

2.1

因為對相同的內容做兩次 XOR，會還原出原本的結果 ($M \oplus K \oplus K = M$)，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 $F2(k, (x, y)) := F(k, x) \oplus F(k, y)$ 是不安全的。

Ex:

用 $(x, 0)$ 帶入 F2 可得 $F(k, x) \oplus F(k, 0)$

用 $(x, 1)$ 帶入 F2 可得 $F(k, x) \oplus F(k, 1)$

$F2(k, (x, 0)) \oplus F2(k, (x, 1)) = F(k, 0) \oplus F(k, 1)$

用 $(y, 0)$ 帶入 F2 可得 $F(k, y) \oplus F(k, 0)$

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:

$$R_i = L_{i-1} \oplus f_i(R_{i-1})$$

$$L_i = R_{i-1}$$

解密:

$$R_{i-1} = L_i$$

$$L_{i-1} = R_i \oplus f_i(L_i)$$

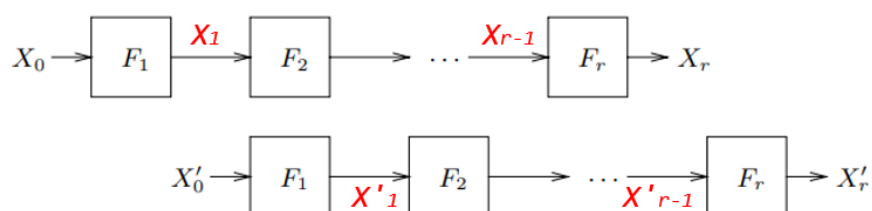
F 為 secure PRF，使用多個不同的 key 代入 F，可湊出一對一函式。

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 X_1 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， X_r 與 X'_{r-1} 也會相同，因此可以使用 known-plaintext attacks 來針對 F_r 進行 key 的破解，並且根據生日悖論，大約只需要 $O(2^{n/2})$ 的已知明文，即可破解 key，再加上兩兩比對時間約 $O(2^n)$ ，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 $R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$

因此 $\overline{R_i} = \overline{L_{i-1}} \oplus \overline{F(R_{i-1}, K_i)} = \overline{L_{i-1}} \oplus F(R_{i-1}, K_i)$

根據提示 $\overline{A \oplus B} = \overline{A} \oplus B$

所以 $\overline{L_{i-1}} \oplus F(R_{i-1}, K_i) = \overline{L_{i-1}} \oplus F(R_{i-1}, K_i)$

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， $DES_{\overline{k}}(\overline{x}) = \overline{Y}$

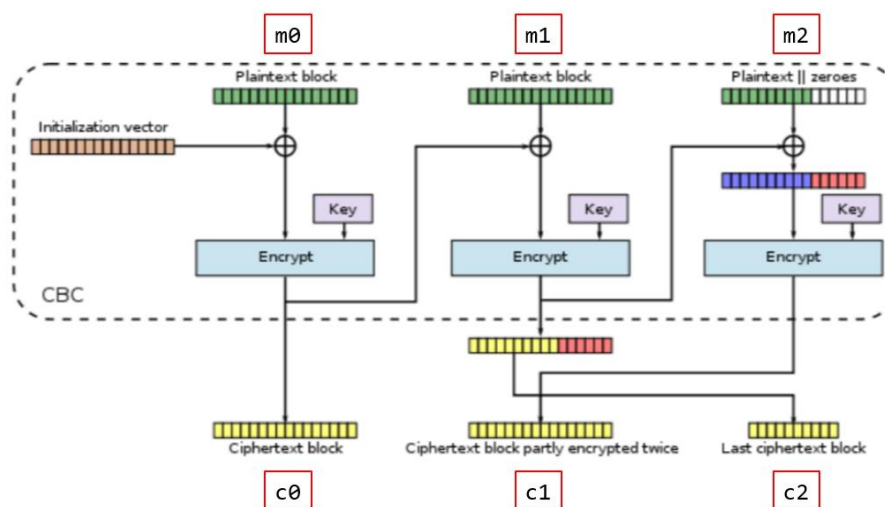
(2)

由(1)知，當我們測試 $DES_k(x)$ 等同於是測試了 $DES_{\overline{k}}(\overline{x})$ ，也就是測試一種 key 等同於測試兩種 key，所以只需要測試 $2^{56}/2$ ，也就是 2^{55} 種 key

參考：

https://www.youtube.com/watch?v=qkBisYq8iIs&t=2s&ab_channel=%23E7%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：

$$m_0 = D_k(C_0) \oplus IV$$

$$m_2 = (D_k(c_1) \text{ 去掉 padding 部分}) \oplus c_2$$

$$m_1 = D_k(c_2 || D_k(c_1) \text{ 後 padding 部分}) \oplus C_0$$

2.6

1. 因為 $\gcd(4, 13) = 1$

根據費馬小定理 $4^{12} \equiv 1 \pmod{13}$

$$4^{255} = 4^{(12)21} * 4^3$$

$$\text{所以 } 4^{255} \equiv 4^{(12)21} * 4^3 \equiv 64 \equiv 12 \pmod{13}$$

2. 因為 $\gcd(7, 93) = 1$

根據費馬小定理 $7^{92} \equiv 1 \pmod{93}$

$$7^{1013} = 7^{(92)11} * 7$$

$$\text{所以 } 7^{1013} \equiv 7^{(92)11} * 7 \equiv 7 \pmod{13}$$

2.7

若 m 與 N 不互質，假設 $m = k'm'$ 以及 $ed - 1 = k(q-1)$ ，則

$$m^{ed} = (k'm')^{ed} \equiv 0 \equiv k'm' \equiv m \pmod{k'}$$

$$m^{ed} = m^{ed-1}m = m^{k(q-1)}m = (m^{q-1})^k m \equiv 1^k m \equiv m \pmod{q}$$

所以 $m^{ed} \equiv m \pmod{N}$ 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 know where you want to go, then it doesn't matter which path you take. Lewis Carroll, Alice in Wonderland.

原理：

$$P_i = D_K(C_i) \oplus C_{i-1},$$

$$C_0 = IV.$$

根據上述公式，從第二個密文 C_1 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",
          "a862217ecb6b80244fdb90fbb13c72b",
          "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[ciIdx],

                           headers={'user-agent': user_agent.random})

        print(req.text, attack_url + newIV + cipher[ciIdx])

        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 connect 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` 到底要機辦你多久 ==)，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)

2.9

Task1

依照原程式重複 5 次結果：

```
[04/03/21] seed@VM:~/Desktop$ ./a.out
1617424104
1ed5206675a62c673cbd907e88492ab3
1617424120
7c4e768f8a066396553dd5483dba6102
1617424136
b40bc85ed9a5b8b15639139601e8b395
1617424153
32baa1bbc51fcda1f53de1bf5d6d6eca
1617424169
aded395e4cadf75348271bbdbc669180
```

註解掉 `srand(time(NULL))` 後，重複 5 次結果：

```
[04/03/21] seed@VM:~/Desktop$ ./a.out
1617424414
67c6697351ff4aec29cdbaabf2fbe346
1617424430
7cc254f81be8e78d765a2e63339fc99a
1617424446
66320db73158a35a255d051758e95ed4
1617424462
abb2cdc69bb454110e827441213ddc87
1617424478
70e93ea141e1fc673e017e97eadc6b96
```

`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

```
GNU nano 4.8 task2.c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define KEYSIZE 16
void main()
{
    for (int t = 1524013729; t<1524020929 ; t++){
        char key[KEYSIZE];
        srand(t);
        for (int i = 0; i< KEYSIZE; i++){
            key[i] = rand()%256;
            printf("%.2x", (unsigned char)key[i]);
        }
        printf("\n");
    }
}
[ File 'task2.c' is unwritable ]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text
^X Exit ^R Read File ^\ Replace ^U Paste Text
```

接著使用 python 套件做 AES 加密，比對密文結果找出使用的 key

```
GNU nano 4.8 lab tk2.py
import time
from Crypto.Cipher import AES

p = bytearray.fromhex("255044462d312e350a25d0d4c5d80a34")
C = bytearray.fromhex("d06bf9d0dab8e8ef880660d2af65aa82")
IV = bytearray.fromhex("09080706050403020100A2B2C2D2E2F2")
with open("time.txt", 'r', encoding='UTF-8') as f:
    for line in f:
        # print(bytearray.fromhex(line), "\n", IV)
        e = AES.new(bytearray.fromhex(line), AES.MODE_CBC, IV)
        if C == e.encrypt(p):
            print("key:", line)
            break

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell
```

最終結果

```
[04/08/21]seed@VM:~/Desktop$ python3 lab_tk2.py
key: 95fa2030e73ed3f8da761b4eb805dfd7
```


Task3

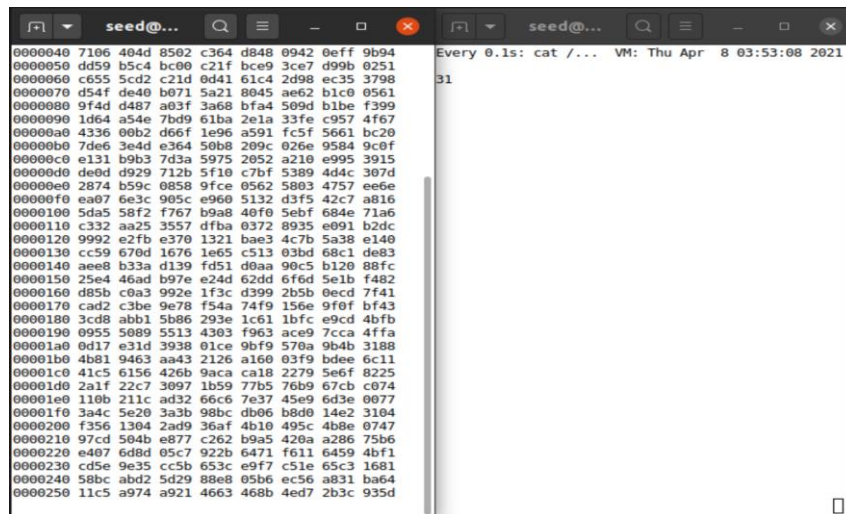
執行結果：

```
Every 0.1s: cat /proc/sys/kerne... VM: Thu Apr 8 03:40:56 2021
3379
```

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。

```
[04/08/21]seed@VM:~/Desktop$ head -c 1M /dev/urandom > output.bin
[04/08/21]seed@VM:~/Desktop$ ent output.bin
Entropy = 7.999850 bits per byte.

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

Chi square distribution for 1048576 samples is 217.99, and randomly
would exceed this value 95.50 percent of the times.

Arithmetic mean value of data bytes is 127.5216 (127.5 = random).
Monte Carlo value for Pi is 3.141300740 (error 0.01 percent).
Serial correlation coefficient is -0.000030 (totally uncorrelated = 0.0).
```


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

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

結果：

```
[04/08/21]seed@VM:~/Desktop$ sudo nano task5.c
[04/08/21]seed@VM:~/Desktop$ gcc task5.c
[04/08/21]seed@VM:~/Desktop$ ./a.out
3a5bb159565f08f1feaaa89a69dd8a4f8865b69bf73bc27a4ebc131acb3039dc
[04/08/21]seed@VM:~/Desktop$ █
```