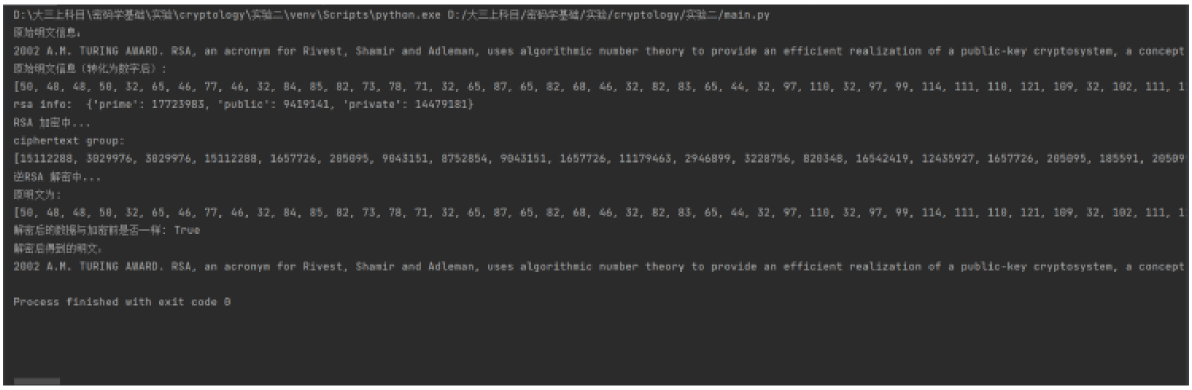
实验2 RSA密码算法

> 200110625 柯炽炜

1. 运行截图



2. 实验过程中遇到的问题有哪些？你是怎么解决的。
    - 如何找到一个足够大的随机素数：
        - 问题描述：由于Miller-Rabin算法提供的是判断素数的必要条件而不是充分条件，通过Miller-Rabin算法判定的数并不一定为素数。
        - 解决措施：在Miller-Rabin算法的基础上进行改进，对于每一个需要判定的数，我们进行10次判断，若10次判断的结果均为素数，则可以说明这个数有极大的概率为素数。
    - 如何合理的对字符进行分组：
        - 解决措施：调用python自带的 `ord`() 函数将读入的每一个字符转化为其对应的ASCII码值，并将这个值存入到一个list中
3. **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。**
    - 字符分组方式：调用python自带的`ord()`函数将读入的每一个字符转化为其对应的ASCII码值，并将这个值存入到一个list中
    - Miller-Rabin算法：

```python
def miller_rabin(test_num):
    # 素性检测次数
    safe_time = 10

    # 找出整数k,q，满足testNum - 1 = 2^k * q
    n = test_num - 1
    k = 0
    q = 0

    # //为整除运算，n经过循环出来的值则为q
    while n % 2 == 0:
        k += 1
        n //= 2
    q = n

    # 素性判定流程
    for test_index in range(safe_time):
```

```
        a = random.randrange(2, test_num - 1)
        # 测试标准1
        first_test = pow(a, q, test_num)
        if first_test == 1 or first_test == test_num - 1:
            continue
        else:
            # 测试标准2
            second_test = first_test
            prime_flag = False

            for j in range(1, k):
                second_test = pow(second_test, 2, test_num)
                if second_test == test_num - 1:
                    prime_flag = True
                    break
            # 如果判定为素数，则继续循环
            if prime_flag:
                continue
            # 不满足标准2则返回False
            return False
    # 若经过10次判定均为很有可能，则返回True
    return True
```

- 拓展的欧几里得算法：

```
def extended_enclid(a, b):
    # a < b 时换个位置
    if a < b:
        t = b
        b = a
        a = t

    x = [1, 0, a]
    y = [0, 1, b]
    while True:
        if y[2] == 0:
            return False
        elif y[2] == 1:
            return y[1] % a
        else:
            q = x[2] // y[2]
            t = []
            for i in range(3):
                t.append(x[i] - q * y[i])
            x = y
            y = t
```

- 快速幂算法：

```python
def fast_mul(base, exponent, modulus):
    binary = bin(exponent).replace('0b', '')

    exponent_count = 0
    result = 1
    for bit in reversed(binary):
        if int(bit) == 1:
            # 对于第i位为1
            result *= pow(base, pow(2, exponent_count), modulus)
        exponent_count += 1
    result = result % modulus
    return result
```

- 获取指定范围内大素数:

```python
def get_prime(low_bound, up_bound):
    result = False
    while not result:
        prime = random.randint(low_bound, up_bound)
        result = miller_rabin(prime)
    return prime
```

- 获取RSA算法密钥:

```python
def get_keys_and_prime():
    # 得到p, q两个质数
    p = get_prime(1000, 10000)
    q = get_prime(1000, 10000)
    # 大素数
    n = p * q
    euler_n = (p - 1) * (q - 1)

    # 得到公钥
    e = get_prime(1, euler_n)
    while not extended_enclid(e, euler_n):
        e = get_prime(1, euler_n)

    # 得到乘法逆d（私钥）
    d = extended_enclid(e, euler_n)

    infos = {
        'prime': n,
        'public': e,
        'private': d
    }
    return infos
```

- 明文读取并分组:

```python
def read_and_divide_text(path):

    # 文件读取
    file = open(path)
    text = file.read()

    # 将文字转换成数字 (Unicode)
    num_text = []
    for word in text:
        word_num = ord(word)
        num_text.append(word_num)

    return num_text
```

- RSA算法:

```python
def rsa(path):

    # 获取明文信息
    plaintext_num_group = read_and_divide_text(path)
    print("原始明文信息（转化为数字后）:")
    print(plaintext_num_group)
    # 获取rsa算法相关元素
    rsa_info = get_keys_and_prime()
    public_key = rsa_info.get('public')
    prime = rsa_info.get('prime')

    # 密文数组
    ciphertext_num_group = []

    # 加密流程
    for num in plaintext_num_group:
        cipher_num = fast_mul(num, public_key, prime)
        ciphertext_num_group.append(cipher_num)

    return ciphertext_num_group, rsa_info
```

- De_RSA 算法:

```python
def de_rsa(private_key, prime, ciphertext_group):

    plaintext_group = []
    for num in ciphertext_group:
        plaintext_num = fast_mul(num, private_key, prime)
        plaintext_group.append(plaintext_num)
    return plaintext_group
```

- 文本恢复:

```python
def text_recovery(plaintext_group):
    text_group = ""
    for num in plaintext_group:
        code = int(num)
        word = chr(code)
        text_group += word
    with open('解密后得到的明文.txt', 'w', encoding='utf-8') as f:
        f.write(text_group)
        f.close()
    return text_group
```

- 主函数:

```python
if __name__ == '__main__':
    # print(fast_mul(2, 20, 100))
    path = r'message/lab2-Plaintext.txt'
    # 获取原始明文信息
    file = open(path)
    text = file.read()
    print("原始明文信息: ")
    print(text)
    plaintext_num_group = read_and_divide_text(path)
    # 加密算法
    ciphertext_num_group, rsa_info = rsa(path)
    print("rsa info: ", rsa_info)
    print("RSA 加密中...")
    print("ciphertext group:")
    print(ciphertext_num_group)
    plaintext_group = de_rsa(rsa_info.get('private'),
rsa_info.get('prime'), ciphertext_num_group)
    print("逆RSA 解密中...")
    print("原明文为: ")
    print(plaintext_group)
    print("解密后的数据与加密前是否一样:", plaintext_group ==
plaintext_num_group)
    print("解密后得到的明文: ")
    print(text_recovery(plaintext_group))
```