2.1

因為對相同的內容做兩次XOR，會還原出原本的結果（Ｍ⊕Ｋ⊕Ｋ＝Ｍ），因此當大量使用F2，並且對產生的結果不斷做XOR，即可藉此推斷其他結果，所以F2(k,(x, y)) := F(k, x) ⊕ F(k, y) 是不安全的。

Ex:

用(x, 0)帶入F2可得F(k, x) ⊕ F(k, 0)

用(x, 1)帶入F2可得F(k, x) ⊕ F(k, 1)

F2(k, (x, 0)) ⊕ F2(k, (x, 1)) = F(k, 0) ⊕ F(k, 1)

用(y, 0)帶入F2可得F(k, y) ⊕ F(k, 0)

將F2(k, (y, 0))與紅色部分做XOR，即可推得F2(k, (y, 1))

2.2

Feistel Network的運算結構是可逆函式

加密:

Ri = Li-1 ⊕ fi(Ri-1)

Li = Ri-1

解密:

Ri-1 = Li

Li-1 = Ri ⊕ fi(Li)

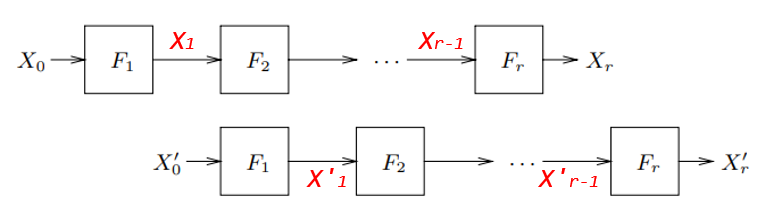
F為secure PRF，使用多個不同的key代入F，可湊出一對一函式。

因此根據PRP的定義，符合有效確定性算法、一對一函式、有效可逆算法，Feistel Network是安全的。

參考:

<https://medium.com/fcamels-notes/prg-prf-prp-b4bc86aa9d81>

2.3



以上圖為例，Slide attack的攻擊方式在於當找到一組X1與X'0結果相同時，此時稱它們為slid pair，由於加密中的Key具有相依性，所以我們可以確定，在經過相同數量的F加密後得出的結果，Xr與X'r-1也會相同，因此可以使用known-plaintext attacks來針對Fr進行key的破解，並且根據生日悖論，大約只需要O(2n/2)的已知明文，即可破解key，再加上兩兩比對時間約O(2n)，因此破解的時間長短取決於key的長度。

參考:

<https://drive.google.com/file/d/1xQm2NRv0HL4MVVwlnbIAYk7DCEsuggra/view>

<https://www.youtube.com/watch?v=IvhLd-1m6tg&t=971s>

2.4

(1)

DES的內部運作包含16次的Feistel Network，根據Feistel Network的運作得 Ri = Li-1 ⊕ F(Ri-1, Ki)

因此 = ⊕ F(,) = ⊕ F(Ri-1, Ki)

根據提示 = ⊕ B

所以 ⊕ F(Ri-1, Ki) =

因為此操作不會影響bit的值，因此S-Box輸出的值保持不變

由上述可知，DES () =

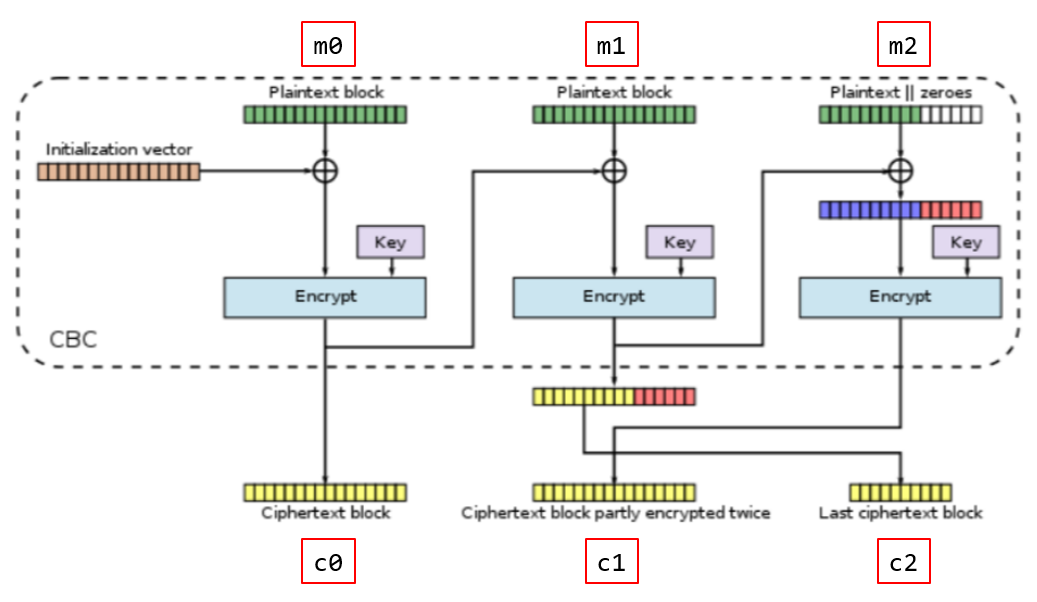
(2)

由(1)知，當我們測試 DESk(x) 等同於是測試了 DES ()，也就是測試一種key等同於測試兩種key，所以只需要測試 256/2，也就是 255 種key

參考:

<https://www.youtube.com/watch?v=qkBisYq8iIs&t=2s&ab_channel=%23%E7%B2%98%E6%B7%BB%E5%A3%BD>

2.5



幫上圖明文與密文區塊加上編號，並根據以下公式解密

解密:

m0 = Dk(C0) ⊕ IV

m2 = (Dk(c1)去掉padding部分) ⊕ c2

m1 = Dk(c2||Dk(c1)後padding部分) ⊕ C0

2.6

1. 因為gcd(4, 13) = 1

根據費馬小定理 412 ≡ 1 mod(13)

4255 = 4(12)21 \* 43

所以 4255 ≡ 4(12)21 \* 43 ≡ 64 ≡ 12 mod(13)

2. 因為gcd(7, 93) = 1

根據費馬小定理 792 ≡ 1 mod(93)

71013 = 7(92)11 \* 7

所以 71013 ≡ 7(92)11 \* 7 ≡ 7 mod(13)

2.7

若m與N不互質，假設m = k'm'以及ed – 1 = k(q-1)，則

med = (k'm')ed ≣ 0 ≣ k'm' ≣ m (mod k')

med = med-1m = mk(q-1)m = (mq-1)km ≣ 1km ≣ m (mod q)

所以 med ≣ m (mod N) 得證。

參考:

<https://zh.wikipedia.org/wiki/RSA%E5%8A%A0%E5%AF%86%E6%BC%94%E7%AE%97%E6%B3%95>

2.8

明文:

If you don't kno

w where you want

to go, then it

doesn't matter w

hich path you ta

ke. Lewis Carrol

l, Alice in Wond

erland.

原理:



根據上述公式，從第二個密文C1開始，將自行產生的IV與cipher送出，並且從最後一個byte開始嘗試，如果送到 Server為合法的 padding，則代表最後一個byte xor D(該密文的最後一個byte)為 0x01，則D(該密文的最後一個byte) = 最後一個byte xor 0x01，可以找到 D(該密文的最後一個byte)，因為D(該密文最後一個byte) xor 0x02在Server端解開後，必為0x02，接著依照以上手法從最後一組開始重複16次，每次最多嘗試256(16\*16)種可能。

最終將取得的16個byte結果與前一組cipher做XOR，即可得到前一組cipher的明文

程式碼:

import requests  
import time  
import random  
from fake\_useragent import UserAgent  
  
cipher = ["00112233445566778899aabbccddeeff",  
 "f9473924bd62ba19f2dd19c309289477",  
 "65786c8d4972fd132ec97a3a3e518191",  
 "7652a0dc44cb493881bdd841103b8bca",  
 "2d4824eef54b306f093bdc5a17dc9f46",  
 "a862217ecb6b80244fdba90fbb13c72b",  
 "ab3de8d9653be21d635a0f8d59712836",  
 "06eb64c0fbb922afd9db007f94fb9e24",  
 "a899a6c0a65b687b85f45d4840d47df4"]  
attack\_url = "http://140.122.185.210:8080/oracle/"  
def decrypt(dkci, ci\_minus\_one):  
 text = ""  
 for i in range(16):  
 temp1 = dkci[2\*i:2\*(i+1)]  
 temp2 = ci\_minus\_one[2 \* i:2 \* (i + 1)]  
 plain = int(temp1, 16) ^ int(temp2, 16)  
 text += chr(plain)  
 print('find:', text)  
 with open('ans.txt', 'a', encoding='UTF-8') as f:  
 print(text, file=f)  
  
for ciIndx in range(1, len(cipher)):  
 dk\_cipher = "0" \* 32  
 for idx in range(1, 17):  
 get\_IV = False  
 for testByte in range(256):  
 # repeat until send success  
 not\_yet = True  
 sleep = 5 # if connect fail sleep  
 while not\_yet:  
 try:  
 # generate IV  
 newIV = ""  
 for i in range(16):  
 if i == 16-idx:  
 newIV += '{:02x}'.format(testByte)  
 elif i <= 16-idx:  
 newIV += dk\_cipher[i \* 2: (i + 1) \* 2]  
 else:  
 hexStr = dk\_cipher[2\*i: 2\*(i+1)]  
 x = int(hexStr, 16) ^ idx  
 newIV += '{:02x}'.format(x)  
  
 # send and receive  
 # generate a random user-agent and add to the header  
 user\_agent = UserAgent()   
 req = requests.get(url=attack\_url + newIV + cipher[ciIndx],  
 headers={'user-agent': user\_agent.random})  
  
 print(req.text, attack\_url + newIV + cipher[ciIndx])  
 print("idx = ", idx, "newIV = ", newIV)  
 not\_yet = False  
  
 # find  
 if req.text == "valid":  
 hexStr = newIV[32-2\*idx: 32-2\*(idx-1)] # c1'  
 x = int(hexStr, 16) ^ idx # Dk(c2) = c1' xor 01, 02, ...  
 dk\_cipher = dk\_cipher[:32 - idx \* 2] + '{:02x}'.format(x) + dk\_cipher[32 - (idx \* 2) + 2:]  
 print(dk\_cipher)  
 get\_IV = True  
 # ignore all error  
 except:  
 # if connecnt fail at the same test add the sleep time  
 sleep += random.randint(3, 5)  
 print("Connection refused by the server(wait {}sec)".format(sleep))  
 time.sleep(sleep)  
 if get\_IV:  
 break  
 with open('key.txt', 'a', encoding='UTF-8') as f:  
 print(dk\_cipher, file=f)  
 # decrypt and write in to ans.txt  
 decrypt(dk\_cipher, cipher[ciIndx-1])

說明:

使用request.get()函式，因為有時候送太快會被拒絕造成error，使用try...except...當發生error時，不會中斷程式，而是讓程式sleep，在同一筆測資中，拒絕越多次，睡的時間越久，程式執行時間約18hr(取決於server到底要機掰你多久==)，最終明文存於ans.txt(後來用學校網路大約跑10分鐘就解出來了，而且還不太會有連線被拒絕問題，感覺跟網域有關…)。

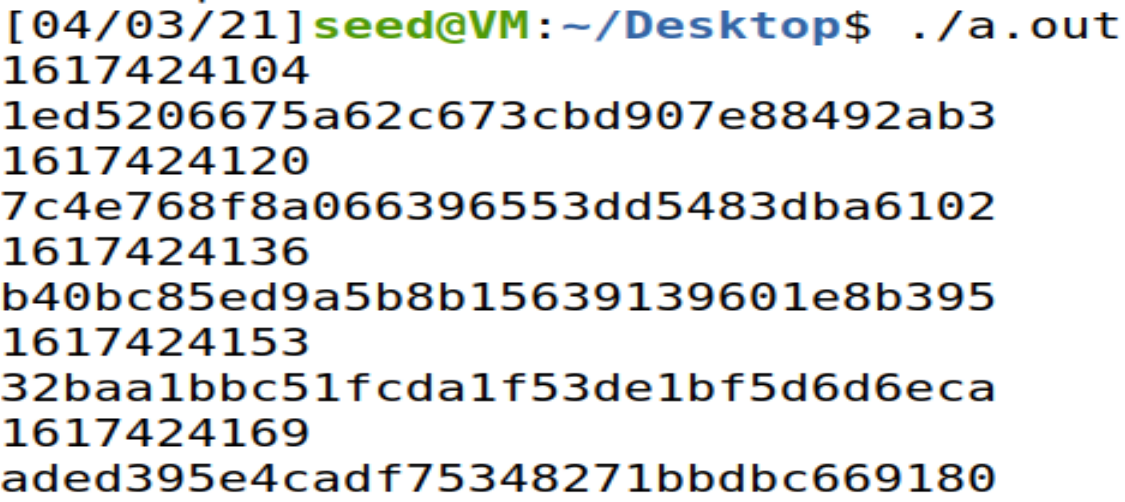
參考網站:

[密文填塞攻擊 - 維基百科，自由的百科全書 (wikipedia.org)](https://zh.wikipedia.org/wiki/%E5%AF%86%E6%96%87%E5%A1%AB%E5%A1%9E%E6%94%BB%E5%87%BB?fbclid=IwAR0Iur4IAyN2lIJc5X4B9KQ2Y7HeT9SYGNomcSPho-uyFqt1yAuQe6hfKTc)

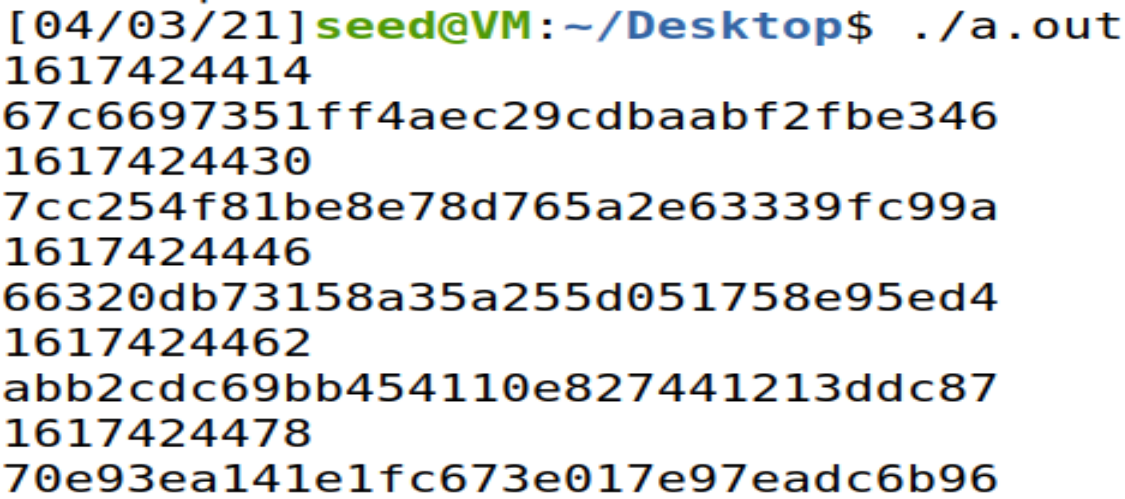
2.9

Task1

依照原程式重複5次結果:



註解掉 srand(time(NULL)) 後，重複5次結果:



srand(): 用於改變rand()種子碼

time(): 在此處用於產生一個隨環境變動的值，使種子碼具隨機性，回傳當下時間距離1970/1/1的秒數

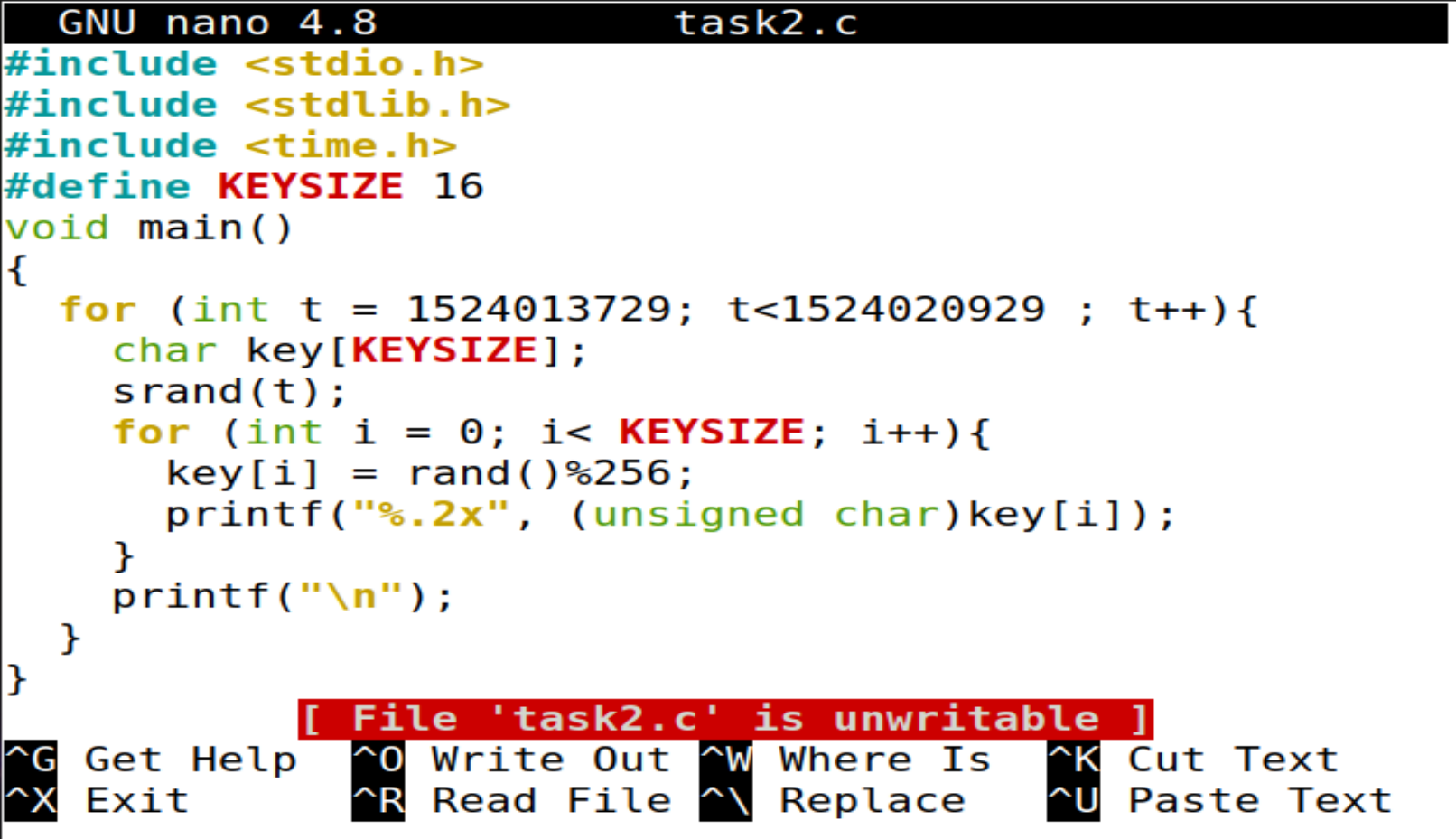
由上述結果可以發現，rand()函式如果在沒有使用srand()函式的情況下，產生的結果皆相同，因為rand()是一個PRNG，因此若未使用srand()來改變種子碼，會使rand()產生的結果皆相同，為了使結果隨機，所以要讓srand()每次改變的種子碼不同，因此使用time()函式來做為改變種子碼的變因，藉此產生隨機結果。

參考:

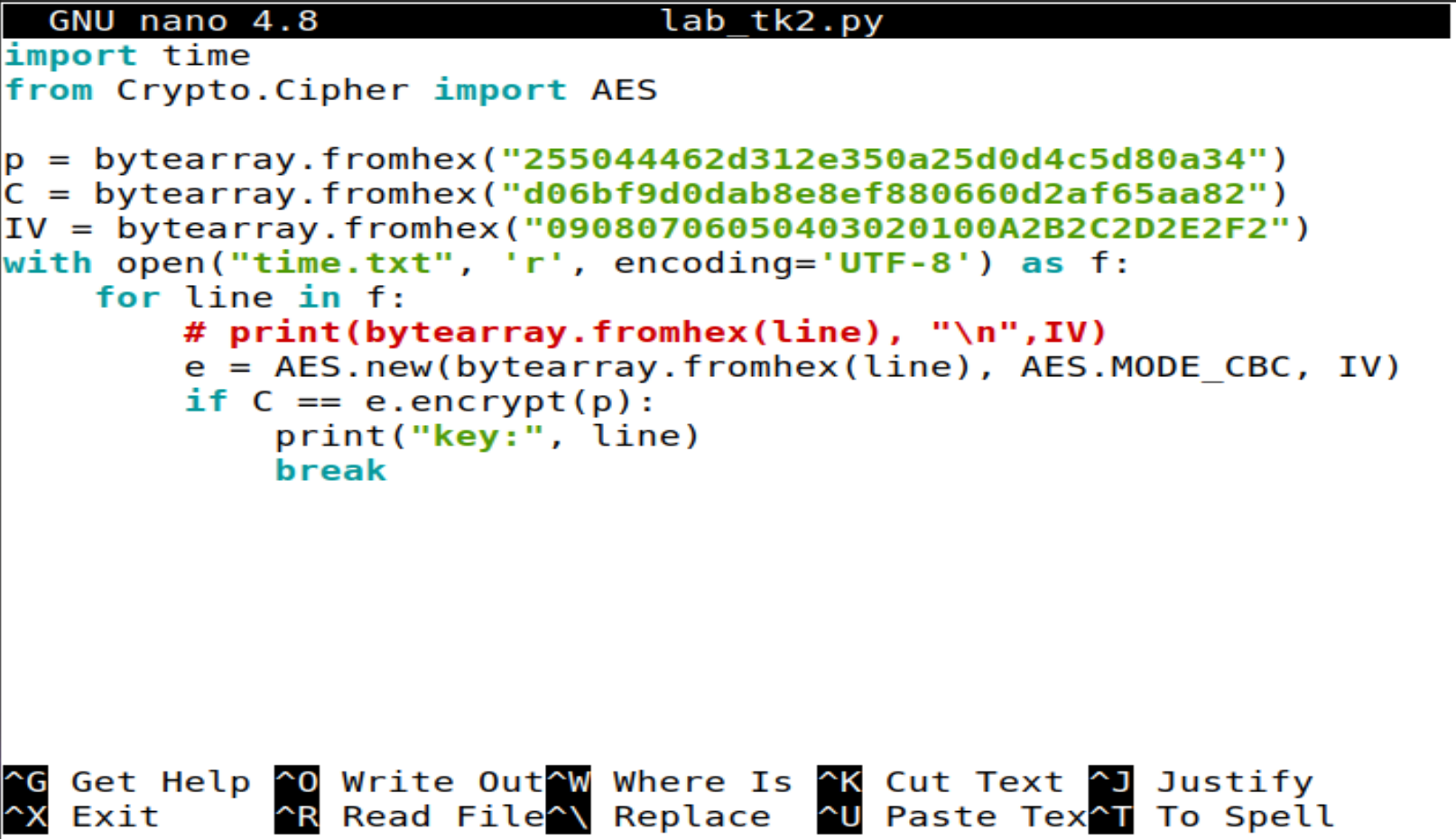
<http://yinlamdevelop.blogspot.com/2015/01/rand-srand.html>

Task2

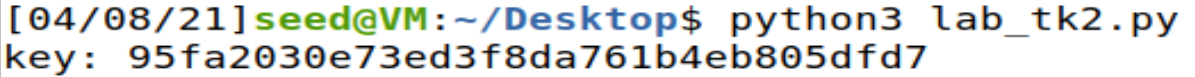
將task1程式改寫產生2018-04-17 21:08:49到2018-04-17 23:08:49的所有rand()值，並存入time.txt中



接著使用python套件做AES加密，比對密文結果找出使用的key

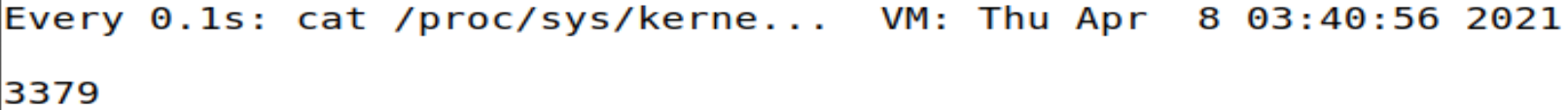


最終結果



Task3

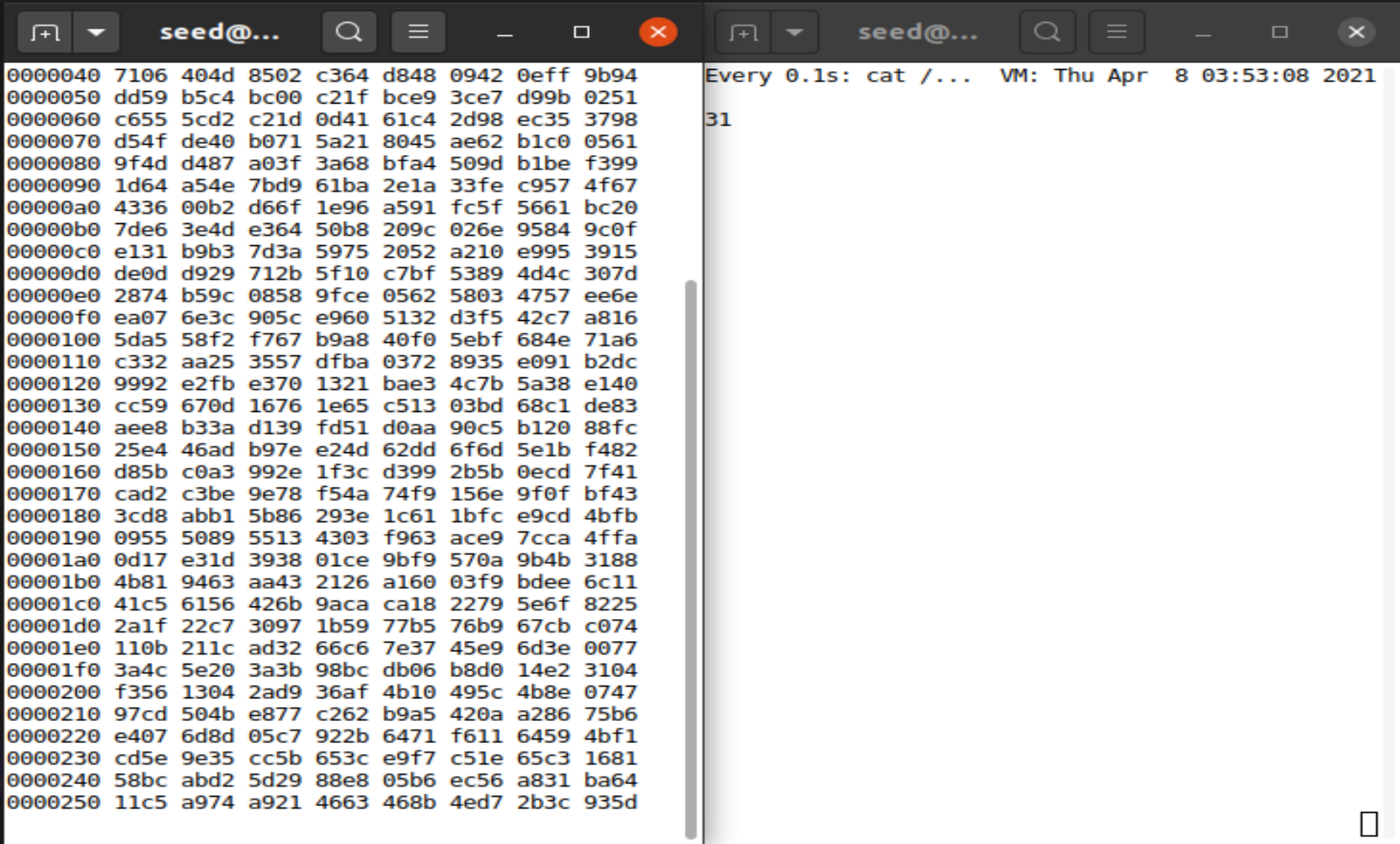
執行結果:



在輸入 watch 指令的時候，entropy就已經有一定的量了，每當使用滑鼠、鍵盤、開關檔案都會增加 entropy，快速移動滑鼠以及狂按鍵盤都會導致entropy快速增加。

Task4

執行指令後，會印出一大串亂碼如下圖



觀察後發現，每次執行該指令會立刻將entropy清空，並印出亂碼，並且之後的entropy每執行到63之後就會歸零並重新累積。

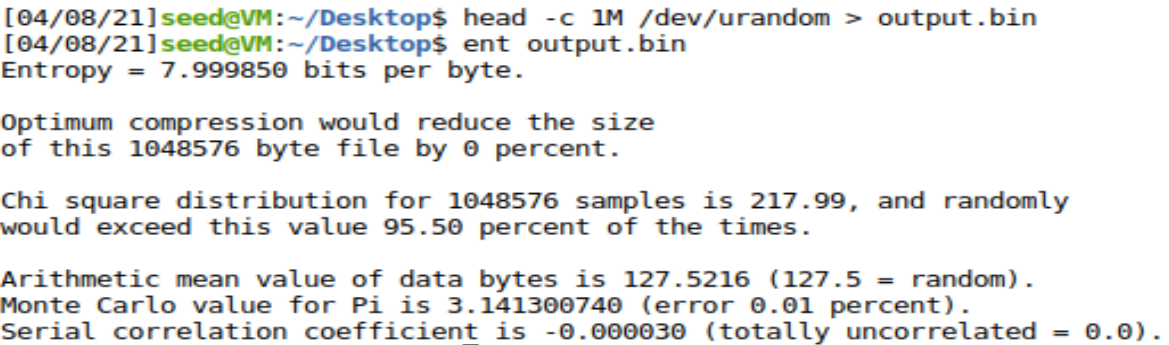
Question:

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.

解法: 持續消耗server的entropy，使其entropy量極低或歸零，即可導致server無法產生新的亂數。

Task5:

使用head –c 1M /dev/urandom 指令所產生的亂數無關滑鼠鍵盤等環境因素，透過ent觀察結果如下圖。



根據要求使用/dev/urandom 產生256-bit的key 程式碼如下圖



結果:

