

一、 实践内容

（一）实践要求：

①实现 RSA 的密钥生成、数据加密、数字签名。

②密钥生成包括生成两个大素数 p, q ，计算 $n = p \times q$ 和 $\varphi(n) = (p - 1)(q - 1)$ ，然后选择与 $\varphi(n)$ 互素且小于 $\varphi(n)$ 的整数 e ，计算 $d = e^{-1} \bmod \varphi(n)$ ，最后得到公钥 $\{e, n\}$ 和私钥 $\{d, n\}$ 。要求 p, q 至少均大于 10^{10} ，将生成的整数 p, q, n, e, d 分别写入文件 `p.txt`、`q.txt`、`n.txt`、`e.txt`、`d.txt` 中。注意，所有整数都必须用 16 进制表示。必须将整数转化成字符串后再写入文件，例如素数 $p=6B1BCF$ (用 16 进制表示)，则写入文件的应是字符串“6B1BCF”而非整数 6B1BCF。

③数据加密是指用公钥 $\{e, n\}$ 对指定的明文进行加密。数字签名是指用私钥 $\{d, n\}$ 对指定的明文进行加密。数据加密和数字签名都有一组对应的测试数据，以便检查程序的正确性。要求以命令行的形式，指定明文文件、密钥文件的位置和名称以及加密完成后密文文件的位置和名称。加密时先分别从指定的明文文件、密钥文件中读取有关信息，然后进行加密，最后将密文写入指定的密文文件。注意，密文(一个整数)必须用 16 进制表示。必须将密文(一个整数)转化成字符串后再写入文件，例如密文 $c=154A6B$ (用 16 进制表示)，则写入文件的应是字符串“154A6B”而非整数 154A6B。

④最终形式：命令行指定参数运行 py 代码，py 代码和 txt 文件都要和 `python.exe` 在同一个路径下面。

（二）相关原理：

（1）RSA 加解密：

1. RSA算法描述

首先明文空间 P =密文空间 $C = Z_n$

① 密钥的生成

1. 选择两个大素数 p, q , (p, q 为互异素数, 需要保密),
2. 计算 $n = p \times q$, $\varphi(n) = (p-1) \times (q-1)$
3. 选择整数 e 使 $(\varphi(n), e) = 1$, $1 < e < \varphi(n)$
4. 计算 d , 使 $d = e^{-1} \bmod \varphi(n)$,
得到: 公钥 $KU = \{e, n\}$; 私钥 $KR = \{d\}$

对于明文: $m < n$

② 加密(用 e, n): 密文 $c = m^e \bmod n$.

③ 解密(用 d, n): 对密文 c , 明文 $m = c^d \bmod n$

(2) RSA 密钥生成:

密钥产生的一般过程:

- ① 产生一个随机数 e (例如使用伪随机数产生器);
- ② 判断 $(\varphi(n), e) = 1$? 若不满足则转①; (已证明两个数互素的概率为0.6)
- ③ 由扩展的Euclid算法计算: $d = e^{-1} \bmod \varphi(n)$

在随机生成大素数 p, q 时, 要针对随机数作素性检测。采用 Miller-Rabin 素性检测方法, 原理如下:

(2) 素数检测 (Miller-Rabin的素数概率检测法)

定理: 若 p 是一个奇素数, 则方程 $x^2 \equiv 1 \bmod p$ 的解只有 $x \equiv 1$ 或 $x \equiv -1$ (有限域上的平方根定理, 亦称“二次探测定理”)

证明: 由 $x^2 \equiv 1 \bmod p$, 有 $x^2 - 1 \equiv 0 \bmod p$

即: $(x-1)(x+1) \equiv 0 \bmod p$ 因此由模运算规则,

一定有: $p \mid (x+1)$ 或 $p \mid (x-1)$, 或 $p \mid (x+1)$ 且 $p \mid (x-1)$

若 $p \mid (x+1)$ 且 $p \mid (x-1)$, 则存在两个整数 k_1 和 k_2 , 使得

$$(x+1) = k_1 p, (x-1) = k_2 p,$$

两式相减得 $2 = (k_1 - k_2)p$ 而 p 为奇素数, 此式不可能成立, 故只有 $p \mid (x+1)$ 或 $p \mid (x-1)$ 。

设 $p \mid (x-1)$, 则由对模 p 同余的充要条件有, $x \equiv 1 \bmod p$

类似地可得 $x \equiv -1 \bmod p$ 得证

考虑定理的逆否命题:

如果方程 $x^2 \equiv 1 \bmod p$ 存在一解 $x_0 \notin \{1, -1\}$, 则 p 不为素数。

Miller-Rabin的核心算法（WITNESS 算法）：

WITNESS(a, n) (n 是待检验数, a 是小于 n 的整数)

(1) 将 $n-1$ 表示成为二进制形式 $b_{k-1}b_{k-2}\dots b_0$;

然后, 用平方-乘法算法计算 $a^{n-1} \bmod n$

(1) $d \leftarrow 1$

for $i = k-1$ downto 0 do {

$x \leftarrow d$;

$d \leftarrow (d \times d) \bmod n$;

For循环结束后, $d = a^{n-1} \bmod n$, 由费马小定理, 若 n 为素数, 则 d 为1, 因此, 若 $d \neq 1$, 则 n 肯定不是素数, 返回FALSE。

因为 $n-1 \equiv -1 \bmod n$, 所以 $(x \neq 1)$ and $(x \neq n-1)$ 意指 $x^2 \equiv 1 \bmod n$ 有非 $\{1, -1\}$ 中的根。因此 n 肯定不是素数, 返回FALSE。

30/80

二、实践环境

pc 操作系统: win10

代码编写 IDE: PyCharm2019.3

执行: 命令行指定参数, 运行 py 代码

编程语言: Anaconda3 下的 python3.7

三、实践过程与步骤

(一) RSA 的密钥生成:

命令行指定明文文件 `rsa_plain.txt` 和写入加密数据的密文文件 `rsa_cipher.txt`; 程序运行过程中会新建 `p/q/n/e/d.txt` 文件并写入相应数据。在加密之后, 再对密文进行解密。以下为一次密钥生成及加解密测试结果:

预置原文:

 rsa_plain.txt - 记事本

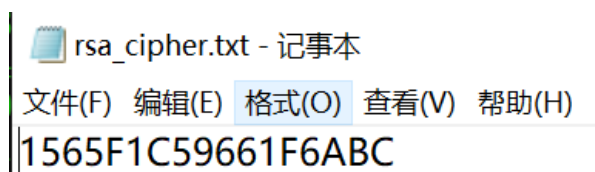
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

1B30746F67726C3A3

随机产生大于 10^{10} 的 p 、 q , 计算 n 、 fn , 选取满足条件的 e , 计算唯一匹配的 d , 这样就生成了新密钥。对预置数据进行一次加解密, 屏幕输出如下:

```
D:\Anaconda3>python RSA.py -p rsa_plain.txt -c rsa_cipher.txt
原明文（十六进制）：1B30746F67726C3A3
p: 10000000019
q: 10000001383
n: 100000014020000026277
e: 10000000033
d: 33026938154781821905
数据加密后（十进制）：24670468454049868476
数据加密后（十六进制）：1565F1C59661F6ABC
反向解密（十六进制）：1B30746F67726C3A3
```

检查写入文件的加密结果：



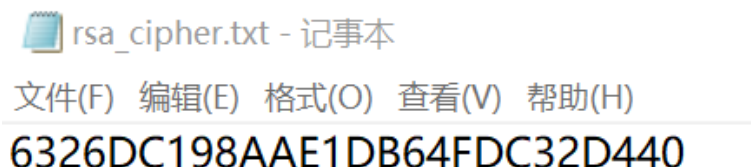
```
rsa_cipher.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
1565F1C59661F6ABC
```

对加密结果反向解密，结果与原给定明文对照，发现完全一致，这表明程序正确且成功运行。

（二）利用 RSA 进行数据加密：

指定明文文件、存放整数 n 的文件、数据加密时存放整数 e 的文件、数据加密结果文件：

```
D:\Anaconda3>python RSA.py -p plainfile.txt -n nfile.txt -e efile.txt -c rsa_cipher.
txt
数据加密：6326DC198AAE1DB64FDC32D440
Already written to rsa_cipher.txt
```



```
rsa_cipher.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
6326DC198AAE1DB64FDC32D440
```

根据测试数据进行 RSA 数据加密，结果正确且成功写入指定的 rsa_cipher 文件。

（三）利用 RSA 进行数字签名：

稍微改动一下代码，变成 RSA_sign 版本。指定明文文件、存放整数 n 的文件、数字签名时存放整数 d 的文件、数字签名结果文件：

```
D:\Anaconda3>python RSA_sign.py -p plainfile.txt -n nfile.txt -d dfile.txt -c rsa_si
gn.txt
数字签名：CA653B30EED2C6B77DCB8381F
Already written to rsa_sign.txt
```

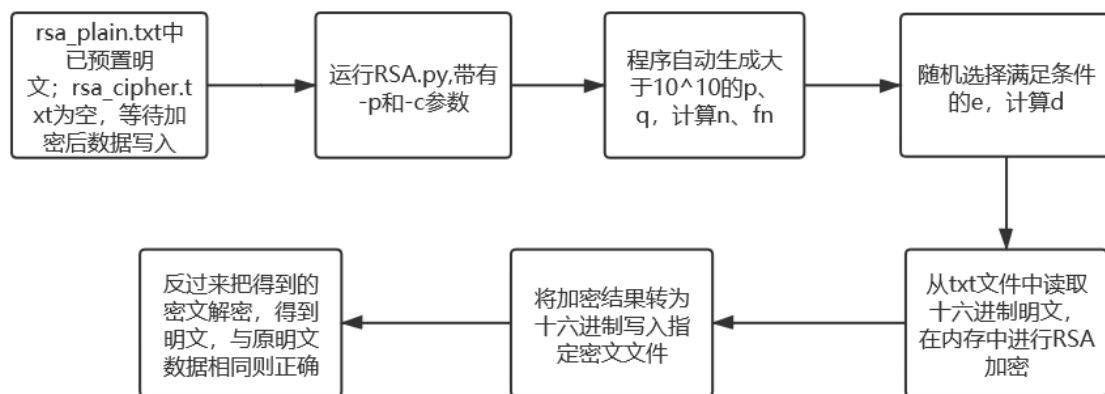


根据测试数据进行 RSA 数字签名，结果正确且成功写入指定的 rsa_sign 文件。

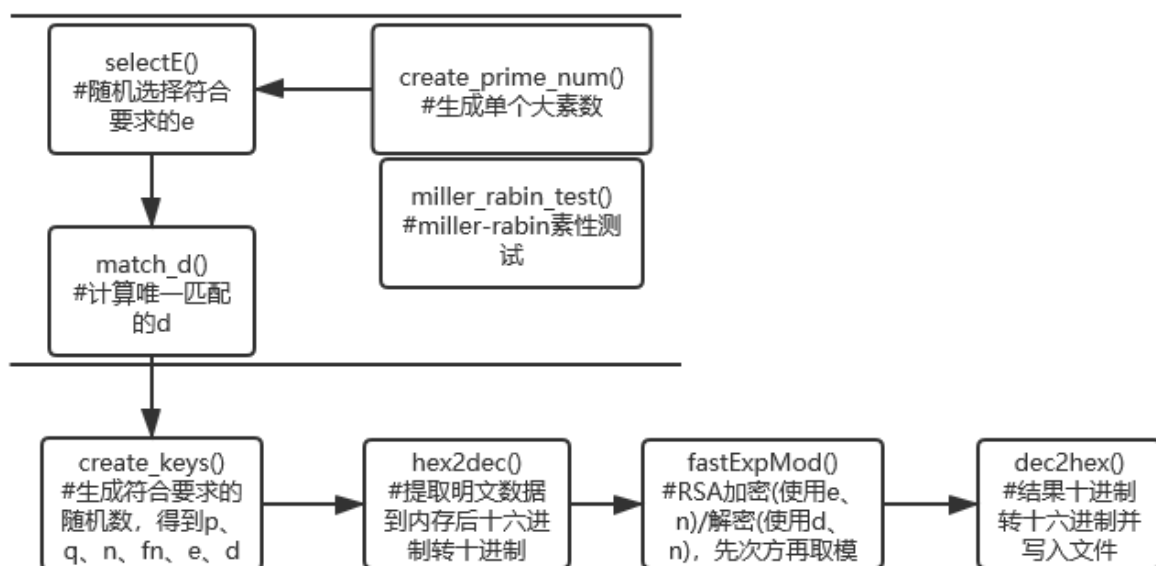
四、 程序设计方案

（一）自生成密钥进行数据加解密

本程序涉及到了文件读写，在 python 解释器的同一目录下提前已准备好 rsa_plain.txt 和 rsa_cipher.txt 两个文件，前者已预置有明文数据，加密之后会将密文以十六进制字符串形式写入第二个文件。具体的程序流程为：



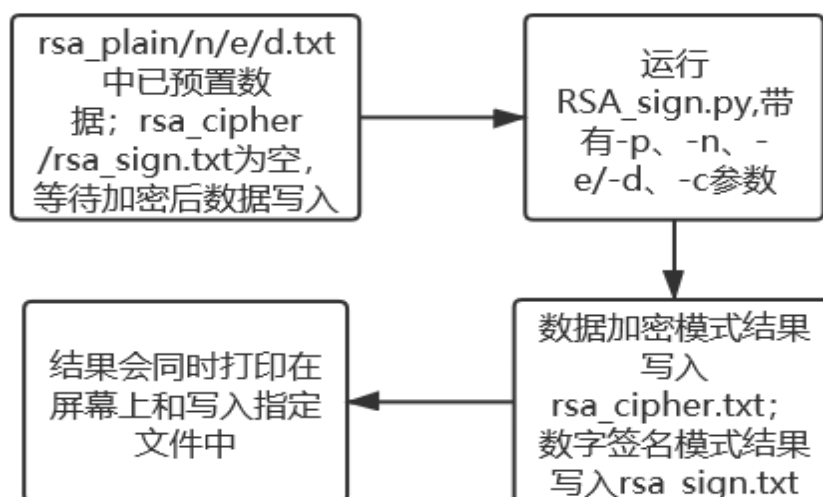
密钥的生成涉及到了几个关键函数。下面通过几个**关键函数**的示意图来介绍程序的编写：



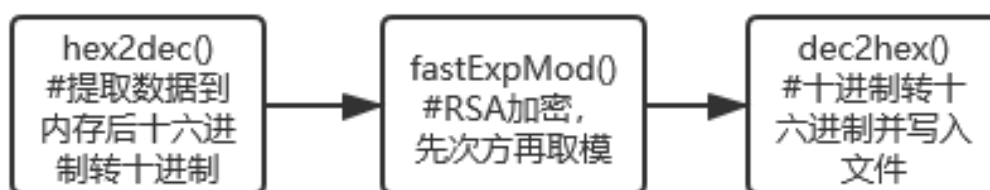
具体代码见附件中 RSA_密钥生成→*RSA.py*。

(二) 明文、密钥已预置在 txt 文件中，数据加密、数字签名测试

明文已预置在 `plainfile.txt` 中, `n` 已预置在 `nfile.txt` 中, `e` 已预置在 `efile.txt` 中, `d` 已预置在 `dfile.txt` 中。从各个文件中读取相应数据到内存, 使用 `{e, n}` 进行数据加密, 结果以十六进制形式写入 `aes_cipher.txt`; 使用 `{d, n}` 进行数字签名, 结果以十六进制形式写入 `aes_sign.txt`。下面是流程示意图:



代码在上一部分实验的基础上稍作改动(注释掉密钥的生成部分)即可, 关键函数如下:



具体代码见附件中 RSA_测试数据→RSA_sign.py。

五、 实践结果与分析

实验结果截图参见上文第四部分——实践过程与步骤。

实践发现，由于涉及到大整数的各种较为消耗计算资源的复杂计算（针对随机数的素性判断、次方运算、取模、求逆等），程序运行速度比较慢、电脑风扇呼呼直响（可能与解释型语言也有关系）；我认为这也可以间接说明 RSA 算法加解密运算复杂、效率低，并不宜直接用于数据加密。另一方面，这也意味着 RSA 算法的安全性高（虽然仍有“可能报文攻击”的威胁）。破译 RSA 不会比大整数分解这样的数学难题更加困难。

在本次实验中，模拟了数据接收方的密钥生成和解密行为、发送方的加密行为；实践了数据加密和数字签名行为。使用 RSA 算法，系统开放性好，密钥管理比较容易；可以提供抗抵赖服务（数字签名）。