第2章 密码体制与技术

本章内容

- 2.1 密码学入门
- 2.2 对称密码
- 2.3 非对称密码
- 2.4 数字签名

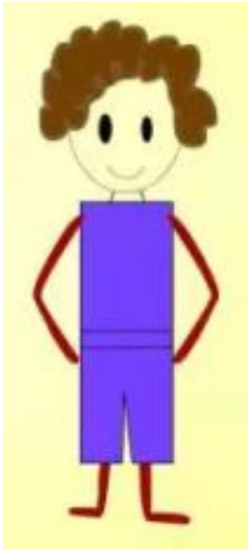
2.1　密码学入门

Alice和Bob的故事

- Alice请Bob进入一个房间

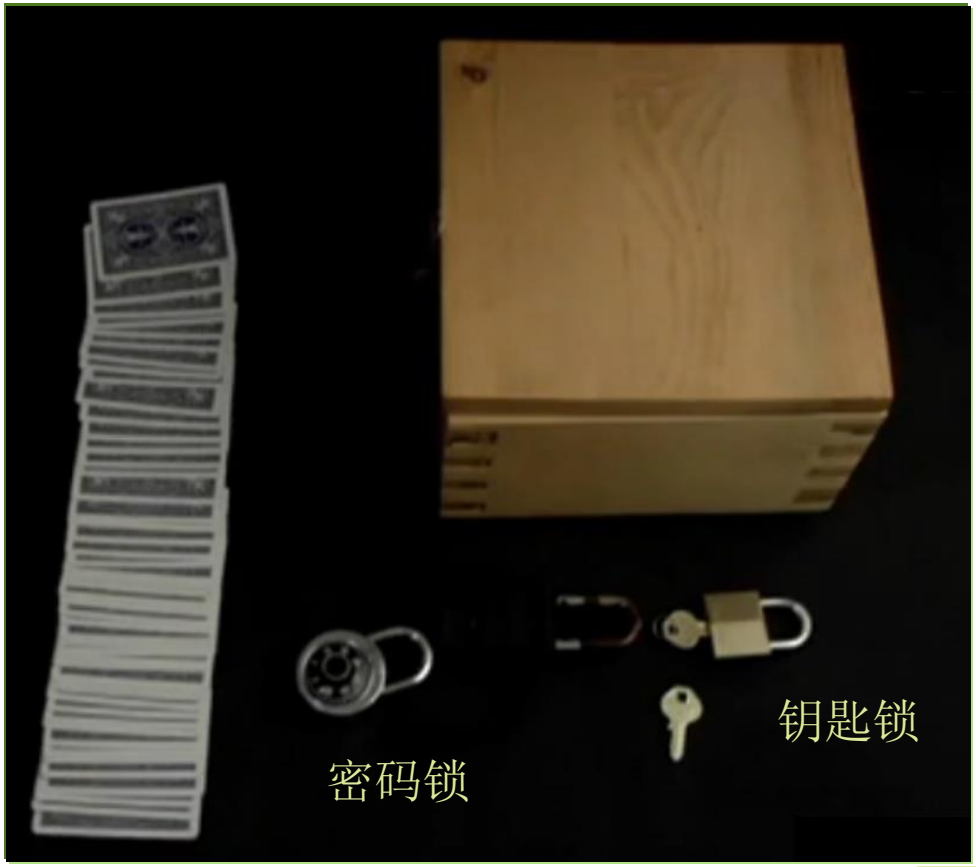
Alice

Bob

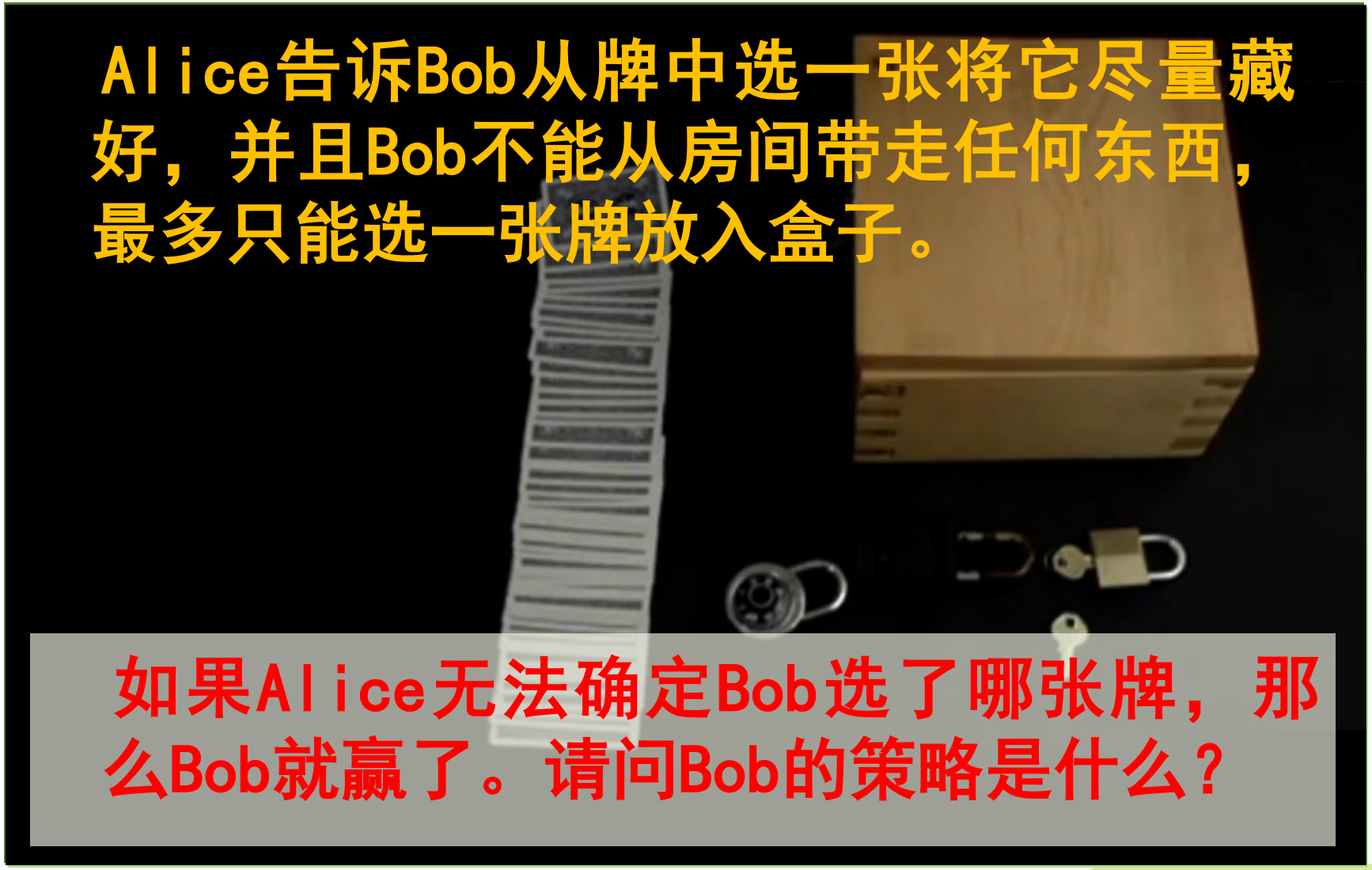


Alice和Bob的故事

- 房间是空的，只有一些锁、一个空盒子、一副牌。



Alice和Bob的故事



Alice、Bob 和他们的“朋友们”

表 2-1　剧中人

人　名	角　色
Alice	所有协议中的第一个参加者
Bob	所有协议中的第二个参加者
Carol	三、四方协议中的参加者
Dave	四方协议中的参加者
Eve	窃听者
Mallory	恶意的主动攻击者
Trent	值得信赖的仲裁者
Walter	监察人：在某些协议中保护 Alice 和 Bob
Peggy	证明人
Victor	验证者

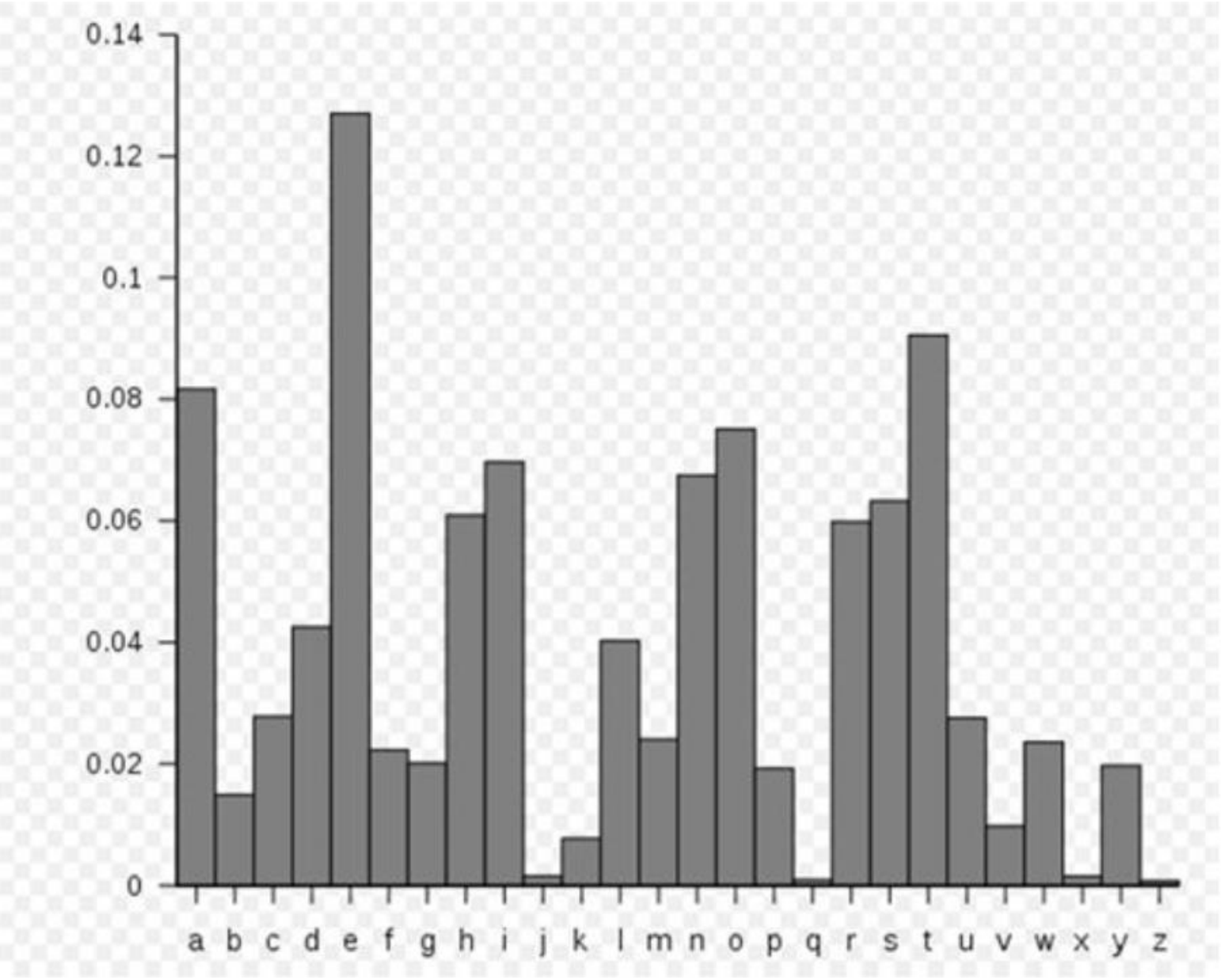
你能破译吗？

- 下面是一段密文，请问明文是什么？

AVCIC GN RJ SGLGA AJ AVC VCGEVAN

CYCIK LPR MPR PAAPGR XK ICLPGRGRE

JR AVC SCYCS


0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
a b c d e f g h i j k l m n o p q r s t u v w x y z

你能破译吗？

AVCIC GN RJ SGLGA AJ AVC VCGEVAN

THE E T T THE HE HT

CYCIK LPR MPR PAAPGR XK ICLPGRGRE

E E TT E

JR AVC SCYCS

THE E E

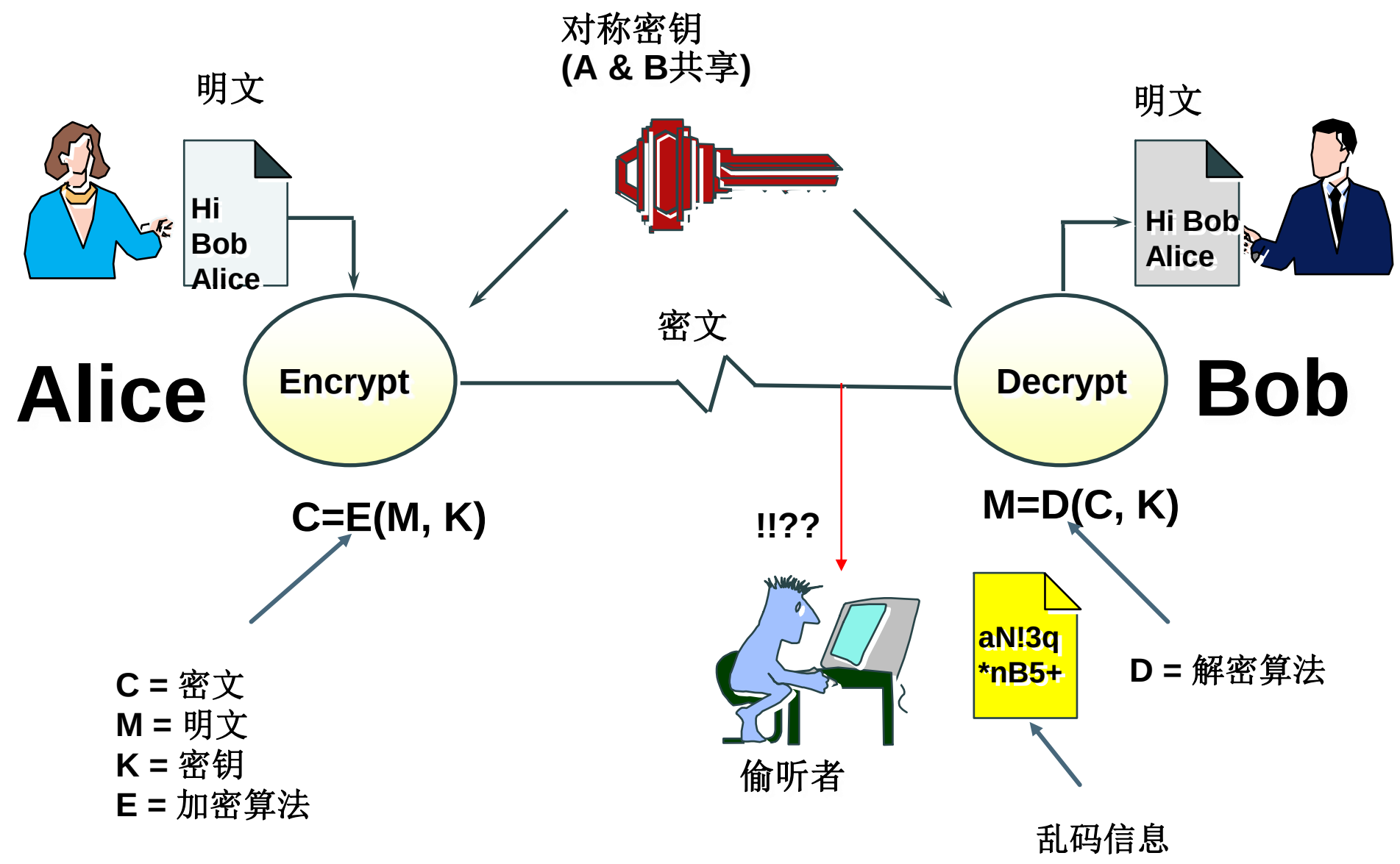
你能破译吗？

- 下面是一段密文，请问明文是什么？

AVCIC GN RJ SGLGA AJ AVC VCGEVAN
THERE IS NO LIMIT TO THE HEIGHTS
CYCIK LPR MPR PAAPGR XK ICLPGRGRE
EVERY MAN CAN ATTAIN BY REMAINING
JR AVC SCYCS
ON THE LEVEL

ABCDEFGHIJKLMNOPQRSTUVWXYZ

2.2　对称密码


对称密钥
(A & B共享)
明文
Hi
Bob
Alice
明文
Hi Bob
Alice
Alice
Encrypt
密文
Decrypt
Bob
C=E(M, K)
!!??
M=D(C, K)
aN!3q
*nB5+
C = 密文
M = 明文
K = 密钥
E = 加密算法
偷听者
D = 解密算法
乱码信息

对称密码体制的优点

- 对称密码体制根据对明文的加密方式的不同而分为**分组密码**和**序列密码**。
  - 分组密码：先按一定长度（如64比特、128比特等）对明文进行分组，以组为单位加/解密；
  - 序列密码：不进行分组，而是按位加密
- **对称密码体制的优点：**
  - 算法简单、速度快、适合加密大量数据

1　序列密码

序列密码实例

设$M=(0110010011)_2$，$K=(0111001001)_2$

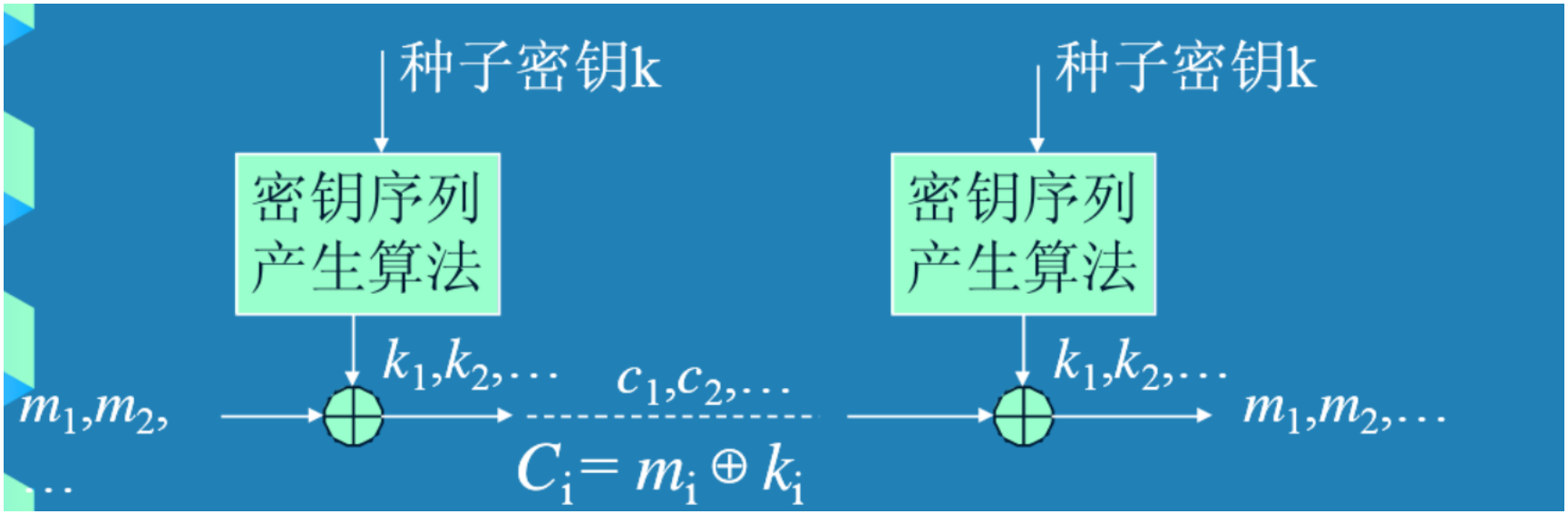
在A，B双方通信之前，A首先通过安全信道把密钥K传送给B，然后A将明文M进行加密变换，再通过公开信道传给B。加密过程：

$$C=E_K(M)=M \oplus K$$
$$=(0110010011)_2 \oplus (0111001001)_2$$
$$=(0001011010)_2$$

B收到密文C后，用密钥K进行解密，即：

$$M'=D_K(C)=C \oplus K$$
$$=(0001011010)_2 \oplus (0111001001)_2$$
$$=(0110010011)_2=M$$

同步序列密码



- 密钥序列的产生算法与明文无关
- 在通信过程中，通信双方必须保持精确的同步，接收方才能正确解密，例如，通信中丢失或增加一个密文字符，则接收方的解密将一直错误。

设密文失步 $c = c_1, c_3, c_4, \cdots c_{n-1}, c_n$ （c_2 丢失）

$\oplus$ $k = k_1, k_2, k_3, \cdots k_{n-1}, k_n$ （密钥正确）

$m = m_1, \times, \times, \cdots \times, \times$ （m_1 后的明文全错）

优点：

- 对失步的敏感性，使我们容易检测插入、删除、重放等主动攻击
- 没有错误传播，当通信中某些密文字符产生了错误，只影响相应字符

设密文错误 $c = c_1, c_2, c_3, \cdots c_{n-1}, c_n$ （c_2 错）

$\oplus$ $k = k_1, k_2, k_3, \cdots k_{n-1}, k_n$ （密钥正确）

$m = m_1, \times, m_3, \cdots m_{n-1}, m_n$ （仅 m_2 错）

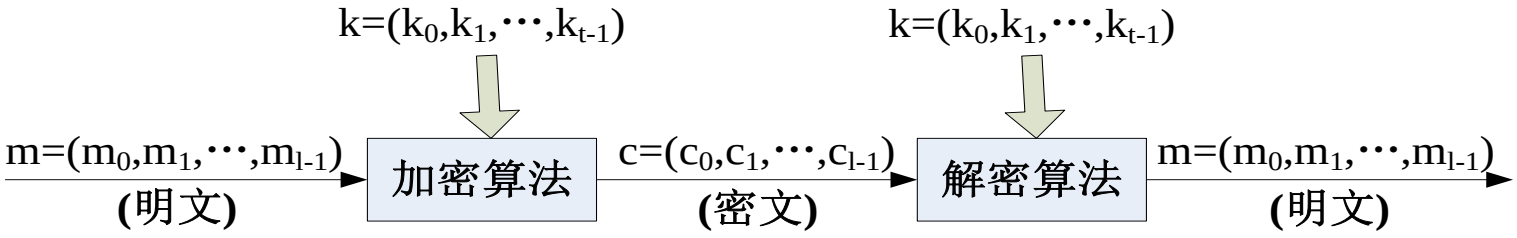
序列密码中的随机数

- 序列密码的保密性完全取决于密钥的随机性。
- 一般采用伪随机序列来代替随机序列作为密钥序列，也就是序列存在着一定的循环周期。这样序列周期的长短就成为保密性的关键。如果周期足够长，就会有比较好的保密性。

2　分组密码

分组密码原理

- 所谓分组密码是将明文分成一组一组，在密钥的控制下，经过加密变换生成一组一组的密文。具体而言，分组密码就是将明文消息序列 $m_1, m_2, \cdots, m_i, \cdots$ 划分成等长的消息组 $(m_1, m_2, \cdots, m_n), (m_{n+1}, m_{n+2}, \cdots, m_{2n}), \cdots$
- 在密钥 $K = k_1, k_2, \cdots, k_n$ 的控制下按固定的加密算法一组一组进行加密，输出一组一组密文 $(c_1, c_2, \cdots, c_l), (c_{l+1}, c_{l+2}, \cdots, c_{2l}), \cdots$



分组密码体制设计准则

分组密码体制的设计准则一般包括以下内容：

1．混乱原则（confusion）

又称混淆原则，是指明文与密钥以及密文之间的统计关系尽可能复杂化，使破译者无法推导出相互间的依赖关系，从而加强隐蔽性。

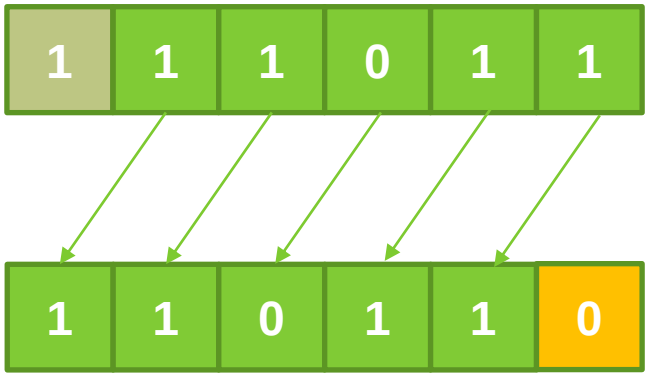
2．扩散原则（diffusion）

扩散原则是让明文中的每一位（包括密钥中的每一位）直接和间接影响输出密文中的许多位，或者让密文中的每一位受制于输入明文以及密钥中的若干位，以便达到隐蔽明文的统计特性。（明文和密钥中任何一比特值得改变，都会在某种程度上影响到密文值的变化，以防止将密钥分解成若干个孤立的小部分，然后各个击破。）

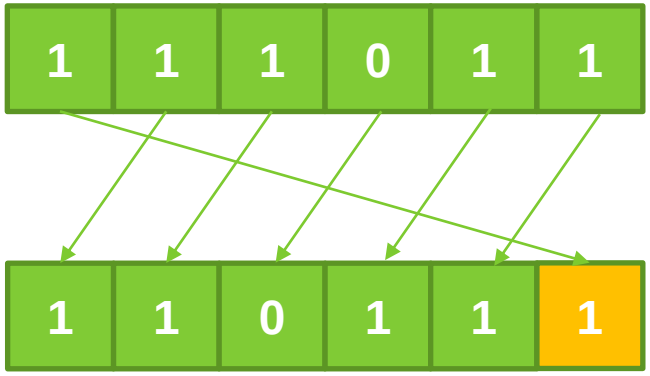
扩散和混淆是由Shannon提出的设计密码系统的两个基本方法，目的是抗击敌手对密码系统的统计分析。

分组密码常用操作：异或、移位、置乱、替换

- 移位（shift）

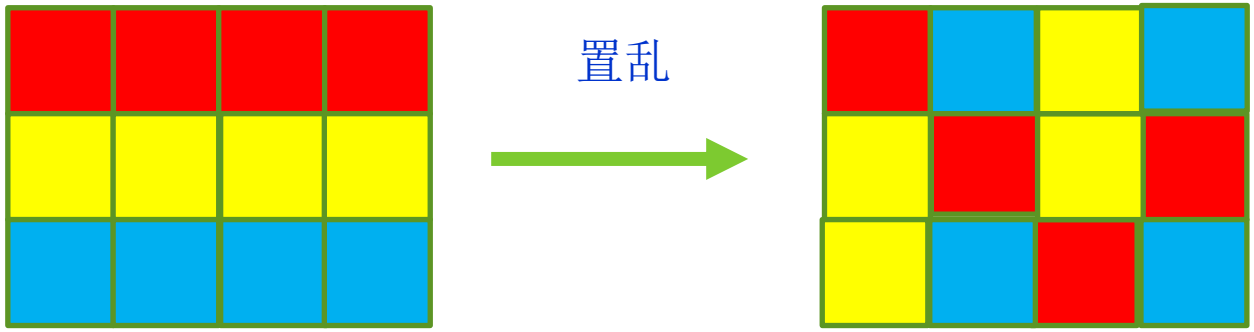


左移1位



循环移位

❖置乱（permute）

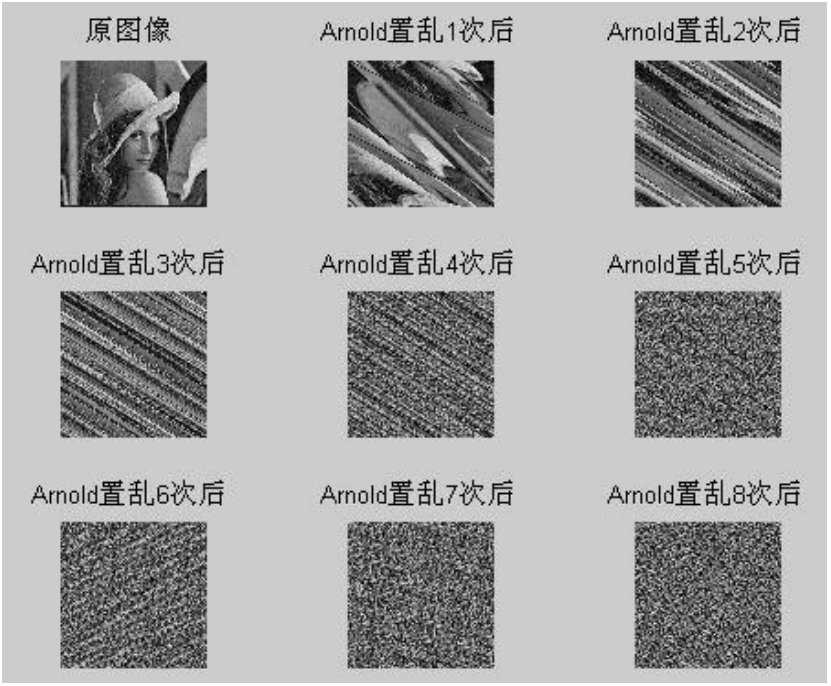


❖图像置乱
❖猫脸映射

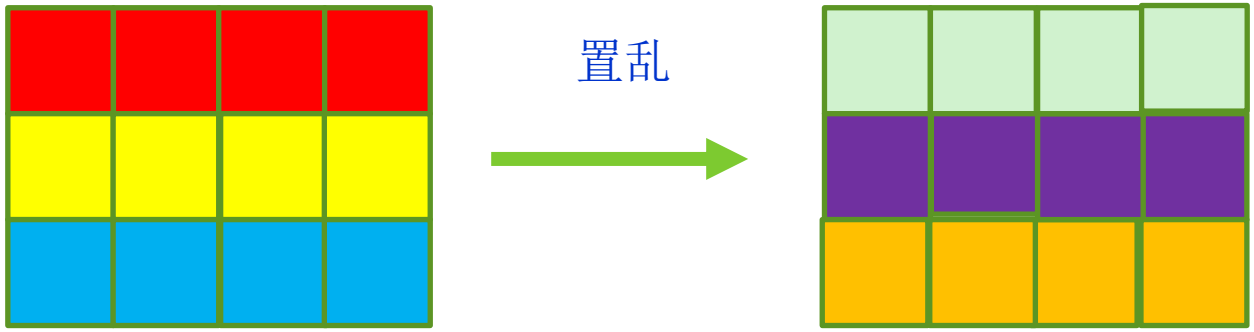
$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} \bmod N, \quad x, y \in \{0, 1, 2, \ldots, N-1\}$$

表 1　不同阶数 N 下数字图像的 A rnold循环次数

阶级 N	循环次数	阶级 N	循环次数	阶级 N	循环次数
2	3	8	6	60	60
3	4	9	12	100	150
4	3	10	30	120	60
5	10	12	12	125	250
6	12	25	50	256	192
7	8	50	150	512	384



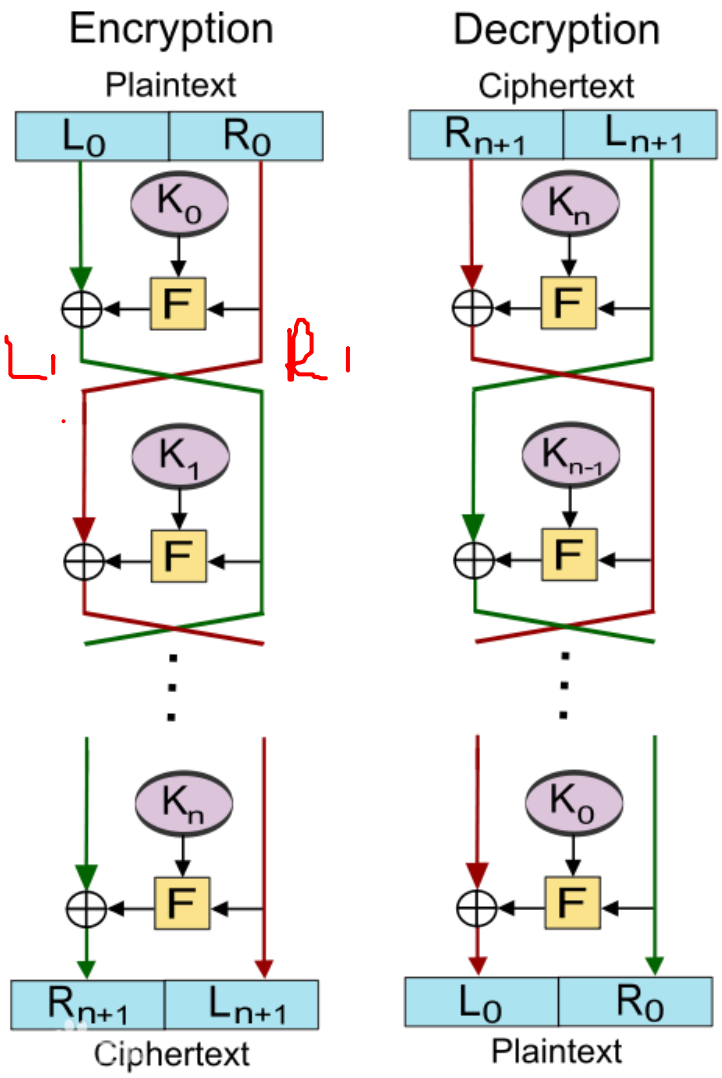
替代（substitute）



DES加密算法

- **发明人：**
  - IBM公司 W.Tuchman和C.Meyer
- **思想起源：**
  - 参照二战德国的恩尼格码机，混淆和扩散
- **密钥和分组长度：**
  - 64位，但每个第8为都用作奇偶校验，所以对于使用者而言，密钥长度是56位。分组长度64位
- **基本操作：**
  - 采用Feistel技术，16个循环，异或、移位、置乱、替代四种基本运算。

DES加密算法——Feistel结构

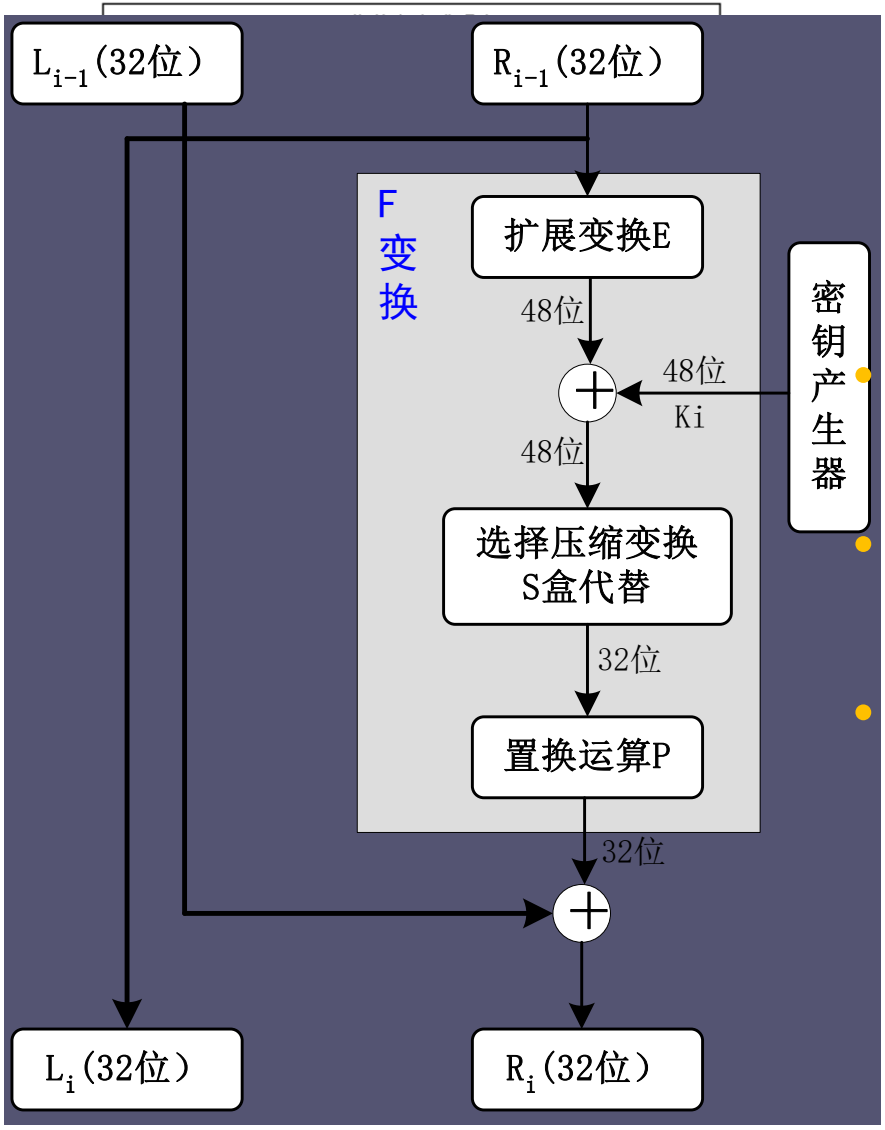


- 单轮基本操作
  - $R_i = L_{i-1} \oplus F(K_i, R_{i-1})$
  - $L_i = R_{i-1}$

❖**思考：**

Feistel结构的优点和缺点是什么？

DES加密算法



初始置换：重排输入明文块中的比特；

最终置换：初始置换的逆变换

扩展置换：32位R_{i-1}根据扩展规则扩展为48比特长度的串；

S盒替换：使用8个S盒S_1……S_8，S盒运算则是非线性的；

P盒置换：P置换使得一个S盒的输出对下一轮多个S盒产生影响，形成雪崩效应：明文或密钥的一点小的变动都引起密文的较大变化；

DES加密算法

- **思考：**

  DES算法中哪个部分实现了混淆？哪个部分实现了扩散？

DES加密算法

- 课后作业：

1. DES算法中为什么要扩展置换？？

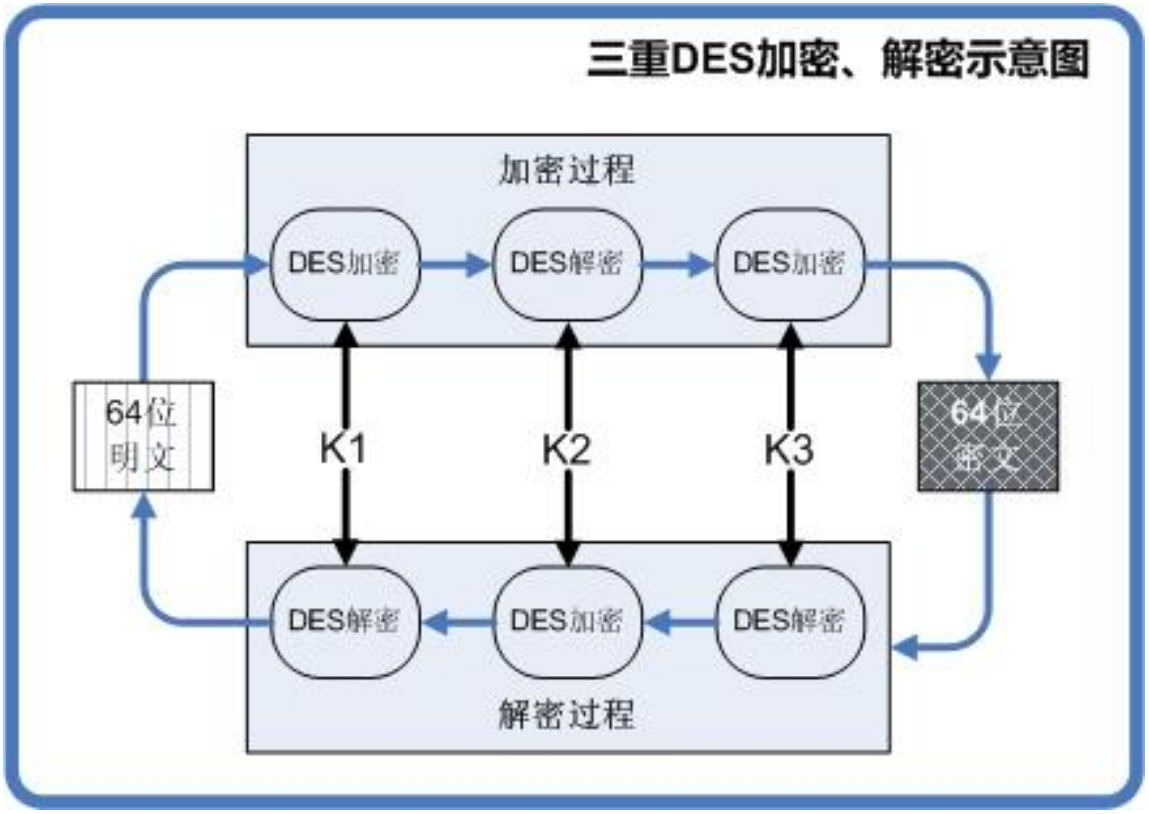
DES密钥短问题

- DES密钥空间多大？
- 2^{56}
- 1997年，用Internet网连上7万个终端，采用举式程序，每台机分配一个DES密钥空间，经过96天找出了正确的密钥，这时搜索了1/4的密钥空间，尽管如此，一般来说，出了极端敏感的场合，DES仍可安全用于商业应用场合。

  这意味着随着计算能力的增长，必须相应地增加算法密钥的长度。

3DES加密算法

- 目的：解决DES密钥短的问题。



- 思考：

为什么是“加密-解密-加密”模式，而不是“加密-加密-加密”模式？

AES加密算法

- 1997年，美国标准技术研究所（NIST）对DES进行再次评测并宣布：DES算法的安全强度已经不足以保障联邦政府信息数据的安全性，所以NIST建议撤销相关标准.
- 同时，NIST开始征集新的数据加密标准----高级数据加密标准(Advanced Encryption Standard).
- 1999年，NIST从提交的15个候选草案中选取了5个优良的算法作为AES的候选算法：MARS、RC6、Rijndael、Serpent和Twofish，
- 综合评价最终确定Rijndael算法为新的数据加密标准，

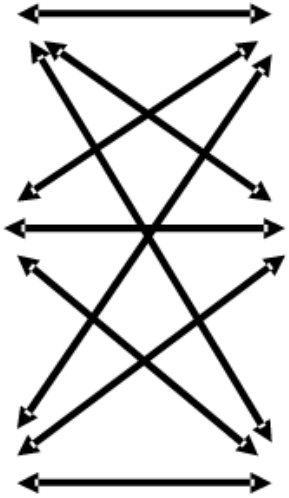
AES加密算法

- 高级加密标准，也叫Rijndael算法
- 由两位比利时密码学家发明，参与了NIST（美国标准和技术委员会）1997年组织的公开密码学竞赛，最终以优异的技术特性胜出成为加密标准。
- 分组长度和密钥长度可变

AES加密算法

表 1. 分组长度和密钥长度的不同取值

分组长度（bit）	密钥长度（bit）
128	128
192	192
256	256



对称密码体制的弱点

密钥管理

- 如何安全的共享秘密密钥，不可能与你未曾谋面的人通信
- 每对通信者间都需要一个不同的密钥。N个人通信需要C_N^2 ＝n(n-1)/2密钥。当用户量增大时密钥空间急剧增大。如：

n= 100 时C(100,2)= 4,995

n= 5000 时C(5000,2)= 12,497,500

没有解决不可抵赖问题

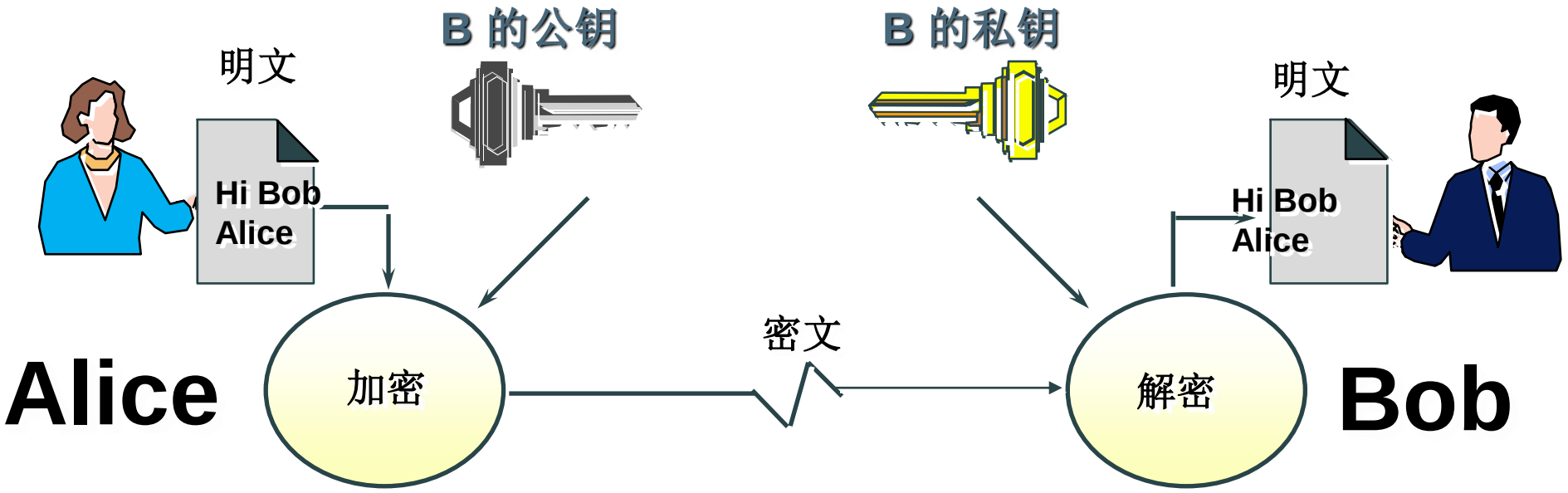
- 文档不能被签名
- 通信双方都可以否认发送或接收过的信息

2.3 非对称密码

起源

- 非对称密码又称为公钥密码和双钥密码
- Diffie和Hellman于1976年在《密码学的新方向》中首次提出了公钥密码的观点，标志着公钥密码学研究的开始
- W. Diffie and M. E. Hellman, New direction in cryptography, IEEE Trans. on Information Theory, 1976, IT-22.(6), pp.644-654.
- 1977年由Rviest，Shmair，Adlmena提出了第一个比较完善的公钥密码算法，即RSA算法

公钥密码的加密过程



- A 发送机密信息给 B, 知道只有 B 可以解密
- A 用 B 的公钥加密 (公开)
- B 使用自己的私钥解密 (保密)

公钥密码的原理

❖公钥密码的理论基础：单向陷门函数

单向函数：已知x, 计算y使得y=f(x)容易；

已知y, 计算x使得y=f(x)是难解的。

陷门单向函数：t是与f有关的一个参数

已知x, 计算y使得y=f(x)容易；

如果不知道t，已知y, 计算x使得y=f(x)是难的，

但知道t时，已知y, 计算x使得y=f(x)是容易的。

参数t称为陷门。

公钥密码的原理

- 设计公钥密码可以转换为寻找单向陷门函数。目前，主要基于如下的数学上的困难问题来设计单向函数和公钥密码体制：
- （1）大整数分解问题

若已知两个大素数p、q，求n=pXq仅需一次乘法，但已知n求p、q则是一道难题。

- （2）有限域上的离散对数问题

对于等式 $y = g^x \bmod p$，给定g、x和p，计算y是容易的。

反过来，若已知y、g和p，当p是大素数时，找到一个x，使 $y = g^x \bmod p$ 成立是困难的。

- （3）椭圆曲线上的离散对数问题

公钥密码特点

- 公钥密码算法是基于数学函数而不是基于替代和置换
- 公钥密码是非对称的，它使用两个独立的密钥，即公钥和私钥，任何一个都可以用来加密，另一个用来解密
- 公钥可以被任何人知道，用于加密消息以及验证签名，私钥仅自己知道，用于解密消息和签名

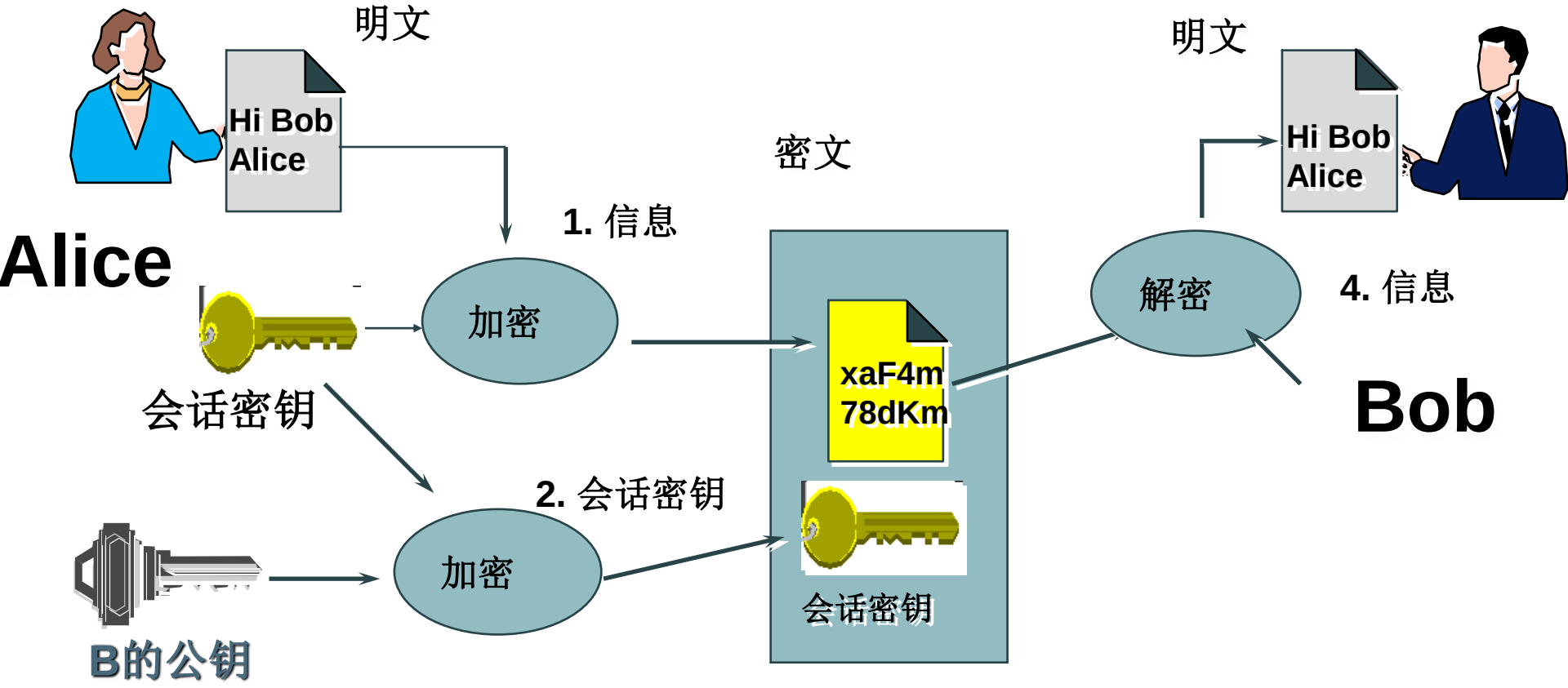
常用的公钥密码算法

- RSA
  - Ron Rivest, Adi Shamir和Len Adleman于1977年研制并且1978年首次发表
  - 可以用私钥加密和公钥加密
- DSA
  - 最初由NIST于1991年发布
  - 只能使用私钥加密，通常用作数字签名
- Diffie-Hellman算法
  - 只能用来进行对称密钥交换

非对称密码的问题

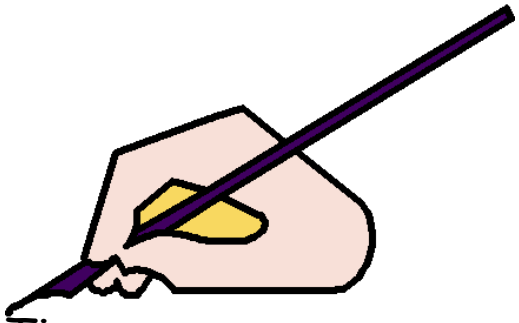
- **公钥加解密对速度敏感**
  - 大数幂运算，因此非常慢
  - 软件，公钥算法比对称密钥算法慢 100 多倍。（硬件可能慢 1000倍 ）
- **公钥加密长信息无法接受的慢，而对称密钥算法非常快**
- **结合公钥算法和对称密钥算法，使用对称密钥与公开密钥的优点**
  - 对称密钥快速而强健
  - 公开密钥易于密钥交换

组合对称密码和非对称密码



- 产生一个一次性，对称密钥——会话密钥
- 用会话密钥加密信息
- 最后用接收者的公钥加密会话密钥——因为它很短

2.4　数字签名



Digital Signature, Date, Time

数字签名技术

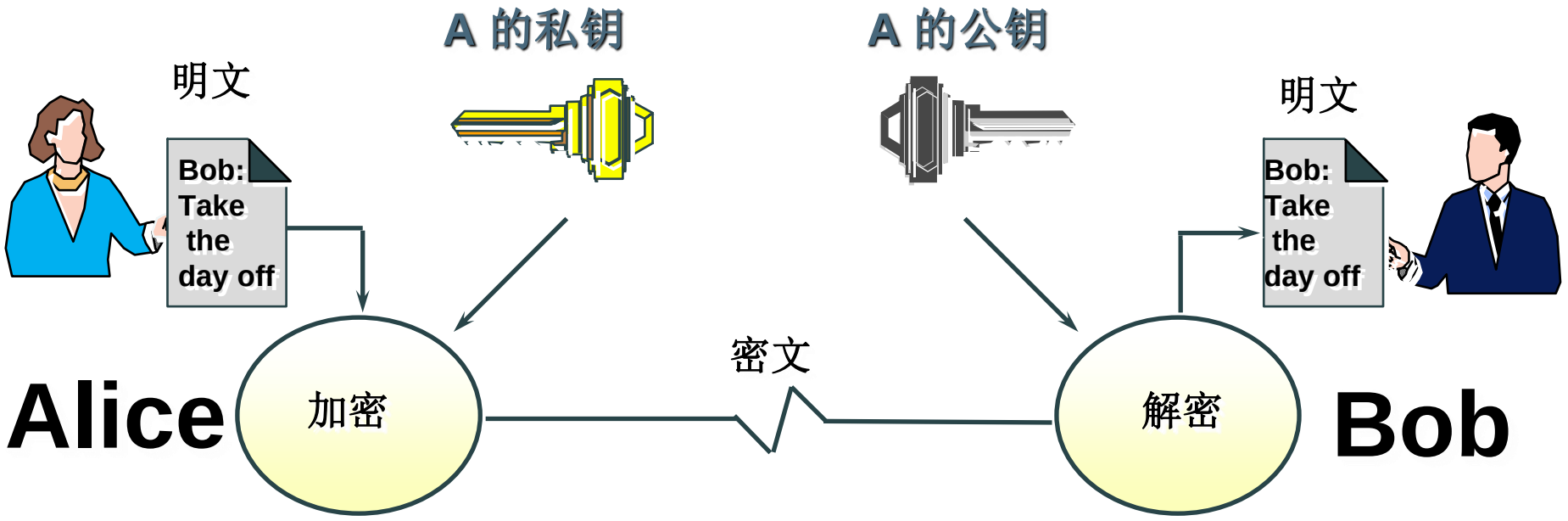
数字签名的需求

- Alice 需要一个方法签名一个信息，必须确认是从她发出，因此需要将她的身份和信息绑定在一起。
- 我们用传统的方法将Alice的普通签名数字化后附加在文档的后面
- 但是这个 *数字化* 的签名 ...
  - 它不能避免通过附加在其他文档中被伪造,
  - 无法防止对机密文档（比如支票）的篡改

数字签名的实现

- 需要一个数字码唯一标识一个人或实体
  - 身份证号码? No, 不保密
  - 私钥? Yes!
- 公钥与私钥是一对镜像
  - 用其中一个加密, 用另一个解密
- 解决方案：用发送者的私钥加密信息，然后用公钥解密
  - 如果能够解开，说明发送者加密并发送了本信息
  - 除非发送者的私钥不再保密

数字签名原理—公钥鉴别



- Alice 用她的私钥加密整个信息
- 所有人都可以解密这个信息
- 因此，Bob 可以确信这个信息是由 Alice 产生的 — 因为只有她的公钥可以解开该信息，而只有Alice 有对应的私钥
- 通过公钥鉴别，可以鉴别签名的真实性。

公钥鉴别的问题

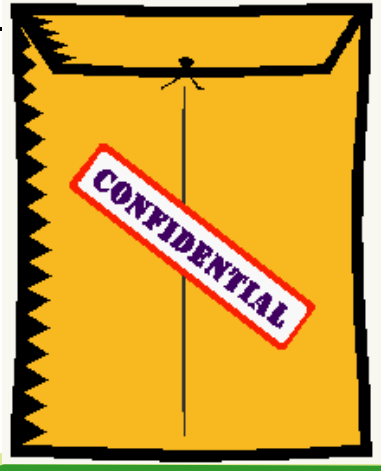
- 问题：签名太长
- 解决方法：签名一个短的信息—数字摘要
- 数字摘要（Message Digest）
  - 一个函数，输入一个任意长度的信息，而输出一个短的固定长度的编码
  - 一般 16 到 20 字节长
  - 对于输入信息 MD 是唯一
  - 无法找到具有相同 MD 的两个信息
  - 对于信息的任何修改，MD将改变

数字摘要技术

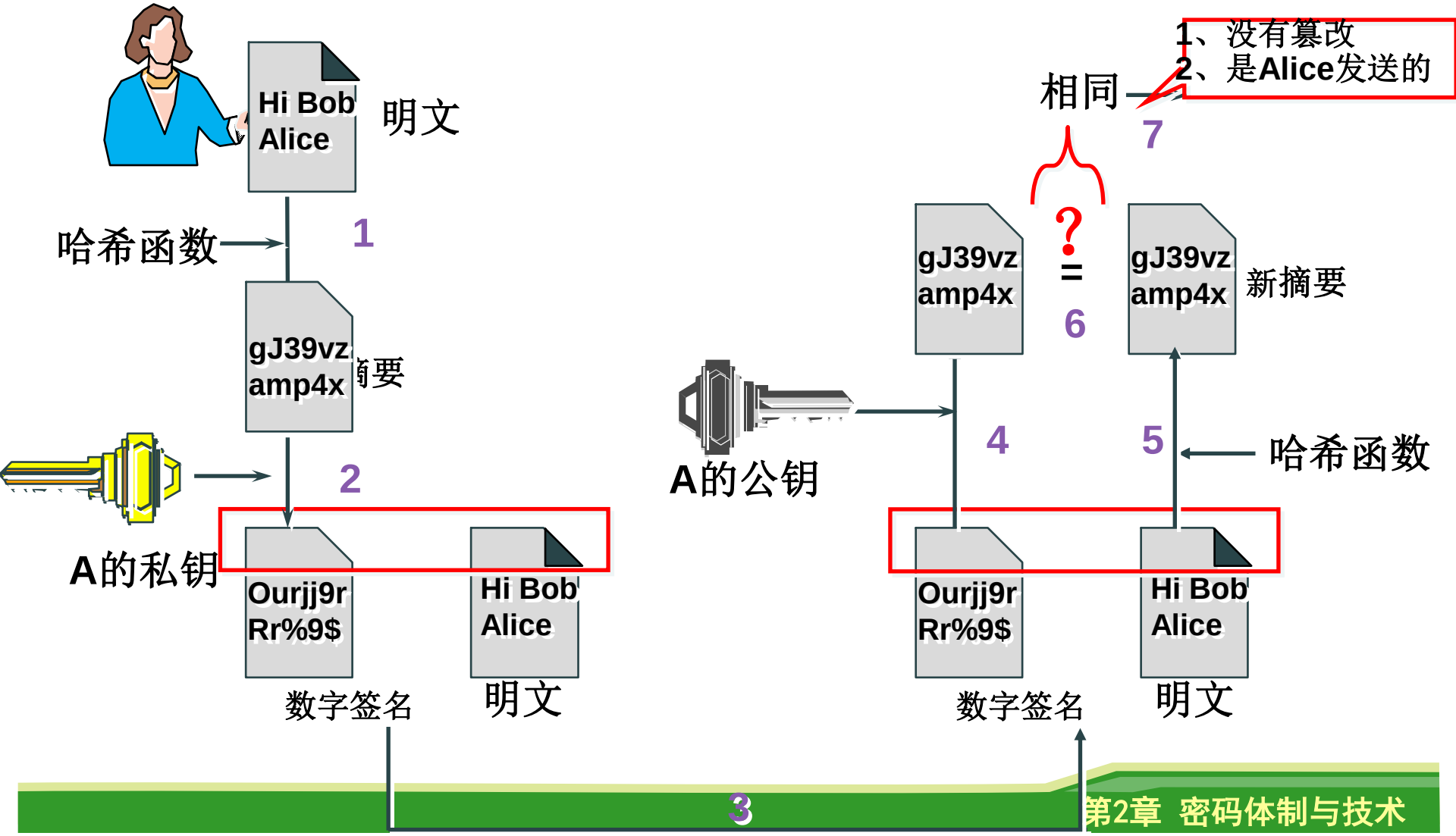
- 散列函数与指纹相象
  - 比原物（本人）信息量小
  - 与本人一一对应
  - 无法找到相同指纹的两个人
  - 知道了指纹，无法重建一个人
- 最常用的散列函数
  - RSA公司的 MD4 和 MD5 （128 位即16 字节）
  - NIST 的安全散列算法 SHA（160 位即20字节）

摘要算法 – 数据的完整性

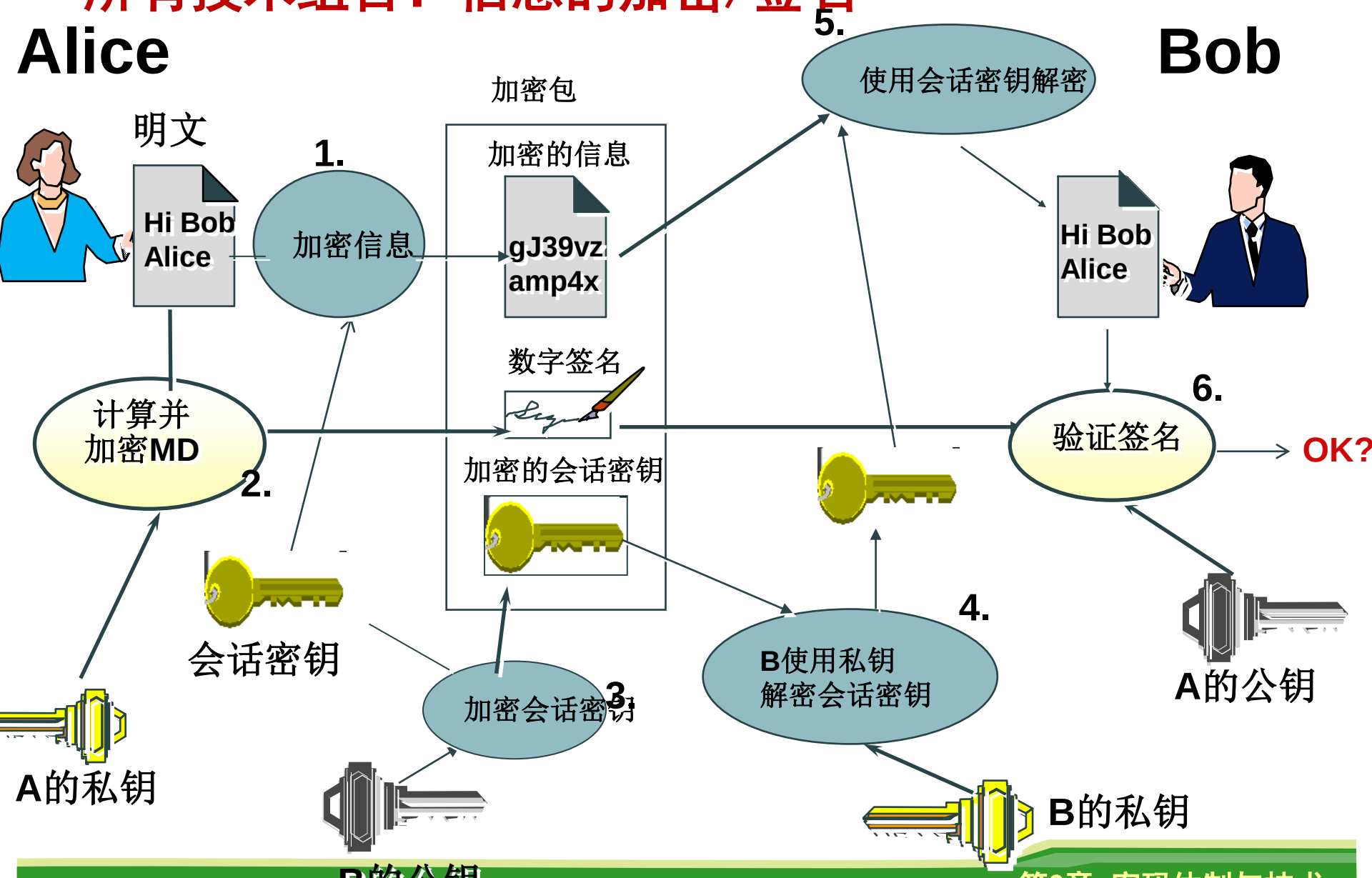
输入	哈希结果（使用 MD5）
Could you please transfer $100 from my checking account to the account of Mr.Smith?	D55f1123532d43a16a08557236615502
Could you please transfer $1000 from my checking account to the account of Mr.Smith?	67b7ba62cae668d8a47bbdf5128a1055



结合数字摘要的数字签名

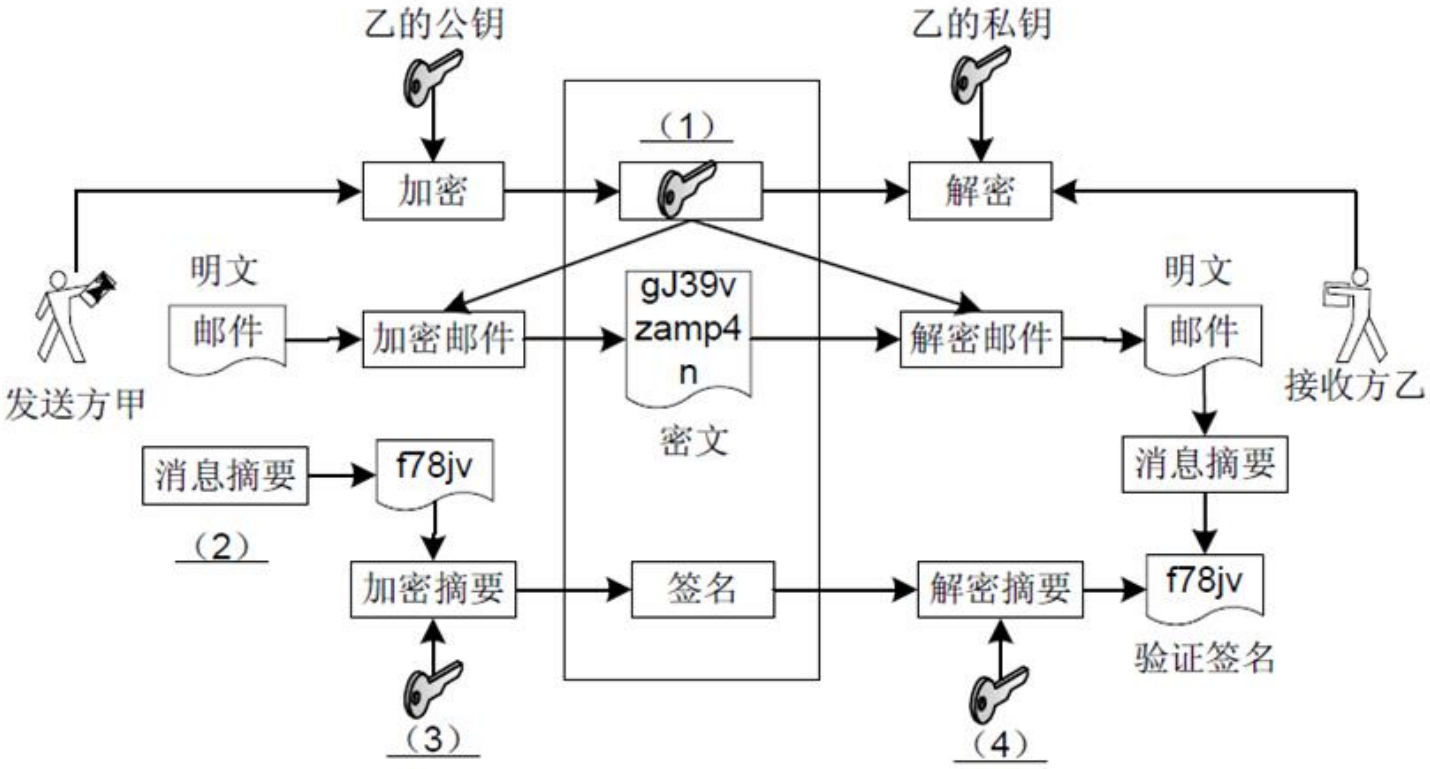


所有技术组合：信息的加密/签名



练习

- 2．某公司的业务员甲与客户乙通过Internet交换商业电子邮件。为保障邮件内容的安全，采用安全电子邮件技术对邮件内容进行加密和数字签名。安全电子邮件技术的实现原理如图所示。



- 1）图中（1）处为_____
- 2）图中（2）处应为_____算法
- 3）图中（3）处为_____
- 4）图中（4）处为_____
- 5）乙收到了地址为甲的含数字签名的邮件，他可以通过验证数字签名来确认哪些信息？

给密码加点“盐”

——密码学中的“盐值 Salt”

第一代密码

- 早期的软件系统或者互联网应用，数据库中设计用户表的时候，大致是这样的结构：

```
mysql> desc User;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| UserName | varchar(50)  | NO   |     |         |       |
| PassWord | varchar(150) | NO   |     |         |       |
+----------+--------------+------+-----+---------+-------+
```

数据存储形式如下：

```
mysql> select * from User;
+----------+----------+
| UserName | PassWord |
+----------+----------+
| lichao   | 123      |
| akasuna  | 456      |
+----------+----------+
```

主要的关键字段就是这么两个，一个是登陆时的用户名，对应的一个密码，而且那个时候的用户名和密码是明文存储的，如果你登陆时密码是 123，那么数据库里存的就是 123。这种设计思路非常简单，但是缺陷也非常明显，数据库一旦泄露，那么所有用户名和密码都会泄露，后果非常严重。

第二代密码

- 为了规避第一代密码设计的缺陷，聪明的人在数据库中不再存储明文密码，转而存储加密后的密码，典型的加密算法是 MD5 和 SHA1，其数据表大致是这样设计的：

```
mysql> desc User;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| UserName | varchar(50)  | NO   |     |         |       |
| PwdHash  | char(32)     | NO   |     |         |       |
+----------+--------------+------+-----+---------+-------+
```

数据存储形式如下：

```
mysql> select * from User;
+----------+----------------------------------+
| UserName | PwdHash                          |
+----------+----------------------------------+
| lichao   | 202cb962ac59075b964b07152d234b70 |
| akasuna  | 250cf8b51c773f3f8dc8b4be867a9a02 |
+----------+----------------------------------+
```

假如你设置的密码是 123，那么数据库中存储的就是 202cb962ac59075b964b07152d234b70 或 40bd001563085fc35165329ea1ff5c5ecbdbbeef。当用户登陆的时候，会把用户输入的密码执行 MD5（或者 SHA1）后再和数据库就行对比，判断用户身份是否合法，这种加密算法称为**散列**。

第三代密码

- 第三代密码设计方法诞生，用户表中多了一个字段：

```
mysql> desc User;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| UserName | varchar(50) | NO   |     |         |       |
| Salt     | char(50)    | NO   |     |         |       |
| PwdHash  | char(32)    | NO   |     |         |       |
+----------+-------------+------+-----+---------+-------+
```

❖数据存储形式如下：

```
mysql> select * from User;
+----------+-----------------------------+----------------------------------+
| UserName | Salt                        | PwdHash                          |
+----------+-----------------------------+----------------------------------+
| lichao   | 1ck12b13k1jmjxrg1h0129h2lj | 6c22ef52be70e11b6f3bcf0f672c96ce |
| akasuna  | 1h029kh2lj11jmjxrg13k1c12b | 7128f587d88d6686974d6ef57c193628 |
+----------+-----------------------------+----------------------------------+
```

Salt 可以是任意字母、数字、或是字母或数字的组合，但必须是随机产生的，每个用户的 Salt 都不一样，用户注册的时候，数据库中存入的不是明文密码，也不是简单的对明文密码进行散列，而是 MD5(明文密码 + Salt)，也就是

```
MD5('123' + '1ck12b13k1jmjxrg1h0129h2lj') = '6c22ef52be70e11b6f3bcf0f672c96ce'
MD5('456' + '1h029kh2lj11jmjxrg13k1c12b') = '7128f587d88d6686974d6ef57c193628'
```

河海大学

Thank You!