資訊安全導論 HW4 - RSA

B10615020 林哲旭

1. 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

1. 操作方式

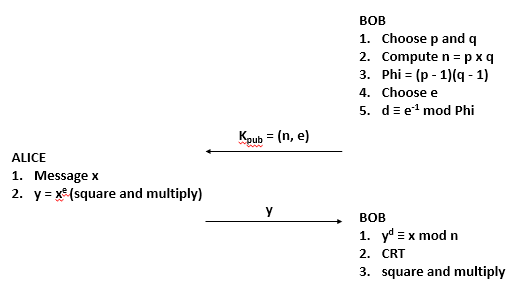
Makefile總共分為三個部分，make i、make r、make clean

* 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

1. 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(messagee mod n)
  + 再來產生m1, m2, h
    - m1 = cipertextdp mod p
    - m2 = cipertextdq 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()

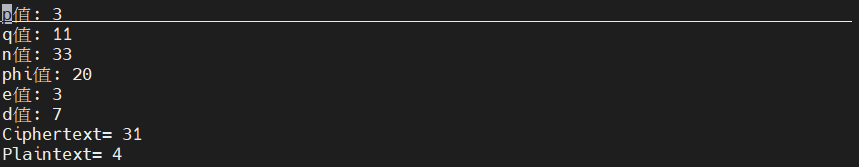
1. Custom Block Cipher Operation架構



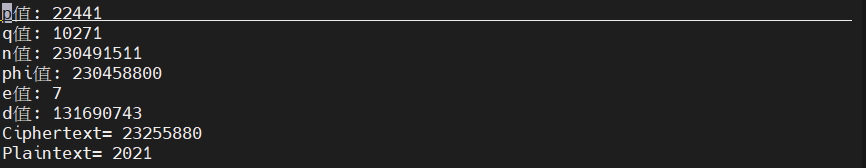
加解密架構

1. 實驗結果

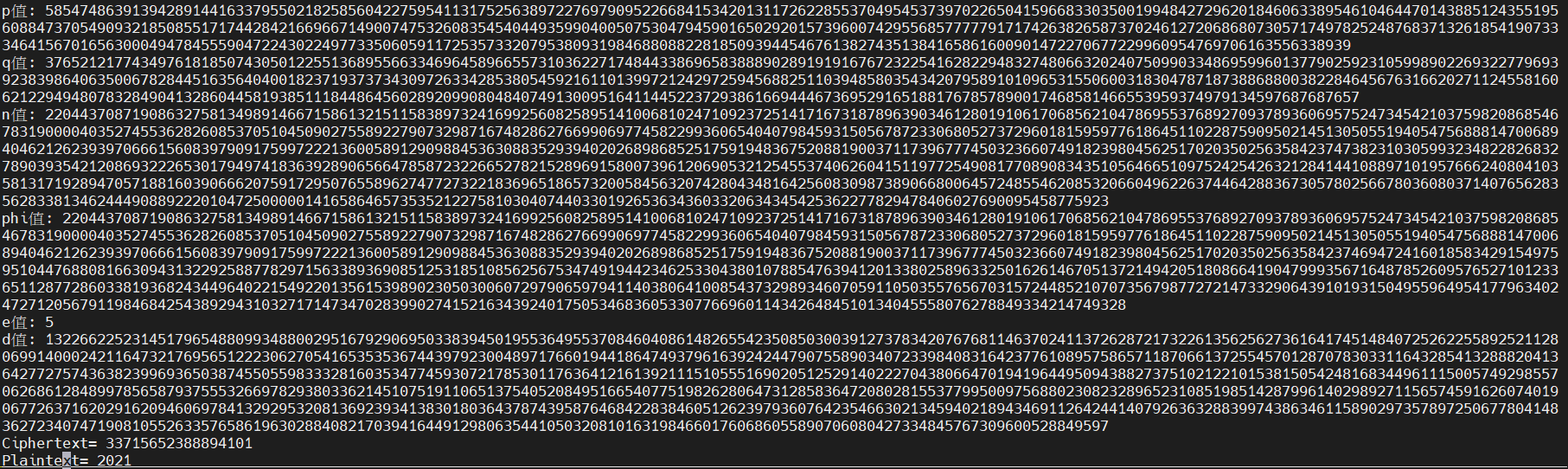
* Indivial 1:手動輸入p = 3, q = 11, plaintext = 4



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



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



1. 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。