**Assignment 01**

**60847013S 資工碩一 蘇冠中**

* 1. **prove Chain rule:** **H(X,Y) = H(X) + H(Y|X)**

H(X,Y) =

=

=

=

= H(X) + H(Y|X)

* 1. **with equality if and only if the random variables are independent.**

呈1.1題，H(X,Y) = H(X) + H(Y|X)

當X,Y是 independent時，H(Y|X) = H(Y)

則H(X,Y) = H(X) + H(Y)

而Chain rule可以容易地推廣到更多的變數集合

且可得到

而當的都是independent時

= + + … +

= 得證

**2.1**

Yes，

**2.2**

yes

**2.3**

yes

**2.4**

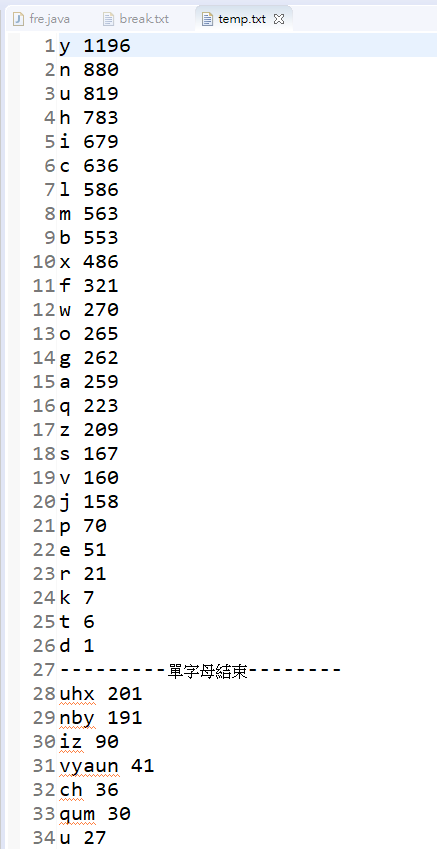
yes

**2.5**

Yes

**3. Frequency Analysis Attack**

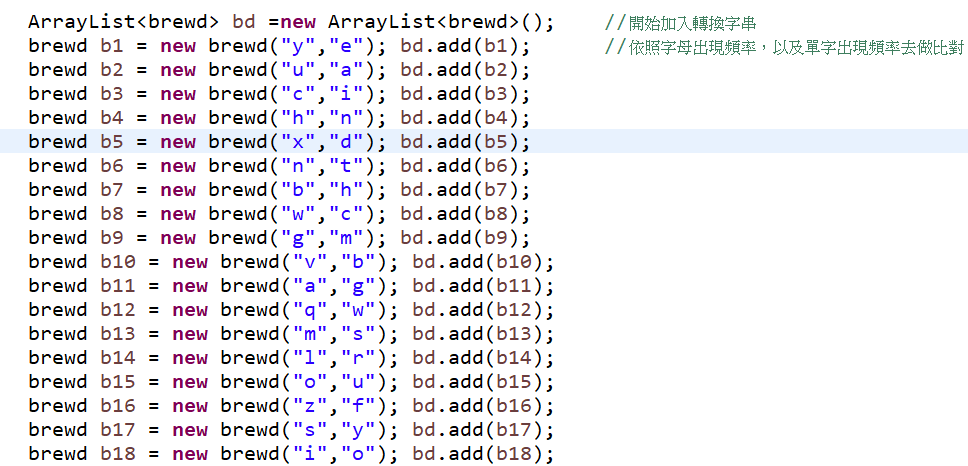
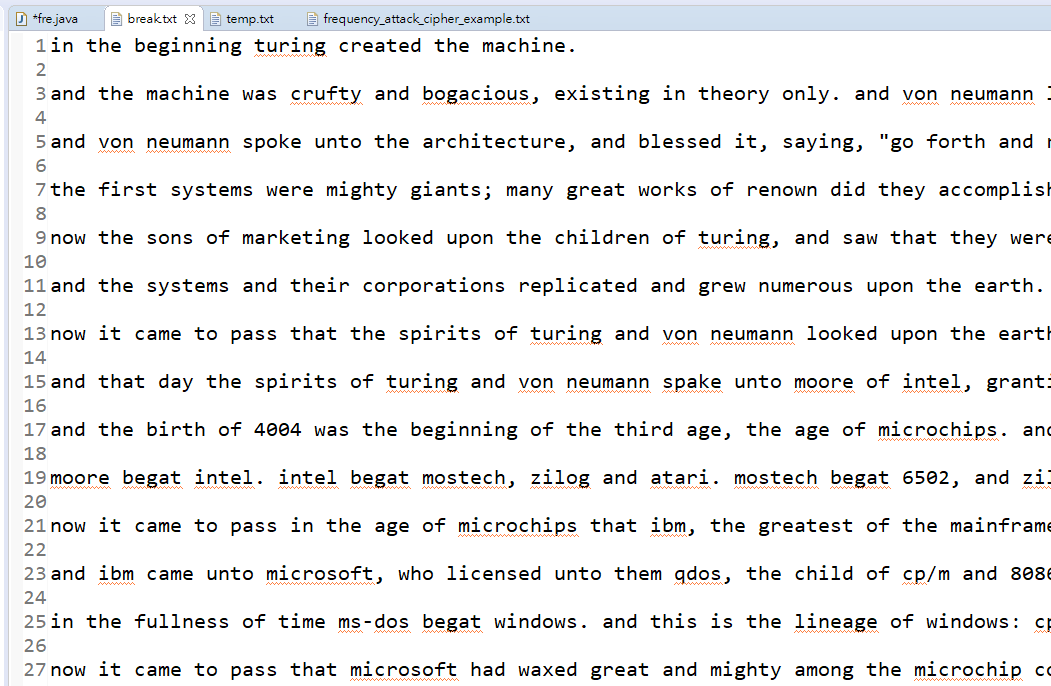
執行fre.java檔案，該檔案進行:

1. 將密文frequency\_attack\_cipher\_example.txt讀取，並且計算英文字母的出現次數、以及英文詞彙的出現次數，將此兩項存入temp.txt檔中。
2. 依照最常出現的英文字母頻率順序、以及最常出現的英文詞彙進行對照(像是the,is,in,a,of……)，一一轉換過去，[nby=the],

[uhx=and],[u=a],[ch=in],

[iz=of],將最多出現的先進行猜測轉換，並且與單一個字母也對照後，慢慢地再將其他字母轉換回來。

1. 將已知的單字先加入自製的

轉換表中，並且讀取密文檔，依照已有的轉換表去解回明文，並將名文中某些轉換不完全的詞彙去做猜測，繼續增加轉換表的單字，一個一個解到26個單字母都解完為止

全部解完寫入break.txt中，**break.txt即為明文**

**4.Never Use One Time Pad Twice**

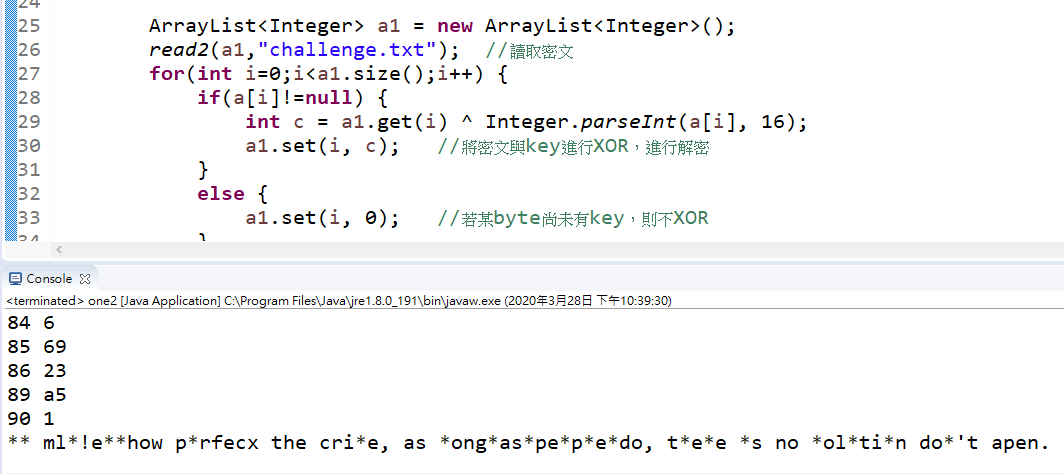
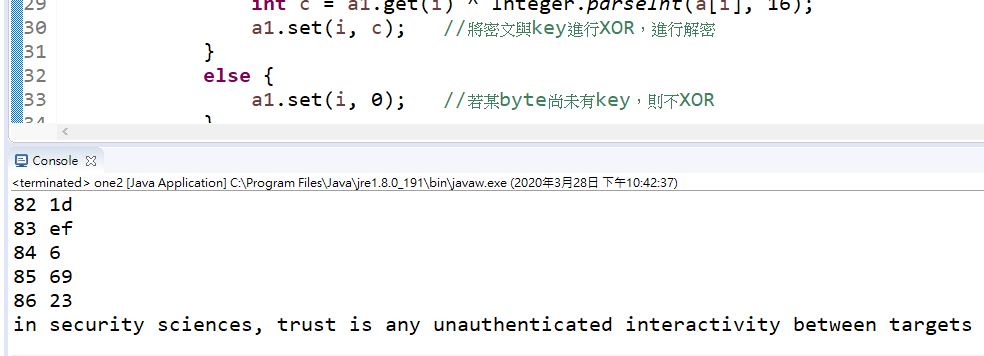
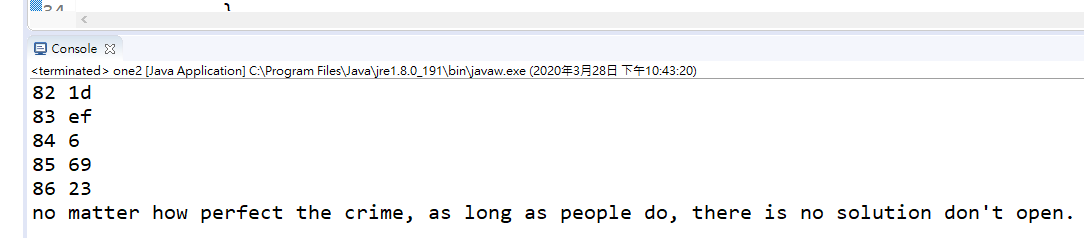
執行one.java檔案，該檔進行:

1. 讀取各個cipher1~10，並存入Array中，依照題目給的題示，當空白” ”與英文字進行XOR時，結果會是該英文字的大小寫對換，因此假設key XOR m1 = c1,key XOR m2 = c2,……，則若

m1 XOR m2為英文字時，可知m1或m2該byte位置其中一個為空白” “。

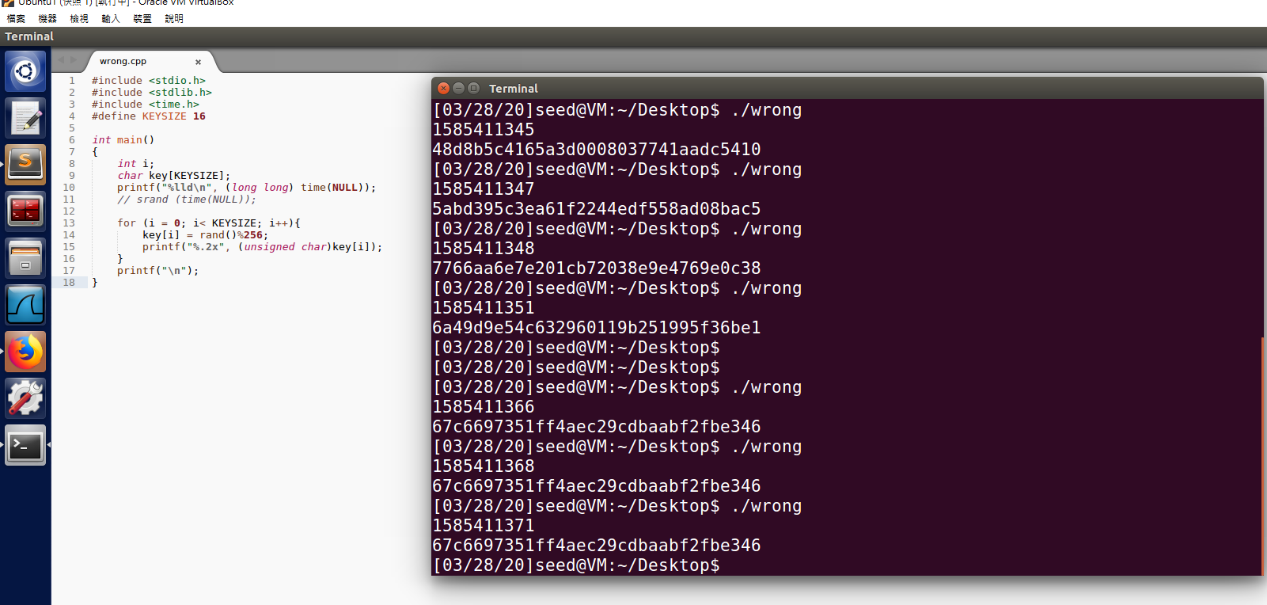
1. 因此將cipher1與其他9個cipher進行XOR，計算每個byte位置出現英文字的次數，該位置出現英文字的次數越高，代表cipher1該位置的原文越有可能是” “空白，則將” “空白與cipher的密文進行XOR，即可得出該位置加密的key。
2. 再來將cipher2與其他9個也做XOR，直到所有cipher都做完XOR後，一每個cipher解出來的不同位置的key，寫入key.txt檔中。

執行one2.java檔案，該檔執行:

1. 讀取key.txt檔，以及challenge.txt檔，將challenge依照可的key去進行解密，首先得到[\*\* ml\*\*e\*\*how p\*rfecx the cri\*e, as \*ong\*as\*pe\*p\*e\*do, t\*e\*e \*s no \*ol\*ti\*n do\*’t apen.]的解密文，可以猜得出的單字文how perfect the crime, as long as people do, there is no solution don’t open.。
2. 先依照前面英文的語法判斷，前面兩個字可能為no matter，依照這兩個字解回去的key，再與其他cipher進行解密後發現所有原文都是解開的，因此再將challenge解密完的的key(使用test2.java檔轉成key)，代回去所有的cipher，得到所有cipher解密的明文，並且都是正確的，則challenge解出來的明文即為[**no matter how perfect the crime, as long as people do, there is no solution don’t open.**]。

**5.Pseudo Random Number Generator**

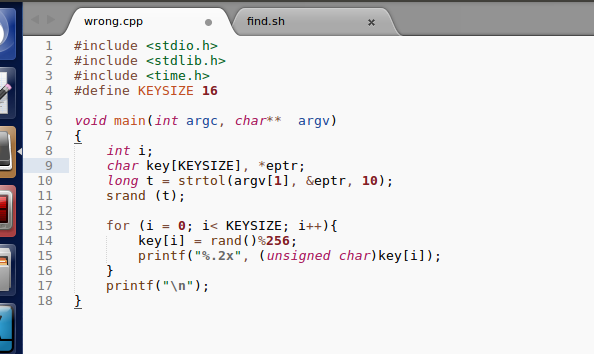
**Task1:Generate Encryption Key in a Wrong Way**

先執行lab中的程式，可得到在不同時間，time(NULL)執行產生的結果都不同，執行的第一行是time(NULL)，第二行是產生的key，執行四次後，將srand(time(NULL))註解掉，可看到後三次執行雖然time(NULL)的部分不一樣，但是產生出來的key是一樣的。

因此可知，srand()的目的是要設定seed，且代入的是time()，因每次代入的time()會隨時間改變，因此可以產生出不同的亂數key

。

**Task2:Guessing the Key**

題目使用的key產生器為Task1的程式，因此我將wrong.cpp改成用terminator執行時後面加參數代入，srand(t)即為代入的參數。

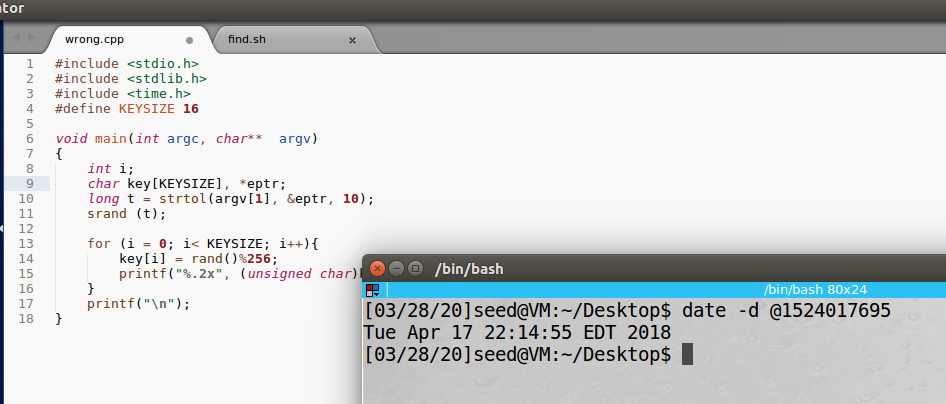
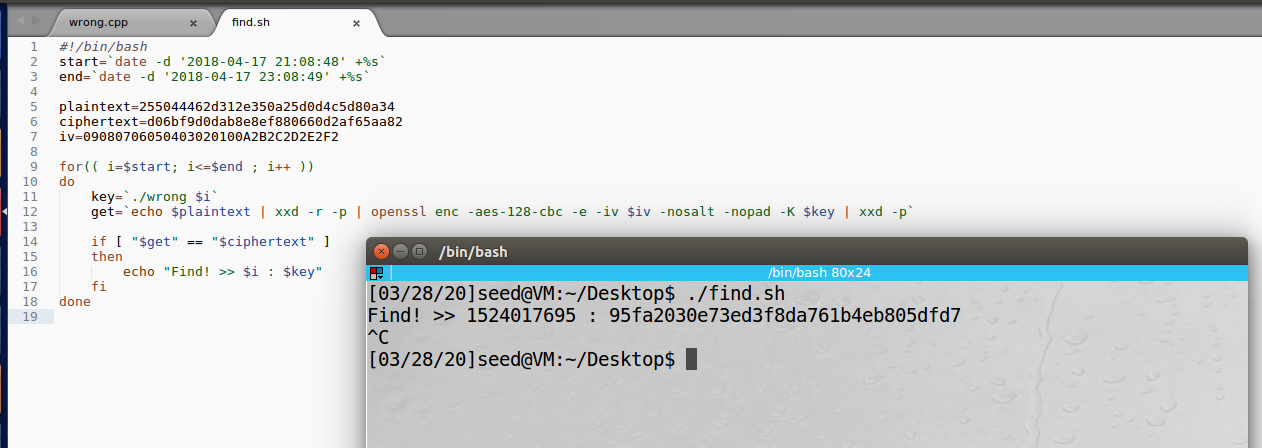
依據題目的提示，srand(t)代入的時間點為[2018-04-17 21:08:48]~

[2018-04-17 23:08:49]這兩個小時中間，因此多寫一個find.sh的shell script檔，該檔從2018-04-17的21:08:48每次加一秒到23:08:49，代入wrong去產生金鑰。

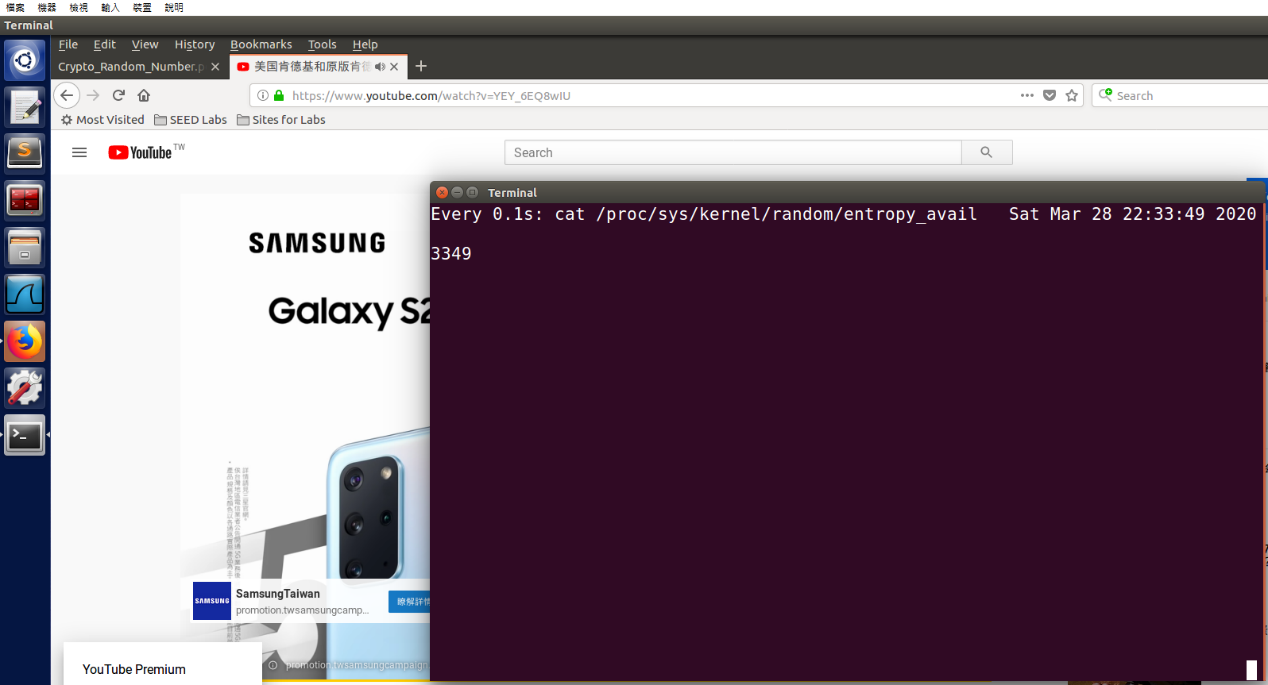
且加入Ubuntu內建的openssl aes-128-cbc進行解密，輸入plaintext、ciphertext、iv，當我的wrong產生的金鑰去加密plaintext後與ciphertext結果相同時，則將該時間點秒數以及加密金鑰印出來。

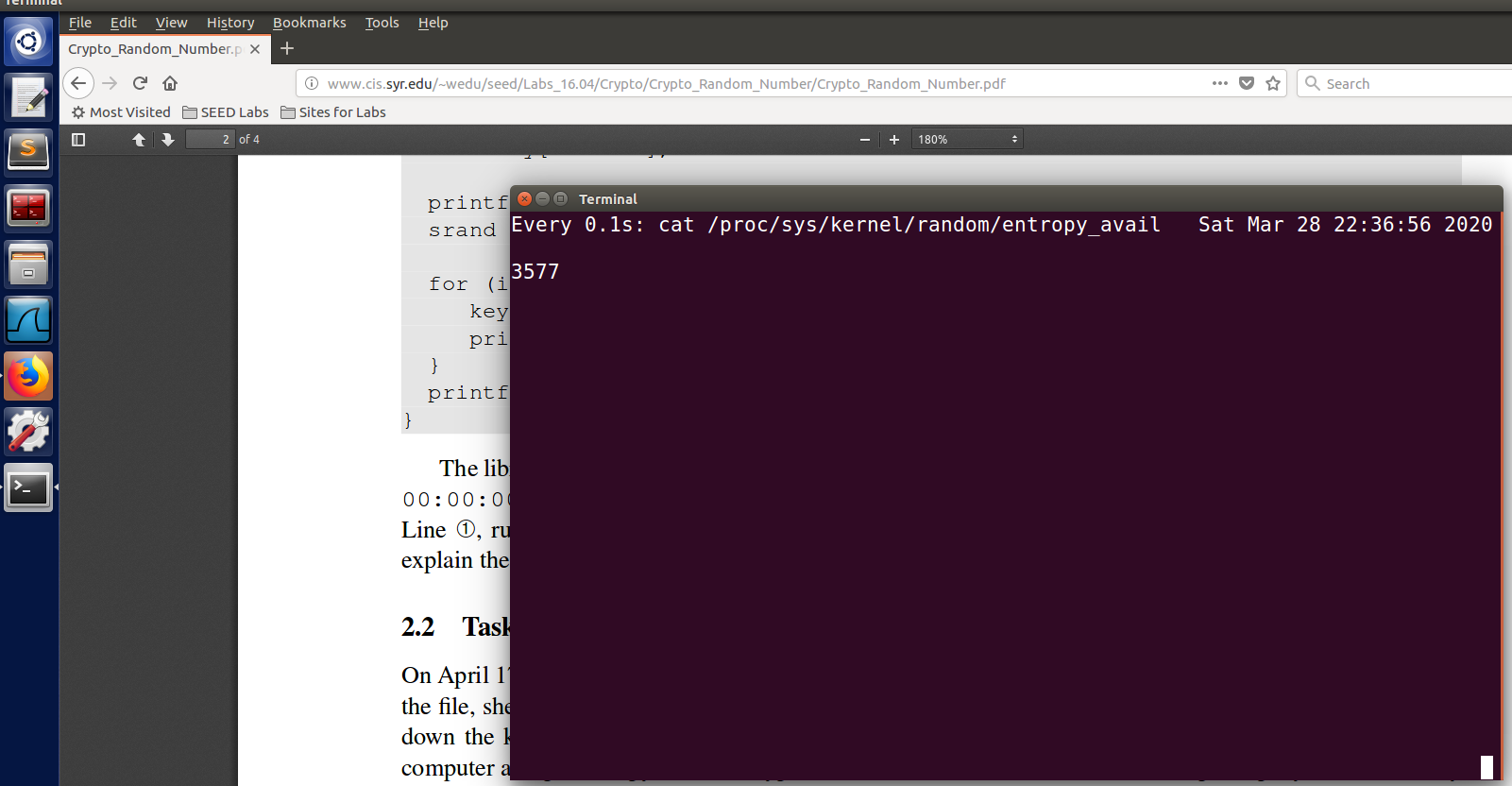
**執行結果顯示金鑰為: 95fa2030e73ed3f8da761b4eb805dfd7**

**時間點為: 2018-04-17-22:14:55**

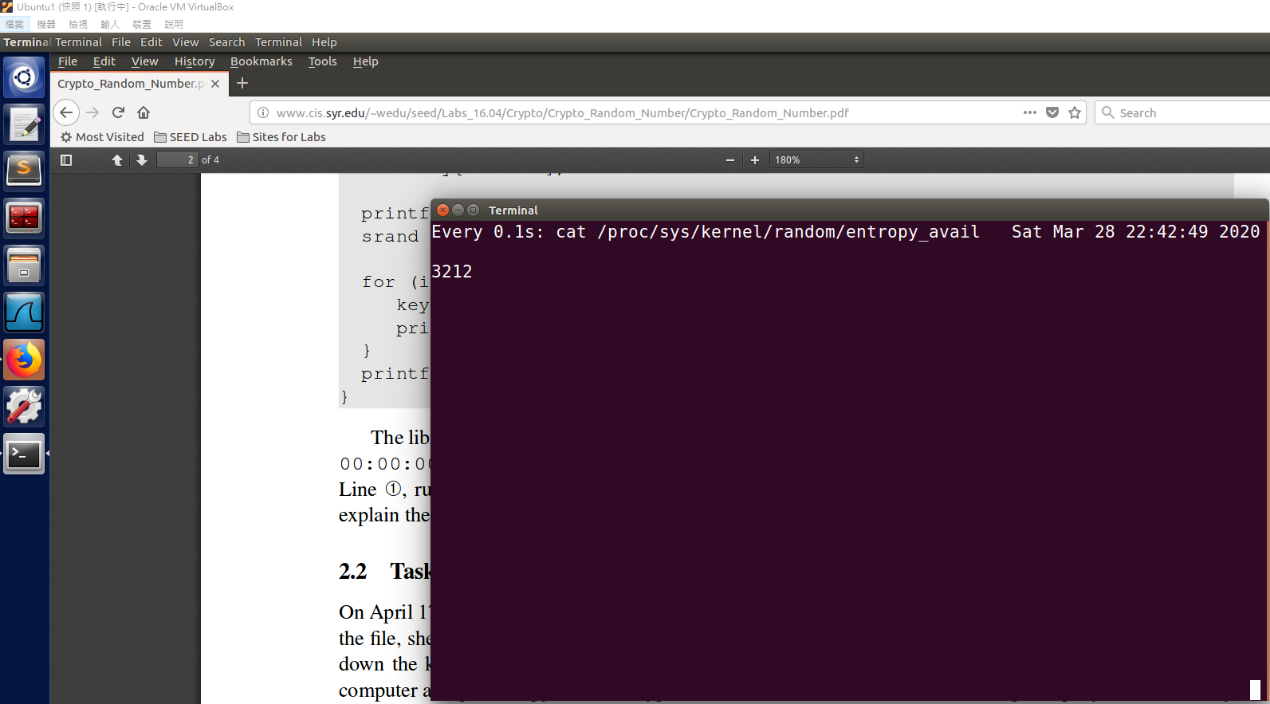
**確實是在2小時的範圍內。**

**Task3:Measure the Entropy of Kernel**

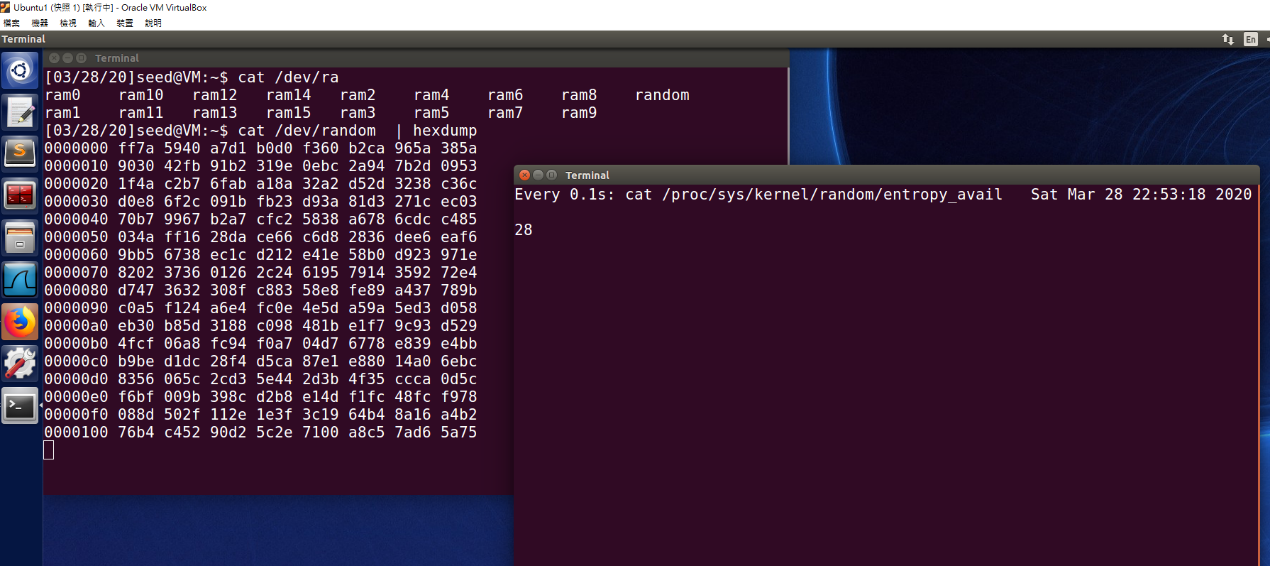
執行 watch -n .1 cat /proc/sys/kernel/random/entropy\_avail後，可以觀測到entropy\_avail每0.1秒的變化。

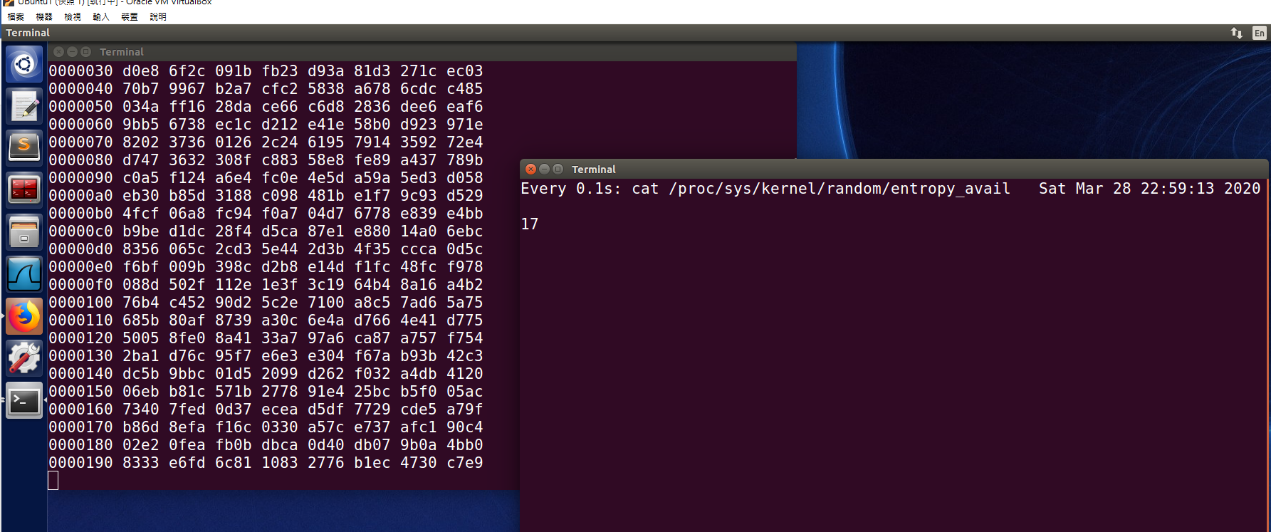


通過觀察可發現:**移動滑鼠、開啟檔案、開啟網頁等動作都會增加數值，而移動滑鼠增加的數值較少，當開啟網頁瀏覽影片時增加的幅度較大，但數值也不是無限增長的，一段時間沒用後，數值也會自動往下掉。**



**Task4: Get Pseudo Random Numbers from /dev/random**

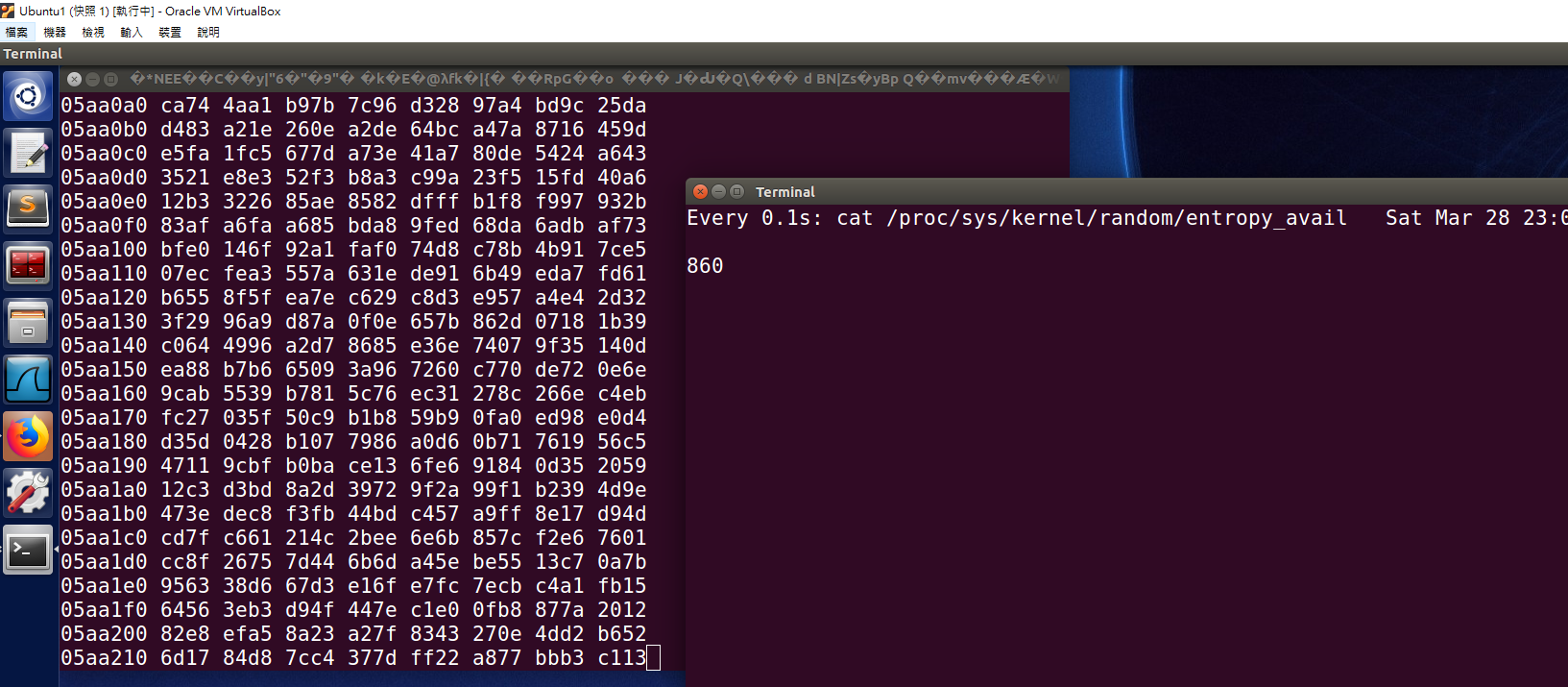
通過執行 cat /dev/random | hexdump後可發現entropy\_avail瞬間從3000多下降到20、30，並且/dev/random會獲得一串16進位的數值。

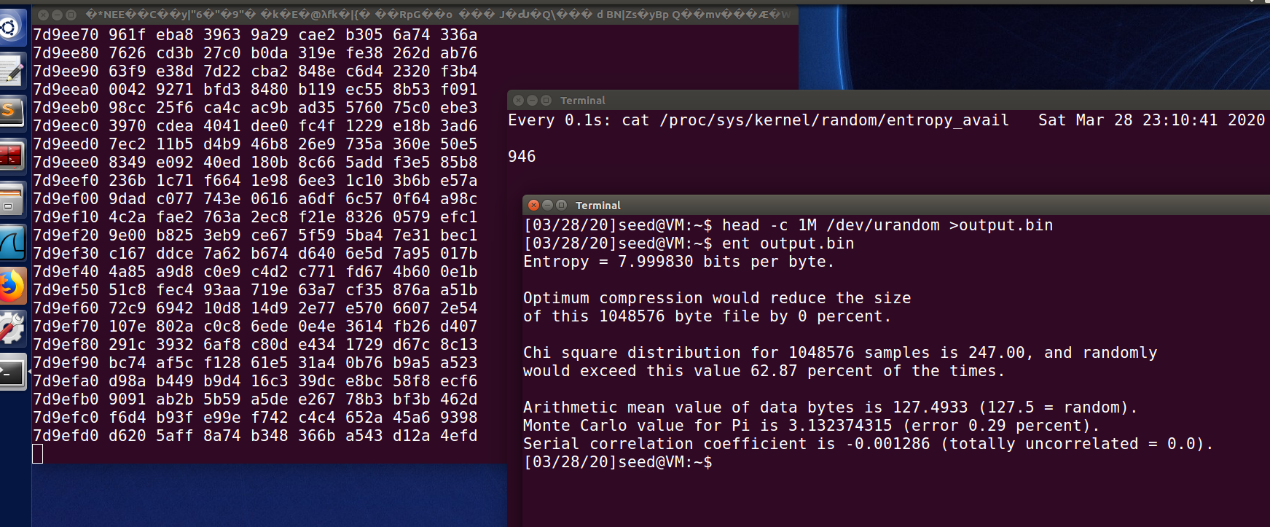
且此時entropy\_avail增加到64後會清空為0，清空2~3次後，/dev/random會再獲得一串16進位的新數值。

觀察結果得知:**執行/dev/random會限制entropy\_avail，並且將數值限於64，當超過會清空為0，且/dev/random收集到一定entropy\_avail數值後會產生新的一串16進位數值。**

**Task5: Get Random Numbers from /dev/urandom**

執行 cat /dev/urandom | hexdump後，發現/dev/urandom會不斷輸出數值，而entropy\_avail會有小幅度的增減，移動滑鼠會增加entropy\_avail的數值，但不會影響到/dev/urandom。

與/dev/random不同，/dev/urandom不會限制entropy\_avail的數值。

依照題目要求，用/dev/urandom產生1M的output.bin檔案，並以ent這個內建程式分析output產生的是不是亂數，ent跑了6個測試:

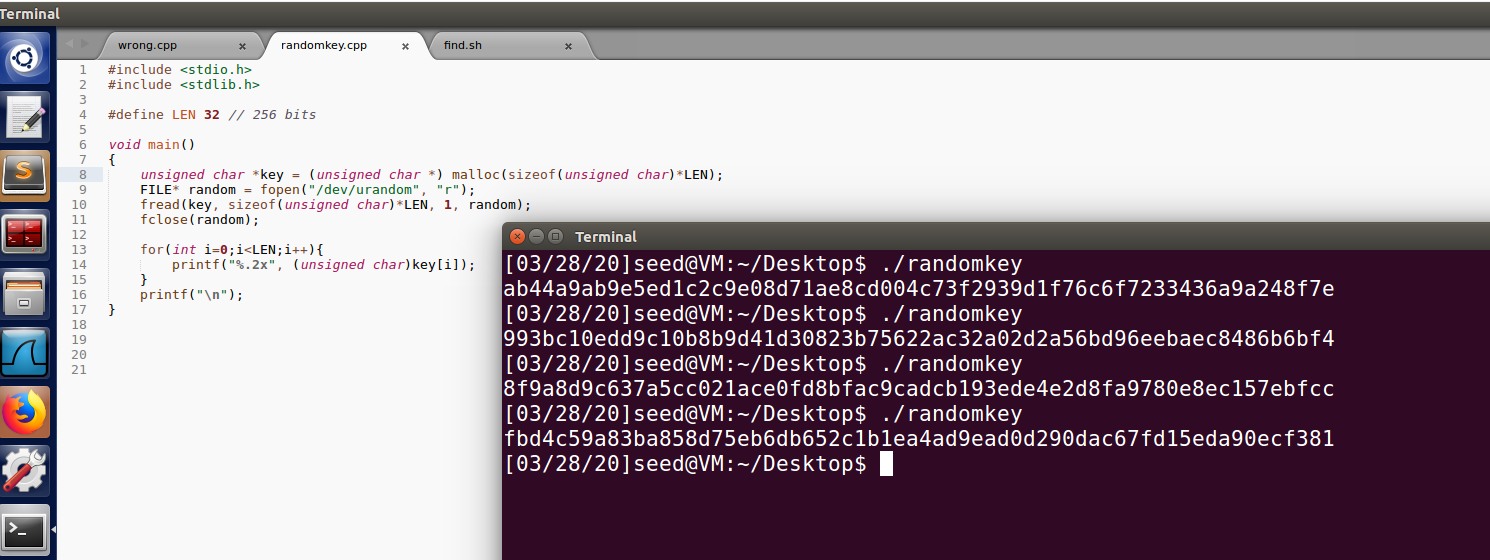
1. **Entropy:資訊密度測試，與8 bits per byte越相近則越random，我的結果為7.999830 bits per byte**
2. **Optimum compression:資料重複程度，越接近0%越好，我的結果為0%**
3. **Chi square distribution:卡方分布，介於10%~90%間比較好，我的結果為62.87%**
4. **Arithmetic mean:越接近127.5越好，我的結果為127.4933**
5. **Monte Carlo value for Pi:越接近3.14159265越好，我的結果為3.132374315**
6. **Serial correlation coefficient:每個byte與前一個byte的相依程度，越接近0越好，我的結果為-0.001286。**

**由此結果來看，用/dev/urandom產生的亂數品質是良好的。**

最後，題目要求我們將範例程式修改，改成產出256-bit的金鑰

**結合wrong與題目給的範例，寫成randomkey.cpp。**

**首先將define LEN 改成 32(256bits)**

**並且讀取/dev/urandom中sizeof(unsigned char)\*LEN長度的值作為key，最後印出key來，執行四次後可看到每次的random key都不同。**

**------------------結束------------------**