四資工三甲 B10715029 陳彥瑋

IS hw4 說明文件

- ## 建置環境 / 依賴套件

  測試環境　　：Ubuntu 20.04.2 LTS
  Python 版本 ：3.7.10
  依賴套件　　：secrets (原生函式庫)
  　　random (原生函式庫)
  　　sys (原生函式庫)
  　　typing (原生函式庫)

- ## 操作方式

  程式執行：python3 RSA.py [--mode <mode>] [--data <data>] [--e <e>] [--d <d>] [–-n <n>] [--p <p>] [--q <q>]

  - [] 內的部分是 Optional
  - <data> 部分為需要做加密或解密的資料，未選擇時採用隨機產生的 1024 bits 的資料
  - <mode> 可填入 encrypt, decrypt, test，分別是做加密、解密、測試使用，未選擇時預設是採用 test 模式
    - 測試模式會將資料做一次加密後再解密，並確認原始資料和解密後的資料是否一致
  - <e> <d> <n> <p> <q> 是填入 RSA 的金鑰參數
  - 在加密操作時
    - 如果有給定 e 和 n 參數，則使用其來做加密
    - 如果沒有 e, n 參數但是有 p 和 q 參數，則會自動從 p, q 參數產生 e, n 參數作加密
    - 如果皆無，則自動產生。
  - 在解密操作時
    - 如果有給定 d 和 n 參數，則使用其來做解密，如果在此情況下還有提供 p 和 q 參數則可使用 CRT 做加速
    - 如果沒有 d, n 參數但是有 p 和 q 參數，則會自動從 p, q 參數產生 d, n 參數作解密
    - 如果皆無，則自動產生。
  - 在測試操作時

- 如果有給定 e 和 d 和 n 參數，則使用其來做加解密，如果在此情況下還有提供 p 和 q 參數則可使用 CRT 做解密部分的加速
- 如果沒有 e, d, n 參數但是有 p 和 q 參數，則會自動從 p, q 參數產生 e, d, n 參數作加解密
- 如果皆無，則自動產生。

- 在執行時可能會因為要產生質數的關係而花一些時間，正常來說會在 20 ~ 30s 內完成（大部分情況只需約 7 秒左右）。

- **執行結果圖**

```
ywc@DESKTOP-4CLHFUV:~/HW4_B10715029$ python3 RSA.py --mode encrypt --data 2021 --e 19403 --n 230491511
[Program Start]
224573706
[Program End]
```

^給定 e, n 的加密

```
ywc@DESKTOP-4CLHFUV:~/HW4_B10715029$ python3 RSA.py --mode encrypt --data 2021 --p 22441 --q 10271
[Program Start]
23255880
[Program End]
```

^給定 p, q 的加密

```
ywc@DESKTOP-4CLHFUV:~/HW4_B10715029$ python3 RSA.py --mode decrypt --data 224573706 --d 187486067 --n 230491511
[Program Start]
2021
[Program End]
```

^給定 d, n 的解密

```
ywc@DESKTOP-4CLHFUV:~/HW4_B10715029$ python3 RSA.py --mode decrypt --data 224573706 --p 22441 --q 10271 --d 187486067 --
n 230491511
[Program Start]
2021
[Program End]
```

^給定 d, n 和 p, q 的解密

```
ywc@DESKTOP-4CLHFUV:~/HW4_B10715029$ python3 RSA.py --mode test --data 2021 --e 19403 --d 187486067 --n 230491511
[Program Start]
Ori: 2021
Enc: 224573706
Dec: 2021
Equ: True
[Program End]
```

^給定 e, d, n 的測試模式

```
ywc@DESKTOP-4CLHFUV:~/HW4_B10715029$ python3 RSA.py --mode test --data 2021 --p 22441 --q 10271
[Program Start]
Ori: 2021
Enc: 23255880
Dec: 2021
Equ: True
[Program End]
```

^給定 p, q 的測試模式

```
ywc@DESKTOP-4CLHFUV:~/HW4_B10715029$ python3 RSA.py --mode test
[Program Start]
Ori: 113685632116232941335695264467577355431288444596176627772573825031869623500359288108400012094394184726573744401456313545553361909728178046086531450812798051664627479218368475536324390309447465864994885358419172613032722061144723077868554989741320507843529846409741747577326452597412604869247256665984797159513
Enc: 248426799099140186908390240933344831144157274839885013312336459385902287440998626723787137980212647870478632834099133135644721772715258736123476824690814823305967682739933640895362155964214128220959110937771911080974714923242688238098916570895097573094140218201915452091013485896616328070267688331845528777114786509432717888765195801541020634484864918001769104592500498178991981426954360671134617428146109398188997788849515596807593462105849385806850262773377494197348419809694983722122974225414652651690426170396382481427360825421472393168825826847319865690239966742082613620272404498169752803853128962277053477152
Dec: 113685632116232941335695264467577355431288444596176627772573825031869623500359288108400012094394184726573744401456313545553361909728178046086531450812798051664627479218368475536324390309447465864994885358419172613032722061144723077868554989741320507843529846409741747577326452597412604869247256665984797159513
Equ: True
[Program End]
```

^不給定資料的測試模式

- **程式碼解說**

```
129 def main():
130     print("[Program Start]")
131 
132     data = RandomNumberGenerator(1024)
133     mode = "test"
134     p = -1
135     q = -1
136     n = -1
137     e = -1
138     d = -1
139     for i in range(1, len(sys.argv)):
140         if (sys.argv[i] == '--mode'):
141             i = i + 1
142             if (sys.argv[i] in ["encrypt", "decrypt", "test"]):
143                 mode = sys.argv[i]
144             else:
145                 print("Input mode is not support. Use default.")
146         elif (sys.argv[i] == '--data'):
147             i = i + 1
148             if ((sys.argv[i]).isdigit()):
149                 data = int(sys.argv[i])
150             else:
151                 print("Input data is not a number. Use default.")
152         elif (sys.argv[i] == '--p'):
153             i = i + 1
154             if ((sys.argv[i]).isdigit()):
155                 p = int(sys.argv[i])
156             else:
157                 print("Input p is not a number. Use default.")
158         elif (sys.argv[i] == '--q'):
159             i = i + 1
160             if ((sys.argv[i]).isdigit()):
161                 q = int(sys.argv[i])
162             else:
163                 print("Input q is not a number. Use default.")
164         elif (sys.argv[i] == '--n'):
165             i = i + 1
166             if ((sys.argv[i]).isdigit()):
```

```
167             n = int(sys.argv[i])
168           else:
169             print("Input n is not a number. Use default.")
170         elif (sys.argv[i] == '--e'):
171           i = i + 1
172           if ((sys.argv[i]).isdigit()):
173             e = int(sys.argv[i])
174           else:
175             print("Input e is not a number. Use default.")
176         elif (sys.argv[i] == '--d'):
177           i = i + 1
178           if ((sys.argv[i]).isdigit()):
179             d = int(sys.argv[i])
180           else:
181             print("Input d is not a number. Use default.")
182
183       #e, d, n, p, q = RSA_init()
184
185       if (mode == "encrypt"):
186         if (e == -1 or n == -1):
187           if (p == -1 or q == -1): # no input any key
188             e, _d, n, _p, _q = RSA_init()
189           else: # has p, q
190             e, _d, n = RSA_keyGen(p, q)
191
192         print(RSA_enc(data, e, n))
193       elif (mode == "decrypt"):
194         if (d == -1 or n == -1):
195           if (p == -1 or q == -1): # no input any key
196             _e, d, n, p, q = RSA_init()
197           else: # has p, q
198             _e, d, n = RSA_keyGen(p, q)
199         else:
200           if (p == -1 or q == -1): # has d, n no p, q
201             p = -1
202             q = -1
203
204         print(RSA_dec(data, d, n, p, q))
205       elif (mode == "test"):
206         if (e == -1 or d == -1 or n == -1):
207           if (p == -1 or q == -1): # no input any key
208             e, d, n, p, q = RSA_init()
209           else:  # has p, q
210             e, d, n = RSA_keyGen(p, q)
211         else:
212           if (p == -1 or q == -1): # has e, d, n no p, q
213             p = -1
214             q = -1
215
216         cipher = RSA_enc(data, e, n)
217         plain = RSA_dec(cipher, d, n, p, q)
218         print("Ori:", data)
219         print("Enc:", cipher)
220         print("Dec:", plain)
221         print("Equ:", data == plain)
222
223       print("[Program End]")
224
225   if __name__ == "__main__":
226       main()
```

main 為主要函式，在程式被呼叫時會啟動（第 225 ~ 226 行）。

在 main 函式中的第 132 ~ 138 行是設定參數的初始值。data 預設是一個隨機的 1024 bits 的資料，mode 預設是測試模式，p, q, n, e, d 預設都是 -1 代表無資料。

第 139 ~ 181 行為讀取使用者設定的參數並進行對應的檢查。

第 185 ~ 192 行是加密的部分，首先會先判斷使用者有提供哪些的參數，並根據一些規則及對應的處理方式後得到 e, n 參數，並使用這二個參數對資料做加密。

第 193 ~ 204 行是解密的部分，跟加密的部分很類似，都是根據一些規則及對應的處理方式後得到 d, n 參數，並使用這二個參數對資料做解密。而對於是否有提供 p, q 參數的部分以用來做 CRT 的加速的判斷則是在 RSA_dec 函數中做判斷。

第 205 ~ 221 行是解密的部分，跟加解密的部分很類似，都是根據一些規則及對應的處理方式後得到 e, d, n 參數甚至是 p, q 參數，並使用這些參數對資料先做加密後再做解密，並在第 221 行時比對原資料和解密後資料是否一致。

```
102 def RSA_init() -> List[int]:
103     random.seed()
104     #p = 22441
105     #q = 10271
106     p = primeGen(1024)
107     q = primeGen(1024)
108     e, d, n = RSA_keyGen(p, q)
109 
110     return [e, d, n, p, q]
111 def RSA_keyGen(p:int, q:int) -> List[int]:
112     n = p * q
113     phi = (p-1) * (q-1)
114     e = get_e(phi)
115     #e = 19403
116     d = inv(e, phi)
117     return [e, d, n]
```

RSA_init 和 RSA_keyGen 是用來產生 RSA 金鑰的函式。

首先在 RSA_init 中，會先把 random 的 seed 初始化，並且使用 primeGen 函式產生 p, q 所需要的 1024 bits 的質數，而 e, d, n 參數則是透過 p, q 參數和 RSA_keyGen 函式產生。

在 RSA_keyGen 函式中，會先算出 n 和 $\Phi(n)$ 參數，並且使用 $\Phi(n)$ 參數來找出合理的 e 參數，接著有了 e 和 $\Phi(n)$ 參數之後就可以找出在 $mod\ \Phi(n)$ 情況下 e 的反元素 d。

```python
def RandomNumberGenerator(k:int=1024) -> int:
    return secrets.randbits(k)
def primeTest(n:int, witness:int=3) -> bool:
    if (n%2 == 0):
        return (n==2) # even is prime iff n == 2
    elif (n == 1): # probably bug in miller rabin test
        return False
    elif (n == 3): # probably bug in miller rabin test
        return True
    else:
        # miller rabin test
        p_temp = n - 1
        r = 0
        while (p_temp%2 == 0):
            r += 1
            p_temp //= 2
        d = p_temp

        for w in range(witness):
            a = random.randint(2, n-2)
            x = SquareAndMultiply(a, d, n)
            if (x != 1 and x != (n-1)):
                for i in range(r-1):
                    x = (x * x) % n
                    if (x == (n-1)):
                        break # witness pass

                if (x != (n-1)):
                    return False

        return True # probable prime

def primeGen(k:int=1024) -> int:
    n = RandomNumberGenerator(k)
    while (primeTest(n) == False):
        n = RandomNumberGenerator(k)
    return n
```

primeGen 函式是一個用來產生質數的函式，首先會先產生一個隨機數，並且檢查此數是否是一個質數，不是的話則一直產生隨機數直到符合是質數。

RandomNumGenerator 是用來產生隨機數的函式，主要是使用 secrets.randbits 這個函式來產生 k bits 的隨機數。

primeTest 則是用來檢查數字是否是質數的函式，首先會先檢查一些簡單的條件來做判斷，如果無法判斷則使用 Miller-Rabin 方法來做測試，並且使用 witness 次來找不同的證據做檢驗。

```python
def exGCD(a:int, b:int) -> List[int]:
    x0, y0 = 1, 0
    x1, y1 = 0, 1

    while (b != 0):
        q = a // b
        x0, x1 = x1, x0-q*x1
        y0, y1 = y1, y0-q*y1

        a, b = b, a%b
    return [a, x0, y0]
def gcd(a:int, b:int) -> int:
    return exGCD(a, b)[0]

def get_e(phi:int) -> int:
    for e in range(2, phi):
        if (gcd(e, phi) == 1):
            return e
    return 1
def inv(a:int, mod:int) -> int:
    # ax + by = gcd(a,b)
    # ax = (-y)b + gcd(a,b)
    # ed = (-k)phi + gcd(e, phi)
    # ed = (-k)phi + 1
    # ed (%phi) = 1
    g, x, y = exGCD(a, mod)

    assert(g == 1)
    if(x < 0):
        x += mod
    return x
```

get_e 是用來找 e 參數的函式，找法是從 2 開始一直向上找與Φ(n) 的 GCD 為 1 的數，這樣的找法能保證找到的 e 一定是最小的，就能保證安全性。

inv 是用來計算在 mod 下與 a 成反元素的值，推導方式如註解，主要是要使用 extend GCD 來求得 x 值，x 值即是反元素。

gcd 函式是用來找二元素的 GCD，是直接使用 exGCD 來取得。

exGCD 是 extend GCD 的函式，會回傳 GCD, x, y 參數。

```python
def RSA_enc(plain:int, e:int, n:int) -> int:
    return SquareAndMultiply(plain, e, n)
def RSA_dec(cipher:int, d:int, n:int, p:int=-1, q:int=-1) -> int:
    if (p == -1 or q == -1):
        return SquareAndMultiply(cipher, d, n)
    else:
        return CRT(cipher, d, n, p, q)
```

RSA_enc 是做 RSA 加密部分的函式，會使用 square and multiply 方式來做 power 的運算。

RSA_dec 是做 RSA 解密的部分。在如果有提供 p, q 參數的情況下，會使用 CRT 的方式做加速；如果沒有，則是使用一般的 square and multiply 方式做解密。

```python
def SquareAndMultiply(x:int, h:int, n:int) -> int:
    h_bin = bin(h)[2:]
    result = 1
    for b in h_bin:
        result = (result ** 2) % n
        if (b == '1'):
            result = (result * x) % n

    return result


def CRT(x:int, d:int, n:int, p:int, q:int) -> int:
    xp = x % p
    xq = x % q

    yp = SquareAndMultiply(xp, d%(p-1), p)
    yq = SquareAndMultiply(xq, d%(q-1), q)

    cp = inv(q, p)
    cq = inv(p, q)
    y = ((q * cp) * yp + (p * cq) * yq) % n
    return y
```

SquareAndMultiply 函式是做 square and multiply 方式的 power 計算。首先會先將 h 次方轉換成二進位，並且每次讀一個位數。每次讀一位數時會將暫存值做平方運算，如果讀的位數是 1 的話還會額外把 x 乘進去暫存值。全部計算完之後的暫存值就是結果。

CRT 函式是做 RSA 解密時的 CRT 計算。首先第一步先算出 xp 和 xq，第二步是分別對 xp 和 xq 計算得出 yp 和 yq，最後一部是計算出 cp 和 cq，並且透過一些計算得出結果 y。

- **遇到困難與心得**

困難：

無

心得：

寫起來很順，沒有遇到太大的困難。唯一一個比較有卡住的部分是質數驗證的部分，老師的投影片和 Wikipedia 的 pseudo code 有一點不太一樣，花了一點時間仔細看過之後才發現投影片的部分只使用一個 witness 但 wiki 則是有擴充至 k 個，這是一個我有稍微花一點時間的部分。

另外我在找 Miller-Rabin Test 的資料時，有看到另外一個用來測試質數且有意思的方法 — Lucas–Lehmer primality test，似乎比其他的方法更有效率，感覺是一個可以改良的方向。