Information Security - HW1

1. Joint entropy

1.1

根據 Entropy 的定義，

$H(x) = -\Sigma_{i=1}^{n} p_i \log_2 p_i.$

$H(x|y) = -\Sigma_{i=1}^{n} \Sigma_{j=i}^{m} p_{i,j} \log_2 \frac{p_{i,j}}{p_j}$

$H(x,y) = -\Sigma_{i=1}^{n} \Sigma_{j=i}^{m} p_{i,j} log_2 p_{i,j} = -\Sigma_{i=1}^{n} \Sigma_{j=i}^{m} p_{i,j} log_2 p_j \frac{p_{i,j}}{p_j}$
$= -\Sigma_{i=1}^{n} \Sigma_{j=i}^{m} p_{i,j} (\log_2 p_j + \log_2 \frac{p_{i,j}}{p_j}) = -\Sigma_{i=1}^{n} \Sigma_{j=i}^{m} p_{i,j} log_2 p_j - \Sigma_{i=1}^{n} \Sigma_{j=i}^{m} p_{i,j} log_2 \frac{p_{i,j}}{p_j}$

因為，

$p_i = \Sigma_j p_{i,j}$

所以

$-\Sigma_{i=1}^{n} \Sigma_{j=i}^{m} p_{i,j} log_2 p_j = -\Sigma_{j=1}^{m} p_j log_2 p_j = H(y)$

得証

$H(x,y) = H(y) - \Sigma_{i=1}^{n} \Sigma_{j=i}^{m} p_{i,j} log_2 \frac{p_{i,j}}{p_j} = H(y) + H(x|y)$

1.2

$H(x,y) = H(y) + H(x|y) = H(y) + -\Sigma_{i=1}^{n} \Sigma_{j=i}^{m} p_{i,j} \log_2 \frac{p_{i,j}}{p_j}$
$= H(y) - \Sigma_{i=1}^{n} \Sigma_{j=i}^{m} p_{i,j} (\log_2 p_{i,j} - \log_2 p_j) \leq H(y) - \Sigma_{i=1}^{n} \Sigma_{j=i}^{m} p_{i,j} (\log_2 p_{i,j})$
$= H(y) - \Sigma_{i=1}^{n} p_i log_2 p_i = H(y) + H(x)$

2. Multiplicative One-Time Pad

根據定義，如果是 Perfect Security

$P[C = c|M = m_0] = P[C = c|M = m_1]$

在這裡來說上述等價於， $c = E(k, m) = k * m \mod p \Rightarrow k = c * m^{-1} \mod p$ ，在 c, m 固定的情況下， 證明 k 會有唯一解。

由於 p 為質數,題目限定 m 介於 $0 \rightarrow p - 1$ ， 所以在模空間上會與質數 p 有一個唯一的模逆元

m^{-1} ，c, m^{-1} 固定，k 唯一，得證

3. Broken Cipher

$m_0 = 0^L$
$m_1 = 0^{L-1}|1$

如果 c 有偶數個 1 $c \to m_0$, 反之亦然 考慮當前 L - 1 位 有基數個 1, 也就是說最後一位必須是 1 來補足 k 要有偶數個 1 的條件，1^0 = 1 也就是說 m0 最後一位必是 1，也就是說 m0 必定會有偶數個 1，反之 m2 亦然，得證。

4. Permutation Cipher

$m_0 = 0^L$
$m_1 = 1^L$

不管怎樣變換，明文 = 密文

5. Malleability

由於題目提示是流密碼，所以猜測大概是 OTP or 線性同於，但暴力破解線性同於找不到 key, 因此就當是 OTP

解法就是直接拿明文跟密文做 xor 得到金鑰，在用金鑰 xor 新的明文做加密得到結果

結果: 6c73d5240a948c86981bc2808548

6. Linear Congruential Generator

$r_0 = 0$ 可以得到 b，接著從已知 r_0 帶入 LCG 得到 r_1，在將 r_1 帶入得到 r_2，得
$r_1 = a * r_0 + b \mod m$， $r_2 = a * r_1 + b \mod m$。
$(r_2 - r_1) = a * (r_1 - r_0) \mod m$， 以此類推可以得到
$(r_3 - r_2) = a * (r_2 - r_1) \mod m$。
假定 $d_i = r_i - r_{i-1}$ ，得 $d_3 = a * d2$ 且 $d_2 = a * d_1$，以此類推可以得 $d_i = d_{i-1} * a$， 得 $d_2 * d_0 = d_0 * a * a * d_0 = d_1^2 \mod m$，在模空間上等價於 $d_2 * d_0 - d_1 * d_1 = mk$
當 k 為整數，由於 d 是由 r 構造，因此可以做多次的 LCG 得到不同的 d， 藉由 $d_{i+2} * d_i - d_{i+1} * d_{i+1} = mk_i$ 得到多組的 mk_i，對這些 mk_i 做 GCD 便有很大的機會得到 m，不過由於這個方法有可能失敗，必須回代得到 a，檢查是否合法，如果不合法就必須找更多的 d_i，但由於m是質數，當k_i 越多 gcd = 1 的機會越大，所以這個方法會很快。

7. Slide attack

Slide attack 假設單次加密的 Feistel Function 很弱，容易被破解，假設整個加密經過 k 輪，我們找到兩組明文密文對 (sliding pair) (P_0, C_0)、(P_1, C_1)， P_0 做 k 次得到 C_0 ， P_1 為 P_0 做 1 次的結果， C_1 為 P_1 做 k 次的結果，也就是說 $P_1 = f(P_0, k_0), C_1 = f(C_1, k_1)$，將兩個明文的輪數控制在差 1，由於前面假設單次很好解，因此可以快速拿到最前面跟最後面的可能 key，接下來只要

比對把 key 的內容就好了，如果內容一樣就是找到了，不一樣就代表不是合法的 sliding pair。又由於整個系統的 key 跟 function 都是共用，所以無關輪數。

Sliding attack 跟普通攻擊不一樣的是普通攻擊著重於找到數學上的近似，而 sliding attck 著重於構造相同的資料。

8. Programming: Never Use One Time Pad Twice

結果: It is the brain, the little gray cells on which one must rely. One must seek the truth within not without.

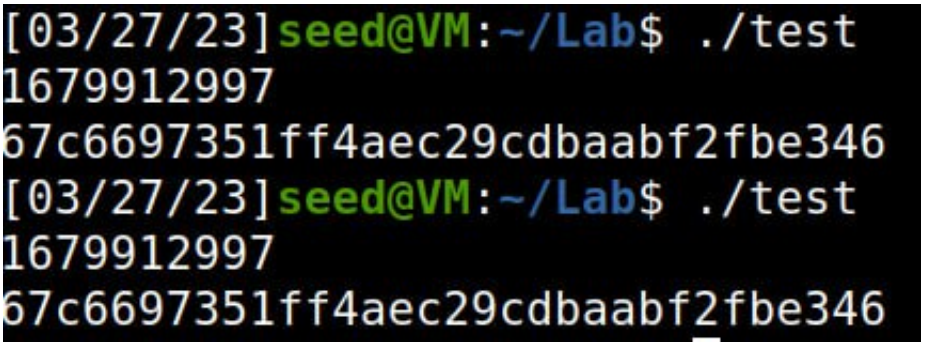
過程: 由於 xor 空格會變成大小寫互換，所以如果是一個正常的明文，隨機找兩個 ciphertext 互相 xor, 結果是字母的部份有很大機會是空格，在這邊我找空格機率最大的，也就是說對一個密文，我去 xor 其他人，結果是字母出現月多的就代表該明文有很大可能是空格，再把密文的部份 xor 空格便可以得到 key, 之後對 challenger 還有 ciphertexts 做 decrypt，在這個情況下可以得到一部分正確的字母但是有缺少，可以利用平常英文熟悉的單字，比如說 __e，那大概律是 the,也就是可以用 the 去回推原本的 key， 以此類推，便可以得到結果。

9. Lab: Pseudo Random Number Generator

9.1 Task 1

- With seed:

- Without seed:



可以看到再有 srand(time(NULL)) 的情況下每次執行的結果都不一樣，而沒有 srand 的則是每次結果都一樣，可以斷定srand() 是在在為亂數設定某種初始條件，而time(NULL) 則是使用時間，所以才會每次都不一樣。

9.2 Task 2

1. 首先先拿到該時間段的 Unix 時間戳

```
[03/27/23]seed@VM:~/Lab$ date -d "2018-04-17 23:08:49" +%s
1524020929
[03/27/23]seed@VM:~/Lab$ date -d "2018-04-17 21:08:49" +%s
1524013729
```

2. 修改並執行原本的程式已得到所有可能的 key (基本上就是第一題的程式 + for 迴圈窮舉)

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

#define KEYSIZE 16

int main() {
        char keys[KEYSIZE];
        for(long i = 1524013729; i <= 1524020929; ++i) {
                srand(i);

                for(int i = 0; i < KEYSIZE; ++i) {
                        keys[i] = rand() % 256;
                        printf("%.2x", (unsigned char) keys[i]);
                }
                printf("\n");
        }
        printf("\n");

}
```

```
[03/27/23]seed@VM:~/Lab$ vim test.c
[03/27/23]seed@VM:~/Lab$ gcc test.c -o test
[03/27/23]seed@VM:~/Lab$ ./test >> output
```

3. 用 Python 的 AES 套件拿密文解密跟明文做比較，相同的話代表找到 key 了

```python
if __name__ == "__main__":
    from Crypto.Cipher import AES
    import Crypto.Cipher.AES

    with open('output') as f:
        keys = f.readlines()

for k in keys:
    key = bytearray.fromhex(k.rstrip('\n'))
    IV = bytearray.fromhex("09080706050403020100A2B2C2D2E2F2")
    plaint = bytearray.fromhex("255044462d312e350a25d0d4c5d80a34")
    ciphert = bytearray.fromhex("d06bf9d0dab8e8ef880660d2af65aa82")
    cipher = AES.new(key,AES.MODE_CBC,IV)
    if cipher.decrypt(ciphert) == plaint:
        print(k)
        break
~
```

```
[03/27/23]seed@VM:~/Lab$ python3 test.py
95fa2030e73ed3f8da761b4eb805dfd7
```

9.3 Task 3

```
Every 0.1s: cat /proc/sys/kernel/random/entropy_avail                VM: Mon Mar 27 08:13:05 2023

1477
```

在實驗中瀏覽網站的增加速度是最快的，其次是移動滑鼠跟打字，最慢是瀏覽檔案跟點擊，不過後面幾項會根據我的速度不同而有不同的增加速度

9.4 Task 4

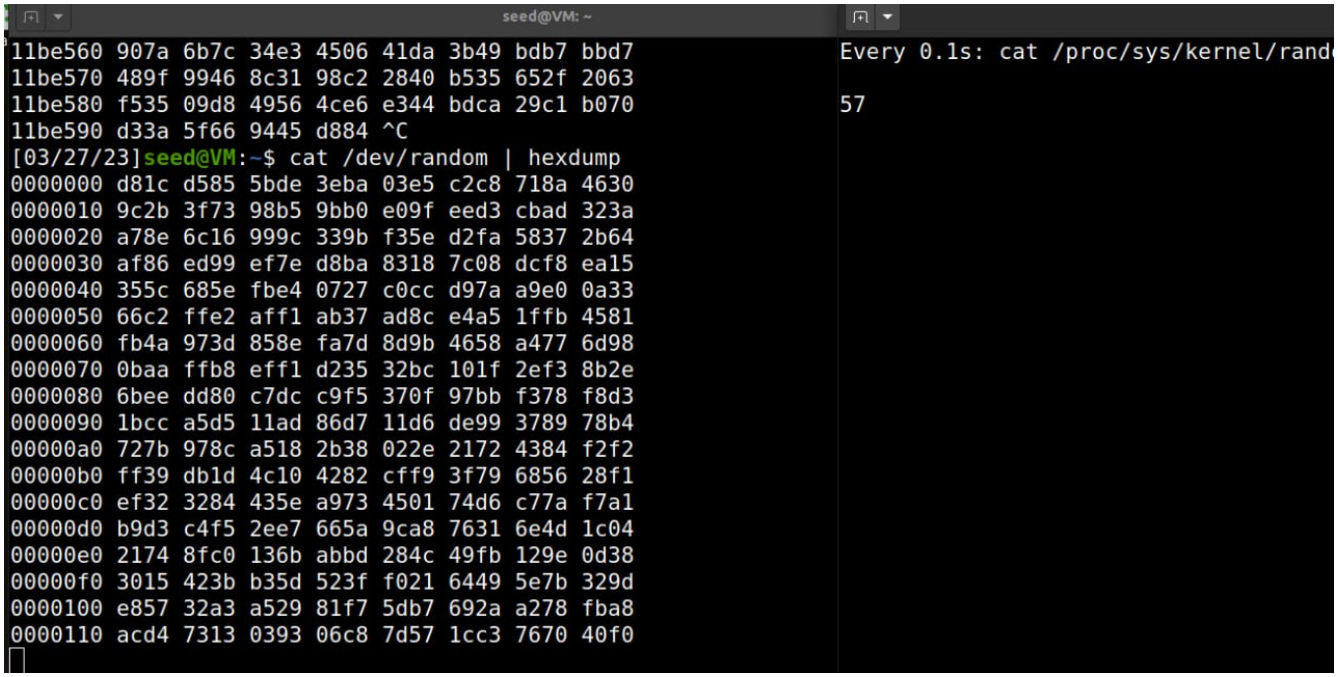
原始：

```
seed@VM: ~
11be540 0b70 7b55 467e 7658 581a a2db f5a3 c4fe
11be550 65a3 8481 9702 6a96 46c5 7d00 b649 33c7
11be560 907a 6b7c 34e3 4506 41da 3b49 bdb7 bbd7
11be570 489f 9946 8c31 98c2 2840 b535 652f 2063
11be580 f535 09d8 4956 4ce6 e344 bdca 29c1 b070
11be590 d33a 5f66 9445 d884 ^C
[03/27/23]seed@VM:~$ cat /dev/random | hexdump
0000000 d81c d585 5bde 3eba 03e5 c2c8 718a 4630
0000010 9c2b 3f73 98b5 9bb0 e09f eed3 cbad 323a
0000020 a78e 6c16 999c 339b f35e d2fa 5837 2b64
0000030 af86 ed99 ef7e d8ba 8318 7c08 dcf8 ea15
0000040 355c 685e fbe4 0727 c0cc d97a a9e0 0a33
0000050 66c2 ffe2 aff1 ab37 ad8c e4a5 1ffb 4581
0000060 fb4a 973d 858e fa7d 8d9b 4658 a477 6d98
0000070 0baa ffb8 eff1 d235 32bc 101f 2ef3 8b2e
0000080 6bee dd80 c7dc c9f5 370f 97bb f378 f8d3
0000090 1bcc a5d5 11ad 86d7 11d6 de99 3789 78b4
00000a0 727b 978c a518 2b38 022e 2172 4384 f2f2
00000b0 ff39 db1d 4c10 4282 cff9 3f79 6856 28f1
00000c0 ef32 3284 435e a973 4501 74d6 c77a f7a1
00000d0 b9d3 c4f5 2ee7 665a 9ca8 7631 6e4d 1c04
00000e0 2174 8fc0 136b abbd 284c 49fb 129e 0d38
00000f0 3015 423b b35d 523f f021 6449 5e7b 329d
```

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

48
```

移動一下之後：

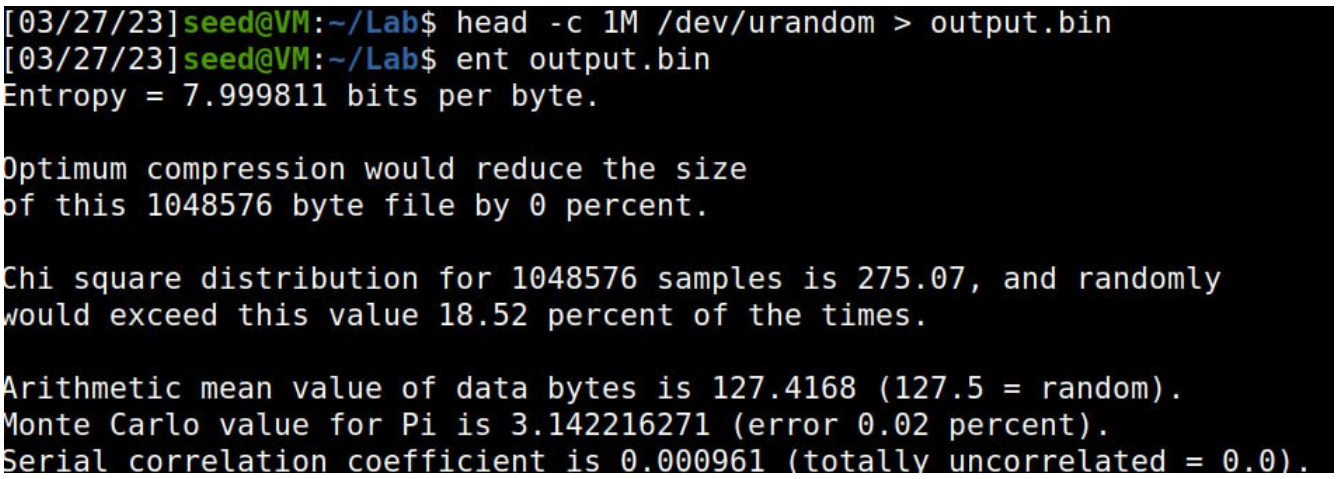


我觀察到當開始從 random 拿東西之後，entropy 下降的很快，直到消耗完便不再生成 random，當我在動幾下滑鼠使 entropy 增加後，超過一定門檻，又會開始生成 random，然後 entropy 會下降

Question: 攻擊者可以一直傳送封包，產生 random 使 entropy 下降，直到為零 Server 便再也收不到任何 request

9.5 Task 5

觀察：不管有沒有移動都不會影響結果



可以看到第二行說壓縮率為 0，也就是說檔案的亂度非常大，幾乎不可壓縮，品質很好

```
[03/27/23]seed@VM:~/Lab$ cat n.c
#include <stdio.h>
#include <stdlib.h>

#define LEN 32

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);
        fclose(random);

        for(int i = 0; i < LEN; ++i)
                printf("%.2x", key[i]);

        printf("\n");
}
[03/27/23]seed@VM:~/Lab$ gcc n.c -o n
[03/27/23]seed@VM:~/Lab$ ./n
e2ef194b979c24ad2e3be9acf5b14bed9e3b70483006eea9cfbc151f05483ea8
```