

# Hw4

## 100分標準\_\_檔名：

rsa\_1024.py

## 建置環境

mac + vscode + python3

## 操作方法：

python3 rsa\_1024.py [plaintext]

## 會輸出：

p,q,n,e,d,ciphertext 和plaintext

## 輸出結果圖：

```
Last login: Wed Dec 4 17:17:56 on tty500
~ - INSERT cd Desktop/Information_Security_Class/hw4/B10615056 黃暉翔
p = 155707699198489935431296396536803232458443824075864740512739854668433485612723830111633916955448215434586352215530950880423860481501455543988902346652105596647059685284385174336651157226153664886020499126567826105498627197932614392
8920300712842465162160183541949559493659666871846911529729711887621196058181
q = 102866872643219310726244626418907795872054161561933595933542769422435367258358294195948991042597290567780700891198002995483353893616850586039533427357754992430127242108626981763482747913714560042487270380214597755314215684285338529
789906970867051062351590480124696177256416612429283318118072050757495220097
n = 160171648630197657245215603616306659841317411850377420470659774372114486521285410327225394786413066281470972082936970962981520457997274406322561015057230097689518689337291893753789335359677582932839929896733129854004685790630260964642
33116214521712650181042577746204985893849628658618888128442526770110533691755517611595258952067343171127024087457217183451859983702143624950391796806067003619994655588671878391880725338020997045420924487509971139754688025042134485733
24845047472056392467303844164719870962473081322842832738259886414841981273190494402717090637979611580272070523831486381781447735909018297767272267
e = 6881
d = 7695501292303934063352460188279462087752599740931223961052349087953857352037431556722446615881644692129491891613083273305760746132668053062733843615792806172574513107232961415531961486839251818004371804388496806480876651393173592657
38529581759601833360971824064049440977338321844055521603251780170662304059773428581187680797365266889001068686579871728004250802736740229038374666190115370836421616149139731806117199823750568322166706660453658126787525199099068870218361
7694911079619712248000959112117071552210590270216344368605879406246080476844080924381090861124645746609044943340621199132002405229097606483601
ciphertext = [1493185617890636863991634301905010141956389077508825413571557369103378940752484884746289458420585039852543968317493770980990666277963998985969382388010582541092159800424667436739914175795884083117962873729087846160185960119
909136225279691919201314803581844473869783121828546250995190833489070666384233788569952176168311448322944910655210660652953446234705003011672471737322915365803430579335362394764290951627658560803581942053758595978004286567739181625945
2228402114452464542013140892560845970927937647510169124710632070002806145523786056068121061310189735886218631188093775063608797014592206148025942322053785997, 71910374923164373855566311135985471796469995399340936542091051709768333506854241
5279476927378032203795449639950480448989605296348197979925729097069152724632434413571394992402407364892367240440744656131617661275172187046984852867129806564103781256384448675230213668616310979660738250945346495497049117230908688411601712
32071081291432940995114246183251527307857815086521958041461996110830079697410584411085390420626424879848949991165217473670918499451445860260136529392342184181614284544702108551461481407613518584863439905077621769416927208687569026746488
3950865337149998907714717414647029139516472104850834027465841, 4139082214382775803338904381437825726695450504354224420592766730178511427733280281247897682968394844002063243774027077061248880702549702594925844106990848934154944754214217
793104528315634109665162222084059740914081207470503126845032470695040436037712022421416933100059976237521150440803166278421930156950476594535975150829902996714505908210597083565810438219264917226167418024892995572944472578012053
35634620231416461533139050235154898427998018041623036891928490375934418140416259260123471826703581312617240897862332193069852720173628408878220594219186722415832555409191912548257650925965060831, 41390822143827758033389043814378257
266954560434522428592766730117851142773328028124789768296839484400206324377402707706124888070254970259492584410699084893415494475421421779310452831563410966516222208405974091408120747050312684503247069504043603771202242141693318
0059976223752115040408816627942793015695044765945435975150825990299671450598821058783556581043821926491723616741892480929955729444725870012053356346202314164615383139050236515489842479080180416230368919284903759344181404182592660123471826
7035813126172408978623321936685272817362840887820594219186722415832555409191912548257650925965060831, 13805671513796739934559337276086835848840650923258096254302891181950136819148127271613672487370331501904723978395754370160234089593
5225003954660879236394021695058932292818101071022962160803641165648334753065283086762990784140018366245849584755118331839075164296274881565653962677765266715139760262570890455607596180827820766115205553957356015763117833162556175642244
37813764115677462625477306178916585958793387710968204719616151980590508215661774940627667688082929428317311936685360210209783805612031020579752138033141062362483619171040969407385829388176385188579139912497897365447513692961546580
58952563, 3086782812933107015508114836023616668041547209568031075117748733475266967067656410456116831009749279140904975753811891848701514025886301655462157191522359962144016024543876197959344709197887365320136469134429329837
71708399724505234339439696605119571598740835229299862161970212607763020731340715611023138239038013592716647430957831877629373583704639112720738308179835857273375614119744581466694443874254315233748415956103866577547822128546391789
571896403048027491581966252876900946804393549024179708639064018391889458452069536271139929165580550178378984805185122728794363969692555712109355640, 6899212191671819568094393663121659026047542052369511931093399080690186961348987
091740374985107608060593904047539436627479441406749264706625293746109285491515270071007356118835449861008419198088620225316328029247017562763934636496591457084601038939094789755666780503424314666721771855677188851735291596596252317412879598
59158366708676584212408449433914491303770993970801775267813209631813806129377039480152287495195857243653125447558534432933727768210968375128134230038884769847950399367371592662854329058680119399368920706743966648493803554330241651370054
574320878022296016054702178039237313294234083361593676, 6284622748300796818694564112827955036532728880642304818625416057203770943391940475273622695180932015902626173243678658080013075267516316096000459326268599038379415361132732759086521
2953919439641826266927820557022242408208787420850613378183263557670292854940221443958373134248501509791397730292525105466781575714831257749857576944606050477086457005981188397754666296156275999607979099128563889815679310234089996
58013888172948465883828242707956472232928208067008313825692346972114547072054313426461730484395376776417414948703626709919521486478981136096395196924375352901665649730221497530937799362591366]
```

## Code

Miller test:

```

def isprime(n):
    n = int(n,2)
    if (n == 2):
        return True
    if (n < 2):
        return False

    a = random.randint(1,n) % (n-2) + 2

    u = n-1
    t = 0
    while (u % 2 == 0):
        u >>= 1
        t+=1

    x = exp_mod(a, u, n) # x = a ^ u % n
    if (x == 1 or x == n-1):
        return True

    for i in range(t-1):

        x = exp_mod(x, 2, n); # x = x * x % n;
        if (x == 1):
            return False
        if (x == n-1):
            return True

    return False

```

Extend gcd:

```

def xgcd(a, b):
    """return (g, x, y) such that a*x + b*y = g = gcd(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 b, x0, y0

```

Modinv:

```
def modinv(a, b):
    """return x such that (x * a) % b == 1"""
    g, x, _ = xgcd(a, b)

    if g == 1:
        return x % b
    else:
        raise Exception('modular inverse does not exist')
```

如果找不到會丟exception

Square and multiply:

```
def exp_mod( a, b, n):
    """return (a ** b )%n"""
    r = int(1)
    while(b):
        if(b&1):
            r=(r*a)%n
        a=(a*a)%n
        b>>=1      # b = b>>1

    return r
```

gen\_prime:

```
def gen_prime():
    prime = ''
    for i in range(1024):
        if i == 1023 or i == 0:
            prime += '1'
        else:
            prime += random.choice(['0','1'])

    while isprime(prime) == False:
        prime = ''
        for i in range(1024):
            if i == 1023 or i == 0:
                prime += '1'
            else:
                prime += random.choice(['0','1'])
    return int(prime,2)
```

key\_generator:

```
def key_generator():
    p = gen_prime()
    q = gen_prime()
    n = p * q
    # print('p = ',p)
    # print('q = ',q)
    # print('n = ',n)
    phi_n = (p - 1) * (q - 1)
    pe = [e for e in range(10000) if hcfnaive(e, phi_n) == 1]
    e = random.choice(pe)
    while (1):
        try:
            d = modinv(e, phi_n)
            break
        except:
            p = gen_prime()
            q = gen_prime()
            n = p * q
            phi_n = (p - 1) * (q - 1)
            continue

    # print('e = ',e)
    # print('d = ',d)
    return p,q,n,e,d
```

會確保e 能產生乘法反元素d

---

心得：

花比較多時間在了解數學定理，產生質數後只要確保 e 能生成 d 並且  $e * d \% \phi_n = 1$  keygenerator就沒什麼問題了，後面只要用square and multiply 做加解密就好了。基本上只要理解數學公式程式就這沒有那麼難了。