RSA 原理與實作

蕭擎軒

前言

核心目標有三：

- 其一在於深入理解 RSA 演算法背後數學原理。
- 其二在於具體實作 RSA 演算法細節與程式碼。
- 其三在於分析 RSA 複雜度並且在具體實作上提出改進辦法，增加效能與可行性。

目錄

Part 1：RSA 介紹

密碼學發展

- 密碼學的發展歷史悠久，其中以第二次世界大戰為分界，可以分為「古典密碼」與「現代密碼」。

- 其中現代密碼又可以分為「對稱加密」與「非對稱加密」兩種主流方法。

- 此次主題「 RSA 演算法」便是屬於現代密碼學中的「非對稱加密」。

密碼學發展

古典密碼學

現代密碼學

對稱加密

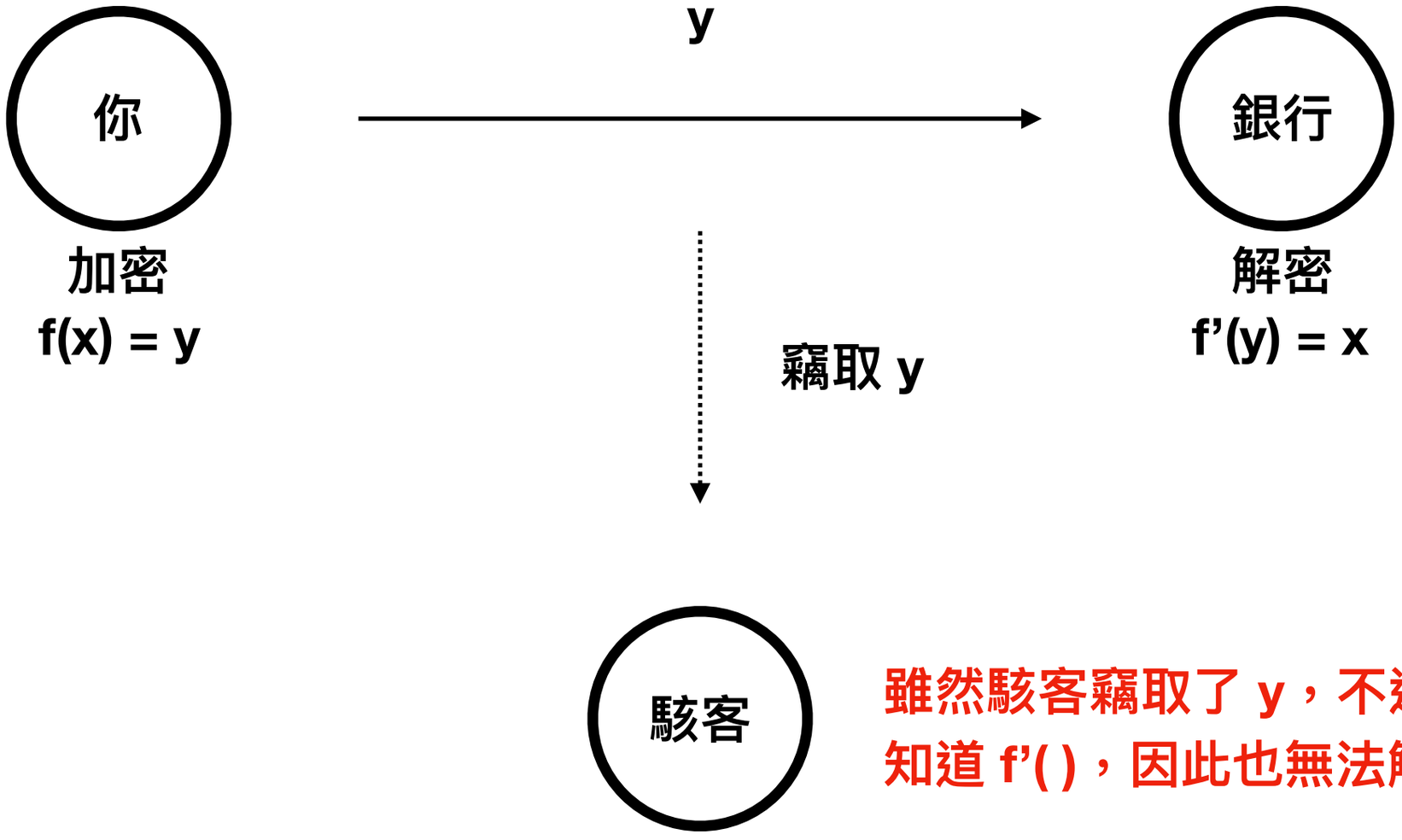
非對稱加密

密碼學與數學

- 定義 f(x) 為一函數，f: X (明文) → Y (密文)
    - f is bijection (one to one, onto)
        - f’(f(x)) = x, f(f’(y)) = y
- 加密 (encode) : f(x) = y
- 解密 (decode) : f’(y) = x
- f(x) 為一反函數

從數學的角度來說，密碼學就是在設計一種 Bijection Function

密碼學與數學



對稱加密 vs 非對稱加密

- 那麼「對稱加密」與「非對稱加密」又有什麼差別呢？

- 差別在於「對稱加密」中加密人與解密人共用使用同一把鑰匙，一但鑰匙洩漏，則整個系統就不安全了。

- 反之，「非對稱加密」加密人與解密人使用了不同的鑰匙（公鑰、私鑰），雖然設計上比較複雜，但是安全許多，因此「非對稱加密」是現今世界上最主流的加密方法。

對稱加密 vs 非對稱加密

	對稱加密	非對稱加密
金鑰	key = f = f’	public key = f private key = f’
複雜度	低	高
著名演算法	DEA, Enigma	RSA, 橢圓曲線

RSA

- RSA 是「非對稱加密」中最經典的方法，其餘的非對稱加密演算法還有橢圓曲線等等...

- RSA 的原理是基於：質因數無法在線性時間內被分解。亦即：質因數愈難被分解，RSA 演算法愈可靠。

- 其發明者：Ron Rivest, Adi Shamir, Leonard Adleman 因此獲得了 2002 年的圖靈獎。

Part 2：RSA 原理

RSA 演算法

1. 亂數生成兩相異質數 p, q

2. 令 $\begin{cases} n = p * q \\ \phi(n) = \phi(p) * \phi(q) = (p-1)(q-1) \text{ (歐拉函數)} \end{cases}$

3. 令 e 為加密鑰，e 須滿足 $\begin{cases} e < \phi(n) \\ gcd(\phi(n), e) = 1 \end{cases}$

4. 令 d 為解密鑰，d 須滿足：$e * d = \phi(n)k + 1, k \in Z \rightarrow d = \frac{\phi(n)k+1}{e}$

5. 令 M 為明文，C 為密文 $\rightarrow \begin{cases} M^e \pmod{n} \equiv C \text{ 即可完成加密} \\ C^d \pmod{n} \equiv M \text{ 即可完成解密} \end{cases}$

相關引理

在理解 RSA 的原理以前，必須先介紹以下 6 項引理：

1. 算術基本定理
2. 歐幾里德演算法
3. 模反元素
4. 歐拉函數
5. 費馬小定理
6. 中國餘式定理

RSA 的證明相當複雜，更嚴謹的證明可以參考離散數學、代數課本

1. 算術基本定理

每個大於 1 的自然數，若非質數，則可以寫為 2 個以上的質數乘積，且所有的質因數由小排到大後具唯一性。

$$\forall a \in N, a > 1, \exists \prod_{i=1}^{n} p_i^{b_i}, \forall p_i \text{ are prime and } p_1 < p_2 < \ldots < p_n, b_i \in Z^+$$

2. 歐幾里德演算法

$\forall m, n, q, r \in Z$, if $m = nq + r$, then: $gcd(m, n) = gcd(n, r)$

$$\begin{cases} gcd(n, 0) & \text{, if } n > 0 \\ gcd(m, n) = gcd(m, r) & \text{, if } m > n \end{cases}$$

「歐幾里德演算法」其實就是「輾轉相除法」，
其可以用來尋找最大公因數、兩數的線性組合。

2. 歐幾里德演算法

證明： $gcd(m,n) = gcd(n,r)$

令： $gcd(m,n) = g_1, gcd(n,r) = g_2$

$\because g_1 | m$ and $g_1 | n$

$\therefore g_1 | (m - nq)$

$\rightarrow g_1 | r$

$\because g_2$ is the greatest common divisor of n, r

$\therefore g_2 \geq g_1$

$\because g_2 | n$ and $g_2 | r$

$\therefore g_2 | (nq + r)$

$\rightarrow g_2 | m$

$\because g_1$ is the greatest common divisor of m, n

$\therefore g_1 \geq g_2$

$\because g_2 \geq g_1$ and $g_1 \leq g_2$

$\therefore g_1 = g_2$

$\rightarrow gcd(m,n) = gcd(q,r)$

3. 模反元素

定義 : $\forall a \in Z, n \in Z^+$, if $\exists x$ st $ax \equiv 1$ mod(n) , x 即為 a 在 n 的乘法反元素 : a^{-1} mod(n)

證明 : $\forall a \in Z, n \in Z^+$, if $gcd(a, n) = 1$, then $\exists x$ st $ax \equiv 1$ mod(n)

$\because gcd(a, n) = 1$

$\therefore \exists x, y \in Z$ st $ax + ny \equiv 1$ （由輾轉相除法推得）

$\rightarrow ax + ny \equiv 1$ (mod n)

$\rightarrow ax \equiv 1$ (mod n)

$\rightarrow x \equiv a^{-1}$ (mod n)

4. 歐拉函數

定義：$\phi(n)$: { $1, 2 \ldots n-1$ } 中與 n 互質的元素個素

$$\forall n \in Z, n = {p_1}^{e_1} {p_2}^{e_2} {p_3}^{e_3} \ldots {p_k}^{e_k} \ (\forall i \in Z, i \leq k, p_i \text{ is prime})$$

$$\phi(n) = n \prod_{i=1}^{k} (1 - \frac{1}{p_i})$$

4. 歐拉函數

證明：$\phi(n) = n - 1 \iff n$ is prime

$\Rightarrow$

prove by contradiction, suppose n is not prime

$\exists m \in Z^+, 1 < m < n$ st $m|n$

$\rightarrow \phi(n) < n - 1$ 矛盾

$\rightarrow$ n is prime

$\Leftarrow$

$\because gcd(m, n) = 1, \forall m = 1, 2, \ldots n - 1$

$\therefore \phi(n) = n - 1$

5. 費馬小定理

$\forall m \in Z, p$ is prime, st $gcd(m, p) = 1 \rightarrow m^{p-1} \equiv 1 \text{ (mod p)}$

證明：$m^{p-1} \equiv 1 \text{ (mod p)}$

考慮以下 p - 1 個數：

$x_1 = m \text{ (mod p)}$

$x_2 = 2m \text{ (mod p)}$

$\ldots$

$x_{p-1} = (p-1)m \text{ (mod p)}$

$\rightarrow \{x_1, x_2, \ldots x_{p-1}\} \subset \{0, 1, 2, \ldots p-1\}$

$\because gcd(m, p) = 1$

$\therefore p \nmid km, \forall k = 1, 2, \ldots p-1$

$\rightarrow x_i \neq 0, \forall i = 1, 2, \ldots p-1$

$\rightarrow \{x_1, x_2, \ldots p-1\} \subseteq \{1, 2, \ldots p-1\}$

5. 費馬小定理

引理：

$\forall a, b, c, n \in Z,$ if $gcd(c, n) = 1$ then:

$ac \equiv bc \pmod{n} \iff a \equiv b \pmod{n}$

$\Rightarrow$

$\because ac \equiv bc \pmod{n}$

$\therefore n \mid ac - bc$

$\rightarrow n \mid (a - b)c$

$\because gcd(c, n) = 1$

$\therefore n \mid a - b$

$\rightarrow a \equiv b \pmod{n}$

$\Leftarrow$

$\because a \equiv b \pmod{n}$

$\therefore n \mid a - b$

$\rightarrow n \mid (a - b)c$

$\rightarrow ac \equiv bc \pmod{n}$

5. 費馬小定理

claim: $\forall i \neq j, x_i \neq x_j$ st $\{x_1, x_2, \ldots p-1\} = \{1, 2, \ldots p-1\}$

prove by contradiction, suppose $\exists i \neq j, x_i = x_j$

$\rightarrow im \equiv jm$ (mod p)

$\because gcd(m, p) = 1$

$\therefore i \equiv j$ (mod p) 矛盾

$\rightarrow \forall i \neq j, x_i \neq x_j$ st $\{x_1, x_2, \ldots p-1\} = \{1, 2, \ldots p-1\}$

$\because 1m * 2m \ldots (p-1)m \equiv 1 * 2 \ldots (p-1)$ (mod p)

$\therefore (p-1)!m^{p-1} \equiv (p-1)!$ (mod p)

將左右兩式同除以 : $(p-1)!$

$\rightarrow m^{p-1} \equiv 1$ (mod p)

6. 中國餘式定理

$\forall n_i \in N, r_i \in Z, i = 1, 2, \ldots k,$ 其中 $n_1, n_2 \ldots n_k$ 彼此互質，則：

$$x \equiv r_1 \pmod{n_1}$$
$$x \equiv r_2 \pmod{n_2}$$
$$\ldots$$
$$x \equiv r_k \pmod{n_k}$$

在模 $n = n_1 n_2 \ldots n_k$ 有唯一解

6. 中國餘式定理

證明：

令：$N_j = \frac{n}{n_j} = \prod_{i \neq j} n_i, j = 1, 2 \ldots k$

$\because n_1, n_2 \ldots n_k$ 彼此互質

$\therefore gcd(N_j, n_j) = 1, \forall j = 1, 2 \ldots k$

$\rightarrow N_j M_j \equiv (\bmod n_j), \forall j = 1, 2 \ldots k$

取 $x \equiv \sum_{j=1}^{k} r_j M_j N_j$ (mod n)

$\because n_i | N_j, \forall i \neq j$

$\therefore r_j M_j N_j \equiv 0 (\bmod n_i), \forall i \neq j$

$\rightarrow x \equiv \sum_{j=1}^{k} r_j M_j N_j \equiv r_i M_i N_i \equiv r_i (\bmod n_i)$

唯一性：

令：$x_1 \equiv x_2 \equiv r_j (\bmod n_j), \forall j = 1, 2 \ldots k$

$\rightarrow n_j | (x_1 - x_2), \forall j = 1, 2 \ldots k$

$\because n = n_1 n_2 \ldots n_k$

$\therefore n | (x_1 - x_2)$

$\rightarrow x_1 \equiv x_2$ (mod n)

RSA 證明

證明：
$$\begin{cases} M^e \pmod{\text{n}} \equiv C \\ C^d \pmod{\text{n}} \equiv M \end{cases}$$

已知：
$$\begin{cases} n = pq \text{ (p, q 皆為質數)} \\ \phi(n) = \phi(p) * \phi(q) = (p-1)(q-1) \\ gcd(\phi(n), e) = 1 \end{cases}$$

$$\because gcd(\phi(n), e) = 1$$
$$\therefore \exists d \text{ st } ed \equiv 1 \pmod{\phi(n)}$$
$$\rightarrow e^{-1} \equiv d \pmod{\phi(n)}$$

$$\because ed \equiv 1 \pmod{(p-1)(q-1)}$$
$$\therefore \exists k \in Z \text{ st } ed = k(p-1)(q-1) + 1$$

RSA 證明

令 $C \equiv M^e \pmod{n}$

根據費馬小定理

$\because gcd(M, p) = 1$ and $gcd(M, q) = 1$

$\therefore M^{(p-1)} \equiv 1 \pmod{p}$ and $M^{(q-1)} \pmod{q}$

$\rightarrow C^d \equiv (M^e)^d \equiv M^{ed} \equiv M^{k(p-1)(q-1)+1} \equiv (M^{(p-1)})^{k(q-1)} M \equiv (1)M \equiv M \pmod{p}$

$\rightarrow C^d \equiv (M^e)^d \equiv M^{ed} \equiv M^{k(q-1)(p-1)+1} \equiv (M^{(q-1)})^{k(p-1)} M \equiv (1)M \equiv M \pmod{q}$

根據中國餘式定理

$\because C^d \equiv M \pmod{p}$ and $C^d \equiv M \pmod{q}$

$\therefore C^d \equiv M \pmod{pq}$ 有唯一解

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

Part 3 : RSA 實作

RSA 實作

介紹完 RSA 的原理，終於要開始實作 RSA 了。實做 RSA 需要完成以下 7 個子程式：

1. 質因數分解
2. 歐幾里德演算法（最大公因數）
3. 歐拉函數
4. 轉碼函數
5. 生成公鑰、私鑰
6. 加密函數
7. 解密函數

1. 質因数分解

In [3]:
```
def factorize2(x):
    fact_list = list()
    if sp.isprime(x) == False:
        fact = 1
        while fact != 0:
            fact, x = factorize(x)
            if fact != 0:
                fact_list.append(fact)
        if x != 1:
            fact_list.append(x)
    else:
        fact_list.append(x)
    return fact_list

print(factorize2(1024))
```

```
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
```

2. 歐幾里德演算法

In [4]:
```
def greatest_common_divisor(a, b):
    if b == 0:
        return a
    else:
        return greatest_common_divisor(b, a % b)

print(greatest_common_divisor(4, 6))
```

```
2
```

3. 歐拉函數

In [5]:
```
def euler_function(p, q):
    return (p - 1) * (q - 1)

print(euler_function(3, 5))
```

```
8
```

4. 轉碼函數

- 在進行加密前必須先進行前置作業：轉碼，也就是說「文字」必須先轉換成「數字」才能進行加密。

- 為了能夠支援中文，在此選擇使用「UTF-8」。

- 另外，一般而言 RSA 都是採用「二進位」加密，不過為了理解方便，在此選擇使用「十進位」進行加密。

UTF-8 → 十進位

In [6]:
```
def string_to_int(string):
    byte = string.encode()                  # string to byte
    hexadecimal = binascii.hexlify(byte) # byte to hex
    decimal = int(hexadecimal, 16)          # hex to dec
    return decimal

string = '你好！'
integer = string_to_int(string)
print(integer)
```

```
27086628833817823194713031788698839469377765346622593
```

轉碼的過程本身其實就是一種「對稱加密」。

十進位 → UTF-8

In [7]:
```
def int_to_string(int):
    hexadecimal = hex(int)                  # dec to hex
    byte = hexadecimal[2:].encode()         # hex to byte (type)
    byte = binascii.unhexlify(byte)         # hex to byte (coding)
    string = byte.decode('utf-8')           # byte to utf-8
    return string

message = int_to_string(integer)
print(message)
```

```
Hello World! 你好！
```

5. 加密函數

In [8]:
```
def encode(plaintext, public_key, n):
    cyphertext = (plaintext ** public_key) % n
    return cyphertext
```

6. 解密函數

- 版本一：直接解密

In [*]:
```python
def decode(cyphertext, private_key, n):
    plaintext = (cyphertext ** private_key) % n
    return plaintext

print(decode(19, 2 ** 100, 7))
```

直接計算：19 ^ (2 ^ 100) (mod 7)
對於一般電腦來說並不容易解密。

6. 解密函數

- 版本二：先將私鑰分解後再解密

版本一的做法是直接對私鑰進行解密，即計算：$C^d \pmod n \equiv M$。這條方程式理論上可行，也有唯一解，但其實並不實際。原因在於當 C, d, n 都很大時，解密要花非常久的時間，且一般電腦有可能發生溢位。但如果能先將 d 分解，即：$d = d_0 d_1 \ldots d_n$ 則只要計算出：$(((C^{d_0} \pmod n)^{d_1} (\mod n))\ldots)^{d_n} \pmod n$ 就可以省下非常多的時間，且較不容易發生益位。

證明：$\forall m, n, a, b \in Z, m^{ab} \equiv (m^a \pmod n)^b \pmod n$

令：$m = nq + r$

$\rightarrow m^{ab} \equiv (nq + r)^{ab} \equiv r^{ab} \pmod n$

$\rightarrow (m^a \pmod n)^b \equiv ((nq + r)^a \pmod n)^b \equiv r^{ab} \pmod n$

6. 解密函數

In [28]:
```
def decode2(cyphertext, private_key_fact_list, n):
    for i in private_key_fact_list:
        cyphertext = (cyphertext ** i) % n
    plaintext = cyphertext
    return plaintext

print(decode2(19, [2 for i in range(100)], 7))
```

```
2
```

先將 2 ^ 100 分解為： 2 * 2 ... 2 ，
批次 mod 7 即可有效降低解密時間。

7. 生成公鑰、私鑰

In [11]:
```
def generate_key():
    public_key, private_key = 0, 0
    prime = list(sp.primerange((10 ** 3), (10 ** 4)))
    while (public_key == 0) or (private_key == 0):
        p, q = random.choices(prime, k=2)                    # 隨機生成兩質數p, q
        n = p * q                                            # n = p * q
        o = euler_function(p, q)                             # 歐拉函數
        public_key, private_key = factorize(o * 10 + 1) # 生成公鑰、私鑰
        gcd = greatest_common_divisor(public_key, o)
        remainder = public_key * private_key % o
        print('gcd: %i' % gcd)
        print('remainder: %i' % remainder)
        if gcd == 1 and remainder == 1:                      # 檢查是否互質
            print('generate key succes\n')
            break
        else:
            print('generate key fail\n')
    return p, q, public_key, private_key, n
```

7. 生成公鑰、私鑰

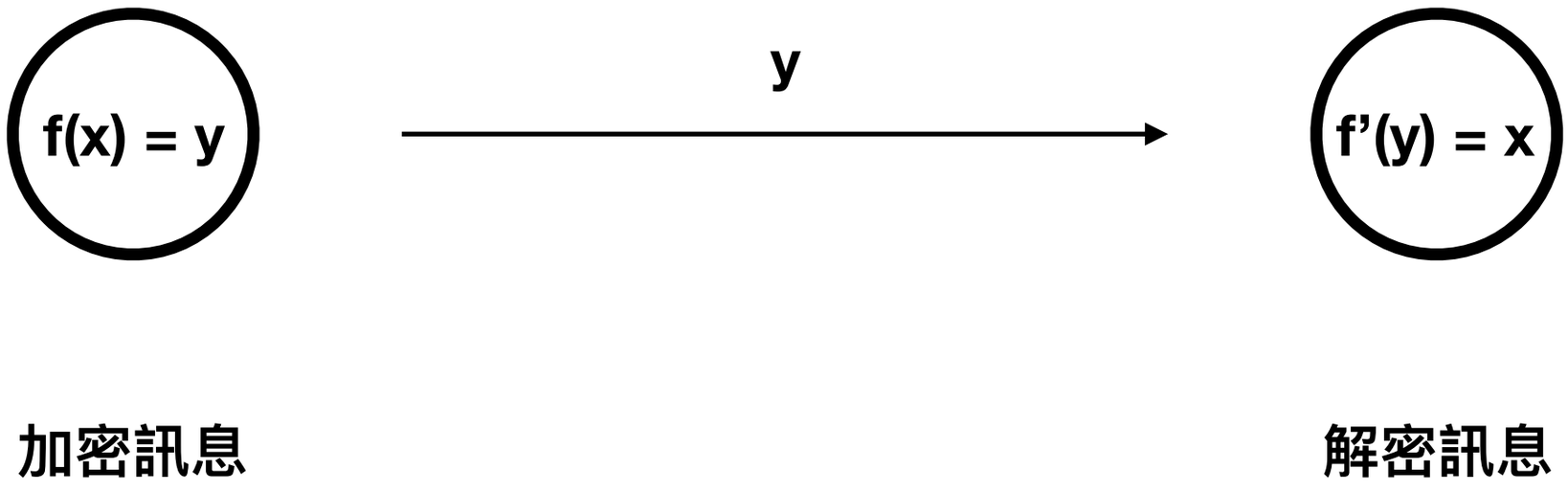
In [12]:
```
p, q, public_key, private_key, n = generate_key()
private_key_fact_list = factorize2(private_key)   # 分解 private key
print('p: %i' % p)
print('q: %i' % q)
print('public key: %i' % public_key)
print('private key: %i = %s' % (private_key, str(private_key_fact_list)))
print('n: %i' % n)
```

```
gcd: 1
remainder: 1
generate key succes

p: 4093
q: 6271
public key: 43
private key: 5966707 = [1721, 3467]
n: 25667203
```

RSA 測試

- 以上流程都實作完成後，接下來將進行實際測試：



加密訊息

In [20]:
```
plaintext = 'Hello World!'
plaintext_list = list()
cyphertext_list = list()
for i in plaintext:
    integer = string_to_int(i)
    plaintext_list.append(integer)
    cyphertext = encode(integer, public_key, n)
    cyphertext_list.append(cyphertext)
print('訊息：%s' % str(plaintext))
print('明文：%s' % str(plaintext_list))
print('密文：%s' % str(cyphertext_list))
```

```
訊息：Hello World!
明文：[72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]
密文：[12605358, 21856033, 20406178, 20406178, 13297971, 10341798, 22576247, 13297971, 5453566, 20406178, 4524525, 21453017]
```

加密「Hello World」

解密訊息

In [21]:
```
message = ''
for i in cyphertext_list:
    plaintext = decode2(i, private_key_fact_list, n)
    message += int_to_string(plaintext)
print('密文：%s' % str(cyphertext_list))
print('明文：%s' % str(plaintext_list))
print('訊息：%s' % str(message))
```

```
密文：[12605358, 21856033, 20406178, 20406178, 13297971, 10341798, 2257624
7, 13297971, 5453566, 20406178, 4524525, 21453017]
明文：[72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 33]
訊息：Hello World!
```

解密成功

Part 4 : RSA 分析

RSA 複雜度分析

- RSA 的安全性是基於現階段傳統電腦並沒有任何演算法能夠在線性時間內 $\mathcal{O}(n^k)$ 分解 $n = pq$。以最簡單的演算法「暴力法」來說：當我們要對 n 進行分解，只要嘗試將 n 除以 $2, 3, 5 ... \frac{\sqrt{n}}{2}$ 的質數即可，即求：$\sum_{i=2}^{\frac{\sqrt{n}}{2}} \mathcal{O}(k)$ ，其複雜度亦會隨著 n 的上升而無法收斂：$\lim_{n\to\infty} \sum_{i=2}^{\frac{\sqrt{n}}{2}} \mathcal{O}(k) = \infty$。

RSA 發展近況

- 現階段最好的質因數分解演算法為 GNFS。然而其複雜度為：$\theta(exp(\frac{64n}{9})^{\frac{1}{3}}(logn)^{\frac{2}{3}})$ ，同樣也無法在線性時間內完成分解。

- 2009/12/12，RSA-768（768 bits, 232 digits）被成功分解。這一事件威脅了現階段 RSA-1024 的安全性，因此 RSA 未來勢必會更新到 2048 或以上。

- 總結來說，只要量子電腦尚未普及，現階段只要增加 n 的位元組即可保障 RSA 的安全性。

Part 5：附錄

參考

- Discrete Mathematics and Its Applications – Rosen
- 離散數學 – 黃子嘉
- Wikipedia – RSA

程式碼

https://github.com/wuling31715/mathematics/blob/master/rsa.ipynb