Jon Franck

4/15/2021

CSC 4413

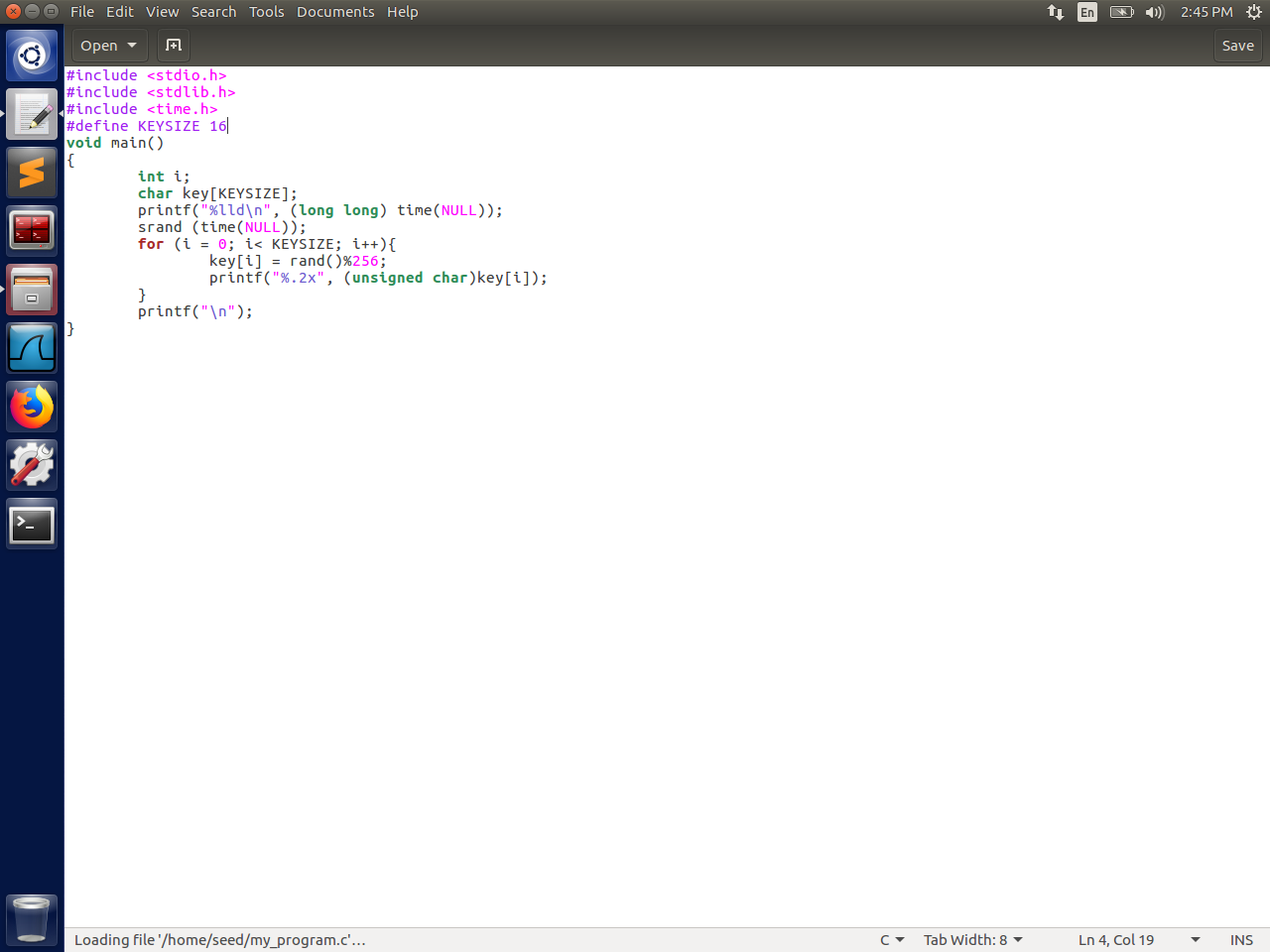
# Pseudo Random Number Generation Lab

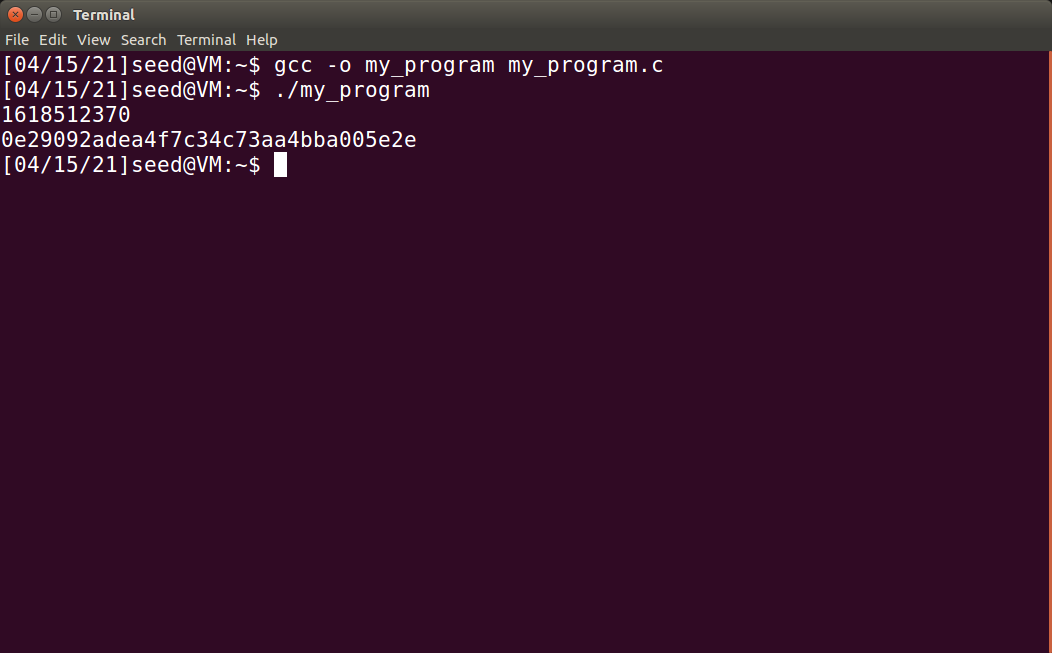
**Objective** In this lab, students will learn why the typical random number generation method is not appropriate for generating secrets, such as encryption keys. They will further learn a standard way to generate pseudo random numbers that are good for security purposes. This lab covers the following topics:

• Pseudo random number generation  
• Mistakes in random number generation  
• Generating encryption key  
• The /dev/random and /dev/urandom device files

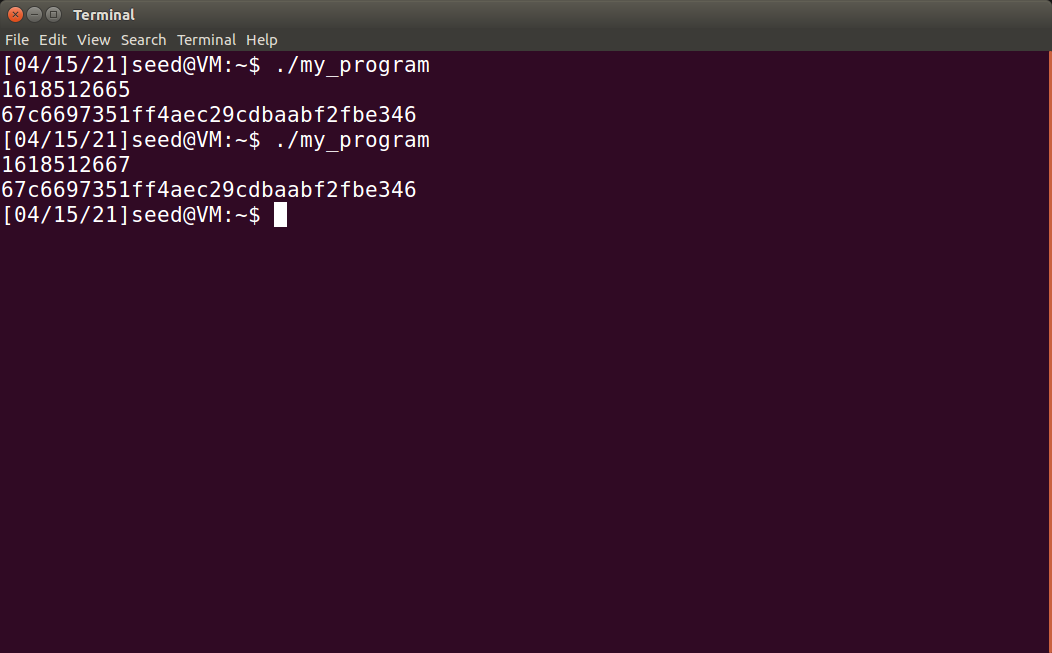
## Task 1: Generate Encryption Key in a Wrong Way

Run the following program to generate an Encryption Key:





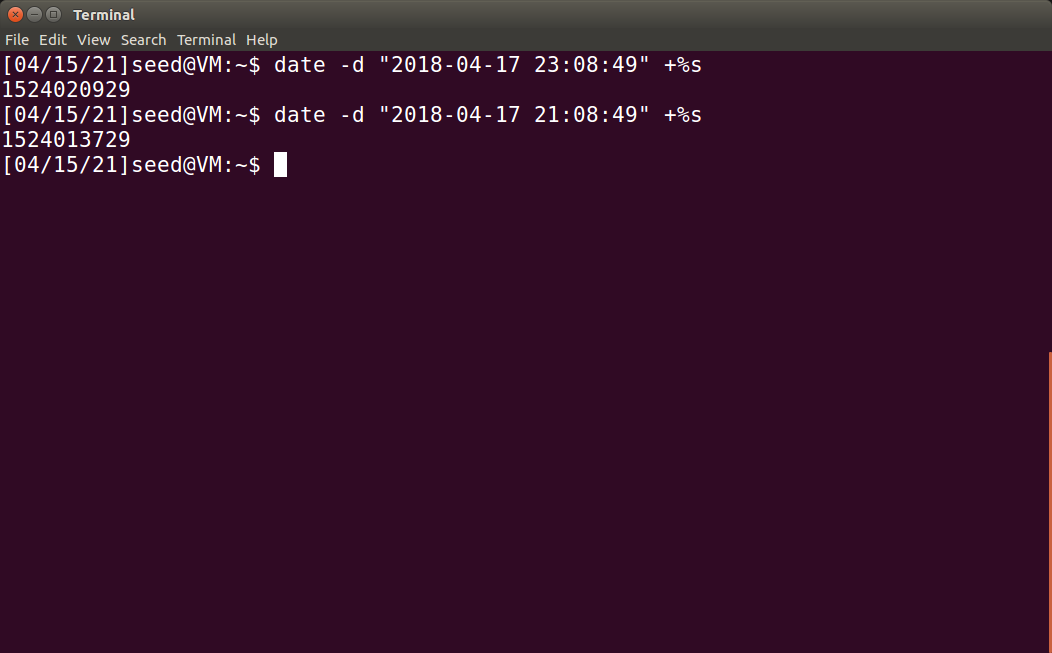
Now Delete the line “srand (time(NULL));” from the code and re-run the program:



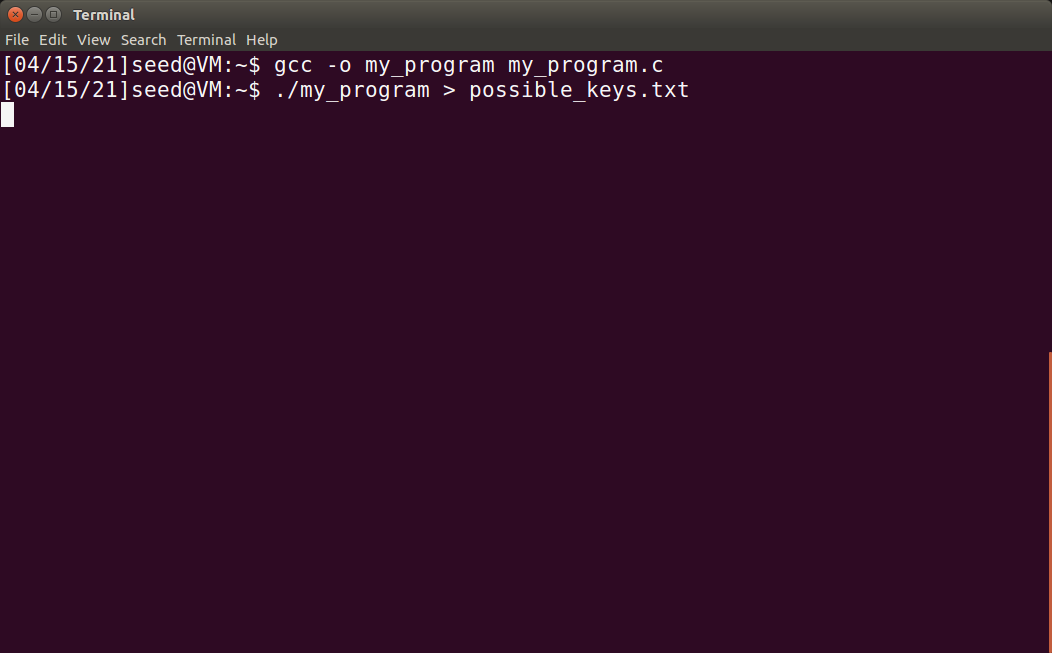
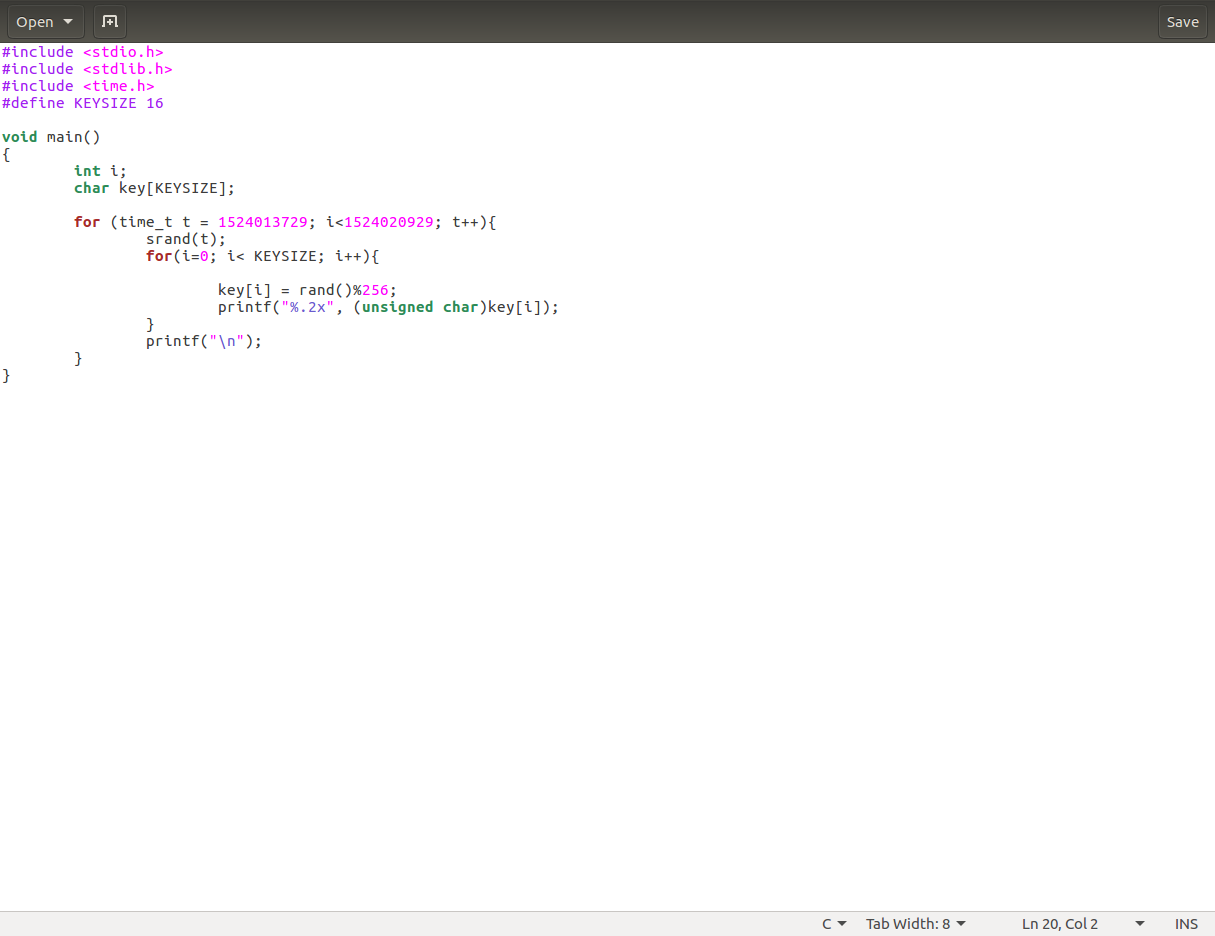
**Observation:** After removing the “Srand()” function, although the timestamp changes, running the program multiple times yields the same Key. From this we can infer that the function seeds the random number that is generated with the “rand()” function, with the current time.

## Task 2: Guessing the Key

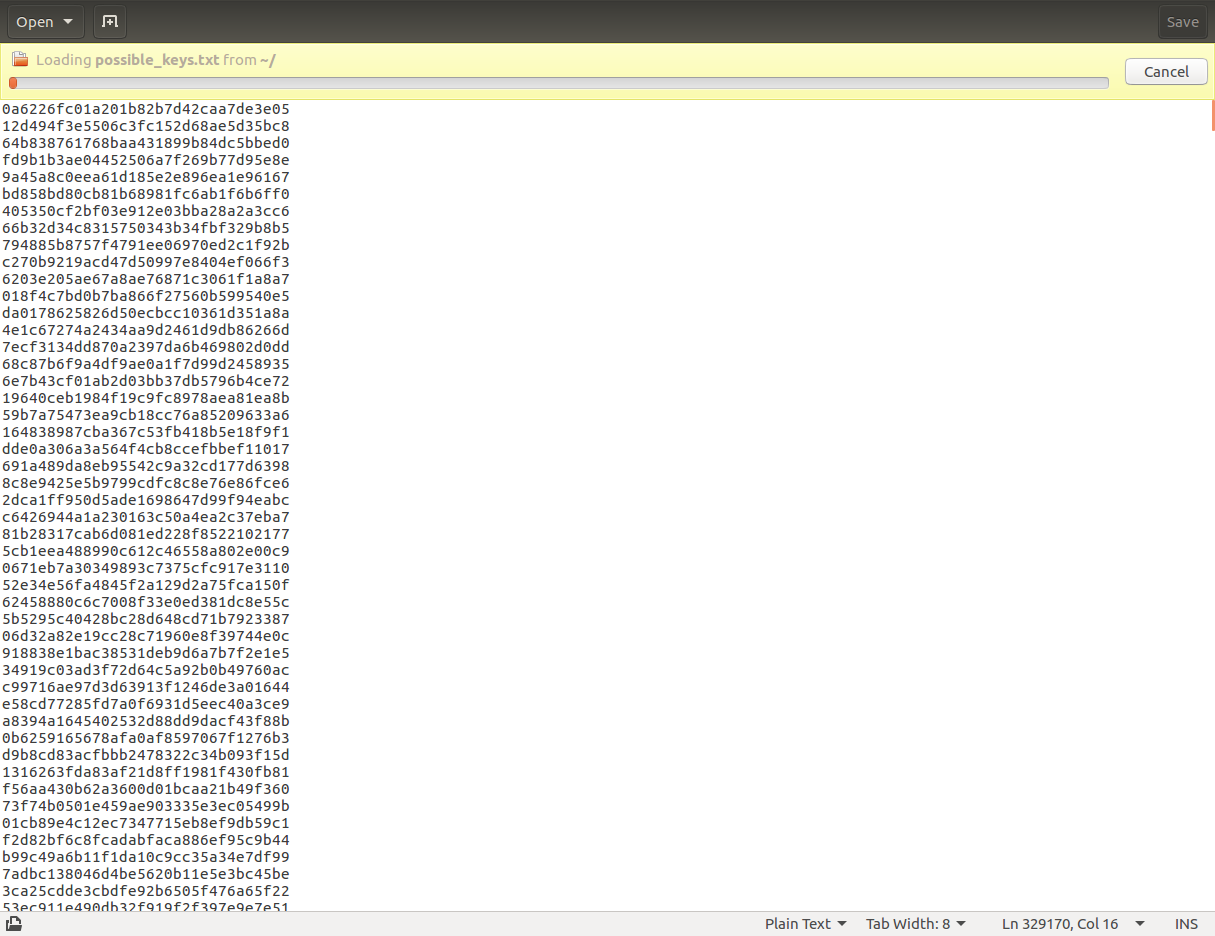
Get the Epoch:



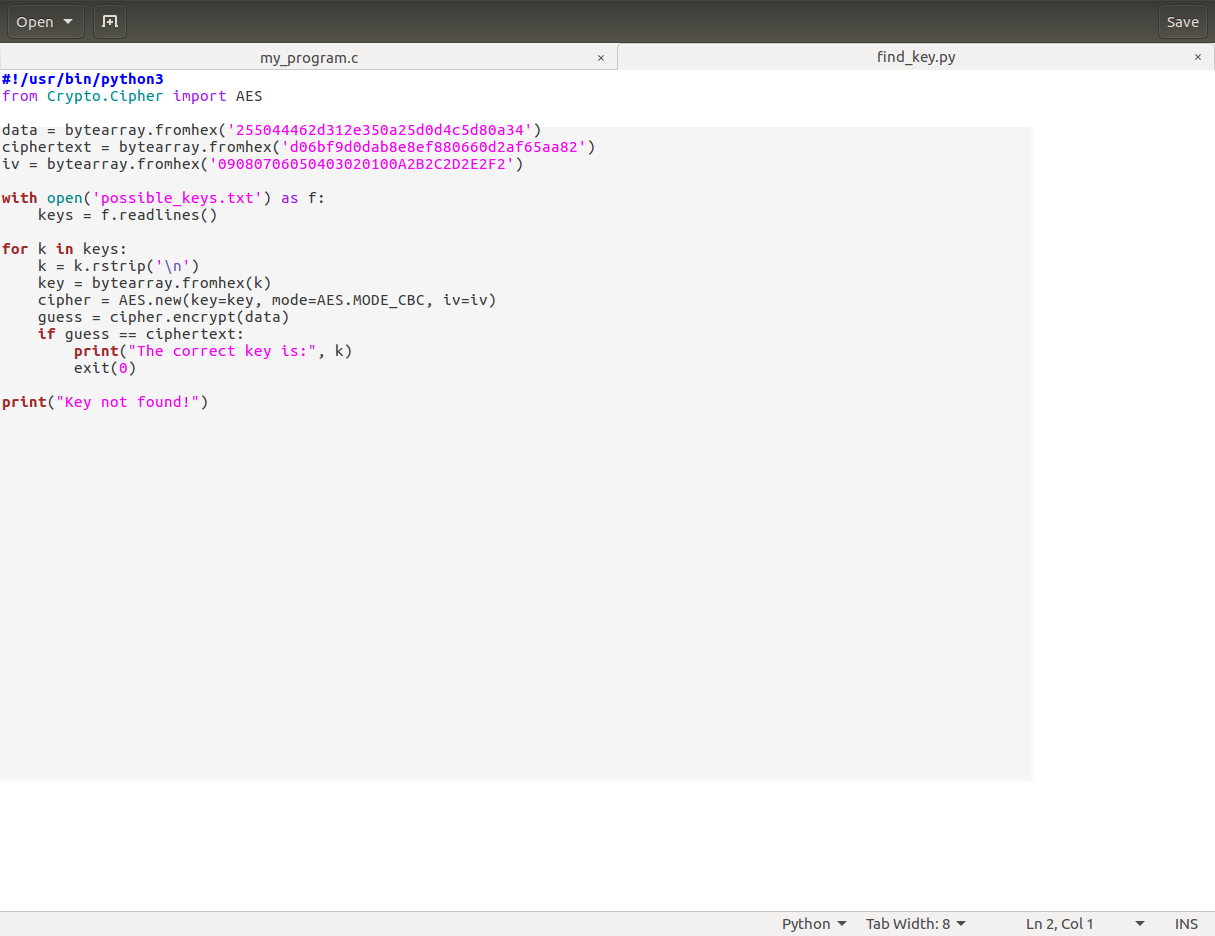
(Since Bob guessed that the key could have been generated up to two hours before the file was created, we must test for every time in the window between the two timestamps above).

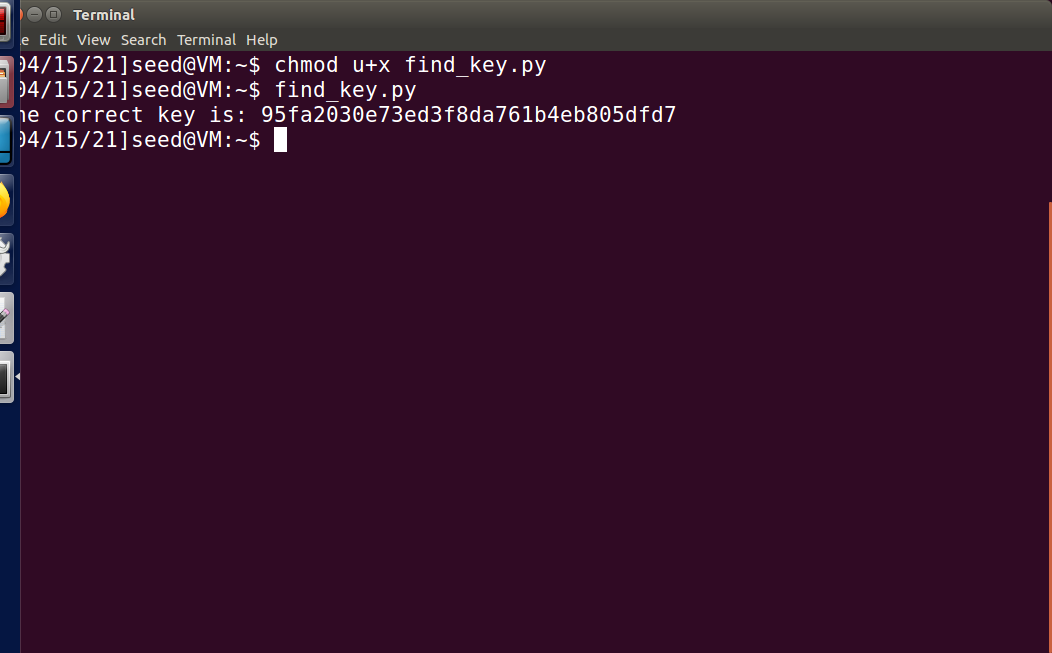


Here we changed the original program to generate every possible key in our time frame and ran the program to export all the keys to a .txt file.



Now we must create a program to use brute force to try to match the encryption with one of these “possible keys”. I used the following python code to do so:



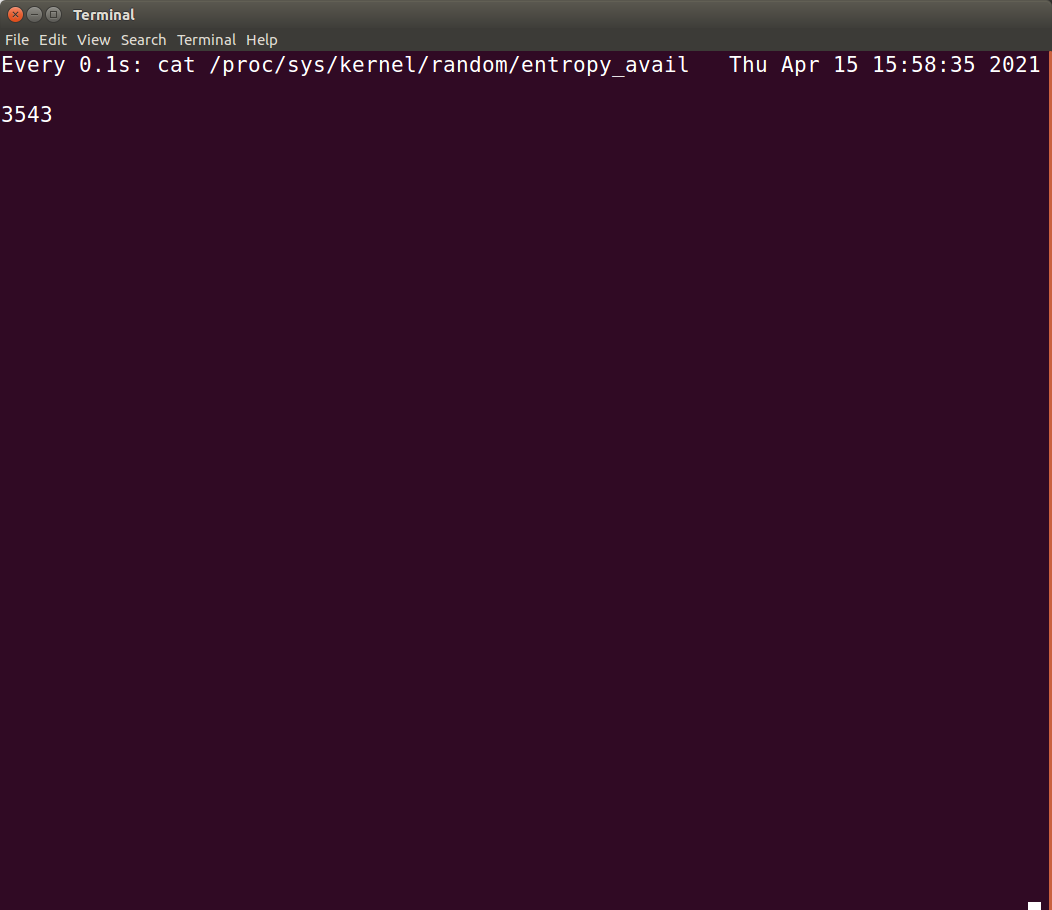


It was a success!

## Task 3: Measure the Entropy of Kernel

Use the following command to measure the system’s entropy:

watch -n .1 cat /proc/sys/kernel/random/entropy\_avail

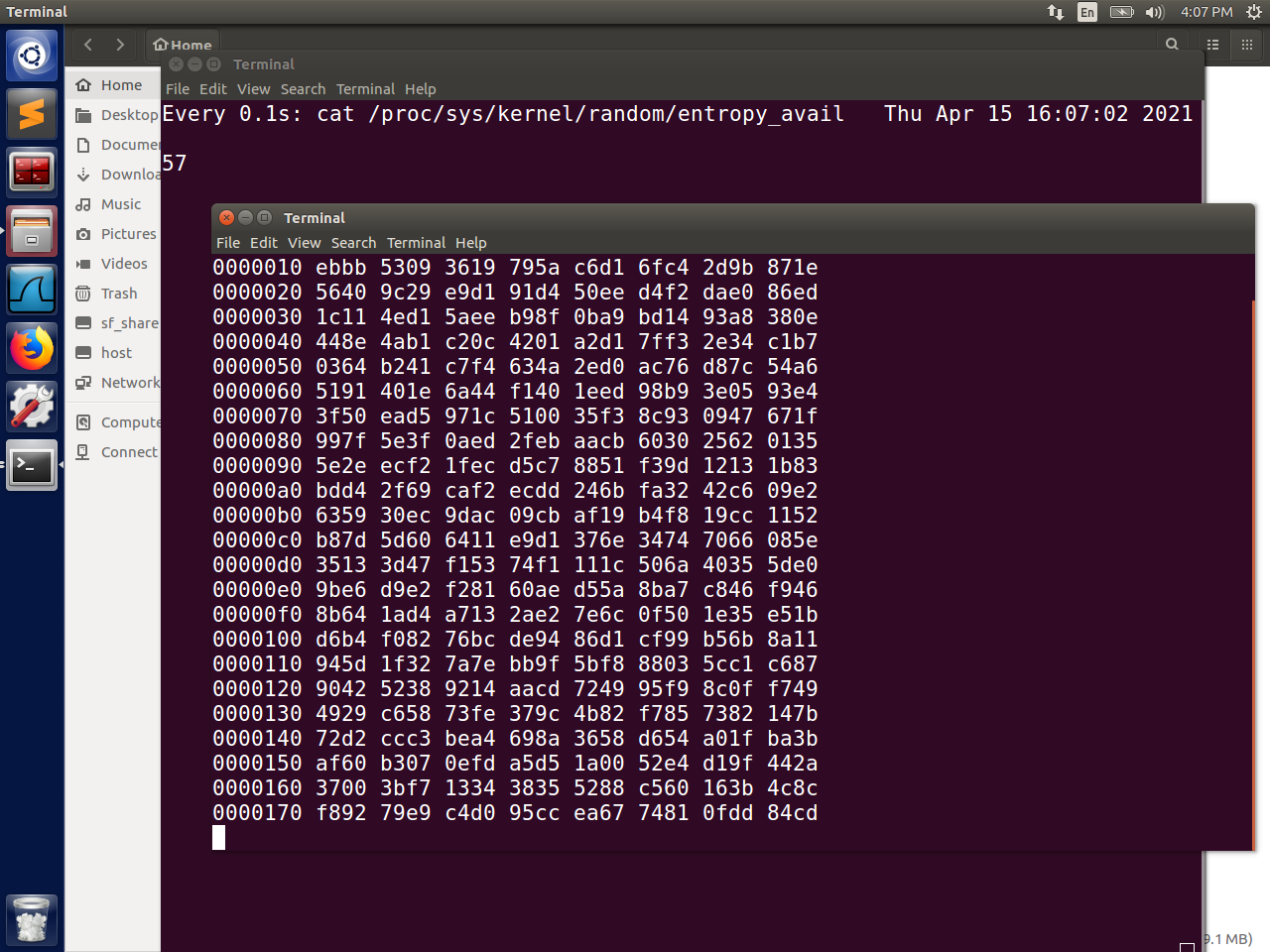


**Observation:** When I do nothing, it takes about 10 seconds for the entropy to increase but when I type on the keyboard or move the mouse the number increases much faster.

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

Use the following command to generate random numbers:

cat /dev/random | hexdump



**Observation:** Again, moving the mouse/typing increases the entropy faster. Once the entropy level gets to about 60, it resets, and a new line of random characters is printed out by the “hexdump” command.

**Question:** If a server uses /dev/random to generate the random session key with a client. Please

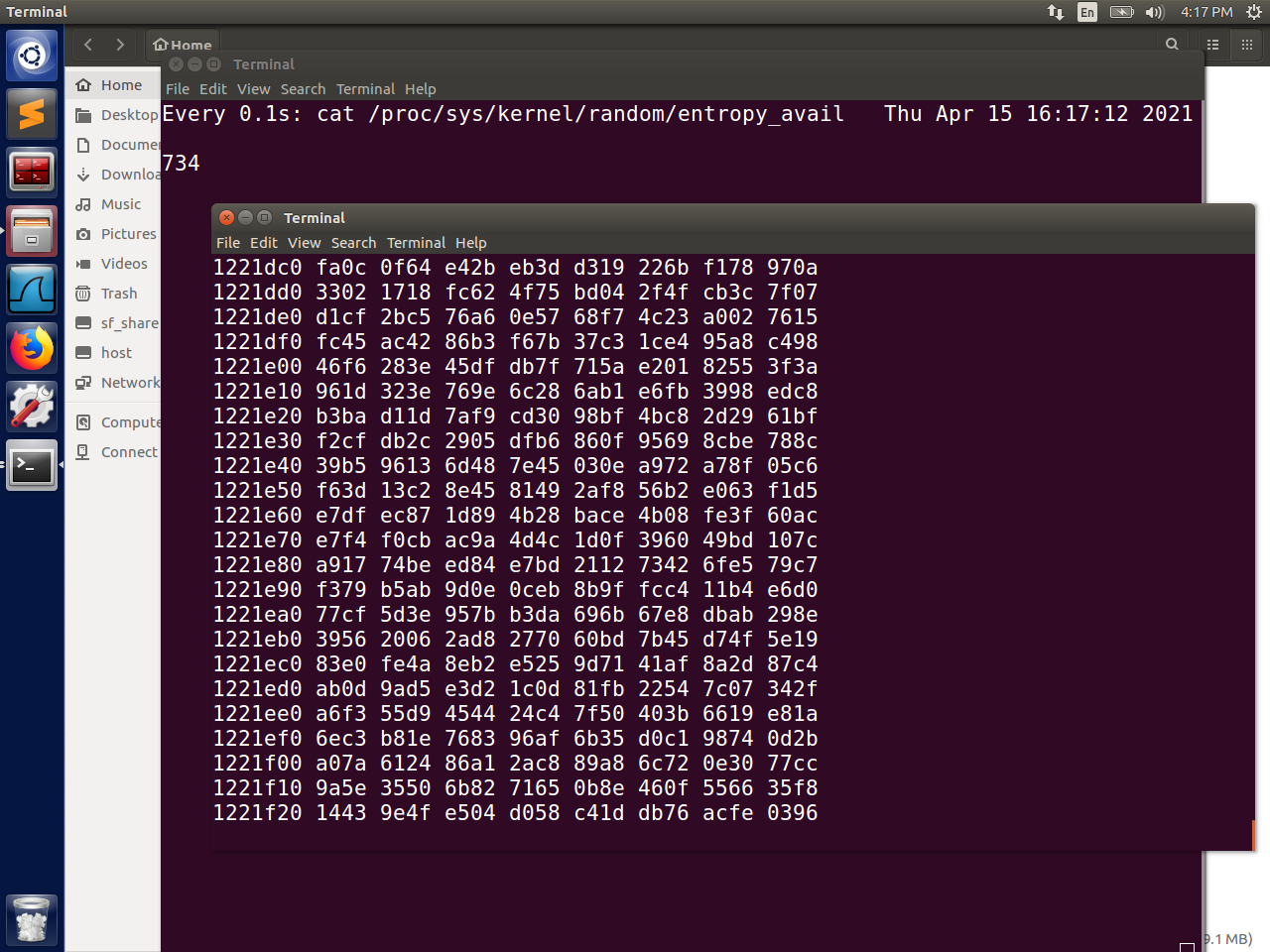
describe how you can launch a Denial-Of-Service (DOS) attack on such a server.

If the attacker continuously makes requests that use up entropy, then the server will not have enough entropy for the /dev/random to generate a key and the server will have to deny service to the client.

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

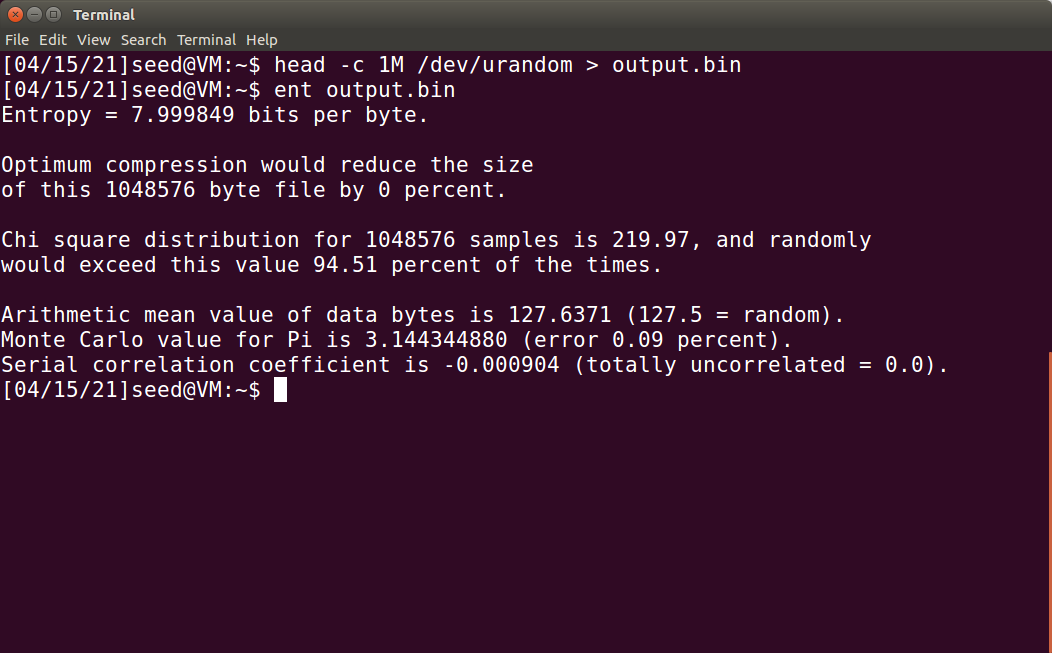
Use the following command to generate random numbers:

cat /dev/urandom | hexdump



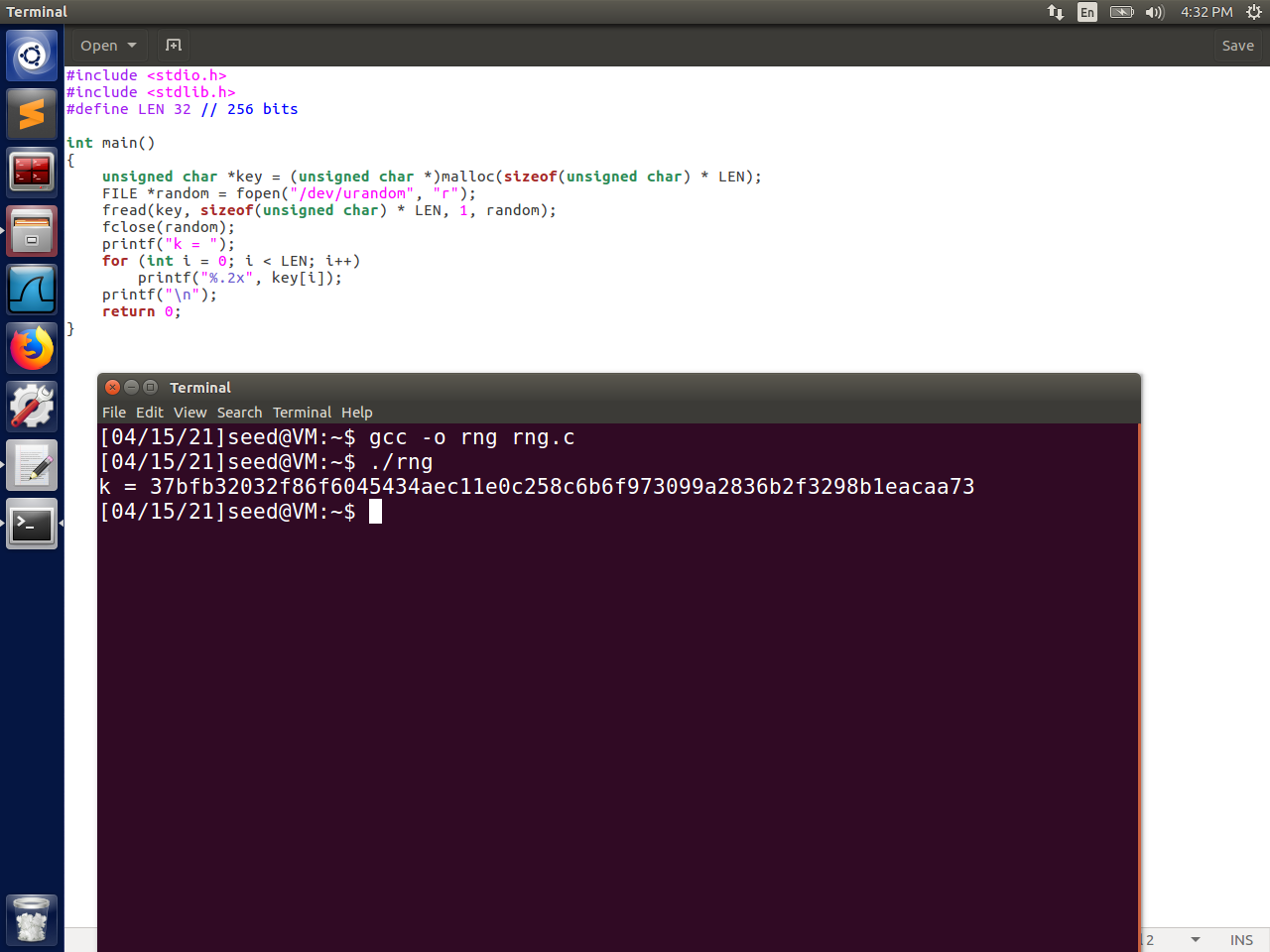
**Observation:** This time, the entropy is steadily increasing and not being reset when new random characters are generated. The mouse movement still speeds up the entropy increase, but there is no longer a cut-off point, where it resets.

Now, let us measure the quality of the randomly generated number:



Based on these results, these numbers can be considered quite random

Modify the program to generate 256-bit random numbers:



In order to go from 128 to 256 bits, we must change the length (LEN) from 16 to 32, then just adapt the original program we used in this lab to complete and print out the random number.