資訊安全導論 HW4 - RSA

B10615020 林哲旭

一、 建置環境/依賴套件

建置環境:

- WSL-Ubuntu-18.04 (主機環境)
- Python 3.6.9 (語言環境)
- vim (文字編輯器)
- pip (套件安裝)

依賴套件:

- secrets 套件: 主要負責產生隨機整數值。
  - import secrets
- sys 套件:主要負責檔案讀寫。
  - import sys

二、 操作方式

Makefile 總共分為三個部分，make i、make r、make clean

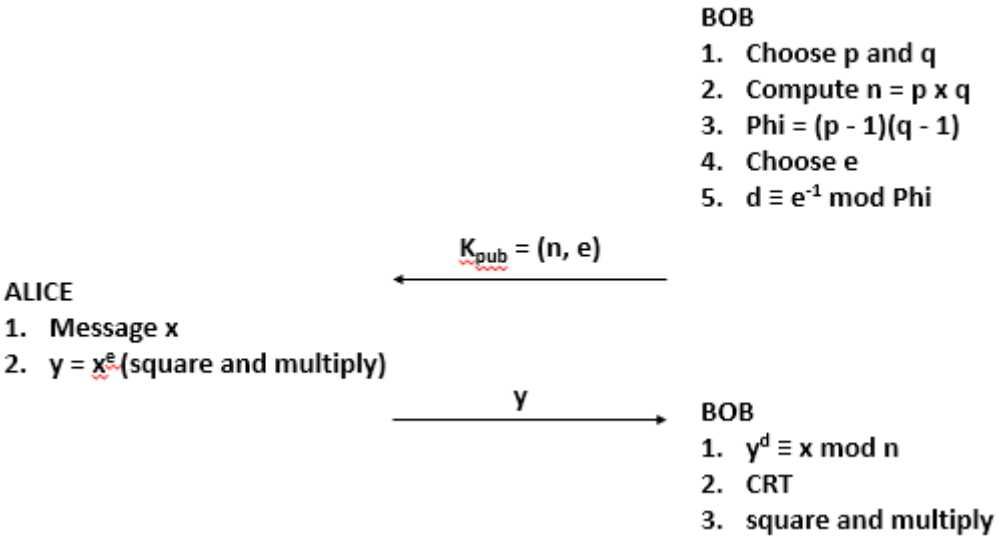
- CC = python3
- make i (執行 RSA 加解密，由使用者輸入 p, q, plaintext)
  - $(CC) RSA.py -i
- make r (執行 RSA 加解密，p, q 由程式自動產生，由使用者輸入 plaintext)
  - $(CC) RSA.py
- make clean (清除執行後產生的結果)
  - rm -rf report.txt

三、 程式碼解說

- def gcd(m,n)
  - 找 m 和 n 的最大公因數回傳
- def ext_euclid(a, b)
  - 擴展歐幾里得算法(Extended Euclidean algorithm)
  - 用來尋找 modular 反元素
- def minv(a,b)
  - 在 ext_euclid(a,b)之後做餘 1 判斷
    - YES, 回傳反元素值
    - NO, raise error
- def sm(x,H,n)
  - Square and Multiply
  - x 為底數，H 為指數，n 為 modular 數
  - 首先產生 h 為 H 的 binary string
  - 每次 iteration 都先平方，再檢查 h[i]是否為 1
    - YES, multiply x
    - NO, do noting
- def FermatTest()
  - 首先用 secrets 套件先產生 512 bit 的整數亂數
  - 接著跑 100 次 iteration
    - 產生 a 為{1 ~ p-2}之間的亂數
    - 用 sm(x,H,n)檢查是否為質數
      - YES, 回傳 prime 和 “composite”
      - NO, 回傳 prime 和 “prime”
- def LargePrime()
  - 首先建立 flag = “composite”
  - 接著跑 FermateTest()直到找到質數為止
  - 回傳找到的質數
- 開啟檔案
  - fout = open("./report.txt","w")
- 選擇是否讓使用者自行輸入 p, q
  - if(len(sys.argv) > 1 and sys.argv[1] == "-i"):
  - p = int(input("請輸入 p 值: "))
  - q = int(input("請輸入 q 值: "))
  - else:
  - p = LargePrime()
  - q = LargePrime()

- 產生 n, phi, e, d
  - n = p * q
  - phi = (p - 1) * (q - 1)
  - e = 0
  - for i in range(2,phi):
  - if gcd(i,phi) == 1:
  - e = i
  - Break
  - d = minv(e,phi)
- 寫入目前已經產生的資料
  - fout.write("p 值: " + str(p) + "\n")
  - fout.write("q 值: " + str(q) + "\n")
  - fout.write("n 值: " + str(n) + "\n")
  - fout.write("phi 值: " + str(phi) + "\n")
  - fout.write("e 值: " + str(e) + "\n")
  - fout.write("d 值: " + str(d) + "\n")
- 中國餘式定理以及 RSA 加解密
  - 首先產生 dp, dq, qinv
    - dp = d mod (p-1)
    - dq = d mod (q-1)
    - qinv = q 的 modular 反元素
  - 讀入 message(plaintext)
  - 接著對 message 加密產生 ciphertext($message^e$ mod n)
  - 再來產生 m1, m2, h
    - m1 = $cipertext^{dp}$ mod p
    - m2 = $cipertext^{dq}$ mod q
    - h = qinv * (m1 – m2) mod p
  - 最後由上述的參數組合出解密結果
    - plaintext = m2 + h * q
  - 寫入 cipertext 和 plaintext
    - fout.write("Ciphertext= " + str(ciphertext) + "\n")
    - fout.write("Plaintext= " + str(plaintext) + "\n")
- 關閉檔案
  - fout.close()

四、 Custom Block Cipher Operation 架構



加解密架構

五、 實驗結果

- Indivial 1:手動輸入 p = 3, q = 11, plaintext = 4

```
p值: 3
q值: 11
n值: 33
phi值: 20
e值: 3
d值: 7
Ciphertext= 31
Plaintext= 4
```

- Indivial 2:手動輸入 p = 22441, q = 10271, plaintext = 2021

```
p值: 22441
q值: 10271
n值: 230491511
phi值: 230458800
e值: 7
d值: 131690743
Ciphertext= 23255880
Plaintext= 2021
```

- Indivial 3:自動產生 p, q, plaintext = 2021

```
p值: 5854748639139428914416337955021825856042275954113175256389722769790952266841534201311726228553704954537397022650415966833035001994842729620184606338954610464470143885124355195
6088473705490932185085517174428421669667149007475326083545404493599044050753047945901650292015739600742955685777779171742638265873702461272068680730571749782524876837132618541907334641567016563000494784555904722430224977335060591172535733207953809319846880882281850939445467613827435138416586160090147227067722996095476970616355633893
9
q值: 3765212177434976181850743050122551368955663346964589665557310362271748443386965838889028919191676723225416282294832748066320240750990334869599601377902592310599890226932277969392383986406350067828445163564040018237193737343097263342853805459216110139972124297259456882511039485803543420795891010965315506003183047871873886880038228464567631662027112455816062122949480783284904132864045819385111844864560289209980848407491300951641144522372938616694446736952916518817678578900174685814665533959374979134597687687657
n值: 2204437087190863275813498914667158613215115838973241699256082589514100681024710923725141716731878963903461280191061706856210478695537689270937893606957524734542103759820868546783190000403527455362826085370510450902755892279073298716748286276699069774582299360654040798459315056787233068052737296018159597761864511022875909502145130505194054756888147006894046212623939706661560839790917599722213600589129098845363088352939402026898685251759194836752088190037117396777450323660749182398045625170203502563584237473823103059932348228268327890393542120869322265301794974183639289065664785872322665278215289691580073961206905321254553740626041511977254908177089083435105646651097524254263212841441088971019576662408041035813171928947057188168390666207591729507655896274772732218369651865732005845632074280434816425608309873890668006457248554620853206604962263744642883673057802566780360837140765628336813462444908892220104725000001416586465735352122758103040744033019265363436033206343454253622778294784060276900954587759231
phi值: 2204437087190863275813498914667158613215115838973241699256082589514100681024710923725141716731878963903461280191061706856210478695537689270937893606957524734542103759820868546783190000403527455362826085370510450902755892279073298716748286276699069774582299360654040798459315056787233068052737296018159597761864511022875909502145130505194054756888147006894046212623939706661560839790917599722213600589129098845363088352939402026898685251759194836752088190037117396777450323660749182398045625170203502563584237469472416185834291549759510447688881663094313229258877829715633893690851253185108562567534749194423462533043801078854763941201338825896332501626146705137214942051808664190479993567164878526095765271012336511287728603381936824344964022154922013561539890230503060729790659794114038066410085437329893460705911050355765703157244852107073567987727214733290643910193150495596495417796340247271205679119846842543892943103271717473470283990274152163439240175053468360533077669601143426484510134045558076278849334214749328
e值: 5
d值: 1322662252314517965488099348800295167929069503383945019553649553708460408614826554235085030039127378342076768114637024113726287217322613562562736164174514840725262255892521128069914000242116473217695651222306270541653535367443979230048971766019441864749379611639242447907558903407233984083164237761089575865711870661372554570128707830331164328541328882041364277275743638239969365038745505598333281603534774593072178530117636412161392111510555169020512529140222704380664701941964495094388273751021221015381505424816834496111500574929855706268612848997856587937555326697829380336214510751911065137540520849516654077519826280647312858364720802815537799580975688023082328965231085198514287996140298927115657459162607401906772637162029162094606978413292953208136923934138301803643787439587646842283846051262397936076423546630213459402189434691126424414079263632883997438634611589029735789725067788414836272346747190810552633576586196302884082170394164491298063544105032081016319846601760686055890706080427334845767309600528849597
Ciphertext= 33715652388941O1
Plaintext= 2021
```

六、 困難與心得

經過這次的作業，清楚了解到 RSA 的加解密流程，以及過去數學家提出各種的同餘觀念，RSA 本身的流程很簡單，但是光是直接靠電腦次方運算需要花費大量的時間，所以加入 square and multiply 幫助次方加速，以及引入 Chinese Remainder Theorem 來對解密更加速，一開始不相信老師所講的去實驗直接靠電腦做次方的運算，過了十多分鐘都算不出來就選擇放棄，這次作業給我們的不只是 RSA 的架構，還有一些數學理論的延伸，我相信這些理論在未來研究所肯定可以幫上忙。