資訊安全導論 HW4

引用：這次的 RSA 只有 import random 來產生隨機大數

流程：

首先會呼叫 myrandom
接著進行 miller-rabin,
驗證是否為質數，
然後將 p,q 生成公私鑰
接著將原文及公鑰加密然
後將密文及私鑰解密

```
if __name__ == '__main__':
#step 1：產出p,q
    miller_p=1#檢查是否符合miller-rabin
    miller_q=1#檢查是否符合miller-rabin
    p=myrandom(1024)#產生一個不被1000內質數整除的隨機數節省時間
    q=myrandom(1024)#產生一個不被1000內質數整除的隨機數節省時間
    while(miller_p):
        if miller_rabin(p):
            miller_p=0
        else: #是合數
            p=myrandom(1024)#重新產生一個不被1000內質數整除的隨機數
    while(miller_q):
        if miller_rabin(q):
            miller_q=0
        else: #是合數
            q=myrandom(1024)#重新產生一個不被1000內質數整除的隨機數
#step 2：產出公私鑰
    print("p為：",p)
    print("q為：",q)
    public,private=gen_key(p,q)
    m=135620532045761028874519896765764416637997218983980438907459156
    cipher = encrypt(m, public)
    print("cipher為：",cipher)
    plain = decrypt(cipher, private)
    print("plain為：",plain)
```

函式：

myrandom：

收到 n 後生成 n 位長度的隨機數，一開始建立一個陣列去紀錄 1000 內的質數，生成後進行驗證是否不被 1000 內的質數整除，如果有則重新產生，**產生的方法為頭尾補 1：確保為 1024 位數且為奇數**，中間則呼叫 random 函式隨機產生。

```
#產生一個1024bit的隨機數頭尾皆為1
def myrandom(n):
    #<1000的質數
    prime_lis=[]
    for i in range(2,1000):
        count=0
        for j in range(2,i):
            if (i % j) == 0:
                count+=1
                break
        if count==0:
            prime_lis.append(i)
    size=(len(prime_lis))
    check=1#用來判斷是否可被1000內質數整除
    while(check):
        ary='1'
        for i in range(n-2):
            a=random.randint(0,1)
            ary+=str(a)
        ary+='1'
        put=int(ary,2)#暫放的數字
        #判斷1000內質數
        count=0
        for i in range(size):
            if (put % prime_lis[i])==0:
                break
            else:
                count+=1
        if count==size:
            return put
```

Miller-Rabin：

```python
#miller-rabin 驗證是否為質數
def miller_rabin(n):
    #n-1寫成(2^s)*d的形式
    s = 0
    d = n - 1
    while True:
        quotient, remainder = divmod(d, 2)
        if remainder == 1:
            break
        s += 1
        d = quotient
    assert(2 ** s * d == n - 1)

    #以a為底,n是否為合數
    def try_composite(a):
        if pow(a, d, n) == 1: #(a^d)%n
            return False
        for i in range(s):
            if pow(a, 2 ** i * d, n) == n - 1: #(a^((2^i)*d))%n
                return False
        return True # n為合數

    #每次取不同a,一次不符合即為合數
    for i in range(5):
        a = random.randrange(2, n)
        if try_composite(a):
            return False

    return True #很可能是質數
```

會先將 n-1 寫成(2^s)*d 的形式，接著隨機取一個 2~n 之間的 a，丟入 try 函式驗證是否是合數（驗證透過(a^d)%n==1 來判斷），總共取五輪 a

GCD：

用 gcd 找最大公因數驗證是否互質，extended gcd 則是用來找出 e^-1

```python
#extended gcd
def xgcd(a, b):
    x0, x1, y0, y1 = 0, 1, 1, 0
    while a != 0:
        q, b, a = b // a, a, b % a
        y0, y1 = y1, y0 - q * y1
        x0, x1 = x1, x0 - q * x1
    return x0
#gcd
def gcd(a,b):
    if b == 0:
        return a
    else:
        return gcd(b,a%b)
```

Gen_key：

```
#產生公私鑰
def gen_key(p, q):
    n = p * q
    phy_n = (p - 1) * (q - 1)   #phy_n
    e=0
    for i in range(2,phy_n): #找出盡量小的e(加速加密)
        e=i
        if gcd(e,phy_n)==1:
            break
    d=xgcd(e,phy_n)   #找出d
    if d<0:
        d+=phy_n
    print("n為：",n)
    print("phy_n為：",phy_n)
    print("e為：",e)
    print("d為：",d)
    # 返回：  公鑰   私鑰
    return  (n, e), (n, d)
```

接著透過這個函式產生 n,phy_n,e 及 d，根據老師的投影片內容選取一個盡量小的 e 加速加密過程，然後再透過 extended gcd 找出 d，如果 d 找出來為負的就加上 phy_n，最後返回公私鑰。

加解密：

```
#加密
def encrypt(m, pubkey):
    n = pubkey[0]
    e = pubkey[1]
    c = modExp(m, e, n)#m^e%n
    return c
# 解密 c是密文，解密為明文m
def decrypt(c, selfkey):
    n = selfkey[0]
    d = selfkey[1]
    m = modExp(c, d, n)#c^d%n
    return m
```

呼叫 modExp 去計算大數的冪模，modExp 為蒙哥馬利演算法

modExp：

```
#蒙哥馬利冪模
def int2baseBinary(x):#將數字轉成binary形式並反過來由低到高位元
    binList = []
    while x != 0:
        binList.append(x%2)
        x = x >> 1
    return binList
def modExp(a, d, n):#計算a^d%n
    res = 1
    lt = int2baseBinary(d)
    for i in lt:
        if i:
            res = (res * a) % n
        a = (a * a) % n
    return res
```

先將傳進來的指數位轉成 2 進制並反過來排列，然後依據哪一位有 1 去進行計算，**這方法中間就會取 mod 避免數字大到無法想像並且加快速度。**

結果：

```
p為： 13742899698535127849514529265372603337862009388278494922858664018707191438356055492307841346933109875946608823951761464533
382176015004783672571981436588622348277970062790106693114316350747822362079026869949004604036656408376034402437650695226097024 4
72346852883143238818223639583757340854095348943214466791559 7
q為： 11812679881027947564381859719876442705795781604526344572367243584800276912789769550036310212707175439167725479117389565177
205950178681029260390461144969033313412745029181983989104092745938309671856852262533928884499913793936619192136550918479353341 1
653513337407927581038602634758985949739019632914484788333553
n為： 16234047477587095049384132649124782392283872886536871319314311318097435440018595658449535987624258173421483642890342804517
257378729314008770063394640468140090488117100385455001623863054821058548626240973594645093525334063101215360749082194191070245 7
916507126464401470698971111632949755075255767939604218135118385908711957134059792797659781768639460096554197111931759432694182 3
568980744670143142385141776868537173637452084236560000945926140743830386185538617606399792539142923501908895573436220109774817 9
096285458588022980157774172753319050198588172184259017596373605789714650127939788046640575418284989274855687126141
phy_n為： 16234047477587095049384132649124782392283872886536871319314311318097435440018595658449535987624258173421483642890342804 0
451725737872931400877006339464046814009048811710038545500162386305482105854862624097359464509352533406310121536074908219419107 0
245791650712646440147069897111163294975507525576793960421813511583035075400082651840315876125686403509431745491662798223684193 3
831610062959884510079823358213658322205930314901508553029035779994705801745322309603604874411308484278315298182133550271234236 42
783973749413375868809479207214962678310445657203512975536121939173955035468090710075745008121713831186692822623087699 2
e為： 5
d為： 64936189910348380197536530596499129569135491546147485277257245272389741760074382633798143950497032693685934571561371218069 0
029514917256035080253578561872560361952468401541820006495452219284234194504963894378580374101336252404861442996328776764280983 1
666028505857605882795884446531799020301023071758416872540463321403016003306073612635045027456140377269819666511928947367735326 4
402518395380403192934328546332888237212596060342121161431199788232069812892384144194976523393711326119272853420108493694571135 8
949976535034752379168288598507132417826288140519021444877566958201418723628403029800324868553247467712904923507 97
cipher為： 71991746796041131792472678817900231080524825550316921864956882928848513289231876485607455940639921105614774387725212 7
511499666341020209275572079572426348370030724098513011187034497063667642957194108303244765210580839307023969992679228802041684 2
897053713271224431755366586354803364870285588826404863371362883108705736145948605727528396171017990658863756133025499847711940 5
764154442699036212298512477498199996223177919641217204633174920353607620788322531570697447853257017126399916730450613537773786 1
428519398759484142371419640980009971369232023003171354805413095259824804540501922121669247929786215496290204128889718
plain為： 13562053204576102887451989676576441663799721898398043890745915636666340666465644106859552178250486260661908656359240 5
966964024022236587593447122392540038493893121248948780525117822889230574978651418075403357439692743398250207060920929117606033 4
905591595609877687683248230115792832233929644543990454267563768
```

測試時將要加密的訊息 m 給定為一個很大的數字，並經過驗證解密完得到的 plain 與 m 相同，上圖為流程中產生的所有數字。

建置環境：

使用 jupyter 撰寫程式碼，操作上都直接執行整個檔案

遇到困難點：

一開始隨機產生使用 os.urandom，後來在型別的轉換上遇到困難，**改成一個一個 bit 產生**。

產出後發現 miller-rabin 執行速度過慢，在網路上尋找後改成產生時先用 1000 內質數去驗證，因此加快 miller-rabin 的速度。

最後則是在網路上尋找蒙哥馬利演算法時，**將轉成 2 進制的函式改成內建的 bin()，後來發現兩種方法在排列上完全相反，才再重寫一個轉換函式。**

心得：

這次的作業將課堂上理解的 RSA 演算法實作出來，但在過程中也遇到了幾個上課時沒有考慮到的問題，在完成後也了解到為什麼 RSA 的安全可以得到保證，因為數字實在大到質因數分解接近不可能，對於安全機制上有了更深入的了解。