RSA 加密演算法探討

簡介

RSA加密演算法是一種非對稱加密演算法。在公開金鑰加密和電子商業中RSA被廣泛使用。RSA是1977年由羅納德·李維斯特（Ron Rivest）、阿迪·薩莫爾（Adi Shamir）和倫納德·阿德曼（Leonard Adleman）一起提出的。當時他們三人都在麻省理工學院工作。RSA就是他們三人姓氏開頭字母拼在一起組成的。

RSA破解的難度就在於對超大的數做質因數分解。到目前為止，世界上還沒有任何可靠攻擊RSA演算法的方式。只要其鑰匙的長度足夠長，用RSA加密的訊息實際上是不能被破解的。

RSA加解密步驟

1. 隨意選擇兩個大的質數 p 和 q ， 且 p 不等於 q ，計算 $n = p * q$ 。

2. 根據尤拉定理，求得 $\varphi(n) = (p-1) * (q-1)$ 。關於尤拉定理，等一下會證明。

3. 取一個很大的隨機數 d，使 $\gcd(d, \varphi(n)) = 1$ 。$(1 < d < \varphi(n))$

4. 求得 d 的模反元素 e ，使 $e * d \equiv 1 \pmod{\varphi(n)}$ ，且 $1 < e < \varphi(n)$ 。（根據假設，

$gcd(d, \varphi(n)) = 1$，所以存在 d 的模反元素 e ，可用輾轉相除法證)

5. 銷毀 p 、q 。

6. < n , e > 作為用戶之公開鑰，於加密時使用。分發 d 給特定用戶(不公開)，於解密時使用。

加密

$$C \equiv M^e \pmod{n}$$

解密

$$M \equiv C^d \pmod{n}$$

(M 為明文， C 為密文)

數論

本文章將探討RSA數學證明，會用到一點數論和同餘的觀念。

尤拉定理及費馬小定理

尤拉定理

若 p 、 a 為正整數，且 p 、 a 互質（即 $\gcd(a,n)=1$），

則

$a^{\varphi(p)} \equiv 1 \pmod{p}$

即 p 整除 $a^{\varphi(p)}-1$；φ(p)為尤拉函數，小於 p 且跟 p 互質的正整數個數。

證明

假設a、 p 互質，令小於 p 且跟 p 互質的正整數集合為 $\{b_1,b_2,b_3,b_4.....b_{\varphi(p)}\}$

將這個集合的每個元素都乘 a ，變成 $\{b_1*a,b_2*a,b_3*a,b_4*a.....b_{\varphi(p)}*a\}$

然後再將每個元素都模 p 。因為 a、 p 互質， a 不為 p 的倍數，對於任兩個相異 b_k*a 、 $b_{k+x}*a$

$(k=1,2,3,4.....\varphi(p), x=1,2,3,4.....\varphi(p), k+x\le\varphi(p))$，其差不是 p 的倍數（所以不會有相同餘 數），且任一個 b_k*a 亦不為 p 的倍數（所以餘數不為 0），餘數所組成的集合剛好是

$\{b_1,b_2,b_3,b_4.....b_{\varphi(p)}\}$ 。

所以說若把 $\{b_1*a,b_2*a,b_3*a,b_4*a.....b_{\varphi(p)}*a\}$ 裡的每個元素乘起來模 p，

餘數為 $\{b_1,b_2,b_3,b_4.....b_{\varphi(p)}\}$ 所有元素的乘積，即:

$(b_1*a)*(b_2*a)*(b_3*a)*(b_4*a).....*(b_{\varphi(p)}*a)\equiv b_1*b_2*b_3*b_4.....*b_{\varphi(p)} \pmod{p}$

所以

$a^{\varphi(p)} \equiv 1 \pmod{p}$

若 p 為質數，則 $\varphi(p)=p-1$ ，則:

$a^{p-1} \equiv 1 \pmod{p}$

此即為費馬小定理。

$\varphi(n)$ 的算法

令 p 、 q 為質數，$n=p*q$ 。

因為 p 、 q 為質數，小於 n 且為 p 倍數的正整數有 $q-1$ 個 $(1*p,2*p,3*p.....(q-1)*p)$；小 於 n 且為 q 倍數的正整數有 $p-1$ 個 $(1*q,2*q,3*q.....(p-1)*q)$。所以 小於 n 且與 n 互質的 正整數個數 $\varphi(n)$ 為 $n-1-(p-1)-(q-1)$

$\varphi(n)=n-1-(p-1)-(q-1)$

$=p*q-1-(p-1)-(q-1)$

$=p*q-p-q+1=(p-1)*(q-1)$

得 $\varphi(n) = (p-1) * (q-1)$

RSA證明

欲證:

若 $C \equiv M^e \pmod{n}$ ，則 $M \equiv C^d \pmod{n}$ ($1 < e < \varphi(n), 1 < d < \varphi(n)$ ， e 、d 不為偶數，且 $0 < M < n$)

證明:

1.由剛才尤拉定理得知，$\varphi(n) = (p-1) * (q-1)$

2.因為 $e * d \equiv 1 \pmod{\varphi(n)}$ ， 所以 $e * d - 1$ 被 $\varphi(n)$ 整除 ，令 $e * d - 1 = k * \varphi(n), k \in$ 整 數

接下來討論

p 、 q 皆不整除 M 時:

則 $\gcd(M, n) = 1$, 根據尤拉定理， $M^{\varphi(n)} \equiv 1 \pmod{n}$

$M^{e*d-1} \equiv M^{k*\varphi(n)} \equiv (M^{\varphi(n)})^k \equiv 1^k \equiv 1 \pmod{n}$

$C^d \equiv (M^e)^d \equiv M^{e*d-1} * M \equiv 1 * M \pmod{n}$

得 $M \equiv C^d \pmod{n}$

p 、 q 其中一數整除 M ，另一數不整除 M 時:

令 p 整除 M ， q 不整除 M，則 $\gcd(M, q) = 1$, 根據尤拉定理， $M^{\varphi(q)} \equiv 1 \pmod{q}$

$M^{\varphi(n)} \equiv M^{(p-1)*(q-1)} \equiv (M^{q-1})^{p-1} \equiv (M^{\varphi(q)})^{p-1} \equiv 1^{p-1} \equiv 1 \pmod{q}$

$C^d \equiv (M^e)^d \equiv M^{e*d-1} * M \equiv M^{\varphi(n)*k} * M \equiv 1^k * M \equiv 1 * M \pmod{q}$

得 $C^d \equiv M \pmod{q}$ ， $C^d - M$ 被 q 整除

又 p 整除 M ，

$\Rightarrow (M^e)^d$ 被 p 整除

$\Rightarrow (M^e)^d - M$ 被 p 整除

$\Rightarrow C^d - M$ 被 p整除

$C^d - M$ 被 p 、 q 整除

$\Rightarrow C^d - M$ 被 n 整除

$\Rightarrow C^d \equiv M \pmod{n}$

得 $M \equiv C^d \pmod{n}$

p 、 q 皆整除 M 時(事實上不會發生，因為 $M < n = p * q$，不會有 p 、q 同時整除 M 的情形):

p 、 q 整除 M ，$(M^e)^d$ 被 p 、 q 整除

$\Rightarrow (M^e)^d - M$ 被 p 、 q 整除

$\Rightarrow C^d - M$ 被 p 、 q 整除

$\Rightarrow C^d - M$ 被 n 整除

$\Rightarrow C^d \equiv M \pmod{n}$

得 $M \equiv C^d \pmod{n}$

密鑰產生

在RSA中，p 、 q 是隨機亂數產生數字，然後再透過一些方法測試該數是否為質數(參見補充資料)。獲得 p 與 q 的值後，接下來就是尋找 d 與 e 。d 由隨機產生，且 $\gcd(d, \varphi(n)) = 1, 1 < d < \varphi(n)$ ，根據擴展歐基里德演算法(參見補充資料)， d 必有一個關於 $\varphi(n)$ 的模反元素 e，使 $e * d \equiv 1 \pmod{\varphi(n)}$ 。

所以我們能透過擴展歐基里德演算法，令 $e * d + k * \varphi(n) = 1$ (k 為整數) 求得 e 。

擴展歐基里德演算法舉例

若 $d = 11, p = 7, q = 13$

$\varphi(n) = (7 - 1) * (13 - 1) = 72$

$e * d \equiv 1 \pmod{\varphi(n)}$

$\Rightarrow d * e + \varphi(n) * k = 1$, $k \in$ 整數

假設 $72 * k + 11 * e = 1$

透過輾轉相除法:

$72 = 11 * 6 + 6$

$11 = 6 * 1 + 5$

$6 = 5 * 1 + 1$

再整理成:

$6 = 72 + 11 * (-6)$ -------(1)

$5 = 11 + 6 * (-1)$ ---------(2)

$1 = 6 + 5 * (-1)$ -----------(3)

然後(2)代入(3)產生(4)， 將(1)代入(4)得 k 、 e

$1 = 6 + [11 + 6 * (-1)] * (-1)$

$1 = 11 * (-1) + 6 * 2$ ------------(4)

$1 = 11 * (-1) + [72 + 11 * (-6)] * 2$

$1 = 72 * 2 + 11 * (-13)$

得 $e = -13, k = 2$

又 e 需大於 0 ，所以要加上 $\varphi(n)$

$e = -13 + \varphi(n) = -13 + 72 = 59$

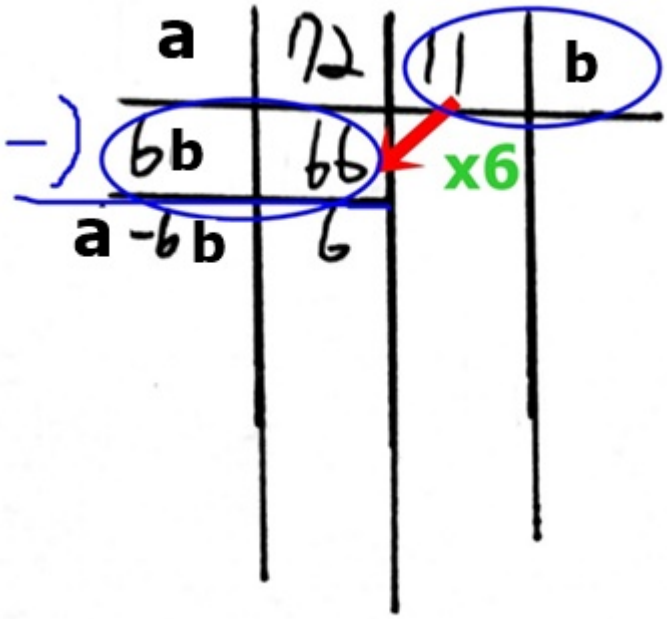
得 $e = 59$

更簡單的方法: 輾轉相除法直式

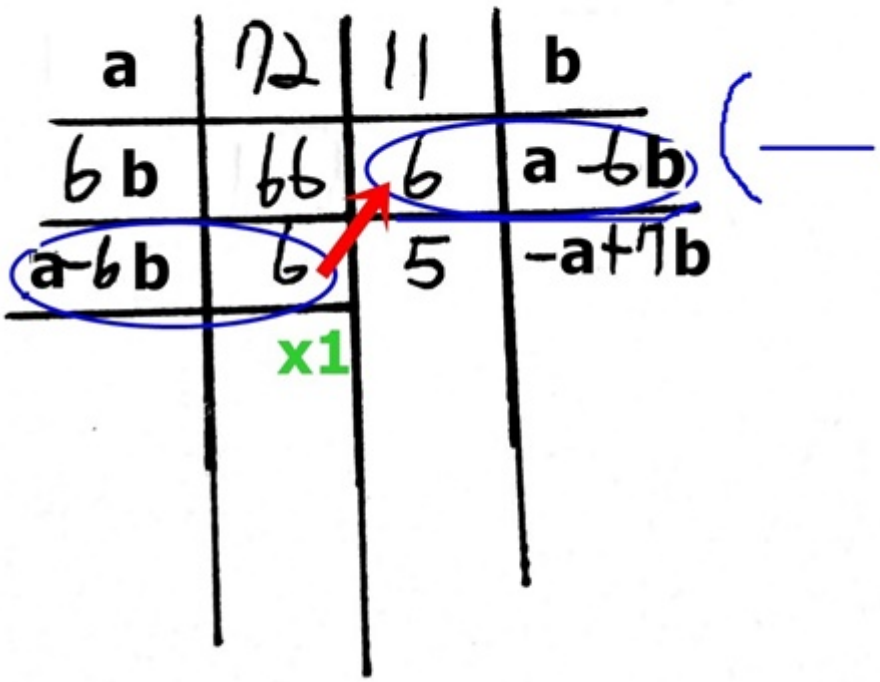
我們把 $72 * k + 11 * e = 1$ 寫成這樣:

step1: a 就是 72，b 就是 11，將 a 寫在 72 旁，b寫在 11 旁，接下來做輾轉相除法。

將 11 乘 6 寫到 72 下方，相減得 6；b 乘 6 寫到 a 下方，相減得 $a - 6 * b$

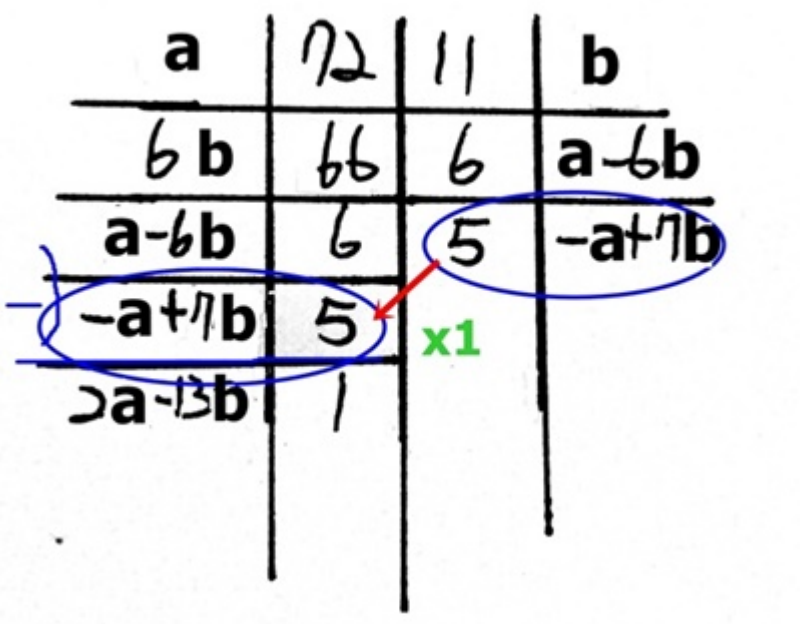


step2: 將 6 乘 1 寫到 11 下方，相減得 5；$a-6*b$ 乘 1 寫到 b 下方，相減得 $-a+7*b$

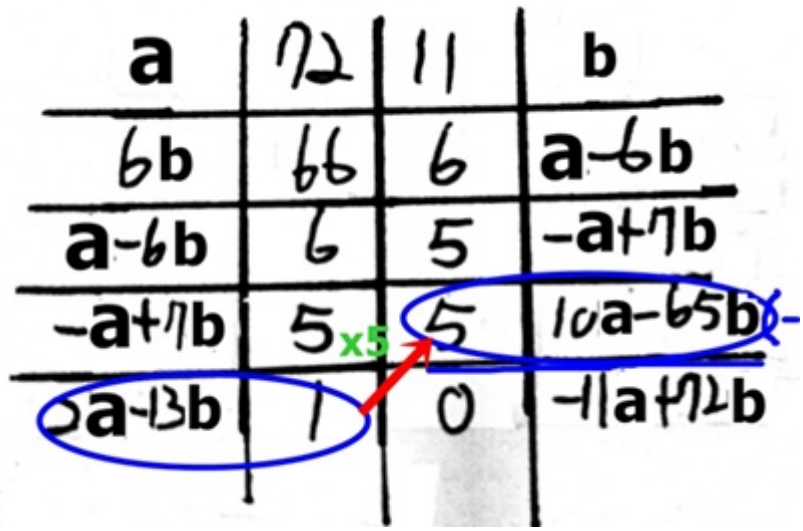


後面依此類推

step3:



step4:



當 $2*a-13*b$ 整除 $10*a-65*b$ 時，$2*a-13*b$ 為 72 和 11 的最大公因數 1，所以 $e=-13, k=2$ 。

又 e 需大於 0 ，所以要加上 $\varphi(n)$

$e=-13+\varphi(n)=-13+72=59$

得 $e=59$

RSA的特性

1. 其實 e 和 d 是可互換的，如果 e 作為 公鑰，則 d 就為私鑰；若 e 作為私鑰，則 d 就作為公鑰。這點可以從 $(M^e)^d=(M^d)^e\equiv M \pmod{n}$ 看出。

2. e 、 d 不能為偶數 。 因為 $\gcd(e, \varphi(n)) = \gcd(e, (p-1)*(q-1)) = 1$ ，$\gcd(d, \varphi(n)) = \gcd(d, (p-1)*(q-1)) = 1$ ， p 、 q 皆為超大奇質數，若 e 、 d 為偶數，與 $\varphi(n) = (p-1)*(q-1)$ 的最大公因數將為 2 。

3. 明文 M 需小於 n 。因為 $M \equiv C^d \pmod{n}$ ，$M = C^d - k*n$ ($k \in$ 整數)，若 M 大於 n ，無法確定M 是多少。換句話說， M 為 C^d 除以 n 的餘數，因為餘數不能超過除數，所以 M 需小於 n 。

4. e 滿足 $1 < e < \varphi(n)$ 。因為 p 、 q 為超大質數， $\varphi(n) = (p-1)*(q-1)$ 位數也會超大，當 $e > \varphi(n)$ ， $C \equiv M^e \pmod{n}$ 的次方運算上會很困難。因為 $M^e \equiv M^{e \pmod{\varphi(n)}} \pmod{n}$，可以先將 e 模 $\varphi(n)$ 化簡。又若 $e = 1$ ，因為 $C \equiv M^e \equiv M^1 \pmod{n}, M < n$ ，所以 $C = M^1$ ，根本沒加密，故 e 需大於 1。對於 d 亦然。

5. p 、 q 滿足 $p \neq q$ 。因為 $n = p*q$ ，當 $p = q \Rightarrow n = p^2$ ，對於電腦來說計算根號 n 是很容易的。

RSA質因數分解攻擊

其實RSA的本質就是在做大整數的質因數分解，只要成功把 n 分解成 p 、 q ，就能得到 $\varphi(n)$ ，搭配公鑰 e ，就能推出 d。但是難就難在把 n 質因數分解，現行的憑證 n大概 600多位數，要分解成兩個超大質數相乘難度超高。

時間攻擊

1995年有人提出了一種非常意想不到的攻擊方式：假如監聽者對加密者的硬體有充分的了解，而且知道它對一些特定的訊息加密時所需要的時間的話，那麼她可以很快地推導出 d 。這種攻擊方式之所以會成立，主要是因為在進行加密時所進行的模指數運算是一個位元一個位元進行的，而位元為 1 所花的運算比位元為 0 的運算要多很多，因此若能得到多組訊息與其加密時間，就會有機會可以反推出私鑰的內容。

Python解題工具實作

利用爬蟲爬取線上質因數分解資料庫 factordb (http://www.factordb.com/) 來做質因數分解，然後用擴展歐基里德演算法算出私鑰。

首先，先做爬蟲，觀察factordb，發現這個網站是用GET method，觀察其格式，寫出爬蟲

```
import urllib.request
import urllib.parse
from bs4 import BeautifulSoup

def factor(n):
    #把數字編碼成query string
    value={'query':str(n)}
```

```
    data=urllib.parse.urlencode(value)
    addr="http://www.factordb.com/index.php?{url}".format(url=data)

    #GET請求
    resp = urllib.request.urlopen(addr)

    #BeautifulSoup 處理網頁
    soup=BeautifulSoup(resp,"html.parser")
    strlist=soup.find_all("font",color="#000000")

    primelist=[]

    #將抓到的質因數放進list裡(搞不好不只兩個質因數)
    for i in strlist:
        primelist.append(int(i.text))

    return primelist

```

擴展歐基里德演算法

```
def extgcd(a, b):
    if b == 0: #終止條件 b==0
        return 1, 0
    else:
        x, y = extgcd(b, a % b) #遞迴做輾轉相除法
        x, y = y, (x - (a // b) * y)

        return x, y
```

關於程式碼第 6 行的解釋：

令$\gcd(a_1,b_1)=a_1*x_1+b_1*y_1$

做輾轉相除法:

> $\gcd(a_1,b_1)=a_1*x_1+b_1*y_1$
>
> $\gcd(a_2,b_2)=a_2*x_2+b_2*y_2=\gcd(b_1,a_1 \pmod{b_1})=b_1*x_2+(a_1 \pmod{b_1})*y_2$
>
> $\dots$
>
> $\gcd(a_i,b_i)=a_i*x_i+b_i*y_i$
>
> $\gcd(a_{i+1},b_{i+1})=a_{i+1}*x_{i+1}+b_{i+1}*y_{i+1}=\gcd(b_i,a_i \pmod{b_i})=b_i*x_{i+1}+(a_i \pmod{b_i})*y_{i+1}$
>
> $\dots$
>
> $\gcd(1,0)=1*x_{i+k}+0*y_{i+k}=1*1+0*0=1$

因為 $\gcd(a_i, b_i) = \gcd(b_i, a_i \pmod{b_i}) = \gcd(a_{i+1}, b_{i+1})$

所以 $a_i * x_i + b_i * y_i = b_i * x_{i+1} + (a_i \pmod{b_i}) * y_{i+1}$

$a_i * x_i + b_i * y_i = b_i * x_{i+1} + (a_i - [a_i/b_i] * b_i) * y_{i+1}$

$= a_i * y_{i+1} + b_i * (x_{i+1} - [a_i/b_i] * y_{i+1})$

得到

$x_i = y_{i+1}$,

$y_i = x_{i+1} - [a_i/b_i] * y_{i+1}$

最後，就是 RSA 以及輸入輸出

```
c,e,n=int(input("c:")),int(input("e:")),int(input("n:")) #輸入 c,e,n

primelist=factor(n) #n的質因數

phi_n=1

print("\nprimes:")
for i in primelist: #計算phi(n)
    print(i)
    phi_n*=i-1

print("\nphi(n):",phi_n)
d,y,q=extgcd(e%(phi_n),phi_n)

#檢查d是否<0，如果是: d加上phi(n)
if d<0:
    d+=phi_n
print("\nd:",d)

m=pow(c,d,n) #c^d%n
print("message:",m)
```

完整的程式碼

```
import urllib.request
import urllib.parse
from bs4 import BeautifulSoup
```

```
4
5  def factor(n):
6      #把數字編碼成query string
7      value={'query':str(n)}
8      data=urllib.parse.urlencode(value)
9      addr="http://www.factordb.com/index.php?{url}".format(url=data)
10
11     #GET請求
12     resp = urllib.request.urlopen(addr)
13
14     #BeautifulSoup 處理網頁
15     soup=BeautifulSoup(resp,"html.parser")
16     strlist=soup.find_all("font",color="#000000")
17
18     primelist=[]
19
20     #將抓到的質因數放進list裡(搞不好不只兩個質因數)
21     for i in strlist:
22         primelist.append(int(i.text))
23
24     return primelist
25
26 def extgcd(a, b):
27      if b == 0:
28          return 1, 0
29      else:
30          x, y= extgcd(b, a % b)
31          x, y = y, (x - (a // b) * y)
32          return x, y
33
34 c,e,n=int(input("c:")),int(input("e:")),int(input("n:"))
35
36 primelist=factor(n)
37
38 phi_n=1
39
40 print("\nprimes:")
41 for i in primelist:
42     print(i)
43     phi_n*=i-1
44
45 print("\nphi(n):",phi_n)
46 d,y=extgcd(e%(phi_n),phi_n)
47
48 #檢查d是否<0
49 if d<0:
50     d+=phi_n
51 print("\nd:",d)
52
53 m=pow(c,d,n) #c^d%n
54 print("message:",m)
```

註: 當數字過大時 factordb的格式會有所改變，這隻爬蟲還可以改進

思考

剛剛在程式碼註解裡有提到， n 的質因數可能不只一個， 想想看當 $n = p * q * r$ 時 RSA 的證明。

跟兩個質因數時的證明差不多。下面這篇可以參考看看。

https://crypto.stackexchange.com/questions/44110/rsa-with-3-primes

後序

在剛接觸資安 CTF 競賽的 Cryptography 項目時，面臨到的最大問題就是對於RSA 加密的理解。上過許多資安課程，但似乎對於RSA理論的部分沒有解釋太多，一直無法搞懂整個加密在做甚麼，所以以前遇到 RSA的題目只能把做法背下來。可是背作法容易忘記，而且一旦遇到變化的題目就卡死了，硬背解題公式實在不是個明智的做法。於是下定決心要把它搞懂，在反覆和數學老師討論和多方參考資料後，一步步證明尤拉定理，推導出 RSA 的證明，發現RSA加密其實還蠻有趣的。

感謝 JZ 老師提供資料以及與我討論 RSA 加密，使得我能夠了解整套演算法，終於能夠不再對於 RSA 的題目感到茫然，了結了一直以來想搞懂的東西，然後寫成這篇文章。

感謝許嘉麟學長點出瑕疵以及提供建議，與學長討論後學到了很多。

補充資料

<https://bindog.github.io/blog/2014/07/19/how-to-generate-big-primes/> RSA周邊——大素數是怎樣生成的？

<https://zh.wikipedia.org/wiki/%E6%A8%A1%E5%8F%8D%E5%85%83%E7%B4%A0> 模反元素

參考資料

<https://zh.wikipedia.org/wiki/RSA%E5%8A%A0%E5%AF%86%E6%BC%94%E7%AE%97%E6%B3%95> RSA加密演算法

<https://zh.wikipedia.org/wiki/%E6%AC%A7%E6%8B%89%E5%AE%9A%E7%90%86_(%E6%95%B0%E8%AE%BA)> 尤拉定理

<https://zh.wikipedia.org/wiki/%E6%AC%A7%E6%8B%89%E5%87%BD%E6%95%B0> 尤拉函數

<https://zh.wikipedia.org/wiki/%E8%B4%B9%E9%A9%AC%E5%B0%8F%E5%AE%9A%E7%90%86> 費馬小定理

<https://codertw.com/%E7%A8%8B%E5%BC%8F%E8%AA%9E%E8%A8%80/537802/> 尤拉定理，費馬小定理證明

<https://zh.wikipedia.org/wiki/%E6%89%A9%E5%B1%95%E6%AC%A7%E5%87%A0%E9%87%8C%E5%BE%97%E7%AE%97%E6%B3%95> 擴展歐幾里得算法

若有謬誤，請不吝指教

Author: TNFSHchin 2019/4/19