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 **R**ivest，Adi **S**hamir和Len **A**dleman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。**R**ivest-**S**hamir-**A**dleman（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至某之间的整数，通常的大小为位二进制数或位十进制数，也就是说小于。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，也就是说，分组的大小必须小于或者等于位（比如说取值为，那么每个分组的二进制位数均要小于或者等于？这里并不需要加1。）

对于明文分组和密文分组，加密和加密的过程如下：

其中手法双方均已知，发送方已知，只有接收方已知，因此公钥加密算法的公钥为，私钥为。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，和，使得对所有，有。（和的积模为时成立）
2. 对所有的，计算和是比较容易的。（快速模幂算法）
3. 由和确定是不可行的。

首先需要知道对于两个素数，。可以简单地看一下这个公式，如果在序列里面找的非因子，那么这些因子必然以或者作为因子。所以分类讨论一下就是与，所以。

**求证**：与互为模的乘法逆

为了方便证明，现在把证明中的条件稍微转化一下。，等价于存，使得。

**求证**：

**Proof:**

**Situation Ⅰ**: ，即, 。所以，从而直接得

**Situation Ⅱ**: ，由欧拉定理可得。

综上，现在有，同理可得。又因为且，都是素数，所以，进一步，，即。**□**

在RSA算法里面，只要比小而且满足即可。求与之匹配的需要使用扩展的欧几里得算法。扩展的欧几里德算法：给定两个正整数和，我们计算它们的最大公因子和两个整数和，使得。

由于，也就是说存在一个整数使得等式成立。

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。

图片包含 屏幕截图, 文字

已生成极高可信度的说明

Figure 1 random\_prime包

图片包含 屏幕截图, 文字

已生成高可信度的说明

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  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64 | #-------------Code from Console-------------  p **=** random\_prime**(**2**^**64**,** 2**^**63**)**  q **=** random\_prime**(**2**^**64**,** 2**^**63**)**  **if** p **==** q**:**  # The probability of p == q is very small.  **raise** ValueError**(**"bad"**)**  n **=** p **\*** q  phi\_n **=** **(**p**-**1**)** **\*** **(**q**-**1**)**  e **=** ZZ**.**random\_element**(**phi\_n**)**  **while** gcd**(**e**,** phi\_n**)** **!=** 1**:**  e **=** ZZ**.**random\_element**(**phi\_n**)**  **print** "e = "**,** e  #-------------------------------------  bezout **=** xgcd**(**e**,** phi\_n**);** bezout  bezout **=** xgcd**(**e**,** phi\_n**);** bezout  d **=** Integer**(**mod**(**bezout**[**1**],** phi\_n**))**  **print** "d = "**,** d  **print** "de mod Phi(n) = "**,** mod**(**d**\***e**,** phi\_n**)**  public\_key **=** **(**n**,** e**)**  private\_key **=** **(**p**,** q**,** d**)**  **print** """  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 |  +----------------------------------------+  """  **print** "---public Key holder ALICE send message to private key holder BOB---\n"  **print** "ALICE ==> BOB"    m **=** "HELLO, WORLD!"  **print** "Message is "**,** m  m **=** **[**ord**(**x**)** **for** x **in** m**];**  m **=** ZZ**(**list**(**reversed**(**m**)),** 100**)**  **print** "m = "**,** m  **if** m **<** n**:**  **print** "Length of message is less than N, Cryption can be done..."  **else:**  **raise** "Block is too big"  C **=** power\_mod**(**m**,** e**,** n**)**  **print** "The Crypted Message is "**,** C  M **=** power\_mod**(**C**,** d**,** n**)**  **print** "Integer Bob received is"**,** M  M **=** str**(**M**)**  message **=** ""  i **=** 0  **while** i **<=** len**(**M**)-**2**:**  tmp **=** int**(**M**[**i**]** **+** M**[**i**+**1**])**  tmp **=** chr**(**tmp**)**  message **+=** tmp  i **=** i **+** 2  **print** "Bob: I received a message read "**,** message  #-------------End of Code------------- |

运行结果

图片包含 屏幕截图

已生成极高可信度的说明

# 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[1]指出通过确定与对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