

云南大学数学与统计学院  
上机实践报告

课程名称：近代密码学实验	年级：2015 级	上机实践成绩：
指导教师：陆正福	姓名：刘鹏	
上机实践名称：因子分解问题实验	学号：20151910042	上机实践日期：2018-05-27
上机实践编号：No.04	组号：	上机实践时间：14:04

一、实验目的

熟悉熟悉整数因子分解问题（IFP）及其有关的密码体制。

二、实验内容

1. 熟悉整数因子分解问题（IFP）及其有关的密码体制
2. 编程实现 RSA 体制。

三、实验平台

Windows 10 ProWorkstation 1803;  
*SageMath* version 8.2, Release Date: 2018-05-05;

四、实验记录与实验结果分析

1 题

编程实现基于整数因子分解问题（IFP）的密码体制。

Solution:

有关的密码体制见第 2 题。

2 题

编程实现 RSA 体制。

Solution:

背景材料

公钥密码学的发展是整个密码学发展历史中最伟大的一次革命，也许可以说是唯一的一次革命。从密码学产生至今，几乎所有的密码体制都是基于替换和置换这些初等方法。几千年来，对算法的研究主要是通过手工计算来完成的。转轮机和 DES 是密码学发展的重要标志，但是它们都是基于替换和置换这些初等方法之上的。

公钥密码学与其前的密码学完全不同。它是基于数学函数而不是基于 P/S 操作。

RSA 体制具有下述重要的特点：

1. 仅仅根据密码算法和加密密钥来确定解密密钥在计算上是不可行的
2. 两个密钥中的任何一个都可以用来加密，另一个用来解密

本实验报告主要依据教材的 9.2 节给出的有关内容。Diffie 和 Hellman 在其早期的著名论文中提出了一种新的密码学方法，事实上，他对密码学家提出了一种挑战，即要去寻找满足公钥体制要求的密码算法。之后很多算法被提出，其中有一些刚提出时似乎很有前途，但后来都被破解了。

MIT 的 Ron Rivest, Adi Shamir 和 Len Adleman 于 1977 年提出并于 1978 年首次发表的算法, 可以说是最早提出的满足有要求的公钥密码算法之一。Rivest-Shamir-Adleman (RSA) 算法自其诞生之日起就成为被广泛接受且被实现的通用公钥加密方法。

RSA 体制是一种分组密码, 其明文和密文均是 0 至某  $n-1$  之间的整数, 通常  $n$  的大小为 1024 位二进制数或 309 位十进制数, 也就是说  $n$  小于  $2^{1024}$ 。RSA 算法使用乘方运算, 明文以分组为单位进行加密, 每个分组的二进制值均小于  $n$ , 也就是说, 分组的大小必须小于或者等于  $\log_2(n) + 1$  位 (比如说  $n$  取值为  $8_{10}$ , 那么每个分组的二进制位数均要小于或者等于  $\log_2(8) + 1 = 4$ ? 这里并不需要加 1。)

对于明文分组  $M$  和密文分组  $C$ , 加密和加密的过程如下:

$$\begin{aligned} C &= M^e \bmod n \\ M &= C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n \end{aligned}$$

其中手法双方均已知  $n$ , 发送方已知  $e$ , 只有接收方已知  $d$ , 因此公钥加密算法的公钥为  $PU = \{e, n\}$ , 私钥为  $PR = \{d, n\}$ 。该算法要能用作公钥算法加密, 必须满足下列条件:

- (1) 可以找到  $e, d$  和  $n$ , 使得对所有  $M < n$ , 有  $M^{ed} \bmod n = M$ 。(  $d$  和  $e$  的积模  $\phi(n)$  为 1 时成立 )
- (2) 对所有的  $M < n$ , 计算  $M^e \bmod n$  和  $C^d$  是比较容易的。(快速模幂算法)
- (3) 由  $e$  和  $n$  确定  $d$  是不可行的。

首先需要知道对于两个素数  $p \neq q$ ,  $\phi(pq) = \phi(p)\phi(q) = (p-1)(q-1)$ 。可以简单地看一下这个公式, 如果在序列  $1, 2, \dots, pq-1$  里面找  $pq$  的非 1 因子, 那么这些因子必然以  $p$  或者  $q$  作为因子。所以分类讨论一下就是  $p, 2p, \dots, (q-1)p$  与  $q, 2q, \dots, (p-1)q$ , 所以  $\phi(pq) = (pq-1) - [(p-1) + (q-1)] = (p-1)(q-1)$ 。

**求证:**  $e$  与  $d$  互为模  $\phi(n)$  的乘法逆  $\Rightarrow M^{ed} \bmod n = M$

为了方便证明, 现在把证明中的条件稍微转化一下。  $ed \bmod \phi(n) = 1$ , 等价于存  $k \geq 0$ , 使得  $ed = k\phi(n) + 1 = k(p-1)(q-1) + 1$ 。

**求证:**  $ed = k\phi(n) + 1 = k(p-1)(q-1) + 1 \Rightarrow M^{ed} \bmod n = M$

**Proof:**

**Situation I:**  $(M, p) \neq 1$ , 即  $p|M$ ,  $M \bmod p = 0$ 。所以  $M^{k(p-1)(q-1)+1} \bmod p = 0$ , 从而直接得  $M^{k(p-1)(q-1)+1} \bmod p = M \bmod p$

**Situation II:**  $(M, p) = 1$ , 由欧拉定理可得  $M^{\phi(p)} \bmod p = 1$ 。

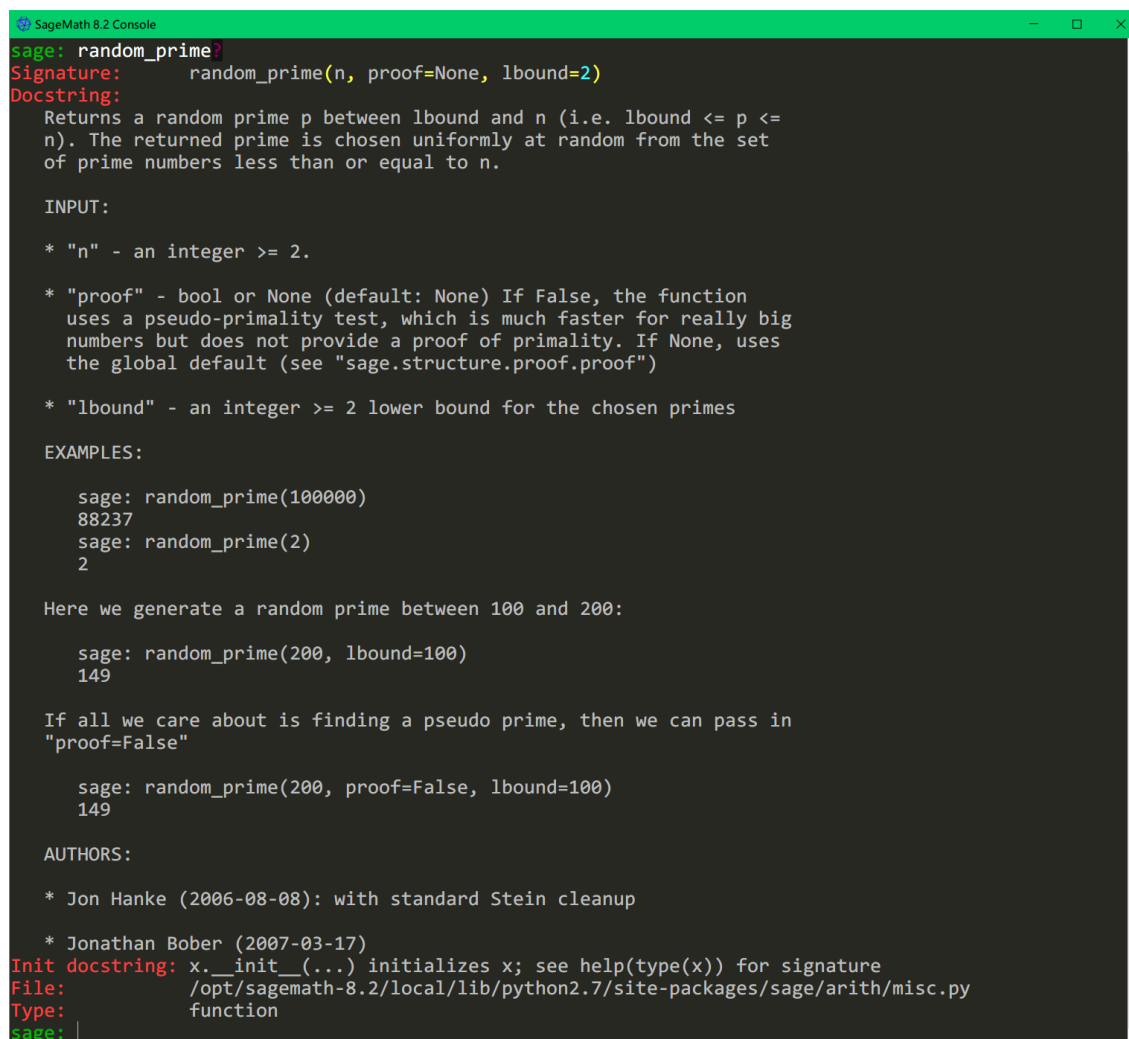
$$\begin{aligned} M^{k(p-1)(q-1)+1} \bmod p &= [(M) M^{k(p-1)(q-1)}] \bmod p \\ &= [(M) (M^{(p-1)})^{k(q-1)}] \bmod p \\ &= [(M) (M^{\phi(p)})^{k(q-1)}] \bmod p \\ &= (M \bmod p) \times [(M^{\phi(p)}) \bmod p]^{k(q-1)} \\ &= (M \bmod p) \times [1]^{k(q-1)} \\ &= M \bmod p \end{aligned}$$

综上, 现在有  $M^{ed} \bmod p = M^{k(p-1)(q-1)+1} \bmod p = M \bmod p$ , 同理可得  $M^{ed} \bmod q = M^{k(p-1)(q-1)+1} \bmod q = M \bmod q$ 。又因为  $p \neq q$  且  $p, q$  都是素数, 所以  $(p, q) = 1$ , 进一步,  $M^{ed} \bmod pq = M^{k(p-1)(q-1)+1} \bmod pq = M \bmod pq$ , 即  $M^{ed} \bmod n = M \bmod n$ 。□

在 RSA 算法里面,  $e$  只要比  $\phi(n)$  小而且满足  $(\phi(n), e) = 1$  即可。求与之匹配的  $d$  需要使用扩展的欧几里得算法。扩展的欧几里德算法: 给定两个正整数  $m$  和  $n$ , 我们计算它们的最大公因子  $d$  和两个整数  $a$  和  $b$ , 使得  $am + bn = d$ 。

由于  $ed \bmod \phi(n) = 1$ ，也就是说存在一个整数  $k$  使得等式  $d \cdot e + (-k) \cdot \phi(n) = 1$  成立。

SageMath 给出了几个很有用的工具包，下面是这两个包的使用手册。



```

SageMath 8.2 Console
sage: random_prime?
Signature:      random_prime(n, proof=None, lbound=2)
Docstring:
Returns a random prime p between lbound and n (i.e. lbound <= p <=
n). The returned prime is chosen uniformly at random from the set
of prime numbers less than or equal to n.

INPUT:

* "n" - an integer >= 2.

* "proof" - bool or None (default: None) If False, the function
uses a pseudo-primality test, which is much faster for really big
numbers but does not provide a proof of primality. If None, uses
the global default (see "sage.structure.proof.proof")

* "lbound" - an integer >= 2 lower bound for the chosen primes

EXAMPLES:

sage: random_prime(100000)
88237
sage: random_prime(2)
2

Here we generate a random prime between 100 and 200:

sage: random_prime(200, lbound=100)
149

If all we care about is finding a pseudo prime, then we can pass in
"proof=False"

sage: random_prime(200, proof=False, lbound=100)
149

AUTHORS:

* Jon Hanke (2006-08-08): with standard Stein cleanup

* Jonathan Bober (2007-03-17)
Init docstring: x.__init__(...) initializes x; see help(type(x)) for signature
File:          /opt/sagemath-8.2/local/lib/python2.7/site-packages/sage/arith/misc.py
Type:          function
sage:

```

Figure 1 random\_prime 包

```

SageMath 8.2 Console
Signature:      xgcd(a, b)
Docstring:
Return a triple "(g,s,t)" such that  $g = s * a + t * b = \gcd(a,b)$ .

Note: One exception is if a and b are not in a principal ideal
domain (see
https://en.wikipedia.org/wiki/Principal\_ideal\_domain), e.g., they
are both polynomials over the integers. Then this function can't
in general return "(g,s,t)" as above, since they need not exist.
Instead, over the integers, we first multiply g by a divisor of
the resultant of a/g and b/g, up to sign.

INPUT:

* "a, b" - integers or more generally, element of a ring for
which the xgcd make sense (e.g. a field or univariate
polynomials).

OUTPUT:

* "g, s, t" - such that  $g = s * a + t * b$ 

Note: There is no guarantee that the returned cofactors (s and t)
are minimal.

EXAMPLES:

sage: xgcd(56, 44)
(4, 4, -5)
sage: 4*56 + (-5)*44
4

sage: g, a, b = xgcd(5/1, 7/1); g, a, b
(1, 3, -2)
sage: a*(5/1) + b*(7/1) == g
True

sage: x = polygen(QQ)
sage: xgcd(x^3 - 1, x^2 - 1)
(x - 1, 1, -x)

sage: K.<g> = NumberField(x^2-3)
sage: g.xgcd(g+2)
(1, 1/3*g, 0)

sage: R.<a,b> = K[]
sage: S.<y> = R.fraction_field()[]
sage: xgcd(y^2, a*y+b)
(1, a^2/b^2, ((-a)/b^2)*y + 1/b)
sage: xgcd((b+g)*y^2, (a+g)*y+b)
(1, (a^2 + (2*g)*a + 3)/(b^3 + (g)*b^2), ((-a + (-g))/b^2)*y + 1/b)

Here is an example of a xgcd for two polynomials over the integers,
where the linear combination is not the gcd but the gcd multiplied
by the resultant:

sage: R.<x> = ZZ[]
sage: gcd(2*x*(x-1), x^2)
x
sage: xgcd(2*x*(x-1), x^2)
(2*x, -1, 2)
sage: (2*(x-1)).resultant(x)
2

Init docstring: x.__init__(...) initializes x; see help(type(x)) for signature
File:
/opt/sagemath-8.2/local/lib/python2.7/site-packages/sage/arith/misc.py
Type:
function
(END)

```

Figure 2 扩展欧几里德程序

利用 SageMath 给出的上面的两个工具包，可以实现一个简单的 RSA 加密解密过程

### 程序代码

```

1  #-----Code from Console-----
2
3  p = random_prime(2^64, 2^63)

```

```

4  q = random_prime(2^64, 2^63)
5  if p == q:
6      # The probability of p == q is very small.
7      raise ValueError("bad")
8  n = p * q
9  phi_n = (p-1) * (q-1)
10
11 e = ZZ.random_element(phi_n)
12 while gcd(e, phi_n) != 1:
13     e = ZZ.random_element(phi_n)
14 print "e = ", e
15 #-----
16
17 bezout = xgcd(e, phi_n); bezout
18 bezout = xgcd(e, phi_n); bezout
19 d = Integer(mod(bezout[1], phi_n))
20 print "d = ", d
21 print "de mod Phi(n) = ", mod(d*e, phi_n)
22
23 public_key = (n, e)
24 private_key = (p, q, d)
25
26 print """
27 The ASCII Value of "HELLO_WORLD"
28 +-----+
29 | H  E  L  L  O  W  O  R  L  D  |
30 | 72 69 76 76 79 87 79 82 76 68 |
31 +-----+
32 """
33
34 print "---public Key holder ALICE send message to private key holder BOB---\n"
35 print "ALICE ==> BOB"
36
37 m = "HELLO, WORLD!"
38 print "Message is ", m
39 m = [ord(x) for x in m];
40 m = ZZ(list(reversed(m)), 100)
41 print "m = ", m
42
43 if m < n:
44     print "Length of message is less than N, Cryption can be done..."
45 else:
46     raise "Block is too big"
47
48 C = power_mod(m, e, n)
49 print "The Crypted Message is ", C
50
51 M = power_mod(C, d, n)
52 print "Integer Bob received is", M

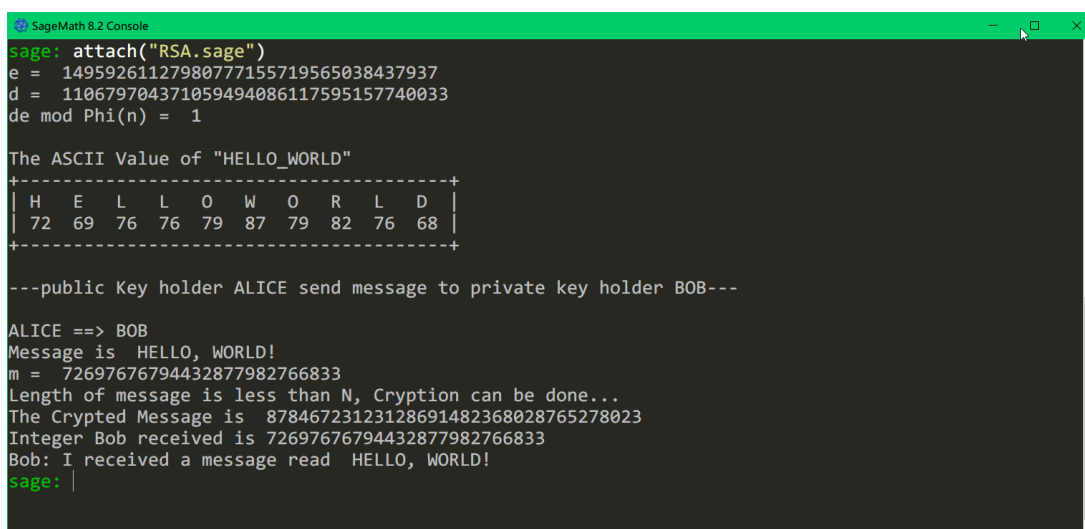
```

```

53
54 M = str(M)
55 message = ""
56 i = 0
57 while i <= len(M)-2:
58     tmp = int(M[i] + M[i+1])
59     tmp = chr(tmp)
60     message += tmp
61     i = i + 2
62 print "Bob: I received a message read ", message
63
64 #-----End of Code-----

```

## 运行结果



```

SageMath 8.2 Console
sage: attach("RSA.sage")
e = 14959261127980777155719565038437937
d = 110679704371059494086117595157740033
de mod Phi(n) = 1

The ASCII Value of "HELLO_WORLD"
+-----+
| H E L L O W O R L D |
| 72 69 76 76 79 87 79 82 76 68 |
+-----+

---public Key holder ALICE send message to private key holder BOB---

ALICE ==> BOB
Message is HELLO, WORLD!
m = 72697676794432877982766833
Length of message is less than N, Crypton can be done...
The Crypted Message is 87846723123128691482368028765278023
Integer Bob received is 72697676794432877982766833
Bob: I received a message read HELLO, WORLD!
sage:

```

## 六、实验体会

RSA 算法的核心子算法是快速幂取模算法与扩展欧几里德算法，RSA 算法所基于的安全条件是大整数的因子分解困难性。在实验中，我发现有函数可以算欧拉 $\phi$ 函数，所以认为可以通过计算欧拉函数来确定私钥。但是后来有文献<sup>[1]</sup>指出通过 $n$ 确定 $\phi(n)$ 与对 $n$ 进行因子分解是一样困难。所以目前来看，改算法在两个质数都比较大的时候，还是安全的。

## 七、参考文献

- [1] STALLINGS W. 密码编码学与网络安全：原理与实践 [M]. 6th ed. 北京：机械工业出版社, 2015.
- [2] [https://doc.sagemath.org/html/en/thematic\\_tutorials/numtheory\\_rsa.html](https://doc.sagemath.org/html/en/thematic_tutorials/numtheory_rsa.html)