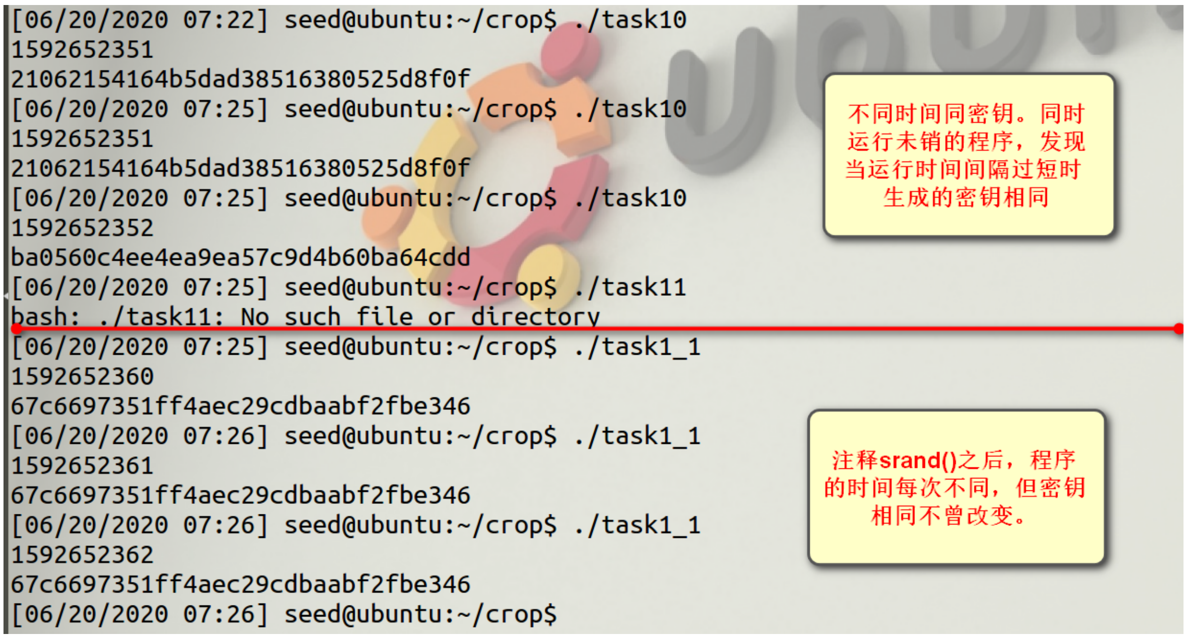
2020 Spring Random Number Generation

Task 1: Generate Encryption Key in a Wrong Way



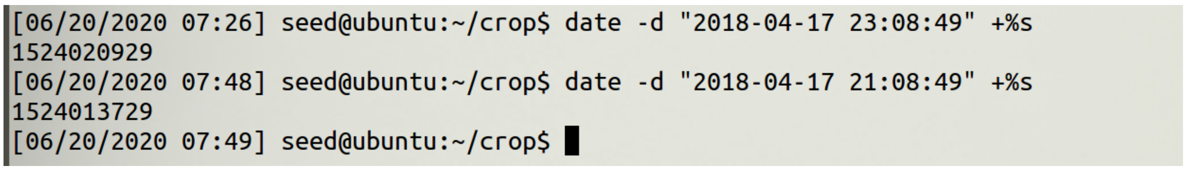
```
[06/20/2020 07:22] seed@ubuntu:~/crop$ ./task10
1592652351
21062154164b5dad38516380525d8f0f
[06/20/2020 07:25] seed@ubuntu:~/crop$ ./task10
1592652351
21062154164b5dad38516380525d8f0f
[06/20/2020 07:25] seed@ubuntu:~/crop$ ./task10
1592652352
ba0560c4ee4ea9ea57c9d4b60ba64cdd
[06/20/2020 07:25] seed@ubuntu:~/crop$ ./task11
bash: ./task11: No such file or directory
[06/20/2020 07:25] seed@ubuntu:~/crop$ ./task1_1
1592652360
67c6697351ff4aec29cdbaabf2fbe346
[06/20/2020 07:26] seed@ubuntu:~/crop$ ./task1_1
1592652361
67c6697351ff4aec29cdbaabf2fbe346
[06/20/2020 07:26] seed@ubuntu:~/crop$ ./task1_1
1592652362
67c6697351ff4aec29cdbaabf2fbe346
[06/20/2020 07:26] seed@ubuntu:~/crop$
```

不同时间同密钥。同时运行未销的程序，发现当运行时间间隔过短时生成的密钥相同

注释srand()之后，程序的时间每次不同，但密钥相同不曾改变。

Task 2: Guessing the Key

将系统时区调成纽约时间 刚好和上海时间差12个小时，继续后面的实验



```
[06/20/2020 07:26] seed@ubuntu:~/crop$ date -d "2018-04-17 23:08:49" +%s
1524020929
[06/20/2020 07:48] seed@ubuntu:~/crop$ date -d "2018-04-17 21:08:49" +%s
1524013729
[06/20/2020 07:49] seed@ubuntu:~/crop$ 
```

先将所有密钥打印出。

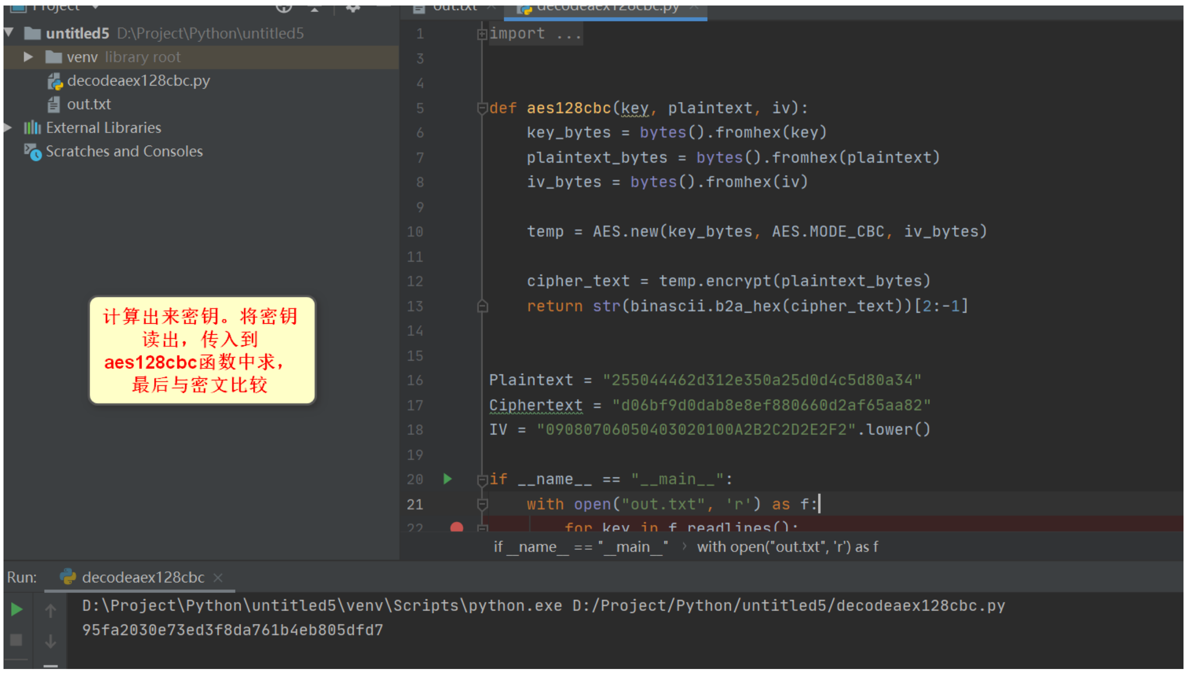
```c
//密钥生成程序
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define KEYSIZE 16

void main()
{
    unsigned char key[KEYSIZE];
    unsigned int num;
    int i;
    for(num=1524013729; num<=1524020929; num++){
        srand (num);
        for (i = 0; i< KEYSIZE; i++){
            key[i] = rand()%256;
            printf("%.2x", (unsigned char)key[i]);
        }
        printf("\n");
    }

}
```

再利用重定向`> out.txt`保存文件到out.txt中。

利用python编写aes 128位 cbc加密程序。

```python
import binascii
from Crypto.Cipher import AES


def aes128cbc(key, plaintext, iv):
    key_bytes = bytes().fromhex(key)
    plaintext_bytes = bytes().fromhex(plaintext)
    iv_bytes = bytes().fromhex(iv)

    temp = AES.new(key_bytes, AES.MODE_CBC, iv_bytes)

    cipher_text = temp.encrypt(plaintext_bytes)
    return str(binascii.b2a_hex(cipher_text))[2:-1]


Plaintext = "255044462d312e350a25d0d4c5d80a34"
Ciphertext = "d06bf9d0dab8e8ef880660d2af65aa82"
IV = "09080706050403020100A2B2C2D2E2F2".lower()

if __name__ == "__main__":
    with open("out.txt", 'r') as f:
        for key in f.readlines():
            c = aes128cbc(key.strip(), Plaintext, IV)
            if c == Ciphertext:
                print(key)
                break
```

最后计算出结果。

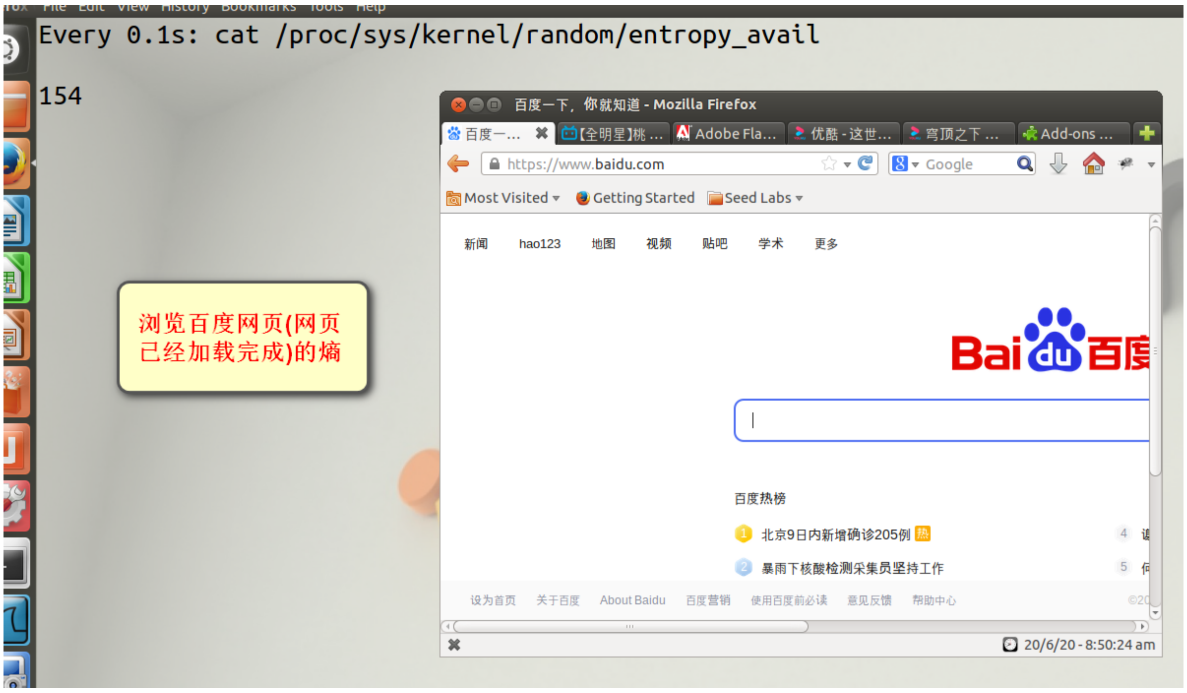


untitled5 D:\Project\Python\untitled5
venv library root
decodeaex128cbc.py
out.txt
External Libraries
Scratches and Consoles
计算出来密钥。将密钥读出，传入到aes128cbc函数中求，最后与密文比较
if __name__ == "__main__" › with open("out.txt", 'r') as f
Run: decodeaex128cbc

```python
import ...


def aes128cbc(key, plaintext, iv):
    key_bytes = bytes().fromhex(key)
    plaintext_bytes = bytes().fromhex(plaintext)
    iv_bytes = bytes().fromhex(iv)

    temp = AES.new(key_bytes, AES.MODE_CBC, iv_bytes)

    cipher_text = temp.encrypt(plaintext_bytes)
    return str(binascii.b2a_hex(cipher_text))[2:-1]


Plaintext = "255044462d312e350a25d0d4c5d80a34"
Ciphertext = "d06bf9d0dab8e8ef880660d2af65aa82"
IV = "09080706050403020100A2B2C2D2E2F2".lower()

if __name__ == "__main__":
    with open("out.txt", 'r') as f:
        for key in f.readlines():
```

```
D:\Project\Python\untitled5\venv\Scripts\python.exe D:/Project/Python/untitled5/decodeaex128cbc.py
95fa2030e73ed3f8da761b4eb805dfd7
```

Task 3: Measure the Entropy of Kernel

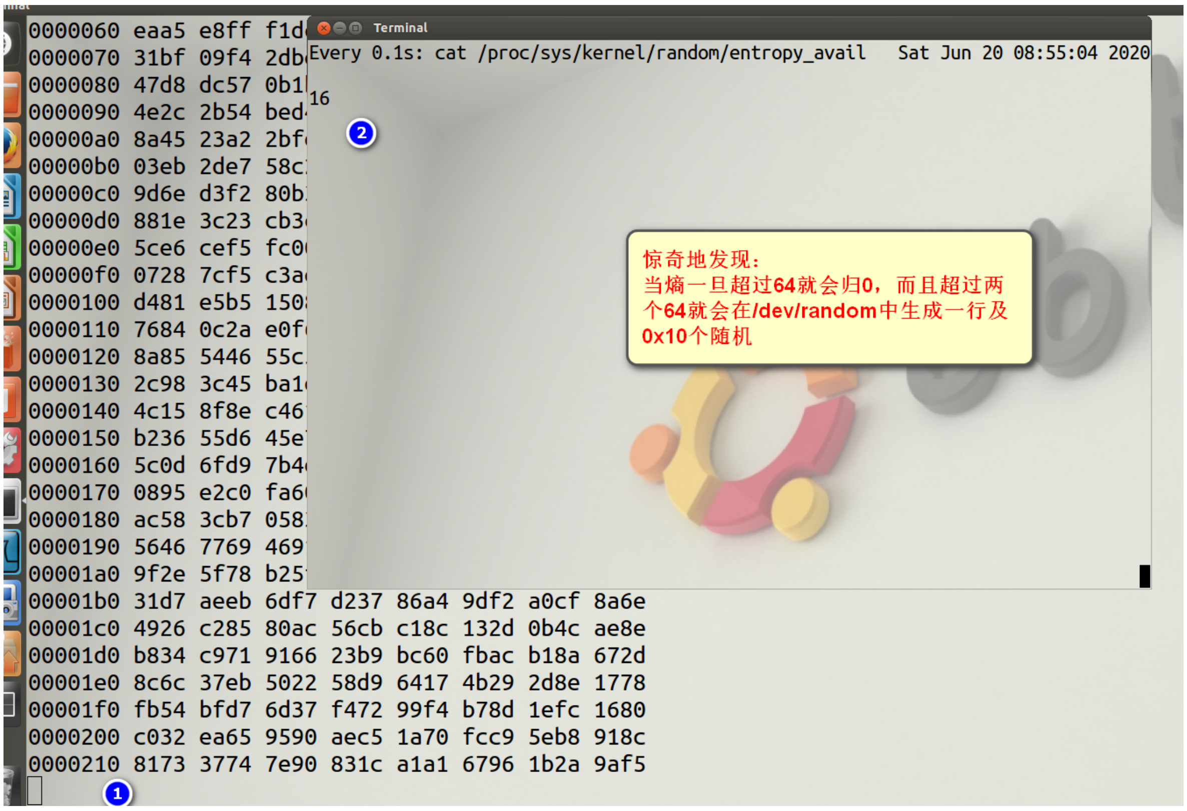


```
Every 0.1s: cat /proc/sys/kernel/random/entropy_avail

154
```

浏览百度网页(网页已经加载完成)的熵
百度一下，你就知道 - Mozilla Firefox
https://www.baidu.com
Most Visited
Getting Started
Seed Labs
新闻 hao123 地图 视频 贴吧 学术 更多
百度热榜
北京9日内新增确诊205例
暴雨下核酸检测采集员坚持工作
20/6/20 - 8:50:24 am

一旦出现动作，包括鼠标移动，浏览器页面加载等等熵都会增加。

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



```
0000060 eaa5 e8ff f1d
0000070 31bf 09f4 2db
0000080 47d8 dc57 0b1
0000090 4e2c 2b54 bed
00000a0 8a45 23a2 2bf
00000b0 03eb 2de7 58c
00000c0 9d6e d3f2 80b
00000d0 881e 3c23 cb3
00000e0 5ce6 cef5 fc0
00000f0 0728 7cf5 c3a
0000100 d481 e5b5 150
0000110 7684 0c2a e0f
0000120 8a85 5446 55c
0000130 2c98 3c45 ba1
0000140 4c15 8f8e c46
0000150 b236 55d6 45e
0000160 5c0d 6fd9 7b4
0000170 0895 e2c0 fa6
0000180 ac58 3cb7 058
0000190 5646 7769 469
00001a0 9f2e 5f78 b25
00001b0 31d7 aeeb 6df7 d237 86a4 9df2 a0cf 8a6e
00001c0 4926 c285 80ac 56cb c18c 132d 0b4c ae8e
00001d0 b834 c971 9166 23b9 bc60 fbac b18a 672d
00001e0 8c6c 37eb 5022 58d9 6417 4b29 2d8e 1778
00001f0 fb54 bfd7 6d37 f472 99f4 b78d 1efc 1680
0000200 c032 ea65 9590 aec5 1a70 fcc9 5eb8 918c
0000210 8173 3774 7e90 831c a1a1 6796 1b2a 9af5
```

```
Every 0.1s: cat /proc/sys/kernel/random/entropy_avail     Sat Jun 20 08:55:04 2020

16
```

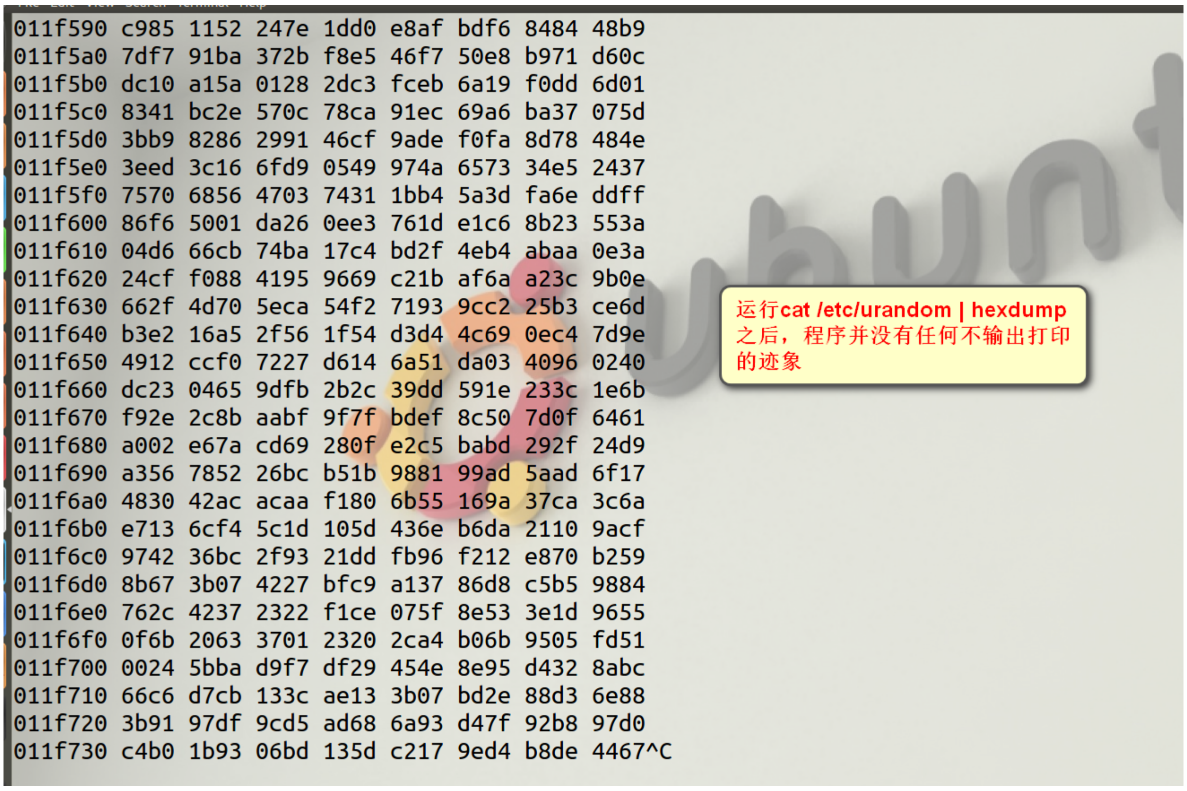
Terminal
1
2
惊奇地发现：
当熵一旦超过64就会归0，而且超过两个64就会在/dev/random中生成一行及0x10个随机

攻击场景：某一个服务使用/dev/random中的随机数进行加密，进行dos攻击。

攻击手段：对该服务器产生大量的恶意链接，但是不进一步的产生任何互动。

Task 5: Get Random Numbers from /dev/urandom

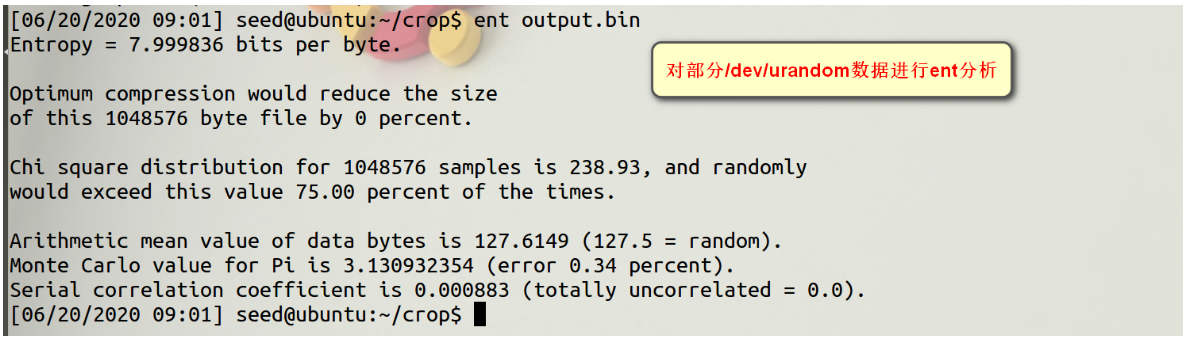
打印/dev/urandom并分析



```
011f590 c985 1152 247e 1dd0 e8af bdf6 8484 48b9
011f5a0 7df7 91ba 372b f8e5 46f7 50e8 b971 d60c
011f5b0 dc10 a15a 0128 2dc3 fceb 6a19 f0dd 6d01
011f5c0 8341 bc2e 570c 78ca 91ec 69a6 ba37 075d
011f5d0 3bb9 8286 2991 46cf 9ade f0fa 8d78 484e
011f5e0 3eed 3c16 6fd9 0549 974a 6573 34e5 2437
011f5f0 7570 6856 4703 7431 1bb4 5a3d fa6e ddff
011f600 86f6 5001 da26 0ee3 761d e1c6 8b23 553a
011f610 04d6 66cb 74ba 17c4 bd2f 4eb4 abaa 0e3a
011f620 24cf f088 4195 9669 c21b af6a a23c 9b0e
011f630 662f 4d70 5eca 54f2 7193 9cc2 25b3 ce6d
011f640 b3e2 16a5 2f56 1f54 d3d4 4c69 0ec4 7d9e
011f650 4912 ccf0 7227 d614 6a51 da03 4096 0240
011f660 dc23 0465 9dfb 2b2c 39dd 591e 233c 1e6b
011f670 f92e 2c8b aabf 9f7f bdef 8c50 7d0f 6461
011f680 a002 e67a cd69 280f e2c5 babd 292f 24d9
011f690 a356 7852 26bc b51b 9881 99ad 5aad 6f17
011f6a0 4830 42ac acaa f180 6b55 169a 37ca 3c6a
011f6b0 e713 6cf4 5c1d 105d 436e b6da 2110 9acf
011f6c0 9742 36bc 2f93 21dd fb96 f212 e870 b259
011f6d0 8b67 3b07 4227 bfc9 a137 86d8 c5b5 9884
011f6e0 762c 4237 2322 f1ce 075f 8e53 3e1d 9655
011f6f0 0f6b 2063 3701 2320 2ca4 b06b 9505 fd51
011f700 0024 5bba d9f7 df29 454e 8e95 d432 8abc
011f710 66c6 d7cb 133c ae13 3b07 bd2e 88d3 6e88
011f720 3b91 97df 9cd5 ad68 6a93 d47f 92b8 97d0
011f730 c4b0 1b93 06bd 135d c217 9ed4 b8de 4467^C
```

运行cat /etc/urandom | hexdump 之后，程序并没有任何不输出打印的迹象

用ent分析



```
[06/20/2020 09:01] seed@ubuntu:~/crop$ ent output.bin
Entropy = 7.999836 bits per byte.

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

Chi square distribution for 1048576 samples is 238.93, and randomly
would exceed this value 75.00 percent of the times.

Arithmetic mean value of data bytes is 127.6149 (127.5 = random).
Monte Carlo value for Pi is 3.130932354 (error 0.34 percent).
Serial correlation coefficient is 0.000883 (totally uncorrelated = 0.0).
[06/20/2020 09:01] seed@ubuntu:~/crop$
```

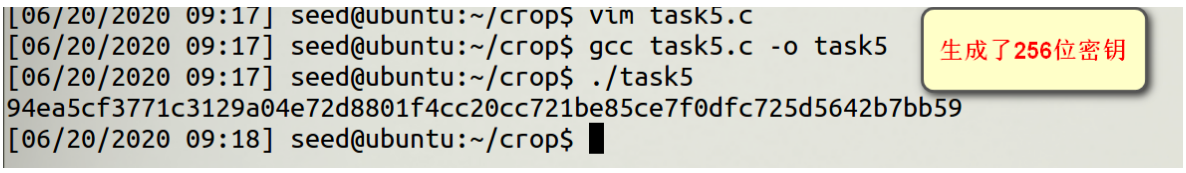
对部分/dev/urandom数据进行ent分析

如图所示：

- 熵是7.999836
- 卡方238.93
- 算数平均数127.61419
- 蒙特卡洛检验的Pi值3.13
- 串行相关系数衡量0.000883

基本由上述数据看出，/etc/urandom 随机性高。

生成256位密钥



```
[06/20/2020 09:17] seed@ubuntu:~/crop$ vim task5.c
[06/20/2020 09:17] seed@ubuntu:~/crop$ gcc task5.c -o task5
[06/20/2020 09:17] seed@ubuntu:~/crop$ ./task5
94ea5cf3771c3129a04e72d8801f4cc20cc721be85ce7f0dfc725d5642b7bb59
[06/20/2020 09:18] seed@ubuntu:~/crop$
```

生成了256位密钥

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <time.h>

#define LEN 32 // 256 bits

void main()
{
    unsigned char* key = (unsigned char * ) malloc(sizeof(unsigned char) * LEN);
    FILE * random = fopen("/dev/urandom", "r");
    fread(key, sizeof(unsigned char) * LEN, 1, random);
    fclose(random);


    int i;
    for (i = 0; i< LEN; i++){
        printf("%.2x", (unsigned char)key[i]);
    }
    printf("\n");
}
```