B10730027 四資工三甲 曾瑄翎

RSA Implementation

建置環境 / 依賴套件

- **Python3+**
- **random**

操作方式

```
python rsa.py
```

Input

1. `Enter first prime number p`
2. `Enter second prime number q`
3. `Enter the plaintext (less than p*q)`

Output

1. `The encrypted number`
2. `The decrypted number`

執行結果範例圖

```
PS C:\Users\Lynn\Desktop\IS\hw4> python rsa.py
Enter first prime number p: 3
Enter second prime number q: 97
Enter the plaintext (less than p*q): 270
The encrypted number is:  231
The decrypted number is:  270
PS C:\Users\Lynn\Desktop\IS\hw4>
```

程式碼解說

大綱

> functions using in **generate_keys**

```
> is_prime
> gcd
> generate_d
```

> 產生公鑰、私鑰

```
> generate_keys
```

> 加解密定義

```
> encrypt
> decrypt
```

RSA 算法

1. 計算n=pq，p、q為輸入之質數

```python
    # p和q為質數
    if (is_prime(p) and is_prime(q)):

        # p is not equal to q
        if not p == q:

            # 計算n=pq
            n = p * q
```

2. 計算f(n)=(p-1)(q-1)

```
        # 計算f(n)=(p-1)(q-1)
        f = (p-1) * (q-1)
```

3. 找一個與f(n)互質的數e，且1<e<f(n)

```
        # 找一個隨機數e，且1<e<f(n)
        e = random.randrange(1, f)

        # 確保隨機數e與f(n)互質
        g = gcd(e, f)
        while g != 1:
            e = random.randrange(1, f)
            g = gcd(e, f)
```

4. 計算d，使得$d \equiv e^{-1} \bmod f(n)$

```
        # generate d for private key
        d = generate_d(e, f)
```

使用擴展歐幾里德算法

```
def generate_d(e, f):

    a = e
    b = f
    x1, x2 = 0, 1
    y1, y2 = 1, 0

    while (b != 0):
        quotient = a // b
        a, b = b, a - quotient * b
        x2, x1 = x1, x2 - quotient * x1
        y2, y1 = y1, y2 - quotient * y1

    if (x2 < 0):
        return x2 + f
    else:
        return x2
```

5. 公鑰=(e,n)，私鑰=(d,n)

```
        # public key is (e, n) and private key is (d, n)
        return ((e, n), (d, n))
```

6. 設明文為P、密文為C，則加密過程為：$C \equiv P^{e} \bmod n$

```python
def encrypt(public_key, plaintext):

    e, n = public_key

    # plaintext 需小於 n=pq
    if(int(plaintext) < n):
        ciphertext = pow(int(plaintext), e, n)
        return ciphertext

    else:
        print("plaintext is too big")
        exit()
```

7. 設明文為P、密文為C，則解密過程為：$P \equiv C^{d} \bmod n$

```python
def decrypt(private_key, ciphertext):

    d, n = private_key
    plaintext = pow(int(ciphertext), d, n)

    return plaintext
```

困難與心得

除了RSA算法之外，花最多時間理解「計算模反元素d」需要的擴展歐幾里得算法。

計算 e 對於 φ(n) 的模反元素：

$ed \equiv 1 \pmod{\varphi(n)}$ 等價於 $ed - 1 = k\varphi(n)$

因此，要找到模反元素d，其實就是對下面的二元一次方程用擴展歐幾里得算法求解：

$ex + \varphi(n)y = 1$

擴展歐幾里得算法（Extended Euclidean algorithm）是輾轉相除法的擴展，已知兩個數a和b，對它們進行輾轉相除，可得它們的最大公因數；而擴展歐幾里得算法還利用了帶餘除法所得的商，在輾轉相除的同時也能得到貝祖等式中的x、y兩個係數。