实验 2 RSA 密码算法

姓名：__王木一__ 学号：_200210231_

一、 运行截图

```
---------RSA加密算法演示脚本---------
是否开始生成密钥, 1-开始 0-退出:1
【密钥生成】
选取的大素数p, q: (8377, 8609)
两数之积n: 72117593, phi_n: 72100608
公钥[e, n]: [60530707, 72117593]
私钥[d, n]: [31954459, 72117593]
请输入地址以保存密钥:key.txt
【加解密演示】
请输入要加密的明文地址:lab2-Plaintext.txt
待加密的明文为: 2002 A.M. TURING AWARD. RSA, an acronym for Rivest, Shamir and Adleman, uses algorithmic number theory to provide an efficient realization of a public-ke
明文编码结果: 181616180033144514005253504146390033553350361400505133120065780065678279788977007079820050738669838412005172657773820065786800336876697765781200858369830
明文加密结果: 4507996766321850102526825387501534640777053194575118615151224558102526824871752915330195346407771820015208895745371982986021868716909351445643582758710865
请输入地址以保存密文:lab2-Ciphertext.txt
是否进行解密, 1-解密 0-退出:1
请输入待解密的密文地址:lab2-Ciphertext.txt
密文解密结果: 2002 A.M. TURING AWARD. RSA, an acronym for Rivest, Shamir and Adleman, uses algorithmic number theory to provide an efficient realization of a public-key
请输入地址以保存已解密的明文:dectypted.txt
演示结束

Process finished with exit code 0
```

二、 实验过程中遇到的问题有哪些？你是怎么解决的。

1. 分组编码

每次加密一个 4 位的十进制数字，但加密后的数字位数并不唯一，就会导致解密时不知如何分组，本次实验采用一种很简单的方法具体请见下方。

三、 请说明你的字符分组方式，以及关键的算法例如扩展欧几里德，素数检测，快速幂等。

1. 分组与编码

编码：采用 ASCII 编码，两个字符一组组成一个 4 位的十进制数字进行加密，若明文长度为奇数则在最后加上字符‘X’。由于部分字符的 ASCII 十进制码超过 3 位，故将每个字符编码值减去 32，保证加密的数字小于 10000。例如：‘z’对应 ASCII 码为 122，‘y’对应 ASCII 码为 121 ‘zy’对应的编码就为 9089。若分组编码不足 4 位将在前面补 0

加密：每组加密的结果位数不一定相等。由于选择的两个大素数 p，q 分别为 14-bit 的小于 10000 的数（范围 8192~10000），两者之积 n 的位数为 8 位。故加密的结果

$$C = m^e \ mod \ n$$

C 的值最大为 8 位十进制数，但也有可能不足 8 位，故加密结果不为 8 位的在前面补 0.

解密：由于每组加密的结果都已经规范化为 8 位，解密时按 8 位一组进行解密即可。

2. 加解密算法

扩展欧几里得算法（求 d）：根据裴蜀定理，$gcd(a,b) = as + bt$ 在 RSA 加密中，有$1 = gcd(\phi(n), e) = \phi(n)s + et.$ t即为 e 的乘法逆，即 d。

例如 gcd(7,3) = 1 = 7 * 1 + 3 * (-2)， -2（5）就是 3 的模 7 乘法逆

商 q	余数 r	s	t
	7	1	0
	3	0	1
7 / 3 = 2	7 – 3 * 2 = 1	1 – 0 * 2 = 1	0 – 1 * 2 = -2
3 / 1 = 3	3 – 1 * 3 = 0(结束		

具体代码：

```
def get_pvtkey(self, e, phi_n):
    old_s, s = 1, 0
    old_t, t = 0, 1
    old_r, r = phi_n, e
    while r != 0:
        q = old_r // r
        old_r, r = r, old_r % r
        old_s, s = s, old_s - q * s
        old_t, t = t, old_t - q * t
    if old_t < 0:
        return phi_n + old_t
    else:
        return old_t
```

快速幂算法：根据模运算化简规则：$(a * b)\ mod\ n = (a\ mod\ n) * (b\ mod\ n)\ mod\ n$ 根据此思想进行快速幂运算。设 m = a mod n，那么 a^2 mod n = m^2 mod n.

例如：

求模指数实例

11^{23}mod 187=[(11^1mod 187)*(11^2mod 187)*(11^4mod 187)
(11^8mod 187)(11^8mod 187)] mod 187
11^1mod 187=11
11^2mod 187=121
11^4mod 187=14641mod 187=55
11^8mod 187=214358881mod 187=33
11^{23}mod187=(11*121*55*33*33)mod 187
=79720245mod 187=88

具体代码：

```
'''快速幂运算'''
def fast_exp_mod(self, a, b, p):
    y = 1
    while True:
        if b == 0:
            return y
        while b > 0 and b % 2 == 0:
            a = (a ** 2) % p
            b /= 2
        b -= 1
        y = (a * y) % p
```

素数产生：首先选取随机数，再使用 miller-rabin 算法检测其是否为素数。算法检测至少 10 轮以降低素性检测错误概率。

```python
def witness(a, n):
    '''miller-rabin核心'''
    m = n - 1
    j = 0
    while m % 2 == 0:
        m /= 2
        j += 1
    x = fast_exp_mod(a, m, n)
    if x == 1 or x == n - 1:
        return True

    j -= 1
    while j > 0:
        x = fast_exp_mod(x, 2, n)
        if x == n - 1:
            return True
        j -= 1
    return False


def miller_rabin(n):
    if n == 2:
        return True
    if n < 2 or n % 2 == 0:
        return False
    for i in range(0, random.randint(10, 20)):
        a = random.randint(2, n - 2)
        if not witness(a, n):
            return False
    return True
```

注：因为老师未要求必须使用 miller-rabin 算法进行素数生成，故最开始使用的是 pycrypto 包来生成素数（Util.number.getPrime()）和判断素数（Util.number.isPrime()）。后考虑到提交后老师的测试环境，还是实现了 miller-rabin 算法，不再额外使用需要单独配置的工具包。

本次实验环境：python==3.7