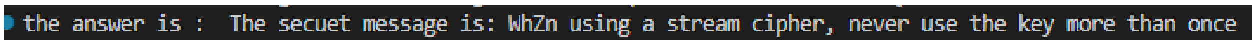
Cryptography Engineering Midterm

110550126 曾家祐

Question 1 :



```
the answer is :  The secuet message is: WhZn using a stream cipher, never use the key more than once
```

The output of my python program is

“The secuet message is: WhZn using a stream cipher, never use the key more than once”

The true plaintext answer may be

“The secret message is: When using a stream cipher, never use the key more than once”

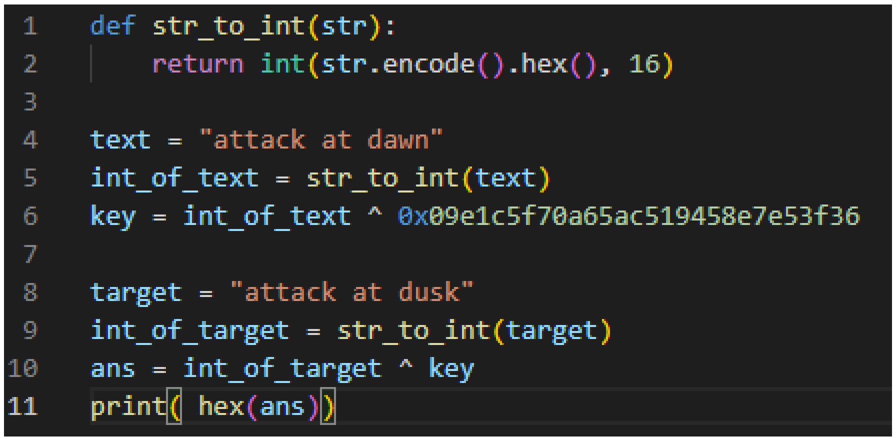
Question 2 :

First we use the plaintext and ciphertext by xor to calculate the key.

From the key we can get the ciphertext of pur plaintext.

(we need to convert the string into ascii code form)

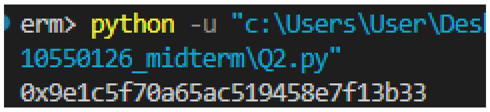
Ans: 9e1c5f70a65ac519458e7f13b33



```
def str_to_int(str):
    return int(str.encode().hex(), 16)

text = "attack at dawn"
int_of_text = str_to_int(text)
key = int_of_text ^ 0x09e1c5f70a65ac519458e7e53f36

target = "attack at dusk"
int_of_target = str_to_int(target)
ans = int_of_target ^ key
print( hex(ans))
```



```
erm> python -u "c:\Users\User\Des
10550126_midterm\Q2.py"
0x9e1c5f70a65ac519458e7f13b33
```

Question 3 :

Ans: (C) 11, (E) 6, (G) 26, (H) 1

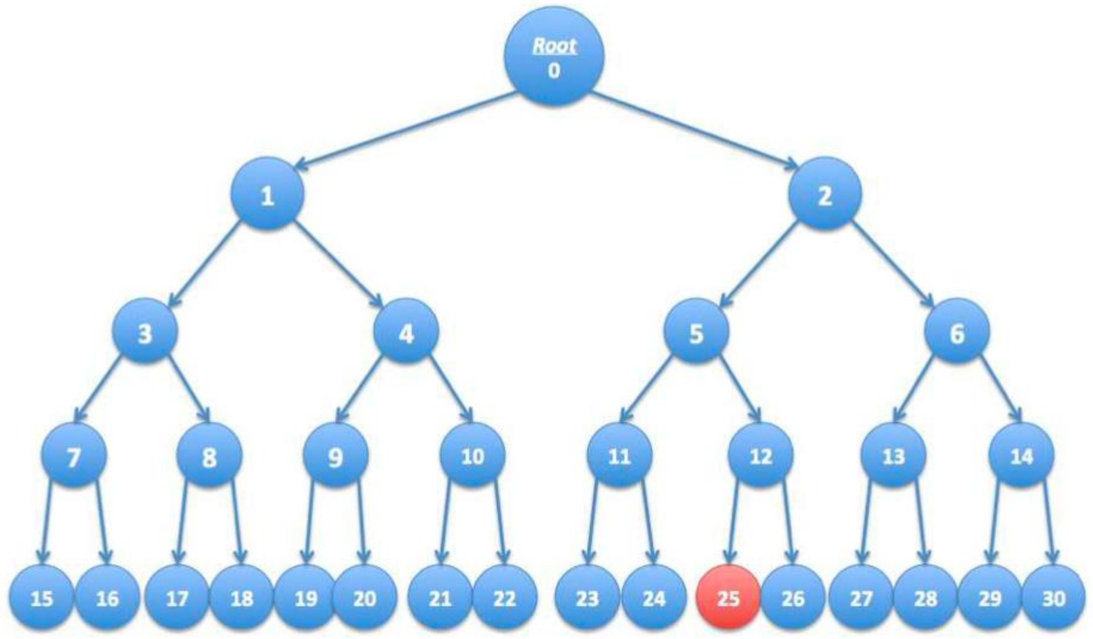
Since key 25 is right of the key 0 → we can safely include all elements under left of key 0 (key 1)

Since key 25 is left of the key 2 → we can safely include all elements under right of key 2 (key 6)

Since key 25 is right of the key 5 → we can safely include all elements under left of key 5 (key 11)

Since key 25 is left of the key 12 → we can safely include all elements under right of key 12 (key 26)

➔ Ans is 1, 6, 11, 26



Question 4 :

Ans: (C) log_2n

Since the key need to be encrypted under the node on the path from node to the target, and the length of the path is log_2n.

Question 5 :

1. First, start from c = mk mod p we get mk-c = pX

   ➔mk = pX+c -> k = (px+c)/m

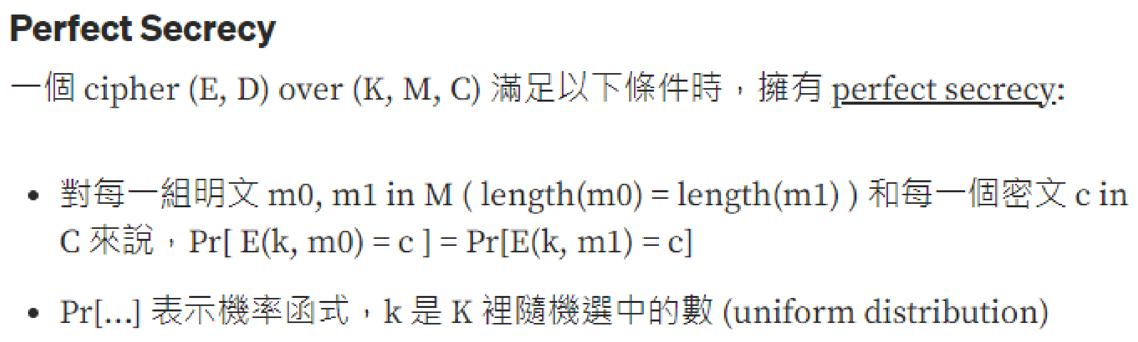
   We have Pr[E(k,m) = c] = 1

   From m0 m1

   We have Pr[E(k, m0) = c] = Pr[E(k, m1)=c]

➔ This cipher provides perfect secrecy

2. The definition of perfect secrecy:

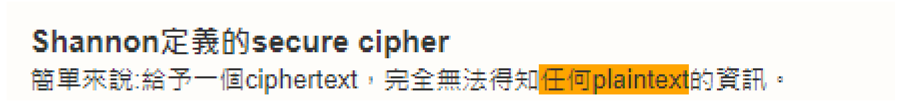


Perfect Secrecy

一個 cipher (E, D) over (K, M, C) 滿足以下條件時，擁有 perfect secrecy:

- 對每一組明文 m0, m1 in M (length(m0) = length(m1)) 和每一個密文 c in C 來說，Pr[E(k, m0) = c] = Pr[E(k, m1) = c]
- Pr[...] 表示機率函式，k 是 K 裡隨機選中的數 (uniform distribution)

The definition of semantic security



Shannon定義的secure cipher

簡單來說:給予一個ciphertext，完全無法得知任何plaintext的資訊。

For one-time pad:

E(k, m) = k XOR m = c => c XOR m = k XOR m XOR m = k , k=1.

We can know that it is one-to-one. So we can prove that OTP is perfect secrecy

For Semantic Secure:

Since we can use random generate key and we can not get the plaintext information from ciphertext.

3. The key is generated by random, and the key will only be used once.

So there is no relation to statistical analysis

4. No, if we have a public key, the attacker can try to encrypt the message by the public key. The attacker can try from 1-bit,2-bit … and looking for a matching ciphertext to encrypt the message.

Question 6 :

1. From Xi = aX(i-1) +b mod p we can get

   X2-X1 = a(X1-X0) mod p

   a = (X2-X1)/(X1-X0 mod p

   b = X1 – aX0 mod p

   so we found the definition formula and may predict the remaining sequence

2. It is not secure to use the congruential generator as the keystream generator for a stream cipher, since if the attacker know about some sequential part, the attacker may calculate the part after it.
3. Since attacker already know a ,b and p from Xi = aX(i-1) +b, we only need one bit we can calculate the rest of all
4. The attacker need three bit to form

   X1 = aX0 – b mod p

   X2 = aX1 – b mod p

   And use this two to calculate a and b, and can use a,b ,p to calculate the rest.

Question 7 :

Ans: (D) $\boldsymbol{\varphi(N) = (p-1)(q-1)(r-1)}$

If N = p_1 * p_2 *…* p_k, $\boldsymbol{\varphi(N)}$ = (p_1 -1) * (p_2 -1)*…*(p_k -1)

Question 8 :

N = 105 = 2*5*7 → φ (N) = (3-1) x (5-1) x (7-1) = 2 x 4 x 6 = 48

To find d, we can start from 13 since 13 is coprime to 48.

We can use Extended Euclidean Algorithm to find the key.

48 = 3*13+9

13 = 1*9+4

9 = 2*4 +1

$\rightarrow$

1 = 9-2*4

1 = 9-2*(13-1*9)

1 = 3*9 – 2*13

1 = 3*(48-3*13) – 2*13

1 = 3*48 – 11*13

$\rightarrow$

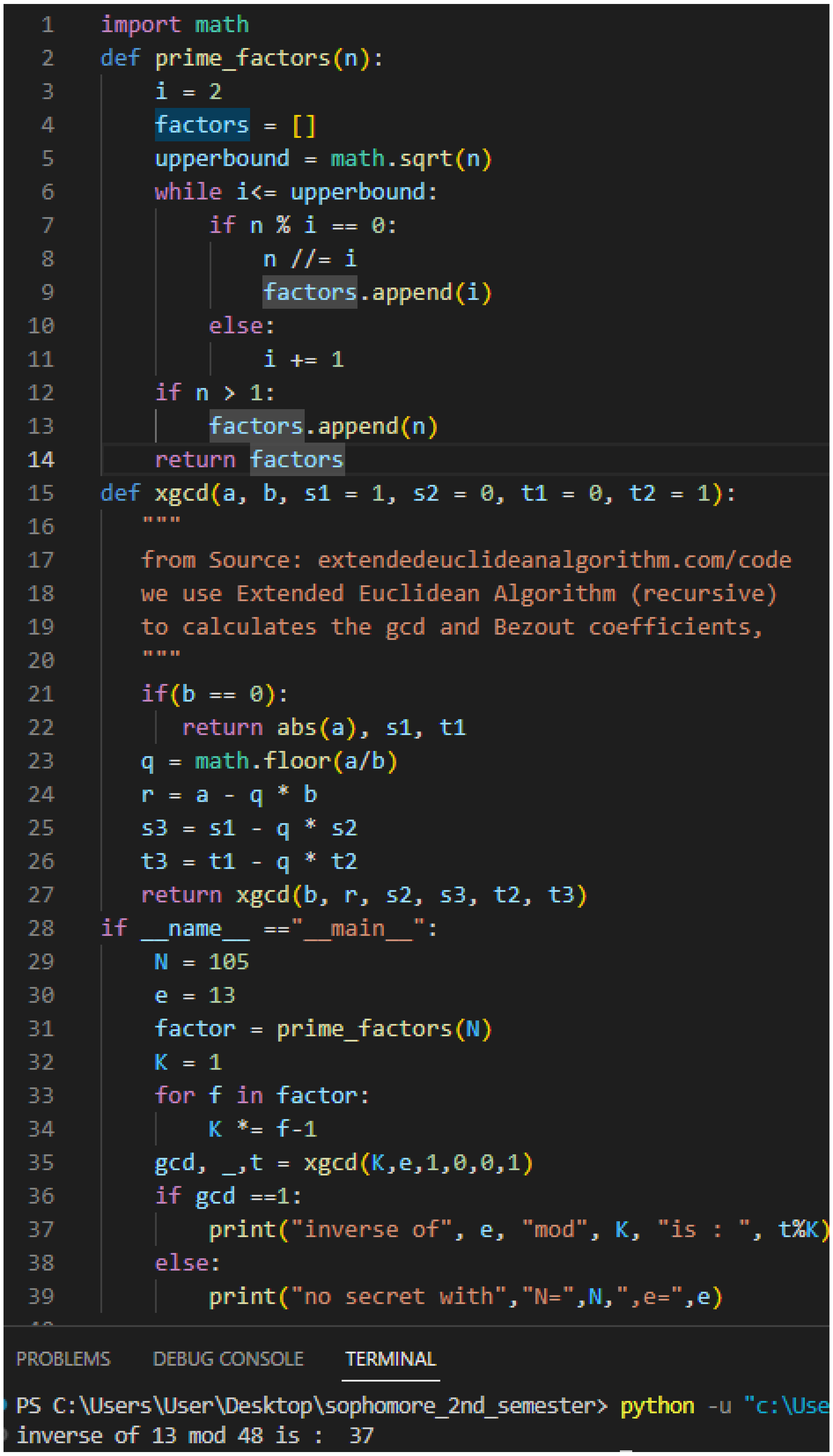
d = -11 mod(48)

-11 +48 =37(add 48 to make it positive)

Ans : the key is 37

(I also use Extend Euclidean Algorithm to write a program to solve this problem)

Below is the code.



```python
import math
def prime_factors(n):
    i = 2
    factors = []
    upperbound = math.sqrt(n)
    while i<= upperbound:
        if n % i == 0:
            n //= i
            factors.append(i)
        else:
            i += 1
    if n > 1:
        factors.append(n)
    return factors
def xgcd(a, b, s1 = 1, s2 = 0, t1 = 0, t2 = 1):
   """
   from Source: extendedeuclideanalgorithm.com/code
   we use Extended Euclidean Algorithm (recursive)
   to calculates the gcd and Bezout coefficients,
   """
   if(b == 0):
      return abs(a), s1, t1
   q = math.floor(a/b)
   r = a - q * b
   s3 = s1 - q * s2
   t3 = t1 - q * t2
   return xgcd(b, r, s2, s3, t2, t3)
if __name__ =="__main__":
    N = 105
    e = 13
    factor = prime_factors(N)
    K = 1
    for f in factor:
        K *= f-1
    gcd, _,t = xgcd(K,e,1,0,0,1)
    if gcd ==1:
        print("inverse of", e, "mod", K, "is : ", t%K)
    else:
        print("no secret with","N=",N,",e=",e)
```

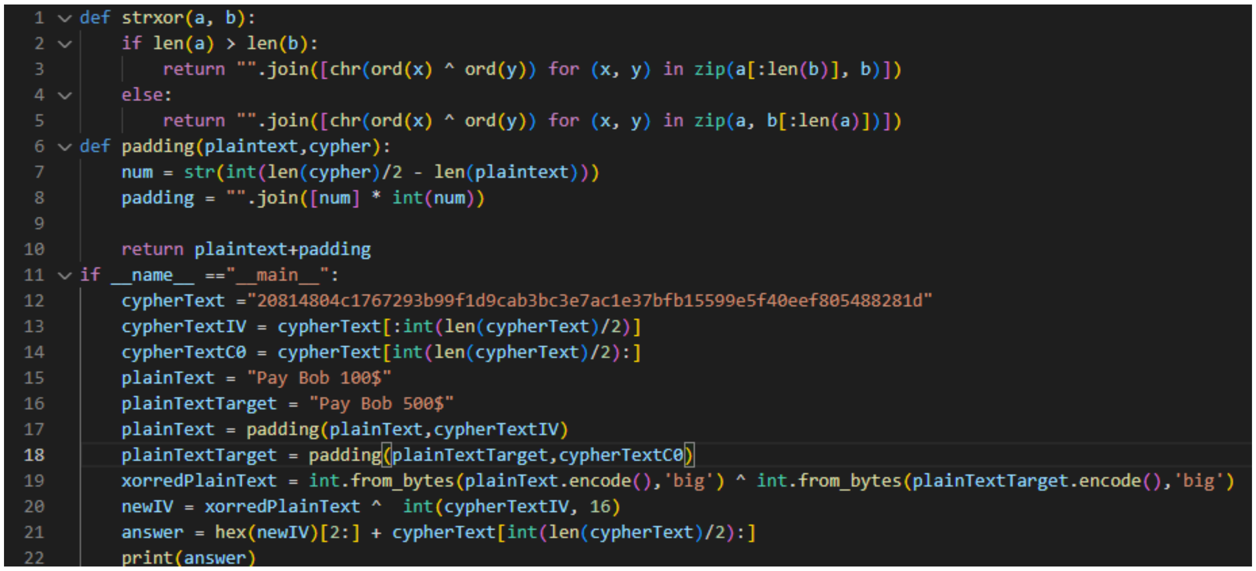
PROBLEMS DEBUG CONSOLE TERMINAL

```
PS C:\Users\User\Desktop\sophomore_2nd_semester> python -u "c:\Use
inverse of 13 mod 48 is :  37
```

Question 9 :

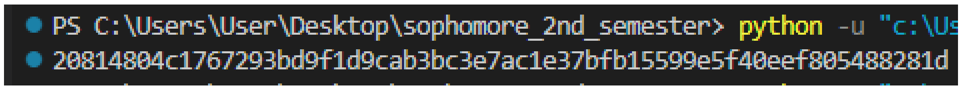
Ans:

20814804c1767293bd9f1d9cab3bc3e7ac1e37bfb15599e5f40eef805488281d



```python
def strxor(a, b):
    if len(a) > len(b):
        return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a[:len(b)], b)])
    else:
        return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a, b[:len(a)])])
def padding(plaintext,cypher):
    num = str(int(len(cypher)/2 - len(plaintext)))
    padding = "".join([num] * int(num))

    return plaintext+padding
if __name__ =="__main__":
    cypherText ="20814804c1767293b99f1d9cab3bc3e7ac1e37bfb15599e5f40eef805488281d"
    cypherTextIV = cypherText[:int(len(cypherText)/2)]
    cypherTextC0 = cypherText[int(len(cypherText)/2):]
    plainText = "Pay Bob 100$"
    plainTextTarget = "Pay Bob 500$"
    plainText = padding(plainText,cypherTextIV)
    plainTextTarget = padding(plainTextTarget,cypherTextC0)
    xorredPlainText = int.from_bytes(plainText.encode(),'big') ^ int.from_bytes(plainTextTarget.encode(),'big')
    newIV = xorredPlainText ^  int(cypherTextIV, 16)
    answer = hex(newIV)[2:] + cypherText[int(len(cypherText)/2):]
    print(answer)
```



```
PS C:\Users\User\Desktop\sophomore_2nd_semester> python -u "c:\Us
20814804c1767293bd9f1d9cab3bc3e7ac1e37bfb15599e5f40eef805488281d
```

Question 10 :

Ans: (A) $f(g^x, g^y) = g^{2xy}$

(C) $f(g^x, g^y) = \sqrt{g^{xy}}$

These two is hard to compute

(B) $(g^2)^{x+y} = g^{2x} * g^{2y}$

(D)$\boldsymbol{g^{x-y} = g^x/g^y}$

B and D can be easy to solve.