|  |
| --- |
| **CPS633 Section 07 Fall2021** |
| Lab 05 Report |
| **Pseudo Random Number Generation Lab**  **Name: Tusaif Azmat (group leader)**  **Student#: 500660278.**  **And**  **Name: Ankit Sodhi**  **Student#: 500958004**  **Group 04.** |

**CPS 633 - Lab 5 Report**

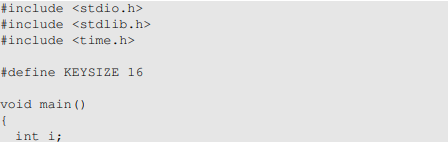
**Pseudo Random Number Generation Lab**

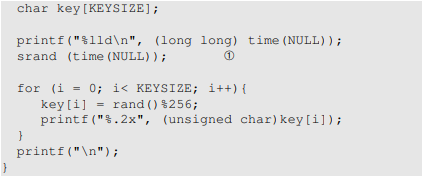
**2 Lab Tasks:**

**2.1 Task 1: Generate Encryption Key in a Wrong Way**

compile time\_random.c and run



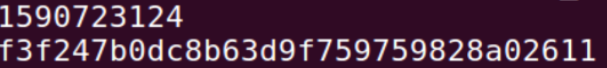












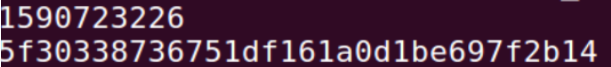








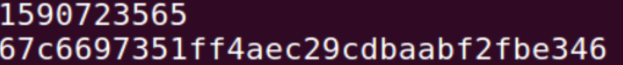




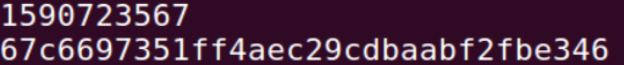
As you can see above, it always gives different results: Because it uses the current time as a random seed to generate random numbers, which guarantee the seed is always different in each run.

Note: When comment out the line srand(time(NULL));, recompile it and run, the numbers generated are the same now:

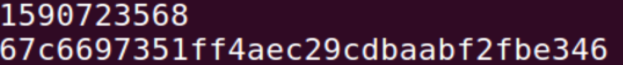




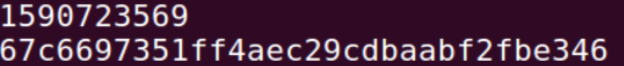




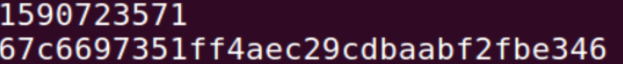












**2.2 Task 2: Guessing the Key**

Get the epoch of 2018-04-17 23:08:49 by:



It returns **1524020929.**

Then we list all possible random numbers generated by time\_random.c **within the two hours** by adding a loop before line 12 in it as time\_guess.c:



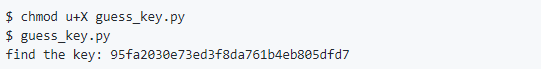
get the list:



Use a brute-force method to crack the key from key\_dict.txt as guess\_key.py:



It finds out the key:



**2.3 Task 3: Measure the Entropy of Kernel**

Let us monitor the change of the entropy by running the above command via watch, which executes a program periodically, showing the output in full screen. The following command runs the cat program every 0.1 second.



Note: When I move the mouse or type something, the value increases fast. Every time it decreases, one line of new random numbers appears. So we can say that /dev/random consumes the available entropy produced by user's behaviors to generate new random numbers.

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

****

It generates output slowly and almost gets stuck when I keep unmoved. Monitor the entropy in another terminal：

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.

Attackers keep asking for establishing connections, which makes the server run out of the available entropy for /dev/random. Then the random number generator is blocked.

Note: When I move the mouse or type something, the value increases fast. Every time it decreases, one line of new random numbers appears. So we can say that /dev/random consumes the available entropy produced by user's behaviors to generate new random numbers.

**2.5 Task 5: Get Random Numbers from /dev/urandom**

****

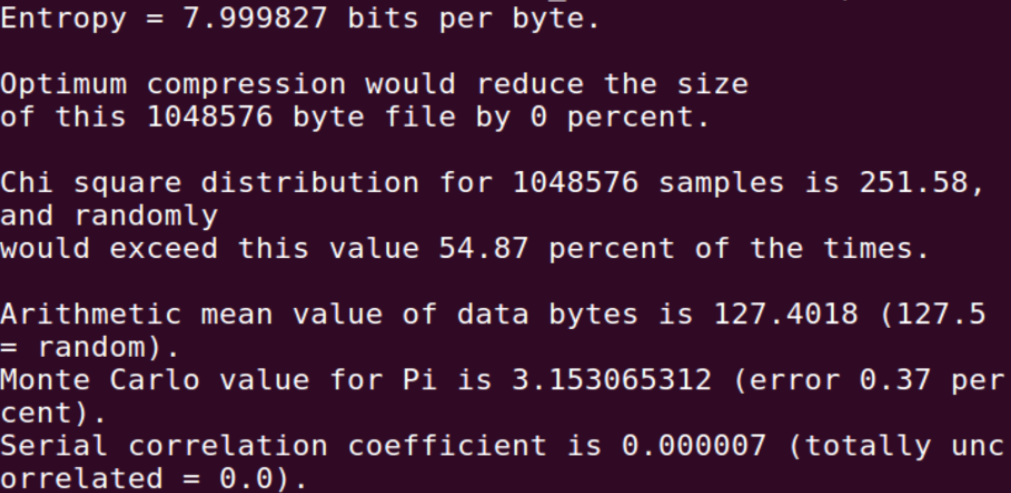
It keeps printing out random numbers. We truncate the first 1 MB outputs into a file named output.bin:

****

Then use ent to evaluate its information density:

****

****

****

See the documentation. It looks random in most measures.

Use **/dev/urandom** to generate a 256-bit random number as a session key by [read\_random\_key.c](https://github.com/li-xin-yi/seedlab/blob/master/Pseudo-Random-Number-Generation/read_random_key.c):

****

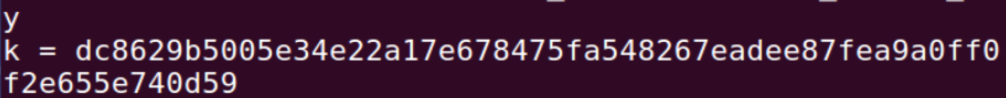
Compile:

****

Run:

** **

****

****

**Key= dc8629b5005e34e34e22a17e678475fa548267eadee87fea9a0ff0fe655e740d59**