RSA 加密演算法探討

作者: 台南一中 王勤 2019/4/19 · 2020/3/7 更新

Github: https://github.com/zinwang

E-mail: chriswangxxxxx@gmail.com

研究動機

在我始對於資訊領域感到興趣的高一時期，接觸了資訊安全的範疇。在資訊安全廣泛的領域中，最令我感到著迷的類別便是密碼學，因為很艱澀，我聽不懂。還記得在一場資訊安全培訓課程中，那天在講現代密碼學，那是我第一次邂逅密碼學。只記得講師以快速的語調解釋著各個理論與加密方法，我只能迷迷糊糊地試圖跟上講師，只見台下死沉沉一片，大家都很迷惘。那一天我沒有學到太多的理論，但是一個名字深深烙印在我的腦海中，RSA。我很不服輸，下定決心研究密碼學，進入密碼學奇妙的世界。

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。

目錄

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 $n = pq$ 。

2. 根據歐拉定理，求得 $\varphi(n) = (p-1)(q-1)$ 。關於歐拉定理，等一下會證明。

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

4. 求得 d 的模反元素 e ，使 $ed \equiv 1 \pmod{\varphi(n)}$ ，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 ，可用輾轉相除法證明)。

5. 銷毀 p 、q 。 (n, e) 作為用戶之公開鑰，於加密時使用。 特定用戶保留 d，於解密時　使用。

加密

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

解密

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

(M 為明文， C 為密文)

數論

模反元素

當 a, b 為互質兩正整數，存在 a 的模反元素 a^{-1} 使得 $aa^{-1} \equiv 1 \pmod{b}$ 。

證明

已知 $\gcd(a, b) = 1$，假設 $1 < a < b$。

我們利用輾轉相除法

$$\begin{cases} b = aq_1 + r_1, & 0 \leq r_1 < d \\ a = r_1 q_2 + r_2, & 0 \leq r_2 < r_1 \\ r_1 = r_2 q_3 + r_3, & 0 \leq r_3 < r_2 \\ \quad\quad . \\ \quad\quad . \\ \quad\quad . \\ r_{m-2} = r_{m-1} q_m + r_m, & 0 \leq r_m < r_{m-1} \\ r_{m-1} = r_m q_{m+1} + 0 \end{cases}, (q_1, q_2 \ldots q_m, q_{m+1} \in \mathbb{Z}).$$

我們能把 r_1 寫成 $b - aq_1$ ，帶入 $a = r_1 q_2 + r_2$ 中，得到 $r_2 = a - (b - aq_1) q_2$ ，$r_2 = a(1 + q_1 q_2) - bq_2$ 。我們能如此一直做下去，把 r_2 整理出來的形式帶入下一式，我們會發現餘數都是 a 和 b 的線性組合 。因為 a 和 b 都是正整數，我們能透過有限步驟的輾轉相除法求得一個 r_m 使得 $r_{m-1} = r_m q_{m+1} + 0$ ，即是使得 $r_{m-2} = r_{m-1} q_m + r_m$ ， r_m 即是 a 和 b 的最大公因數。於是我們把 $r_{m-2} = r_{m-1} q_m + r_m$ 化成 a 和 b 的線性組合，必有 $ah + bk = r_m = \gcd(a, b) = 1$ (其中 h, k 為整數) 的關係，即是 $ah \equiv 1 \pmod{b}$ 。

其實我們證明 a 的關於 b 的模反元素，就是在探討二元一次不定方程整數解的情形: 當 $a, b \in \mathbb{N}$ ，則不定方程 $ax + by = \gcd(a, b)$ 有解。

歐拉定理

歐拉函數

對於正整數 n ，歐拉函數 $\varphi(n)$ 是小於或等於 n 的正整數中與 n 互質的數的數目。

歐拉函數的乘法性

若 m, n 為兩正整數滿足 $\gcd(m,n) = 1$ ，則 $\varphi(mn) = \varphi(m)\varphi(n)$ 。

歐拉函數的計算公式

(1) 令 p 為質數， $m = p^r$ 。在小於 m 的正整數中，除了 p 的倍數，其餘皆與 p 互質。

因此 $\varphi(m) = p^r - p^{(r-1)} = p^r(1-p^{-1})$ 。

(2) 令 n 標準分解式為 $n = p_1^{a_1}p_2^{a_2}p_3^{a_3}\dots p_k^{a_k}$ ，其中 $p_1, p_2, p_3, \dots, p_k$ 皆為質數。依據歐拉函數乘法性，我們有 $\varphi(n) = \varphi(p_1^{a_1}p_2^{a_2}p_3^{a_3}\dots p_k^{a_k}) = \varphi(p_1^{a_1})\varphi(p_2^{a_2})\varphi(p_3^{a_3})\dots\varphi(p_k^{a_k})$ 。

由 (1) 可得到

$$\varphi(n) = \prod_{i=1}^{k} p_i^{a_i}(1-p_i^{-1})。$$

歐拉定理

若 p 、 a 為正整數，且 p 、 a 互質，

則

$a^{\varphi(p)} \equiv 1 \pmod{p}$ 。

我們將用歐拉定理證明 RSA 演算法。

歐拉定理證明

證明步驟

在這裡我們用簡單集合的概念解釋整個證明過程。首先，假設兩正整數 a、 p 互質，令小於 p 且跟 p 互質的正整數構成集合 $\{b_1, b_2, b_3, b_4 \dots\dots b_{\varphi(p)}\}$ 。將這個集合的每個元素都乘 a ，變成 $\{b_1a, b_2a, b_3a, b_4a \dots\dots b_{\varphi(p)}a\}$ 。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_ma 、 b_na，其中 m, n 都介於 1 和 $\varphi(n)$ 之間且$m \neq n$，其差不是 p 的倍數(所以不會有相同餘數)，且任一個 b_ka 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 $\{b_1, b_2, b_3, b_4 \dots\dots b_{\varphi(p)}\}$ 。

假設

$$\begin{cases} b_1a = pk_1 + r_1, & 0 \le r_1 < p \\ b_2a = pk_2 + r_2, & 0 \le r_2 < p \\ b_3a = pk_3 + r_3, & 0 \le r_3 < p \\ \qquad\qquad . & \qquad\qquad , (k_1, k_2, k_3 \ldots k_{\varphi(n)} \in \mathbb{Z}). \\ \qquad\qquad . & \\ \qquad\qquad . & \\ b_{\varphi(n)}a = pk_{\varphi(n)} + r_{\varphi(n)}, & 0 \le r_{\varphi(n)} < p \end{cases}$$

若把 $\{b_1a, b_2a, b_3a, b_4a \ldots\ldots b_{\varphi(p)}a\}$ 裡的每個元素乘起來模 p，則餘數為

$r_1r_2r_3 \ldots r_{\varphi(n)}$ ，即為$\{b_1, b_2, b_3, b_4 \ldots\ldots b_{\varphi(p)}\}$ 所有元素的乘積。所以我們有如下等式：

$$(b_1a)(b_2a)(b_3a)(b_4a) \ldots\ldots (b_{\varphi(p)}a) \equiv b_1b_2b_3b_4 \ldots\ldots b_{\varphi(p)} \pmod{p}。$$

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

當然，在上段的証明過程中仍然有些沒有很漂亮地解釋的敘述。

所以我們要引入群的概念來進一步解釋並由群來重新詮釋整個證明過程。

群

群 $(G, *)$ 是由集合 G 和二元運算 $*$ ($*$ 指的是任意二元運算)所組成的代數結構。具有四個群公理分別為封閉性、單位元素反元素、及結合律。在歐拉定理證明中，我們將用到單位元素和反元素以及消去律。

單位元素:

群 G 中必存在單位元素 e 使得對於群 G 中的所有元素 a ， $a * e = e * a = e$ 成立。

反元素:

在群 G 中對於所有的元素 a ，存在 a 的反元素 a^{-1} ，使得 $a * a^{-1} = a^{-1} * a = e$ 成立。

消去律:

消去律不是群公理之一，它其實是建立在反元素上。對於群 G 中任三元素 a, b, c 所構成的等式: $a * b = a * c$ 中，可利用 a 的反元素 a^{-1} 消去 a，即 $a^{-1}a * b = a^{-1}a * c$ ，得 $b = c$。

利用群證明歐拉定理

令 p, a 為整數且 $\gcd(p, a) = 1$。令整數模 p 的同餘類所構成的交換群(具有交換律的群) $\mathbb{Z}_p^\times = \{k_1, k_2, k_3, \dots k_{\varphi(p)}\}$ ，一般稱為整數模 n 乘法群。使 $\mathbb{Z}_p^\times$ 上每個元素都乘上 a ， $\mathbb{Z}_p^\times k = \{k_1 a, k_2 a, k_3 a, \dots k_{\varphi(p)} a\}$ 。由消去律可知 $\mathbb{Z}_p^\times a$ 中所有元素都不重複，所以 $\mathbb{Z}_p^\times = \mathbb{Z}_p^\times a \pmod p$ 。因此 $(k_1 a)(k_2 a)(k_3 a)\dots(k_{\varphi(p)} a) \equiv k_1 k_2 k_3 \dots k_{\varphi(p)} \pmod p$ 。

交換律使我們能整理成 $k_1 k_2 k_3 \dots k_{\varphi(p)} a^{\varphi(p)} \equiv k_1 k_2 k_3 \dots k_{\varphi(p)} \pmod p$ ，由於 $\mathbb{Z}_p^\times$ 內的元素都與 p 互質，所以$k_1, k_2, k_3, \dots k_{\varphi(p)}$ 都有關於 p 的模反元素，因此同餘式左右兩邊都乘上 $k_1, k_2, k_3, \dots k_{\varphi(p)}$ 的模反元素 $k_1^{-1}, k_2^{-1}, k_3^{-1}, \dots k_{\varphi(p)}^{-1}$ ，消去 $k_1 k_2 k_3 \dots k_{\varphi(p)}$ ，我們有 $a^{\varphi(p)} \equiv 1 \pmod p$ ，證明完畢。

我們用交換群的群公理及特性完美地執行了上方 **證明步驟** 裡所做的事，簡潔明白。

此外，群論還有一樣絕招，拉格朗日定理。

拉格朗日定理描述有限群 G 的子群 H 之階整除 G 之階。

此定理能更簡潔的證明歐拉定理，詳情可參見補充資料 [2] 。

費馬小定理

費馬小定理為歐拉定理的特殊情況。

若 p 為質數， $\varphi(p) = p - 1$ ，則:

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

此即為費馬小定理。

RSA 證明

欲證:

若 $C \equiv M^e \pmod{n}$ ，則 $M \equiv C^d \pmod{n}$ ，其中 e, d 介於 1 與 $\varphi(n)$ 之間的奇數並滿足$ed \equiv 1 \pmod{\varphi(n)}$ ，且 $0 < M < n$。

證明:

(1) 由歐拉函數的計算公式可得 $\varphi(n) = (p-1)(q-1)$。

(2) 因為 $ed \equiv 1 \pmod{\varphi(n)}$ ， 所以 $ed-1$ 被 $\varphi(n)$ 整除 ，令 $ed-1 = k\varphi(n), k \in \mathbb{Z}$ 。

(3) 我們把 M^{ed} 寫成 $(M^{ed-1})M$。我們又能利用步驟(2) 的假設把 $ed-1$ 替換掉 ，得到 $(M^{k\varphi(n)})M$ ，並寫成 $M(M^{\varphi(n)})^k$ 。

(4) 接下來討論三種情況:

(i) p 、 q 皆不整除 M 時:

此時 $\gcd(M, n) = 1$。

已知 $C \equiv M^e \pmod{n}$ ，則 C^d 在模 n 情況下同餘 $(M^e)^d$ 。

由(3)，我們有

$$C^d \equiv (M^e)^d \equiv (M^{ed-1})M \equiv (M^{k\varphi(n)})M \pmod{n} = (M^{\varphi(n)})^k M \pmod{n}。$$

根據歐拉定理，我們知道 $M^{\varphi(n)} \equiv 1 \pmod{n}$。

所以我們有 $C^d \equiv (M^{\varphi(n)})^k M \equiv (1^k)M \pmod{n} = M \pmod{n}$。

$M \equiv C^d \pmod{n}$ 在 p, q 不整除 M 時成立。

(ii) p 、 q 其中一數整除 M ，而另一數不整除 M 時:

令 p 整除 M ， q 不整除 M，則 $\gcd(M, q) = 1$ 。

已知 $C \equiv M^e \pmod{n}$ ， 令 $C + kn = M^e, k \in \mathbb{Z}$ ，

則 $C + (kp)q = M^e$ ，我們有 $C \equiv M^e \pmod{q}$ 。

由(3)，我們有

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

我們把 $\varphi(n)$ 拆開成 $(p-1)(q-1)$，即 $\varphi(n)=\varphi(q)(p-1)$。

於是我們有

$C^d \equiv (M^{\varphi(n)})^k M \pmod{q} = (M^{\varphi(q)(p-1)})^k M \pmod{q} = (M^{\varphi(q)})^{k(p-1)} M \pmod{q}$ 。

根據歐拉定理 $M^{\varphi(q)} \equiv 1 \pmod{q}$ ，

得到 $C^d \equiv (1^{k(p-1)})M \pmod{q} = M \pmod{q}$ 。

同樣的手法，令 $C^d-(rp)q=M, r \in Z$ ，

由 $C^d - r(pq) = C^d - rn = M$ 可清楚知道，

$C^d \equiv M \pmod{n}$ 在 p 、 q 其中一數整除 M 而另一數不整除 M 時成立。

(iii) p 、 q 皆整除 M 時

此情況不會發生，因為 $M < n = pq$，不會有 p 、q 同時整除 M 的情形。

但若依照上方的做法 ，亦不難證明就算 $M \geq n$ 而 p 、 q 皆整除 M 時，$C^d \equiv M \pmod{n}$ 也會成立。

密鑰產生

在RSA中，p 、 q 是隨機亂數產生數字，再透過一些方法測試該數是否為質數(參見補充資料[1])。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 $\gcd(d,\varphi(n))=1$，並符合$1<d<\varphi(n)$ ， 則 d 必有一個關於 $\varphi(n)$ 的模反元素 e，使 $ed \equiv 1 \pmod{\varphi(n)}$ ，這在稍早已證明過。

擴展輾轉相除法產生密鑰

我們能透過擴展輾轉相除法來求 e 。

擴展輾轉相除法舉例

若 $d=11, p=7, q=13$ ，

$\varphi(n)=(7-1)(13-1)=72$ ，

$ed \equiv 1 \pmod{\varphi(n)}$ ，

則 $de+\varphi(n)k=1$, $k \in \mathbb{Z}$ 。

假設 $72k + 11e = 1$ 。

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 $e = -13, k = 2$

然而 e 需大於 0 ，所以要加上 $\varphi(n)$

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

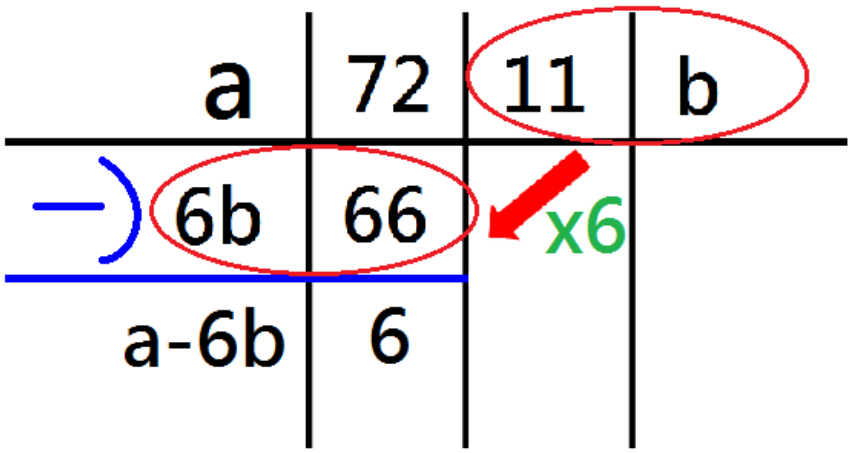
更簡單的方法: 輾轉相除法直式

我們把 $72k + 11e = 1$ 寫成這樣:

步驟一:

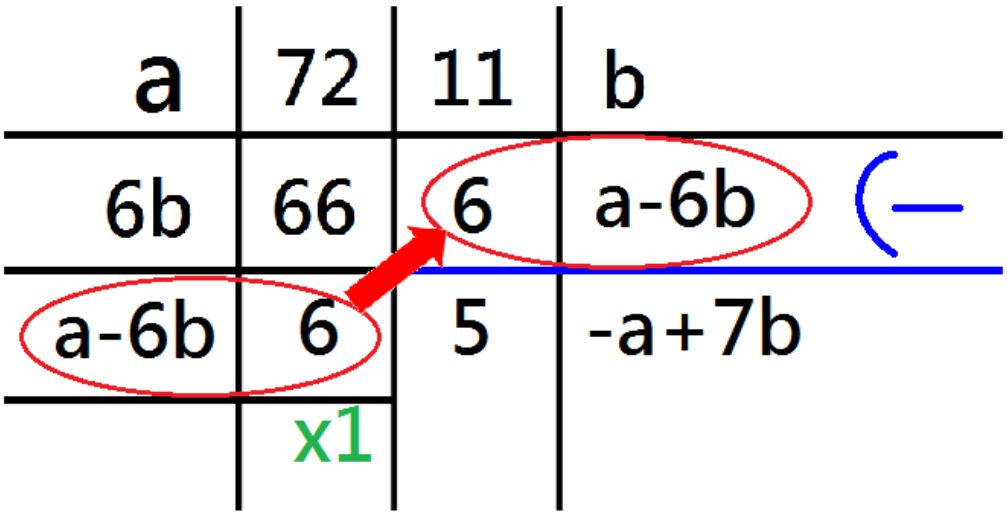
a 就是 72，b 就是 11，將 a 寫在 72 旁，b寫在 11 旁，接下來做輾轉相除法。

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



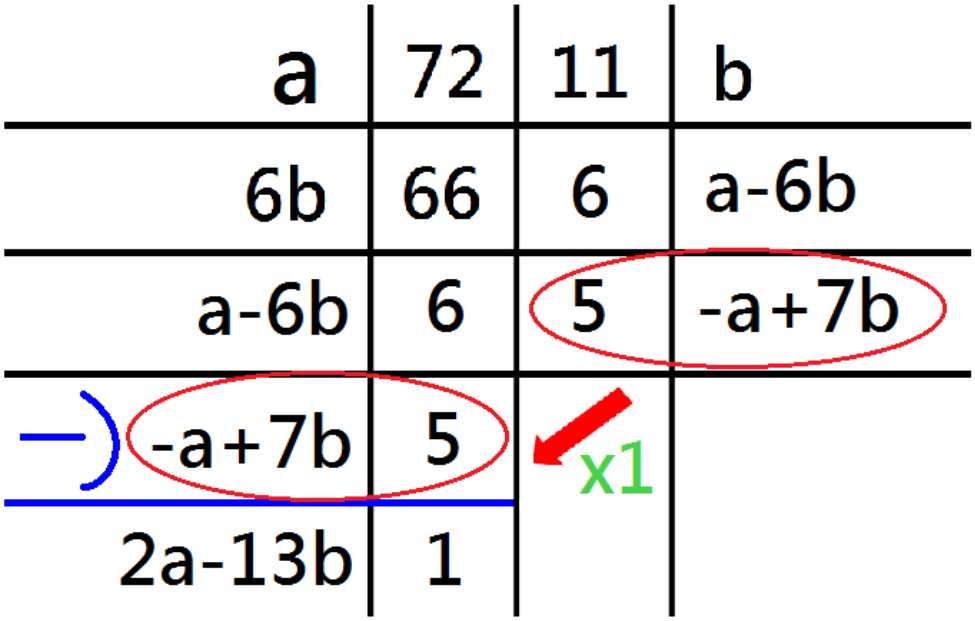
步驟二:

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

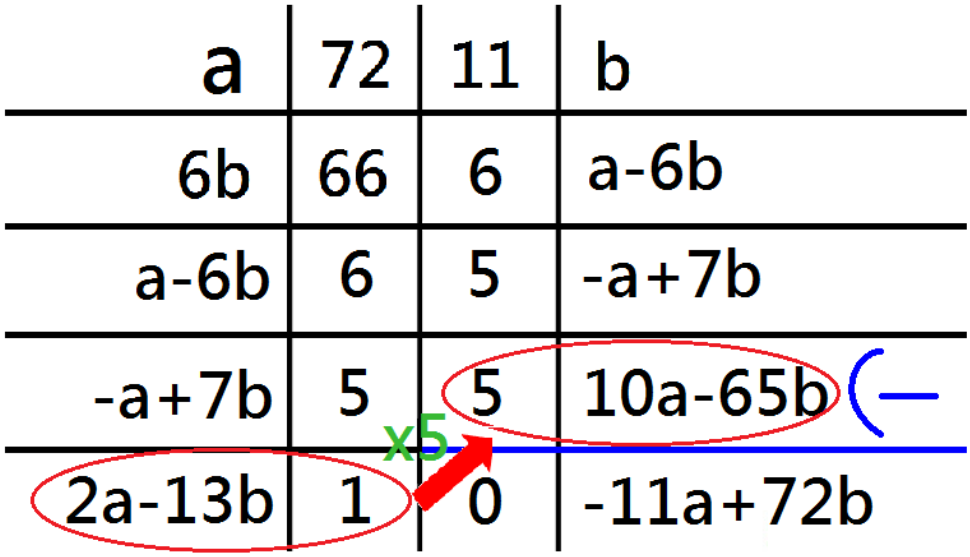


後面依此類推，

步驟三:



步驟四:



當 $2a-13b$ 整除 $10a-65b$ 時，$2a-13b$ 為 72 和 11 的最大公因數 1，

所以 $e=-13, k=2$ 。

然而 e 需大於 0 ，所以要加上 $\varphi(n)$ ，

$e=-13+\varphi(n)=-13+72=59$ 。

但此方法只適合手算， **擴展輾轉相除法舉例** 裡的方法較適合程式化。

RSA的細節

1. 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 e, d 應與 $\varphi(n)$ 互質，而 p 、 q 皆為超大奇質數，若 e 、 d 為偶數，e 或 d 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 $M = C^d \pmod{n}$ ，$M = C^d - kn, k \in \mathbb{N}$ ，若 M 大於 n ，M 有很多種可能，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 需要滿足 $1 < e < \varphi(n)$ 。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，無端造成硬體運算困擾。由於 $(e \mod \varphi(n))d$ 可寫成 $(e - k\varphi(n))d = ed - kd(\varphi(n))$ ，我們知道 $(e \mod \varphi(n))d \equiv ed \pmod{\varphi(n)} \equiv 1$。所以可以先將 e 模 $\varphi(n)$ 化簡。又若 $e = 1$ ，因為 $C \equiv M^e \equiv M^1 \pmod{n}$ ，其中 $M < n$ ，所以 $C = M^1$ ，根本沒加密，故 e 需大於 1。此性質對於 d 亦然。

5. p 、 q 需滿足 $p \neq q$。因為 $n = pq$ ，當 $p = q$，則 $n = p^2$ 。雖然理論上沒錯，但是實務上對於電腦來說計算根號 n 是很容易的，所以不安全。

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 $\varphi(n)$ ，再搭配公鑰 e ，就能推出 d。但是困難點在於把 n 質因數分解，現行的2048位元憑證 n 有617位數 $(\log_{10} 2^{2048} \approx 2048 * 0.301 \approx 616)$，要分解成兩個超大質數相乘難度超高。

Python CTF 解密工具實作

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，然後用擴展輾轉相除法算出私鑰。

爬蟲

觀察 factordb 網頁格式，利用 GET method 查詢質因數分解資料，以 BeautifulSoup 處理網頁資料，抓出我們想要的資訊。

```python
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 #回傳質數表
```

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

擴展輾轉相除法

剛才模反元素之證明中提到，當 a, b 為兩互質正整數，a, b 做輾轉相除法，

$$\begin{cases} b = aq_1 + r_1, & 0 \le r_1 < a \\ a = r_1 q_2 + r_2, & 0 \le r_2 < r_1 \\ r_1 = r_2 q_3 + r_3, & 0 \le r_3 < r_2 \\ \quad\vdots & \\ r_{m-2} = r_{m-1} q_m + 1, & 0 \le 1 < r_{m-1} \\ 1 = r_m q_{m+1} + 0 & \end{cases}, (q_1, q_2 \dots q_m, q_{m+1} \in \mathbb{Z}).$$

我們將上一項餘數整理後帶入下一式，發現每個餘數都是 a 與 b 的線性組合。事實上這就是擴展輾轉相除法，而我們也能稍微嗅出一點遞迴的味道。

我們能將餘數以 a, b 表示，$r_i = ax_i + by_i, 0 < i \leq m + 1$ 。

當 $i = 1$ ，$r_1 = ax_1 + by_1 = b - aq_1$ 。

當 $i = 2$ ，$r_1 = -aq_1 + b$ 帶入 $r_2 = a - r_1 q_2$ ，

我們有 $r_2 = ax_2 + by_2 = a - (-aq_1 + b)q_2 = a(1 + q_1 q_2) - bq_2$ 。

依此類推，我們列表分析，

i	x_i	y_i
1	$x_1 = -q_1$	$y_1 = 1$
2	$x_2 = 1 + q_1 q_2$	$y_2 = -q_2$
3	$x_3 = -q_1 - (1 + q_1 q_2)q_3$	$y_3 = 1 + q_2 q_3$
4	$x_4 = (1 + q_1 q_2) - [-q_1 + (1 + q_1 q_2)q_3]q_4$	$y_4 = -q_2 - (1 + q_2 q_3)q_4$
...	...	...

觀察規律，我們找到遞迴式

$$x_i, y_i = \begin{cases} x_1 = -q_1, \quad y_1 = 1 \\ x_2 = 1 + q_1 q_2, \quad y_2 = -q_2 \\ x_j = x_{j-2} - x_{j-1} q_j, \quad y_j = y_{j-2} - y_{j-1} q_j \quad , \quad (2 < j \leq m + 1) \end{cases} .$$

而 x_i, y_i 即為滿足一元二次不定方程 $ax + by = \gcd(a, b)$ 的正整數解。

我們能寫成 $\gcd(a, b) = ax_i + by_i$ 。

如此我們能建構出 bottom-up 的遞迴程式。

我們先做輾轉相除法求出 m ，因為我們的目標是 $ax_m + by_m = 1$ 。

算出 m 我們就知道遞迴要跑幾次，

```
a=int(input("a: "))
b=int(input("b: "))

if b<a:
    a,b=b,a #python 偷懶交換

a1=a
b1=b
q=[]
m=0
while(a1!=0): #輾轉相除法
    m+=1
    q.append(b1//a1) #將商存進list裡
    a1,b1=b1%a1,a1
#print(m)

#x前兩項
x=[]
x.append(-(b//a))
x.append(1+(b//a)*(a//(b%a)))

#y前兩項
y=[]
y.append(1)
y.append(-(a//(b%a)))

#x遞迴
for i in range(2,m-1):
    x.append(x[i-2]-x[i-1]*q[i])
#y遞迴
for i in range(2,m-1):
    y.append(y[i-2]-y[i-1]*q[i])

print(x[m-2],y[m-2])

#檢查是否 ax+by=1
print(a*x[m-2]+b*y[m-2])
```

然而，還有更快的方法，Top-down 遞迴。

遞迴做輾轉相除法時，

```
def gcd(a,b):
    if b==0: #終止條件 b==0
        return a
    else:
        return gcd(b,a%b)

```

令第 i 次遞迴的 a 為 a_i ， b 為 b_i 。

並設 $\gcd(a_i, b_i) = a_i * x_i + b_i * y_i$ 。

根據輾轉相除法， $\gcd(a_i, b_i) = \gcd(b_i, a_i \mod b_i) = \gcd(a_{i+1}, b_{i+1})$ ，

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

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

我們有

$x_i = y_{i+1}$,

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

(" [] " 為高斯符號，表示不大於高斯符號中之數的最大整數)

所以我們把輾轉相除法改寫成如此，

```
def extgcd(a, b):
     if b == 0: #終止條件 b==0
         return 1, 0 #此時 a==gcd(a,b)==1
     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
```

如此一來只有一個遞迴式，比前一個方法更快速。

最後再加上輸入輸出及 RSA 加密。

完整的程式碼

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

def factor(n):
    value={'query':str(n)}
    data=urllib.parse.urlencode(value)  #把數字編碼成query string
    addr="http://www.factordb.com/index.php?{url}".format(url=data)
    resp = urllib.request.urlopen(addr)  #GET請求
    soup=BeautifulSoup(resp,"html.parser")  #BeautifulSoup 處理網頁
    strlist=soup.find_all("font",color="#000000")
    primelist=[]
    for i in strlist: #將抓到的質因數放進list裡(搞不好不只兩個質因數)
        primelist.append(int(i.text))
    return primelist

def extgcd(a, b):
     if b == 0:
         return 1, 0
     else:
         x, y= extgcd(b, a % b)
         x, y = y, (x - (a // b) * y)
         return x, y

c,e,n=int(input("c:")),int(input("e:")),int(input("n:"))
primelist=factor(n)
factorlist=[]
n1=n
for p in primelist: #質因數分解
    while(not n1%p):
        factorlist.append(p)
        n1/=p
print(primelist,factorlist)
phi_n=1
print("\nprimes:")
for i in factorlist: #利用歐拉函數乘法性計算phi(n)
    print(i)
    phi_n*=i-1

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

if d<0: #檢查d是否<0
    d+=phi_n
print("\nd:",d)
m=pow(c,d,n) #c^d%n
print("message:",m)
```

感想

由資安一頭栽進密碼學，這是高一時的我想也沒想過的。只是起初因為體會到密碼學很困難，又聽到我資安的啟蒙老師說:「要做就做最難的，只要做出來，就可以贏過很多人。」，因而挑起這巨大的挑戰。從一開始不熟稔數學寫作，開始慢慢寫起這篇筆記，研讀密碼學，慢慢把不足的數學補足。我要感謝李佳哲老師願意花課餘與我討論數學，我從密碼學學到了很多高中較少接觸的數學，我在密碼學中找到了數學、資訊與我的連結。我也要感謝許嘉麟學長能包容我當時不成熟的數學寫作，願意幫我審稿，提點意見，與學長討論後我有了更廣泛的視野。而現今高三學了更多知識的我，再回頭看看過去的筆記，又有完全不同的感覺，不漂亮、不完整的瑕疵頓時出現眼前 。 所以我以高三更成熟的 筆，重新修改，希望能帶來更加易讀的文章。 不過，我還要學的東西還有很多，我接觸到的只是密碼學裡的冰山一角，也許再過個幾年，再次讀這篇筆記，又會有不同的感覺吧。

補充資料

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

[2] proof of Euler-Fermat theorem using Lagrange's theorem
https://planetmath.org/proofofeulerfermattheoremusinglagrangestheorem

參考資料

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

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

[3] 歐拉函數

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

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

[5] 歐拉定理，費馬小定理證明

https://codertw.com/%E7%A8%8B%E5%BC%8F%E8%AA%9E%E8%A8%80/537802/

[6] 擴展歐幾里得算法

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

[7] 模反元素

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

[8] 群

https://zh.wikipedia.org/wiki/%E7%BE%A4

[9] 模 n 乘法群

https://zh.wikipedia.org/wiki/%E6%95%B4%E6%95%B0%E6%A8%A1n%E4%B9%98%E6%B3%95%E7%BE%A4

[10] 沈淵源:《不可能的任務─公鑰密碼傳奇》，三民，2016年。

如有謬誤，請不吝指教