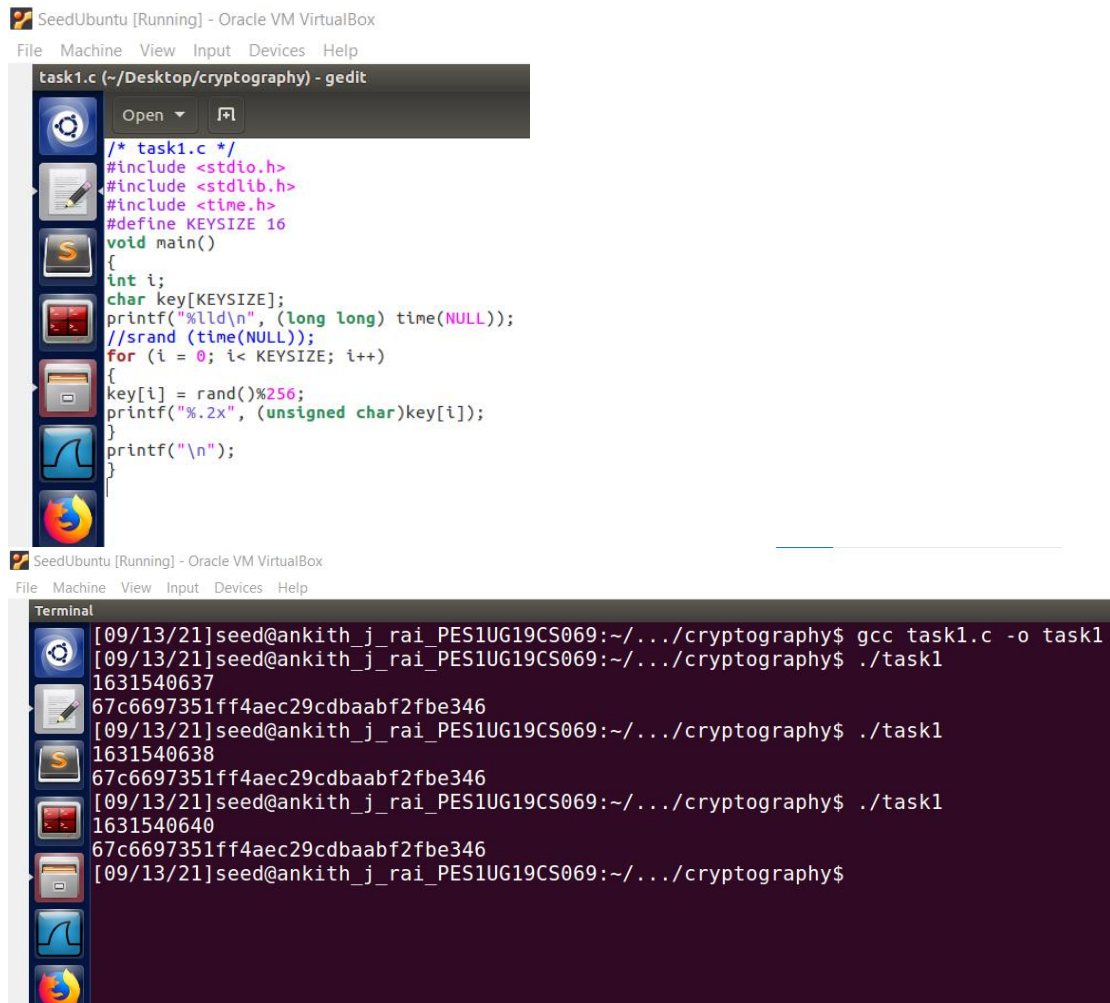
The image shows a SeedUbuntu [Running] - Oracle VM VirtualBox window. The top pane displays the source code for `task1.c` in a gedit editor. The code includes `<stdio.h>`, `<stdlib.h>`, and `<time.h>`, defines `KEYSIZE` as 16, and implements a `main` function that generates a 16-byte key using `rand()` seeded with `time(NULL)`.

```
/* task1.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");
}
```

The bottom pane shows a terminal window with the following commands and output:

```
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/.../cryptography$ gcc task1.c -o task1
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/.../cryptography$ ./task1
1631540421
cb70b3e2674b917e2fb31a79ea744f63
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/.../cryptography$ ./task1
1631540423
2003f5214dd1b3cf6812b91a002605ef
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/.../cryptography$ ./task1
1631540424
f0975df0bf2658b8ed3b6b691fda6565
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/.../cryptography$
```

Step 2: Here also a 128 bit encryption key is generated but the same key is generated even after running the program multiple times this is because the `srand(time(NULL))` has been commented out.



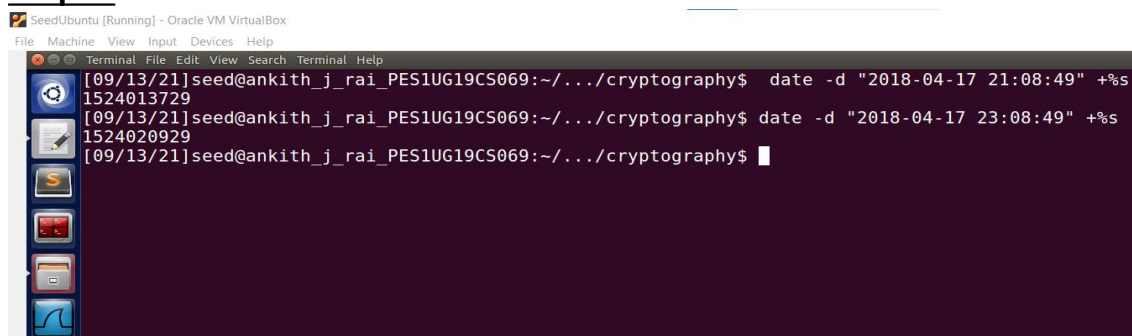
The screenshot shows a SeedUbuntu [Running] - Oracle VM VirtualBox window. The top pane displays a C program named `task1.c` in a gedit editor. The code includes `<stdio.h>`, `<stdlib.h>`, and `<time.h>`, defines `KEYSIZE 16`, and contains a `main` function that prints the time, generates a key using `rand()`, and prints the key. The bottom pane shows a terminal window where the program is compiled and run multiple times, showing the same key: `67c6697351ff4aec29cdbaabf2fbe346`.

```
/* task1.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");
}
```

```
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ gcc task1.c -o task1
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ ./task1
1631540637
67c6697351ff4aec29cdbaabf2fbe346
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ ./task1
1631540638
67c6697351ff4aec29cdbaabf2fbe346
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ ./task1
1631540640
67c6697351ff4aec29cdbaabf2fbe346
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$
```

Task 2: Guessing the Key

Step 1:



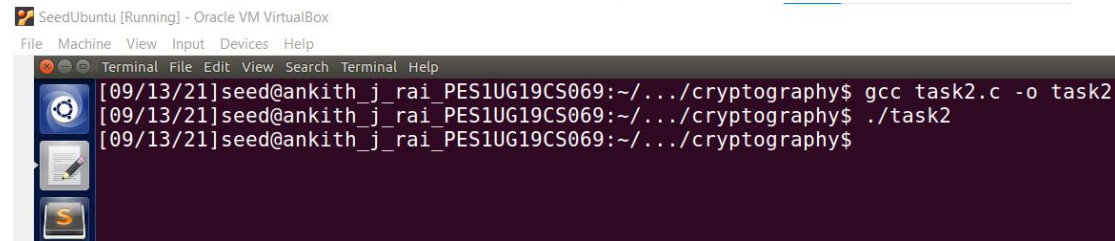
The screenshot shows a SeedUbuntu [Running] - Oracle VM VirtualBox window. The terminal window displays the following commands and output:

```
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ date -d "2018-04-17 21:08:49" +%s
1524013729
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ date -d "2018-04-17 23:08:49" +%s
1524020929
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$
```

Here we calculate the time in seconds till the specified date. Hence we get **Value1** as **1524013729** and **Value2** as **1524020929**.

Step 2:

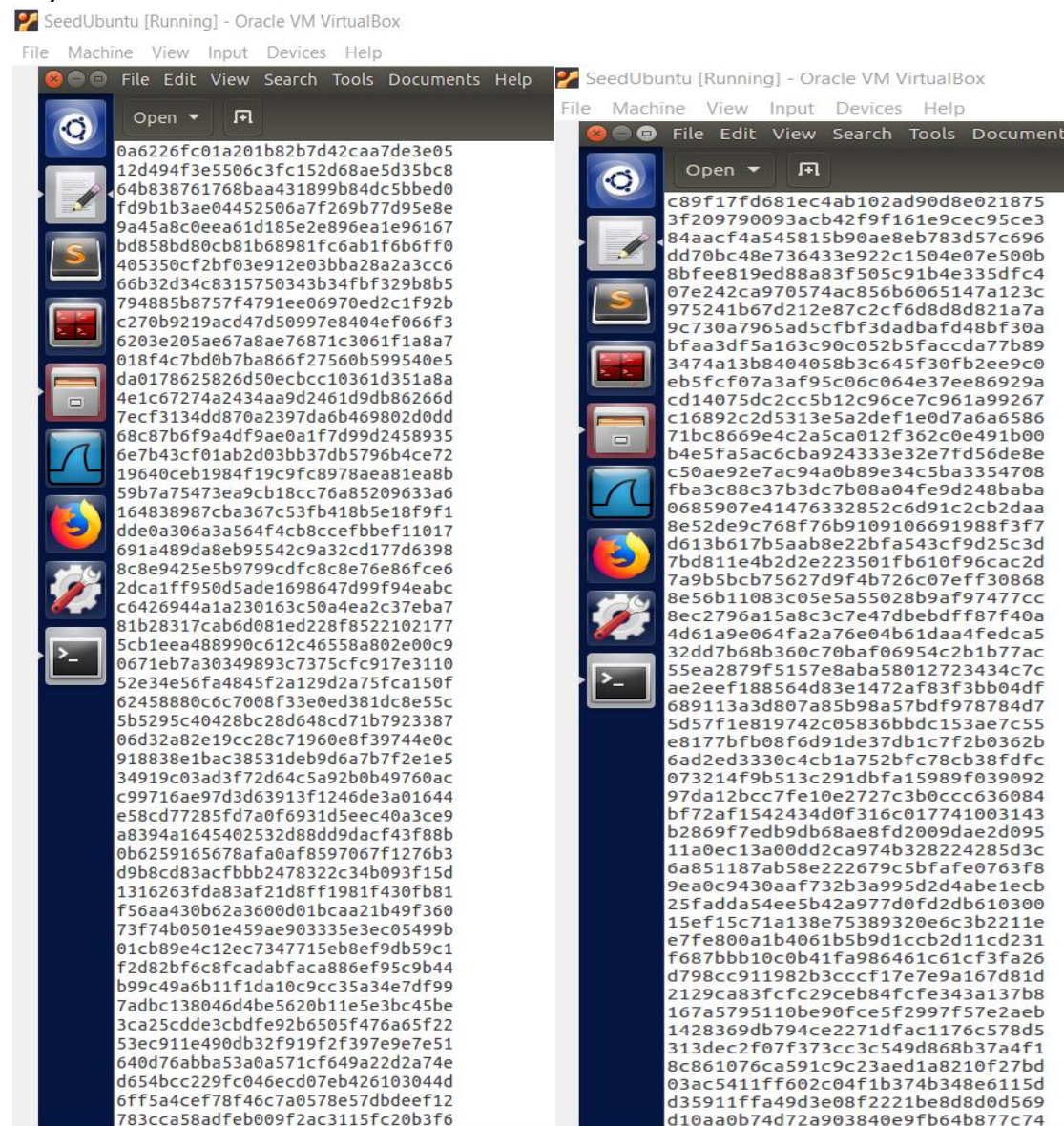
Screenshot of the terminal after running task2.c



```
SeedUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal File Edit View Search Terminal Help
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ gcc task2.c -o task2
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ ./task2
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$
```

Once we run task2.c all the keys that are possible using the range of time values between value1 and value2 is generated and stored in a file called keys.txt .

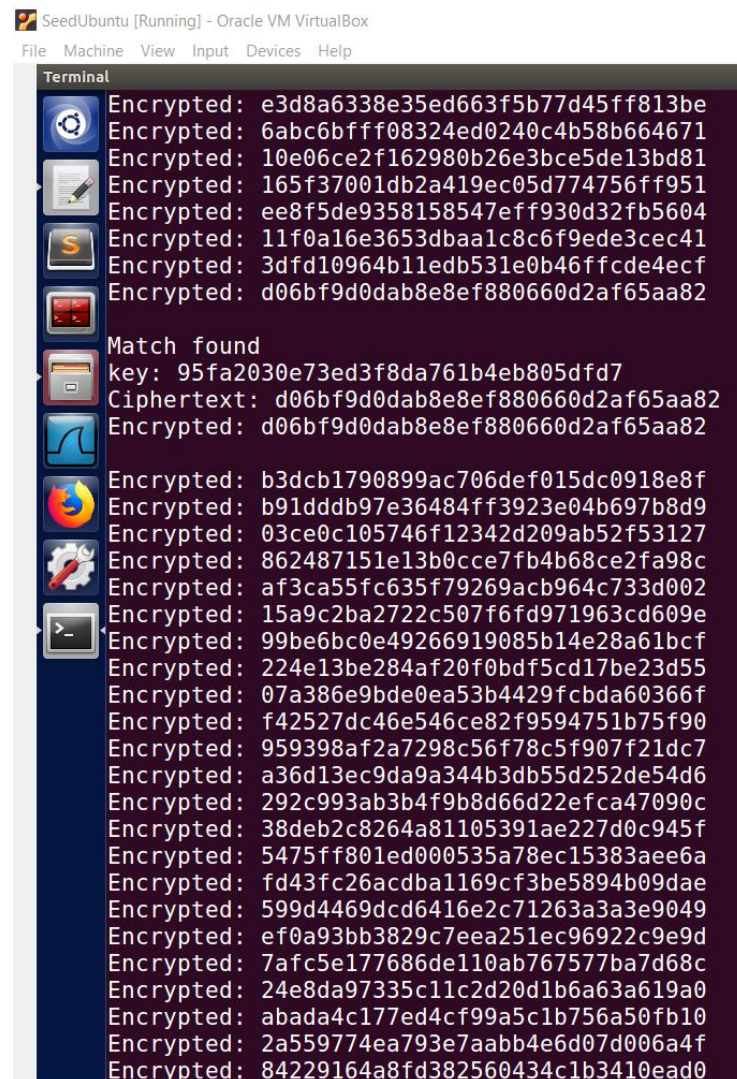
Some of the screenshot of the keys.txt file which contains all possible keys.



```
SeedUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Open
0a6226fc01a201b82b7d42caa7de3e05
12d494f3e5506c3fc152d68ae5d35bc8
64b838761768baa431899b84dc5bbcd0
fd9b1b3ae04452506a7f269b77d95e8e
9a45a8c0eea61d185e2e896ea1e96167
bd858bd80cb81b68981fc6ab1f6b6ff0
405350cf2bf03e912e03bba28a2a3cc6
66b32d34c8315750343b34bf329b8b5
794885b8757f4791ee06970ed2c1f92b
c270b9219acd47d50997e8404ef066f3
6203e205ae67a8ae76871c3061f1a8a7
018f4c7bd0b7ba866f27560b599540e5
da0178625826d50ecbcc10361d351a8a
4e1c67274a2434aa9d2461d9db86266d
7ecf3134dd870a2397da6b469802d0dd
68c87b6f9a4df9ae0a1f7d99d2458935
6e7b43cf01ab2d03bb37db5796b4ce72
19640ceb1984f19c9cf8978aea81ea8b
59b7a75473ea9cb18cc76a85209633a6
164838987cba367c53fb418b5e18f9f1
dde0a306a3a564f4cb8ccfbbef11017
691a489da8eb95542c9a32cd177d6398
8c8e9425e5b9799cdfc8c8e76e86fce6
2dca1ff950d5ade1698647d99f94eabc
c6426944a1a230163c50a4ea2c37eba7
81b28317cab6d081ed228f8522102177
5cb1eea488990c612c46558a802e00c9
0671eb7a30349893c7375cfc917e3110
52e34e56fa4845f2a129d2a75fca150f
62458880c6c7008f33e0ed381dc8e55c
5b5295c40428bc28d648cd71b7923387
06d32a82e19cc28c71960e8f39744e0c
918838e1bac38531deb9d6a7b7f2e1e5
34919c03ad3f72d64c5a92b0b49760ac
c99716ae97d3d63913f1246de3a01644
e58cd77285fd7a0f6931d5eecd40a3ce9
a8394a1645402532d88dd9dacf43f88b
0b6259165678afa0af8597067f1276b3
d9b8cd83acfbb32478322c34b093f15d
1316263fda83af21d8ff1981f430fb81
f56aa430b62a3600d01bcaa21b49f360
73f74b0501e459ae903335e3ec05499b
01cb89e4c12ec7347715eb8ef9db59c1
f2d82bf6c8fcadabfaca886ef95c9b44
b99c49a6b11f1da10c9cc35a34e7df99
7adbc138046d4be5620b11e5e3bc45be
3ca25cdde3c3bdf92b6505f476a65f22
53ec911e490db32f919f2f397e9e7e51
640d76abba53a0a571cf649a22d2a74e
d654bcc229fc046ecd07eb426103044d
6ff5a4cef78f46c7a0578e57dbdeef12
783cca58adfeb009f2ac3115fc20b3f6

SeedUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Open
c89f17fd681ec4ab102ad90d8e021875
3f209790093acb42f9f161e9cec95ce3
84aacf4a545815b90ae8eb783d57c696
dd70bc48e736433e922c1504e07e500b
8bfee819ed88a83f505c91b4e335dfc4
07e242ca970574ac856b6065147a123c
975241b67d212e87c2cf6d8d8d21a7a
9c730a7965ad5cfbf3dadabaf48bf30a
bfaa3df5a163c90c052b5faccda77b89
3474a13b8404058b3c645f30fb2ee9c0
eb5fcf07a3af95c06c064e37ee86929a
cd14075dc2cc5b12c96ce7c961a99267
c16892c2d5313e5a2def1e0d7a6a6586
71bc8669e4c2a5ca012f362c0e491b00
b4e5fa5ac6cba924333e32e7fd56de8e
c50ae92e7ac94a0b89e34c5ba3354708
fba3c88c37b3dc7b08a04fe9d248baba
0685907e41476332852c6d91c2cb2daa
8e52de9c768f76b9109106691988f3f7
d613b617b5aab8e22bfa543cf9d25c3d
7bd811e4b2d2e223501fb610f96cac2d
7a9b5bcb75627d9f4b726c07eff30868
8e56b11083c05e5a55028b9af97477cc
8ec2796a15a8c3c7e47dbdbdf87f40a
4d61a9e064fa2a76e04b61daa4fedca5
32dd7b68b360c70bafe06954c2b1b77ac
55ea2879f5157e8aba58012723434c7c
ae2eef188564d83e1472af83f3bb04df
689113a3d807a85b98a57bdf978784d7
5d57f1e819742c05836bbdc153ae7c55
e8177bfb08f6d91de37db1c7f2b0362b
6ad2ed3330c4cb1a752bfc78cb38fdcf
073214f9b513c291dbfa15989f039092
97da12bcc7fe10e2727c3b0ccc636084
bf72af1542434d0f316c017741003143
b2869f7edb9db68ae8fd2009dae2d095
11a0ec13a00dd2ca974b328224285d3c
6a851187ab58e222679c5bfafef0763f8
9ea0c9430aaf732b3a995d2d4abe1ecb
25fadda54ee5b42a977d0fd2db610300
15ef15c71a138e75389320e6c3b2211e
e7fe800a1b4061b5b9d1ccb2d11cd231
f687bbb10c0b41fa986461c61cf3fa26
d798cc911982b3ccc17e7e9a167d81d
2129ca83fcfc29ceb84fcfe343a137b8
167a5795110be90fce5f2997f57e2aeb
1428369db794ce2271dfac1176c578d5
313dec2f07f373cc3c549d868b37a4f1
8c861076ca591c9c23aed1a8210f27bd
03ac5411ff602c04f1b374b348e6115d
d35911ffa49d3e08f2221be8d8d0569
d10aa0b74d72a903840e9fb64b877c74
```


Step 3: Here we use brute force method to guess the key. For this we run the decrypt.c to compare the cipherText and the encryption of the plain text using the keys present in the keys.txt . When a match is found we can get to know the key used in encryption.



```
SeedUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

Terminal
Encrypted: e3d8a6338e35ed663f5b77d45ff813be
Encrypted: 6abc6bffff08324ed0240c4b58b664671
Encrypted: 10e06ce2f162980b26e3bce5de13bd81
Encrypted: 165f37001db2a419ec05d774756ff951
Encrypted: ee8f5de9358158547eff930d32fb5604
Encrypted: 11f0a16e3653dbaa1c8c6f9ede3cec41
Encrypted: 3dfd10964b11edb531e0b46ffcd4ecf
Encrypted: d06bf9d0dab8e8ef880660d2af65aa82

Match found
key: 95fa2030e73ed3f8da761b4eb805dfd7
Ciphertext: d06bf9d0dab8e8ef880660d2af65aa82
Encrypted: d06bf9d0dab8e8ef880660d2af65aa82

Encrypted: b3dcb1790899ac706def015dc0918e8f
Encrypted: b91dddb97e36484ff3923e04b697b8d9
Encrypted: 03ce0c105746f12342d209ab52f53127
Encrypted: 862487151e13b0cce7fb4b68ce2fa98c
Encrypted: af3ca55fc635f79269acb964c733d002
Encrypted: 15a9c2ba2722c507f6fd971963cd609e
Encrypted: 99be6bc0e49266919085b14e28a61bcf
Encrypted: 224e13be284af20f0bdf5cd17be23d55
Encrypted: 07a386e9bde0ea53b4429fcbda60366f
Encrypted: f42527dc46e546ce82f9594751b75f90
Encrypted: 959398af2a7298c56f78c5f907f21dc7
Encrypted: a36d13ec9da9a344b3db55d252de54d6
Encrypted: 292c993ab3b4f9b8d66d22efca47090c
Encrypted: 38deb2c8264a81105391ae227d0c945f
Encrypted: 5475ff801ed000535a78ec15383aee6a
Encrypted: fd43fc26acdba1169cf3be5894b09dae
Encrypted: 599d4469dcd6416e2c71263a3a3e9049
Encrypted: ef0a93bb3829c7eea251ec96922c9e9d
Encrypted: 7afc5e177686de110ab767577ba7d68c
Encrypted: 24e8da97335c11c2d20d1b6a63a619a0
Encrypted: abada4c177ed4cf99a5c1b756a50fb10
Encrypted: 2a559774ea793e7aabb4e6d07d006a4f
Encrypted: 84229164a8fd382560434c1b3410ead0
```

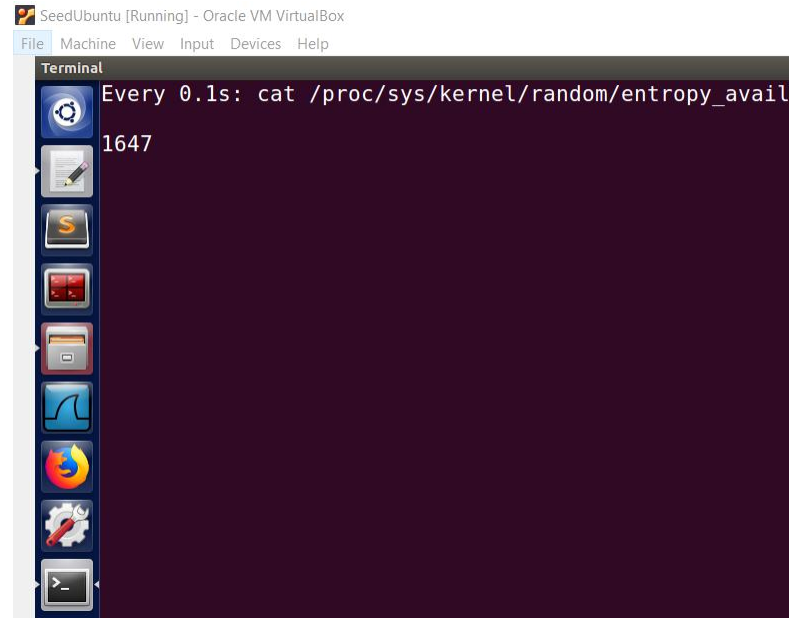
Task 3: Measure the Entropy of Kernel

In this task we find the entropy of the kernel which is the number of bits of random numbers the system currently has.

SeedUbuntu [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

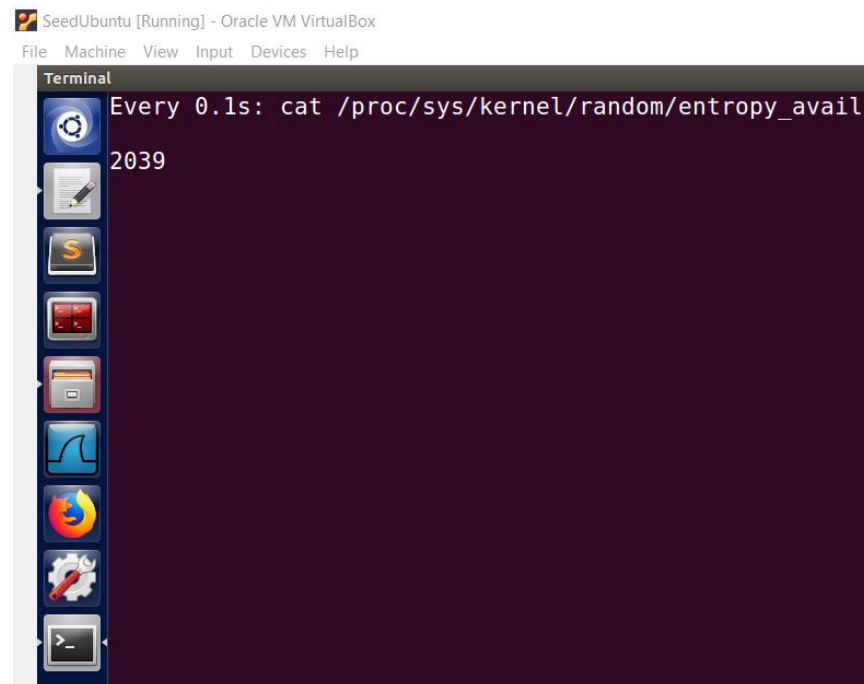
```
Terminal
Every 0.1s: cat /proc/sys/kernel/random/entropy_avail
1647
```

A screenshot of a terminal window within an Oracle VM VirtualBox. The terminal title bar reads 'SeedUbuntu [Running] - Oracle VM VirtualBox'. The menu bar includes 'File', 'Machine', 'View', 'Input', 'Devices', and 'Help'. The terminal content shows a command 'Every 0.1s: cat /proc/sys/kernel/random/entropy_avail' followed by the output '1647'. The terminal has a dark purple background and a light blue border. On the left side of the terminal window, there is a vertical toolbar with icons for various applications like a file manager, terminal, and web browser.

SeedUbuntu [Running] - Oracle VM VirtualBox

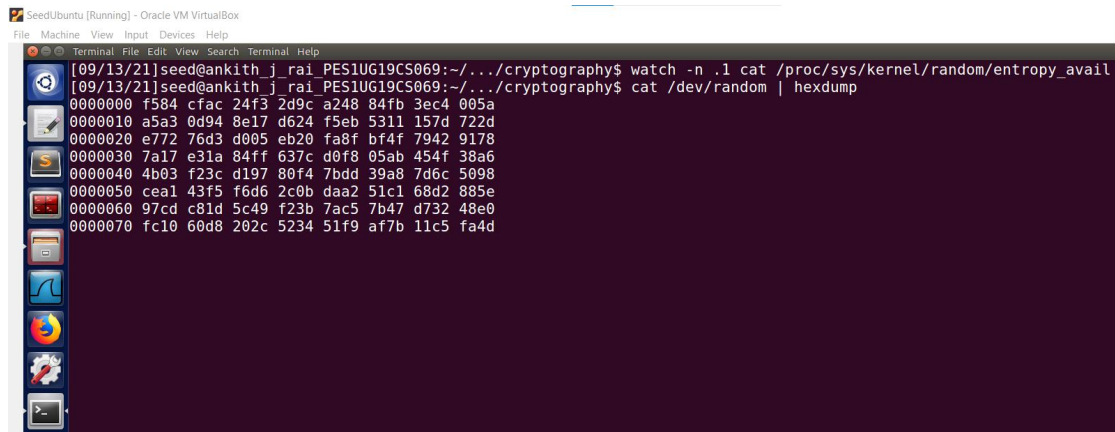
File Machine View Input Devices Help

```
Terminal
Every 0.1s: cat /proc/sys/kernel/random/entropy_avail
2039
```

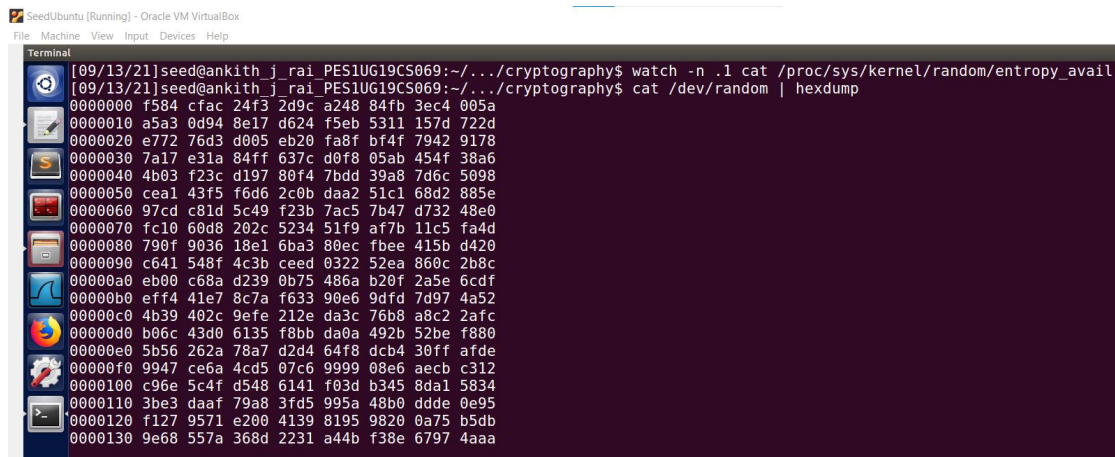
A screenshot of a terminal window within an Oracle VM VirtualBox, similar to the one above. The terminal title bar reads 'SeedUbuntu [Running] - Oracle VM VirtualBox'. The menu bar includes 'File', 'Machine', 'View', 'Input', 'Devices', and 'Help'. The terminal content shows the same command 'Every 0.1s: cat /proc/sys/kernel/random/entropy_avail' but with a different output, '2039'. The terminal has a dark purple background and a light blue border. On the left side of the terminal window, there is a vertical toolbar with icons for various applications like a file manager, terminal, and web browser.

By moving the mouse and pressing keys it is noted that the entropy of the kernel has been increased.

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



```
SeedUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal File Edit View Search Terminal Help
[09/13/21]seed@ankith_j_rai PES1UG19CS069:~/.../cryptology$ watch -n .1 cat /proc/sys/kernel/random/entropy_avail
[09/13/21]seed@ankith_j_rai PES1UG19CS069:~/.../cryptology$ cat /dev/random | hexdump
00000000 f584 cfac 24f3 2d9c a248 84fb 3ec4 005a
00000010 a5a3 0d94 8e17 d624 f5eb 5311 157d 722d
00000020 e772 76d3 d005 eb20 fa8f bf4f 7942 9178
00000030 7a17 e31a 84ff 637c d0f8 05ab 454f 38a6
00000040 4b03 f23c d197 80f4 7bdd 39a8 7d6c 5098
00000050 ceal 43f5 f6d6 2c0b daa2 51c1 68d2 885e
00000060 97cd c81d 5c49 f23b 7ac5 7b47 d732 48e0
00000070 fc10 60d8 202c 5234 51f9 af7b 11c5 fa4d
```

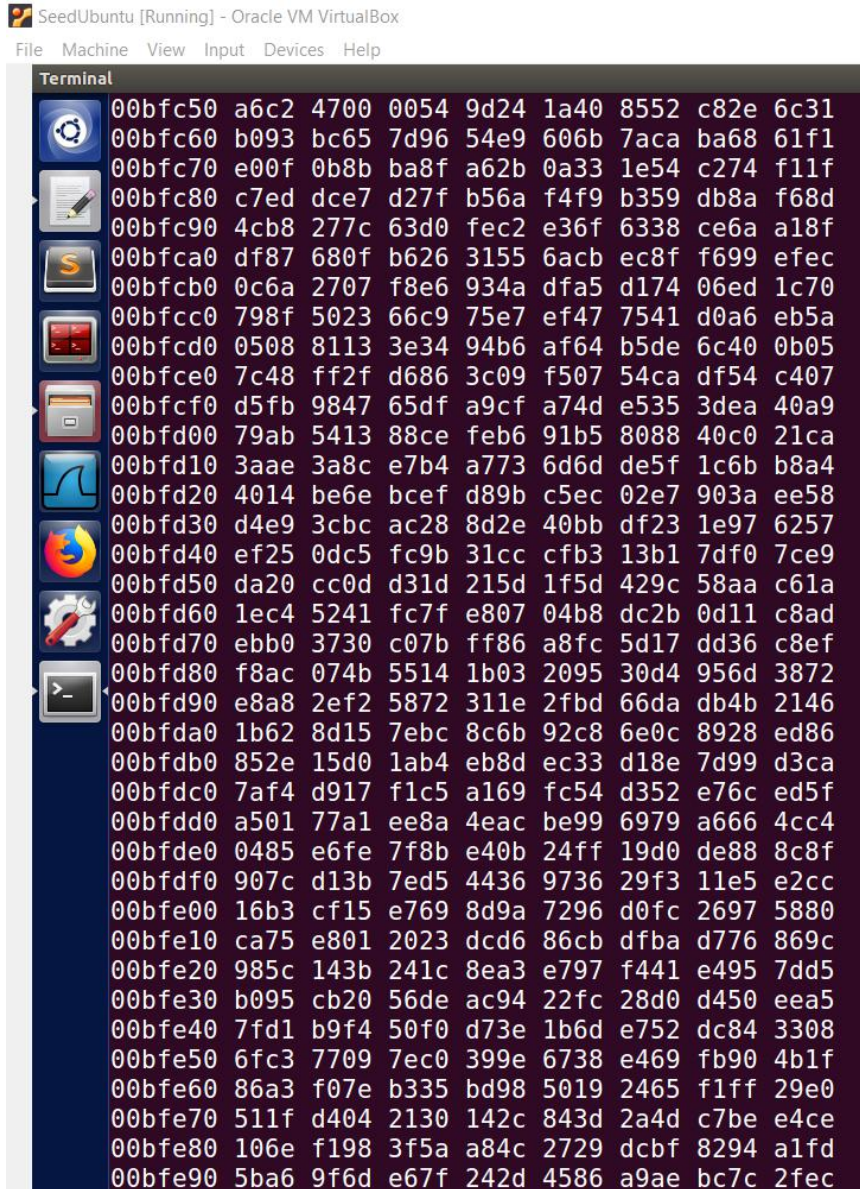


```
SeedUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[09/13/21]seed@ankith_j_rai PES1UG19CS069:~/.../cryptology$ watch -n .1 cat /proc/sys/kernel/random/entropy_avail
[09/13/21]seed@ankith_j_rai PES1UG19CS069:~/.../cryptology$ cat /dev/random | hexdump
00000000 f584 cfac 24f3 2d9c a248 84fb 3ec4 005a
00000010 a5a3 0d94 8e17 d624 f5eb 5311 157d 722d
00000020 e772 76d3 d005 eb20 fa8f bf4f 7942 9178
00000030 7a17 e31a 84ff 637c d0f8 05ab 454f 38a6
00000040 4b03 f23c d197 80f4 7bdd 39a8 7d6c 5098
00000050 ceal 43f5 f6d6 2c0b daa2 51c1 68d2 885e
00000060 97cd c81d 5c49 f23b 7ac5 7b47 d732 48e0
00000070 fc10 60d8 202c 5234 51f9 af7b 11c5 fa4d
00000080 790f 9036 18e1 6ba3 80ec fbee 415b d420
00000090 c641 548f 4c3b ceed 0322 52ea 860c 2b8c
000000a0 eb00 c68a d239 0b75 486a b20f 2a5e 6cdf
000000b0 eff4 41e7 8c7a f633 90e6 9dfd 7d97 4a52
000000c0 4b39 402c 9efe 212e da3c 76b8 a8c2 2afc
000000d0 b06c 43d0 6135 f8bb da0a 492b 52be f880
000000e0 5b56 262a 78a7 d2d4 64f8 dcb4 30ff afde
000000f0 9947 ce6a 4cd5 07c6 9999 08e6 aecb c312
00001000 c96e 5c4f d548 6141 f03d b345 8da1 5834
00001100 3be3 daaf 79a8 3fd5 995a 48b0 dddc 0e95
00001200 f127 9571 e200 4139 8195 9820 0a75 b5db
00001300 9e68 557a 368d 2231 a44b f38e 6797 4aaa
```

It is noted that if we do not move mouse or press any key it is noted that random numbers are not generated as entropy has become 0. It is also noted that if we move the mouse or press any key the entropy increases hence random numbers are generated.

Task 5: Get Pseudo Random Numbers from /dev/urandom

Step 1:



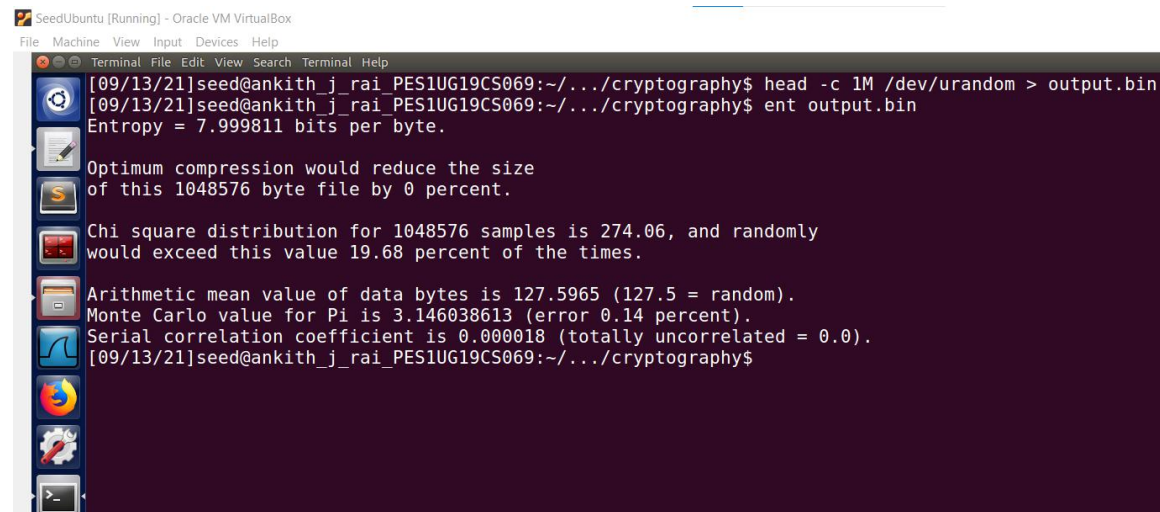
The screenshot shows a terminal window titled "SeedUbuntu [Running] - Oracle VM VirtualBox". The terminal displays a list of 40 hexadecimal strings, each 16 characters long, generated from the /dev/urandom device. The strings are listed in a single column, separated by newlines. The first string is 00bfc50 a6c2 4700 0054 9d24 1a40 8552 c82e 6c31 and the last is 00bfe90 5ba6 9f6d e67f 242d 4586 a9ae bc7c 2fec.

```
00bfc50 a6c2 4700 0054 9d24 1a40 8552 c82e 6c31
00bfc60 b093 bc65 7d96 54e9 606b 7aca ba68 61f1
00bfc70 e00f 0b8b ba8f a62b 0a33 1e54 c274 f11f
00bfc80 c7ed dce7 d27f b56a f4f9 b359 db8a f68d
00bfc90 4cb8 277c 63d0 fec2 e36f 6338 ce6a a18f
00bfca0 df87 680f b626 3155 6acb ec8f f699 efec
00bfcb0 0c6a 2707 f8e6 934a dfa5 d174 06ed 1c70
00bfcc0 798f 5023 66c9 75e7 ef47 7541 d0a6 eb5a
00bfcd0 0508 8113 3e34 94b6 af64 b5de 6c40 0b05
00bfce0 7c48 ff2f d686 3c09 f507 54ca df54 c407
00bfcf0 d5fb 9847 65df a9cf a74d e535 3dea 40a9
00bfd00 79ab 5413 88ce feb6 91b5 8088 40c0 21ca
00bfd10 3aae 3a8c e7b4 a773 6d6d de5f 1c6b b8a4
00bfd20 4014 be6e bcef d89b c5ec 02e7 903a ee58
00bfd30 d4e9 3cbc ac28 8d2e 40bb df23 1e97 6257
00bfd40 ef25 0dc5 fc9b 31cc cfb3 13b1 7df0 7ce9
00bfd50 da20 cc0d d31d 215d 1f5d 429c 58aa c61a
00bfd60 1ec4 5241 fc7f e807 04b8 dc2b 0d11 c8ad
00bfd70 ebb0 3730 c07b ff86 a8fc 5d17 dd36 c8ef
00bfd80 f8ac 074b 5514 1b03 2095 30d4 956d 3872
00bfd90 e8a8 2ef2 5872 311e 2fbd 66da db4b 2146
00bfda0 1b62 8d15 7ebc 8c6b 92c8 6e0c 8928 ed86
00bfdb0 852e 15d0 1ab4 eb8d ec33 d18e 7d99 d3ca
00bfdc0 7af4 d917 f1c5 a169 fc54 d352 e76c ed5f
00bfdd0 a501 77a1 ee8a 4eac be99 6979 a666 4cc4
00bfde0 0485 e6fe 7f8b e40b 24ff 19d0 de88 8c8f
00bdfd0 907c d13b 7ed5 4436 9736 29f3 11e5 e2cc
00bfe00 16b3 cf15 e769 8d9a 7296 d0fc 2697 5880
00bfe10 ca75 e801 2023 dcd6 86cb dfba d776 869c
00bfe20 985c 143b 241c 8ea3 e797 f441 e495 7dd5
00bfe30 b095 cb20 56de ac94 22fc 28d0 d450 eea5
00bfe40 7fd1 b9f4 50f0 d73e 1b6d e752 dc84 3308
00bfe50 6fc3 7709 7ec0 399e 6738 e469 fb90 4b1f
00bfe60 86a3 f07e b335 bd98 5019 2465 f1ff 29e0
00bfe70 511f d404 2130 142c 843d 2a4d c7be e4ce
00bfe80 106e f198 3f5a a84c 2729 dcbf 8294 a1fd
00bfe90 5ba6 9f6d e67f 242d 4586 a9ae bc7c 2fec
```

From the above screenshot we can notice that when we run /dev/urandom a lot of random numbers are generated.

Step 2:Measuring the quality of the random number

ent tool is used to measure the quality of the random number generated.



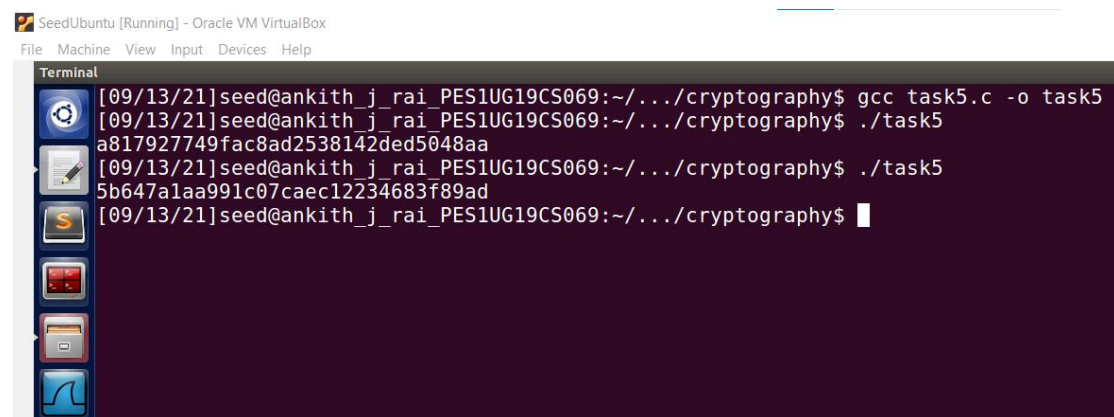
```
SeedUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal File Edit View Search Terminal Help
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ head -c 1M /dev/urandom > output.bin
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ ent output.bin
Entropy = 7.999811 bits per byte.

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

Chi square distribution for 1048576 samples is 274.06, and randomly
would exceed this value 19.68 percent of the times.

Arithmetic mean value of data bytes is 127.5965 (127.5 = random).
Monte Carlo value for Pi is 3.146038613 (error 0.14 percent).
Serial correlation coefficient is 0.000018 (totally uncorrelated = 0.0).
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$
```

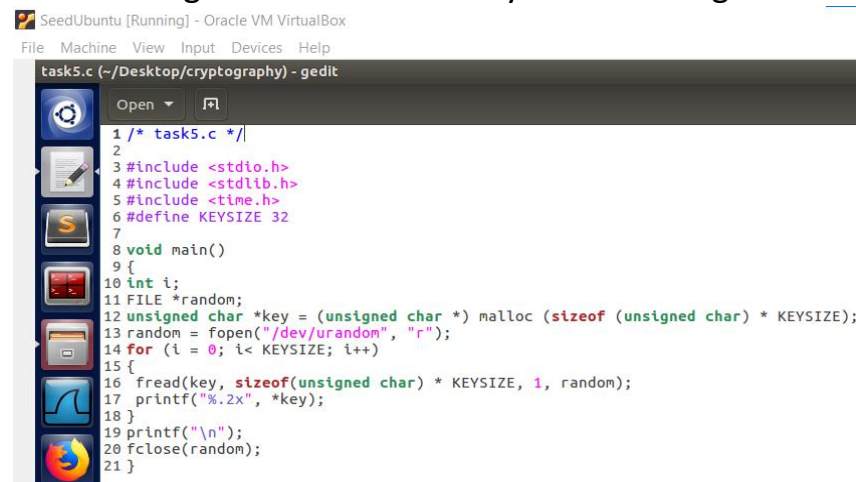
Step 3:



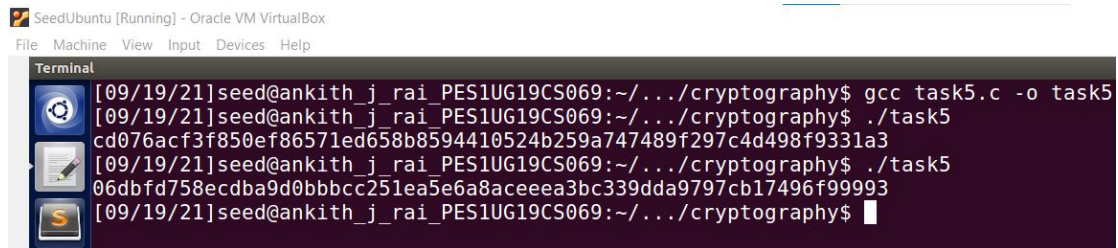
```
SeedUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ gcc task5.c -o task5
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ ./task5
a817927749fac8ad2538142ded5048aa
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ ./task5
5b647a1aa991c07caec12234683f89ad
[09/13/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$
```

From the above screenshot it can be seen that a 128 bit key is generated.

In order to generate a 256 bit key the following modification is done:



```
SeedUbuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
task5.c (-/Desktop/cryptography) - gedit
1 /* task5.c */
2
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <time.h>
6 #define KEYSIZE 32
7
8 void main()
9 {
10 int i;
11 FILE *random;
12 unsigned char *key = (unsigned char *) malloc (sizeof (unsigned char) * KEYSIZE);
13 random = fopen("/dev/urandom", "r");
14 for (i = 0; i < KEYSIZE; i++)
15 {
16 fread(key, sizeof(unsigned char) * KEYSIZE, 1, random);
17 printf("%.2x", *key);
18 }
19 printf("\n");
20 fclose(random);
21 }
```

```
[09/19/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ gcc task5.c -o task5
[09/19/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ ./task5
cd076acf3f850ef86571ed658b8594410524b259a747489f297c4d498f9331a3
[09/19/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$ ./task5
06dbfd758ecdba9d0bbbcc251ea5e6a8aceeea3bc339dda9797cb17496f99993
[09/19/21]seed@ankith_j_rai_PES1UG19CS069:~/../cryptography$
```

From the above screenshot we can be seen that a 256 bit key is generated.