# 天津大学本科生实验报告专用纸

学院 智能与计算学部 年级 2017 专业 软件工程 班级 2 班 姓名 侯雨茜 学号 3017218092

课程名称 信息安全技术 实验日期 2019-10-25 成绩

同组实验者

## 实验 2：RSA 加密算法实现

### 一、实验目的

实现 RSA 加密算法。

### 二、实验环境

1. 操作系统：Mac OS
2. 实现语言：Python

### 三、实验内容

1. 利用 Miller-Rabin 测试生成两个 256（或 512）比特的素数，记录素数生成的时间。
2. 计算$n = pq$. 利用 Euclidean 算法生成与$\Phi(n)$互素的整数作为私钥, 利用扩展 Euclidean 算法计算$b = a^{-1}$作为公钥。
3. 随机生成明文$m \in Z_n$, 利用平方乘积（Square and Multiply）算法计算密文$c = m^b \ mod \ n$，并进行解密运算$m' = c^a \ mod \ n$，检验解密的正确性，即$m = m'$是否成立。
4. 记录一组明文的 RSA 加密和解密时间。使用 DES 对相同的明文进行加解密，记录 DES 加密和解密时间.（建议选取 10 组以上明文，对比这些明文加密时间总和.）
5. RSA 算法正确性检验。

### 四、实验步骤

**4.1 算法代码如下。**

Prime.py 文件：

```python
import math
import random


# 扩展欧几里得算法求模反元素
def ex_euclid(a, b, list):
    if b == 0:
        list[0] = 1
        list[1] = 0
        list[2] = a
    else:
        ex_euclid(b, a % b, list)
        temp = list[0]
        list[0] = list[1]
        list[1] = temp - a // b * list[1]
```


# 天津大学本科生实验报告专用纸


```python
# 求模反元素
def mod_inverse(a, b):
    list = [0, 0, 0]
    if a < b:
        a, b = b, a
    ex_euclid(a, b, list)
    if list[1] < 0:
        list[1] = a + list[1]
    return list[1]


# 快速幂模运算，把b拆分为二进制，遍历b的二进制，当二进制位为0时不计入计算
def quick_pow_mod(a, b, c):
    a = a % c
    ans = 1
    while b != 0:
        if b & 1:
            ans = (ans * a) % c
        b >>= 1
        a = (a % c) * (a % c)
    return ans
```

```python
# n为要检验的大数，a < n,k = n - 1
def miller_rabin_witness(a, n):
    if n == 1:
        return False
    if n == 2:
        return True
    k = n - 1
    q = int(math.floor(math.log(k, 2)))
    while q > 0:
        m = k // 2 ** q
        if k % 2 ** q == 0 and m % 2 == 1:
            break
        q = q - 1
    if quick_pow_mod(a, n - 1, n) != 1:
        return False
    b1 = quick_pow_mod(a, m, n)
    for i in range(1, q + 1):
        if b1 == n - 1 or b1 == 1:
            return True
        b2 = b1 ** 2 % n
        b1 = b2
    if b1 == 1:
        return True
    return False
```

教师签字：

年 月 日

```python
# Miller-Rabin素性检验算法,检验8次
def prime_test_miller_rabin(p, k):
    while k > 0:
        a = random.randint(1, p - 1)
        if not miller_rabin_witness(a, p):
            return False
        k = k - 1
    return True


# 判断 num 是否与 prime_arr 中的每一个数都互质
def prime_each(num, prime_arr):
    for prime in prime_arr:
        remainder = num % prime
        if remainder == 0:
            return False
    return True


# return a prime array from begin to end
def get_con_prime_array(begin, end):
    array = []
    for i in range(begin, end):
        flag = judge_prime(i)
        if flag:
            array.append(i)
    return array


# judge whether a number is prime
def judge_prime(number):
    temp = int(math.sqrt(number))
    for i in range(2, temp + 1):
        if number % i == 0:
            return False
    return True
```

```python
# 根据 count 的值生成若干个与质数数组都互质的大数
def get_rand_prime_arr(count):
    arr = get_con_prime_array(2, 100000)
    prime = []
    while len(prime) < count:
        num = random.randint(pow(2, 255), pow(2, 256))
        if num % 2 == 0:
            num = num + 1
        while True:
            if prime_each(num, arr) and prime_test_miller_rabin(num, 8):
                if num not in prime:
                    prime.append(num)
                break
            num = num + 2
    return prime
```

RSA.py 文件：

```python
import random
import Prime
import time


# encryption ,according to the formula:m^e = c (mod n) , calculate c ,c == secret,m == me
def encryption(plaintext, puk):
    return Prime.quick_pow_mod(plaintext, puk[1], puk[0])


# decryption ,according to the formula:c^d = m (mod n),  calculate m ,
def decryption(ciphertext, prk):
    return Prime.quick_pow_mod(ciphertext, prk[1], prk[0])
```

```python
def get_RSAKey():
    RSAKey = {}

    start = time.perf_counter()
    prime_arr = Prime.get_rand_prime_arr(2)
    p = prime_arr[0]
    q = prime_arr[1]
    while p == q:
        q = random.choice(prime_arr)
    end = time.perf_counter()

    n = p * q
    s = (p - 1) * (q - 1)
    a = 65537
    b = Prime.mod_inverse(a, s)

    print("随机生成的素数p = ", p)
    print("随机生成的素数q = ", q)
    print('素数生成的时间为: ', end - start, 's')

    print("n = pq = ", n)
    print("利用Euclidean算法生成的私钥a = ", a)
    print("利用扩展Euclidean算法生成的公钥b = ", b)

    puk = [n, a]
    prk = [n, b]
    RSAKey['puk'] = puk
    RSAKey['prk'] = prk
    return RSAKey
```

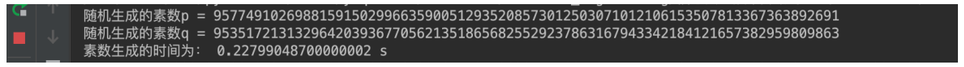
```python
if __name__ == '__main__':
    RSAKey = get_RSAKey()
    # print("Enter a number less and shorter than ", len(str(RSAKey['puk'][0])), ",", RSA
    print('请输入明文: ')
    # only encrypt a number type
    m = int(input())
    c = encryption(m, RSAKey['puk'])
    print("RSA加密后的密文为:", c)
    # print(len(str(c)))
    m1 = decryption(c, RSAKey['prk'])
    print("RSA解密后的密文为:", m1)
    # print(len(str(m1)))
    # 检验解密的正确性
    if m == m1:
        print('解密成功')
    # 记录10组明文的RSA加密和解密时间
    sumEnTime = 0
    sumDeTime = 0
    for i in range(10):
        print('请输入明文: ')
        # only encrypt a number type
        m = int(input())

        # RSA加密
        start = time.perf_counter()
        c = encryption(m, RSAKey['puk'])
        end = time.perf_counter()
        sumEnTime = sumEnTime + (end - start)
        print("RSA加密后的密文为:", c)
        # print(len(str(c)))

        # RSA解密
        start = time.perf_counter()
        m1 = decryption(c, RSAKey['prk'])
        end = time.perf_counter()
        sumDeTime = sumDeTime + (end - start)
        print("RSA解密后的密文为:", m1)
    print('10组明文的RSA加密时间为: ', sumEnTime)
    print('10组明文的RSA解密时间为: ', sumDeTime)
```

**4.2 利用 Miller-Rabin 测试生成两个 256（或 512）比特的素数，记录素数生成的时间。**

测试结果如下：




随机生成的素数 p =

9577491026988159150299663590051293520857301250307101210615350781336736389269 1

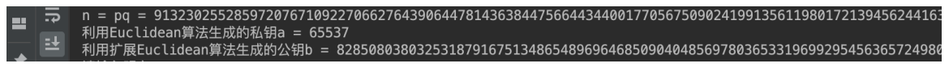
随机生成的素数 q =

9535172131329642039367705621351865682552923786316794334218412165738295980986 3

素数生成的时间为： 0.22799048700000002 s

**4.3 计算$n = pq$. 利用 Euclidean 算法生成与$\phi(n)$互素的整数$a$作为私钥，利用扩展 Euclidean 算法计算$b = a^{-1}$作为公钥.**

测试结果如下：




n = pq =

913230255285972076710922706627643906447814363844756644344001770567509024199135611980172139456244163613511029158716741377992082022746388767510147689541133 3

利用 Euclidean 算法生成的私钥 a = 65537

利用扩展 Euclidean 算法生成的公钥 b =

828508038032531879167513486548969646850904048569780365331969929545636572498083063000926053895426632027668688878848042027576371726948484915487535452910705 3

**4.4 随机生成明文$m \in Z_n$，利用平方乘积（Square and Multiply）算法计算密文$c = m^b \ mod \ n$，并进行解密运算$m' = c^a \ mod \ n$，检验解密的正确性，即$m = m'$是否成立。**

测试结果如下：


请输入明文：

7345874874086706

RSA 加密后的密文为:

3899893164165662650305062235305811581694247742952136769146324534199029087754317272019098344422525827117140346252602799872341380795588940691844422411975878

RSA 解密后的密文为: 7345874874086706

解密成功

4.5. **记录一组明文的 RSA 加密和解密时间。使用 DES 对相同的明文进行加解密，记录 DES 加密和解密时间.（建议选取 10 组以上明文，对比这些明文加密时间总和.）**

选取的 10 组明文为：

0123456789012345

0123456789123456

0123456789234567

0123456789345678

0123456789456789

0123456789567890

1234567890123456

1234567891234567

1234567892345678

1234567893456789

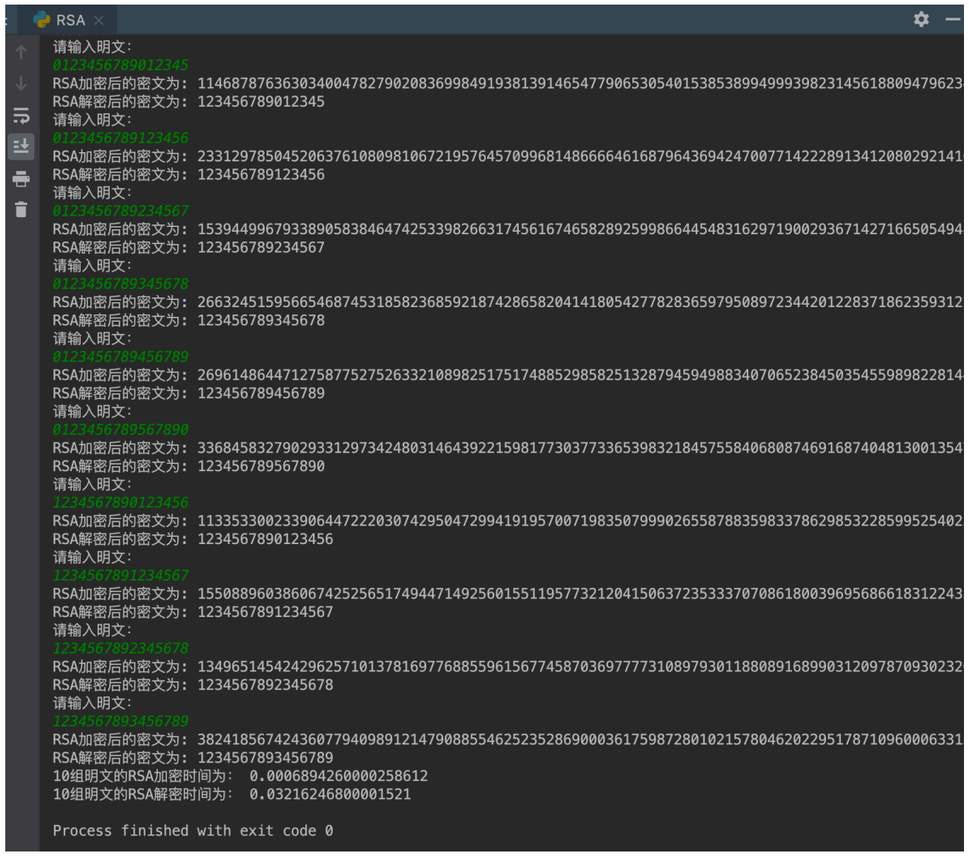
测试结果如下：

RSA 加密时间总和为：0.0006894260000258612 s

RSA 解密时间总和为：0.03216246800001521 s

DES 加密时间总和为：0.10484764400002611 s

RSA 测试结果：




DES 测试结果：

```
/Users/yuqianhou/Documents/Software_Engineering/信息安全技术/实验/DES/DES_py/venv/bin/python /Users/yuqianhou/Documents/Soft
密钥k = 4F6556C45580DAD9
请输入16进制明文x:
0123456789012345
加密后的密文z为: 73E22465B140101F
请输入16进制明文x:
0123456789123456
加密后的密文z为: 4FD52DD09A56793F
请输入16进制明文x:
0123456789234567
加密后的密文z为: E7222D1B4BC3BC76
请输入16进制明文x:
0123456789345678
加密后的密文z为: F1BB9D9102F67A0D
请输入16进制明文x:
0123456789456789
加密后的密文z为: 124D54093BDD32C3
请输入16进制明文x:
0123456789567890
加密后的密文z为: 10FB0E0950FC8CDF
请输入16进制明文x:
1234567890123456
加密后的密文z为: CA11669E214605AE
请输入16进制明文x:
1234567891234567
加密后的密文z为: 79193ADAE2200C6E
请输入16进制明文x:
1234567892345678
加密后的密文z为: 5290163AA7B0C6DD
请输入16进制明文x:
1234567893456789
加密后的密文z为: 01A977237230520E
加密10组明文所用时间为: 0.10484764400002611

Process finished with exit code 0
```

**4.6. RSA 算法正确性检验。**

检验通过。

**五、结论**

通过本次实验的实践和学习，我掌握了使用 python 语言编写 RSA 加密算法的方法。同时，我还在对算法的测试过程中发现，RSA 算法生成素数和加密的速度都是非常快的，加密一组数据仅需要不到$7 * 10^{-5}$秒的时间。在与 DES 加密算法的对比过程中发现，虽然 DES 加密算法的速度也很快，加密一组数据仅需要 0.01s，但是这还远远比 RSA 算法慢了许多。