安阳工学院

密码学应用

RSA 加密和解密算法的实现

姓名: 申恒恒
学号: 13031110141
班级: 网络工程 13-1
课程: 密码学应用
老师: 齐万华

2017 年 3 月 4 日

摘要

传统的密码在密码分配和管理上是极为困难的，其中对于传统的对称密码来说，其中密钥配送很一个很大的问题，然而使用公钥密码可以解决上述密钥配送的问题。RSA 作为公钥密码的代表[1]，在公开密钥加密和电子商业中 RSA 被广泛使用。RSA 由 Ron River、Adi Shamir 和 Recnard Adleman 在 1978 年共同发表了一种公钥密码加密算法[2]. 本文主要阐述了公钥密码的基本原理、公钥密码的通信流程和其代表 RSA 算法的具体描述和其数学原理，RSA 核心就是对极大整数做因数分解, 对极大整数做因数分解的难度决定了 RSA 算法的可靠性。对一极大整数做因数分解越困难，RSA 算法也就越可靠。但到现在到 2016 年为止，世界上还没有任何可靠的攻击 RSA 算法的方式。只要其钥匙的长度足够长，用 RSA 加密的信息实际上是不能被解破的。因此在了解了其背后的数学原理的基础上，利用 Python 语言实现了 RSA 加解密算法。

关键字：公钥密码、RSA 算法、质因数分解、python 语言实现

[1]有时也称做非对称密码的代表

[2]这三个个人于 2002 年授予计算机图灵奖

目 录

前言 **2**

第一章 公钥密码 **4**

1.1 公钥密码的起源和概念 4

1.2 公钥通信的流程 4

第二章 RSA 算法 **6**

2.1 RSA 加密 6

2.2 RSA 解密 6

2.3 密钥对的生成 7

2.3.1 计算 N 8

2.3.2 计算 L 8

2.3.3 计算 E 8

2.3.4 计算 D 8

第三章 RSA 算法——Python 实现 **10**

3.1 概要描述 10

3.2 整体设计 10

3.3 具体实现 11

3.4 测试环节 14

参考文献 **17**

附录 A RSA 源码 **18**

前言

RSA 作为计算机通信安全的基石，极为重要，有网络存在的地方就一定会有 RSA 的影子，RSA 常常用于公钥密码和数字签名，RSA 作为一种公钥密码算法，在公钥密码中，密钥常常分为加密密钥 E 和解密密钥 D 两种，发送方用 E 对消息进行加密，而接收方则用解密密钥 D 进行解密，而在传统的对称密码中，E 和 D 是相同的，因此必须向接收者配送密钥。用于解密的密钥必须被配送到接收者手中，然而密钥的配送问题将在现实生活中确实很难解决的。

比如 Alice 在网络上认识了 Bob，现在她想给 Bob 发一封 E-mail，而且不想让别人知道邮件的内容，因此 Alice 决定使用对称密码进行加密，这样即便被窃听者 Eve 窃听到通信内容也没有问题.Alice 将邮件内容进行加密生成了密文，当然现在如果将这封密文用邮件发送给 Bob 是不可以的，因为此时 Bob 是不可以直接对密文进行解密的。要使用对对称密码进行解密就必须使用和加密时相同的密钥才可以，也就是说只有同时将密钥发送给 Bob，Bob 才可以完成解密。

那么，此时采取将密文和密文通过 E-mail 发送给 Bob，如果这样做，会导致窃听者 Eve 同时窃听到，这样的话，就相当于 Bob 一样的完成密文的解密并看到明文的内容，所以对于对称加密来说，解密密钥的配送是一个很大的挑战，但是对公钥密码来说，其完全不用考虑解密密钥的配送问题.

在公钥密码中，加密密钥和解密密钥是不同的，只要拥有加密密钥任何人都可以加密，但没有解密密钥是无法解密的，因此公钥密码相对对称加密又被称作做非对称密码，所以公钥密码其中一个重要的性质就是只有拥有解密密钥的人才能够进行解密。接受者事先将加密密钥发送给发送方，其中这个加密密钥被窃听者窃听也没有问题。发送方使用加密密钥对要发送的消息进行加密并将加密后的消息发送给接收方，而且只有拥有解密密钥的人才能够解密，这样以来就不需要将解密密钥配送给接收者，所以对于对称加密的密钥配送问题，公钥密码都可以解决，因此可以说公钥密码是密码学历史上的最伟大的发明。

了解了公钥密码的基本的作用后，那么在公钥密码系统中加密密钥又被称作公钥 (pubic key)，公钥密码一般是公开的，相对的，加密密钥是绝对不能公开的，这个加密密钥只能被自己使用，因此又被称作私钥 (private key)，私钥不可以被任何人知道，包括自己的通信对象。另外，公钥和私钥是一一对应的，一对公钥和私钥统称为密钥对。密钥对中的

两个密钥之间具有非常密切的关系——数学上的关系——因此公钥和私钥是不能分别单独生成的。生活中常见的利用公钥密码的例子有银行卡，网络银行（网银），等等，其中公钥密码的私有密钥加密的文件只有公开密钥能解开. 这一特点被用于 PGP。

对于 RSA 作为公钥密码最成功的算法之一，在论文中会讨论公钥密码的基本流程和 RSA 密码的基本数学原理和加密、解密流程，同时给出加解密的具体操作，最后，在研究了 RSA 密码加解密流程基础之上。我使用 python 语言实现了一般的 RSA 加解密算法程序。在论文中阐述了主要的设计思路和实现方法。

第一章　公钥密码

1.1　公钥密码的起源和概念

公开密钥加密（英语：public-key cryptography，又译为公开密钥加密），也称为非对称加密（asymmetric cryptography），一种密码学算法类型，在这种密码学方法中，需要一对密钥，一个是私人密钥，另一个则是公开密钥。这两个密钥是数学相关，用某用户密钥加密后所得的信息，只能用该用户的解密密钥才能解密。如果知道了其中一个，并不能计算出另外一个。因此如果公开了一对密钥中的一个，并不会危害到另外一个的秘密性质。称公开的密钥为公钥；不公开的密钥为私钥。

该思想最早由瑞夫·墨克（Ralph C. Merkle）在 1974 年提出，之后在 1976 年。惠特菲尔德·迪菲（Whitfield Diffie）与马丁·赫尔曼（Martin Hellman）两位学者以单向函数与单向暗门函数为基础，为发讯与收讯的两方创建密钥。

如果加密密钥是公开的，这用于客户给私钥所有者上传加密的数据，这被称作为公开密钥加密（狭义）。例如，网络银行的客户发给银行网站的账户操作的加密数据。

如果解密密钥是公开的，用私钥加密的信息，可以用公钥对其解密，用于客户验证持有私钥一方发布的数据或文件是完整准确的，接收者由此可知这条信息确实来自于拥有私钥的某人，这被称作数字签名，公钥的形式就是数字证书。例如，从网上下载的安装程序，一般都带有程序制作者的数字签名，可以证明该程序的确是该作者（公司）发布的而不是第三方伪造的且未被篡改过（身份认证/验证）。

常见的公钥加密算法有：RSA、ElGamal、背包算法、Rabin（RSA 的特例）、迪菲－赫尔曼密钥交换协议中的公钥加密算法、椭圆曲线加密算法（英语：Elliptic Curve Cryptography, ECC）。使用最广泛的是 RSA 算法（由发明者 Rivest、Shmir 和 Adleman 姓氏首字母缩写而来）是著名的公开金钥加密算法，ElGamal 是另一种常用的非对称加密算法。

1.2　公钥通信的流程

下面简单描述一下公钥通信的基本流程，假设 Alice 要给 Bob 发送一条消息，Alice 是发送者，Bob 是接受者，而这一次窃听者 Eve 仍然窃听他们的通信内容。

在公钥密码通信中，通信是由接收者 Bob 启动的

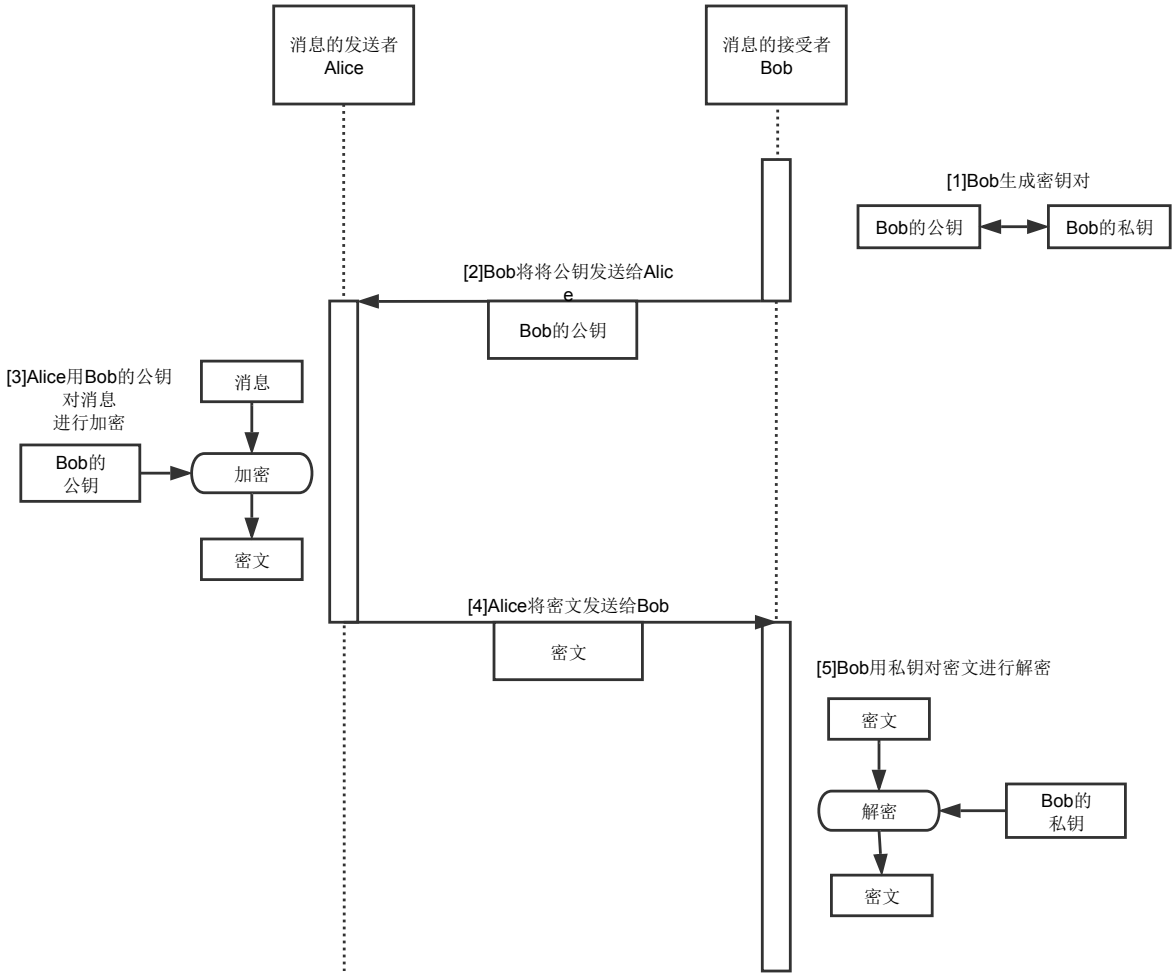

图 1.1: 公钥通信流程图

(1) Bob 生成一个包含公钥和私钥的密钥对私钥由 Bob 自行妥善保管

(2) Bob 将自己的公钥发送给 Alice

(3) Alice 使用 Bob 的公钥对消息加密

(4) Alice 将密文将密文发送给 Bob

第二章　RSA 算法

RSA 加密算法是一种非对称加密算法。在公开密钥加密和电子商业中 RSA 被广泛使用。RSA 是 1977 年由罗纳德·李维斯特（Ron Rivest）、阿迪·萨莫尔（Adi Shamir）和伦纳德·阿德曼（Leonard Adleman）一起提出的。当时他们三人都在麻省理工学院工作。RSA 就是他们三人姓氏开头字母拼在一起组成的。

RSA 可以被用于数字签名和公钥密码。下面介绍一下 RSA 的加解密算法和相关流程

2.1　RSA 加密

假设 Bob 想给 Alice 送一个消息 m，他知道 Alice 产生的 N 和 E。他使用起先与 Alice 约好的格式将 m 转换为一个小于 N，且与 N 互质的整数 n，比如他可以将每一个字转换为这个字的 Unicode 码，然后将这些数字连在一起组成一个数字。假如他的信息非常长的话，他可以将这个信息分为几段，然后将每一段转换为 n。用下面这个公式他可以将 n 加密为 c：

$$c \equiv n^E \pmod{N}$$

计算 c 并不复杂。Bob 算出 c 后就可以将它传递给 Alice。

在 RSA 中，明文、密钥和密文都是数字，在上面的公式里,E 和 N 的组合就是公钥，即，任何人只要拿到这两个数字，就可以完成加密的操作。

2.2　RSA 解密

Alice 得到 Bob 的消息 c 后就可以利用她的密钥 D 来解码。她可以用以下这个公式来将 c 转换为 n：

$$n \equiv c^D \pmod{N}$$

得到 n 后，她可以将原来的信息 m 重新复原。

解码的原理是

$$c^D \equiv n^{E \cdot D} \pmod{N}$$

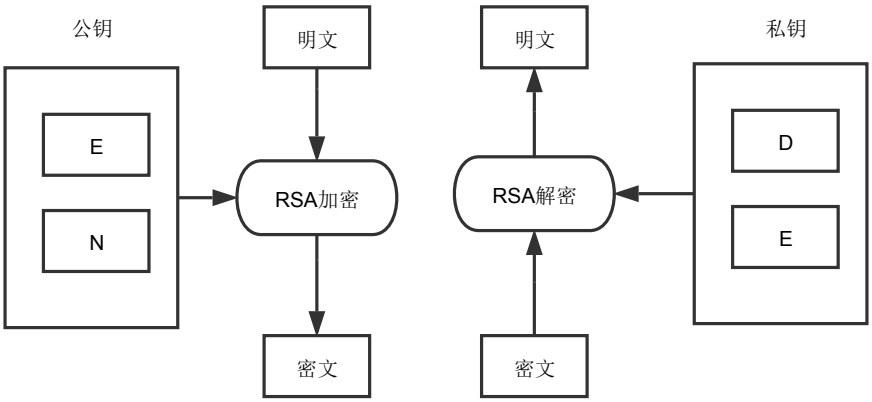

图 2.1: RSA 的加密和解密

已知 $ED \equiv 1 \pmod{r}$，即 $ED = 1 + h\varphi(N)$。由欧拉定理得：

$$n^{ED} = n^{1+h\varphi(N)} = n\left(n^{\varphi(N)}\right)^h \equiv n(1)^h \pmod{N} \equiv n \pmod{N}$$

注意在这里使用的 N 和 RSA 加密是的 N 是一样的，那么数字 D 和 N 的组合就是 RSA 的私钥.

密钥对	公钥	数 E 和数 N
	私钥	数 D 和数 N
加密		密文 $=$ 明文 $^E \bmod N$
解密		密文 $=$ 明文 $^E \bmod N$

表 2.1: RSA 的加密和解密

2.3 密钥对的生成

由于 E 和 N 是公钥，D 和 N 是私钥，因此求 E、D 和 N 这三个数就是生成密钥对，RSA 生成密钥对的生成步骤如下：

(1) 求 N

(2) 求 L

(3) 求 E

(4) 求 D

2.3.1 计算 N

首先准备两个很大的质数 p 和 q，它们的乘积就是 N。

$$N = p \cdot q$$

这里选择 $p = 29$, $q = 31$, 则 $N = p * q = 899$。

2.3.2 计算 L

L 是在 RSA 的加密和解密过程中都不出现，只出现在生成密钥对的过程中。L 是 $p-1$ 和 $q-1$ 的最小公倍数，用 $lcm(X,Y)$ 来表示 "X 和 Y 的最小公倍数"，则 L 可以写作如下式子：

$$L = lcm(p-1, q-1)$$

在例子中，计算 $L = (p-1) \cdot (q-1) = 840$。

2.3.3 计算 E

E 是一个比 1 大比 L 小的数，另外，E 和 L 的最大公约数 (gcd) 必须为 1，若用 gcd 表示 X 和 Y 的最大公约数，则 E 和 L 之间存在如下关系：

$$1 < E < L$$

$$gcd(E, L) = 1$$

对于例子来说，这里选择 $E = 37$。

2.3.4 计算 D

数 D 是由 E 计算得到的，D、E 和 L 之间必须具备如下关系：

$$1 < D < L$$

$$E \times D \bmod L = 1$$

在例子中用程序算的，算出来最小的 D 就是 613

现在，得到了 N,D,E ，把 p 和 q 扔掉，(n, e) 作公钥，(n, d) 作私钥，就可以执行 RSA 加密运算了。

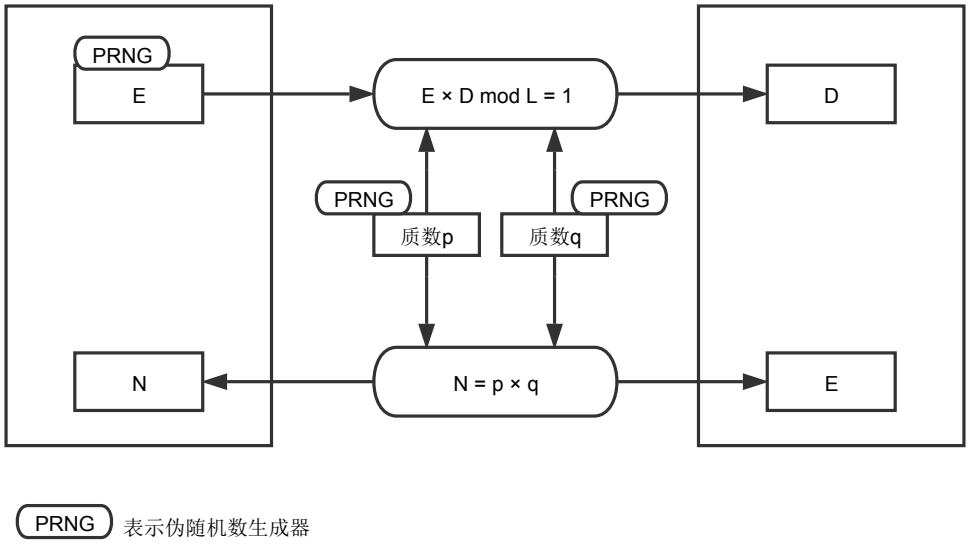

图 2.2: RSA 密钥对生成过程

第三章　RSA 算法——Python 实现

为了实现一个简单的 RSA 加密和解密算法，可以使用模块化的设计方法, 基于函数驱动的思想，将加解密算法中涉及到的各个过程（编码，加密编码，编码转化为密文或明文）抽象成一个个函数模块。然后通过各个函数之间的关系来涉及并实现游戏。通过前面中所介绍的 RSA 加密和解密的基本简单的过程实现。我们可以将所要实现的 RSA 加密描述如下：输入为明文字串格式，比如 aaabbbbcccc,cccddeddfgggd 等。然后根据前面描述的可知在 RSA 中，明文、密钥、密文均为数字，所以必须要对明文字串编码，编码为数字形式的列表这种数据结构，然后利用 RSA 的密文 = 明文 ^{E} mod N 求得密文。我们可以将所要实现的 RSA 解密描述如下：输入为密文的字串格式，比如 aaabbbbcccc,cccddeddfgggd 等。然后根据前面描述的可知在 RSA 中，明文、密钥、密文均为数字，所以必须要对密文字串编码，编码为数字形式的列表这种数据结构，然后利用 RSA 的明文 = 密文 ^{D} mod N 求得明文。

3.1　概要描述

阐述了 RSA 的基本原理和算法后，利用 Python 设计和实现一个 RSA 的加解密程序，在这里我将程序部署在 Web 上，利用 Python 的 Flask 框架结合 Python 程序作为后台加解密算法的引擎，以极为友好的方式展示出来。

Python 作为一门相当高级语言来说，具有简单易用，极为灵活的特点，本次 RSA 的实现主要根据 RSA 算法进行设计的，在 N,D,E 生成，往往会用到伪随机数生成器，但在项目中，只是验证性课题，只是对 N,E,D 进行了赋值，并没有密钥对的生成环节，换言之，已经将密钥对生成完毕，直接拿过来取加解密就可以，所以该程序主要实现了如下功能:

(1) 在密钥对生成的基础上，如何利用 N,E,D 对数据 (只支持字符串) 进行加密和解密

(2) 实现了米勒-拉宾素性测试

3.2　整体设计

加密和解密过程图如下

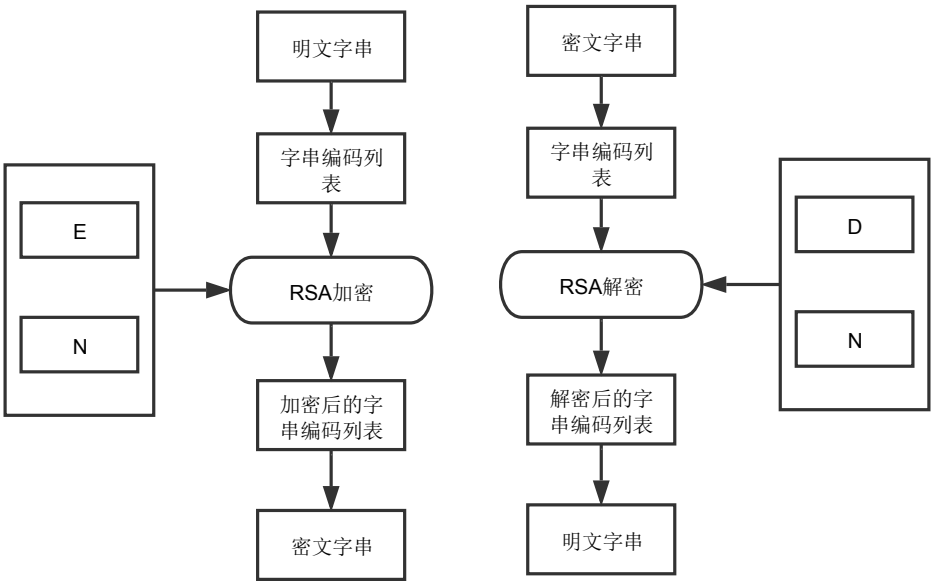

图 3.1: RSA 加密和解密

3.3 具体实现

在程序中定义一个类 RSA，该类实现了加密和解密函数及其辅助函数，具体如下：

```
class RSA(object):
    def __init__(self):pass
    def __chr_to_num(self, char):pass
    def __num_to_chr(self, num):pass
    def __text_to_array(self, text):pass
    def __array_to_text(self, array):pass
    def __array_to_cipher(self, array):pass
    def __cipher_to_array(self, cipher):pass
    def RSA_encrypt(self, text):pass
    def RSA_decrypt(self, cipher):pass
```

___init___() 实现了 RSA 的初始化，在这里主要初始化了 n,e,d, 在 RSA 中分别表示大数 N, 公钥 E, 私钥 D, 由于前面算法给出的例子，这里我使用了手算指定 N 和公钥和私钥, 并使他们符合上面的密钥对生成规则. 根据前面的知识前面可以知道,n 为两个大素数的乘积，这里 p,q 分别为 29，31, 那么 e 和 d 就可以计算出来。例如，在程序中初始化了以下:

```
def __init__(self):
        self.n = 29*31
        self.e = 11
        self.d = 611

        self.__encryptnum = lambda m: calc_mod(m, self.e, self.n)
```

```
7           self.__decryptnum = lambda c: calc_mod(c, self.d, self.n)
```

由于后期的加解密的需要，这里没有使用到伪随机数生成器来初始化 p,q,E, 有以下几个原因：

(1) 伪随机数生成器会导致后续的加解密的计算的时间的开销

(2) 这里的目的我只是实现其 RSA 的加解密原理即可，后续的复杂的数学理论计算要求不高

另外 ___encryptnum 和 ___decryptnum 利用了 python lambda 表达式表示的, 通过 *calc_mod()* 函数来实现加解密编码后的数字的，表示对于每个“消息”来说，进行加密和解密操作，且这里指示的“消息”为数字。

___chr_to_num() 实现了 ascill 码编码成十进位数字，同理 *___num_to_chr()* 实现了十进位数字编码为 ascill 码。程序如下：

```
1  def __chr_to_num(self, char):
2          if char == " ":
3              return 0
4          elif char == "\0":
5              return 27
6          else:
7              return ord(char)-64
8
9      def __num_to_chr(self, num):
10         if num == 0:
11             return " "
12         elif num == 27:
13             return "\0"
14         else:
15             return chr(num+64)
```

其中 ord() 函数和 chr() 函数（对于 8 位的 ASCII 字符串）的配对函数,ord() 函数以一个字符（长度为 1 的字符串）作为参数，返回对应的 ASCII 数值，同理 chr() 函数用一个范围在 range（256）内的（就是 0 ～ 255）整数作参数，返回一个对应的字符。

___text_to_array() 和 *___array_to_text()* 分别实现了如下功能：

(1) 将明文字串变为编码列表

(2) 将编码列表变为明文字串

其中涉及到了 python 中的内建高级函数 *map()*, 又称为 map[1] 操作,*map()* 函数接收两个参数，一个是函数，一个是 Iterable，map 将传入的函数依次作用到序列的每个元素，并把结果作为新的 Iterator 返回[2]。代码实现如下：

[1]map/reduce 的概念来自 Google 的那篇大名鼎鼎的论文“MapReduce: Simplified Data Processing on Large Clusters”，他们作为 google 搜索引擎的基础算法，也是当今构建大数据，分布式系统的核心算法和操作

[2]引用自廖雪峰官方网站：http://www.liaoxuefeng.com/wiki

```
def __text_to_array(self, text):
        array = []
        if len(text)%2 != 0:
            text += "\0"
        t = map(self.__chr_to_num, text)
        for item in izip(t[::2], t[1::2]):
            pre, after = tuple(item)
            array.append(pre*28+after)
        return array

    def __array_to_text(self, array):
        text = ""
        for num in array:
            pre = num//28
            after = num%28
            text += "".join(map(self.__num_to_chr, [pre, after]))
        return text
```

___array_to_cipher() 和 *___cipher_to_array()* 分别实现了编码列表变成了密文字串和将密文字串转化为编码列表，在这里主要利用了 python 中内建的 base64 模块，base64 模块是用来作 base64 编码解码的。这种编码方式在电子邮件中是很常见。它可以把不能作为文本显示的二进制数据编码为可显示的文本信息。编码后的文本大小会增大 1/3。在程序中分别使用了 encodestring 和 decodestring 方法，encodestring,decodestring 作为 base64 模块的一组，它们专门用来编码和解码字符串，因为 base64 编码后的字符除了英文字母和数字外还有三个字符 + / =, 其中 = 只是为了补全编码后的字符数为 4 的整数，在程序中会看到每次加密后的密文后面会出现 =。程序如下：

```
def __array_to_cipher(self, array):
        text = ""
        for num in array:
            pre = num//28
            after = num%28
            text += "".join(map(chr, [pre, after]))
        return base64.encodestring(text)

    def __cipher_to_array(self, cipher):
        array = []
        t = map(ord, base64.decodestring(cipher))
        for item in izip(t[::2], t[1::2]):
            pre, after = tuple(item)
            array.append(pre*28+after)
        return array
```

RSA_encrypt() 函数是整合的方法，主要实现 RSA 加密功能，其通过传递字串类型的参

数，首先利用 *___text_to_array()* 将 ascill 码字符串转化为编码列表类型，然后利用 *map()* 操作对编码列表[3]的每个元素（编码）进行加密操作，加密后编码列表通过 *___array_to_cipher* 方法转化为密文。具体程序如下：

```
def RSA_encrypt(self, text):
        text_array = self.__text_to_array(text)
        cipher_array = map(self.__encryptnum, text_array)
        cipher = self.__array_to_cipher(cipher_array)
        return cipher
```

RSA_decrypt() 函数同样也是整合的方法，主要实现 RSA 解密功能，其通过传递字串（密文）类型的参数，首先利用 *___cipher_to_array* 方法将密文转化为编码列表 cipher_array，然后利用 *map()* 操作对编码列表 cipher_array 的每个元素（编码）进行解密操作，解密后编码列表再通过 *___array_to_text* 方法转化为明文。具体程序如下：

```

    def RSA_decrypt(self, cipher):
        cipher_array = self.__cipher_to_array(cipher)
        text_array = map(self.__decryptnum, cipher_array)
        text = self.__array_to_text(text_array)
        if text[-1] == "\0":
            return text[:-1]
        return text
```

最后,RSA 类定义好了，通过 *RSA()* 实例化一个 RSA 对象，然后传递参数，调用 *RSA_encrypt* 和 *RSA_decrypt* 对其加解密即可。

3.4 测试环节

要求：由于程序只对 26 个英文字母和空格进行了字符编码，所以输入要求须为 26 个字母和空格，程序只接受字串元素为字母和空格，其他将会给出警告.

[3]其实列表数据结构就是一个 Interable 类型的数据结构，即迭代器，类似的还有字典，元祖等等

图 3.2: RSA 加密

图 3.3: 解密前密文

图 3.4: 解密后

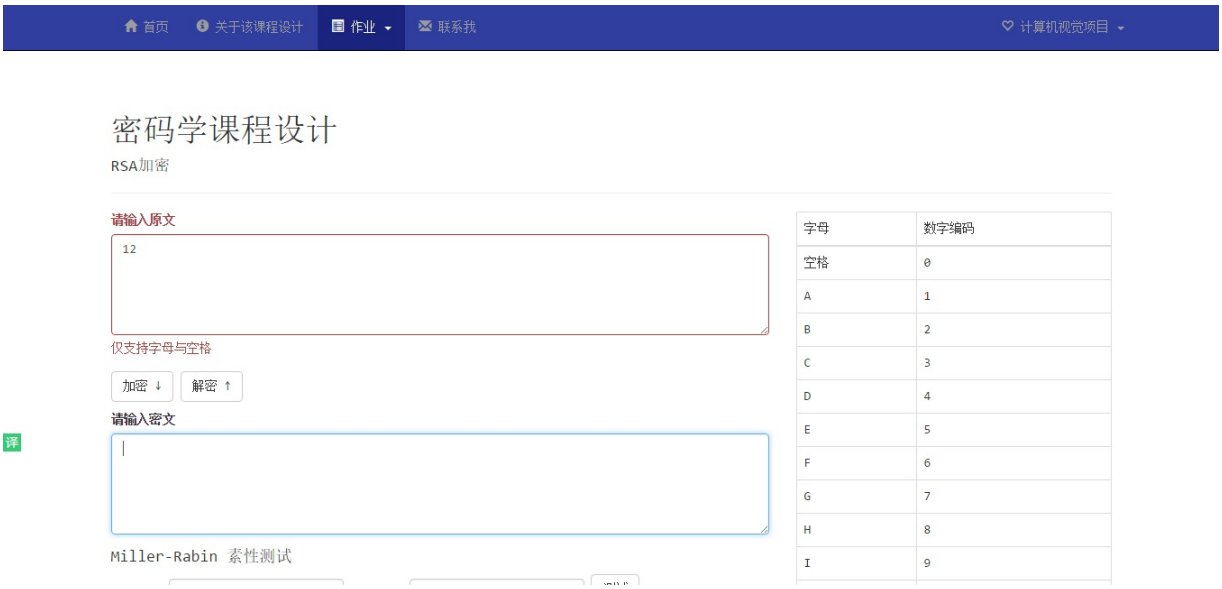

图 3.5: 输入不符合规格给出警告

参考文献

[1] 结城浩，图解密码技术，北京，人民邮电出版社，2015，98 ～ 123

[2] 朱文余，孙琦计算机密码应用基础，北京，科学出版社，2003，93 ～ 97

[3] Miguel Grinberg,Flask Web 开发: 基于 Python 的 Web 应用开发实战，北京，人民邮电出版社，2015,33 ～ 62

附录 A　RSA 源码

RSA.py

```
# coding: utf-8
from utils import *
import base64
from itertools import izip

class RSA(object):
    def __init__(self):
        self.n = 29*31
        self.e = 11
        self.d = 611

        self.__encryptnum = lambda m: calc_mod(m, self.e, self.n)
        self.__decryptnum = lambda c: calc_mod(c, self.d, self.n)

    def __chr_to_num(self, char):
        if char == " ":
            return 0
        elif char == "\0":
            return 27
        else:
            return ord(char)-64

    def __num_to_chr(self, num):
        if num == 0:
            return " "
        elif num == 27:
            return "\0"
        else:
            return chr(num+64)

    def __text_to_array(self, text):
        array = []
        if len(text)%2 != 0:
            text += "\0"
```

```
35
36          t = map(self.__chr_to_num, text)
37          for item in izip(t[::2], t[1::2]):
38              pre, after = tuple(item)
39              array.append(pre*28+after)
40
41          return array
42
43      def __array_to_text(self, array):
44          text = ""
45          for num in array:
46              pre = num//28
47              after = num%28
48              text += "".join(map(self.__num_to_chr, [pre, after]))
49          return text
50
51      def __array_to_cipher(self, array):
52          text = ""
53          for num in array:
54              pre = num//28
55              after = num%28
56              text += "".join(map(chr, [pre, after]))
57          return base64.encodestring(text)
58
59      def __cipher_to_array(self, cipher):
60          array = []
61          t = map(ord, base64.decodestring(cipher))
62          for item in izip(t[::2], t[1::2]):
63              pre, after = tuple(item)
64              array.append(pre*28+after)
65          return array
66
67      def RSA_encrypt(self, text):
68          text_array = self.__text_to_array(text)
69          cipher_array = map(self.__encryptnum, text_array)
70          cipher = self.__array_to_cipher(cipher_array)
71          return cipher
72
73      def RSA_decrypt(self, cipher):
74          cipher_array = self.__cipher_to_array(cipher)
75          text_array = map(self.__decryptnum, cipher_array)
76          text = self.__array_to_text(text_array)
77          if text[-1] == "\0":
78              return text[:-1]
79          return text
80
```

```
81 if __name__ == '__main__':
82     rsa = RSA()
83     a = "I LOVE YOU"
84     s = rsa.RSA_encrypt(a)
85     print s
86     d = rsa.RSA_decrypt(s)
87     print d
```

utils.py

```
1  # coding: utf-8
2  import timeit
3  import random
4
5  def calc_mod(x, r, n):
6      """Calculate the value of x**r mod n"""
7      a, b, c = x, r, 1
8      while b != 0:
9          while b%2 == 0:
10             b = b/2
11             a = a*a%n
12         else:
13             b = b-1
14             c = (c*a)%n
15     return c
16
17 def calc_reverse(m, n):
18     """Calculate the value of m^(-1) mod n"""
19     N = n
20     m %= n
21     m, n = n, m
22     q = []
23     while True:
24         q.append(m//n)
25         m, n = n, m%n
26         if n == 1:
27             break
28     P, Q = [1, q[0]], [0, 1]
29     for i in range(2, len(q)+1):
30         P.append(P[i-2]+P[i-1]*q[i-1])
31         Q.append(Q[i-2]+Q[i-1]*q[i-1])
32
33     return (-1)**len(q)*P[-1]%N
34
```

```
35 def is_prime(n, k=5000):
36     """Test whether n is a prime, repeat k times."""
37     def mill_rab(n):
38         b = 0
39         while b%2 == 0:
40             b = random.randint(2, n-1)
41 
42         s, m = 0, n-1
43         while m%2 == 0:
44             m //= 2
45             s += 1
46 
47         if calc_mod(b, m, n) == 1:
48             return True
49         else:
50             for r in xrange(s):
51                 if calc_mod(b, 2**r*m, n) == n-1:
52                     return True
53         return False
54 
55     if n in [0, 1]:
56         return False
57     elif n == 2:
58         return True
59     for i in range(k):
60         if not mill_rab(n):
61             return False
62     return True
63 
64 if __name__ == '__main__':
65     print(calc_reverse(11, 840))
```