DSA 實作 [張家成]

小組分工
A10615003 張家成 二資工四 —> Generate Key & Signature Generation
A10615001 趙梓丞 二資工四 —> Signature Verification

Digital Signature Algorithm 實作 [Java]

- 建置環境

- MacOS Mojave 10.14.1
- Eclipse IDE for Java Developers [Version: Oxygen]
- Java SE 9.0.1

- 執行結果截圖

- **註釋 —output test— 後的結果**

```
<DSA>
Generating big prime number [p](1024bits) [q](160bits)...
p = 179135506188397394456488777968711364282605397472469064883251776739689169703076197772746733178466886531963310271288232181774 6
bitLength = 1024

q = 1456348864671469590400938145357403601317861112883
bitLength = 160

Picking [d] [k]... and Calculating [a] [y] [r]...
a = 2^((p - 1)/q) mod p = 121153685963471171229866537273163707938770750378345373918295859914245318605612437905914177307507886849
bitLength = 1024

d is Private key (0<d<q) = 70364969800313560917658211321
bitLength = 96

y is Public key = (a^d mod p) = 392050802069362094200272720045987624419293768283199701121470785857983375929430202128089104064352
bitLength = 1022

k is an random key (1<k<q) = 6899827861052179
bitLength = 53

r = ( (a^k mod p) mod q ) = 137818255197938886405768233425258255278759616016 8
bitLength = 160

Message is "myDSAbooo"
Calculating SHA1(m) and converting to Decimal...
SHA1(m)[Hex] = fcca1c13b462ef9ccd8693f863268a68446b5aea
h (SHA1(m)[Dec]) = 144317287062685655101859640061723308764789652554 6
bitLength = 160

Calculating [s]...
s = ( (SHA1(m)+d*r)*k^(-1) mod q ) = 22454764031292716449822077882256912193698495405 0
bitLength = 158

Signature Generation Finished
```

- 執行時間

- **最快2s左右，最慢可能會20s左右,平均10s內（上述的建置環境中）**

- 困難與心得

- 1024bits長度的素數「p」，如何找到160bits長度的素數「q」，並且「q」是「p-1」的素因子。
  - 解決方案：一開始嘗試先生成1024bits的素數「p」，得到p-1之後在通過計算（例如：短除法）找到160bits的素因子「q」，但發存在很多問題，例如規定160bits長度以及大量的計算都是難點。於是嘗試逆向思維：先找到160bits長度的素數「q」，再通過不斷的乘以「2」，讓其結果長度到達1024bits為止，然後將得到結果+1，會得到一個奇數，再通過費馬測試判定這個數是否是素數，如果是，則我們就找到了「p」，和「p-1」，如果不是，則重新生成一個160bits長度的素數「q」，循環上述操作，直到找到1024bits的素數「p」為止。
- 在上述找「p」和「q」的方法中，存在一個「不斷乘以『2』」的操作，將如果將160bits的素數不斷乘以「2」讓其達到1024bits，將會乘大約「864」次「2」，這樣將會消耗大量計算機資源，且程式運行速度非常慢。
  - 解決方法：將需要操作的數「q」轉換為二進制（binary），並且使用向左shift的操作（在二進制數末位添加0），來達到乘以「2」的效果，最後將得到的二進制數轉換為十進制（decimal）即可。

```java
public static BigInteger[] get_p_q() {
    while (true) {
        //random Generate a 160bits length prime q
        BigInteger q = pickPrime(160);
        //converting to binary string
        String qStr = q.toString(2);
        BigInteger p_1 = BigInteger.ZERO;
        // 160bits + 864 "0" = 1024 bits means multiply 864 times 2
        String p_1Str = qStr
                + "000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
                + "000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
                + "000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
                + "000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
                + "000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
                + "000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
                + "000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
                + "000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000";
        p_1 = new BigInteger(p_1Str, 2);
        //we have a 1024bits length number p
        BigInteger p = p_1.add(BigInteger.ONE);
        //check it is prime or not
        if (fermatTest(p, 10)) { // test 10 times
            return new BigInteger[] { p, q };
        }
    }
}
```