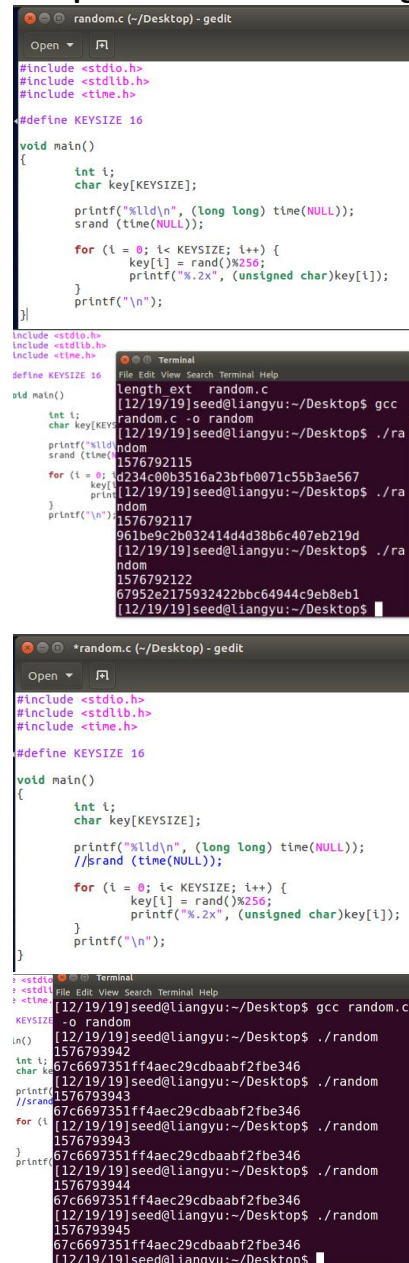


# Pseudo Random Number Generation Lab

Liangyu W

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

Compile and run the following program:



The image shows two screenshots of a C program and its execution. The first screenshot shows the source code of a program named `random.c` in a text editor. The code includes `<stdio.h>`, `<stdlib.h>`, and `<time.h>`. It defines a constant `KEYSIZE` as 16. The `main` function prints the current time, sets a seed using `srand(time(NULL))`, and then generates a 16-character key using `rand()` and `printf`. The second screenshot shows the terminal output of the program. It displays the time, the seed, and the generated key. The key is a 16-character hexadecimal string: `1576792115961be9c2b032414d4d38b6c407eb219d`.

```
#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");
}
```

```
length ext random.c
[12/19/19]seed@liangyu:~/Desktop$ gcc
random.c -o random
[12/19/19]seed@liangyu:~/Desktop$ ./ra
ndom
1576792115
961be9c2b032414d4d38b6c407eb219d
[12/19/19]seed@liangyu:~/Desktop$ ./ra
ndom
1576792122
67952e2175932422bbc64944c9eb8eb1
[12/19/19]seed@liangyu:~/Desktop$
```

```
#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");
}
```

```
> <stdio.h>
> <stdlib.h>
> <time.h>
KEYSIZE
in()
char key
//srand
for (
)
printf
[12/19/19]seed@liangyu:~/Desktop$ gcc random.c
[12/19/19]seed@liangyu:~/Desktop$ ./random
1576793942
67c6697351ff4aec29cdbaabf2fbe346
[12/19/19]seed@liangyu:~/Desktop$ ./random
1576793943
67c6697351ff4aec29cdbaabf2fbe346
[12/19/19]seed@liangyu:~/Desktop$ ./random
1576793943
67c6697351ff4aec29cdbaabf2fbe346
[12/19/19]seed@liangyu:~/Desktop$ ./random
1576793944
67c6697351ff4aec29cdbaabf2fbe346
[12/19/19]seed@liangyu:~/Desktop$ ./random
1576793945
67c6697351ff4aec29cdbaabf2fbe346
[12/19/19]seed@liangyu:~/Desktop$
```

Time() function returns the current system time. srand() function sets the seed for the random number generator. In this case, the current system time is used as seed. If srand() is function is not used, then the default seed of 1 is used for rand(), this causes the same number to be generated every time the program runs.

## Task 2: Guessing the Key

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <openssl/aes.h>
#include <string.h>

#define KEYSIZE 16

void main() {
    unsigned char plaintext[] = "\x25\x50\x44\x46\x2d\x31\x2e\x35\x8a\x25\x25\x2d\x4c\x5\x2d\x8a\x34";
    unsigned char ciphertext[] = "\xd0\xdb\xf9\xdb\xda\xbe\xef\x88\x60\x62\xaf\x65\xaa\x82";
    int i;
    int j;
    char cipher[128];
    char key[KEYSIZE];
    AES_KEY aeskeyEnc;
    int k;

    FILE *fo;
    fo = fopen("keys.txt", "w");

    for (i = 1524013729L; i <= 1524020929L; i++) {
        srand(i);
        for (j = 0; j < KEYSIZE; j++) {
            key[j] = rand() % 256;
            fprintf(fo, "%.2x", (unsigned char)key[j]);
        }
        fprintf(fo, "\n");

        unsigned char lv[] = "\x00\x00\x07\x06\x05\x04\x03\x02\x01\x00\x02\x02\x02\x02\x02";
        AES_set_encrypt_key(key, 128, &aeskeyEnc);
        AES_encrypt(plaintext, cipher, 33, &aeskeyEnc, lv, AES_ENCRYPT);

        if (memcmp(ciphertext, cipher, 16) == 0) {
            printf("Match Found\nSeed: %d\nKey : ", i);
            for (k = 0; k < KEYSIZE; k++) {
                printf("%.2x", (unsigned char)key[k]);
            }
            printf("\nCipherText: ");
            for (k = 0; k < KEYSIZE; k++) {
                printf("%.2x", (unsigned char)cipher[k]);
            }
            printf("\n\n");
        }
    }
    fclose(fo);
}
```

```
Terminal
[12/23/19]seed@liangyu:~/Desktop$ gcc random.
c -o random -lcrypto
[12/23/19]seed@liangyu:~/Desktop$ ./random

Match Found
Seed: 1524017695
Key : 95fa2030e73ed3f8da761b4eb805dfd7
CipherText: d06bf9d0dab8e8ef880660d2af65aa82

[12/23/19]seed@liangyu:~/Desktop$
```

### Task 3: Measure the Entropy of Kernel

```
Terminal
[12/23/19]seed@liangyu:~$ cat /proc/sys/kernel/random/entropy_avail
3596
[12/23/19]seed@liangyu:~$
```

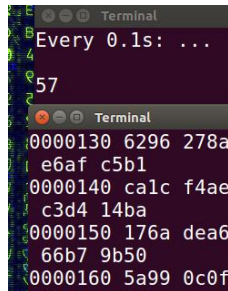
```
Terminal
Every 0.1s: c...

3429
```

Moving mouse around randomly generates significant amount of entropy.

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

Run command “cat /dev/random | hexdump”:

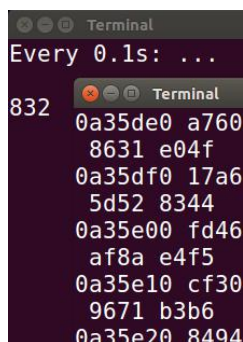
A terminal window titled 'Terminal' showing the command 'cat /dev/random | hexdump' being executed. The output displays hexadecimal data in pairs of columns. The first line shows '0000130 6296 278a' and 'e6af c5b1'. The second line shows '0000140 ca1c f4ae' and 'c3d4 14ba'. The third line shows '0000150 176a dea6' and '66b7 9b50'. The fourth line shows '0000160 5a99 0c0f'. The terminal also shows a prompt 'Every 0.1s: ...' and a cursor at the end of the last line of output.

```
Terminal
Every 0.1s: ...
57
0000130 6296 278a
e6af c5b1
0000140 ca1c f4ae
c3d4 14ba
0000150 176a dea6
66b7 9b50
0000160 5a99 0c0f
```

Not moving mouse or type anything generates little to no entropy, while moving mouse around generates a lot of entropy. /dev/random require a lot of entropy to generate a pseudo random number.

If a server uses /dev/random to generate the random session key with a client, an attacker can launch a DOS attack by keep requesting random number from /dev/random until the entropy of the randomness pool drops to zero. When this happens, /dev/random will block and the server will no longer be able to generate random session keys for other users.

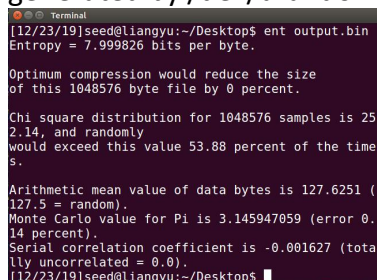
#### Task 5: Get Random Numbers from/dev/urandom

A terminal window titled 'Terminal' showing the command 'cat /dev/urandom | hexdump' being executed. The output displays hexadecimal data in pairs of columns. The first line shows '0a35de0 a760' and '8631 e04f'. The second line shows '0a35df0 17a6' and '5d52 8344'. The third line shows '0a35e00 fd46' and 'af8a e4f5'. The fourth line shows '0a35e10 cf30' and '9671 b3b6'. The fifth line shows '0a35e20 8494'. The terminal also shows a prompt 'Every 0.1s: ...' and a cursor at the end of the last line of output.

```
Terminal
Every 0.1s: ...
832
0a35de0 a760
8631 e04f
0a35df0 17a6
5d52 8344
0a35e00 fd46
af8a e4f5
0a35e10 cf30
9671 b3b6
0a35e20 8494
```

/dev/urandom keeps generating pseudo random number even when there is no user interaction with the system. Moving the mouse around increases the amount of entropy in the kernel, but doesn't seem to effect how quickly /dev/urandom generates pseudo random numbers.

Ent applies various statistical tests to the sequence of pseudo random numbers generated by /dev/urandom:

A terminal window showing the output of the 'ent' command. The output includes: 'Entropy = 7.999826 bits per byte.', 'Optimum compression would reduce the size of this 1048576 byte file by 0 percent.', 'Chi square distribution for 1048576 samples is 25.214, and randomly would exceed this value 53.88 percent of the time s.', 'Arithmetic mean value of data bytes is 127.6251 (127.5 = random).', 'Monte Carlo value for Pi is 3.145947059 (error 0.14 percent).', 'Serial correlation coefficient is -0.001627 (totally uncorrelated = 0.0).', and the prompt '[12/23/19]seed@liangyu:~/Desktop\$'.

```
Terminal
[12/23/19]seed@liangyu:~/Desktop$ ent output.bin
Entropy = 7.999826 bits per byte.

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

Chi square distribution for 1048576 samples is 25
2.14, and randomly
would exceed this value 53.88 percent of the time
s.

Arithmetic mean value of data bytes is 127.6251 (
127.5 = random).
Monte Carlo value for Pi is 3.145947059 (error 0.
14 percent).
Serial correlation coefficient is -0.001627 (tota
lly uncorrelated = 0.0).
[12/23/19]seed@liangyu:~/Desktop$
```

code to generate a 256-bit encryption key using /dev/urandom:

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

#define LEN 32 // 256 bits

int 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);

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

Terminal
[12/23/19]seed@liangyu:~/Desktop$ gcc enckey.c -o
enckey
[12/23/19]seed@liangyu:~/Desktop$ ./enckey
cf31a8c43709b3ae41c441c3c431a26fcc3b220daa9ba29c9
fe5a6b3aa20ea89
[12/23/19]seed@liangyu:~/Desktop$ ./enckey
bd5371f86339c43fd7fcd9a4b75823cea554370f94affc062
8b3cda9b105c7b2
[12/23/19]seed@liangyu:~/Desktop$ ./enckey
229dff126ab435c0125a1b0ab7bfd929508159bb64a18a16
96a1599392b90f4
[12/23/19]seed@liangyu:~/Desktop$ ./enckey
d6715367e9c227abe8c16f0ae0664a7a4d1756ef1cd5d5234
65cab38c432e09a
[12/23/19]seed@liangyu:~/Desktop$
```