密碼學期末報告

NTRU加解密方法python實作

1090536018 蔡昊璋

1. 加解密方式

Ntru 公鑰演算法需要的參數有N、p、q、 d：

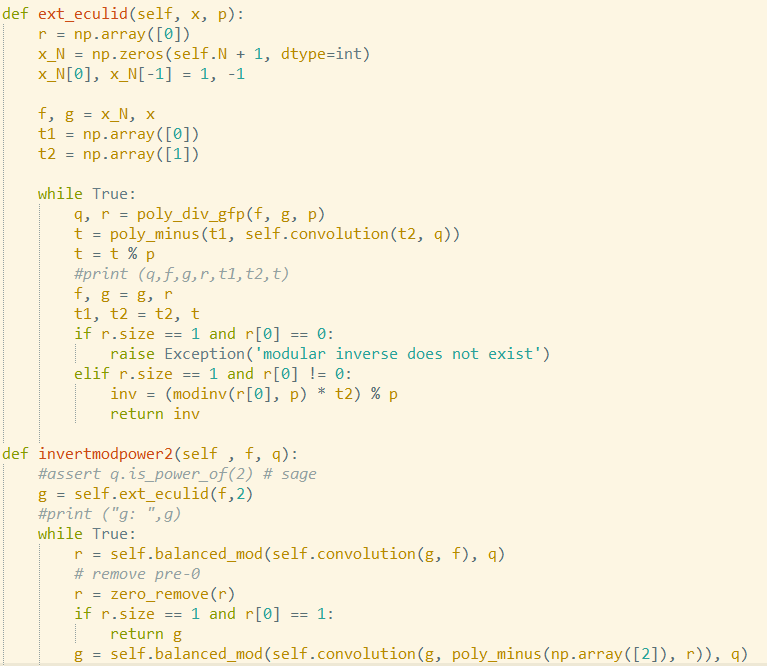
N 為多項式的最大階數，所有多項式運算都在模xN-1下執行

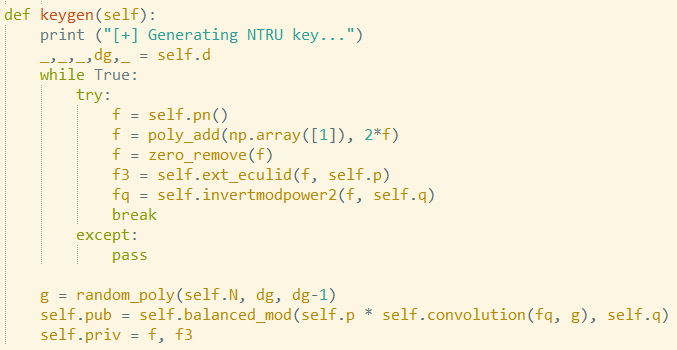
p 為係數運算的小模數

q 為係數運算的大模數

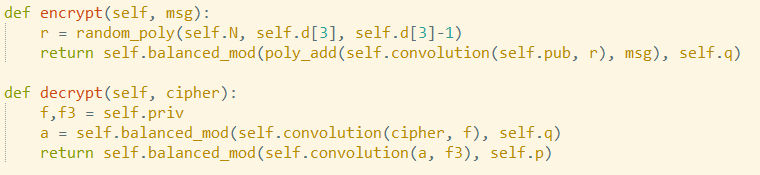
d 為產生隨機多項式的重要參數，提高加解密成功的可行性

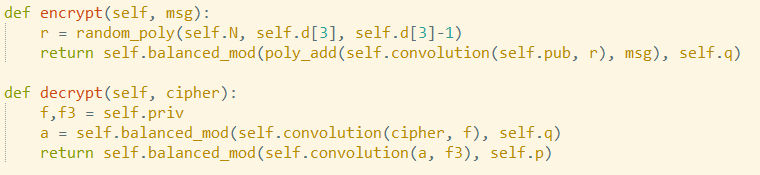
加密時先隨機產生一組多項式f，並分別在模p與模q下算出其乘法反元素Fp、Fq，在程式碼中，分別以NTRU類別下的成員函式ext\_eculid(self, f, p) 與invertmodpower2(self, f, q)來實現，值得一提的是，q選用的數必須為2的正整數次方，在計算Fq時，會先計算出mod 2 的乘法反元素，若mod 2沒有乘法反元素，則mod q就不會有乘法反元素，反之，可以迭代得到模q的乘法反元素。



產生模p模q都有乘法反元素的多項式需要花一點時間，產生完後，接下來隨機產生g，g必須要有dg個1和dg-1個-1，算出f, Fp, Fq, g後，指定f, Fp為私鑰，h=p(Fq\*g) mod q為公鑰。

加密時，會隨機產生一組多項式r，r必須要有dg個1和dg-1個-1，c = (r\*h + m) mod q，得到加密後的結果c，c為一個係數在0~2047之間的多項式。



解密時，先算出c\*f mof q的結果a，在d = a\*Fp mod p解回d，d=m即為解密成功。

1. 資料運算格式

多項式我一律使用numpy的array來儲存，以降冪排列，使用numpy的原因是在運算上方便改寫成c,c++或matlab的語法，而且在加減法以及純量乘法可以簡單的+-\*來完成，十分的方便。

1. 密文明文儲存方式

在明文映射的時候，我使用類似base64編碼的方式 – base3encode，把原本bytes的資料格式轉成係數為1、0、-1 的多項式。

這個多項式會以N為區塊作切分依序加密，因此在最後一塊不足N的地方補上亂數產生的1、0、-1做為填充，並以最後十個位置紀錄填充資料的長度，解密時依此長度移除填充的資料。

密文儲存方面，由於係數的範圍落在0 ~ 2048-1之間，我將係數由整數轉換為11位元(2048=211)的位元串(bit stream in numpy array)，再將位元串轉成位元組寫入檔案中。

完整流程為：加密時將檔案讀入為bytes格式，base3encode將byte轉為係數為0、1、-1的多項式，然後作填充，填充完的以CBC區塊加密模式進行加密，輸出cipherbinary，為多項式各係數的binary stream連接成的結果，在將cipherbinary 轉為byte格式儲存，解密時反向操作。

1. 執行方式

執行環境:python 3.7

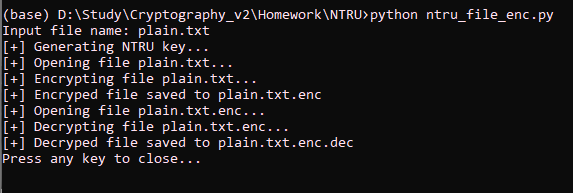
所需套件:numpy

自己寫的函式庫:polynomial\_arithmetic, gcd, ntru, libnum

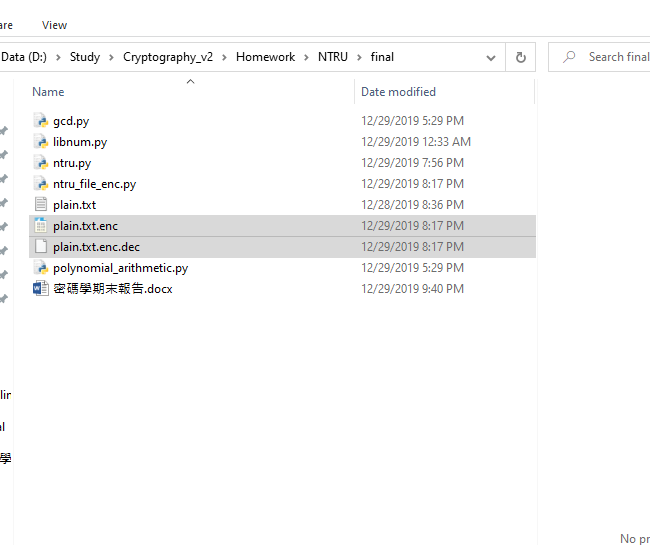
指令: python ntru\_file\_enc.py

執行結果:

輸入欲加密檔名



產生密文檔filename.enc 與解密檔filename.enc.dec



1. 心得

在這個作業中，大致有兩個困難點，第一是NTRU加解密的核心，也就是求多項式係數在模質數運算下的乘法反元素，這裡我卡很久，但後來我發現是我模多項試設錯(我設成xN+1)導致結果錯誤，後來把它改回xN – 1後就順利成功了，再來就是明文映射的方式，為了力求密文膨脹率不要太大，因為懶得去參考別的資料，我自己想出來一個最符合經濟效益的轉換方式，測試後大約膨脹7倍。

原本想說寫好加解密模組後來寫寫基於NTRU的簽章演算法，但發現NTRU加解密所用的參數並非對稱，因此並沒有辦法用私鑰來加密資料，所以無法實現簽章演算法，這讓我體會了其實並不是所有的公鑰密碼系統都是能做數位簽章的，公私鑰必須要有一定的對稱性才能作為一個數位簽章的好選擇。

但NTRU在基於公鑰加密實現的金鑰交換演算法是可行的，因此可將NTRU來做為替代RSA金鑰交換演算法的一個選擇。

1. 參考資料

* https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4800404
* https://latticehacks.cr.yp.to/ntru.html
* https://github.com/kpatsakis/NTRU\_Sage/blob/master/ntru.sage