摘 要

1978 年 Rivest、Shamir 和 Adleman 提出的 RSA 公钥密码体制是公钥密码体制的代表。它在加密、解密和数字签名等方面都占有重要的地位，使用也最广泛。但是，RSA 公钥密码体制是一种确定型公钥密码体制，因而无法抵抗选择明文攻击。针对这一弱点，Goldwasser 和 Micali 在 1984 年提出概率加密的思想，即概率密码体制。这种体制可以抵抗选择明文攻击，但是由于其密文膨胀率太大而无实用价值。随后，Blum 和 Goldwasser 于 1985 年设计出一种效率较高的概率加密方案，简称为 BG 概率公钥密码体制，将密文膨胀率降至$1+k/t$，基本上满足实用要求。近年来，有一些关于概率公钥密码方面的研究成果，但密文膨胀率都较高。

本文的研究工作是在 RSA 公钥密码体制的基础上，进行概率公钥密码算法的设计展开的，具体工作包括：

1. 设计基于 RSA 的概率公钥密码算法，该算法不仅具有多项式安全性而且密文膨胀率为 1，但加密、解密速度较慢，特别是在加密较长消息时，效果不好。因而针对该算法又进行改进，证明了改进算法具有多项式安全性并对算法的性能进行了具体分析。

2. 设计基于 RSA 的概率数字签名方案。针对该数字签名方案，作者用 C 和 C++语言编写一个具体的程序，运行程序用具体实例验证了方案的可行性。

关键词 RSA，概率密码，数字签名，多项式安全性

ABSTRACT

RSA public key cryptography was created by Rivest, Shamir and Adleman in 1978, it represents the public key cryptography system. RSA is the most universe application in the public key cryptography, and has taken the important position in encryption、discryption and signature. It is deterministic cryptology and has no security against chosen cleartext attachs, so the probabilistic encryption scheme has been proposed by Goldwasser and Micali in 1984, but it can not be used because of its higher message expansion. A new system is designed by Blum and Goldwasser in 1985, which can lower the message expansion to $1+k/t$, and it has been presented to meet application needs. There are many good public key cryptography schemes recently, but the schemes' message expansion is still higher.

The paper works on RSA public key cryptography, and proposes RSA probabilistic public key cryptography scheme. The following is my main research works:

1. The paper designs a probabilistic cryptography scheme based on RSA, which has polynomial secure and the message expansion is 1. The improved scheme is designed because of the low efficiency of longer message, it is proved have polynomial security and the writer analyses it' s function .

2. The paper presents a probabilistic digital signature scheme based on RSA and a program is designed based on this algorithm, then we get an example

of this digital signature scheme by operating the program.

KEY WORDS RSA, probabilistic cryptography, digital signature, polynomial security

中央民族大学研究生学位论文作者声明

本人声明：本人呈交的学位论文是本人在导师指导下取得的研究成果。对前人及其他人员对本文的启发和贡献已在论文中作出了明确的声明，并表示了谢意。论文中除了特别加以标注和致谢的地方外，不包含其他人和其它机构已经发表或者撰写过的研究成果。

本人同意学校根据《中华人民共和国学位条例暂行实施办法》等有关规定保留本人学位论文并向国家有关部门或资料库送交论文或者电子版，允许论文被查阅和借阅；本人授权中央民族大学可以将本人学位论文的全部或者部分内容编入有关数据库进行检索，可以采用影印、缩印或者其它复制手段和汇编学位论文（保密论文在解密后应遵守此规定）。

作者签名：贾杰　　日期：2009年6月22日

绪 论

一、课题背景和意义

21 世纪是信息化的时代，人们每天都会有大量的信息要进行传输、交换、存储和处理，同时信息在社会中的地位和作用越来越重要，已成为社会发展的重要战略资源。一方面，它为人类的生产和生活带来很多的便利，有力地推动了人类的文明进步和社会发展；另一方面，它也带来了许多新的问题，其中信息的安全性就是一个值得特别关注的问题。

密码技术是信息安全技术的核心，它是实现保密性、数据完整性、实体认证、不可否认性的关键。纵观密码学的发展过程，可以把它大致分为古典密码体制、传统的密码体制和公钥密码体制三个阶段[1]。其中，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。因为它与传统密码学完全不同，公钥算法是基于数学函数而不是基于替换和置换，更重要的是，它是非对称的密码体制，使用两个独立的密钥，因而在消息的保密性、密钥分配和认证领域有着重要意义。

到目前为止，RSA 公钥密码体制在加解密和数字签名方面都占有重要的地位，是使用最广泛的公钥密码体制。经过实践的考验，它的安全性没有受到很大的冲击，当前它仍有很大的发展前景。但是，RSA 公钥密码体制是一种确定型公钥密码体制，因而无法抵抗选择明文攻击。针对这一弱点，1984 年 Goldwasser 和 Micali 提出了概率加密思想[2]（简称 GM）。近年来，概率公钥加密在密码学界备受重视，因为该密码体制具有目前所有已知的密码体制中的最高安全性即多项式安全性。所以本文主要集中于RSA概率公钥密码算法领域的研究。

二、课题研究概况

GM 概率加密算法的理论基础是二次剩余的难解问题，采用逐比特位加密的方法对明文进行加密，因而较好地克服了确定型公钥密码体制的本质缺陷。但是它的密文膨胀率太大，因而没有什么实用价值。为了解决这一问题，Blum 和 Goldwasser 在 1985 年

提出了另外一种方法，即 BG 概率公钥密码体制[3]，将密文膨胀率降至$1+k/t$（其中k表示体制规模的安全参数，t表示明文长度），基本上满足实用要求。但是，该算法不能进行数字签名，同时还存在着密文中含有密钥种子的代数结构的缺陷。

近年来，取得了一些关于概率公钥密码方面的研究成果。例如：1986 年何敬民提出随机迭代加密方法[4]（简称 RIE），它在时间复杂性和密文膨胀率方面都比 BG 方法略好一点；1989 年，李大兴和张泽增提出一种基于 RSA 的概率公钥密码体制[5]（PEC-RSA），它通过引入随机参数，利用概率加密思想将明文逐位迭代加密而实现。其加、解密算法具有与 RSA 相同数量级的时间复杂性，密文膨胀率与 BG 体制相同；1996 年李献刚[6]等提出基于 Rabin 体制的概率加密算法，并证明了安全性，但该体制是解密多值的，因此当明文不是文本而是随机比特流时，其解密将发生困难；1997 年刘花云[7]等提出一种基于 RSA 的概率加密算法，利用随机比特来填充明文，但要求通信双方需要秘密约定随机比特填充的位置，而且没有解决密文膨胀率较高的问题；2002 年赵泽茂[8]在随机比特思想中引入奇、偶扩展明文的概念，但算法中扩展明文影响了信息的传输速率；2007 年王小非、崔国华[9]等采用时间戳和 hash 函数技术，并利用以 Blum 数为模的二次同余式中求平方根的不可计算性，设计一个概率公钥密码系统，使密文膨胀率降为 1，但它要求用户选择的模数n为 Blum 数，而且运算速度比较慢，对加密较长消息效果不好。

三、论文组织结构

本文致力于对 RSA 公钥密码体制进行概率加密的研究，使其克服选择明文攻击。针对已有的概率公钥密码算法的研究成果，作者设计了基于 RSA 的概率公钥密码算法。算法的基础是数论的欧拉定理，它的安全性依赖于大数因子分解的困难性，对模数n的选择和 RSA 的公钥密码算法一样，是两个大素数的乘积。这一算法不仅具有多项式安全性而且密文膨胀率为 1，但其加密速度和王小非等人提出的概率公钥密码算法一样，都比较慢，对加密较长消息效果不好。因而又对该算法进行改进，提高其加密、解密速度，同时还具体分析了算法的性能及其安全性。本文设计了基于 RSA 的概率数字签名方案，用 C 和 C++语言编写程序，并用实例验证方案的可行性。

本文各章节的组织结构如下：

第一章介绍关于密码学的一些基本概念、术语，密码体制的分类和主要的密码分析技术。

第二章介绍构造公钥密码体制的单向陷门函数，RSA 算法的思想，并对 RSA 的安

全性进行分析，指出该算法在使用过程中存在的一些问题及应注意的事项。

第三章前两节分别介绍概率密码体制的数学基础及 GM 和 BG 两种概率密码体制。第三节在 RSA 体制的基础上，设计基于 RSA 的概率公钥密码算法并分析评价算法。第四节针对第三节算法的不足设计相应的改进算法，对改进算法的性能及安全性方面进行具体分析。

第四章前三节简要介绍数字签名原理、功能以及 RSA 数字签名方案。第四节设计基于 RSA 的概率数字签名方案。同时针对该方案编写程序，用实例验证方案的可行性。

第五章对本论文的工作进行总结。

第一章　密码学概述

密码学是研究信息系统安全保密的科学，密码学理论是保障网络信息安全的核心理论。密码学是集数学、计算机科学、电子与通信等诸多学科于一身的交叉学科。它不仅能够保证机密性信息的加密，而且能够实现数字签名、身份验证、系统安全等功能。随着计算机网络的延伸与渗透，尤其是近年来电子商务、网上银行等方面的蓬勃发展，密码学的重要地位也日趋明显。

密码学是一门既古老而又年轻的科学，它最早的应用可追溯到几千年前的古罗马，从古代社会以来，密码技术一直在军事领域中被应用，特别是在 20 世纪两次世界大战期间，密码技术在信息保密方面发挥了十分重要的作用。但它成为一门独立的学科则是从近几十年才开始的。1949 年，C.E.Shannon 发表的论文《保密系统的信息理论》[10]，标志着密码学这门新学科的诞生。1976 年，由 Diffie 和 Hellman 发表的《密码学的新方向》[11]提出公钥密码学的新思想，标志着现代密码学的开端，开创了公钥密码学的新纪元。1977 年美国数据加密标准 DES 的正式发布和 1978 年 R.L.Rivest，A.shamir 与 L.Adleman 提出的 RSA 公钥密码体制[12]是现代密码学发展史上的两个重要的里程碑。

本章介绍密码学的基础知识，包括密码学的基本概念、密码体制以及密码分析技术。

第一节　密码学基本概念

密码学所包含的内容广泛，在此我们仅介绍一下本文所使用的密码学的一些基本概念[13]。

定义 1.1　密码学（Cryptology）是研究信息系统安全保密的科学。它包括两个分支，即密码编码学和密码分析学。

定义 1.2　密码编码学（Cryptography）是对信息进行编码实现信息隐蔽的技术和科学。从事这种工作的人叫密码编码者。密码编码学的主要目的是保持明文（或密钥，或明文和密钥）的秘密以防止偷听者（也叫对手、攻击者、截取者、入侵者、敌手）知晓。

定义 1.3　密码分析学（Cryptanalysis）是研究分析破译密码的技术与科学。密码分析学的目的是在不知道密钥的情况下，恢复出明文。成功的密码分析能恢复出消息的明

文或密钥。密码分析也可以发现密码体制的弱点，最终得到上述结果（密钥通过非密码分析方式的丢失叫做泄露）。

定义 1.4　明文（Plaintext）是指发送方想要发送给接受方的消息。

定义 1.5　密文（Cipher text）是指明文被加密后的消息。

定义 1.6　加密（Encryption）是将明文变换为密文的过程。

定义 1.7　解密（Decryption）是将密文恢复为明文的过程。

第二节　密码体制分类

通常人们用香农（Shannon）的密码模型来说明密码体制。它由以下几部分组成：明文空间 M，密文空间 C，密钥空间 K_1 和 K_2。加密变换 $E_{k_1}: M \to C$，由加密器完成；解密变换 $D_{k_2}: C \to M$，由解密器完成。称六元 $(M, C, K_1, K_2, E_{k_1}, D_{k_2})$ 为一保密系统。根据加密和解密所使用的密钥是否相同，可以把密码体制分为对称密码体制和非对称密码体制。

一、对称密码体制

对称密码体制是一种传统密码体制，也称为私钥密码体制或单钥密码体制。即加解密双方在加解密过程中使用完全相同的密钥。因为加解密密钥相同，需要通信的双方必须选择和保存他们共同的密钥，各方必须信任对方不会将密钥泄密出去，这样就可以实现数据的机密性和完整性。单钥密码体制的安全性主要取决于算法和密钥的长度，泄漏密钥就意味着任何人都能对消息进行加密和解密。

单钥密码体制不仅可以用于数据加密，也可以用于消息认证。但是单钥密码体制存在着严重的缺陷：在进行安全通信之前，通信的双方必须通过安全信道商定和传送密钥，而在实际的通讯网中，通信双方很难确定一条合理的安全通道；另一个问题就是密钥的管理，对于具有 n 个用户的网络，需要 $n(n-1)/2$ 个密钥，在用户群不是很大的情况下，对称加密系统是有效的。但是对于大型网络，当用户群很大，分布很广时，密钥管理的开销极大，密钥只能记录在计算机内存或外存上，这是极不安全的，因而密钥的分配和保存就成了问题。

比较典型的对称密码算法有 DES[14]（Data Encryption Standard 数据加密标准）算法

及其变形 Triple DES[15](三重 DES)，GDES（广义 DES)；欧洲的 IDEA 等。DES 标准由美国国家标准局提出，主要应用于银行业的电子资金转帐（EFT）领域。DES 的密钥长度为 56bit。Triple DES 使用两个独立的 56bit 密钥对交换的信息进行 3 次加密，从而使其有效长度达到 112bit。

对称密码算法的优点是计算开销小，加密速度快，是目前比较常见的信息加密算法之一。它的局限性在于它存在着通信的贸易关系，要维护专用密钥。它也没法鉴别贸易发起方或贸易最终方，因为贸易的双方的密钥相同。另外，由于对称加密系统仅能用于对数据进行加解密处理，提供数据的机密性，不能用于数字签名。因而人们迫切需要寻找新的密码体制。

二、非对称密码体制

非对称密码体制也叫公钥密码体制，该技术就是针对私钥密码体制的缺陷被提出来的。在公钥加密系统中，加密和解密是相对独立的，加密和解密会使用两把不同的密钥，加密密钥（公钥）向公众公开，谁都可以使用，解密密钥（私钥）只有解密人自己知道，非法使用者根据公开的加密密钥无法推算出解密密钥，故其称为公钥密码体制。如果一个人选择公布了他的公钥，另外任何人都可以用这一公钥来加密传送消息给他。私钥是秘密保存的，只有私钥的所有者才能利用私钥对密文进行解密。也就是说在公钥密码体制中，通信双方无须事先交换密钥就可建立起保密通信。公钥密码体制克服了单钥密码体制的缺点，特别适用于计算机网络中的多用户通信，它大大减少了多用户通信所需要的密钥量，节省了系统资源，也便于密钥管理。公钥密码体制的算法中最著名的代表是 RSA 系统。之后，人们基于不同的计算问题，提出了大量的公钥密码算法。比较重要的有 RSA 算法、Merkle-Hellman 背包算法、McEliece 算法、ELGamal[16]算法和椭圆曲线密码[17]算法等。

设计公钥密码体制的关键是先要寻找一个合适的单向函数，大多数的公钥密码体制都是基于计算单向函数的逆的困难性而建立的。例如，RSA体制就是典型的基于单向函数模型的实现。这类密码的强度取决于它所依据的问题的计算复杂性。值得注意的是，公钥密码体制的安全性是指计算安全性，而绝不是无条件安全性，这是由它的安全性理论基础即复杂性理论决定的。单向函数在密码学中起一个中心作用[18]。它对公钥密码体制的构造的研究是非常重要的。虽然目前许多函数(包括RSA算法的加密函数)被认为或被相信是单向的，但目前还没有一个函数能被证明是单向的。

公钥加密系统可提供以下功能：1. 保密性(confidentiality)：保证非授权人员不能非

法获取信息，通过数据加密来实现；2. 数据完整性（data integrity）保证信息内容不被篡改，入侵者不可能用假消息代替合法消息，通过数字签名来实现；3. 实体认证（entity authentication）保证对方属于所声称的实体，通过数字签名来实现；4. 不可否认性（non-repudiation）发送者不可能事后否认他发送过的消息，消息的接受者可以向中立的第三方证实所指的发送者确实发出了消息，通过数字签名来实现。可见公钥加密系统满足信息安全的所有主要目标。

在至今为止的所有公钥密码体系中，RSA 系统是其中最著名的一种。RSA 公开密钥密码系统是由 R.Rivest、A.Shamir 和 L.Adleman 三位教授于 1977 年提出的。RSA 的取名就是来自于这三位发明者的姓的第一个字母。RSA 算法的优点是密钥空间大，缺点是加密速度慢，如果 RSA 和 DES 结合使用，则正好弥补 RSA 的缺点。即 DES 用于明文加密，RSA 用于 DES 密钥分配的问题。公钥密码学仅限于用在密钥管理和签名这类应用中，这几乎是已被广泛接受的事实。

与传统的对称密码体制相比较，非对称密码体制具有如下的优点：

1. 密钥分发简单。由于加密和解密密钥不同，而且不能从加密密钥推导出解密密钥，因而加密密钥表可以像电话号码本一样分发。

2. 需要保存的密钥量减少，存储的空间减少。

3. 公钥的出现使得非对称密码体制可以适应开发的使用环境。

4. 可以实现数字签名。所谓数字签名，主要是为了保证接收方能够对公正的第三方证明其收到的报文的真实性和发送源的真实性而采取的一种安全措施。即要求接收方能确认发送方的签名，而发送方签名以后，不能否认其签名，一旦发生矛盾，第三方能仲裁收发方的问题。

运用公开密钥密码体制技术实施构建完整的加密及签名体系，可以有效地解决对于保障信息保密性、真实性、完整性、不可抵赖性的难题，在充分利用互联网实现资源共享的前提下，能够真正意义上确保网上交易与信息传递的安全。

第三节　密码分析技术

密码分析学是密码学的一个重要分支，与密码编码学并列。一般的，密码分析学是研究在不掌握密钥的情况下，利用密码体制的弱点来恢复明文或密钥的一门学科。密码系统可能遭受的攻击有被动攻击和主动攻击。其中被动攻击(Passive attack)是对一个密

码系统采取截获密文进行分析的攻击。而主动攻击(Active attack)是指非法入侵者主动向系统窜扰，采用删除、更改、增添、重放、伪造等手段向系统中插入假消息，以达到损人利己的目的。如果能够根据密文确定出明文或密钥，或者能够根据明文与密文对确定出密钥，那我们说这个密码是可破译的。否则，我们说这个密码是不可破译的。

密码分析者攻击密码的方法通常主要有以下三种[19]：

1. 穷举攻击：是指密码分析者用试遍所有密钥的方法来破译密码。穷举攻击所花费的时间等于尝试次数乘以一次解密(加密)所需的时间。显然可以通过增大密钥量或加大解密(加密)算法的复杂性来对抗穷举攻击。当密钥量增大时，尝试的次数必然增大。当解密(加密)算法的复杂性增大时，完成一次解密(加密)所需的时间增大。从而使穷举攻击在实际上不能实现。

2. 统计分析攻击：是指密码分析者通过分析密文和明文的统计规律来破译密码。统计分析攻击在历史上为破译密码做出过极大的贡献。许多古典密码都可以通过分析明文字母和字母组的频率而破译。对抗统计分析攻击的方法是设法使明文的统计特性不带入密文。这样，密文不带有明文的痕迹，从而使统计分析成为不可能。

3. 数学分析攻击：是指密码分析者针对加密算法的数学依据通过数学求解的方法来破译密码。为了对抗这种数学分析攻击，应选用具有坚实数学基础和足够复杂的加密算法。

此外，根据攻击密码分析者可利用的数据来分类，将破译密码的类型分为以下三种：

第一种是唯密文攻击：即密码分析者仅根据截获的密文来破译密码。唯密文攻击是最容易防范的，因为攻击者拥有的信息量最少。

第二种是已知明文攻击：即密码分析者根据已经知道的某些明文与密文对来破译密码。与已知明文攻击紧密相关的是可能词攻击。如果攻击者处理的是一般散文信息，他可能对信息的内容一无所知，但是如果他处理的是一些特定的信息，他就可能知道其中的部分内容。比如，对于一个完整的会计文件，攻击者可能知道放在文件最前面的是某些密钥词。又比如，某某公司开发的程序源代码可能含有该公司的版权信息，并且放在某个标准位置。近代密码学认为，一个密码仅当它能够经得起已知明文攻击时才是可取的。

第三种是选择明文攻击：即密码分析者能够选择明文并获得相应的密文。这是对密码分析者最有利的情况。计算机文件系统和数据库特别容易受到这种攻击，因为用户可随意选择明文，并得到相应的密文文件和密文数据库。

除了上述三种攻击之外，还有两种类型的攻击方法：选择密文攻击和选择文本攻击。

它们在密码分析技术中很少用到，但是不失为两种较好的可能攻击方法。

密码编码学的任务是寻求生成高强度密码的有效算法，满足对消息进行加密或认证的要求。密码分析学的任务是破译密码或伪造认证密码，窃取机密信息或进行诈骗破坏活动。进攻与反进攻、破译与反破译是密码学中永无止境的矛盾。一个密码，如果无论密码分析者截获了多少密文和用什么方法进行攻击都不能被攻破，则称为是绝对不可破译的或无条件安全的。绝对不可破译的密码在理论上是存在的。但是，如果能够利用足够的资源，那么任何实际的密码都是可破译的。因此，对我们更有实际意义的是在计算上不可破译的密码，如果一个密码不能被密码分析者根据可利用的资源所破译，则称为在计算上是不可破译的（计算上安全的）。因此，加密算法的使用者应挑选尽量满足以下标准的算法：

1. 破译密码的代价超出密文信息的价值。

2. 破译密码的时间超出密文信息的有效生命期。

第二章　RSA公钥密码体制

公钥密码体制有多种实现方式，RSA算法是其中的典型代表。1978年美国麻省理工学院三名密码学者R.L.Rivest、A.Shamir、L.M.Adleman发表了题为《数字签名和公钥密码的一个方法》的论文，提出了RSA密码算法。RSA算法的基础是数论的欧拉定理，它的安全性依赖于大数因子分解的困难性，至今没有很好的破解方法，并且RSA算法既可以用于加密，又可以用于数字签名，因此RSA公钥密码算法在信息交换过程中使用比较广泛，安全性很高。本章我们讨论RSA算法。首先介绍RSA算法的数学基础，给出算法的描述，然后对RSA算法的安全性进行分析，指出该算法在使用过程中存在的一些问题及应注意的事项。

第一节　数学基础

RSA的基础是数论的Euler定理，它的安全性依赖于大数因子分解的困难性，即在已知两个大素数的情况下，可以很容易求得两者之积；但是反过来，如果已知两个大素数的乘积，想求出这两个大素数是相当困难的。即它的安全性主要取决于构造其加密算法的数学函数求逆的困难性，我们称这样的函数为单向函数。单向函数在密码学中起一个中心作用。它对公钥密码体制的构造是非常重要的。单向函数的研究是公钥密码体制理论中的一个重要课题。但是，虽然很多函数(包括RSA算法的加密函数)被认为或被相信是单向的，但目前还没有一个函数能被证明是单向的。下面介绍RSA公钥密码体制的数学基础。

一、单向函数

定义2.1（单向函数[20]）　单向函数（One-way function）是满足下列性质的函数：每个函数值都存在惟一的逆，并且计算函数值是容易的，但求逆却是不可行的：

$$y = f(x) \qquad \text{容易}$$

$$x = f^{-1}(y) \quad \text{不可行}$$

通常，“容易”是指一个问题可以在输入长度的多项式时间内得到解决。“不可行”

的定义比较模糊，一般而言，若解决一个问题所需的时间比输入规模的多项式时间增长更快，则称该问题是不可行的。

定义2.2（单向陷门函数[20]） 一个函数，若计算函数值很容易，并且在缺少一些附加信息时计算函数的逆是不可行的，但是已知这些附加信息时，可在多项式时间内计算出函数的逆，那么我们称这样的函数为单向陷门函数（Trap-door one-way function），即单向陷门函数是满足下列条件的一类单向函数 f_k：

若 k 和 x 已知，则容易计算 $y=f_k(x)$；

若 k 和 y 已知，则容易计算 $x=f_k^{-1}(y)$；

若 y 已知但 k 未知，则计算出 $x=f_k^{-1}(y)$ 是不可行的。

由此可见，寻找合适的单向陷门函数是公钥密码体制应用的关键。单向陷门函数是贯穿整个公钥密码体制的一个核心概念。单向陷门函数之所以称为“单向”，就是这样的函数可以轻易地计算一个方向，但另一个方向却是非常难以计算的。而“陷门”意味着，只有加上特定的额外信息，才能将任何经加密函数加密的信息有效地解密，以RSA公开密钥密码系统而言，陷门信息就是对模数因数分解的信息。

二、相关概念及定理[21]

定义2.3 设 a，b 是任意两个整数，其中 $b\neq 0$，如果存在一个整数 q 使得等式 $a=bq$ 成立，就称 b 整除 a 或者 a 被 b 整除，记作 $b\mid a$。

定义2.4 若 $a\mid b$ 且 $a\mid c$，就说 a 是 b 和 c 的公因子。

定义2.5 若 a 是 b 和 c 的公因子，且 b 和 c 的每一个公因子都除尽 a，则称 a 是 b 和 c 的最大公因子，用 $\gcd(b,c)$ 或 (b,c) 表示它，即

$$a=\gcd(b,c) \text{ 或 } a=(b,c)$$

定义2.6 设 m 是一个正整数，则 $m-1$ 个整数中 $1,2,\cdots m-1$ 中与 m 互素的数的个数记作 $\varphi(m)$，通常叫做欧拉（Euler）函数。

定义2.7 若 $m\mid a-b$，即 $a-b=km$，我们就说 a 和 b 模 m 同余，记为

$$a\equiv b(\mathrm{mod}\, m)$$

m 被称为这个同余式的模。

RSA的理论基础是数论中的欧拉定理。

定理2.1（欧拉定理） 设 m 是大于1的整数，$(a,m)=1$，则 $a^{\varphi(m)}\equiv 1(\mathrm{mod}\, m)$。

推论2.1（Fermat定理） 若 p 为素数，$(a,p)=1$，则 $a^p\equiv a(\mathrm{mod}\, p)$。

定理2.2 若 $(m,n)=1$，则有 $\varphi(mn)=\varphi(m)\varphi(n)$。一般的，对 $n=\prod_{i=1}^{t}p_i^{\alpha_i}$，可证明

$\varphi(n)=n\prod_{i=1}^{t}(1-1/p_i)$。

定理2.3　若$m\geq 1$，$(a,m)=1$，则存在c使得$ca\equiv 1(\mathrm{mod}\,m)$，我们把$c$称为是$a$模$m$的逆，记作$a^{-1}(\mathrm{mod}\,m)$或$a^{-1}$。

定理2.4　（Wilson定理）　设p是一个素数，则$(p-1)!\equiv -1(\mathrm{mod}\,p)$。

定理2.5　设p和q是两个不同的素数，$n=pq$，对任意的整数$x(0\leq x<n)$及任意的非负整数k，有$x^{k\varphi(n)+1}\equiv x(\mathrm{mod}\,n)$。

证明：如果p不整除x，则由推论2.1(Fermat定理)知，$x^{p-1}\equiv 1(\mathrm{mod}\,p)$，而$\varphi(p)=p-1$整除$\varphi(n)$，所以$x^{k\varphi(n)+1}\equiv x^{k\varphi(n)}x\equiv x(\mathrm{mod}\,p)$。

如果p整除x，则$x\equiv 0(\mathrm{mod}\,p)$，显然$x^{k\varphi(n)+1}\equiv x(\mathrm{mod}\,p)$也成立。

同理可证，$x^{k\varphi(n)+1}\equiv x(\mathrm{mod}\,q)$恒成立。

假设$x^{k\varphi(n)+1}=tp+x$（t为非负整数），且$x^{k\varphi(n)+1}=t'q+x$（t'为非负整数），所以$tp+x=t'q+x$，即$tp=t'q$。

由于$(p,q)=1$，故q整除t（同样地，p整除t'），进一步

$$x^{k\varphi(n)+1}=spq+x(s=t/q)$$

因此，$x^{k\varphi(n)+1}\equiv x(\mathrm{mod}\,n)$成立。

第二节　RSA 密码体制

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。许多国家标准化组织，如ISO、ITU和SWIFT等都已接受把RSA作为标准。Internet网的Email保密系统GPG以及VISA和MASTER组织的电子商务协议(SET协议)中都将RSA密码作为传送会话密钥和数字签名的标准。在数字电视应用中，更是把RSA算法应用在智能卡中，用来传送控制字。可见RSA算法本身的特性得到业内的共识。本节介绍RSA算法。

一、RSA 算法描述

RSA 算法主要包括密钥的产生、加密过程和解密过程[19]，具体如下：

1. 密钥的产生

(1)随机选择两个保密的大素数p和q，满足$|p|\approx|q|$；

(2)计算$n=pq$，$\varphi(n)=(p-1)(q-1)$，其中$\varphi(n)$是n的欧拉函数值；

(3)选一整数e，满足$1<e<\varphi(n)$，且$(\varphi(n),e)=1$；

(4)计算 d，满足 $de \equiv 1(\mathrm{mod}\,\varphi(n))$，即 d 是 e 在模 $\varphi(n)$ 下的乘法逆元，因 e 与 $\varphi(n)$ 互素，由模运算可知，它的乘法逆元一定存在，求解 d 使用（扩展）欧几里德算法；

(5)公钥为 (n,e)，私钥为 d。

2. 加密过程

加密时首先将明文比特串分组，使得每个分组对应的十进制数小于 n，即分组长度小于 $\log_2 n$，然后对每个明文分组 m，作加密运算：

$$c = E(m) \equiv m^e (\mathrm{mod}\, n)$$

得到的结果 c 是 RSA 密码体制加密得到的密文。

3. 解密过程

对密文分组的解密运算为：

$$m = D(c) \equiv c^d (\mathrm{mod}\, n)$$

二、算法的正确性

下面证明算法是正确的。

证明　因为 $de \equiv 1(\mathrm{mod}\,\varphi(n))$，设 $de = k\varphi(n)+1$，其中 k 是一个整数且 $k \geq 1$。由加密过程可知 $c \equiv m^e (\mathrm{mod}\, n)$，所以根据定理 2.5，有

$$\begin{aligned} D(c) &= c^d (\mathrm{mod}\, n) \\ &= (m^e)^d (\mathrm{mod}\, n) \\ &= m^{k\varphi(n)+1} (\mathrm{mod}\, n) \\ &= m \end{aligned}$$

第三节　RSA 的安全性

RSA的安全性是基于分解大整数的困难性假定，这是因为至今还未能证明分解大整数就是NP问题，这既不说明分解这个大整数是攻击RSA唯一的（或者说是最佳的）途径，也不能说明这种分解真的那么困难，也许还有尚未发现的多项式时间分解算法。不过目前这还只是展望，甚至连发展的方向都还没有找到。分解技术、计算机能力的提高和计算机造价的降低会威胁到RSA的安全性。其中分解技术对RSA的威胁最大。

如果RSA的模数 n 被成功地分解为 pq，则立即获得 $\varphi(n) = (p-1)(q-1)$，从而能够确定 e 模 $\varphi(n)$ 的乘法逆元 d，即 $d \equiv e^{-1} (\mathrm{mod}\,\varphi(n))$，因此攻击成功。

随着计算机运算速度的不断提高，原来被认为是不可能分解的大数已被成功分解。

例如[19]RSA-129（即n为129位十进制数，大约428比特）已在网络上通过分布式计算历时8个月于1994年4月被成功分解，RSA-130已于1996年4月被成功分解，RSA-140已于1999年2月被成功分解，RSA-155（512比特）已于1999年8月被成功分解，得到了两个78位（十进制）的素数。

对于大整数的威胁除了人类的计算能力外，还主要来自分解算法的进一步改进。分解算法过去都采用二次筛法，如对RSA-129的分解。而对RSA-130的分解则采用了一个新算法，称为一般数域筛法（GNFS），该算法在分解RSA-130时所做的计算仅比分解RSA-129多10%，对RSA-140和RSA-155的分解，采用的也是GNFS。将来也可能还有更好的分解算法，因此在使用RSA算法时对其密钥的选取要特别注意。估计在未来一段时间内，RSA的密钥长度介于1024比特至2048比特之间是安全的。

是否有不通过分解大整数的其他攻击途径？下面证明出n直接确定$\varphi(n)$等价于对n的分解[21]。

设$n=pq$中，$p>q$，由$\varphi(n)=(p-1)(q-1)$，则有

$$p+q=n-\varphi(n)+1,$$

以及 $p-q=\sqrt{(p+q)^2-4n}=\sqrt{(n-\varphi(n)+1)^2-4n}$

由此可得

$$p=1/2[(p+q)+(p-q)]$$

$$q=1/2[(p+q)-(p-q)]$$

所以，由 p，q确定$\varphi(n)$和由$\varphi(n)$确定p，q是等价的。

一、RSA参数的选取

为保证算法的安全性，必须要认真的选择RSA算法的参数[19-21]。

1. p和q的选择

⑴p和q要为强素数，且通常p和q的位数相等。

只有p和q足够大，$n=pq$才能足够大，才能抵抗因子分解的攻击。根据目前因子分解的能力，应当选择n为1024位或2048位。这样就要求p和q的选择应为512位或1024位左右。

⑵$|p-q|$要大，最好与p、q位数接近。

由$[(p+q)^2/4]-n=[(p+q)^2/4]-pq=(p-q)^2/4$，如果$|p-q|$小，则$(p-q)^2/4$也小，因此$(p+q)^2/4$ 稍大于n，$(p+q)/2$稍大于$\sqrt{n}$。可得n的如下分解法：顺序检查大于$\sqrt{n}$的每一个整数x，直到找到一个x使得x^2-n是某一整数（记为y）的平方。由

$x^2-n=y^2$，得$n=(x+y)(x-y)$。

(3) $p-1$和$q-1$都应有大素因子

这是因为RSA算法存在着可能的重复加密攻击法。设攻击者截获密文c，可如下进行重复加密：

$$c^e \equiv (m^e)^e \equiv m^{e^2} (\mathrm{mod}\, n)$$
$$c^{e^2} \equiv (m^e)^{e^2} \equiv m^{e^3} (\mathrm{mod}\, n)$$
$$\cdots\cdots\cdots\cdots\cdots\cdots$$
$$c^{e^{t-1}} \equiv (m^e)^{e^{t-1}} \equiv m^{e^t} (\mathrm{mod}\, n)$$
$$c^{e^t} \equiv (m^e)^{e^t} \equiv m^{e^{t+1}} (\mathrm{mod}\, n)$$

若$m^{e^{t+1}} \equiv c(\mathrm{mod}\, n)$，即$(m^{e^t})^e \equiv c(\mathrm{mod}\, n)$，则有$m^{e^t} \equiv m(\mathrm{mod}\, n)$，即$c^{e^{t-1}} \equiv m(\mathrm{mod}\, n)$，所以在上述重复加密的倒数第2步就已恢复出明文$m$，这种攻击法只有在$t$较小时才是可行的。为抵抗这种攻击，$p$与$q$的选取应保证使$t$很大。为使$t$大，就要$p-1$和$q-1$都应有大的素因子。

2. 加密密钥e的选择

对于公开密钥(n,e)，如果$m<n^{1/e}$，则加密$c=m^e(\mathrm{mod}\, n)$将不会用到模简约运算，所以通过在整数范围内求e次方根就可以很快地求得m。这就是为什么应该避免$e=3$这种情况。当$e=3$时，如果对同一条消息m用三个不同的模数来加密，即$c_i=m^3(\mathrm{mod}\, n_i), i=1,2,3$，则由于这三个模数是两两互素的，可以运用中国剩余定理算法来计算$c=m^3(\mathrm{mod}\, n_1n_2n_3)$。由于$m<(n_1n_2n_3)^{1/3}$，加密指数运算实际上与在整数上运算一样。于是对$c$的解密就是在求3次整数根，这可以有效地完成。

Coppersmith进一步把这种平凡的情况扩充到非平凡的情况：对于$m'=m+t$，其中m是知道的，t是不知道的，但是$t<n^{1/e}$，已知$c=(m')^e(\mathrm{mod}\, n)$，可以有效地求得$t$。因为在实际应用中，通常知道明文的部分信息，现在普遍认为RSA加密应该避免用很小的加密指数。广泛认可的$e=2^{16}+1=655537$，这也是个素数。这个指数提高了加密效率，同时也避免了小指数攻击。

3. 解密密钥d的选择

如果解密指数d很小，RSA是CPA（选择明文攻击）不安全的。当$d<n^{1/4}$时，Wiener基于对e/n的连分式展开发现了一种求d的方法[22]。这个结果已经改进到$d<n^{0.292}$[23]。

4. 对模数n的选择

在实现RSA时，为方便起见，可能给每一用户相同的模数n，而加解密密钥不同，这样做是不行的，它会使系统的安全性降低，其原因如下：

一方面，当选择公用的RSA模数n，然后把(e_i,d_i)分发给众多用户。则任何一对(e_i,d_i)都能分解模数n。所以从理论上来说，用数学方法任何用户都可以求出共享该模数n的其他用户的解密密钥d，这样他们就可以解密被发送给其他用户的信息，造成消息泄露。

另一方面，设两个用户拥有相同的RSA模数n，他们的公开钥分别为e_1和e_2，且e_1和e_2互素，明文消息是m，密文分别是$c_1 = m^{e_1}(\mathrm{mod}\, n)$，$c_2 = m^{e_2}(\mathrm{mod}\, n)$。当攻击者截获$c_1$和$c_2$后，可如下处理恢复出明文$m$。即利用推广的Euclid算法求出满足$re_1 + se_2 = 1$的两个整数$r$和$s$，其中一个为负，不妨设为$r$。再次利用推广的Euclid算法求出$c_1^{-1}$，从而得到$(c_1^{-1})^{-r}c_2^s \equiv m(\mathrm{mod}\, n)$。

通过上面的分析可知，当多个用户共用模数n时，攻击者利用数学推导可以比较容易地破译密文，所以是不安全的。

在实际应用中，不同用户选用的素数也不能相同。不然的话，假设两个用户选用了同一个素数p，设$n_1 = pq_1$与$n_2 = pq_2$分别是他们的模数，那么任何人都可以用Euclid算法求得$(n_1,n_2) = p$，从而得到n_1与n_2的分解式。

二、对 RSA 算法的攻击

对RSA算法的攻击可能有如下三种方式[24]：

1. 穷举攻击：这种方法试图穷举所有可能的私钥。

像其他的密码体制一样，RSA抗穷举攻击的方法也是使用大密钥空间。所以e和d的位数越大越好，但是密钥产生过程和加解密过程都包含复杂的计算，因此密钥越大，系统运行速度越慢。

2. 数学攻击：有多种数学攻击方法，它们的实质都是试图分解两个素数的乘积。

用数学方法攻击RSA的途径有以下三种[20]：

⑴分解n为两个素因子。这样就可以计算出$\varphi(n) = (p-1)(q-1)$，从而可以确定$d = e^{-1}(\mathrm{mod}\,\varphi(n))$。

⑵直接确定$\varphi(n)$而不先确定p和q。这样也可以确定$d = e^{-1}(\mathrm{mod}\,\varphi(n))$。

⑶直接确定d，而不先确定$\varphi(n)$。

对RSA的密码分析大都集中于第一种攻击方法，即将n分解为两个素因子。由给定的n来确定$\varphi(n)$等价于因子分解n。现在已知的，从e和n确定d的算法至少和因子分解问题一样费时。因此，我们通过将因子分解的性能作为基准来评价RSA的安全性。

尽管因子分解具有大素数因子的数n仍然是一个难题，但已不像以前那么困难。计

算能力的不断增强和因子分解算法的不断改进，给大密钥的使用造成威胁。在20世纪90年代中期以前一直是用二次筛法来进行因子分解，对RSA-130的攻击使用了称为一般数域筛的新算法，该算法能够因子分解比RSA-129更大的数，但计算代价仅是二次筛法的20%。我们期望GNFS还可以进一步改进并能设计出更好的算法。事实上，对某种特殊形式的数，用特殊数域筛（SNFS）算法进行因子分解比用一般数域筛法要快得多。我们期望算法上会有所突破，使一般的因子分解的性能在时间上大约与SNFS一样或者甚至比它更快[25]。因此我们在选择RSA的密钥大小时应谨慎小心。在最近一段时间里，密钥大小取在1024位到2048位是合适的。

3. 计时攻击：这类方法依赖于解密算法的运行时间。

密码学家Paul Kocher已证明，攻击者可以通过记录计算机解密消息所用的时间来确定私钥[26]。计时攻击不仅可以用于攻击RSA，而且可以用于攻击其他的公钥密码系统，由于这种攻击的完全不可预知性以及它仅依赖于明文，所以计时攻击具有很大的威胁。

计时攻击类似于窃贼通过观察他人转动保险柜拨号盘的时间长短来猜测密码，对RSA中的模幂算法来说，模幂运算是通过一位一位来实现的，每次迭代执行一次模乘运算，且若该位为1，则还需执行一次模乘运算[20]。假定在模幂算法中，计时攻击是从最左位b_k开始，一位一位进行的。攻击者已知前面的j位（为了得到整个指数，攻击者可以从$j=0$开始，重复攻击直到已知整个指数为止）则对给定的密文，攻击者可以完成for循环的前j次迭代，其后的操作依赖于未知的指数位。若该位为1，则要执行$d \leftarrow (d \times a) (\mathrm{mod}\, n)$。对有些$a$和$d$的值，模乘运算的执行速度异常慢，假定攻击者知道是哪些值。由于位为1时，对它执行迭代的速度很慢，若攻击者观察到解密算法的执行总是很慢，则可认为该位为1；若攻击者多次观察到整个算法的执行都很快，则可认为该位为0。

尽管计时攻击会造成严重的威胁，但是有一些简单可行的解决方法，包括[20]：

不变的幂运算时间：保证所有的幂运算在返回结果前等待的时间都相同。这种方法虽然很简单，但会降低算法的性能。

随机延时：通过在求幂算法中加入随机延时来迷惑计时攻击者可提高性能。Kocher认为，如果不设足够的干扰，那么攻击者可以通过收集额外的观察数据来抵消随机延时，仍然可能攻击成功。

隐蔽：在执行幂运算之前先将密文乘上一个随机数，这一过程可使攻击者不知道计算机正在处理的是密文的哪些位，这样可以防止攻击者一位一位地进行分析，而这种分析正是计时攻击的本质所在。

第三章　基于 RSA 的概率公钥密码算法

由于确定型公钥密码系统存在缺陷，对于相同的明文进行加密所得的密文总是相同的，那么密码学者就要考虑改变这种公钥密码系统。在加密过程中通过引入随机数使得对于相同的明文经过加密后得到的密文却不相同，即设计概率密码体制。本章在前两节简要介绍概率密码的数学基础和两种著名的概率公钥密码体制。第三节设计一种基于RSA的概率公钥密码算法，该算法不仅具有多项式安全性，而且加密后不产生数据膨胀，但是它的加密速度较慢，对长消息的加密效果不好。第四节对第三节设计的算法进行改进，在保证多项式安全性和密文膨胀率为1的前提下，提高加解密速度。

第一节　概率密码的数学基础

设 $Z_n = \{x \mid 1 \le x < n\}$，$Z_n^* = \{x \mid x \in Z_n, (x,n) = 1\}$。

1. 不可逼近的陷门谓词

“不可逼近的陷门谓词”，简称为UTP(Unapproximable Trapdoor Predicates)，是由S.Goldwasser和S.Micali于1984年提出的[2]。它的定义如下：

如果一个函数的值域为 {0,1}，则称其为一个谓词。

定义 3.1（ε-逼近）　设 $C[\bullet]$、$B[\bullet]$ 是两个定义在集合 U 上的谓词，如果集合 $A = \{x \mid x \in U, C[x] = B[x]\}$，且 $|A|/|U| \ge 1/2 + \varepsilon$，则我们称 $C[\bullet]$ 是 ε-逼近 $B[\bullet]$ 的。其中，$|A|$ 表示集合 A 中元素的个数，$\varepsilon > 0$。

令 N 表示自然数集，N' 是自然数集的一个子集，对 $\forall k \in N'$，记 S_k 为长度为 k 的二进制数集合的子集。对 $\forall i \in S_k$，记 R_i 表示长度小于等于 k 的二进制数集合的子集。并令

$$B_k = \{B_i : R_i \to \{0,1\} \mid i \in S_k\}$$

是用 k 标记的谓词集。且记 $B = \bigcup_{k \in N'} B_k$。

定义 3.2（不可逼近的陷门谓词）　如果上面定义的谓词集 B 满足下列条件，则 B 称为不可逼近的陷门谓词。

(1)（B 是不可逼近的）设 P_1、P_2 是两个给定的多项式，$k \in N'$，$C[\bullet,\bullet]$ 是定义在 $R_i \times S_k$ 上的环道函数，并令 $c_k = \min\{|C[\bullet,\bullet]|\}$：其中对 S_k 中至少 $1/P_2(k)$ 的元 i，$C[\bullet,i]$ 是 $1/P_1(k)$

逼近B_1的}。如果对于充分大的k及任何以k为元的多项式Q有：$c_k > Q(k)$，则我们就称B是不可逼近的。

⑵（B是陷门的）对$v \in \{0,1\}$，令：

$$R_i^v = \{x \in R_i, B_i(x) = v\}$$

如果下列条件满足，我们称B是陷门的。

①存在以k为元的多项式时间概率图灵机T_1，T_1对输入(i,v)，等概率地输出$x \in R_i^v$，其中$i \in S_k$，$v \in \{0,1\}$。

②存在函数$\sigma: \bigcup_{k \in N'} S_k \to N$，并可找到多项式$Q$，使得：对$\forall x \in \bigcup_{k \in N'} S_k$，有$|\sigma(x)| \lhd Q(x)|$，且存在多项式时间图灵机$T_2$，满足：$\forall i \in S_k$，$\forall x \in R_i$，有$T_2[i,\sigma(i),x] = B_i(x)$。我们称$\sigma(i)$是$i$的秘密。

③（构造性条件）

首先，$\forall k \in N'$，可以$1/|S_k|$的概率在k为元的随机多项式时间内选取任意一对$(i \in S_k, \sigma(i))$。

其次，确保选取得到$(i,\sigma(i))$的任何人要计算$B_i(x)$也是困难的，其中$i \in S_k$是公开的。

2.二次剩余[20]

二次剩余在数论中扮演着重要的角色。例如，整数分解算法都用到了二次剩余。二次剩余也经常用在加密和一些有趣的密码协议中。

定义 3.3(二次剩余) 设整数$n > 1$，对于$a \in Z_n^*$，a叫做模n的二次剩余，如果存在$x \in Z_n$，满足$x^2 \equiv a(\mathrm{mod}\, n)$；否则，$a$就叫做模$n$的二次非剩余。用$QR_n$表示模$n$的二次剩余集合，用$QNR_n$表示模$n$的二次非剩余集合。

定理 3.1 假设p是素数，则

⑴$QR_p = \{x^2 (\mathrm{mod}\, p) \mid 0 < x \le (p-1)/2\}$；

⑵恰好存在$(p-1)/2$个模p的二次剩余和$(p-1)/2$个模p二次非剩余，即Z_p^*被分成大小相等的两个子集QR_p和QNR_p。

推论 3.1 假设p为素数，则对任意的$a \in QR_p$，恰好存在a模p的两个平方根。用x表示其中的一个，则另一个是$-x(= p-x)$。

给定一个模数，我们经常需要判断一个数是否为二次剩余。这就是所谓的二次剩余判定问题。

定理 3.2(欧拉准则) 设p为素数，则对任意的$x \in Z_p^*$，$x \in QR_p$当且仅当

$$x^{(p-1)/2} \equiv 1(\mathrm{mod}\, p)$$

定理 3.3 设n为合数，并且有式子$n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$形式的完全分解。则$x \in QR_p$当

且仅当$x(\mathrm{mod}\,p_i^{e_i})\in QR_{p_i^{e_i}}$，于是当且仅当对任意的素数$p_i$，$i=1,2,\cdots k$，有$x(\mathrm{mod}\,p_i)\in QR_{p_i}$成立。

所以，如果知道n的分解，给定$x\in Z_n^*$，x模n的二次剩余就可以通过确定每个素数$p\mid n$的二次剩余来确定。后者可以通过验证欧拉准则来完成。

然而，如果不知道n的分解，确定模n的二次剩余是很困难的。

运用欧拉准则检验以素数为模的二次剩余需要复杂的模指数运算。然而，二次剩余可以用更快的算法来检验。这样的算法是基于勒让德-雅可比符号的定义。

定义 3.4(Legendre 符号) 对于奇素数p，定义a模p的 Legendre 符号为：

$$\left(\frac{a}{p}\right)=\begin{cases}1, a\in QR_p\\ -1, a\in QNR_p\end{cases}$$

定义 3.5(Jacobi 符号) 对于奇合数$n=p_1^{e_1}p_2^{e_2}\cdots p_k^{e_k}$，定义$a$模$p$的 Jacobi 符号为：

$$\left(\frac{a}{n}\right)=\left(\frac{a}{p_1}\right)^{e_1}\left(\frac{a}{p_2}\right)^{e_2}\cdots\left(\frac{a}{p_k}\right)^{e_k}$$

定理 3.4 设$n=pq$，$p\neq q$且p与q互素，则$x\in Z_n^*$是模n的二次剩余的充要条件是x是模p的二次剩余且x是模q的二次剩余。

定理 3.5 设$n=pq$，其中p和q是不同的奇素数，则分解n计算上等价于求模n的平方根。

推论 3.2 设$n=pq$，其中p和q是不同的奇素数，则对任意的$x\in QR_n$，x的两个平方根小于$n/2$，另外两个平方根大于$n/2$。

3. Blum 数[20]

Blum 整数广泛地用于公钥密码学中。

定义 3.6(Blum 整数) 一个合数n称为 Blum 整数，条件是$n=pq$，其中p和q是两个不同的素数并满足$p\equiv q\equiv 3(\mathrm{mod}\,4)$。

Blum 整数有很多有趣的性质，下面介绍一些在公钥密码学中常用的性质。

定理 3.6 设n为 Blum 整数，则下面的性质成立：

性质 1 $\left(\frac{-1}{p}\right)=\left(\frac{-1}{q}\right)=-1$（所以$\left(\frac{-1}{n}\right)=1$）。

性质 2 对于$x\in Z_n^*$，如果$\left(\frac{x}{n}\right)=1$，则或者$x\in QR_n$，或者$-x=n-x\in QR_n$。

性质 3 任意$x\in QR_n$有 4 个根$u,-u,v,-v$，并且都满足

(1) $\left(\frac{u}{p}\right)=1,\left(\frac{u}{q}\right)=1$，即$u\in QR_n$；

(2) $\left(\frac{-u}{p}\right)=1,\left(\frac{-u}{q}\right)=-1$；

(3) $\left(\frac{v}{p}\right)=-1,\left(\frac{v}{q}\right)=1$；

(4) $\left(\frac{-v}{p}\right)=1,\left(\frac{-v}{q}\right)=-1$。

性质 4　函数 $f(x)=x^2(\mathrm{mod}\,n)$ 是集合 QR_n 的一个置换。

性质 5　对于任意的 $x\in QR_n$，x 的平方根中恰好有一个小于 $n/2$ 且 Jacobi 符号为 1。

性质 6　Z_n^* 被分为 4 个等价类：一个为乘群 QR_n，另外 3 个是陪集 $(-1)QR_n$，ξQR_n，$(-\xi)QR_n$；其中，ξ 是 1 的平方根且 Jacobi 符号为 -1。

定理 3.7　设 $p\equiv 3(\mathrm{mod}\,4)$ 是素数，且 $y\in Z_p^*$。令 $x\equiv y^{(p+1)/4}(\mathrm{mod}\,p)$，

(1)如果 y 有一个模 p 平方根，那么 y 模 p 的平方根是 $\pm x$；

(2)如果 y 没有模 p 平方根，那么 $-y$ 有一个模 p 平方根，并且 $-y$ 模 p 的平方根是 $\pm x$。

定理 3.8　设 n 是 Blum 数，对于任意 $x\in QR_n$，那么 x 的平方根中有且仅有一个是平方剩余。

第二节　GM 和 BG 概率密码

根据 Legendre 符号和 Jacobi 符号的定义和定理，我们可以看出二者之间有一定的关系。假设 $n=pq$，其中 p 和 q 是两个不同的素数，如果只有 $\left(\frac{a}{p}\right)=-1$ 或 $\left(\frac{a}{q}\right)=-1$ 时，那么 $\left(\frac{a}{n}\right)$ 的结果一定等于 -1；如果 $\left(\frac{a}{p}\right)=-1$ 且 $\left(\frac{a}{q}\right)=-1$，或者 $\left(\frac{a}{p}\right)=1$ 且 $\left(\frac{a}{q}\right)=1$ 时，$\left(\frac{a}{n}\right)$ 的结果一定等于 1。它们之间的这种关系可以看出，如果 $a\in Z_n^*$，且 $\left(\frac{a}{n}\right)=-1$，意味着只有 $\left(\frac{a}{p}\right)=-1$ 或 $\left(\frac{a}{q}\right)=-1$，则 a 是模 n 的非二次剩余。如果 $\left(\frac{a}{n}\right)=1$ 且不知道 n 的素因子分解，则“a 是否是模 n 的二次剩余”的判定问题是计算上的难解问题。

概率密码系统就是根据二次剩余的这个难解的问题进行设计的。在加密时，传送者不仅要使用接收者公开的加密密钥 k 对固定的明文 m 加密，还要选择随机数 r，使得密文 $c=E_k(m,r)$。这样一来，对于相同的明文 m 和相同的加密密钥 k，得到的密文 c 却是不同的，从而提高了加密的安全性。

1. GM 概率公钥密码系统[2]

1984年Goldwasser和Micali使用二次剩余方案实现概率密码。该密码体制是公钥密码的一种，它的加密和解密密钥是不同的，根据解密算法是无法得到加密过程的。

如果实体A要将明文信息 m 经过GM概率公钥密码加密以后再传送给实体B，那么实体B必须先产生加密和解密使用的密钥对。实体B选择两个互不相同的大素数 p 和 q，并计算 $n=pq$，然后选取随机数 t（t 是一个与 n 互素的数，且是模 n 的二次非剩余）。实体B将 (n,t) 通过PKI（公钥基础设施）公开，而保留 (p,q) 作为自己的私钥。

实体A在将数据传送给实体B以前，先从PKI获得实体B的公钥，然后对每一位明文，发送者都要选择一个随机数 r_i，并计算

$$c_i = t^{m_i} r_i^2 (\mathrm{mod}\, n)$$

从这个式子可以看出，如果明文 $m_i=0$，那么 c_i 就是一个模 n 的二次剩余；相反，如果明文 $m_i=1$，c_i 就是一个模 n 的二次非剩余。最后，实体A将 $(c_1,c_2,\cdots c_t)$ 传输给实体B。

当实体B接收到密文后，只要对每一个密文 c_i 判断是否为模 n 的二次剩余，就可以断定 m_i 是0还是1，从而恢复出原来的明文。

2. BG 概率公钥密码系统[3]

BG概率公钥密码系统是在GM概率公钥密码系统基础上发展起来的。

当实体A要向实体B传送明文 m 所对应的密文时，首先实体B生成加解密所需的密钥对。实体B先要选择两个不同的大素数 p 和 q，使得 $p\equiv q\equiv 3(\mathrm{mod}\,4)$，计算 $n=pq$ 并找到两个整数 a 和 b，使 $ap+bq=1$。然后实体B将 n 通过PKI(公钥基础设施）公开，而保留 (p,q,a,b) 作为自己的私钥。

实体A在加密时，将明文 m 划分为 $(m_1,m_2,\cdots m_t)$ 组，其中每组 m_i 的长度都为 h。选择随机数 x，使 $x\in Z_n^*$，令 $x_0=x^2(\mathrm{mod}\,n)$。然后对 $i=1,2,\cdots t$，分别计算

$$x_i = x_{i-1}^2 (\mathrm{mod}\, n)$$

$$c_i = m_i \oplus x_i \text{的最右} h \text{位}$$

最后计算出 $x_{t+1}=x_t^2(\mathrm{mod}\,n)$。发送者将密文 $(c_1,c_2,\cdots c_t,x_{t+1})$ 传输给接收者。

接收者在接收到密文后，先利用自己的私钥将 x_{t+1} 恢复成 x_0，具体的计算步骤为：

(1)计算 $a_1=\left(\frac{p+1}{4}\right)^{t+1}(\mathrm{mod}(p-1))$，$a_2=\left(\frac{q+1}{4}\right)^{t+1}(\mathrm{mod}(q-1))$；

(2)计算 $b_1=x_{t+1}^{a_1}(\mathrm{mod}\,p)$，$b_2=x_{t+1}^{a_2}(\mathrm{mod}\,q)$；

(3)使用自己的私钥 p、q、a 和 b，计算 $x_0=b_2ap+b_1bq(\mathrm{mod}\,n)$。

接收者在通过上述过程得到x_0的情况下，重复发送者同样的计算过程，即对$i=1,2,\cdots t$，分别计算

$$x_i = x_{i-1}^2 (\mathrm{mod}\, n)$$

$$m_i = c_i \oplus x_i \text{的最右} h \text{位}$$

从而恢复出明文m。

3. 概率密码体制的安全性分析及其评价

概率密码是公钥密码中的一种，它具有公钥密码体制的所有特征，除此以外它还具有与其他公钥密码不同的特点。

(1)安全性分析

GM概率密码和BG概率密码的加密和解密密钥都不相同，而且通过公开的密钥利用有限的资源在多项式时间内是无法得到私钥的。所以这两种加密方法在公开公钥的情况下，并不会影响系统的安全性。

公钥密码体制都是以数学中某一难解的问题为基础的，这两种概率密码体制也正是基于二次剩余中的难解问题。GM概率密码系统所基于的数学难题是在不知道合数$n(n=pq)$的两个素因子(其中n是公钥，p和q是私钥)的情况下，要判定一个整数是否是模n的平方剩余是很困难的。而BG概率密码系统是基于在不知道合数$n(n=pq)$的两个素因子(其中n是公钥，p和q是私钥)的情况下，要计算一个模n平方剩余的数的模n的平方根等价于大数分解的困难性。

多项式安全性的概念是由S.Goldwasser和S.Micali最早提出来的，多项式安全性是概率公钥密码体制所特有的。一个公钥密码体制是多项式安全的，是指在已知c是两给定的明文m_1和m_2之一的密文的条件下，用任何多项式时间的概率算法来判断c是由两个明文中的哪一个加密而得到的，与用抛硬币的方法来猜测相比较，其正确的概率“几乎”一样，即判断正确的概率为$1/2+\varepsilon$，ε为一极小值。由于语言的统计特性，明文一般都会以一定的概率分布，这导致通过密文就可以获得明文的部分信息(例如，明文的二进制表示的某些有效位)。保证被加密信息的所有部分信息的安全性也是非常重要的。这对普通公钥密码体制是难以做到的。多项式安全性保证了明文的部分信息安全性。对于拥有多项式资源的攻击者而言，要想计算明文的某些特征或二进制表示的某个或某些比特的值(即部分信息)，仅仅知道密文是不够的。多项式安全性在公钥密码体制的各种安全性中是最强的一种。S.Goldwasser和S.Micali在文献中给出了定理每个概率公钥密码体制都具有多项式安全性的严格证明[2]。无论是GM概率公钥密码系统，还是BG概率公钥密码系统，均具有多项式安全性。

(2)效率分析[27]

在加解密速度方面。与RSA密码体制相比，这两种概率密码体制在计算上所需的时间大大缩短。GM概率密码体制只是在做一个数的平方运算，而BG概率密码体制在加密时计算一个数的平方，在解密时也只是做开平方操作。这种运算的运算量比模幂运算要小得多，所以概率密码体制的效率要比RSA高。

从形式上看，令模数n的长度为k位，明文的长度为k位，那么RSA加密和解密能在时间$O(k^3)$内完成。GM概率密码的加密需要花$O(k^3)$的时间，而解密需要花$O(k^4)$的时间；而在BG概率密码中，加密需要时间为$O(k^3/\log_2 k)$，解密需要的时间为$O(k^3)+O(k^3/\log_2 k)$。

密文膨胀率问题方面。密文膨胀率是指密文与明文长度之比。GM概率公钥密码系统中，为了确保密文c_i在传输的过程中是安全的(即几乎不可能或者说是很难被攻击者破译)，所以我们都会规定一个安全参数。安全参数是用来反映一密码系统安全性的一个数值。我们知道，GM和BG概率密码的数学基础都是二次剩余问题。无论是判断一个数是否是模合数n的平方剩余，还是计算模合数n的平方根，这两个问题的困难性都与合数n的大小有关。当n越大则系统越安全。所以，采用n的位数k表示系统的安全参数。

假设GM和BC概率密码的安全参数为k。经过GM概率加密以后，每一位明文信息都会变成具有安全参数长度的一串密文，从而使得在通信线路中传送的密文长度实际上变成tk，其中t是明文的长度。传送的数据由t变成tk，可以看出数据长度变得太大，所以当初这个密码系统没有实用价值。

BG概率公钥密码在密文膨胀率方面已经有了一定的改善，使密文膨胀率降到$1+k/t$，从而在一定程度上缓解了密文膨胀率过大的问题，但并没有完全消除密文膨胀率。到目前为止，许多专家学者仍然在试图解决概率密码中的这个问题。

第三节　基于 RSA 的概率公钥密码算法设计

概率加密的基本思想是在加密过程中通过引入随机数使得对于相同的明文经过加密后得到的密文却不相同，从而保证使用密文不能推出有关明文或密钥的任何信息。我们知道，GM和BG概率公钥加密体制，可以抵抗选择明文攻击，但它们的密文膨胀率都太大。本节设计一种基于RSA的加密后不产生数据膨胀的概率公钥密码算法。在介绍该

算法之前，补充两个概念。

定义3.7（时间戳[24]） 当动作发生时用来标识的惟一的时间日期戳。

定义3.8（hash函数[28]） hash函数又称为杂凑函数、散列函数、消息摘要或数据摘要等，它把任意长度的消息变换为一个固定长度的散列值。当消息哪怕只有一个比特位产生变化时，数据摘要“显著”地发生变化，不论消息的长度如何，其目的是把大的消息压缩了，变短了，因此，我们把数据摘要称为“数据指纹”。当消息在插入、篡改、重排之后，其数据指纹也要发生变化，显然可以提供完整性服务。

一、算法描述

1. 算法的预处理阶段

(1)用户i随机选择两个保密的大素数p_i和q_i，满足$|p_i| \approx |q_i|$；

(2)计算$n_i = p_i q_i$，$\varphi(n_i) = (p_i - 1)(q_i - 1)$，其中$\varphi(n_i)$是$n_i$的欧拉函数值；

(3)选一整数e_i，满足$1 < e_i < \varphi(n_i)$，且$(\varphi(n_i), e_i) = 1$，产生公钥e_i和n_i；

(4)计算d_i，满足$d_i e_i \equiv 1 (\mathrm{mod}\, \varphi(n_i))$，即$d_i$是$e_i$在模$\varphi(n_i)$下的乘法逆元，产生私钥$d_i$；

(5)不妨假设用户 A 要将明文分组m传输给用户 B，则用户 A 随机选择一个整数$x_0 \in Z_{n_B}^*$，其中n_B是用户B产生的模数，计算$x_1 \equiv x_0^{e_B} (\mathrm{mod}\, n_B)$，公开$x_1$，对$x_0$保密。

2. 加密过程

(1)用户A将x_0联接此次通信的时间戳t，得到$s = x_0 \| t$；

(2)通过hash函数h产生随机数$h(s)$；

(3)计算$c_1 = m \oplus h(s)$；

(4)计算$c \equiv c_1^{e_B} (\mathrm{mod}\, n_B)$，将密文$c$传输给用户B。

3. 解密过程

当用户B接收到密文c后，进行如下解密操作：

(1)计算$c_1 \equiv c^{d_B} (\mathrm{mod}\, n_B)$；

(2)计算$x_0 \equiv x_1^{d_B} (\mathrm{mod}\, n_B)$；

(3)计算$s = x_0 \| t$，通过hash函数h产生随机数$h(s)$；

(4)计算$m = c_1 \oplus h(s)$。

二、算法的正确性

算法是否正确我们主要看当用户A把明文m按照加密算法得到密文c，并把密文c

传输给用户 B 后，用户 B 是否能根据密文 c 而得到明文 m。

证明　因为用户 B 拥有解密密钥 d_B，所以用户 B 可以得到：

$$c^{d_B}(\mathrm{mod}\, n_B) \equiv (c_1^{e_B})^{d_B}(\mathrm{mod}\, n_B) \equiv (c_1)^{k\varphi(n_B)+1}(\mathrm{mod}\, n_B) \equiv c_1 。$$

又由　　$x_1^{d_B}(\mathrm{mod}\, n_B) \equiv (x_0^{e_B})^{d_B}(\mathrm{mod}\, n_B) \equiv x_0^{k\varphi(n_B)+1}(\mathrm{mod}\, n_B) \equiv x_0$，

再由解密的第三步产生 s，而得到 $h(s)$，最后得到

$$c_1 \oplus h(s) = (m \oplus h(s)) \oplus h(s) = m，即得到明文 m 。$$

三、数字签名

在通信过程中，为了防止信息伪造、修改及纠纷，需要在加密的基础上同时进行数字签名，得到签名后的密文，任何一方均不能进行更改。但是 GM 和 BG 算法都不能进行数字签名，本节设计的基于 RSA 的概率公钥密码算法可以进行数字签名，具体实现如下[29]（记号与上面相同）：

用户 A 随机选 $x_0 \in \min\{Z_{n_A}^*, Z_{n_B}^*\}$，按照上面加密过程由明文 m 得到对应的密文 c，然后作签名密文

$$R \equiv c^{d_A}(\mathrm{mod}\, n_A)$$

这里使用用户 A 身份的私人密钥 d_A 进行数字签名（其中 n_A 是用户 A 的模数）。用户 B 获得签名密文 R 后，解密计算

$$R^{e_A}(\mathrm{mod}\, n_A) \equiv (c^{d_A})^{e_A}(\mathrm{mod}\, n_A) \equiv c^{k\varphi(n_A)+1}(\mathrm{mod}\, n_A) \equiv c$$

这样就化成了无签名时的密文 c，由解密过程便可解密获得 m。

四、算法的安全性分析及其评价

1. 算法的安全性分析

⑴算法具有多项式安全性

与传统的 RSA 公钥密码体制一样，本节设计的基于 RSA 的公钥密码体制仍然是基于 RSA 单向陷门函数的，其安全性仍然是大整数因子分解的困难性[9]。但是，与传统 RSA 公钥密码体制不同的是，本节构造的公钥密码体制在加密时引入了随机数 x_0 与时间戳联接产生的 hash 函数值与明文的一系列运算，使得相同的明文经过加密后所对应的密文是不同的，这样就大大降低了攻击者进行明文攻击的可能性，使新体制具有概率加密的特点，从而使它不仅具有公钥密码体制的特征，而且还具有概率密码体制的特征，即具有多项式安全性。这样对任一密文，要在多项式时间内，求得明文是不可能的。

(2)可以抵抗选择明文攻击[9]

由于随机加密参数x_0是保密的，只有通信双方能够得到它，而其他用户都不可能在多项式时间内得到x_0。再有，由于加密算法中引入了以时间戳为种子的 hash 函数值与明文的一系列运算，使得同一明文所对应的密文具有很好的随机性，所以攻击者无法进行选择明文攻击。

2. 算法的效率分析

(1)加解密效率分析

我们知道 RSA 公钥密码加密、解密的时间复杂度为$O(k^3)$，其中k为体制规模的安全参数。本节所设计的基于 RSA 的概率公钥密码体制的加解密过程只不过是在原 RSA 公钥密码体制的基础上增加了一个 hash 函数值的计算，由于计算 hash 函数值的速度比 RSA 的加密、解密的速度快得多，因此，这个概率密码体制的加密、解密的时间复杂度也为$O(k^3)$。

(2)密文膨胀率分析

密文膨胀率是指密文与明文长度之比。密文膨胀率一直都是困扰概率密码体制发展的一个障碍。设本节所设计的密码体制规模的安全参数为k，则对于长度为k的明文分组m，得到的密文长度也为k，所以密文膨胀率=密文长度/明文长度=1，说明这种算法设计方案完全消除了密文的膨胀。

3. 算法的缺陷

本节所设计的基于 RSA 的概率公钥密码体制解决了选择明文攻击的问题，不仅具有多项式安全性，而且没有产生密文膨胀的问题，这是它的优点。但是它也存在着某些不足之处。首先最主要的是加、解密速度问题，这主要是受模幂运算的制约，随着模幂运算算法的不断改进，那么它的速度也会不断提高。其次是该算法对处理短消息时效果较好，因而可用于密钥的管理和数字签名等方面，但是对长消息的处理效果不好，因而有待进一步的改进。

第四节　改进算法设计

针对第三节算法的不足本节设计相应的改进算法，并对改进算法的性能及安全性方面进行具体分析。具体如下：

一、算法描述

1. 算法的预处理阶段

(1)用户 i 随机选择两个保密的大素数 p_i 和 q_i，满足 $|p_i| \approx |q_i|$；

(2)计算 $n_i = p_i q_i$，$\varphi(n_i) = (p_i - 1)(q_i - 1)$，其中 $\varphi(n_i)$ 是 n_i 的欧拉函数值；

(3)选一整数 e_i，满足 $1 < e_i < \varphi(n_i)$，且 $(\varphi(n_i), e_i) = 1$，产生公钥 e_i 和 n_i；

(4)计算 d_i，满足 $d_i e_i \equiv 1 (\mathrm{mod}\, \varphi(n_i))$，即 d_i 是 e_i 在模 $\varphi(n_i)$ 下的乘法逆元，产生私钥 d_i；（与第三节相同）

(5)不妨假设用户 A 要将明文 m 传输给用户 B，则用户 A 随机选择一个整数 $x_0 \in Z_{n_B}^*$，计算 $x_1 \equiv x_0^{e_B} (\mathrm{mod}\, n_B)$，$x_2 \equiv x_1^{e_B} (\mathrm{mod}\, n_B)$，公开 x_2，保密 x_0 和 x_1。

2. 加密过程

(1)用户 A 使用 x_1 联接此次通信的时间戳 t，得 $s = x_1 \| t$，通过 hash 函数产生随机数 $h(s)$；

(2)将明文 m 分成长度适当的等长（设长度为 L）数据块，$m = (m_1, m_2, \cdots m_k)$，每个数据块的长度 L 一般取为 8 的倍数。若 m_k 的长度小于 L，则对其用特殊字符进行填充。每个数据块的长度为 $h(s)$ 的长度；

(3)计算 $c_t = m_t \oplus h(s)$，其中 $t = 1,2,\cdots,k$；用户 A 将密文 $c = (c_1, c_2, \cdots c_k)$ 传输给用户 B。

3. 解密过程

(1)用户 B 计算 $x_1 \equiv x_2^{d_B} (\mathrm{mod}\, n_B)$；

(2)用 x_1 联接此次通信的时间戳 t，得 $s = x_1 \| t$，通过 hash 函数产生随机数 $h(s)$；

(3)计算 $m_t = c_t \oplus h(s)$，其中 $t = 1,2,\cdots,k$；得明文 $m = (m_1, m_2, \cdots m_k)$。

二、算法的正确性

算法是否正确我们主要看两个变换 $D(E(m)) = m$ 与 $E(D(c)) = c$ 是否成立。我们首先证明 $D(E(m)) = m$ 成立，也即当用户 A 把明文 m 按照加密算法得到密文 c，并把密文 c 传输给用户 B 后，用户 B 是否能根据密文 c 而得到明文 m。

证明　根据上面的加解密过程我们知道，加密变换可表示为：

$$c = E(m) = m \oplus h(s)$$

解密变换可表示为：　　$D(c) = c \oplus h(s)$

用户 B 拥有解密密钥 d_B，根据

$$x_2^{d_B} (\mathrm{mod}\ n_B) \equiv (x_1^{e_B})^{d_B} (\mathrm{mod}\ n_B) \equiv x_1^{k\varphi(n_B)+1} (\mathrm{mod}\ n_B) \equiv x_1,$$

再由解密的第二步产生 s, 用户 B 得到 $h(s)$，

所以　　$D(E(m)) = D(c) = D(m \oplus h(s)) = (m \oplus h(s)) \oplus h(s) = m$，

即得到明 m。从而证明出 $D(c) = m$，也即 $D(E(m)) = m$。

同理可证 $E(D(c)) = c$。

三、算法的安全性分析及其评价

1. 安全性分析

(1)多项式安全性

改进算法是基于RSA单向陷门函数的。因此, 其安全性仍然依赖于大整数的因数分解的困难性。 然而, 与传统体制不同的是, 该体制在加密时引入了随机数s（s的产生是通过随机数 x_1 与时间戳的联结后使用hash函数而获得的），使相同的明文加密后对应于许多不同的密文，这样就大大降低了破译者进行信息积累的可能性, 从而弥补了原有体制确定性的缺陷, 使其具有概率加密的特点，成为一种概率公开密钥密码体制。

下面证明该密码算法是多项式安全的。

证明　假设有明文 $M = M_1, M_2, \cdots M_k$ 和 $M' = M'_1, M'_2, \cdots M'_k$，其密文编码分别为 C 和 C'，其中：

$C = C_1, C_2, \cdots, C_k$，$C_i = M_i \oplus h(s)(i = 1,2,\cdots k)$，$h(s) = h(x_1 \| t)$，$x_2 \equiv x_1^{e_B} (\mathrm{mod} n_B)$。

$C' = C'_1, C'_2, \cdots C'_k$，$C'_i = M'_i \oplus h'(s)$，$h'(s) = h(x'_1 \| t)$，$x'_2 \equiv (x'_1)^{e_B} (\mathrm{mod} n_B)$。

用反证法：

假设体制不是多项式安全的。即存在一多项式时间算法能确定 M 或 M' 是属于 C、C' 中哪一个编码的明文。

不失一般性，假设 M、M' 只有一位不同，不妨设第 i 块中的第 t 位不同，即 $M_{it} \neq M'_{it}$ [30]。因为 $h(s)$ 和 $h'(s)$ 中不包含任何关于 M 或 M' 的信息，故仅利用 $h(s)$ 和 $h'(s)$ 的信息不可能达到确定明文编码的目的，必须用到 C 和 C' 中所含的信息。因为

$$C = C_1, C_2, \cdots C_k,\quad C_i = M_i \oplus h(s)$$

$$C' = C'_1, C'_2, \cdots C'_k,\quad C'_i = M'_i \oplus h'(s)$$

其中，$C_{it} = M_{it} \oplus h(s)|_t$，$C'_{it} = M'_{it} \oplus h'(s)|_t$，$C_{it}$ 和 C'_{it} 分别是密文 C 和 C' 第 i 块中的第 t 位。

由于异或运算是按位进行的，并且各位运算之间互不牵连，要区分 M_{it} 与 M'_{it} 编码的差别，只能借助于唯一和它们有关系的 C_{it}、$h(s)|_t$ 和 C'_{it}、$h'(s)|_t$，其中 C_{it} 和 C'_{it} 已知，

而$h(s)|_t$和$h'(s)|_t$未知（这主要是因为不论x_1和x_1'是已知还是未知，那么它与时间戳联接后再经过哈希函数的作用都会成为未知的）。这样$h(s)|_t$就有0或1两种可能情况，并且0或1出现的概率均为1/2。 因此，不论x是0还是1，都可能使公式$C_{it}=x\oplus h(s)|_t$成立，也即无论x是M_{it}还是M_{it}'，$C_{it}=x\oplus h(s)|_t$都可能成立，并且成立的概率都为1/2。这样就无法区分出C_{it}是两个明文M_{it}和M_{it}'哪一个的编码；同理，也不可能区分出C_{it}'是两明文中哪一个的编码。从而与假设矛盾。

综上所述，假设不成立。即不存在一多项式时间算法能确定M或M'是属于C 和C'中哪一个编码的明文。因此，体制是多项式安全的。

⑵可以抵抗选择明文攻击

改进算法中为了增强随机加密参数的安全性，在算法的预处理阶段采用了两次模幂运算产生x_1。由于随机加密参数x_1是保密的，只有通信双方能够得到它，任何其他用户都不可能在多项式时间内得到x_1。而且，由于加密算法中引入了以时间戳为种子的hash 函数值与明文的异或作用，使得同一明文所对应的密文是不同的，即密文具有很好的随机性，所以攻击者无法进行选择明文攻击。

2. 效率分析

⑴加解密效率分析

改进后的算法，由于加密前传递了保密的加密参数x_1，而在加密中只进行了一个hash 函数值的计算和它与明文的一个异或作用，因而加密的时间复杂度和 hash 函数的复杂度相同。而 hash 函数值的运算速度要比 RSA 的加密速度快得多，这样就使得算法的加密速度较改进前的算法大大提高了。解密过程中在得到随机加密参数x_1时使用了RSA 公钥密码算法的模幂运算，而其中的 hash 函数值的计算以及异或作用的运算相对模幂运算要快得多，因此，解密的时间复杂度与 RSA 的时间复杂度一样为$O(k^3)$。但是与改进前的算法相比，由于过程的简化，模幂运算量减少了一半，因而解密速度提高一倍。

⑵密文膨胀率分析

改进算法仍然是基于 RSA 的概率公钥密码算法，是以模数的长度为组长的分组公钥密码，而密文的长度也是模数的长度，在加密的准备阶段，依据 RSA 的安全性产生随机数x_1，在加密过程中，仅使用了异或作用，因而不产生密文的膨胀问题，即密文的膨胀率为 1。

3. 与文献[9]的比较

在文献[9]中，对大素数p和q的选择必须为 Blum 数，这是它解密能够完成的先决

条件，因而就限制了模数$n(n = pq)$的取法，从而使明文的加密使用范围也大大的缩小了。本文在此基础上提出了直接采用公钥的加密指数来选择x_1，从而使模数n的选取与传统的RSA公钥密码体制的模数n相同，没有增加任何限制，同样可以保证对明文m进行随机加密，具有多项式安全性，并且达到密文膨胀率为1的效果。

再有，加解密的速度大大提高了。通过加密过程我们可以看到，算法的设计，在保证多项式安全性的基础上使算法过程更加简化，只使用了hash函数以及hash函数值与明文的异或作用，加密中并不存在模幂运算，而hash函数值的运算速度要比RSA的加密速度快得多，因而速度大大提高了。解密过程中，也只是在求随机加密参数x_1时使用模幂运算，所以模幂运算量减少了一半，即解密速度提高一倍。

第四章　RSA概率数字签名方案

第一节　数字签名原理

在公钥密码学中，密钥是公开密钥和私有密钥组成的密钥对。数字签名就是用私有密钥进行加密，接收方用公开密钥进行解密，由于从公开密钥不能推算出私有密钥，所以公开密钥不会损害私有密钥的安全；公开密钥无须保密，可以公开传播，而私有密钥必须保密。因此，当某人用其私有密钥加密消息，能够用他的公开密钥正确解密，就可肯定该消息是某人签字的，这就是数字签名的基本原理。因为其他人的公开密钥不可能正确解密该加密过的消息，其他人也不可能拥有该人的私有密钥而制造出该加密过的消息。

一个数字签名方案由两部分组成[28]：签名算法（Signature Algorithm）和验证算法（Verification Algorithm）。一般说来，数字签名就是由信息的发送者通过一个单向函数对要传送的报文进行处理产生别人无法伪造的一段数字串。这个数字串用以认证报文的来源并核实报文是否发生了变化。发送者用自己的私有密钥加密数据传给接收者，接收者用发送者的公钥解开数据后，就可确定消息来源，同时也是对发送者发送信息的真实性的一个证明，发送者不能抵赖。

数字签名方案普遍都是基于某个公钥密码体制，签名者用自己的私钥对消息进行签名，验证人用相应的公钥对签名进行验证。从表面上看，数字签名与公钥加密是用密钥的顺序不同。实际上，数字签名与公钥加密一样也是用单向陷门函数确保其安全性。本质上，大数分解困难问题和离散对数困难问题等各种计算困难问题的存在是安全的数字签名方案存在的根本。

第二节　数字签名的功能

归纳起来，数字签名技术可以解决伪造、篡改、冒充、抵赖等问题，其功能表现在以下几个方面[28]。

机密性。数字签名中报文不要求加密，但在网络传输中，可以将报文信息用接收方的公钥进行加密，以保证信息的机密性。

完整性。数字签名与原始文件或其摘要一起发送给接收者，一旦信息被篡改，接收者可通过计算摘要和验证签名来判断该文件无效，从而保证了数据的完整性。

身份认证。在数字签名中，用户的公钥是其身份的标志，当使用私钥签名时，如果接收方或验证方用其公钥进行验证并获通过，那么可以肯定签名人就是拥有私钥的那个人，因为私钥是签名人唯一知道的秘密。身份认证包括通信实体认证和数据源认证。

防伪造。除签名人外，任何其他人不可能伪造消息的签名，因为签名密钥即私钥只有签名者自己知道，其他人不可能构造出正确的签名数据。

防抵赖。数字签名既可作为身份认证的依据，也可作为签名者签名操作的证据，防止抵赖。要防止接收者的抵赖，可以在数字签名系统中要求接收者返回一个自己签名的表示收到的报文，给发送者或受信任第三方。如果接收者不返回任何信息，此次通信可终止或重新开始，签名方也没有任何损失，由此双方均不可抵赖。

防重放攻击。如在电子商务中，公司 A 向公司 B 发送了一份商品订单，如果有攻击者中途截获订单并发送多份给公司 B，这样会导致 B 以为公司 A 订购了多批商品。在数字签名中，通常采用了对签名报文加盖时间戳或添加处理流水号等技术，可以防止这种重放攻击。

随着计算机网络的发展，过去信赖于手书签名的各种业务都可以用这种电子数字签名代替，它是实现电子贸易、电子支票、电子货币、电子购物、电子出版及知识产权保护等系统安全的重要保证。

第三节　RSA 数字签名方案

第一个数字签名方案是由Rivest，Shamir和Adleman这三位密码学家首先提出的[12]。RSA密码体制用到了初等数论中的一个重要定理—欧拉定理，其安全性依赖于大整数的因数分解的困难性。其用作数字签名的方案如下[24]：

签名人A，任意选取两个大素数 p 和 q，计算 $n=pq$，$\varphi(n)=(p-1)(q-1)$，随机选择整数 $1<e<\varphi(n)$，满足 $(\varphi(n),e)=1$；计算整数 d，满足 $de\equiv 1(\mathrm{mod}\,\varphi(n))$。则A的公钥为 (n,e)，私钥为 d。

签名：对于消息 $m(m<n)$，计算 $s=m^d(\mathrm{mod}\,n)$，则签名为 (m,s)，并将其发送给接

收人或验证人。

验证：接收人或验证人收到签名(m,s)后，利用A的公钥，计算$\overline{m}=s^e(\mathrm{mod}\,n)$，检查$\overline{m}=m$是否成立。如果成立，则签名正确，否则，签名不正确。

事实上，若签名正是A所签，则有

$$\overline{m}=s^e(\mathrm{mod}\,n)=(m^d)^e(\mathrm{mod}\,n)=m^{ed}(\mathrm{mod}\,n)=m\text{。}$$

在该签名方案中，任何人都可以用A的公钥进行解密，不具备加密功能，只起到鉴别签名人身份的目的。如果消息$m>n$，时，可用哈希函数进行压缩，计算$s=(h(m))^d(\mathrm{mod}\,n)$，接收方或验证方收到$(m,s)$后，先计算$\overline{m}=s^e(\mathrm{mod}\,n)$，然后检查$\overline{m}=h(m)$是否成立，即可鉴别签名是否正确。在这里，$m$只起到一种支撑的作用，没有实质性的意义。如果$m$包含重要的信息，不能泄露，那么签名还需要进行加密处理后再传送。基于此，本文设计了下面的RSA概率数字签名方案。

第四节　RSA概率数字签名方案设计

一、方案设计

首先假设系统的初始化过程与上节相同，A为签名人，A产生的公钥为(n,e)，私钥为d。则方案设计如下[31]：

签名：(1)设所签消息为m；

(2)产生随机填充的伪随机数r；

(3)联接m和r，并去掉其二进制的所有奇数位得到s_1；

(4)用hash函数计算摘要值$h=h(s_1)$；

(5)产生签名$s\equiv h^d(\mathrm{mod}\,n)$；

(6)输出：(s_1,s)。

验证：(1)收到：(s_1,s)；

(2)用 hash 函数计算摘要值$h=h(s_1)$；

(3)计算$h'\equiv s^e(\mathrm{mod}\,n)$；

(4)比较，若$h=h'$，则接收签名；

若$h\neq h'$，则拒绝签名。

很容易证明该数字签名方案是正确的。而且这种数字签名方案，有效的解决了当信

息 m 包含重要信息而泄露的问题。方案是采用通过信息 m 与随机填充的伪随机数 r 的联接作用，然后去掉其二进制的奇数位的方法，防止了信息的泄露。即通过引入随机数的方法，实现了在 RSA 数字签名的基础上进行概率数字签名的目的，从而使得它的签名方案的安全性大大提高，具有多项式安全性。

二、程序实现

根据上面的 RSA 概率数字签名方案设计原理，作者用 C 和 C++语言编写相应的程序，主要程序如下：

```
// bmrsa.cpp : Defines the entry point for the console application.
//
#include <math.h>
#include "md5.h"
#include <string.h>
#include <stdio.h>
#include <time.h>
#include "bignum.h"
#define MAXPRIMECOUNT 1000 // 寻找素数的最大次数
unsigned int nSmallPrimes[MAXPRIMECOUNT][2]; // 小素数数组
unsigned int nPrimeCount = 0; // 寻找素数的次数

// 寻找小素数
void MakeSmallPrimes()
{
    unsigned int n;
    unsigned int j;
    nPrimeCount = 3;
    // 首先定义几个小素数
    nSmallPrimes[0][0] = 2;
    nSmallPrimes[1][0] = 3;
    nSmallPrimes[2][0] = 5;
    nSmallPrimes[0][1] = 4;
```

```
    nSmallPrimes[1][1] = 9;
    nSmallPrimes[2][1] = 25;
    // 开始寻找素数
    for (n=7; nPrimeCount < MAXPRIMECOUNT; n+=2)
    {
        for (j=0; nSmallPrimes[j][1] < n; j++)
        {
            if (j>= nPrimeCount) // 如果大于当前素数组标号，则返回
            {
                return;
            }
            if (n % nSmallPrimes[j][0]==0)  // 如果不是素数，则退出循环
            {
                break;
            }
        }
        // 找到素数，设置素数
        if (nSmallPrimes[j][1] > n)
        {
            nSmallPrimes[nPrimeCount][0] = n;
            nSmallPrimes[nPrimeCount++][1] = n*n;
        }
    }
}

// 生成随机大数
CBigNum GenerateBigRandomNumber(unsigned short nBytes)
{
    CBigNum Result=0U; // 初始化大数
    int i;
    clock_t ctStart;
```

```
    unsigned long ctr=0;

    // 设置时间间隔
    clock_t ctInterval = CLOCKS_PER_SEC / 50 + 1;

    for (i=0; i<nBytes*2; i++)
    {
        ctStart = clock();
       // 等到大于时间间隔再开始
        while (clock() - ctStart < ctInterval)
            ctr++;

        ctr = (ctr % 33) & 0xF;

        Result <<= 4U; // 大数左移 4 位
        Result |= ctr; // 做或运算
    }
    putchar('\n');
    return Result; // 返回大数
}

CBigNum FindABigPrime(unsigned short nBytes)
{
    CBigNum nBig, nBig2;
    DWORD j;
    DWORD nTestCount = 0;
    DWORD nLehmanCount = 0;
    clock_t ctStartTime = clock(); // 记录开始时间
    DWORD nOffset=0;
    bool bPrime=false; // 素数标志位
```

```
        // 开始寻找对应位数的大素数
        for (nBig = GenerateBigRandomNumber(nBytes) | 1U; !bPrime; nBig+=2U,
nOffset+=2)
        {
            nTestCount++;
            for (j=0; j<nPrimeCount; j++)
            {
                // 如果不是大素数，则退出循环
                if (nBig % nSmallPrimes[j][0] == 0)
                {
                    break;
                }
            }

            if (j<nPrimeCount)
                continue;
            nLehmanCount++;
            nBig2 = (nBig - 1U) / 2U;
            // 设置一些随机的素数
            DWORD arnLehmanPrimes[] = { 89, 5179, 25981, 25439, 25013, 25667,
27397 };
            // 初始化大数组
            CBigNum LehmanResults[sizeof(arnLehmanPrimes) /
sizeof(arnLehmanPrimes[0])];
            nBig2 = nBig - 1U;
            bPrime = true;
            for (j=0; j<sizeof(arnLehmanPrimes) / sizeof(arnLehmanPrimes[0]); j++)
            {
                // 开始生成大素数
                LehmanResults[j] =
                    CBigNum(arnLehmanPrimes[j]).PowMod(nBig2,nBig,
```

```
CLOCKS_PER_SEC);
                if (LehmanResults[j] == nBig2)
                {
                }
                else if (LehmanResults[j] == 1U)
                {
                }
                else // 不是大素数
                {
                    bPrime = false;
                    break;
                }
            }
            // 找到大素数
            if (bPrime)
            {
                break;
            }
        }
        return nBig; // 返回大素数
    }

    // 生成公钥和私钥
    void GenKeyPair(CBigNum &PublicMod, CBigNum &PublicKey, CBigNum
&PrivateKey, CBigNum &P, CBigNum &Q, unsigned int nByteCount = 32)
    {
        if (0U==(P | Q))
        {
            P=FindABigPrime(nByteCount); // 生成 nByteCount 位的大素数
            Q=FindABigPrime(nByteCount); // 生成 nByteCount 位的大素数
            PublicKey=GenerateBigRandomNumber(nByteCount) | 1U;
```

```
        } else {
            PublicKey |= 1U;
        }
        PrivateKey = (P-1U) * (Q-1U); // 初始化私钥
        while (PublicKey > PrivateKey)
            PublicKey=GenerateBigRandomNumber(nByteCount-1) | 1U;
        while(CBigNum::gcd(PublicKey,PrivateKey) != 1U) // 生成公钥
            PublicKey+=2; // 累加直至互素
        PrivateKey = PublicKey.Inverse(PrivateKey); // 生成私钥

        PublicMod = P*Q; // 公钥 n 的生成
    }

    // 生成特定格式的公钥和私钥
    void GenerateKeys(CBigNumString &PublicMod, CBigNumString &PublicKey,
CBigNumString &PrivateKey, unsigned short nBytes)
    {
        CBigNum PubMod, PubKey, PriKey, PriP, PriQ;
        MakeSmallPrimes();
        GenKeyPair(PubMod, PubKey, PriKey, PriP, PriQ, nBytes); // 生成公钥和私钥
        // 将公钥和私钥转换成 16 进制形式
        PublicMod = PubMod.ToHexString();
        PublicKey = PubKey.ToHexString();
        PrivateKey = PriKey.ToHexString();
    }

    CBigNumString strMod, strPubKey, strPriKey; // 初始化大数字符变量
    // RSA 加密函数
    void RSAEncrypt(char *publickey,char *publicmod, char *output, unsigned int
*outputlen, char *input, unsigned int inputlen)
    {
```

```
    CBigNum Transform;
    CBigNum PubMod, PubKey;
    CBigNumString strTransform;
    // 将公钥转换成大数
    PubMod = CBigNum::FromHexString(publicmod);
    PubKey = CBigNum::FromHexString(publickey);

    // 转换输入的明文
    Transform = Transform.FromByteString(input);
    // 使用 RSA 加密明文
    Transform = Transform.PowMod(PubKey,PubMod);
    // 将密文转换成 16 进制的字符
    strTransform = Transform.ToHexString();

    // 输出密文长度
    *outputlen = strlen((const char*)strTransform)+1;
    // 输出密文
    memcpy(output,(const char*)strTransform,(*outputlen)+1);
}

// RSA 解密
//void RSADecrypt(char *output, unsigned int *outputlen, char *input, unsigned int
inputlen)
void RSADecrypt(char *privatekey,char *publicmod,char *output, unsigned int
*outputlen, char *input, unsigned int inputlen)
{
    CBigNum Transform;
    CBigNum PubMod,PriKey;
    CBigNumString strTransform;
    // 将私钥转换成大数
    PubMod = CBigNum::FromHexString(publicmod);
```

```
    PriKey = CBigNum::FromHexString(privatekey);
//  PubMod = CBigNum::FromHexString((const char*)strMod);
//  PriKey = CBigNum::FromHexString((const char*)strPriKey);
    // 转换输入的密文
    Transform = Transform.FromHexString(input);
    // 使用 RSA 对密文进行解密
    Transform = Transform.PowMod(PriKey,PubMod);
    // 将解密文转换成字节字符串
    strTransform = Transform.ToByteString();

    // 输出解密文长度
    *outputlen = strlen((const char*)strTransform)+1;
    // 输出解密文
    memcpy(output,(const char*)strTransform,(*outputlen)+1);

}

#define DIGEST_LEN 16 // 摘要长度
#define HASH_HEX_LEN 1+2*DIGEST_LEN // Hash 值长度

// 根据 MD5 算计算字符串的 Hash 值
void MD5_Hash(char *hash, char *input)
{
//  int i;
    int status = 0;
    md5_state_t state;
    md5_byte_t digest[DIGEST_LEN];

    int di;
    // 初始化算法
    md5_init(&state);
```

```
    // 添加校验字符串
    md5_append(&state, (const md5_byte_t *)input, strlen(input));
    // 完成 MD5 算法，输出摘要长度
    md5_finish(&state, digest);

    // 转换成 Hash 值
    for (di = 0; di < 16; ++di)
    {
        sprintf(hash + di * 2, "%02x", digest[di]);
    }
}

// 校验字符串 MD5 值
bool MD5_Check(char *hash, char *input)
{
    char input_str_hash[HASH_HEX_LEN];
    // 计算输入字符串的 Hash 值
    MD5_Hash(input_str_hash,input);

    // 比较输入的 Hash 值与计算出的 Hash 值是否相同
    if(strcmp(hash,input_str_hash))
    {
        return false; // 不同的 Hash 值
    }
    else
    {
        return true; // 相同的 Hash 值
    }
}

int main(int argc, char* argv[])
```

```
{

    char pubkey[300]="BC7A20007068FCB9541F8C31B26DBF03"; // 公钥
    char
pubmod[300]="9A4034363D76EF0C06E9A04BDBFF6A922BD92556960D8281EBACA6B
3550048B7"; // 公钥
    char
prikey[300]="47F6A6605373F4C71FDD101CCA8D8F29D61E9B2BF7BA87FE11ECAD81
D5C5B0B3"; //private key
    char encrypt_text[300]; // 输出的密文
    char decrypt_text[300]; // 输出的解密文
//  unsigned int pubkeylen;
    unsigned int encrypt_len;
    unsigned int decrypt_len;
    char plain_text[HASH_HEX_LEN];
     char file[30]="WelcometoMD5!";
    char fake_file[50]="HelloRSA!";
    char random[HASH_HEX_LEN];
    char k=0x55;
     unsigned int i,f_longth,ff_longth;
    int j;
    printf("请输入随机因子:");
    scanf("%s",random);
    printf("公钥 n :   %s\n",pubmod);
    printf("公钥 e :   %s\n",pubkey);
    printf("私钥 d :   %s\n",prikey);
     MD5_Hash(random,random);
    strcat(file,random);
    printf("原文+随机数为:\n");
    printf("%s\n",file);
    f_longth = strlen(file);
```

```
    //  int i;
        for(i = 0; i < f_longth; i++)
            file[i] = file[i]&k;
        printf("S1 为:\n");
        printf("%s\n",file);
        MD5_Hash(plain_text,file);
        printf("原 Hash 值 h 为:\n");
        printf("%s\n",plain_text);
        // 使用 RSA 进行加密
        RSAEncrypt(prikey,pubmod,encrypt_text,&encrypt_len,plain_text,HASH_HEX_LE
N);
        printf("签名 S 为 :\n");
        printf("%s\n",encrypt_text);
        // 使用 RSA 进行解密
        RSADecrypt(pubkey,pubmod,decrypt_text,&decrypt_len,encrypt_text,encrypt_len);
        printf("验证签名的 Hash 值 h'为:\n");
        printf("%s\n",decrypt_text);
        printf("请选择是否改动原文：1 为改动，0 为不改动\n");
        scanf("%d",&j);
        if(j==1)
        {
            strcat(fake_file, random);
            printf("改动后+随机数为:\n");
            printf("%s\n",fake_file);
            ff_longth = strlen(fake_file);
            for(i = 0; i < ff_longth; i++)
                fake_file[i] = fake_file[i]&k;
            printf("S1 为:\n");
                printf("%s\n",fake_file);
            MD5_Hash(plain_text,fake_file);
            printf("h 为:\n");
```

```
        printf("%s\n",plain_text);
    }
    else
    {
        printf("h 为:\n");
        printf("%s\n",plain_text);
    }

    if(strcmp(plain_text,decrypt_text))
    {
        printf("\n 拒绝签名\n");
    }
    else
    {
        printf("\n 接收签名\n");
    }
    printf("\n 请输入任意键退出！ ");
    scanf("%c\n",&k);
    return 0;


}
```

三、具体实例

根据上面编写的程序，形成可执行文件 RSA.exe。运行可执行文件 RSA.exe，作者用具体实例验证了本节设计的基于 RSA 的概率数字签名方案的可行性。

运行可执行文件 RSA.exe 文件则显示：“请输入随机因子”，只要在键盘上随机输入后按 Enter 键，则显示用户的公钥、私钥以及明文与随机数的联接结果，并计算出 hash 函数 h 的值与要验证签名的 hash 值 h'。当你选择“1”时为改动原文，则显示“拒绝签名”。具体如图 4-1 所示：

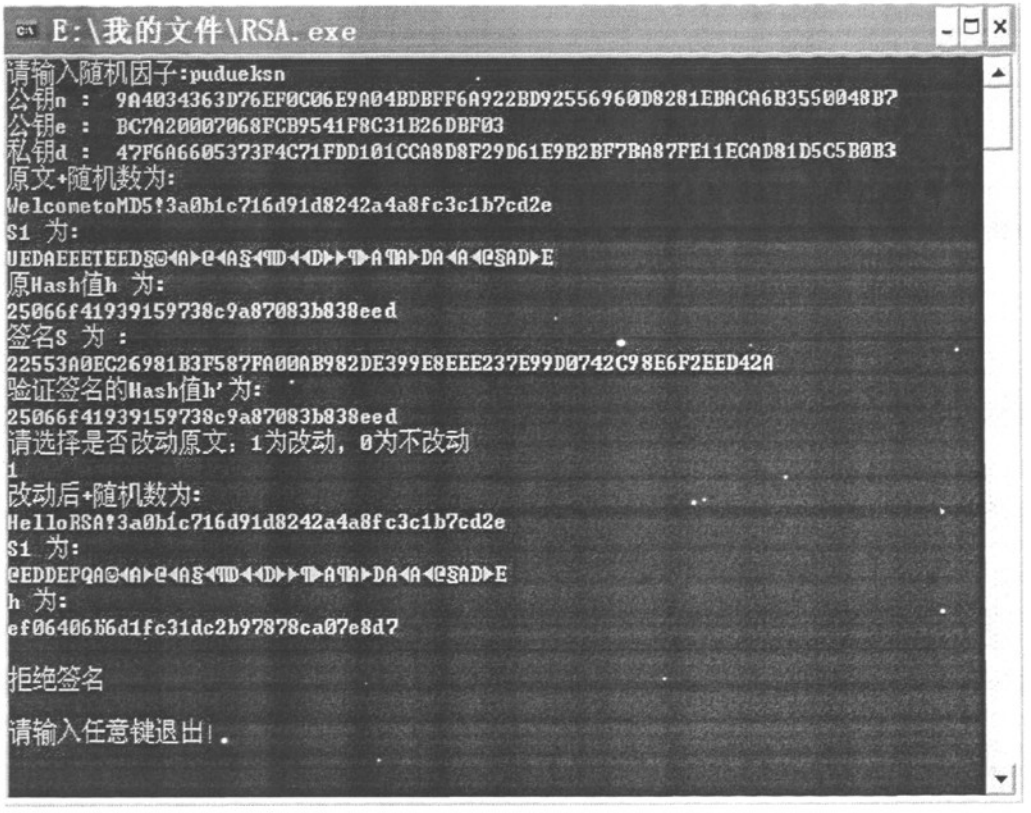

图 4-1　拒绝签名

图 4-1 说明当攻击者试图用更改原文的方法来欺骗信息的接收者时，则接收者会得到签名验证结果：$h \neq h'$，因而拒绝签名。

当运行可执行文件 RSA.exe，选择“0”时为不改动原文，则显示“接收签名”。具体如图 4-2 所示：

图 4-2 接收签名

第五章　结束语

本文设计了基于RSA的概率公钥密码算法，该算法对模数n的选择和RSA公钥密码算法一样，只要是两个大素数的乘积就可以，该算法不仅具有多项式安全性而且密文膨胀率为1，但是加密速度较慢，加密较长消息时效果不好，因而又对该算法进行了改进，在保证安全性的基础上提高了加密速度。本文又设计了基于RSA的概率数字签名方案，并对该数字签名方案进行了程序实现，运行该程序通过实例证明了方案的可行性。

参考文献

[1] 陈振宇, 王丽维. 信息安全在公钥密码体制中的实现技术. 现代计算机. 2000第95期: 60-61, 69.

[2] Goldwasser S, Micali S. Probabilistic encryption. Journal of Computer and System Sciences, 1984, 28 (2): 270-229.

[3] Blum M, Goldwasser S. An efficient probabilistic public-key encryption scheme which hides all partial information. Advances in Cryptology Procedings of CRYPTO'84 (LNCS196), 1985: 289-299.

[4] 何敬民. 概率加密方法的设计. 计算机研究与发展. 1989第9期:61-65.

[5] 李大兴, 张泽增. 一种基于RSA的公开钥密码体制及其安全性. 计算机学报. 1989.4: 279-288.

[6] 李献刚, 姚小波, 程时昕. 一种有效的概率加密体制. 密码学进展. 1996. 43-48.

[7] 刘花云, 罗静, 胡予濮. 基于RSA的概率加密. 西安电子科技大学报. 1997.12: 554-559.

[8] 赵泽茂. 基于RSA的随机加密算法与安全可靠度分析. 河海大学学报. 2002.7: 75-77.

[9] 王小非, 崔国华等. 一个数据膨胀率为 1 的概率公钥密码系统. 计算机科学. 2007. vol. 34. No.1:117-119.

[10] C.E. Shannon, Communication theory of secrecy systems, Bell system Technical Journal, 28(1949), 656-715.

[11] W. Diffie and M.E Hellman, New directions in cryptography, IEEE Transaions on Information Theory, 22(1976), 644-654.

[12] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 1978. 21(2): 120- 126.

[13] 施奈尔. 应用密码学第二版. 北京:机械工业出版社, 2000.

[14] Williams H. A modification of the RSA public-key encryption procedure (Corresp.). Information Theory, IEEE Transactions on Computers.1980, 3(9): 726-729.

[15] W. Tuchman. Hellman presents no shortcut solutions to the DES. IEEE Spectrum, 16(7): 40-41, 1979.

[16] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory, 1985.31 (4): 469-472.

[17] Niimura M. A high-speed processing LSI for RSA cryptograms using an improved adder circuit. IEEE Transactions on Computers. 2004.1(12): 175-178.

[18] 郑惠芳. 数据加密技术的研究. 福建高教研究. 2004 (3): 98一101.

[19] 杨波 编著. 肖国镇 审. 现代密码学第2版, 清华大学出版社. 2007.4.

[20] Wenbo Mao 著. 王继林, 伍前红等译. 现代密码学理论与实践电子工业出版社. 2006.2: 124-133,173-177.

[21] 卢开澄. 计算机密码学. 北京:清华大学出版社, 1999.

[22] M. Wiener. Cryptanalysis of short RSA secret exponents. IEEE Transactions on Information Theory, 36(3):553-558, 1990.

[23] D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $n^{0.292}$. In J. Stern, editor, Advances in Cryptology—Proceedings of EUROCRYPT'99, Lecture Notes in Computer Science 1592, pages 1-11. Springer-Verlag,1999.

[24] William Stallings 著. 刘玉珍, 王丽娜, 傅建明等译. 《密码编码学与网络安全—原理与实践（第三版）》. 电子工业出版社. 2006.1:200-420.

[25] Odlyzko, A. "The Future of Integer Factorization." CryptoBytes, Summer 1995.

[26] Kocher, P. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems." Proceedings, Crypto'96, August 1996.

[27] 余玛俐. 概率密码的研究. 华中科技大学硕士学位论文. 2005.11.

[28] 赵泽茂. 数字签名理论. 科学出版社. 2007.1: 2-6.

[29] 王尚平. RSA-概率公钥密码体制. 西安理工大学学报. 1995.11. (2): 145-148.

[30] 余梅生, 邹惠. 一种改进的RSA公钥密码体制. 大连理工大学学报. 2003.10.

[31] 贾杰, 徐赐文. 基于RSA的数字签名改进方案. 电脑知识与技术. 2008.11.第四卷第六期: 1345-1346.

攻读学位期间发表的学术论文目录

[1] 贾杰, 徐赐文. 基于RSA的数字签名改进方案. 电脑知识与技术. 2008.11.第四卷第六期: 1345-1346.

[2] 贾杰, 徐赐文, 李必涛等. 一种基于 RSA 的概率公钥密码体制及其安全性. 中央民族大学学报(自然科学版). 2009. 5(已接收).

致 谢

在论文撰写过程中，许多人给予我无私帮助，借此机会我向他们表示衷心的感谢！

首先，感谢我的导师徐赐文教授。徐教授渊博的学识、孜孜不倦的开拓精神、严谨求实的治学态度、精益求精的工作作风、高尚的人格和谦诚的为人之道将使我受益终生。从最初论文的选题、框架的搭建到最后的成稿，每一个环节都倾注了他大量的心血和精力。在此，谨向导师表示崇高的敬意和衷心的感谢！

本论文的顺利完成，离不开基础数学专业各位导师、同班同学以及朋友的关心和帮助。在此一并感谢！

还要特别感谢我的父母，在经济、生活上给我的支持和关心，使我得以顺利完成学业。

最后，真诚的感谢在百忙中抽出时间审阅我论文的专家教授们！

贾 杰

2009 年 3 月