

# 密码学实验五实验报告

姓名：孟衍璋 学号：16337183

## 一、 实验目的

实现 RSA 算法。

## 二、 实验内容

将自己学号作为私钥，并生成对应的公钥。加密：SUN YAT-SEN UNIVERSITY（包含空格），给出密文，并给出解密函数，进行验证。

## 三、 实验及算法原理

首先，需要考虑的第一件事就是如何生成 512 位的大素数。基本思路为，先随机生成一个 512 位的伪素数，确保其为奇数，再利用 Miller-Rabin 素性检测判断该数是否为素数，如果不是，在附近不断+2 应该就能很快地找到一个素数。

其中生成伪素数的代码为：

```
def proBin(w): # w为随机生成的伪素数的比特数
    Pseudo_prime = ""
    Pseudo_prime += '1'
    for i in range(w - 2):
        c = random.randrange(0,2)
        Pseudo_prime += str(c)
    Pseudo_prime += '1'

    return Pseudo_prime
```

用 Miller-Rabin 素性检测来判断生成的伪素数是否为素数：

```

def is_probable_prime(n, trials = 5): # n为十进制
    assert n >= 2
    # 2是素数~
    if n == 2:
        return True
    # 先判断n是否为偶数，用n&1==0的话效率更高
    if n % 2 == 0:
        return False
    # 把n-1写成(2^s)*d的形式
    s = 0
    d = n - 1
    while True:
        quotient, remainder = divmod(d, 2)
        if remainder == 1:
            break
        s += 1
        d = quotient
    assert(2 ** s * d == n - 1)

    # 测试以a为底时，n是否为合数
    def try_composite(a):
        if pow(a, d, n) == 1: # 相当于(a^d)%n
            return False
        for i in range(s):
            if pow(a, 2 ** i * d, n) == n - 1: #相当于(a^((2^i)*d))%n
                return False
        return True # 以上条件都不满足时，n一定是合数

    # trials为测试次数，默认测试5次
    # 每次的底a是不一样的，只要有一次未通过测试，则判定为合数
    for i in range(trials):
        a = random.randrange(2, n)
        if try_composite(a):
            return False

    return True #通过所有测试，n很大可能为素数

```

最后得到生成的素数：

```

def proPrime(w): # w为生成的素数的比特数
    count = 0
    a = proBin(w) # 返回的a为二进制
    a_int = int(a, base = 2) # 将a转换为十进制
    for i in range(1000):
        count += 1
        if is_probable_prime(a_int) == 1:
            print("trial:", count)
            print("The prime number generated is:", a_int)
            return a_int
        a_int += 2
    print("can not find a prime.")

```

得到生成的 512 位的大素数  $p$  与  $q$  之后，接下来要做的事就是生成公钥与私钥。其中私钥就是  $p$ 、 $q$  与  $a$ ，且选取自己的学号作为  $a$  的值。再根据私钥  $a$  生成公钥  $b$ ，这里需要计算  $a*b \equiv 1 \pmod{\Phi(n)}$ ，即  $a^{-1} \pmod{\Phi(n)}$ ，根据书上的乘法逆的算法，实现如下：

```
# 计算乘法逆  $a^{-1} \pmod{\Phi(n)} = t \pmod{\Phi(n)}$ 
def multiplicative_inverse(n,a):
    a0 = n
    b0 = a
    t0 = 0
    t = 1
    q = a0 // b0
    r = a0 - q * b0
    while r > 0:
        temp = (t0 - q * t) % n
        t0 = t
        t = temp
        a0 = b0
        b0 = r
        q = a0 // b0
        r = a0 - q * b0
    if(b0 != 1):
        pass
    else:
        return t
```



生成私钥与公钥之后，便可以开始加密与解密。加密与解密都是模指数运算，即  $x^c \pmod{n}$ 。如果  $c$  非常大，效率就会十分低下，而使用平方-乘算法，可以把计算  $x^c \pmod{n}$  所需模乘的次数降低为最多  $2L$  次，其中  $L$  是  $c$  的二进制表示的比特数。于是  $x^c \pmod{n}$  可以在时间  $O(Lk^2)$  内算出。其中平方-乘算法实现如下：

```
# 平方-乘算法，计算(base ^ exponent) mod n
def square_and_multiply(base, exponent, n):
    # 将exponent化为二进制存入bin_exponent中。
    # 切片[2:]是因为bin()函数作用后前面会有0b
    bin_exponent = bin(exponent)[2:]
    # print("exponent:", exponent)
    # print("bin_exponent:", bin_exponent)
    length = len(bin_exponent)

    z = 1
    for i in range(length):
        z = (z ** 2) % n
        # 从高位访问到低位
        if int(bin_exponent[i]) == 1:
            # print("true")
            z = (z * base) % n
        # print(z)
    return z
```

实现了平方-乘算法之后，便可以完成加密与解密操作。

## 四、 实验代码

 _pycache_	2018/11/8 22:46	文件夹	
 RSA.py	2018/11/8 22:46	Python File	3 KB
 generate_prime.py	2018/11/8 21:35	Python File	2 KB

## 五、 运行截图

```
trial: 111
The prime number generated is: 1244356950664156837907667493294170976003596070795972
06217625314968247618823828634785703958812999901387282046923971573945666410340275687
79306055446714027261
trial: 318
The prime number generated is: 8665889537262655700093789451852748773919546989586053
98834282434212710479166604631077746946441932120283617830507213579482096616070471937
6801694490557237241

The message that needs to be encrypted: SUN YAT-SEN UNIVERSITY
message: 83857832896584458369783285787386698283738489
ciphertext: 71965805643463206262899039815609773557064256191014759458763030493863820
30236159437102074028601874197920085093508933536303638138835553669256693419204435690
55923800223758003481428799378633632208490016008005270119199361178988982065374782843
04278890170406721159758122468829508145875548989588526445236781966762669
decode message: 83857832896584458369783285787386698283738489
```

## 六、 结果分析

在结果中，消息经过加密之后得到密文，密文解密之后得到的消息与之前的相匹配，证明加密与解密的过程均正确。

## 七、 总结

这次实验是实现 RSA 算法，实现的过程还算比较顺利，没有遇到什么大问题，而且经过多次阅读书上的相关内容，也能很清楚地了解 RSA 算法的实现过程。

在写代码的过程中，还是有遇到一些小问题，比如在实现平方-乘算法的时候，最开始将指数转化为二进制的时候，考虑到要从高位访问到低位，便将其倒叙输出，所以出现了问题。仔细思考之后，原来本身将指数转化为二进制的时候，得到的是一个字符串，按字符串下标从小到大的顺序刚好是二进制数从高位到低位的顺序。

这次的实验难度不算太大，最主要是搞清楚 RSA 算法实现的过程，实现过程中用到的最主要的两个算法便是平方-乘算法与计算乘法逆的算法，只要搞清楚了这两部分，其他的问题都不是太大。总而言之，这次实验还是有很大的收获。