第3章 密码学基础

中国科学技术大学

曾凡平

billzeng@ustc.edu.cn

主要内容

1. 密码学概述
2. DES和AES加密算法简介
3. RSA加密算法
4. 消息摘要和数字签名：实现完整性和抗抵赖的方法
5. 使用OpenSSL中的密码函数
6. Windows系统提供的密码算法
7. python语言的加密模块
8. PGP (Pretty Good Privacy)：
   - 日常工作中常用的加密工具，使用PGP产生密钥，加密文件和邮件。

第3讲 密码学基础

- 信息安全的主要目标是保护信息的**机密性、完整性和可用性**。机密性主要通过密码技术实现，而信息的完整性也直接或间接地使用了密码的相关技术，因此密码学是信息安全的基础。

- 长期以来，密码技术只在很小的范围内使用，如军事、外交、情报等部门。随着人类社会向信息社会的演进，**基于计算复杂性的计算机密码学**得到了前所未有的重视并迅速普及和发展起来。

- 目前，密码学已成为计算机网络安全领域的主要研究方向之一。

3.1 密码学概述

- 密码学是研究如何**隐密地传递信息**的学科，其**首要目的是隐藏信息的涵义**。密码学涉及信息的加密/解密及密码技术在信息传递过程中的应用。
- 早期的密码技术的安全性基于密码算法的保密，**现代的密码技术要求密码算法公开、密钥必须保密，密码算法的强度基于计算的复杂性**。
- 密码学包括**密码编码学**和**密码分析学**。
- 著名的密码学者Ron Rivest（RSA密码算法的发明者之一）对密码学的解释是：**“密码学是关于如何在敌人存在的环境中通讯”**。

基于密码学的保密通信系统的模型

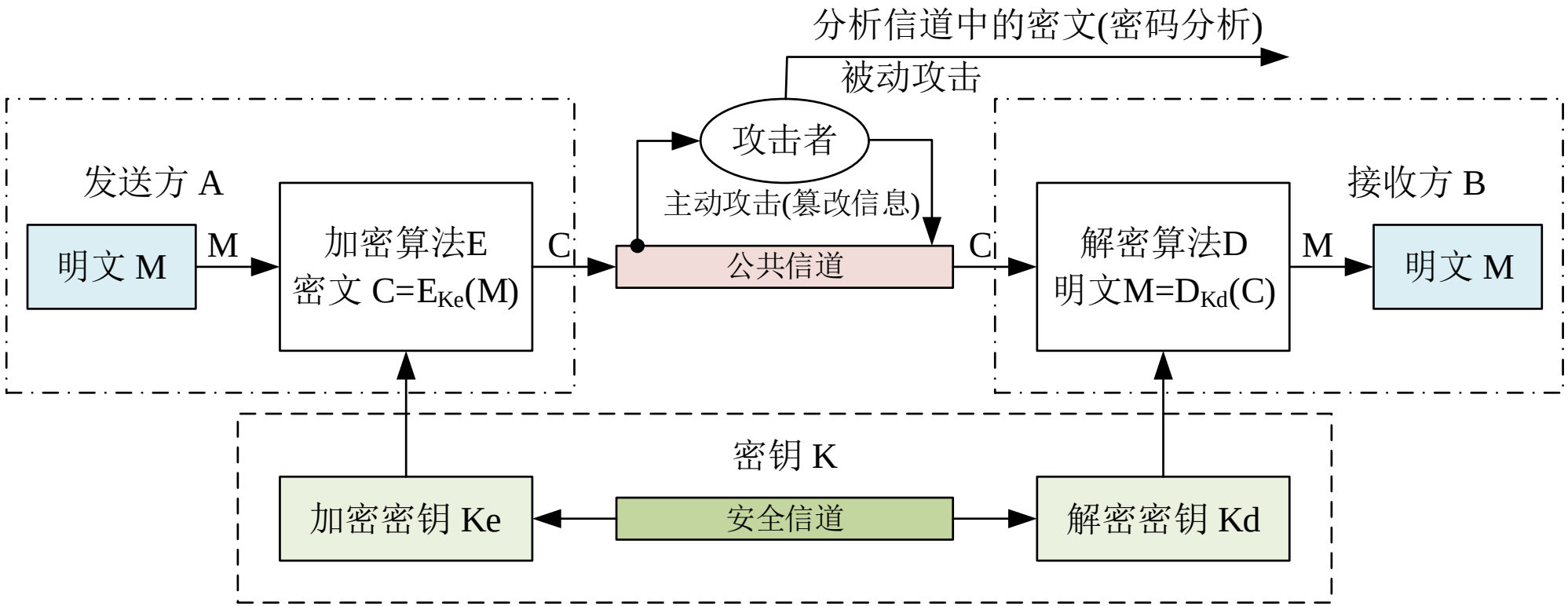


图3-1 保密通信系统的模型

目前的大部分**量子保密通信**技术：主要是通过量子技术实现**安全的密钥分发**

消息和加/解密

- 遵循国际命名标准，加密和解密可以翻译成：“Encipher（译成密码)”和“Decipher (解译密码)”。也可以这样命名：“Encrypt (加密)”和“Decrypt (解密)”。

→消息被称为明文。

→用某种方法伪装消息以隐藏它的内容的过程称为加密。

→加了密的消息称为密文。

→把密文转变为原始明文的过程称为解密。

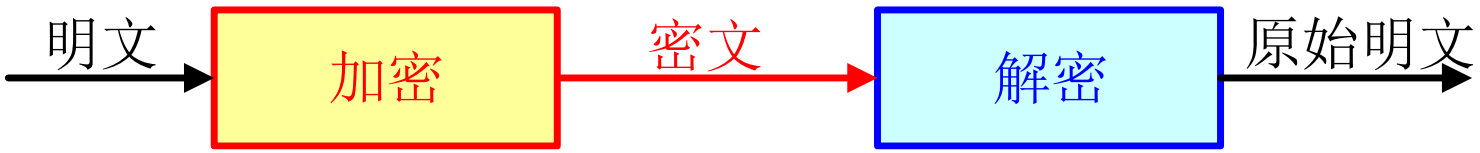


图3-2 加密和解密

明文、密文

- **明文**用**M** (Message，消息) 或**P** (Plaintext，明文)表示，它可能是比特流、文本文件、位图、数字化的语音流或者数字化的视频图像等。
- **密文**用**C** (Cipher) 表示，也是二进制数据，有时和M一样大，有时稍大。通过压缩和加密的结合，C有可能比M小些。

- 加密函数E作用于M得到密文C，用数学公式表示为：

$$E(M)=C$$

- 解密函数D作用于C产生M，用数学公式表示为：

$$D(C)=M$$

- **先加密、再解密，原始的明文将恢复出来**，下式必须成立：

$$D(E(M))=M$$

鉴别、完整性和抗抵赖性

- 除了提供机密性外，密码学需要提供三方面的功能：鉴别、完整性和抗抵赖性。
- **鉴别：**消息的接收者应该能够确认消息的来源；入侵者不可能伪装成他人。
- **完整性：**消息的接收者应该能够验证在传送过程中的消息没有被修改；入侵者不可能用假消息代替合法消息。
- **抗抵赖性：**发送消息者事后不可能虚假地否认他发送的消息。

算法和密钥

- **现代密码学要求密码算法公开，密钥保密。**
- **密钥用K表示，是加密函数和解密函数的输入参数。**
- 密钥K的可能值的范围叫做**密钥空间**。对称密码算法加密和解密运算都使用这个密钥，即运算都依赖于密钥，并用K作为下标表示，加/解密函数表达为：

$E_K(M)=C$

$D_K(C)=M$

$D_K(E_K(M))=M$

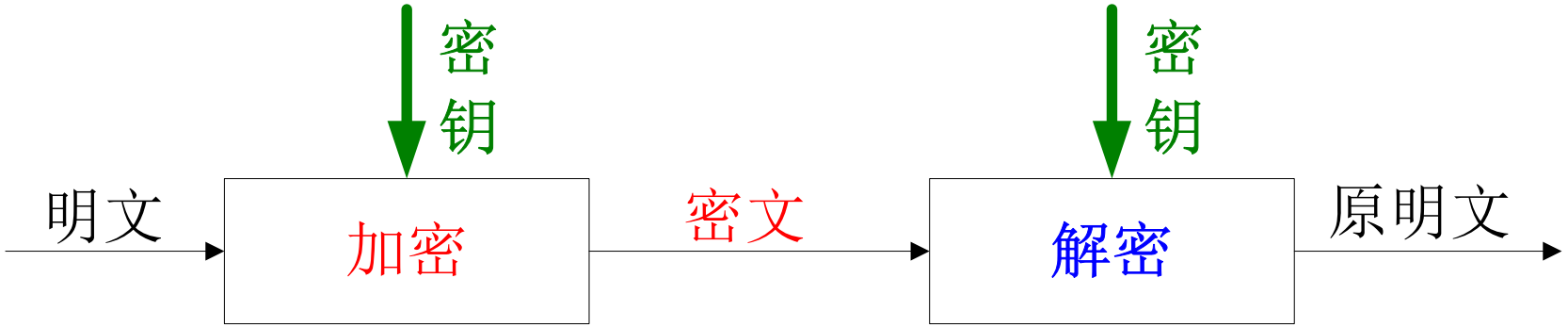


图3-3 对称密码体制的加密和解密

不同的加密密钥和解密密钥

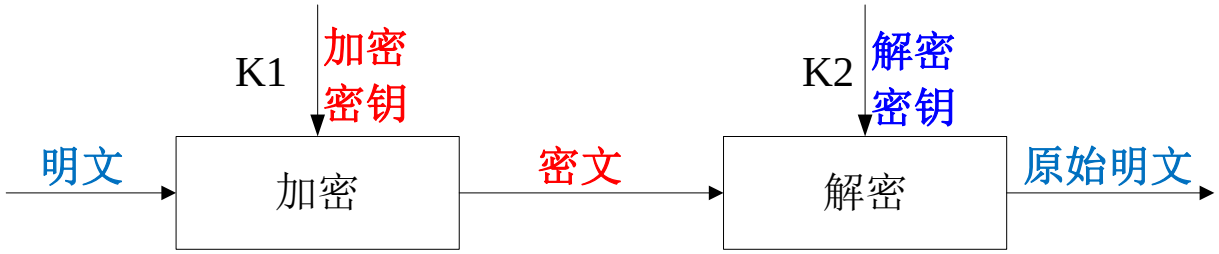


图3-4 公钥密码体制的加密和解密

- 有些算法使用不同的加密密钥和解密密钥，也就是说**加密密钥K1与相应的解密密钥K2不同**，在这种情况下，加密和解密的函数表达式为：

  $\mathbf{E_{K1}(M)=C}$

  $\mathbf{D_{K2}(C)=M}$

- 函数必须具有的特性是：$\mathbf{D_{K2}(\ E_{K1}(M)\ )=M}$

对称密码算法的基本原理

- 基于密钥的密码算法通常有两类：对称密码算法和公开密钥算法（非对称算法）。
- **对称密码算法**有时又叫**传统密码算法**或**单密钥密码算法**，**加密密钥能够从解密密钥中推算出来**，反过来也成立。
- 在大多数对称算法中，加/解密的密钥是相同的。对称算法要求发送者和接收者在安全通信之前，协商一个密钥。**对称算法的安全性依赖于密钥的保密**，泄漏密钥就意味着任何人都能对消息进行加/解密。
- 对称算法的加密和解密表示为：

$$E_K(M) = C$$
$$D_K(C) = M$$

对称密码算法的两个分支

- **分组密码算法**：对明文的一组位进行加密和解密运算，这些位组称为分组，相应的算法称为分组算法。常见的分组算法有DES、DES3、IDEA、AES等。分组密码算法也称为**块密码算法**。
  - DES分组长度为64位、密钥长度为64位;
  - AES加密数据块分组长度为128比特，密钥长度可以是128比特、192比特、256比特中的任意一个（如果数据块及密钥长度不足时，会补齐）。
- **序列密码（流密码）算法**：一次只对明文的单个位（有时对字节）运算的算法称为序列密码算法或**流密码**。
  - 常见的流密码有RC4、A5、SEAL、PIKE等。

公开密钥密码算法的基本原理

- **公开密钥算法(非对称算法)**的**加密密钥和解密密钥不同**，而且解密密钥不能根据加密密钥计算出来，或者至少在可以计算的时间内不能计算出来。

- 之所以叫做公开密钥算法，是因为加密密钥能够公开，即陌生者能用加密密钥加密信息，但只有用相应的解密密钥才能解密信息。**加密密钥叫做公开密钥（简称公钥），解密密钥叫做私人密钥（简称私钥）**。

- 用公开密钥K1加密表示为：$E_{K1}(M)=C$
- 用相应的私人密钥K2解密可表示为：$D_{K2}(C)=M$

安全协议

- **密码编码原则：密码算法应建立在算法的公开不影响明文和密钥的安全**。
- **密码协议：**也称作安全协议，是使用密码技术的协议，是以密码学为基础的消息交换协议，其目的是在网络环境中提供各种安全服务。
- 安全协议是网络安全的一个重要组成部分，我们需要通过安全协议进行实体之间的认证、在实体之间安全地分配密钥或其它各种秘密、确认发送和接收的消息的不可否认性等。
- 常见的安全协议有：认证协议、不可否认协议、公平性协议、身份识别协议、密钥管理协议。

3.2 对称密码技术

- 对称密码体制的加密密钥和解密密钥是相同的，其中最负盛名的是曾经广泛使用的DES(Data Encryption Standard，数据加密标准)和正在普遍使用的AES(Advanced Encryption Standard，高级加密标准)。
- **与公开密钥密码技术相比，其最大的优势就是速度快，一般用于对大量数据的加/解密**。
- 本节简要介绍DES和AES。

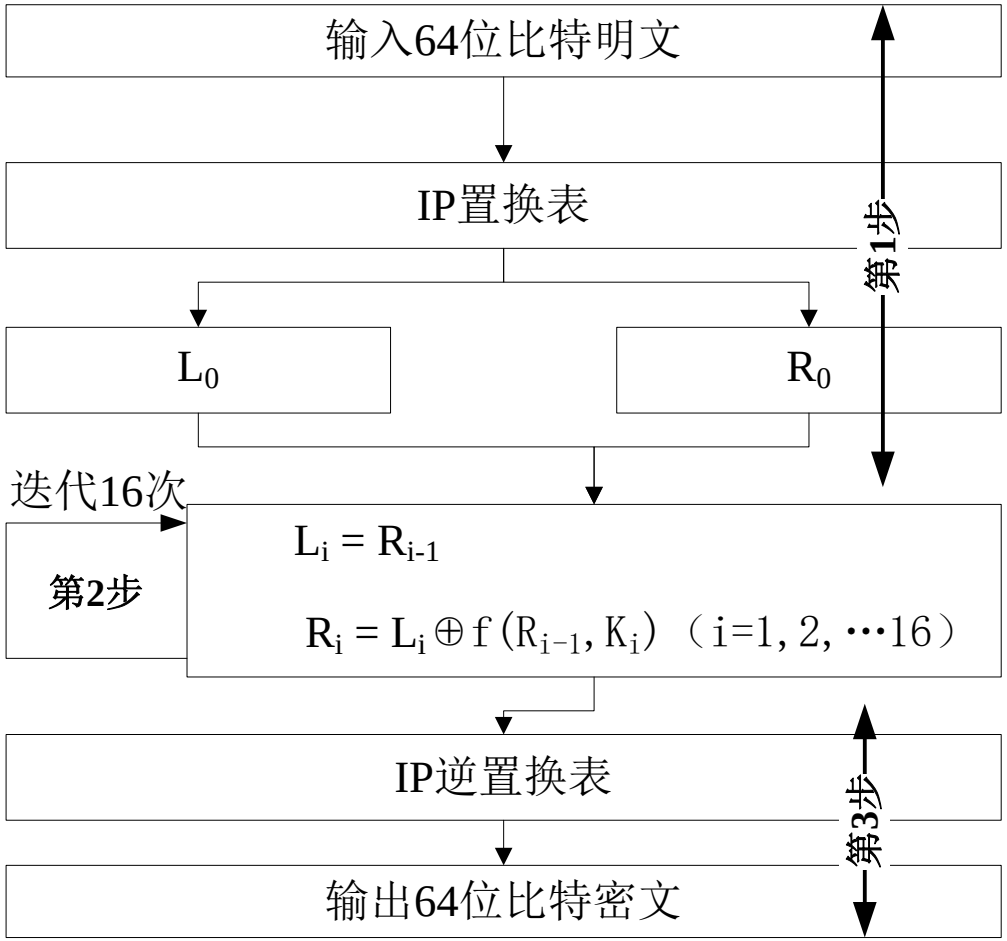
3.2.1 DES算法简介

- DES是一种使用密钥加密的块密码，1976年被美国联邦政府的国家标准局确定为联邦资料处理标准（FIPS），随后在国际上广泛流传开来。
- **DES使用56位密钥对64位明文块进行加密**。
- 这个算法因为包含一些机密设计元素、相对短的密钥长度以及怀疑内含美国国家安全局（NSA）的后门而在开始时有争议，DES因此受到了强烈的学院派式的审查，并以此推动了现代的块密码及其密码分析的发展。
- 尽管如今已被更安全的算法（如AES）取代，但其在密码学历史上具有重要意义。

DES加密流程

DES的加密过程包括以下步骤:

- (1) **初始置换（IP置换）**：将64位明文重新排列，增加加密的复杂性。
- (2) **16轮加密**：每轮包括数据分组、密钥生成、异或运算、S盒替换和P盒置换。
- (3) **逆置换（FP置换）**：对最终结果进行逆置换，生成密文。
- DES算法的详细内容请参考密码学方面的专著，其具体实现的源代码请参考OpenSSL源代码(http://www.openssl.org)。



DES算法的整体结构

DES算法的安全性

- DES的56位密钥过短，现在已经不是一种安全的加密方法。
- 早在**1999年1月**，distributed.net与电子前哨基金会合作，就**在22小时15分钟内破解了一个DES密钥**。
- 随着计算机的升级换代，运算速度大幅度提高，破解DES密钥所需的时间也将越来越短。
- 为了保证实用应用所需的安全性，可以使用DES的派生算法3DES来进行加密。3DES被认为是十分安全的，虽然它的速度较慢。
- 另一个计算代价较小的替代算法是DES-X，它通过将数据在DES加密前后分别与额外的密钥信息进行异或来增加密钥长度。
- GDES则是一种速度较快的DES变体，但它对微分密码分析较敏感。

DES的各种变种

- 由于DES的密钥长度仅为56比特，破解密文需要2^{56}次穷举搜索，在目前已难以保证密文的安全。
- 为了解决DES密钥长度过短的问题，可以采用组合密码技术，也就是将密码算法组合起来使用。三重DES(简写为DES3或3DES)是最常用的组合密码技术，破解密文需要2^{112}次穷举搜索，其算法如图3-6所示。

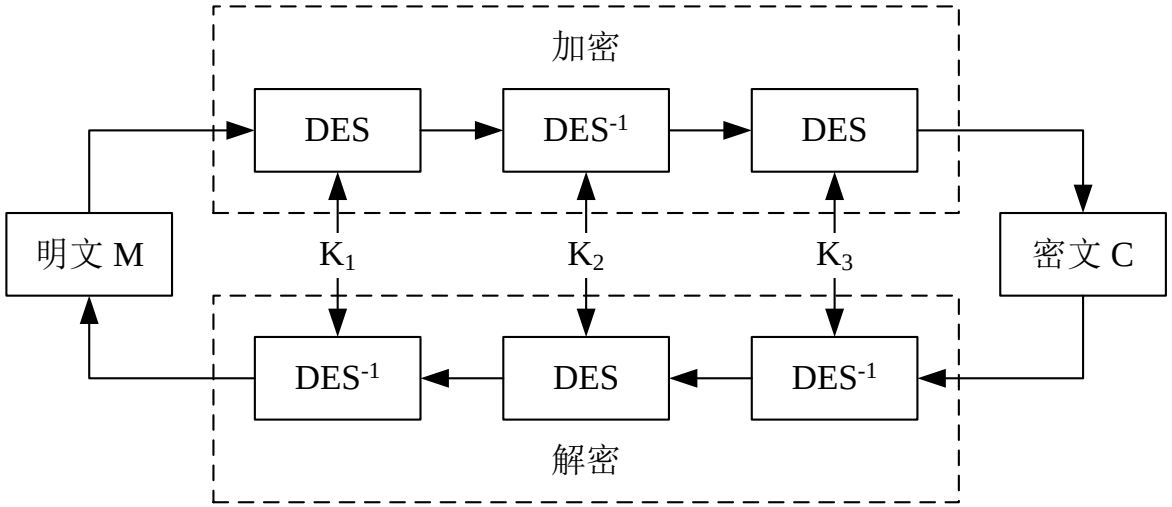


图3-6 三重DES

基于DES的其它算法还有DESX、CRYPT(3)、GDES、RDES、更换S盒的DES、使用相关密钥S盒的DES等。

3.2.2 AES算法简介

- 2000年10月，在历时接近5年的征集和选拔之后，NIST选择了一种新的密码，即高级加密标准（AES），用于替代DES。2001年2月28日，联邦公报发表了AES标准，从此开始了其标准化进程，并于2001年11月26日成为FIPS PUB 197标准。
- AES算法在提交的时候称为Rijndael。选拔中其它进入决赛的算法包括RC6、Serpent、MARS和Twofish。
- AES是一种分组加密算法，将明文分成固定长度的块进行加密。每个块的长度为128位（16字节），密钥长度可以是128位、192位或256位。
- AES的加密过程包括多个轮次，每轮次包含四个主要步骤：字节替换、行移位、列混合和轮密钥加。

AES加密过程

(1) **字节替换：**使用一个固定的S盒对状态矩阵中的每个字节进行替换。S盒提供了非线性变换，增强了加密的安全性。

(2) **行移位：**对状态矩阵的每一行进行循环移位操作，不同行的移位数不同。这个步骤增加了数据的扩散性。

(3) **列混合：**通过矩阵乘法将状态矩阵的每一列进行混合，使得每个字节都受到其他字节的影响。

(4) **轮密钥加：**将状态矩阵与轮密钥进行按位异或操作，增加了加密的复杂性。

AES解密过程

- AES的解密过程是加密过程的逆操作。解密同样包括字节替换、行移位、列混合和轮密钥加，但这些操作是逆向进行的。

- AES算法的详细内容请参考密码学方面的专著，其具体实现的源代码请参考OpenSSL源代码(http://www.openssl.org)。

3.2.3 对称密码技术的应用举例

- OpenSSL是使用非常广泛的SSL的开源实现，本节用实例演示在命令行下运行OpenSSL，用对称密码算法实现文件的加密和解密。

[实例1] 用des3算法实现对文件test.data加密和解密。

- 步骤1：生成24字节的随机密钥，把密钥保存在文件key.txt中。
  openssl rand -base64 24 | tr -dc 'a-zA-Z0-9' | cut -c 1-24 > key.txt
  生成一个明文文件test.data：**echo "愿向世界播撒阳光" > test.data**
- 步骤2：加密test.data为test.3des
  openssl enc -e -des3 -in test.data -out test.3des -kfile key.txt
  #openssl enc -e -des3 -pbkdf2 -in test.data -out test.3des -kfile key.txt
- 步骤3：解密test.3des为test.dddd
  openssl enc -d -des3 -in test.3des -out test.dddd -kfile key.txt
  #openssl enc -d -des3 -pbkdf2 -in test.3des -out test.dddd -kfile key.txt
- 步骤4：验证test.dddd和原始文件test.data相同
  md5sum test.dddd test.data

对称密码技术的应用举例

[实例2] 使用 AES-128，对二进制文件file.bin进行加密/解密。

- 准备二进制文件：cp ./cryptoDemo file.bin
- 步骤1：加密file.bin为file.aes128

  openssl enc -e -aes-128-cbc -in file.bin -out file.aes128 -pass pass:12345678

  # openssl enc -e -aes-128-cbc -in file.bin -out file.aes128 -pass file:key.txt
- 步骤2：解密file.aes128为file.bind：

  openssl enc -d -aes-128-cbc -in file.aes128 -out file.bind -pass pass:12345678

  # openssl enc -d -aes-128-cbc -in file.aes128 -out file.bind -pass file:key.txt
- 步骤3：验证file.bind和原始文件file.bin相同：

  md5sum file.bin file.bind

3.3 RSA公钥加密技术

- 公开密钥加密（public-key cryptography）也称为非对称(密钥)加密，该思想最早由Ralph C. Merkle在1974年提出。之后在1976年，Whitfield Diffie（迪菲）与Martin Hellman（赫尔曼）两位学者在现代密码学的奠基论文“New Direction in Cryptography”中首次公开提出了公钥密码体制的概念。
- 公钥密码体制中的密钥分为**加密密钥**与**解密密钥**，这**两个密钥是数学相关的**，用加密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密钥，并不能计算出另外一个密钥。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。

  公开的密钥称为**公钥（PK）**，用于加密。

  不公开的密钥称为**私钥（SK）**，用于解密。

常见的公钥加密算法

- 常见的公钥加密算法有RSA、ElGamal、背包算法、Rabin（RSA的特例）、Diffie－Hellman密钥交换协议中的公钥加密算法、椭圆曲线加密算法（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），ElGamal是另一种常用的非对称加密算法。

- RSA和ElGamal是同时很好地用于加密和数字签名的公开密钥算法。此类算法要求**加密、解密的顺序可以交换**，即满足以下公式：

$$\mathbf{E_{PK}(\ D_{SK}(M)\ )} = D_{SK}(\ E_{PK}(M)\ ) = M$$

RSA算法

- RSA算法于1977年由Rivest、Shamir和Adleman(当时他们三人都在麻省理工学院工作)发明，是第一个既能用于数据加密也能用于数字签名的算法。
- RSA算法易于理解和操作，虽然其安全性一直未能得到理论上的证明，但是它经历了各种攻击，至今未被完全攻破，所以，实际上是安全的。

- 1973年，在英国政府通讯总部工作的数学家克利福德·柯克斯（Clifford Cocks）在一个内部文件中提出了一个与RSA相似的算法，但他的发现被列入机密，一直到1997年才被发表。

RSA算法

- 对极大整数做因数分解的难度决定了RSA算法的可靠性。换言之，对一极大整数做因数分解愈困难，RSA算法就愈可靠。
- 如果有人找到了一种快速因数分解的算法，那么用RSA加密的信息的可靠性就肯定会极度下降，但找到这样的算法的可能性是非常小的，目前只有短的RSA密钥才可能被暴力方式解破。
- 到2013年为止，世界上还没有任何可靠的攻击RSA算法的方式。
- **只要其密钥的长度足够长，用RSA加密的信息实际上是不能被解破的。**

- 1983年麻省理工学院在美国为RSA算法申请了专利。这个专利2000年9月21日失效。由于该算法在申请专利前就已经被发表了，在世界上大多数其它地区这个专利权不被承认。

3.3.1 RSA算法描述

- **密钥计算**方法：

  选择两个大素数p和q (典型值为1024位)

  计算 $n=p\times q$ 和 $z=(p-1)\times(q-1)$

  选择一个与 z 互质的数，令其为 d

  找到一个 e 使满足 $e\times d=1 \pmod z$

  公开密钥为 (e, n)，私有密钥为 (d, n)

- **加密**方法：

  将明文看成比特串，将明文划分成k位的块P即可，这里k是满足$2^k<n$的最大整数。

  对每个数据块P，计算$C = P^e \pmod n$，C即为P的密文。

- **解密**方法：

  对每个密文块C，计算$P = C^d \pmod n$，P即为明文。

3.3.2 RSA算法举例

- **密钥计算**:

  取$p = 3$，$q = 11$

  则有$n = p \times q = 33$，$z = (p-1) \times (q-1) = (3-1) \times (11-1) = 20$

  7和20没有公因子，可取$d = 7$

  解方程$7 \times e = 1 \pmod{20}$，得到$e = 3$

  公钥为（3, 33），私钥为（7, 33）

- **加密**:

  若明文$P = 4$，则密文$C = P^e \pmod{n} = 4^3 \pmod{33} = 31$

- **解密**:

  计算$P = C^d \pmod{n} = 31^7 \pmod{33} = 4$，恢复出原文

3.3.3 RSA算法的安全性

- 假设偷听者乙获得了甲的公钥(e,n)以及丙的加密消息C，但她无法直接获得甲的私人密钥d。

- 要获得d，最简单的方法是将n分解为p和q，这样她可以得到同余方程$d \times e \equiv 1 \pmod{(p-1)(q-1)}$并解出d，然后代入解密公式：

$$P = C^d \pmod{n}$$

- 这样就破解了密文C，导出了明文P。

RSA算法的安全性

- **至今为止还没有人找到一个多项式时间的算法来分解一个大的整数的因子，同时也还没有人能够证明这种算法不存在。**至今为止也没有人能够证明对n进行因数分解是唯一的从C导出P的方法，但今天还没有找到比它更简单的方法（至少没有公开的方法）。因此今天一般认为只要n足够大，那么攻击者就没有办法了。

- 目前，假如n的长度小于或等于256位，那么用一台个人电脑在几个小时内就可以分解它的因子。1999年，数百台电脑合作分解了一个512位长的n。2009年12月12日，编号为 RSA-768 （768 bits, 232 digits）数也被成功分解。这一事件威胁了1024 bit密钥的安全性，普遍认为用户应尽快升级到2048 bit或以上。

3.3.4 RSA算法的速度

- 比起DES和其它对称算法来说，RSA要慢得多。速度慢一直是RSA的缺陷，一般来说只用于少量数据加密。
- 事实上RSA一般用于数字签名和对工作密钥的加密，对数据的加密一般采用速度更快的对称密码算法。

- RSA是被研究得最广泛的公钥算法，从提出到现在已经过了几十年，经历了各种攻击的考验，逐渐被人们接受，普遍认为是目前最优秀的公钥方案之一。

3.3.5 RSA算法的程序实现

- 根据RSA算法的原理，可以利用C语言实现其加密和解密算法。RSA算法比DES算法复杂，加/解密所需要的时间也比较长。

- 具体实现见OpenSSL的源代码 (http://www.openssl.org)

演示

3.3.6 公开密钥密码技术的应用举例

- 用RSA公钥密码技术时，首先用openssl genrsa生成一对密钥（公钥和私钥），接着用openssl rsa导出公钥，然后用openssl pkeyutl对文件进行加密和解密。
- [实例1] 随机生成2048位的RSA密钥，将其存入文件rsa_key.pem中，将公钥导出到文件rsa_pub.pem，用rsa算法对文件test.data加密和解密，并验证其正确性。

  (1) 生成RSA密钥文件rsa_key.pem：

  openssl genrsa -out rsa_key.pem 2048

  - 这里-out指定生成文件的文件名。需要注意的是这个文件包含了公钥和密钥两部分，也就是说这个文件即可用来加密也可以用来解密。后面的2048是生成密钥的长度。
  - 查看密钥信息：

  openssl rsa -in rsa_key.pem -text -noout

(2) 将密钥文件rsa_key.pem中的公钥导出到文件rsa_pub.pem：

openssl rsa -in rsa_key.pem -pubout -out rsa_pub.pem

-in指定输入文件，-out指定提取生成公钥的文件名。至此，我们手上就有了一个公钥，一个私钥（包含公钥）。现在可以用公钥来加密文件了。

(3) 用公钥rsa_pub.pem加密文件test.data：

openssl pkeyutl -encrypt -in test.data -inkey rsa_pub.pem -pubin -out test.en

-in指定要加密的文件，-inkey指定密钥，-pubin表明是用纯公钥文件加密，-out为加密后的文件。

(4) 解密文件：

openssl pkeyutl -decrypt -in test.en -inkey rsa_key.pem -out test.de

-in指定被加密的文件，-inkey指定私钥文件，-out为解密后的文件。

(5) 验证test.de和原始文件test.data相同：

md5sum test.de test.data

3.4 消息摘要和数字签名

3.4.1 消息摘要

3.4.2 数字签名

3.4.3 应用举例

- 消息摘要用于鉴别信息的完整性。
- 数字签名用于实现信息的不可否认性。

3.4.1 报文摘要(消息摘要)

- 消息摘要的目的是将消息鉴别与数据保密分开，其**基本设想**是：**发送者用明文发送消息，并在消息后面附上一个标签，允许接收者利用这个标签来鉴别消息的真伪**。
- 用于鉴别消息的标签必须满足以下两个条件：
  - ✓ 第一，能够验证消息的完整性，即能辨别消息是否被修改；
  - **方案：**将一个散列函数作用到一个任意长的消息m上，生成一个固定长度的**散列值H(m)**，这个散列值称为该消息的**数字指纹**，也称**消息摘要(message digest，MD)**。
  - ✓ 第二，标签不可能被伪造。
  - **方案：**发送方用密码技术对消息摘要M1进行加密保护，得到加密后的消息摘要C，接收方对C进行解密恢复M1，再与计算接收消息的消息摘要M2比较。**加密后的消息摘要也称为消息鉴别标签。**

用于消息鉴别的散列函数H的特性

- 用于消息鉴别的散列函数H必须满足以下特性：

  (1)H能够作用于任意长度的数据块，并生成固定长度的输出。

  (2)对于任意给定的数据块x，H(x)很容易计算。

  (3)对于任意给定的值h，要找到一个x满足H(x)=h，在计算上是不可能的(单向性)。(这一点对使用加密散列函数的消息鉴别很重要)

  (4)对于任意给定的数据块x，要找到一个y≠x并满足H(y) = H(x)，在计算上是不可能的。(这一点对使用加密算法计算消息鉴别标签的方法很重要)

  (5)要找到一对（x, y）满足H(y) = H(x)，在计算上是不可能的。(抵抗生日攻击)

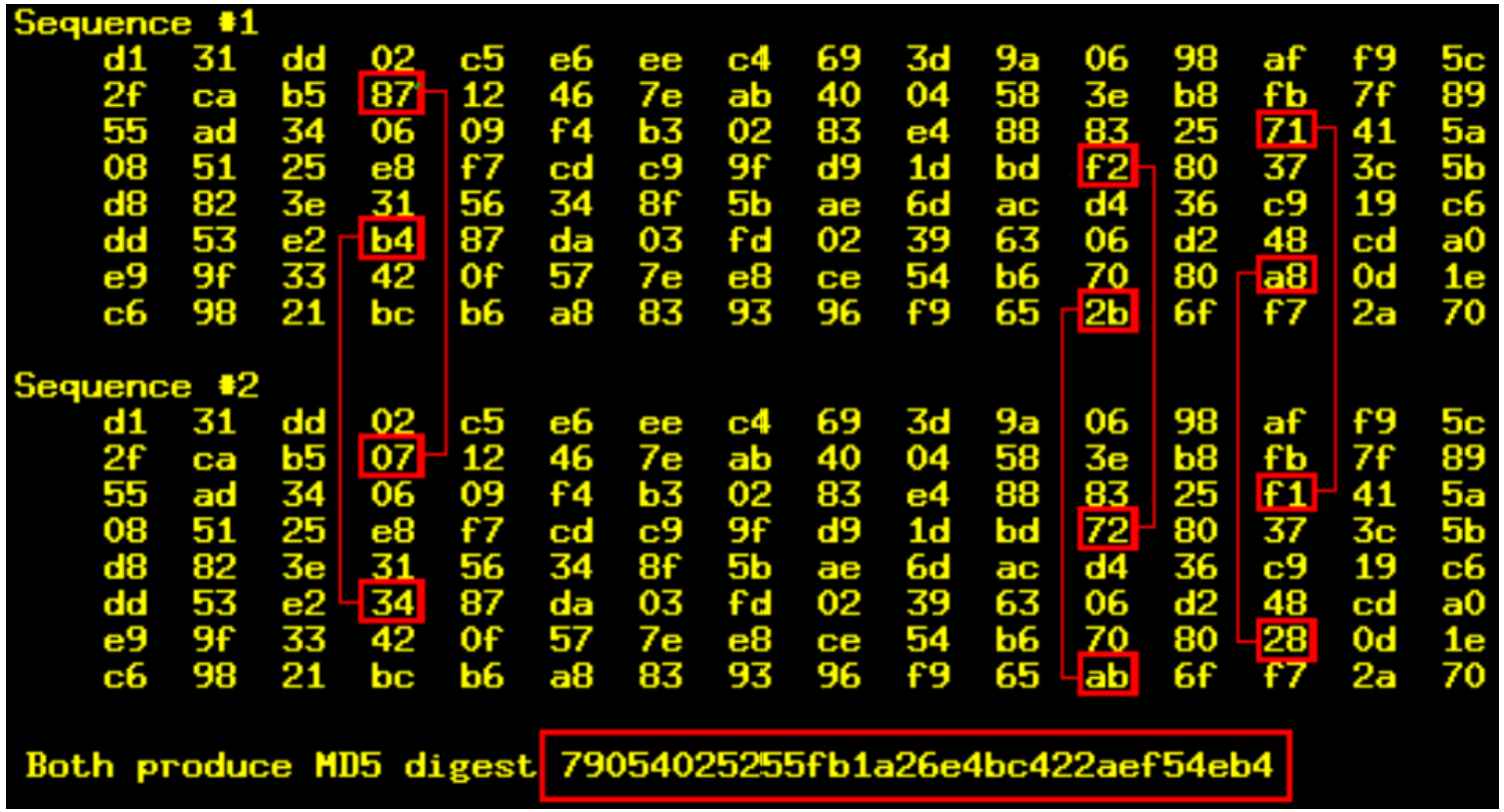
- 如果不满足(3)(4)(5)中的之一，则称**存在碰撞问题**，是不安全的。

最常用的两种散列函数：MD5和SHA

- 目前使用最多的两种散列函数是MD5和SHA序列函数。
- MD5的散列码长度为128比特。
- SHA序列函数是美国联邦政府的标准，如SHA-1散列码长度为160比特，SHA-2散列码长度为256、384和512位。

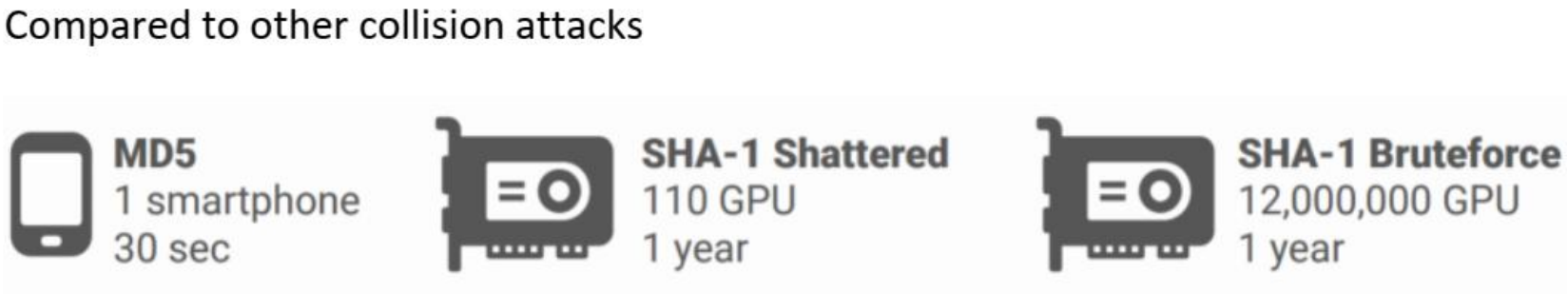
MD5的碰撞问题

- MD5的散列码长度为128比特，已经被证明是不安全的。
- **2004年，山东大学的王小云（现为科学院院士，清华大学和山东大学的教授）在国际上第一次发现MD5算法存在碰撞的可能**，并给出了实例。



SHA-1的碰撞问题

- On February 23, 2017, CWI Amsterdam and Google announced they had performed a collision attack against SHA-1. They had given 2 different PDF files with the same SHA-1 outputs.
- https://shattered.io/



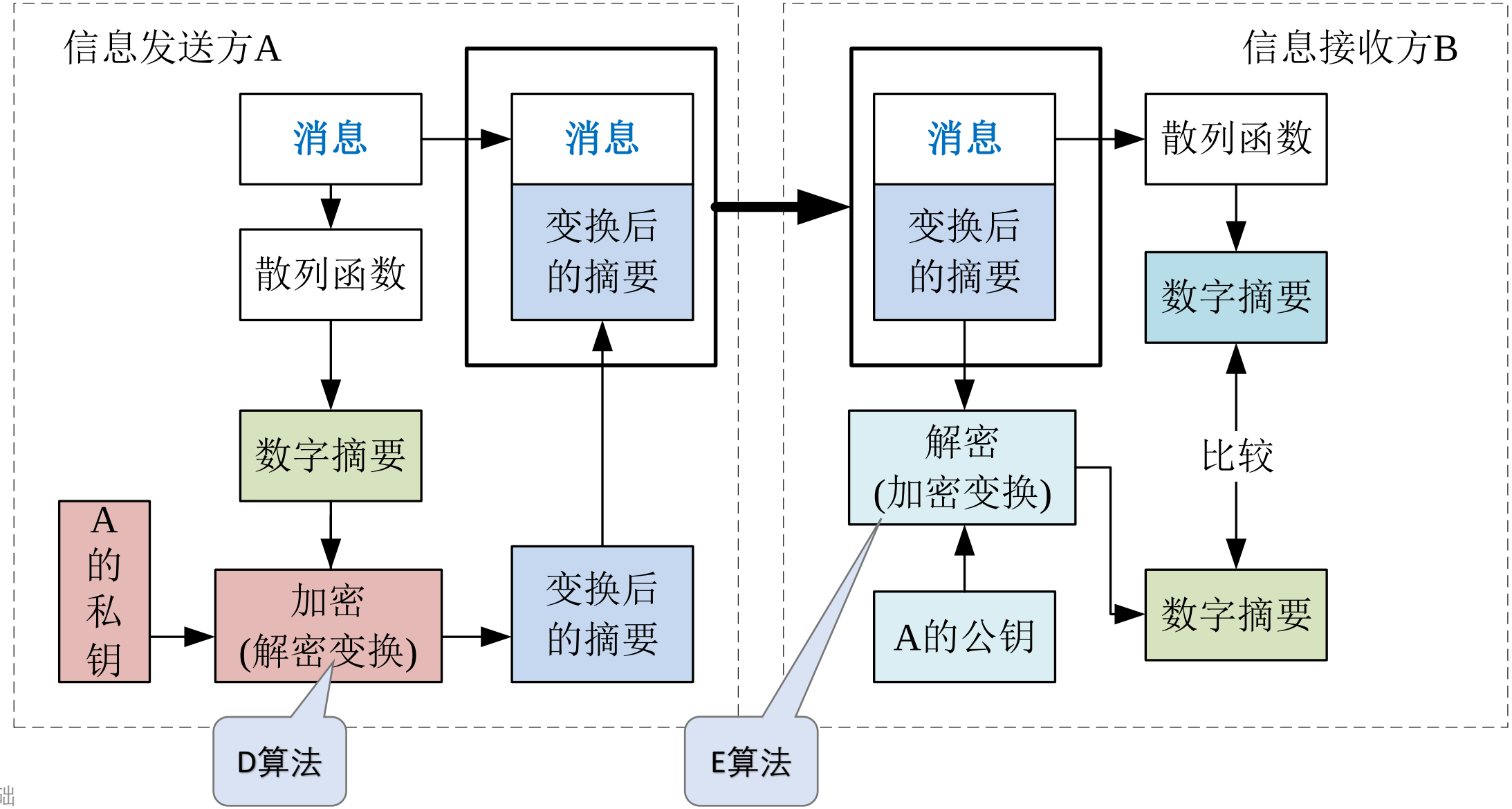
目前：
① 对于安全要求较高的应用，不能使用MD5
② 找到SHA-1的碰撞是较困难的；SHA-2是安全的

3.4.2 数字签名

- **数字签名(Digital Signature)：**
  - 是指用户用自己的私钥对原始数据的消息摘要进行加密所得的数据，即加密的摘要。
- **数字签名的验证：**
  - 信息接收者使用信息发送者的公钥对附在原始信息后的数字签名进行解密后获得消息摘要M1，并与原始数据产生的消息摘要M2对照，便可确信原始信息是否被篡改。

- **签名和验证**保证了消息来源的**真实性**和数据传输的**完整性**。

图3-7 用公钥算法实现数字签名及完整性验证



消息的保密传输

- 为了对消息进行保密传输，通常将公钥密码技术和对称密码技术结合起来使用。

① 在发送方A随机生成一个对称密码算法的密钥K。

② 用K对消息加密得到密文C并生成密文的数字摘要M，接着用A的私钥对M签名，用B的公钥对K及签过名的M进行加密，将**密文C**、**加密的K及加密的(签过名的)M**发送给接收方B。

③ 接收方B进行相反的操作，就可以实现消息的保密传输及完整性验证。

3.4.3 应用举例

[实例1] 分别使用MD5和SHA256算法，计算二进制文件test.bin的消息摘要。

openssl md5 file.bin

MD5(file.bin)= cdba07307835eb9818cfda3f574e137e

openssl sha256 file.bin

SHA256(file.bin)=
7189a476902350df6e7bc2c2cb0481c6a23480648c998b002366276770 9a21ec

openssl sha384 file.bin

SHA384(file.bin)=
3107c0a3c2ec92270d8df494c26a13d023cfa4110535d78d64dd0ec470e3f13a21541ddd aa39ef972e9ec979890a7da2

openssl sha512 file.bin

SHA512(file.bin)=
879ebaf7549b419ff6a90e6123819362493 65c9134bb2fe5925fa5b7fde1e96d77d99ddb be74f70c74af189b25d1f0be7cb742a37708db8152bc2105ccb58408

[实例2] 利用rsa算法，用私钥rsa_key.pem对文件test.sha256签名并输出为文件sig。用公钥rsa_pub.pem恢复签名sig的原始值，并验证是正确的。

- **步骤1：** 将文件test.bin的消息摘要保存在文件test.sha256中，用私钥rsa_key.pem签名并输出为文件sig:

  openssl sha256 file.bin > test.sha256

  openssl pkeyutl -sign -in test.sha256 -inkey rsa_key.pem -out sig

- **步骤2：** 用公钥rsa_pub.pem恢复签名sig的原始值，存入文件sig.src:

  openssl pkeyutl -verifyrecover -in sig -inkey rsa_pub.pem -pubin -out sig.src

- **步骤3：** 验证test.sha256和文件sig.src的内容一致:

  md5sum test.sha256 sig.src

  - e74bd28d466055f02e5121aca1434c67 test.sha256
  - e74bd28d466055f02e5121aca1434c67 sig.src

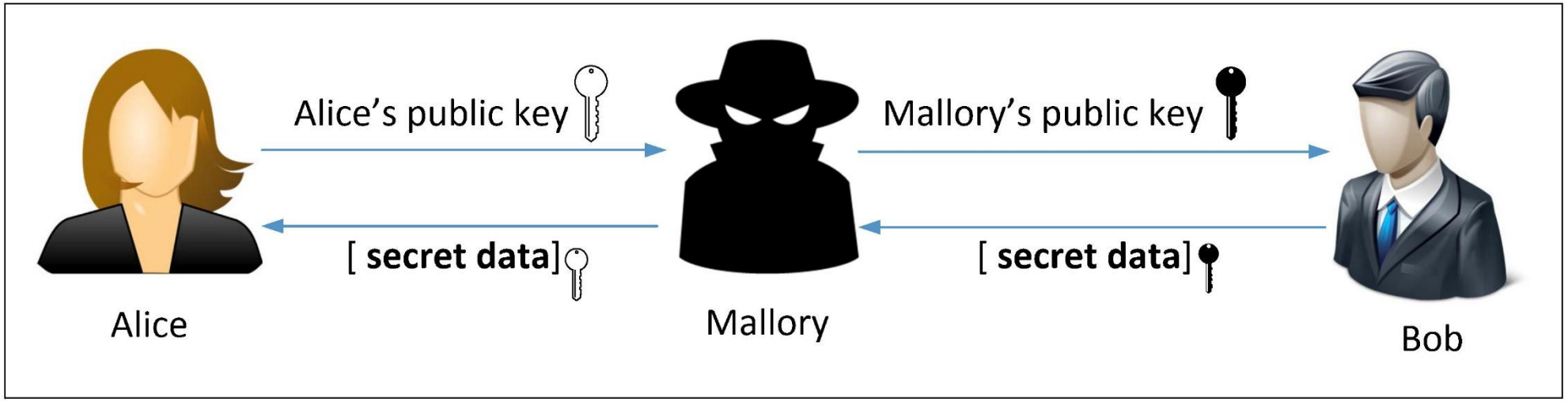
3.5 公钥基础设施及数字证书

- 为了实现保密通信，通信双方需要交换密钥。公钥密码体制是实现密钥交换的最佳方式，因为公钥是公开的，通信双方可以通过网络以明文的方式获取对方的公钥，用公钥对通信密钥进行加密。然而，该方式可能遭遇中间人攻击。考虑如下场景：

  (1) Bob和Alice生成一对公/私钥，各自保存私钥，把公钥发送给对方；

  (2) Bob用Alice的公钥加密一个文件并发送给Alice；

  (3) Alice用私钥解密文件，获得原始文件。



中间人攻击的原理

- 中间人攻击中的基本问题是Bob收到声称是Alice的公钥，但他没有办法判别这个公钥是否属于Alice。

- 为了解决该问题，可以使用**基于可信第三方的公开密钥基础设施（Public Key Infrastructure）**方案，把公钥和所有者的身份绑定在一起。

- **公开密钥基础设施简称公钥基础设施，即 PKI** 。
- PKI通过数字证书和数字证书认证机构（Certificate Authority，CA）确保用户身份和其持有公钥的一致性，从而解决网络空间中的信任问题。

3.5.1 PKI的定义和组成

- PKI是一种利用公钥密码理论和技术建立起来的提供信息安全服务的基础设施。PKI的核心是解决网络空间中的信任问题，确定网络空间中各行为主体身份的唯一性和真实性。
- PKI系统主要包括以下六个部分：

(1) 证书机构(CA)

证书机构（以下简称CA）也称为数字证书认证中心(或认证中心)，是PKI应用中权威的、可信任的、公正的第三方机构，必须具备权威性的特征

(2) 注册机构(RA)：是CA的延伸，负责对证书申请进行资格审查。

(3) 数字证书库：集中存放CA颁发的证书和证书撤销列表

(4) 密钥备份及恢复系统

(5) 证书撤销系统

(6) 应用接口(API)

3.5.2 数字证书及其应用

表3-1 X.509 数字证书的结构

Version	version 1	version 2	version 3
Serial Number			
Signature Algorithm Identifier			
Issuer Name			
Validity Period			
Subject Name			
Subject Public Key Information			
Issuer Unique ID			
Subject Unique ID			
Extensions			
Certification Authority's Digital Signature			

目前最常用的数字证书是X.509格式的证书，其结构如表3-1所示。

(1) 版本：标明X.509公钥证书所使用的版本(1~3，目前使用的是第3个版本);

(2) 序列号：列出证书序列号码，它在同一个证书机构内不能重复使用;

(3) 签名算法名称:

列出证书机构给证书加密所使用的算法名称，如sha1RSA表明证书机构将用SHA-1散列函数求出公钥证书的散列值，然后用RSA将散列值加密;

(4) 签发者：列出证书签发者的标准名称;

(5) 有效期：列出证书的有效期，它含起始日期和终止日期两个日期;

(6) 用户名：列出证书拥有者的姓名或名称;

(7) 用户公钥信息：列出公钥的算法名称和公钥值;

(8) 签发者唯一ID（第2版，第3版）;

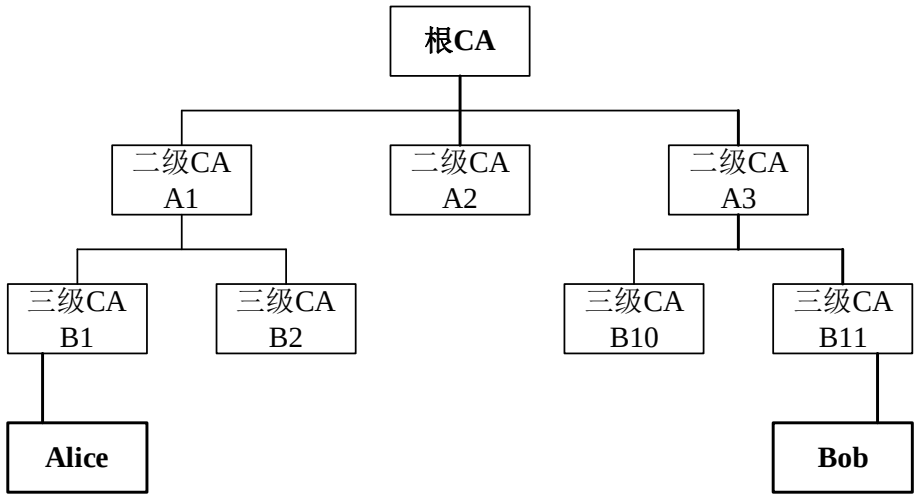
(9) 用户唯一ID（第2版，第3版）;

(10) 扩展项目：列出其他有关信息（仅第3版）;

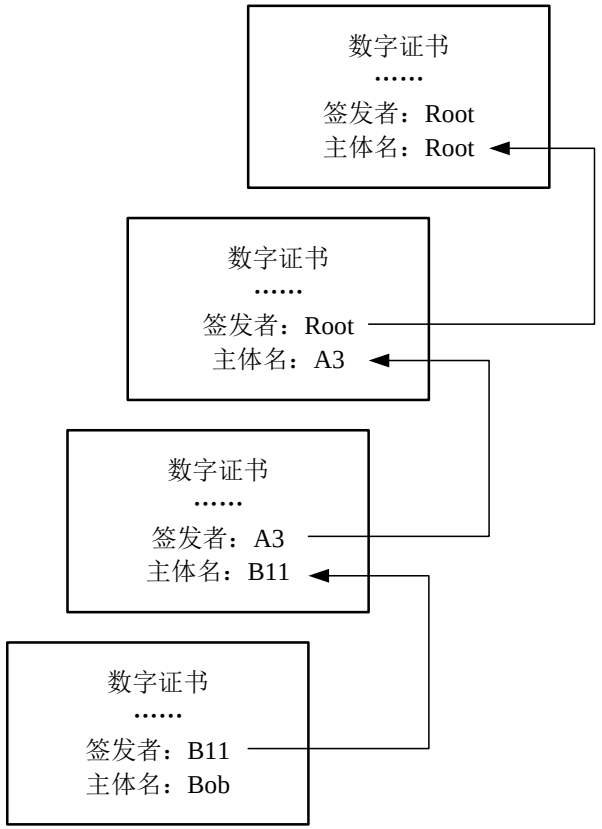
(11) 数字签名：列出经证书机构私钥加密的证书散列值。

数字证书的签发和验证

证书的签发



(a) 分层的CA



(b) 证书的逆向验证

图3-8 层次化的CA及证书的验证

证书的验证

- 用户Alice和Bob的数字证书从第三级CA获得。
- 若Alice要向Bob发送加密的信息，则首先从证书库（或Bob）中获得Bob的证书，然后按图(b)的步骤验证证书的真伪。
- **根CA的证书是自签名的证书，随操作系统、可信软件或通过可信途径导入。**

3.5.3 应用举例

- PKI的数字证书认证机构（Certificate Authority，CA）通过发行数字证书，以确保数字证书中的用户身份和其持有公钥的一致性。
- Openssl实现了PKI所需的所有功能。
- 本节用Openssl建立一个模拟的根认证机构，用该CA发行数字证书，并用数字证书部署HTTPS。

3.5.3.1 成为CA

- 此处建立的CA是根CA，机构名为nisCA，其域名为zengfp.ustc.edu.cn。需要为根CA生成密钥及自签名的证书。

步骤1：部署CA。

- 对数字证书进行签名时，openssl会使用一个默认的配置文件(**/usr/lib/ssl/openssl.cnf**)，该文件中的[CA_default]已经配置了需要的文件夹和文件的名字，部分内容如下：

```
dir             = ./demoCA          # Where everything is kept
certs           = $dir/certs        # Where the issued certs are kept
crl_dir         = $dir/crl          # Where the issued crl are kept
database        = $dir/index.txt    # database index file.
new_certs_dir   = $dir/newcerts     # default place for new certs.
serial          = $dir/serial       # The current serial number
```

步骤1：部署CA：配置环境

- 根据这些配置，需要在当前的工作目录中创建一个名为demoCA的目录，并且在demoCA中创建三个文件夹：certs、crl和newcerts。还需要在demoCA中创建以下两个文件：index.txt和serial。
- serial文件包含证书的序列号。可以将任意数字（比如1000）放在文件中来初始化序列号。
- 实现以上操作的命令如下：

```
mkdir demoCA
cd demoCA
mkdir certs crl newcerts
touch index.txt  serial
echo 1000 > serial
```

步骤2：为CA生成“公钥—私钥”对和证书。

- 每一个CA都需要有自己的数字证书。如果CA是一个中间CA，那么它的证书必须由其他CA颁发。如果CA是一个根CA，则需要自己给自己颁发公钥证书，也就是自己在公钥证书上签名。这样的证书称为自签名证书。
- 本例的nisCA为根CA，我们需要为nisCA创建“公钥—私钥”对，并且为它生成一个自签名的证书。命令如下：

  openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -keyout nisCA_key.pem -out nisCA_cert.pem

- 以上命令会向用户询问一些信息（比如，口令，假设为：12345678），然后生成一个“公钥—私钥”对(4096位的RSA)。**特别注意：应该输入正确的Organization Name和Common Name等信息，并记录下来备用。**
- 选项“-sha256”指定使用SHA2算法生成单向哈希值，选项“-days 3650”指定该证书的有效期是3650天。公钥和私钥存储在受密码保护的文件nisCA_key.pem中。选项“-x509”指明生成一个自签名的公钥证书，而不是一个证书请求。自签名的证书保存在nisCA_cert.pem中。
- 以下命令查看证书的信息：**openssl x509 -text -noout -in nisCA_cert.pem**

3.5.3.2 用户从CA获取X.509数字证书

- 假设用户xyz的域名为www.xyz.com，为了获取X.509证书，它首先需要生成一对公钥和私钥，然后从一个CA那里得到公钥的X.509证书。

步骤1：生成“公钥—私钥”对。

openssl genrsa -aes128 -out xyz_key.pem 2048

- 以上命令生成一个名为xyz_key.pem的文件，这个文件由用户设置的密码保护（如果没有“-aes128”选项，则不需要密码保护），密钥长度为2048比特。

- 查看xyz_key.pem的实际内容。

openssl rsa -text -noout -in xyz_key.pem

导出公钥：**openssl rsa -in xyz_key.pem -pubout -out xyz_pub.pem**

Enter pass phrase for xyz_key.pem:

Private-Key: (2048 bit, 2 primes)

modulus: **（n=p×q，模数）**

00:a7:d1:82:cf:69:9d:47:3e:29:0e:28:2a:3a:c7:

.....

publicExponent: 65537 (0x10001) **（公钥指数，e）**

privateExponent: **（私钥指数，d）**

08:81:3c:59:39:47:ad:e8:9f:39:5c:72:67:70:2c:

.....

prime1: **（质数p）**

00:e2:46:dd:70:55:74:04:ad:01:90:12:5f:08:3f:

.....

prime2: **（质数q）**

00:bd:dc:d5:2d:80:2b:67:a7:6d:c4:48:76:ee:57:

用户从CA获取X.509数字证书

- **步骤2：生成证书签名请求。**

  openssl req -new -key xyz_key.pem -out xyz.csr -sha256

  - openssl程序会要求提供主体信息，如公司名、地址、邮件等。在常用名(CN)域，使用 xyz.com 。
  - 查看CSR文件的实际内容

    openssl req -in xyz.csr -text -noout

步骤3：CA的验证与签名。

cd ..

openssl ca -in xyz/xyz.csr -out xyz/xyz_cert.pem -md sha256 -cert demoCA/nisCA_cert.pem -keyfile demoCA/nisCA_key.pem

- 查看证书的信息：

  openssl x509 -text -noout -in xyz/xyz_cert.pem

3.5.3.3 在网络服务器中部署公钥证书

- 首先使用openssl自带的服务器用于测试，需要将用户的私钥文件和公钥证书合并到一个文件(xyz_all.pem)中，然后使用命令“openssl s_server”启动服务器。服务器监听4433端口。

  cp xyz_key.pem xyz_all.pem

  cat xyz_cert.pem >> xyz_all.pem

  openssl s_server -cert xyz_all.pem -accept 4433 -www

- 当用火狐浏览器访问https://xyz.com:4433时，将会提示错误信息，表明这个连接是不安全的。这是因为浏览器没有nisCA的公钥，因此它不能验证xyz证书中的签名。为此，用户手动添加一个CA证书(nisCA_cert.pem)到它的信任列表中。
- 重新浏览器后，将可以正常访问。
- 可以使用“openssl s_client”命令访问网络服务器，观察客户端与服务器之间的交互：

  openssl s_client -connect xyz.com:4433

3.5.3.4 使用Apache部署HTTPS

- 为了建立一个HTTPS网站，只需配置Apache服务器，让它知道从哪里得到私钥和证书即可。添加下面的VirtualHost条目到Apache配置文件default-ssl.conf中，该文件位于/etc/apache2/sites-available/目录中(443是HTTPS默认的端口号)。

```
<VirtualHost *:443>
    ServerName xyz.com
    DocumentRoot /var/www/html
    DirectoryIndex index.html
    SSLEngine On
    SSLCertificateFile  /home/i/ns/ch03/xyz/xyz_cert.pem
    SSLCertificateKeyFile /home/i/ns/ch03/xyz/xyz_key.pem
</VirtualHost>
```

- 运行下面的一系列命令来启用SSL

  sudo apachectl configtest //测试Apache的配置文件，看是否有错误

  sudo a2enmod ssl //启用SSL

  sudo a2ensite default-ssl //启用刚才加入的网站

  sudo service apache2 restart //重启Apache 服务器

浏览网站https://.xyz.com

3.6 使用OpenSSL中的密码函数

- 在C语言程序中使用OpenSSL中的密码函数，需要利用openssl提供的C语言接口，以函数调用的形式使用加密函数库。

3.6.1 在Linux的C程序中使用OpenSSL

- Linux系统的发行版一般预装了命令行OpenSSL程序，没有安装openssl库。为了在C程序中使用OpenSSL，需要安装openssl库。
- 在ubuntu Linux 系统中运行以下命令安装openssl库：

  sudo apt-get install libssl-dev

- 在fedora Linux 系统中切换到root，再运行以下命令安装openssl库：

  yum install openssl-devel.x86_64

  或 yum install openssl-devel.i686

示例程序cryptoDemo.cpp：AES算法的实例

```
#include <memory.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "openssl/aes.h"
void testAes(char inString[], int inLen, char passwd[], int pwdLen)
{       int i,j, len, nLoop, nRes;
        char enString[1024]; char deString[1024];
        unsigned char buf[16];
        unsigned char buf2[16];
        unsigned char aes_keybuf[32];
        AES_KEY aeskey;
        // 准备32字节(256位)的AES密码字节
        memset(aes_keybuf,0x90,32);
        if(pwdLen<32){ len=pwdLen; } else { len=32;}
        for(i=0;i<len;i++) aes_keybuf[i]=passwd[i];
        // 输入字节串分组成16字节的块
        nLoop=inLen/16; nRes = inLen%16;
        // 加密输入的字节串
        AES_set_encrypt_key(aes_keybuf,256,&aeskey);
        for(i=0;i<nLoop;i++){
                memset(buf,0,16);
                for(j=0;j<16;j++) buf[j]=inString[i*16+j];
                AES_encrypt(buf,buf2,&aeskey);
                for(j=0;j<16;j++) enString[i*16+j]=buf2[j];
        }
```

```
        if(nRes>0){
                memset(buf,0,16);
                for(j=0;j<nRes;j++) buf[j]=inString[i*16+j];
                AES_encrypt(buf,buf2,&aeskey);
                for(j=0;j<16;j++) enString[i*16+j]=buf2[j];
                //puts("encrypt");
        }
        enString[i*16+j]=0;
        // 密文串的解密
        AES_set_decrypt_key(aes_keybuf,256,&aeskey);
        for(i=0;i<nLoop;i++){
                memset(buf,0,16);
                for(j=0;j<16;j++) buf[j]=enString[i*16+j];
                AES_decrypt(buf,buf2,&aeskey);
                for(j=0;j<16;j++) deString[i*16+j]=buf2[j];
        }
```

```
        if(nRes>0){
                memset(buf,0,16);
                for(j=0;j<16;j++) buf[j]=enString[i*16+j];
                AES_decrypt(buf,buf2,&aeskey);
                for(j=0;j<16;j++) deString[i*16+j]=buf2[j];
                //puts("decrypt");
        }
        deString[i*16+nRes]=0;
        //比较解密后的串是否与输入的原始串相同
        if(memcmp(inString,deString,strlen(inString))==0)
       { printf("test success\n");} else { printf("test fail\n");}
        printf("The original string is:\n\t %s\n", inString);
       printf("The encrypted string is:\n\t %s\n", enString);
       printf("The decrypted string is:\n\t %s\n", deString);
}
```

cryptoDemo.cpp的编译和执行

```
int main(int argc, char* argv[])
{
    char inString[] = "The sample program cryptoDemo.cpp uses the AES algorithm to encrypt and decrypt strings.";
    char passwd[] = "0123456789ABCDEFGHIJK";

    testAes(inString, strlen(inString), passwd, strlen(passwd));

    return 0;
}
```

- 运行以下命令编译和执行程序，结果如下所示。

  gcc -o cryptoDemo cryptoDemo.cpp -lcrypto

  ./cryptoDemo

- **输出如下：**

  test success

  The original string is:

  The sample program cryptoDemo.cpp uses the AES algorithm to encrypt and decrypt strings.

  The encrypted string is:

  ◆◆
  ◆◆◆◆◆Xμ◆"□a◆)◆◆◆◆◆ゲA◆ezj◆◆$Rpr/◆P◆◆66
  ◆8Ä◆42FKKb◆◆.◆oQ`)◆

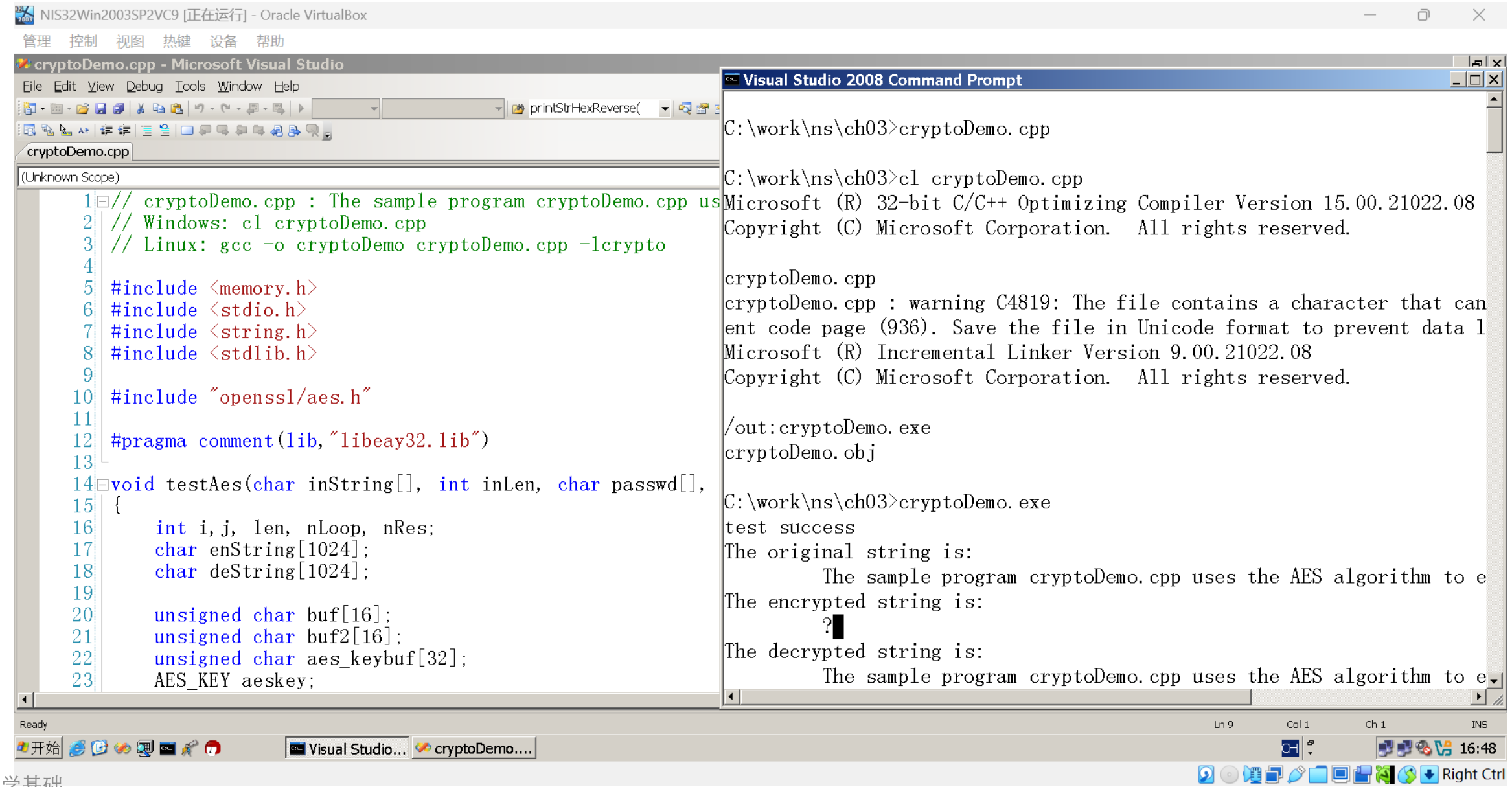
  The decrypted string is:

  The sample program cryptoDemo.cpp uses the AES algorithm to encrypt and decrypt strings.

3.6.2 在Windows的C程序中使用OpenSSL

- OpenSSL提供了C语言接口所需的头文件、库文件和动态链接库。
- 为了使用该接口，**必须安装面向软件开发人员的软件包**(安装文件较大)，并将openssl的lib和include目录添加到lib和环境变量中。
- 对于Visual Studio C++开发平台，最简单的方法是将openssl的lib和include目录拷贝到VC目录(默认安装在C:\Program Files\Microsoft Visual Studio 9.0\VC)中，这样就不需要额外设置环境变量。
- 为了使用OpenSSL库函数，在C程序中必须包含相应的头文件，链接的时候必须加入相关的库。
- 本例的演示环境为32位的Windows 2003系统，对应的openssl软件为Win32OpenSSL-1_0_1i.exe。

演示：在Windows的C程序中使用OpenSSL



3.7 python语言的密码模块

- 在pipy.org有2个可用的python密码模块

  旧的pycrypto:

  - Latest version Released: Oct 18, 2013
  - https://pypi.org/project/pycrypto/

  新的pycryptodome :

  - https://pypi.org/project/pycryptodome/

- pycryptodome的安装和使用

  安装： pip install pycryptodome

  文档： https://www.pycryptodome.org/

  使用AES： from Crypto.Cipher import AES

示例程序：aes_demo.py

用python密码库的AES算法，实现数据的加密和解密

加密

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_EAX)
ciphertext, tag = cipher.encrypt_and_digest(data)

file_out = open("encrypted.bin", "wb")
[ file_out.write(x) for x in (cipher.nonce, tag, ciphertext) ]
file_out.close()
```

解密

```
from Crypto.Cipher import AES

file_in = open("encrypted.bin", "rb")
nonce, tag, ciphertext = [ file_in.read(x) for x in (16, 16, -1) ]

# let's assume that the key is somehow available again
cipher = AES.new(key, AES.MODE_EAX, nonce)
data = cipher.decrypt_and_verify(ciphertext, tag)
```

3.8 PGP及其应用

PGP（Pretty Good Privacy）是一个基于RSA公钥加密体系的邮件加密软件。PGP加密技术的创始人是美国的Phil Zimmermann。他创造性地把RSA公钥体系和传统加密体系的结合起来，并且在数字签名和密钥认证管理机制上有巧妙的设计，因此PGP成为目前几乎最流行的公钥加密软件包。

PGP最初在Windows实现，直到PGP Desktop9.0一直为免费共享软件，后来PGP被Symantec收购，成为了收费软件。

OpenPGP (http://www.openpgp.org/index.shtml)是源自PGP 标准的免费开源实现，目前是世界上应用最广泛的电子邮件加密标准。OpenPGP 由IETF 的OpenPGP工作组提出，其标准定义在**RFC 4880**。在Windows和Linux(Unix)下均有**免费开源**的版本。

GunPG(The GNU Privacy Guard)是OpenPGP的最典型实现，目前支持Windows、Linux、MacOS等流行操作系统。相关软件可以从 http://www.gnupg.org/ 下载。

Windows版本的GunPG可从 http://www.gpg4win.org/ 网站下载，截至2025年9月最新版本为Gpg4win 4.4.1。

Gpg4win的应用

自学（不考核）

- 请从 http://www.gpg4win.org/ 网站下载合适版本的软件，安装到系统后试用。

谢谢！