

# 第7章 杂凑算法与消息认证

本章主要内容:

- 哈希函数简介
- 哈希函数算法举例(SHA-1)
- 哈希函数的安全性
- 口令的安全性
- 消息认证

# 7.1 杂凑函数

## 引言

在一个开放的通信网络环境中，信息面临的攻击包括窃听、伪造、修改、插入、删除、否认等。

因此，需要提供用来验证消息完整性的一种机制或服务---消息认证。这种服务的主要功能包括：

- 确保收到的消息确实和发送的一样；
- 确保消息的来源真实有效；

注：对称密码体制和公钥密码体制都可提供这种服务，但用于消息认证的最常见的密码技术是基于哈希函数的消息认证码。

# 认证码和检错码

现代密码学中的消息认证码与通信学的消息检错码有密切的联系，并由其演变而来，但其根源和目的是不同的，采用的技术手段有本质的差别。

- 检错码是检测由于通信的缺陷而导致消息发生错误的方法。(客观环境造成的)
- 认证码是用来检查由于恶意或有目的等方式修改消息的技术。(主观人为造成的)

## 7.1.1 哈希 ( Hash ) 函数的概念

也称散列函数、杂凑函数等，是一种单向密码体制，即它是一个从明文到密文的不可逆映射，即只有“加密”过程，不存在“解密”过程。同时，Hash函数可以将“任意”长度的输入经过变换以后得到固定长度的输出。Hash函数的这种单向特征和输出数据长度固定的特征使得它可以生成消息或数据块的“数据指纹” (也称消息摘要、哈希值或散列值)，因此，哈希函数在数据完整性和数字签名等领域有广泛的应用，在现代密码学中起着非常重要作用。

Hash函数是一个将任意长度的消息序列映射为较短的、固定长度的一个值的函数。密码学上的Hash函数能够保障数据的完整性，它通常被用来构造数据的“指纹”(即函数值)，当被检验的数据发生改变的时候，对应的“指纹”信息也将发生变化。这样，即使数据存储在不安全的地方，也可以通过数据的“指纹”信息来检测数据的完整性。

➤生活中，把“指纹”印在纸质文档的底部。

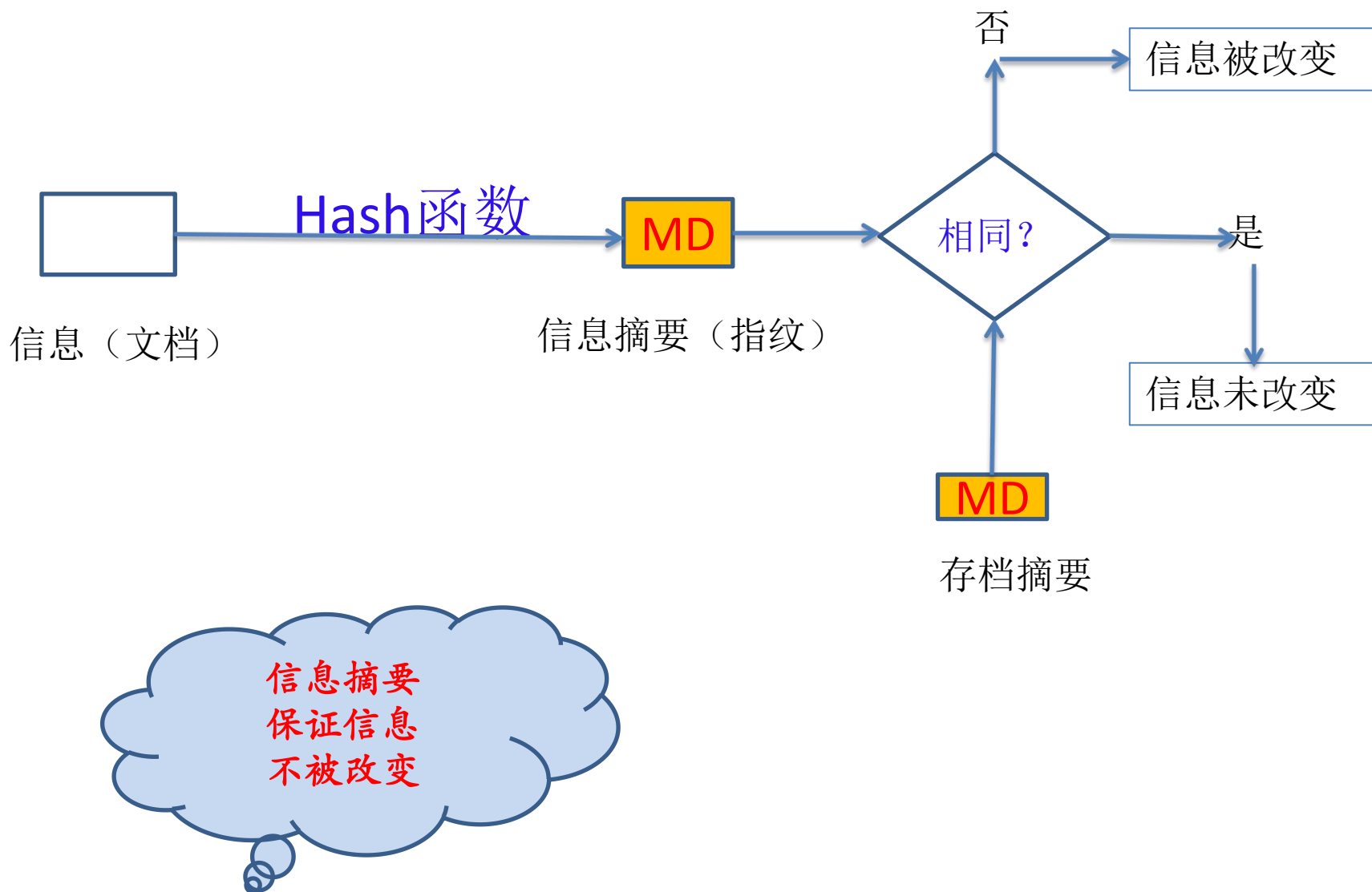
纸质文档上的指纹可以与保存的指纹对照，  
用以保证文件的完整性。

➤信息摘要（Message Digest---MD）

文档与指纹的电子形式



## ➤ 信息完整性验证



# Hash函数的起源

- 消息是任意长度，哈希值是固定长度。
- Hash的概念起源于1965年，Dumey用它来解决symbol table question (符号表问题)。使得数据表的插入、删除、查询操作可以在平均常数时间完成。



# 哈希函数的表示

对不同长度的输入消息，产生固定长度的输出。  
这个固定长度的输出称为原输入消息的“哈希值”  
或“消息摘要”（ Message Digest）。

公式表示形式：

$$h=H(M)$$

M: 任意长度的消息；

H: 哈希（ Hash） 函数或杂凑函数或散列函数；

h: 固定长度的哈希值。

# 哈希算法的性质（特点）

- 输入：消息是任意有限长度。
- 输出：哈希值是固定长度。
- 容易计算：对于任意给定的消息，容易计算其哈希值。（正向容易）
- 单向性（抗原像）：对于给定的哈希值 $h$ ，要找到 $M$ 使得 $H(M)=h$ 在计算上是不可行的。（逆向不可行即原像计算困难性）

# 哈希算法的性质（安全性）

若有两个信息得到同一个摘要，称为冲突或碰撞

- **抗弱碰撞性（抗二次原像）**：对于给定的消息 $M_1$ ，要发现另一个消息 $M_2$ ，满足 $H(M_1)=H(M_2)$ 在**计算上**是不可行的。
- **抗强碰撞性（抗冲突性）**：找任意一对不同的消息 $M_1$ ， $M_2$ ，使 $H(M_1)=H(M_2)$ 在**计算上**是不可行的。
- **随机性**：当一个输入位发生变化时，输出位将发生很大变化。（**雪崩效应**）

# 哈希函数的分类

## ➤ 改动检测码MDC (Manipulation Detection Code)

不带密钥的哈希函数

主要用于消息认证

## ➤ 消息认证码MAC (Message Authentication code)

带密钥的哈希函数

主要用于消息源认证和消息完整性

# Hash函数的作用

1、Hash函数用于保护数据的完整性。

生成消息摘要MD。

2、Hash函数**不能**用于保护数据的机密性。

不同于数据加密。

(不同于AES,DES这类加密, hash函数对明文操作后还叫做明文, 不是密文, 因为hash函数不满足保护数据机密性的要求)

3、Hash函数**不能**用于数据的抵赖性。

不同于数据签名。

# Hash函数的用途

- 消息完整性检测
- Hash链用于口令认证
- 数字签名（速度快；防止消息伪造）
- 消息完整性和消息源认证（MAC）

# Hash函数的用途

## 消息完整性检测

“网站卫士”是一个网络安全软件产品。它的主要功能是通过网络扫描网站的网页，监测网页是否被修改，当发现网页被修改后，系统能够自动报警和恢复。

- 初始化过程

- (1) 对监视网站的文件备份到监控主机上。

- (2) 对每个备份的文件生成一个结构：文件位置、文件的哈希值。

- 监控过程

监控主机对监控网站进行轮回扫描，对扫描的文件进行如下操作：

- (1) 计算文件的哈希值，并与备份的文件哈希值进行比较，如果相同，转(4)步。

- (2) 如果不同，上载备份文件替换网站现有文件，转(4)步。

- (3) 如果备份文件不存在，则删除网站上这个文件，转(4)步。

- (4) 监控程序扫描下一文件。

HashCalc是一款完全免费的文件“指纹”校验工具，通过识别文件的“数字指纹”，可以判断它们是否为同一文件，是否被修改过。

[SlavaSoft HashCalc - Hash, CRC, and HMAC Calculator](#)

查看此网页的中文翻译, 请点击 [翻译此页](#)

HashCalc 2.02 FREEA fast and easy-to-use calculator that allows to compute message digests, checksums and HMACs for files, as well as for text and...

[www.slavasoft.com/hash...](#) - 百度快照

[HashCalc最新版\\_HashCalc官方下载\\_HashCalc2.02汉化版-华军软件园](#)

[↓ 下载地址](#) 大小: 0.31MB 更新时间: 2008-07-11

一个超强的文件校验码计算工具！速度快，支持算法多，直接拖动即可计算，十分方便！还可对字符串直接计算！

[www.onlinedown.net/sof...](#) - 百度快照

[HashCalc是什么软件?\\_百度知道](#)

2个回答 - 回答时间: 2016年6月9日

答案提到: tiger - rmd

最佳答案: 系统完整校验工具fsum与文件校验工具hashcalc2007-11-07 20:10 SlavaSoft出品,小巧强大,向作者致敬。“支持 md2,md4,md5,sha1,sha256,sha384,sha512...

[更多关于hashcalc的问题>>](#)

[百度知道](#) - 百度快照

[HashCalc简体中文版-CSDN下载](#)

2018年12月18日 - HashCalc简体中文版,用来检验文件的一致性,可以更方便的选择文件,也可选择多种加密方式。 HashCalc 2018-12-18 上传大小:167KB 所...

[CSDN下载频道](#) - 百度快照

[几款主流文件校验软件HashCalc、Hasher、WinMD...-360doc个人图书馆](#)



# Hash函数的用途

## 口令认证

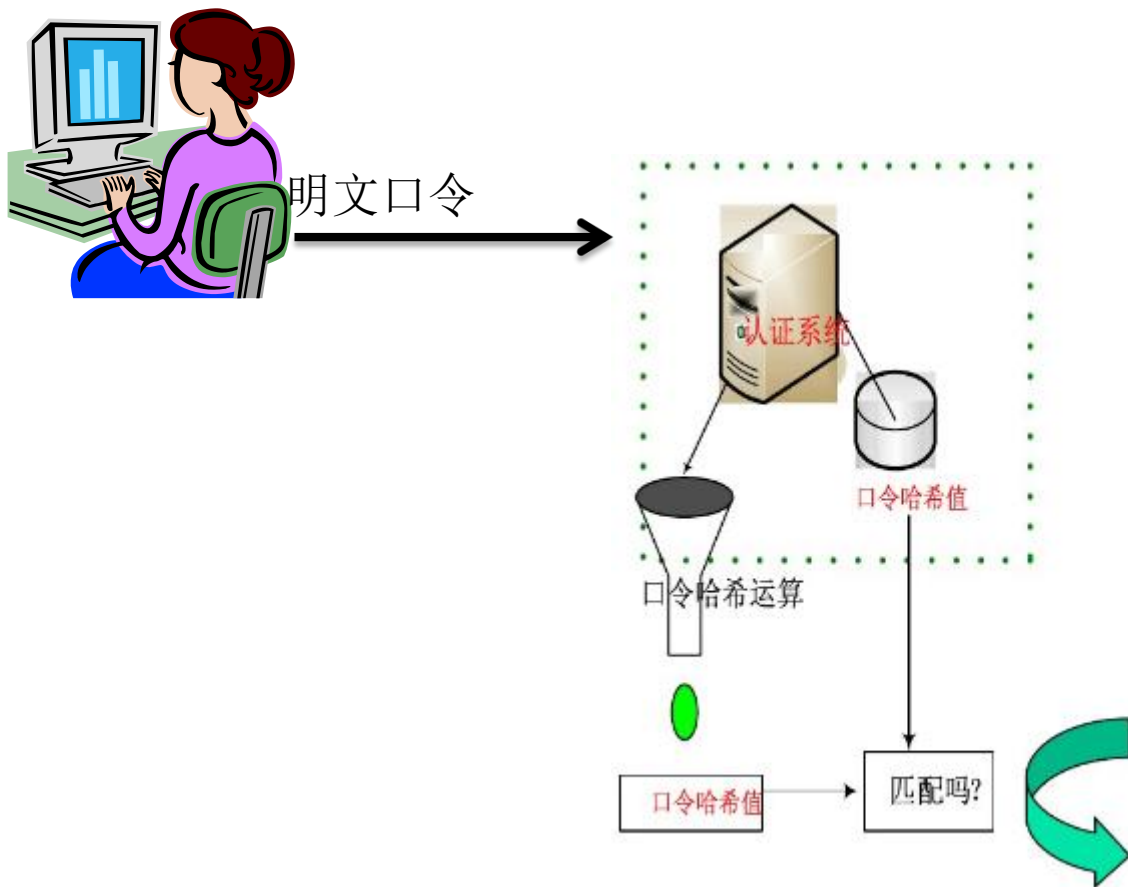
常见的Unix系统口令

以及多数论坛/社区

系统口令都是经MD5

处理后保存其摘要信

息串。

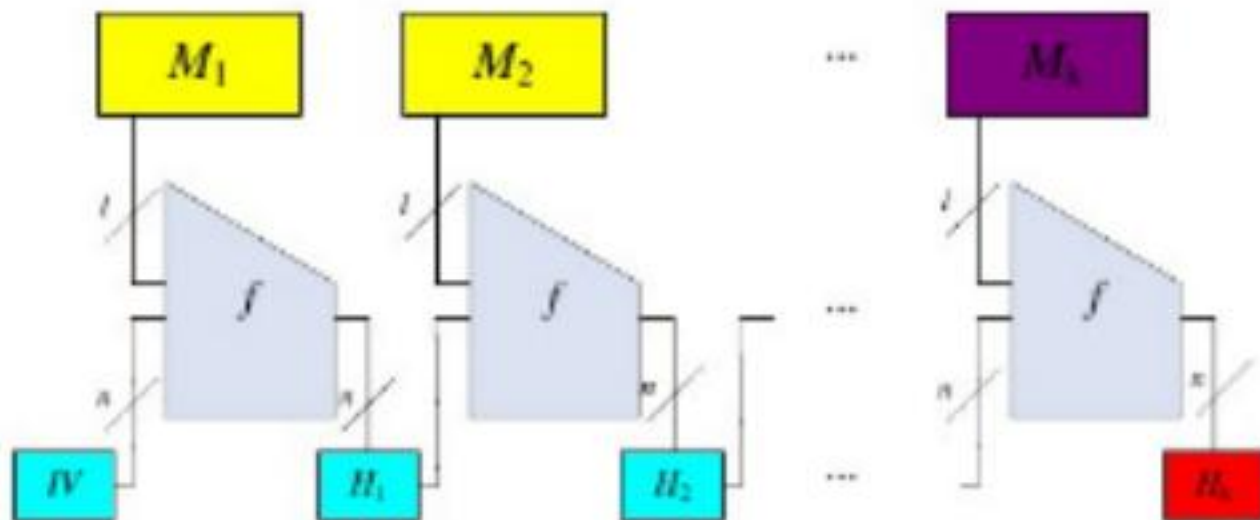


2024年9月28日消息，爱尔兰数据监管机构对meta开出巨额罚单，因其明文存储用户密码。据悉，此次处罚金额高达9100万欧元，折合人民币约7.11亿元。

# Hash函数的发展历史

1978年，Merkle和Damagad设计MD迭代结构

1993年，来学嘉和Messay改进为MD加强结构



# Hash函数的发展历史

散列算法MD族是在上个世纪90年代初由MIT laboratory for computer science和RSA data security Inc的Ron·Rivest设计的，MD代表消息摘要(Message-Digest)，MD2(1989)、MD4(1990)和MD5(1991)都产生一个128位的信息摘要。

## ◆ MD2

1989年开发出MD2算法，在这个算法中，首先对信息进行数据补位，使信息的字节长度是16的倍数。然后，以一个16位的检验和追加到信息末尾。并且根据这个新产生的信息计算出散列值。后来，rogier和chauvaud发现如果忽略了检验和将产生MD2碰撞。

## ◆ MD4

1990年开发出md4算法。Den boer和bosselaers以及其他的人很快的发现了攻击md4版本中第一步和第三步的漏洞，dobbertin向大家演示了如何利用一台普通的PC在几分钟内找到md4完整版本中的碰撞。

## ◆ MD5

1991年，Rivest开发出技术上更为趋近于成熟的MD5算法。Den boer和bosselaers曾发现MD5算法中的伪碰撞（pseudo-collisions），但除此之外就没有其他被发现的分析结果了。

## ◆ RIPEMD-128/160/320

RIPEMD由欧洲财团开发和设计。

# Hash函数的发展历史

**SHA**系列算法是美国国家标准与技术研究院(NIST)根据Rivest设计的MD4和MD5而开发的算法，国家安全当局发布SHA作为美国政府标准，SHA(**S**ecure Hash **A**lgorithm)表示**安全散列算法**。

安全散列算法

- ◆**SHA-0**：正式地称作SHA，这个版本在发行后不久被指出存在弱点。
- ◆**SHA-1**：NIST于1994年发布的，它与MD4和MD5散列算法非常相似，被认为是MD4和MD5的后继者。(160位)
- ◆**SHA-2**：2005年被推荐使用，实际上分为SHA-224、SHA-256、SHA-384和SHA-512算法。

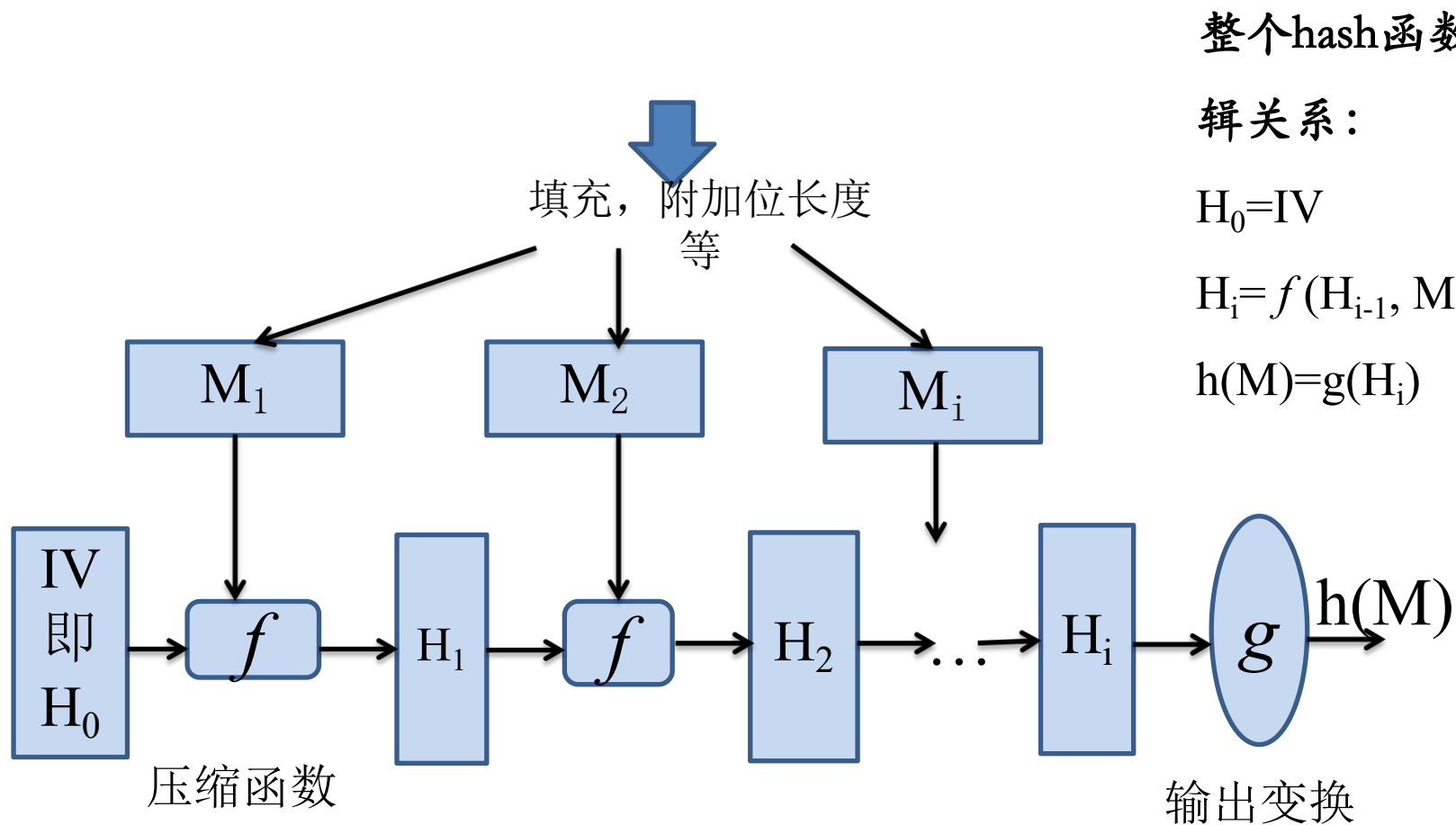
## ◆ HAVAL

1992年Yuliang Zhang 设计了HAVAL函数，它与许多其他散列函数不同。

## ◆ GOST

GOST是一套苏联标准。

## 7.1.2 哈希函数的一般结构



### 7.1 安全杂凑函数的一般结构

# 哈希函数的核心技术

设计“无碰撞”的压缩函数 $f$ ，而攻击者对算法的攻击重点是压缩函数 $f$ 的内部结构，由于哈希函数和分组密码一样是由若干轮迭代处理过程组成，所以对哈希函数的攻击需通过对各轮之间的位模式分析来进行，分析过程常常需要先找出压缩函数 $f$ 的碰撞。由于是压缩函数，其碰撞是不可避免的。因此，在设计哈希函数时就应保证找出其碰撞在计算上是不可行的。

常用的散列函数：

MD5；

SHA系列；



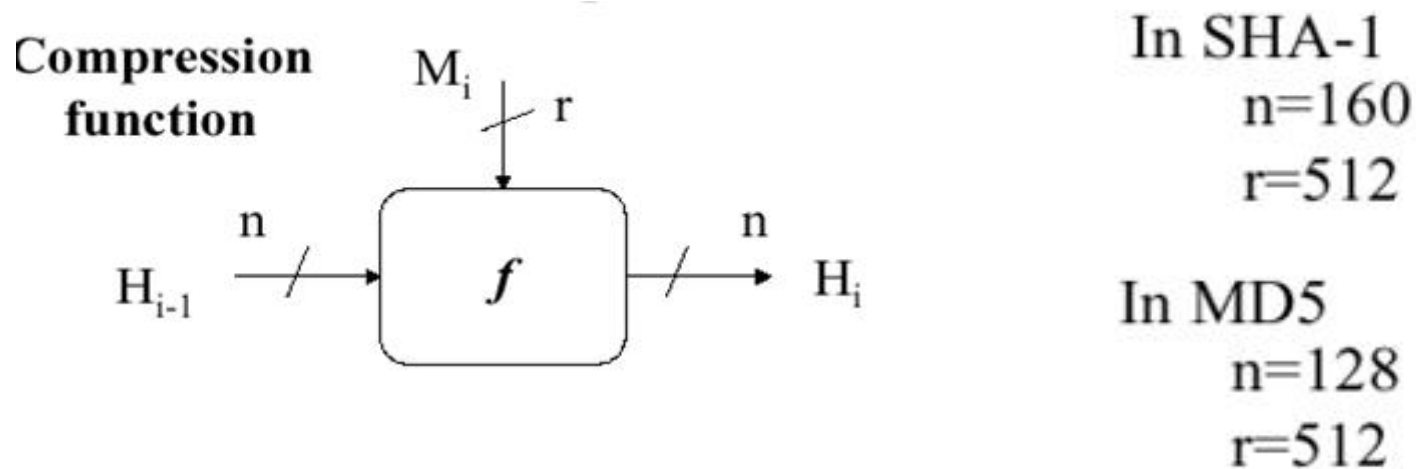


图7.2 压缩函数

$r$ 为分组的比特数， $n$ 为输出的比特数。

通常， $r > n$ ，故称 $f$ 为压缩函数。

## 7.1.3 HASH填充

**填充方法：**在最后一块分组后进行填充，保证填充后的分组最后64比特为整个消息的总长度（以比特为单位），然后在中间进行填充。

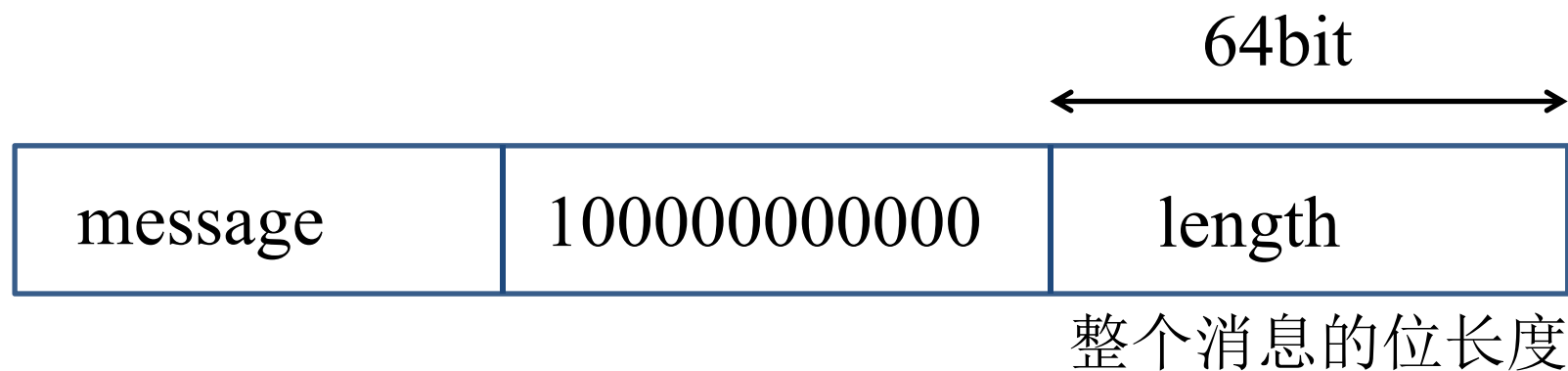


图7.3 填充

**填充方式：**一种是全部填充0；另一种是填充比特的最高位为1，其余为0.

# 7.1.4 hash函数的应用

## 1. 数字签名

- 摘要是不是明文是否完整的“指纹”。
- 由于消息摘要比消息本身小得多，因此对消息摘要进行数字签名在处理上要比直接对消息签名高效得多，所以，数字签名通常都是对消息摘要进行操作。

优点：签名短，计算快，容易管理。

## 2. 生成程序或文档的“数字指纹”

Hash函数可以用来保证数据的完整性，实现消息认证，防止消息未经授权地修改。

### 3. 用于安全存储口令

基于hash函数生成口令的杂凑值，在系统中保存用户的ID及其password的hash值，而不是口令本身，有助于改善系统的安全性。

# 7.2 Hash算法

## 7.2.1 杂凑算法的设计方法

杂凑算法的设计可分为三大类：

### ➤ 基于模数运算

使用**公开密钥**算法来设计单向杂凑函数。使用CBC模式对消息加密。如密钥丢失，将无法解密。

代表：MASH-1.

特点：速度很慢，不实用。

### ➤ 基于分组加密

使用**对称分组密码**算法来设计单向杂凑函数。使用CBC或CFB模式来产生杂凑值。

代表：MDC-2, MDC-4.

方法描述： $h_0 = IV$   $h_i = E_{Mi}(h_{i-1})$   $h = h_n$

## ➤ 定制的

不基于任何假设和密码体制，而是通过构造复杂的非线性关系达到单向要求，设计单向杂凑函数。如MD2、MD4、MD5、SHA-1、RIPEMD-1等。

## 7.2.1 SHA -1

在众多的Hash算法中，MD5（128位）和SHA（160位）是目前使用最广泛的两个算法。SHA系列包括多个散列算法标准，其中，SHA-1是数字签名标准中要求使用的算法。

字 符	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
最大信息长度	$2^{64}-1$	$2^{64}-1$	$2^{64}-1$	$2^{128}-1$	$2^{128}-1$
分组大小	512	512	512	1024	1024
信息摘要大小	160	224	256	384	512
轮数	80	64	64	80	80
字大小	32	32	32	64	64

必考

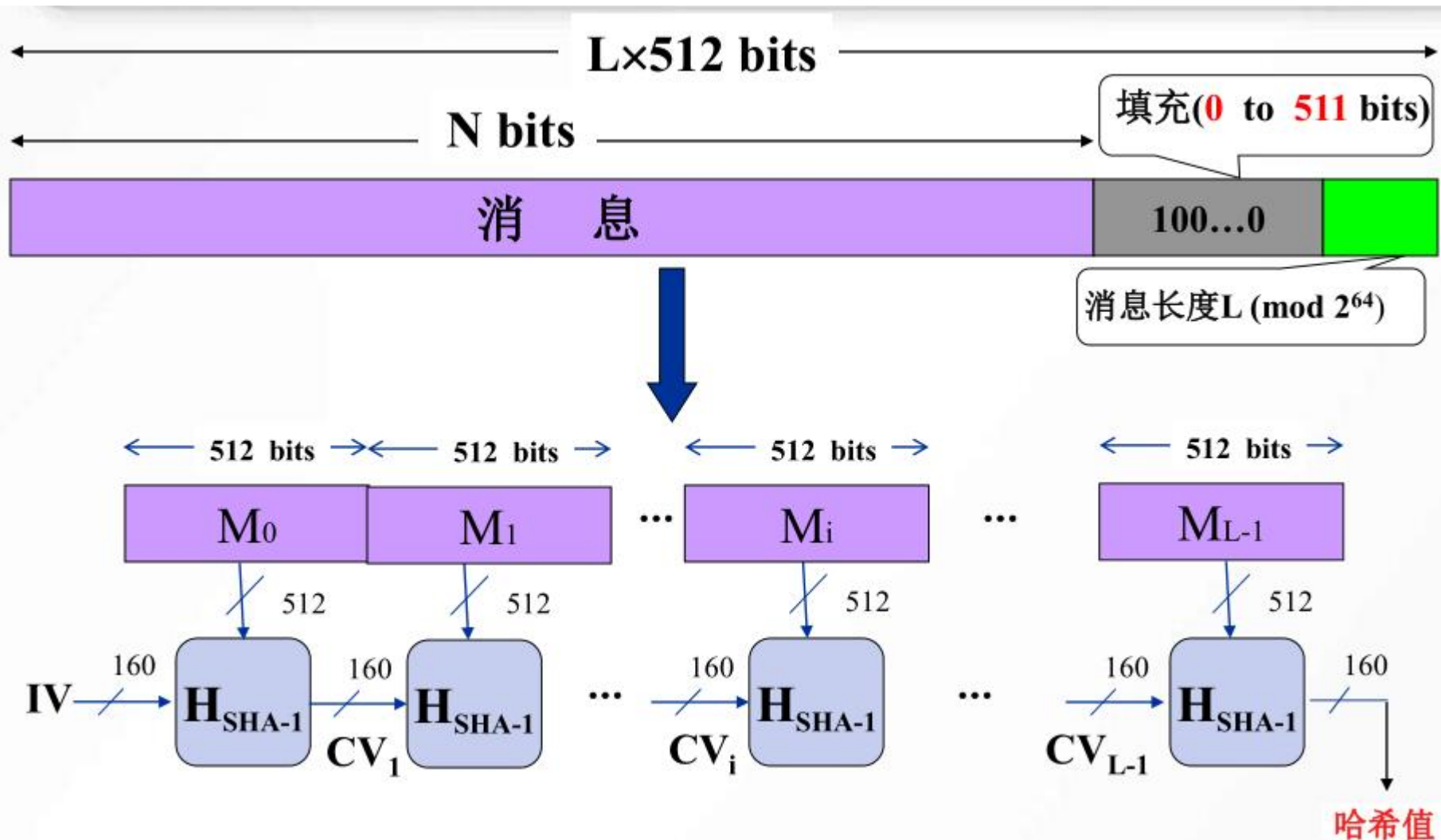
SHA-1接受任何有限长度的输入消息，并产生长度为160比特的Hash值(MD5仅仅生成128位的摘要)，因此抗穷举攻击的能力更强。SHA-1设计时基于和MD4相同原理，它有5个参与运算的32位寄存器字，消息分组和填充方式与MD5相同，主循环也同样是4轮，但每轮进行20次操作，非线性运算、移位和加法运算也与MD5类似，但非线性函数、加法常数和循环左移操作的设计有一些区别。



# 1. SHA-1算法逻辑

- 输入：最大长度为 $2^{64}$ 位的消息；
- 输出：160位消息摘要；
- 处理：输入以512位数据块为单位处理。

# SHA-1 哈希值的生成过程



# SHA-1 算法逻辑

步骤0：将消息摘要转换成位字符串。

以“abc”字符串为例，因为'a'=97, 'b'=98, 'c'=99，所以将其转换为位串后为：01100001 01100010 01100011

步骤1：添加填充位(一个1和若干个0)。使数据位的长度 =  $448 \bmod 512$ （因为有64bit用于描述长度， $448+64=512$ ）

以“abc”为例，其补位过程如下：

初始的信息摘要: 01100001 01100010 01100011

第一步补位: 01100001 01100010 01100011 1

.....

补位最后一位: 01100001 01100010 01100011

10.....0 (后面补了423个0)

将补位操作后的信息摘要转换为十六进制, 如下所示:

61626380 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000 00000000 00000000

00000000 00000000

步骤2: 添加长度。一个64位块，表示原始消息长度。

在这步操作之后，信息报文便是512bit的倍数。通常用一个64位的数据表示原始消息的长度. 在进行附加长度值操作后，其“abc”数据报文即变成如下形式.

一共16个字，1个字是4个字节

61626380	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000018

因为“abc”占3个字节，即24位，换算为十六进制即为0x18。

步骤3: **初始化**消息摘要的缓冲区（即设定IV值）。  
160位，表示为5个32位的寄存器 (A,B,C,D,E)。初始化为：

A = 67452301

B = EFCDAB89

C = 98BADCFE

D = 10325476

E = C3D2E1F0

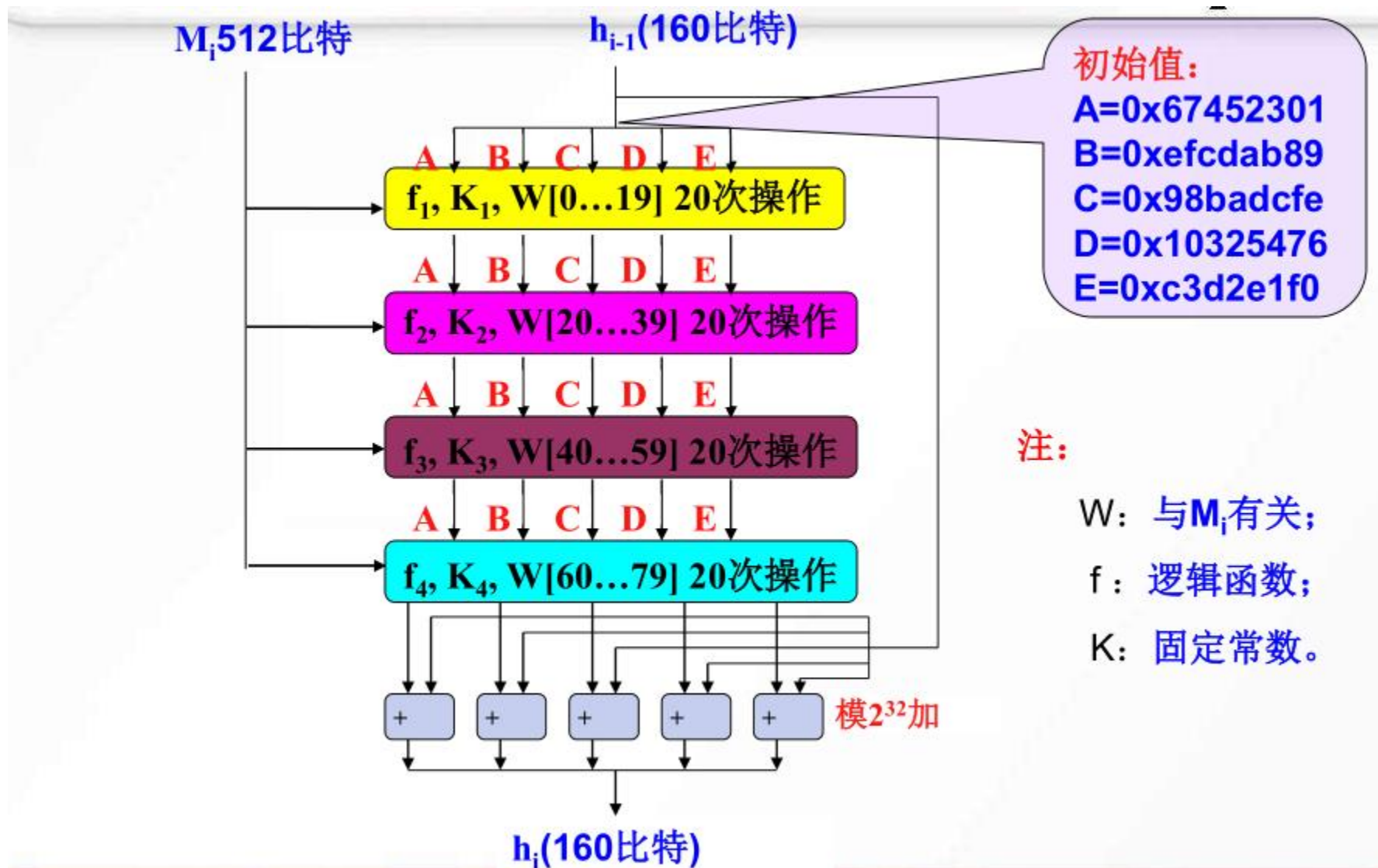


步骤4: 以**512位数据块**为单位处理消息。经过四轮，  
每轮20步。四个基本逻辑函数： $f_1, f_2, f_3, f_4$ 。

注意，仔细看图，经过输出变换后才会变成hi

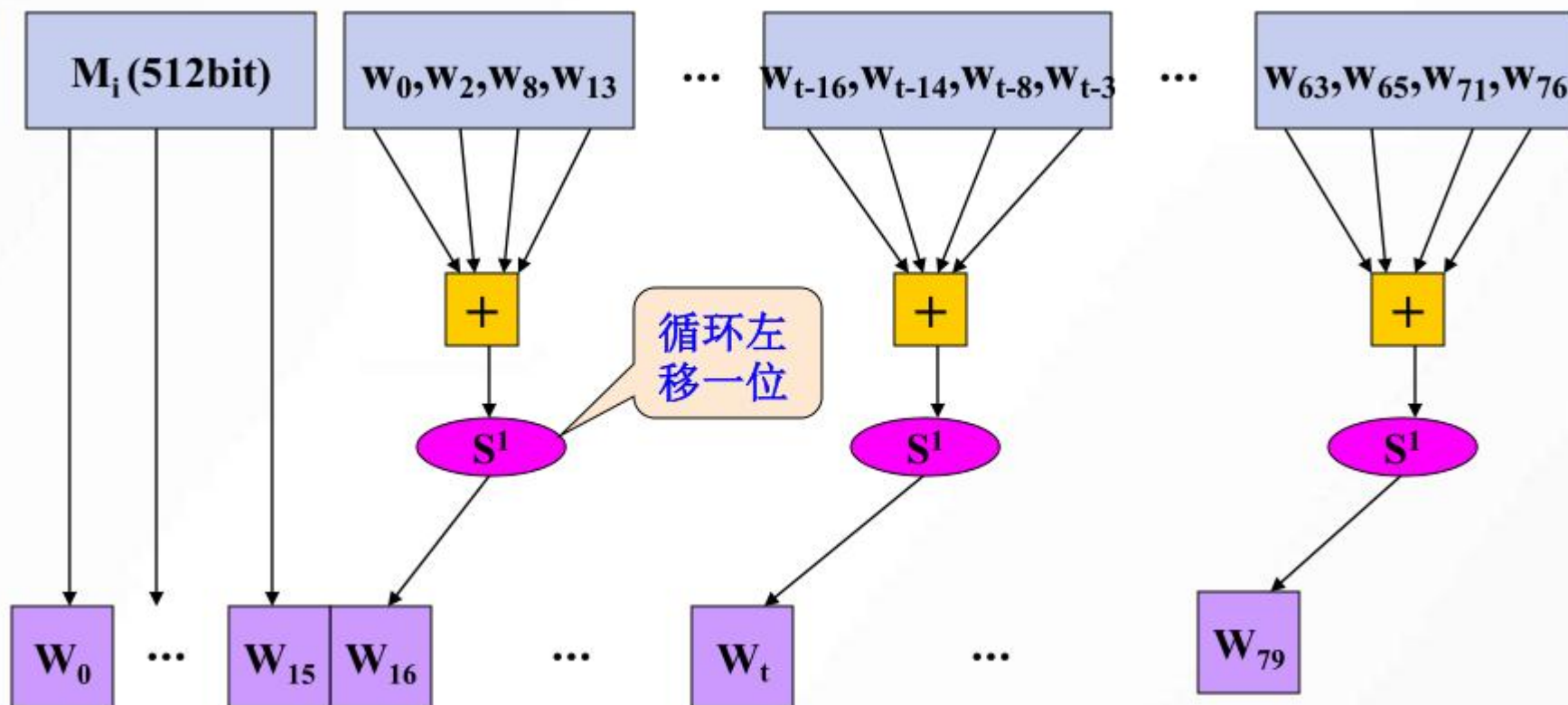
步骤5: 输出。全部L个512位数据块处理完毕后，  
输出160位消息摘要。

# SHA-1对单个 512 位分组的处理过程



# SHA-1 生成字 $W_t$ 的方法

$$W_t = S^1(W_{t-16} + W_{t-14} + W_{t-8} + W_{t-3}) \quad (t=16, 17, \dots, 79)$$



注：前16个32比特字取自这个分组的所有字，即  
 $w_0 = 61626380, \dots, w_{15} = 00000018$ ；  
+表示异或。



# SHA-1 的基本操作

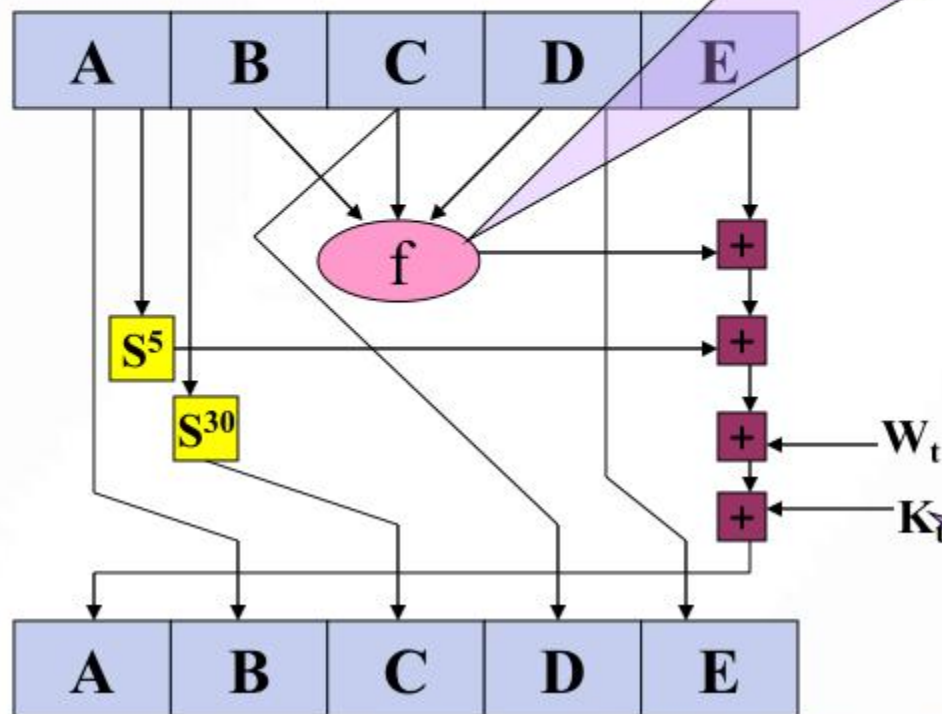
$A = E + f(t, B, C, D) + S^5(A) + W_t + K_t;$

$B = A;$

$C = S^{30}(B);$

$D = C;$

$E = D;$



$f_1(X, Y, Z) = (X \wedge Y) \vee (\sim X \wedge Z)$  (t=0, ..., 19)

$f_2(X, Y, Z) = X \oplus Y \oplus Z$  (t=20, ..., 39)

$f_3(X, Y, Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$  (t=40, ..., 59)

$f_4(X, Y, Z) = X \oplus Y \oplus Z$  (t=60, ..., 79)

$K_t$  的4个取值2、3、5、10的平方根，然后再乘以 $2^{30}$ ，最后取结果的整数部分。

$K_1 = 0x5a827999$  (t=0, ..., 19)

$K_2 = 0x6ed9ebal$  (t=20, ..., 39)

$K_3 = 0x8f1bbcdc$  (t=40, ..., 59)

$K_4 = 0xca62c1d6$  (t=60, ..., 79)

注： $w_t$ 为32比特字，+表示模 $2^{32}$ 加。

## 常数值Kt

步数	16进制
$0 \leq t \leq 19$	Kt=5A827999
$20 \leq t \leq 39$	Kt=6ED9EBA1
$40 \leq t \leq 59$	Kt=8F1BBCDC
$60 \leq t \leq 79$	Kt=CA62C1D6

第四循环（第80步）的输出加到第一循环的输入 $H_i$ 。  
相加是缓存中5个字分别与 $H_{i-1}$ 中对应的5个字模 $2^{32}$ 加，  
即输出变换。

最后输出的是160比特的消息摘要。

# SHA-1总结

$$H_0 = IV$$

$$(ABCDE)_i = f(M_i, H_{i-1}) \quad (i=1, \dots, L)$$

$$H_i = \text{SUM}_{32}(H_{i-1}, (ABCDE)_i) \quad (i=1, \dots, L)$$

$$MD = H_L$$

其中：

$IV$  = ABCDE的初始值;

$ABCDE_i$  = 第*i*轮消息数据块处理最后一轮所得的结果;

$L$  = 数据块的个数;

$\text{SUM}_{32}$  = 模 $2^{32}$ 加;

$MD$  = 最后的消息摘要值.

# 哈希函数的攻击

➤ 攻击者的主要目标不是恢复原始的明文，而是寻找与合法消息**哈希值相同**的**非法消息**，然后用**非法消息**替代**合法消息**进行伪造和欺骗，所以，对哈希函数的攻击也是寻找**碰撞**的过程。

➤ **生日攻击**

# 生日问题

- 问题：假定每个人的生日是等概率的，每年有365天（不考虑闰年）。在 $k$ 个人中至少**有两个人**的生日相同的概率大于0.5，问 $k$ 的最小值是多少？

把每个人的生日看作在[1, 365]中独立均匀分布的随机变量。n个人的生日不重复的概率为:

$$\bar{p}(n) = 1 \cdot \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdots \left(1 - \frac{n-1}{365}\right) = \frac{364}{365} \cdot \frac{363}{365} \cdot \frac{362}{365} \cdots \frac{365-n+1}{365}$$

因为第二个人不能跟第一个人有相同的生日(概率是364/365),第三个人不能跟前两个人生日相同(概率为363/365),依此类推。

$$p(n\text{个人生日各不相同}) = \frac{P_{365}^n}{365^n}$$

P (至少有两个学生的生日在同一天的概率)  
=1-P (n人生日各不相同)

即：至少有两个学生的生日在同一天

$$p(n) = 1 - \frac{365^n}{365^n}$$

$$K=23, p(k)=0.5073$$

$$K=100, P(k)=0.99999997$$



# 生日悖论

生日悖论是指，如果一个房间里有23个或23个以上的人，那么至少有两个人的生日相同的概率要大于50%。这就意味着在一个典型的标准小学班级(30人)中，存在两人生日相同的可能性更高。对于60或者更多的人，这种概率要大于99%。从引起逻辑矛盾的角度来说生日悖论并不是一种悖论，从这个数学事实与一般直觉相抵触的意义上，它才称得上是一个悖论。大多数人会认为，23人中有2人生日相同的概率应该远远小于50%。

# 安全问题的思考

目前安全的哈希值位数是多少？

假设哈希值位数为 $m$ ，其信息集合大小为 $2^m$ ，那么，这集合中多少个消息时使得出现碰撞的概率大于50%？

有意义的攻击：发生碰撞的消息对攻击者有利。

# 两个集合相交问题

已知两个 $k$ 元素集合 $X=\{x_1, x_2, \dots, x_k\}$ ,  $Y=\{y_1, y_2, \dots, y_k\}$ , 其中 $x_i, y_i, (1 \leq i, j \leq k)$ 是 $\{1, 2, \dots, n\}$ 上均匀分布的随机变量。

取定 $x_i$ , 若 $y_i=x_i$ , 则称 $y_i$ 与 $x_i$ 匹配。

固定 $i, j$ ,  $y_i$ 与 $x_i$ 匹配的概率是 $\frac{1}{n}$ 。  $y_i \neq x_i$ 的概率是 $1-\frac{1}{n}$ 。

$Y$ 中所有 $k$ 个随机变量都不等于 $x_i$ 的概率是  $(1-\frac{1}{n})^k$ 。

若 $X, Y$ 中的 $k$ 个随机变量两两互不相同, 则 $X$ 与 $Y$ 中不存在

任何匹配的概率是 $(1-\frac{1}{n})^{k^2}$ . 则 $Y$ 与 $X$ 至少有一个匹配的概率

是 $p=1-(1-\frac{1}{n})^{k^2}$ 。

由数学分析的知识知  $\lim_{n \rightarrow \infty} (1 + \frac{x}{n})^n = e^x$ . 故得

$$p = 1 - (1 - \frac{1}{n})^{k^2} \approx 1 - (e^{-\frac{1}{n}})^{k^2}.$$

若要  $p > 0.5$ , 则  $e^{-\frac{k^2}{n}} < 0.5$ , 可求得  $k$  与  $n$  之间的关系为  $k = \sqrt{(\ln 2) \times n} \approx 0.83\sqrt{n} \approx \sqrt{n}$

# 结论

假设哈希函数H输出长度为 $m$ ，全部可能的输出有 $n=2^m$ 个。哈希函数H接受 $k$ 个随机输入产生集合 $X$ ，接受另外 $k$ 个随机输入产生集合 $Y$ 。根据前面的讨论知，当 $k = \sqrt{n} = 2^{m/2}$ 时， $X$ 与 $Y$ 至少存在一对匹配(即哈希函数产生碰撞)的概率大于0.5。由此看出， $2^{m/2}$ 将决定输出长度为 $m$ 的哈希函数h强碰撞的强度。

如果MD的长度为160比特，采用穷举法进行生日攻击，大约需要 $2^{80}$ 次尝试，才能使至少找到一次碰撞的可能性超过 $1/2$ 。

# 哈希函数的攻击过程

- 签名者对消息 $x$ 的签名其实就是使用私钥对 $H(x)$ 加密得到的签名值  $\text{Sig}_k(h(x))$ 。
- 敌手对消息 $x$ 产生 $2^k$ 个变形的消息(即表达同样的意思消息)，再拟定一个准备替换消息 $x$ 的假消息 $x_1$  (它表示另外一个意思，但对攻击者有利)，产生 $x_1$ 的 $2^k$ 个变形消息。计算所有这些消息的哈希函数值。
- 比较这两个哈希函数值集合，以便发现具有相同哈希值(匹配)的消息。如果没发现，则再产生其他一对有效消息和假消息，直到出现一个匹配为止。
- 用找到匹配的假消息 $x_1$  代替消息 $x$ ，后面仍然附加签名值 $\text{Sig}_k(h(x))$ 一起送给验证者。

# 实例场景

## 1.场景说明

- A要对一个合同文件进行**签名**，然后把合同文件和签名一起发送给接收者。签名的方法：计算文件的Hash值（m位），然后使用**A的私钥**对这个Hash值进行加密。
- 接收者使用A的公钥进行解密，然后比较Hash值，这样他就能确认：接收到的合同文件是A发送的，并且合同文件未被修改过。
- 攻击者B想要伪造一份假合同文件，然后发送给接收者，并使接收者仍然相信：接收到的合同文件是A发送的，并且合同文件未被修改过。



## 2. 攻击方法

B先准备  $2^{m/2}$  个有效合同文件（集合X），每个文件包含与原合同文件相同的意思。

B再准备  $2^{m/2}$  个伪造合同文件（集合Y），每个文件也都是希望的伪造合同的意思。

然后比较集合X和集合Y，找到Hash值相同的两个文件，分别是有效合同和伪造合同。

B成功的概率会大于0.5，如果没有找到匹配的文件，就准备更多的有效文件和伪造文件，直到找到一对匹配的文件。

B把找到的有效合同文件提供A，A看了一下，没什么问题，就做了签名，然后发送给接收者。

在接收者收到消息之前，B截获了这个消息，然后使用伪造合同替换有效合同，再把伪造合同和原签名一起发送给接收者。

因为伪造合同与有效合同的Hash值相同，所以它们产生相同的签名。

这样，即使B不知道A的私钥，他也能成功！

我先写封信：这个(LZ/楼主)是个(老手/专家)，(大家/同志们)一定要(向/像)他学习。

然后再写封信：这个(LZ/楼主)是个(新手/菜鸟)，(大家/回帖的各位)记住(鄙视他/陪他玩玩)。

注意，同样的一句话，根据简单的同义词，我就可以产生 $2^N$ 种不同组合。当N很大的时候，比如说64对同义词，那么：对于这个意思完全相反的两段话，分别得到了 $2^{64}$ 个不同的文件，计算得到两组摘要，每组 $2^{64}$ 。

根据生日攻击方法，这两组摘要中很可能有一组匹配的。

然后我把第一段发给你的某个朋友，让他对这个摘要签名。他肯定会同意。

我再把第二段公开出来，声称你的朋友对它签名了，也就是你的朋友同意了骂你的那一段。

由于这两段的摘要完全相同，那么他无法为自己有效的辩解——第三方很容易识别出，他对那段骂你的摘要进行了签名。

# 王氏攻击 (Wang's Attack)

- 2004年8月，国际密码大会 (Crypto' 2004) 上王小云教授在美密会上宣布破解了MD5算法。国际著名密码学家Eli Biham教授在大会总结报告中写到：“我们该怎么办？MD5被重创了；它即将在应用中被淘汰。SHA-1仍然活着，但也见到了它的末日。”
- 让密码学界更为震惊的是,半年之后在美国召开的国际信息安全RSA大会上,包括三位图灵奖得主Adi Shamir、Ronald L. Rivest和Whitfield Diffie在内的五位密码学家一起宣布了王小云和她的团队对SHA-1的破解结果,这是继MD5之后,她们又一次取得的突破性成果。Shamir宣布时指出：“SHA-1的攻击将引起轩然大波，……无论如何这动摇了我们对电子签名安全的信心”。

- 对MD5和SHA-1的相继破解，证明了电子签名是可以被有效伪造的，设计更为安全的密码HASH函数标准迫在眉睫。
- 2005年起，为了应对SHA-1的攻击，NIST就开始探讨向全球密码学者征集新的HASH函数算法标准的可行性，并于2007年启动了新HASH函数SHA-3五年设计工程。如果设计的算法被采纳为国际标准，那将是密码学家的最高殊荣。国际密码学界都将目光投向了王小云教授，然而，她却毅然放弃了这次机会，全力带领国内专家为我国设计了第一个HASH函数算法标准SM3。

➤ SM3自2010年公布以来，经过国内外密码专家的评估，其安全性得到高度认可。该算法被纳入我国30多个行业规范中，经国家密码管理局审批的含SM3的密码产品达千余款。受SM3保护的智能电网用户6亿多，含SM3的USBKey出货量过10亿张。目前SM3已在高速公路联网ETC中广泛使用，并且在全国教育信息系统、居民健康卡、社保卡、工业控制系统等领域广泛推广使用。该算法还支持国际可信计算组织（TCG）发布的可信平台模块库规范（TPM2.0）。

## 7.2.2 SM3算法

- SM3是中国国家密码管理局颁布的中国商用密码标准算法，它是一类密码杂凑函数，可用于数字签名及验证、消息认证码生成及验证、随机数生成。
- 标准起草人：**王小云**、李峥、于红波、张超、罗鹏、吕述望
- 2012年3月，成为中国商用密码标准 (GM/T0004-2012)
- 2016年8月，成为中国国家密码标准 (GB/T32905-2016)
- 2018年11月22日，含有我国SM3杂凑密码算法的ISO/IEC10118-3:2018《信息安全技术杂凑函数第3部分：专用杂凑函数》最新一版(第4版)由国际标准化组织(ISO)发布，SM3算法正式成为国际标准。

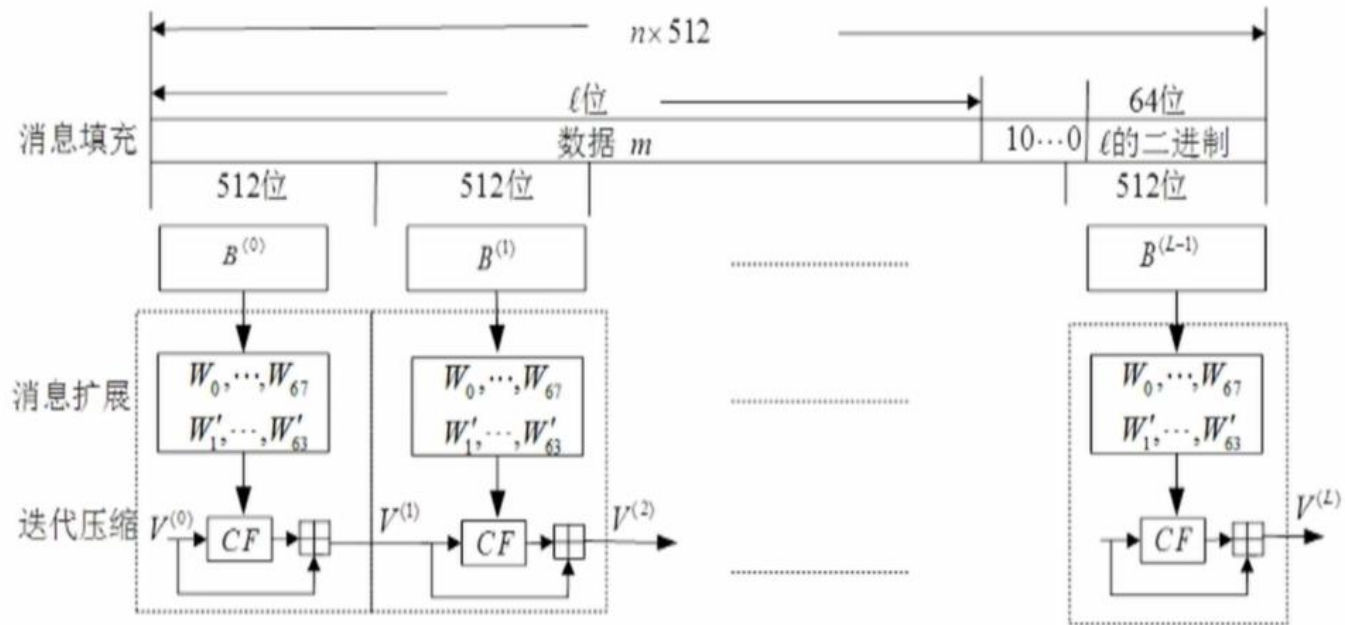


# SM3概述

- SM3适用于商用密码应用中的数字签名和验证，是在SHA-256基础上改进实现的，其安全性和SHA-256相当。SM3和MD5的迭代过程类似，也采用Merkle-Damgard结构。
- 消息分组长度为512位，摘要长度为256位。
- 整个算法的执行过程可以概括成四个步骤：消息填充、消息扩展、迭代压缩、输出结果。

# SM3算法

- 输入：长度为  $l$  ( $l < 2^{64}$ ) 比特的消息  $m$
- 输出：杂凑值长度为  $256$  比特。



## 1. 消息填充

假设消息 的长度为  $l$  比特。首先将比特 “1” 添加到消息的末尾，再添加  $k$  个 “0”， $k$  是满足  $l+1+k \equiv 448 \pmod{512}$  的最小的非负整数。然后再添加一个 64 位比特串，该比特串是长度  $l$  的二进制表示。填充后的消息的比特长度为 512 的倍数。

例如：对消息 01100001 01100010 01100011，其长度  $l=24$ ，经填充得到比特串：

01100001 01100010 01100011 1  $\overbrace{00 \cdots 00}^{423 \text{ 比特}}$   $\overbrace{00 \cdots 011000}^{64 \text{ 比特}}$   
I 的二进制表示

## 2. 迭代压缩

### (1) 迭代过程

将填充后的消息 $m'$ 按512比特进行分组:

$$m' = B^{(0)}B^{(1)}\dots B^{(n-1)} \quad \text{其中 } n = (l+k+65)/512.$$

对 $m'$ 按下列方式迭代:

*FOR*  $i=0$  *TO*  $n-1$

$$V^{(i+1)} = CF(V^{(i)}, B^{(i)})$$

*ENDFOR*

其中,  $CF$ 是压缩函数,  $V^{(0)}$ 为256比特初始值 $IV$ ,  $B^{(i)}$ 为填充后的消息分组, 迭代压缩的结果为 $V^{(n)}$ 。

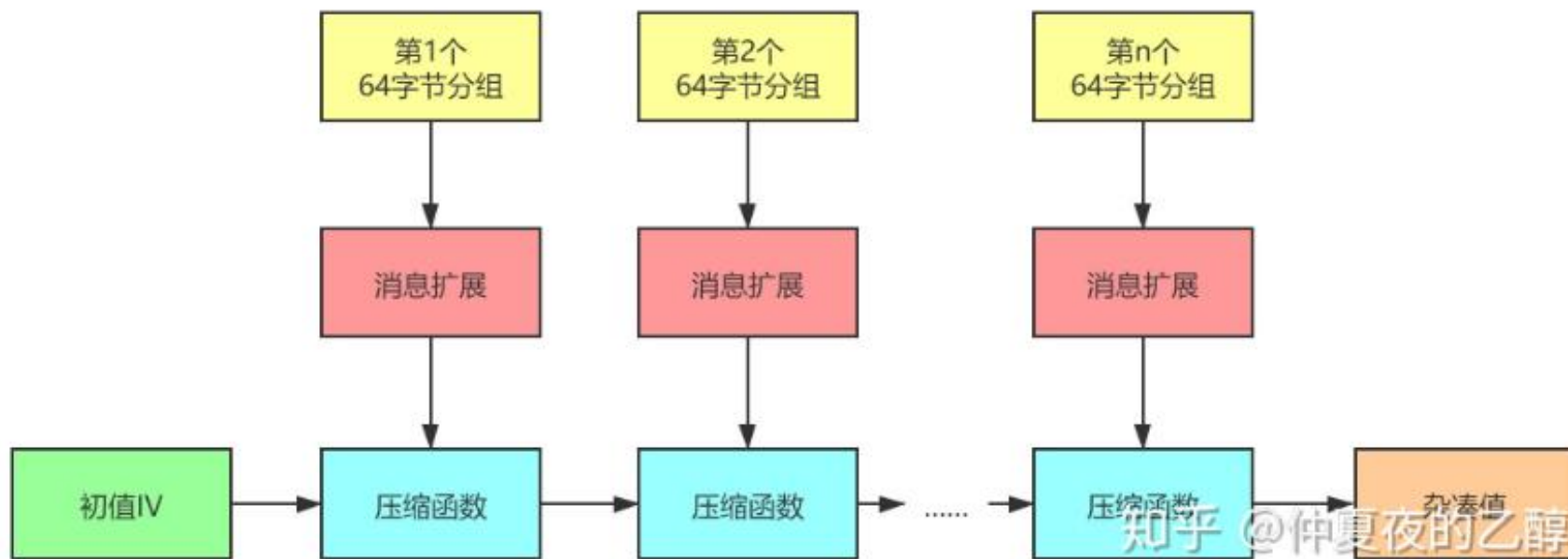


图1. SM3的迭代过程

## (2) 消息扩展

在对消息分组  $B^{(i)}$  进行迭代压缩之前，首先要对其进行消息扩展。进行消息扩展的目的有两个：一是将16个字的分组消息扩展成132个字  $W_0, W_1, \dots, W_{67}, W'_0, W'_0, W'_1, \dots, W'_{63}$ ，供压缩函数CF使用；二是通过消息扩展把原消息位打乱，隐蔽原消息位之间的关系，增强了Hash函数的安全性。

消息扩展先将一个512位数据划分为16个消息字，并且作为生成132个消息字的前16个，再用这16个消息字递推生成剩余的116个消息字。

消息扩展的步骤如下：

a) 将消息分组 $B^{(i)}$ 划分为16个字 $W_0, W_1, \dots,$

~~W~~<sub>15</sub> FOR  $j = 16$  TO 67

$$W_j \leftarrow P_1(W_{j-16} \oplus W_{j-9} \oplus (W_{j-3} \lll 15)) \oplus (W_{j-13} \lll 7) \oplus W_{j-6}$$

ENDFOR

置换函数

c) FOR  $j = 0$  TO 63

$$W'_j \leftarrow W_j \oplus W_{j+4}$$

ENDFOR

消息分组 $B^{(i)}$ 经消息扩展后就可以进行迭代压缩了。

### (3) 压缩函数

令A, B, C,  
D, E, F, G, H  
为字寄存器,  
SS1, SS2, TT1,  
TT2为中间变量,  
压缩函数 $V^{i+1} =$   
 $CF(V^{(i)}, B^{(i)})$ ,

$0 \leq i \leq n-1$ 。计  
算过程描述如下:

$ABCDEFGH \leftarrow V^{(i)}$

赋初值

FOR  $j=0$  TO 63

$SS1 \leftarrow ((A \lll 12) + E + (T_j \lll j)) \lll 7$

$SS2 \leftarrow SS1 \oplus (A \lll 12)$

$TT1 \leftarrow FF_j(A, B, C) + D + SS2 + W'_j$

$TT2 \leftarrow GG_j(E, F, G) + H + SS1 + W_j$

$D \leftarrow C$

$C \leftarrow B \lll 9$

$B \leftarrow A$

$A \leftarrow TT1$

$H \leftarrow G$

$G \leftarrow F \lll 19$

$F \leftarrow E$

$E \leftarrow P_0(TT2)$

ENDFOR

$V^{(i+1)} \leftarrow ABCDEFGH \oplus V^{(i)}$

逻辑函数

常量

置换函数



# 关于压缩函数的说明

- 压缩函数中的+为 $\text{mod } 2^{32}$ 运算，字的存储为大端格式。大端格式规定左边为高有效位，右边位低有效位。数的高位字节放在存储器的低地址，数的低位字节放在存储器的高地址。
- 压缩函数由线性和非线性结构构成，在压缩函数中进行了64轮迭代。
- 压缩函数是Hash函数安全的关键。它的一个作用是数据压缩，即把每一个512位的消息分组压缩成256位；第二个作用是提供安全性。根据香农的密码设计理论，压缩函数必须有混淆和扩散的作用。

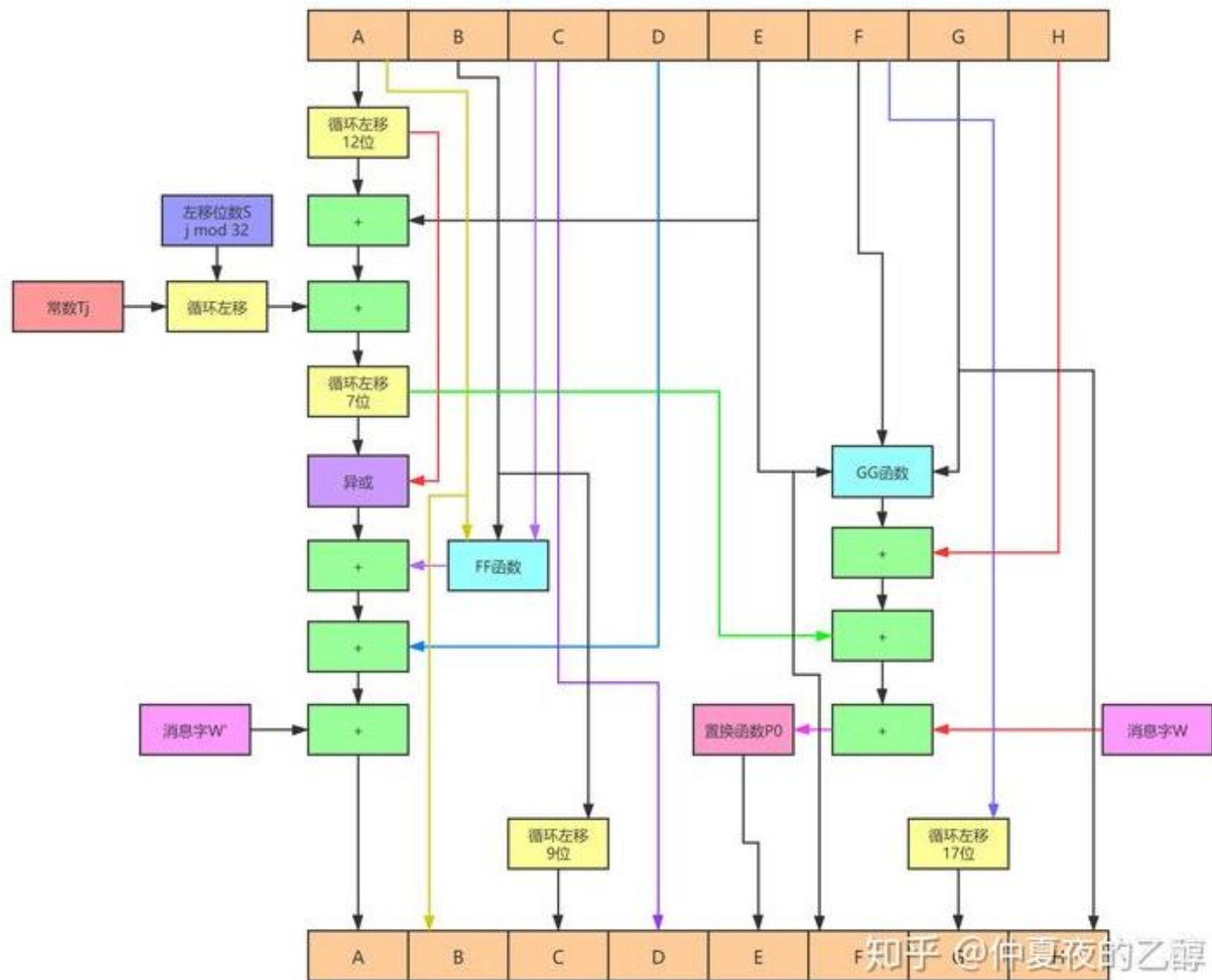


图1. SM3的压缩函数

## (4) 输出结果

$$ABCDEFGH \leftarrow V^{(n)}$$

输出256比特的杂凑值  $y = ABCDEFGH$

# 涉及到的常数与函数

## ➤初始向量IV

SM3的初始值IV 为256bit的数据块，即8个32bit的字。

$IV=7380166f\ 4914b2b9\ 172442d7\ da8a0600\ a96f30bc\ 163138aa\ e38dee4d\ b0fb0e4e$

## ➤常量

$$T_j = \begin{cases} 79cc4519 & 0 \leq j \leq 15 \\ 7a879d8a & 16 \leq j \leq 63 \end{cases}$$

## ➤逻辑函数

SM3算法总共64轮，使用2个逻辑函数，以32bit字进行操作，逻辑函数定义如下：

$$FF_j(X, Y, Z) = \begin{cases} X \oplus Y \oplus Z & 0 \leq j \leq 15 \\ (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z) & 16 \leq j \leq 63 \end{cases}$$

$$GG_j(X, Y, Z) = \begin{cases} X \oplus Y \oplus Z & 0 \leq j \leq 15 \\ (X \wedge Y) \vee (\neg X \wedge Z) & 16 \leq j \leq 63 \end{cases}$$

式中X,Y,Z 为字。

逻辑函数是**非线性**函数，经过循环迭代后提供**混淆**作用。

## ➤ 置换函数

$$P_0(X) = X \oplus (X \lll 9) \oplus (X \lll 17)$$

$$P_1(X) = X \oplus (X \lll 15) \oplus (X \lll 23)$$

式中 $X$ 为字。

置换函数是线性函数，经过循环迭代后提供扩散作用。

# 练习题

## 1. 教材P318-321。

## 7.2.3 哈希函数的应用

### ➤ 口令的安全性

譬如可信计算、文件病毒的检验、网页防篡改、黑白名单等许多应用。

### ➤ 文件的完整性

### ➤ 密码协议的应用

譬如零知识证明、比特承诺、不经意传送、电子商务的安全协议等许多应用。

### ➤ 消息认证

### ➤ 数字签名

譬如随机数的生成、密钥更新等许多应用。

### ➤ 其它应用



# 可信计算

- “可信计算”可信计算 (Trusted Computing) 是在计算和通信系统中广泛使用基于硬件安全模块支持下的可信计算平台，以提高系统整体的安全性。
- 可以从几个方面来理解：用户的身份认证，这是对使用者的信任；平台软硬件配置的正确性，这体现了使用者对平台运行环境的信任；应用程序的完整性和合法性，体现了应用程序运行的可信；平台之间的可验证性，指网络环境下平台之间的相互信任。

# 比特承诺

**比特承诺**(Bit Commitment, BC)是密码学中的重要基础协议，其概念最早由1995年图灵奖得主Blum提出。**比特承诺**方案可用于构建零知识证明、可验证秘密分享、硬币投掷等协议，是信息安全领域研究的热点。

## 基本思想：

发送者 Alice 向接收者 Bob 承诺一个比特 $b$  (如果是多个比特，即比特串 $t$ ，则称为比特串承诺)，要求：

- 在第1阶段即**承诺阶段** Alice 向 Bob 承诺这个比特 $b$ ，但是 Bob 无法知道 $b$ 的信息；
- 在第2阶段即**揭示阶段** Alice 向 Bob 证实她在第1阶段承诺的确实是 $b$ ，但是 Alice 无法欺骗 Bob(即不能在第2阶段篡改 $b$ 的值)。

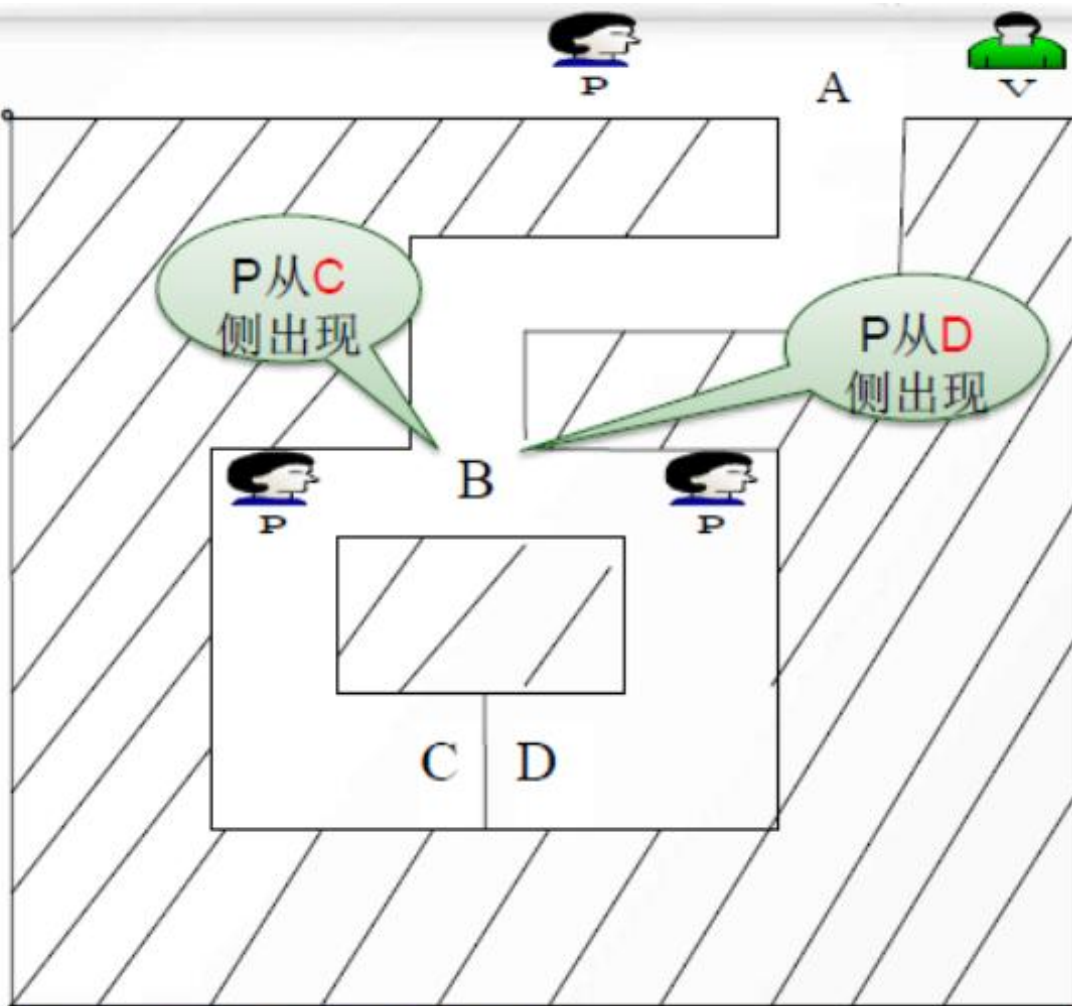
# 零知识证明

零知识证明指的是证明者能够在不向验证者提供任何有用的信息的情况下，使验证者相信某个论断是正确的。

**混淆电路**是设计零知识证明协议的一个工具。混淆电路是我国唯一图灵奖得主姚期智院士在1982年首次提出的，它允许双方在不泄露隐私的前提下，使用各自的隐私计算一个公开的结果。例如，两个百万富翁不想让对方知道自己有多少资产，但是都希望知道谁的钱比较多。两个企业，一个企业有机器学习的模型，另一个企业有数据，他们不希望让对方知道自己的模型和数据，但是希望共同计算预测结果。

# 经典的形象例子

- 验证者V开始停留在位置A。
- 证明者P一直走到迷宫的深处，随机选择到位置C或位置D。
- V看不到P后，走到位置B，然后命令P从某个出口返回B。
- P服从V的命令，要么原路返回至位置B，要么使用秘密口令打开门后到达位置B。
- P和V重复上述步骤n次。



# 不经意传输协议

- 不经意传输协议，是一种可保护隐私的双方通信协议，能使通信双方以一种选择模糊化的方式传送消息。不经意传输协议是密码学的一个基本协议，它使得服务接收方以不经意的方​​式得到服务发送方输入的某些消息，这样就可以保护接受者的隐私不被发送者所知道。
- 设A有一个秘密，想以 $1/2$ 的概率传送给B，即B有50%的机会收到这个秘密，另外50%的机会什么也没有收到，协议执行完后，B知道自己是否收到了这个秘密，但是A却不知道B是否收到了这个秘密。这种协议就称为不经意传输协议。

例如A是机密的出售者，A列举了很多问题，意欲出售某个问题的答案，B想买其中一个问题的答案，但又不想让A知道自己买的是哪个问题的答案。

# 基于口令的身份认证

由于口令**易实现**、**成本低**、**使用方便**等特点，成为当前最常用的认证方式，尤其在安全性要求不是很高的应用场合。

对于大部分系统而言，初始化时，系统管理员给每个用户指定一个缺省(临时)口令，认证数据库中保存**缺省口令及其哈希值**。然后，管理员通过某些方式，如邮件或者电话，告诉用户他的缺省口令。

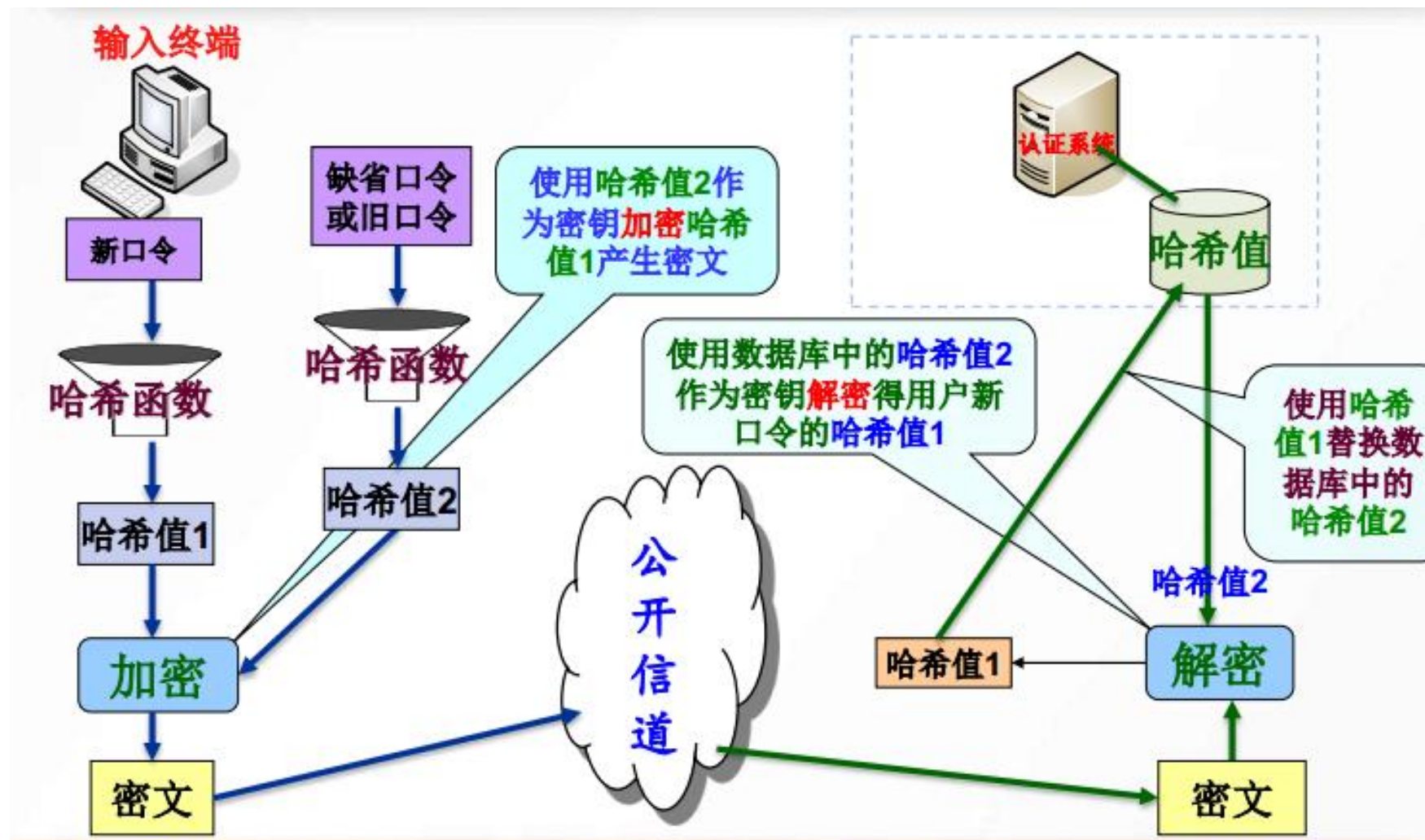
- **缺省口令的更新**
- **远程口令的认证**

哈希函数的好处：

- 口令信息实现安全传输
- 管理员不知道用户的口令

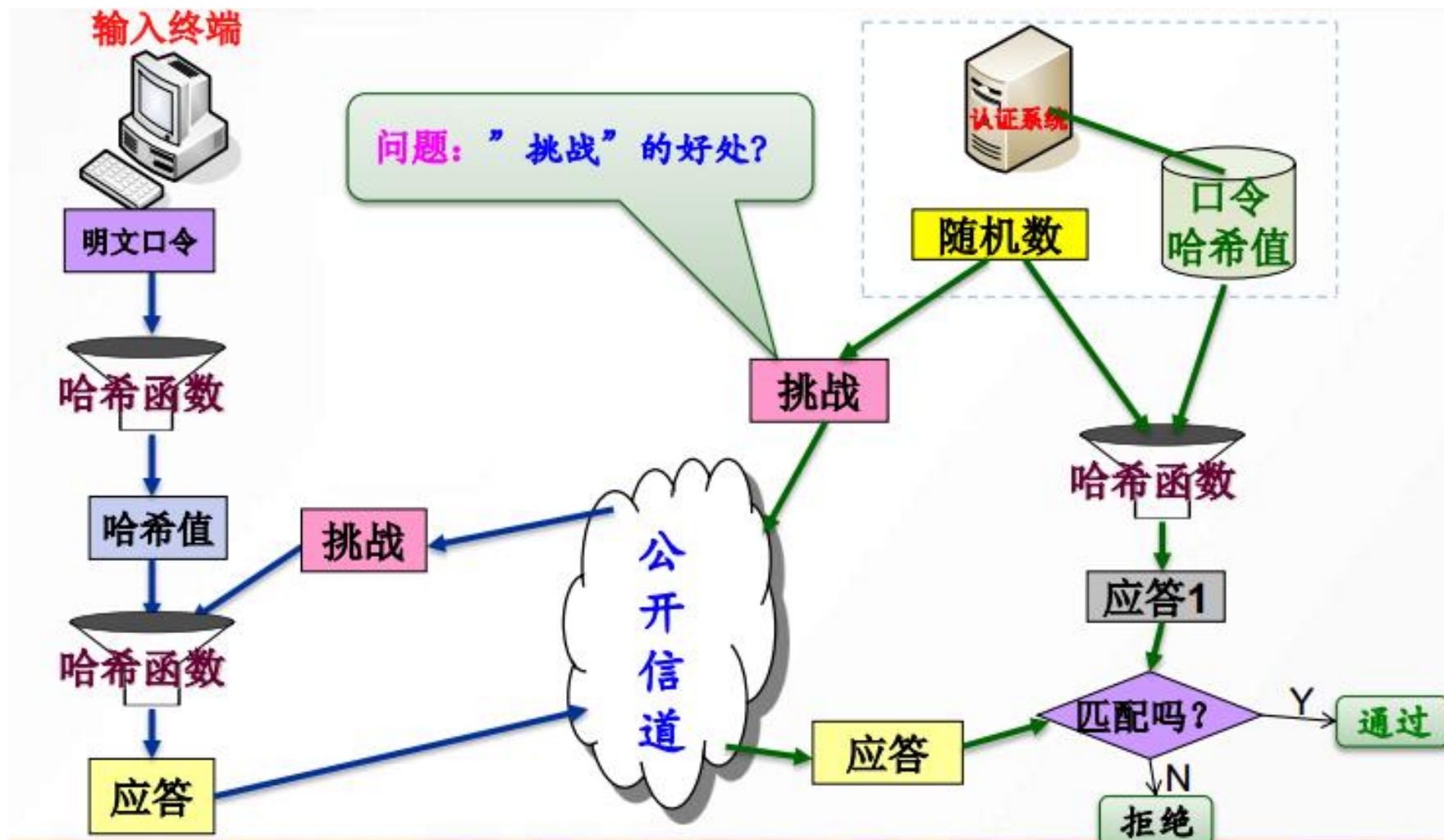


# 新口令设置



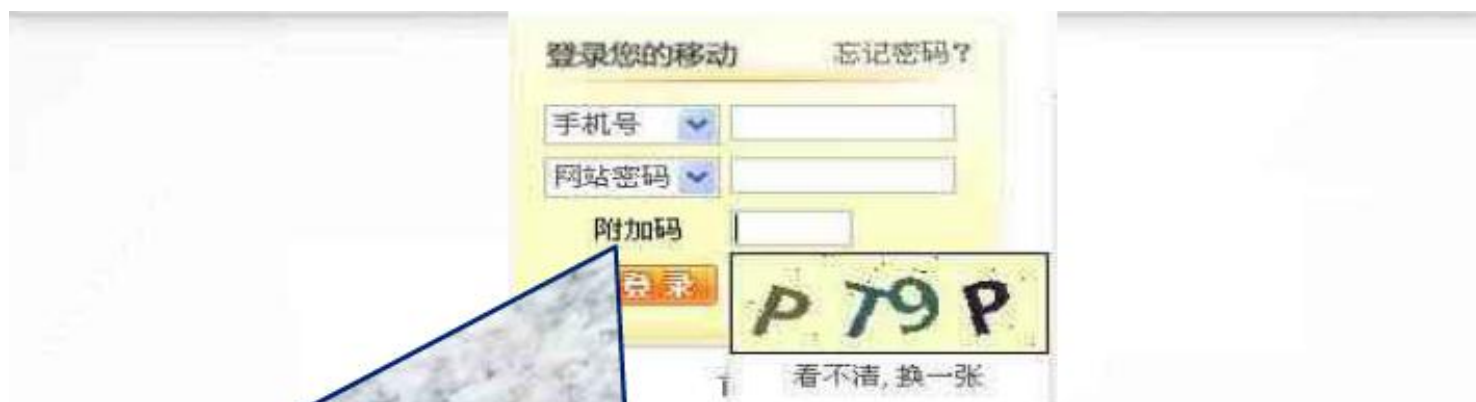


# 远程口令认证



挑战是由验证者发送的随时间变化的值；应答是把函数应用于挑战的结果。

# 附加码（验证码）的介绍

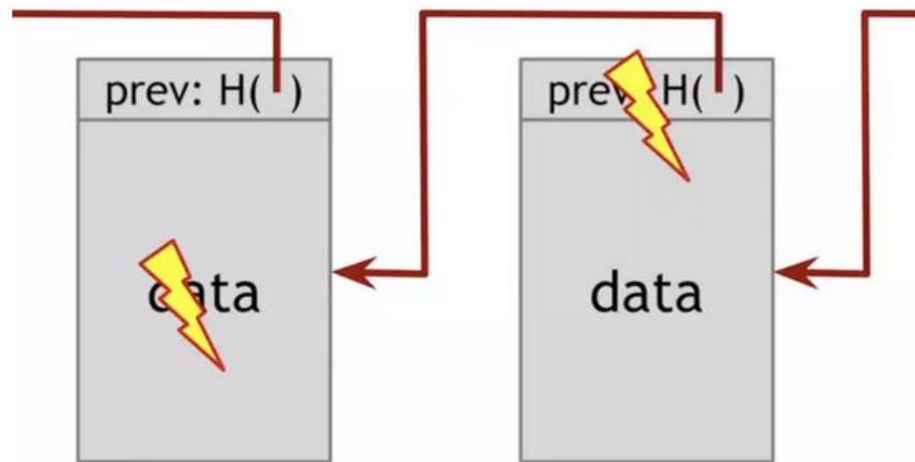


附加码(验证码)其实就是一串随机产生的数字或符号，使用时生成一幅图片，在图片里加一些干扰像素，现由用户肉眼识别，然后提交给网站验证，验证成功后才能使用某项功能。附加码一般是防止攻击者利用程序自动批量尝试，对特定的注册用户用特定的程序进行暴力破解，另外，如果攻击者不断地进行登录，将可能导致服务中断或不能有效地提供服务。因为附加码是一个混合了数字或符号的图片，人眼看起来都费劲，程序识别起来更加困难。

# 区块链（blockchain）

区块链是什么？它是一种去中心化的分布式数据库。

区块链由一个个区块（block）组成。区块很像数据库的记录，每次写入数据，就是创建一个区块。



每个区块包含两个部分。

- 区块头 (Head) : 记录当前区块的元信息
- 区块体 (Body) : 实际数据

区块链的 Hash 长度是256位。

两个重要的结论:

结论1: 每个区块的 Hash 都是不一样的, 可以通过 Hash 标识区块。

结论2: 如果区块的内容变了, 它的 Hash 一定会改变。

区块头



区块体



区块与 Hash 是一一对应的，每个区块的 Hash 都是针对“区块头”（Head）计算的。

$\text{Hash} = \text{SHA256}(\text{区块头})$

区块头包含很多内容，其中有当前区块体的 Hash

（注意是“区块体”的 Hash，而不是整个区块），还有上一个区块的 Hash。这意味着，如果当前区块的内容变了，或者上一个区块的 Hash 变了，一定会引起当前区块的 Hash 改变。

如果有人修改了一个区块，该区块的 Hash 就变了。为了让后面的区块还能连到它，该人必须同时修改后面所有的区块，否则被改掉的区块就脱离区块链了。由于Hash 的计算很耗时，同时修改多个区块几乎不可能发生，除非有人掌握了全网51%以上的计算能力。

正是通过这种联动机制，区块链保证了自身的可靠性，数据一旦写入，就无法被篡改。这就像历史一样，发生了就是发生了，从此再无法改变。



## 7.3 消息认证

网络系统安全一般要考虑两个方面：一方面，加密保护传送的信息，使其可以抵抗**被动**攻击；另一方面，就是要能防止对手对系统进行**主动**攻击，如伪造、篡改信息等。认证是对抗主动攻击的主要手段，它对于开放的网络中的各种信息系统的安全性有重要作用。认证分为**实体认证（身份认证）**和**消息认证**。

数字签名  
技术实现

### 消息认证的目的：

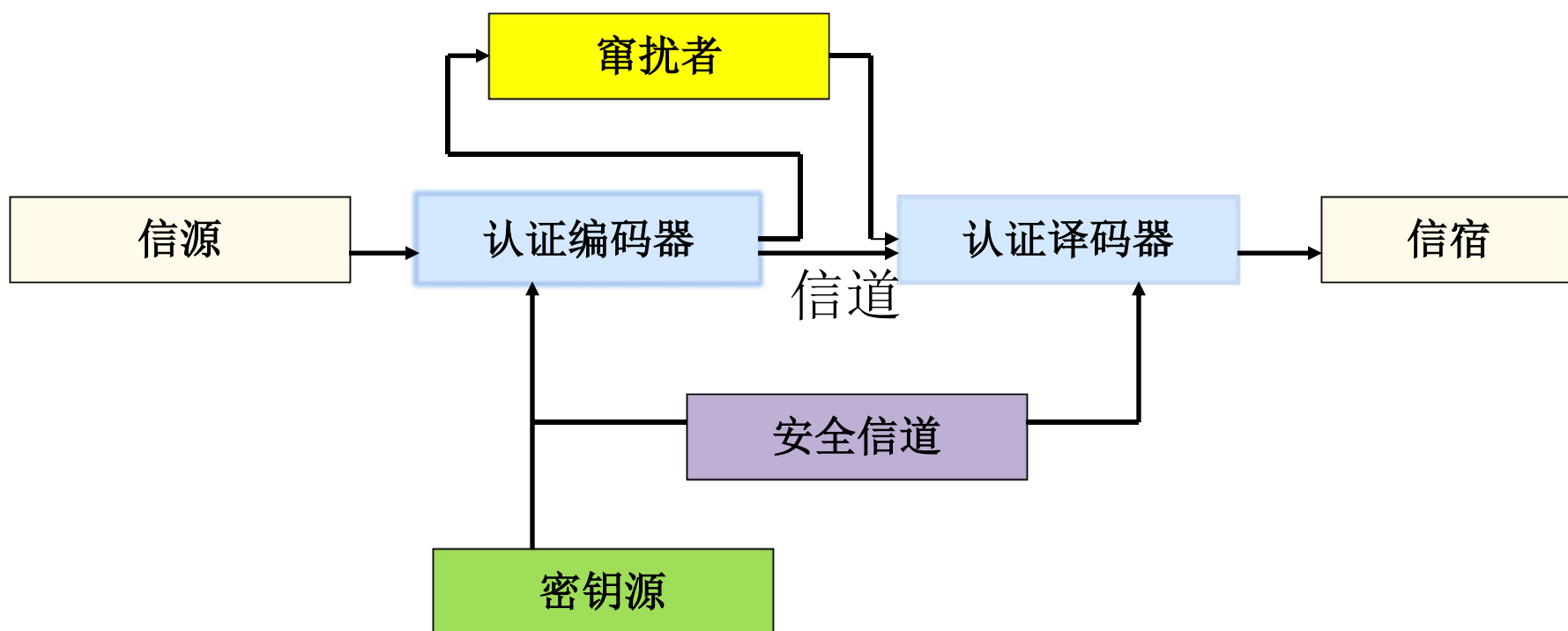
- 验证信息的来源是**真实**的，而不是冒充的，此为消息源认证。
- 验证消息的**完整性**，即验证信息在传送或存储过程中是否被修改。

**实体认证（身份认证）** 要求声称方要向认证方证实自己的身份，一般有三种证实的方法可供选用：

- ①**知道什么**：该方法利用只有声称方知道的某个秘密，该秘密能被认证方核查，如口令、PIN（Personal Identification Number）、对称密钥或私钥等；
- ②**拥有什么**：该方法利用存在能证实声称方身份的某种事物，如护照、驾照、身份证、信用卡、智能卡等；
- ③**内在属性**：该方法利用声称方的内在特征或属性，如手写签名、指纹、声音、面部特征、视网膜、笔迹等。



# 一个纯认证系统的模型



系统中：

- 发送者：通过一个公开的无扰信道将消息送给接收者；
- 接收者：不仅想收到消息本身，而且还要验证消息是否来自合法的发送者及消息是否经过篡改；
- 窜扰者：不仅要截收和破译信道中传送的密报，而且可伪造密文送给接收者进行欺诈。

实际认证系统还要防止收方、发方之间的相互欺诈。

- 认证系统的功能层次
  - 底层的认证函数：产生一个用来认证消息的认证标识；
  - 上层的认证协议：基于认证标识提供了一种能使接收方验证消息真实性的机制；

# 认证函数分类

## ➤ 消息加密函数(Message encryption)

用完整信息的密文作为对信息的认证的认证标识。

## ➤ 消息认证码MAC(Message Authentication Code)

以消息和密钥作为输入、产生定长的输出作为认证标识；

主要用于消息认证和消息完整性。

## ➤ 散列函数(Hash Function)

是一个不需要密钥的公开函数，它将任意长的信息映射成一个固定长度的信息作为认证标识。

## 7.3.1 基于消息加密的认证

消息加密本身可以提供一种认证手段。

1. 使用对称密码体制：提供机密性和认证

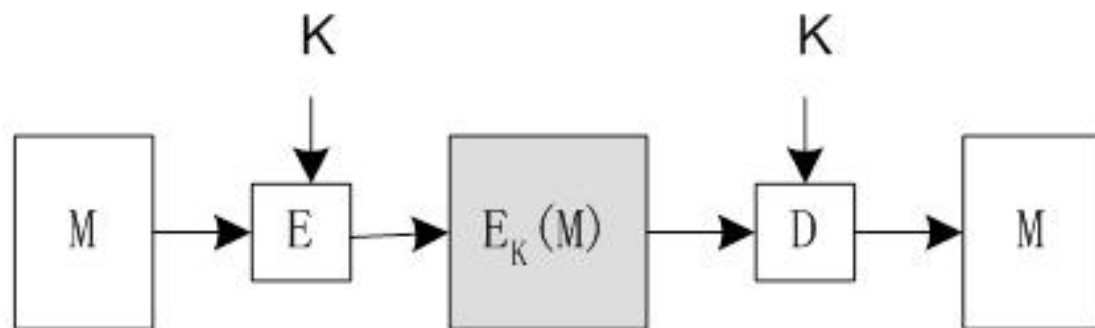


图7-11 对称加密：机密性和认证

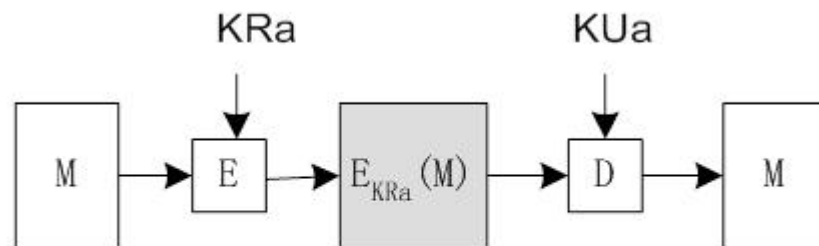
➤ 困难性：

- 接收方需要确定解密消息的合法性；
- 确定消息来源的真实性。

➤ 特点：

- ① 提供机密性；
- ② 提供认证；
- ③ 不能提供数字签名。

## 2. 基于公钥密码体制：提供认证



$K_{Ra}$ : 发送方私钥

$K_{Ua}$ : 发送方公钥

图7-12 公钥加密：认证和数字签名

只有发送方拥有并保存自己的私钥，因此，只有发送方才能产生用自己的公钥可解密的密文，所以消息一定来自于拥有该密钥的发送方。

### ➤ 特点：

- 能实现数字签名，不能提供机密性。
- 提供认证

### 3. 基于公钥密码体制：实现签名、加密和认证

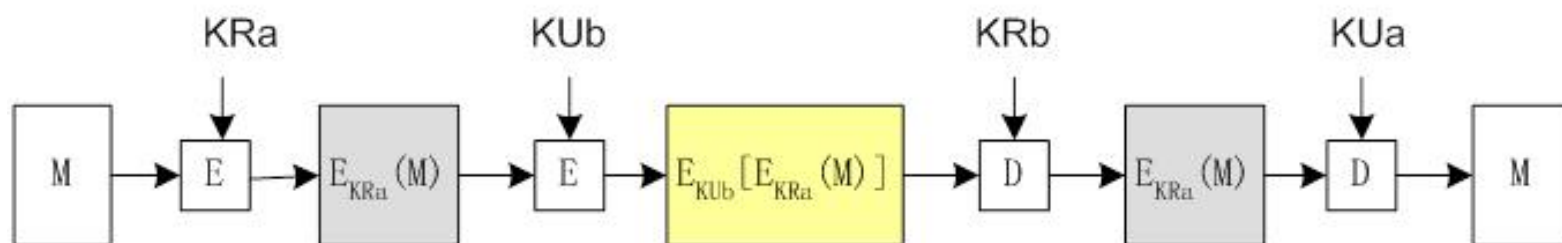


图7-13 公钥加密：机密性、认证和数字签名

#### ➤ 特点：

- 提供机密性
- 数字签名
- 认证

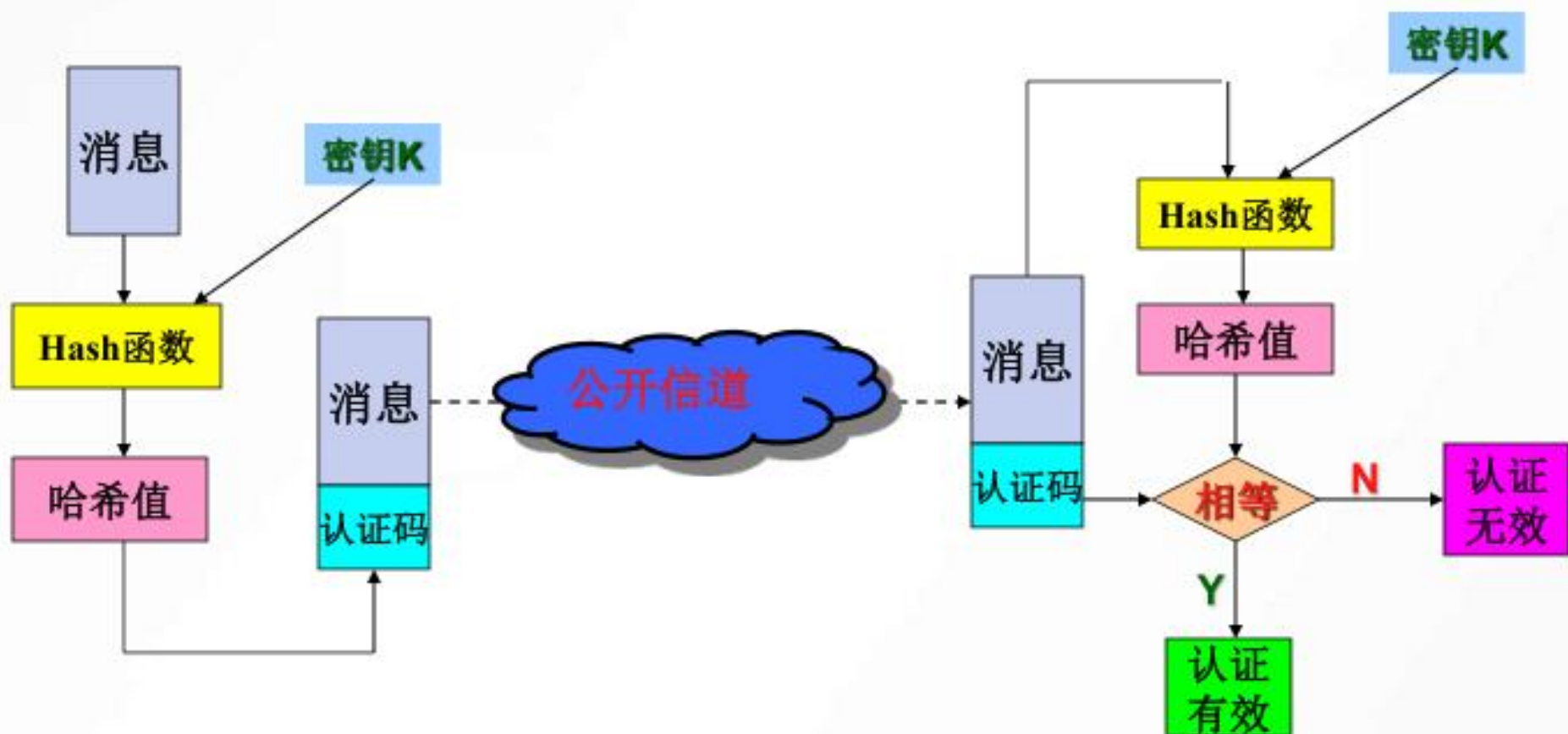
#### ➤ 缺点：一次完整的通信需要执行公钥算法的加密、解密操作各两次

## 7.3.2 基于消息认证码(MAC)的认证

消息认证码(MAC, Messages Authentication Codes), 是与密钥相关的单向散列函数, 也称为消息鉴别码或是消息校验和。此时需要通信双方A和B共享一密钥K。

消息认证码(MAC)也叫做带密钥的hash函数。消息鉴别码能提取消息的指纹。

# 消息认证的常用实现过程





设A欲发送给B的消息是M，A首先计算  $MAC = C_K(M)$ ，其中  $C_K(\cdot)$  是密钥控制的公开函数 (如哈希函数)，然后向B发送  $M \parallel MAC$ ，B收到后做与A相同的计算，求得一新MAC，并与收到的MAC做比较。

由于只有A和B知道密钥K，故可判断：

- 接收者确信报文未被更改过。
- 接收者确信报文来自声称的发送者。

## 消息认证的特点:

- MAC函数无需可逆，因为它不需要解密。
- 收发双方使用相同的密钥，MAC不能提供数字签名。
- 只提供消息认证，不能提供机密性（可在生成MAC之前或之后加密）。

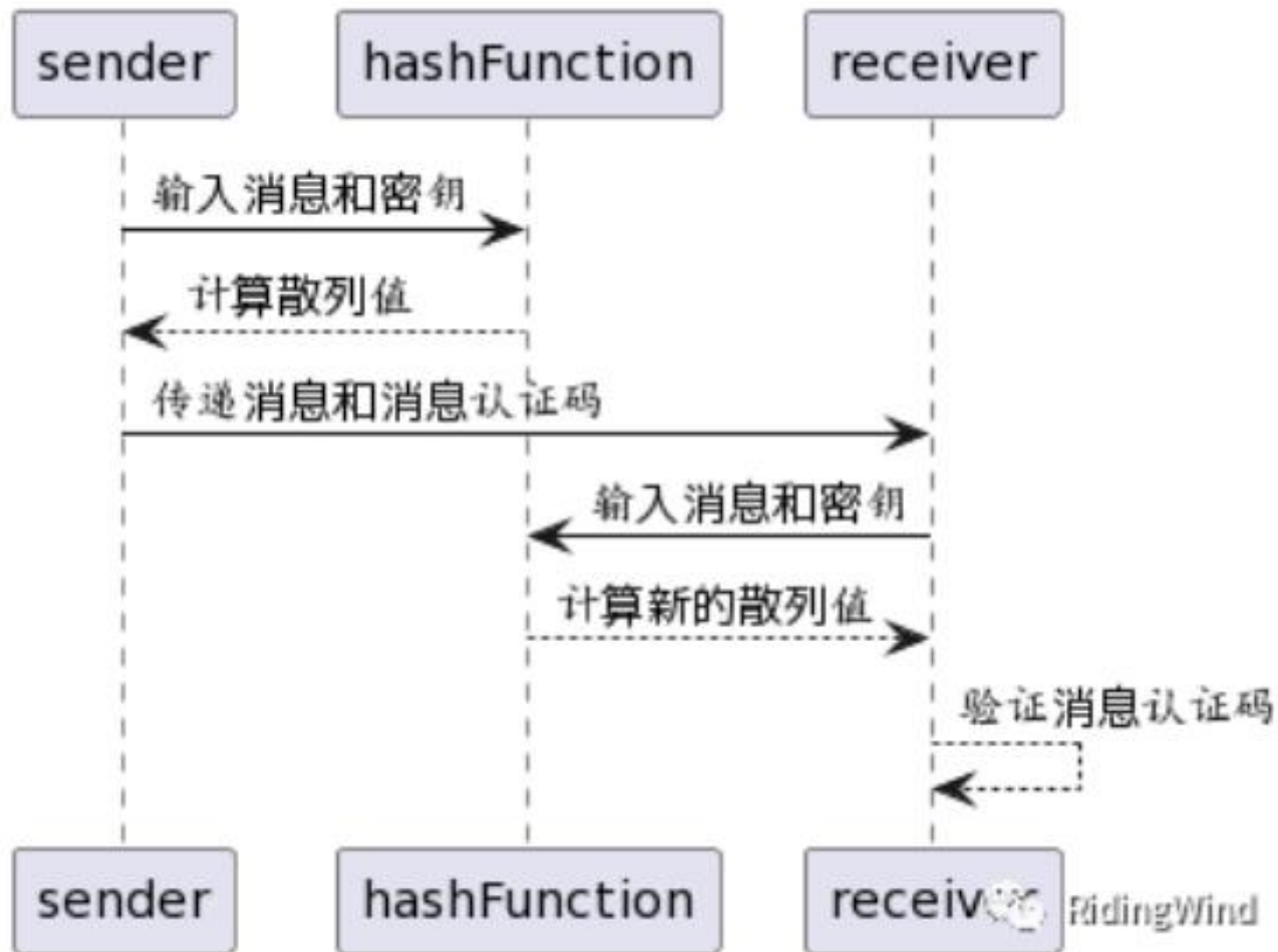
## 7.3.3 基于散列函数(HASH)的认证

利用对称分组密码体制（如DES、AES）的密码分组链接模式（CBC）一直是构造MAC的最常见的方法。然而，近几年，人们越来越感兴趣于利用哈希函数来设计MAC，这是因为像MD5、SHA-1这样的哈希函数，其软件执行速度比诸如DES、AES这样的对称分组密码要快。

然而，诸如SHA-1这样的哈希函数并不是专门为MAC而设计的，由于哈希函数不依赖于密钥，所以它不能直接用于计算MAC。目前，已经提出了许多方案将密钥加到现有的哈希函数中（HMAC—Hash-based Message Authentication Code）。

- ◆ HMAC是密钥相关的哈希运算消息认证码，HMAC运算利用**哈希算法**（常见的哈希函数包括 SHA-256 和 SHA-3），以一个**密钥**和一个**消息**为输入，生成一个消息摘要作为输出。
- ◆ 由于杂凑值是输入消息的函数值，只要输入消息有任何改变，就会导致得到不同的杂凑值，因此可用于验证消息的完整性和合法性。
- ◆ 这个消息摘要**只有知道密钥的人**才能生成，因此，它可以有效地防止未经授权的访问和篡改。（**消息认证**）

# HMAC 的工作方式



1. 发送者将消息作为输入传递给哈希函数，并将同时将密钥也作为输入传递给哈希函数。
2. 哈希函数将消息和**密钥**结合在一起进行处理，生成一个固定长度的散列值。
3. 发送者将这个散列值发送给接收者，作为消息认证码。
4. 接收者收到消息和消息认证码后，使用相同的密钥和哈希函数对消息进行处理，生成一个新的散列值。
5. 接收者将这个新的散列值与发送者提供的消息认证码进行比较。如果它们匹配，那么消息就被认为是合法的，没有被篡改。

# HMAC的应用

1. **网络通信** HMAC 用于确保通过网络传输的数据完整性。
2. **电子邮件验证** HMAC 可以用于验证电子邮件的发件人身份。通过将电子邮件的内容与发件人的密钥一起传递给哈希函数，可以生成消息认证码。这样，收件人可以验证邮件的完整性并确定它是否来自合法的发件人。
3. **身份认证** HMAC 也用于身份认证系统中。例如，当你登录银行或社交媒体帐户时，系统可以使用 HMAC 生成一个身份认证令牌，该令牌可以帮助系统验证你的身份，确保你是合法用户。

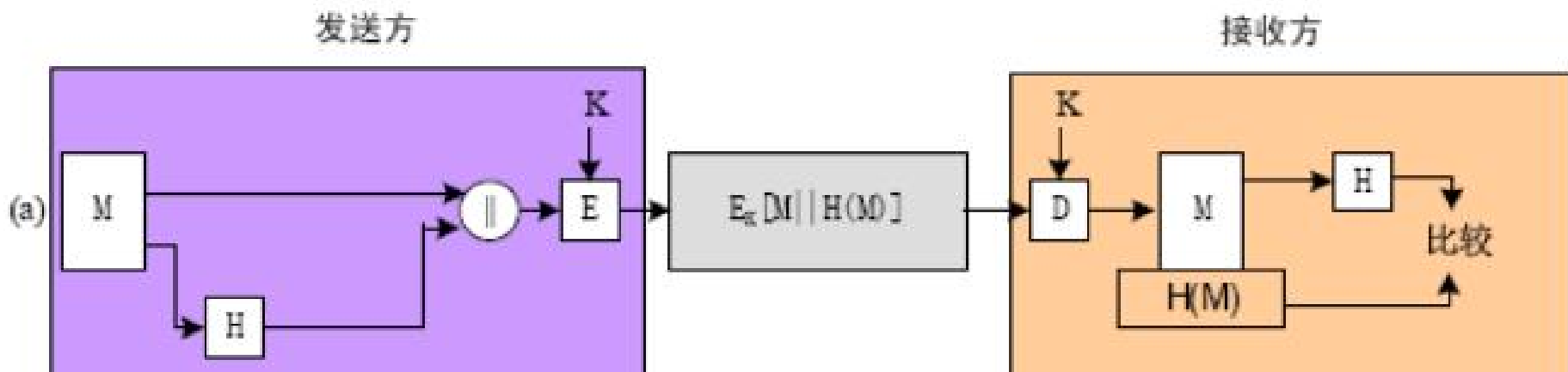
## 4.数据完整性检查

HMAC 可以用于检查数据是否在传输或存储期间发生了更改。这在云存储、备份系统和文件传输中特别有用。

## 5. 国密算法中的应用

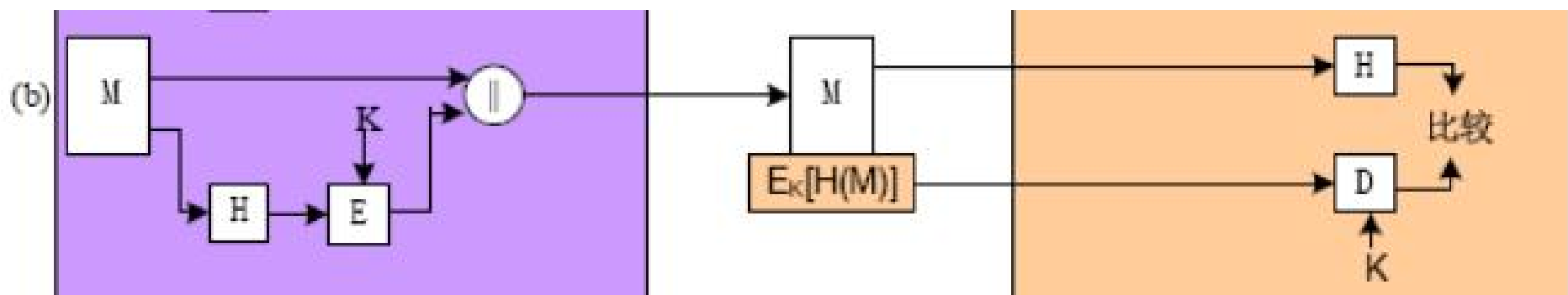
国密算法使用特定的哈希函数和密钥管理策略，以满足中国国家标准的要求。



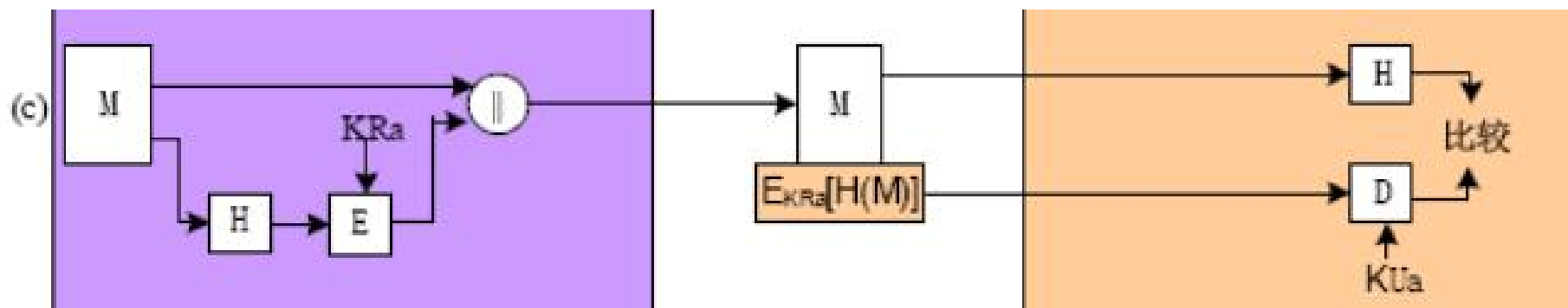


由于发送方A和接收方B共享密钥K,因此可以确定消息一定来自于A,且未被修改过。

该方法实现了消息的明文加密保护,因此可提供机密性。

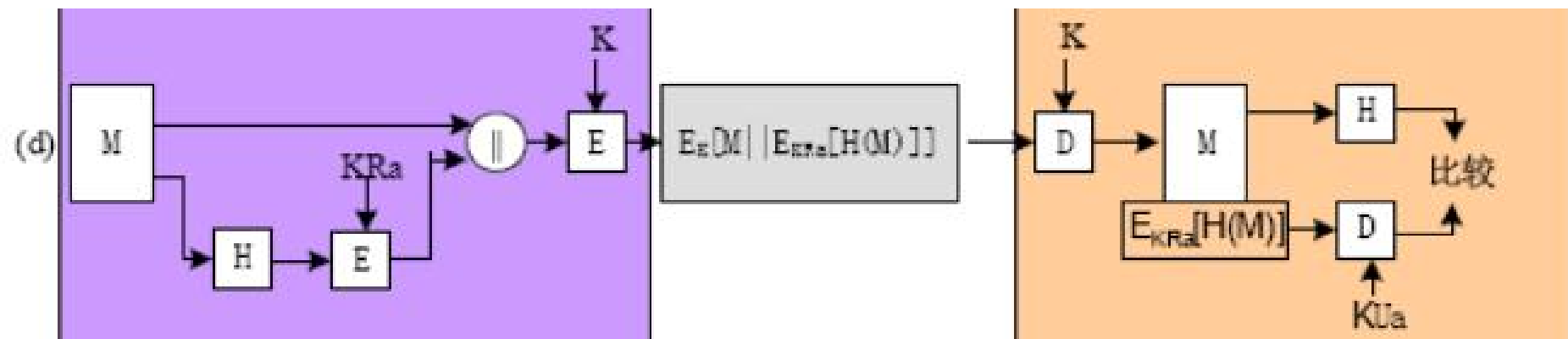


该方法中消息以明文的形式传递，因此**不能提供机密性**，适合不要求机密性的场合，有助于减少处理代价。

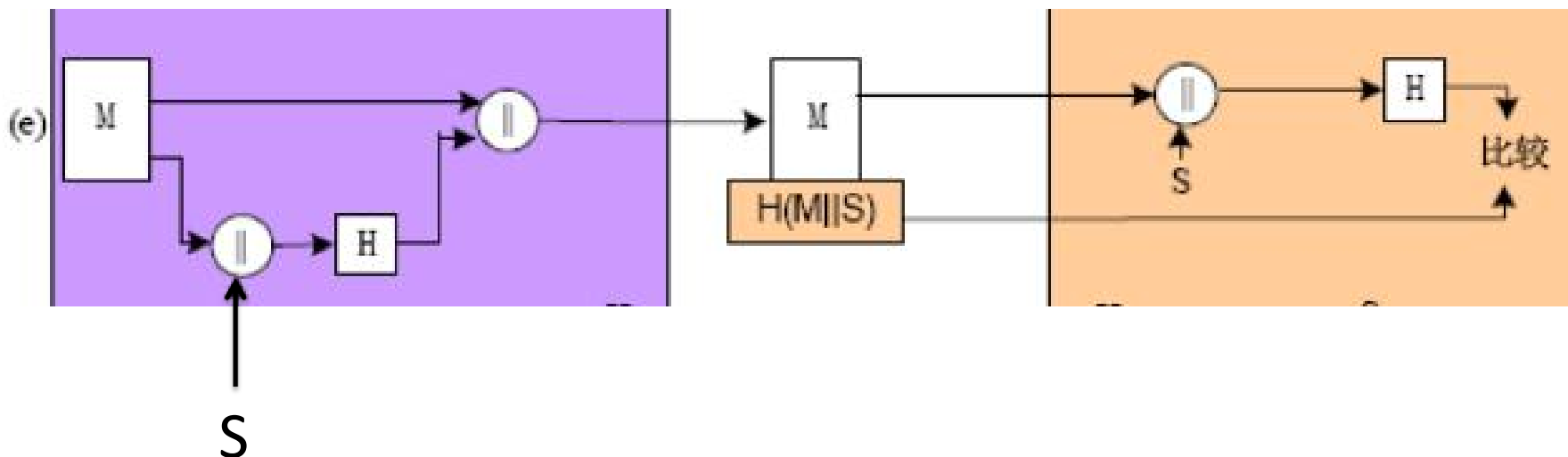


用发送方的私钥对 $H(M)$ 进行加密保护，接收方用发送方的公钥进行解密鉴别。

该方案能提供鉴别和数字签名，因为 $H(M)$ 受到加密保护，而且只有A能生成 $E_{K_{Ra}}[H(M)]$ 。

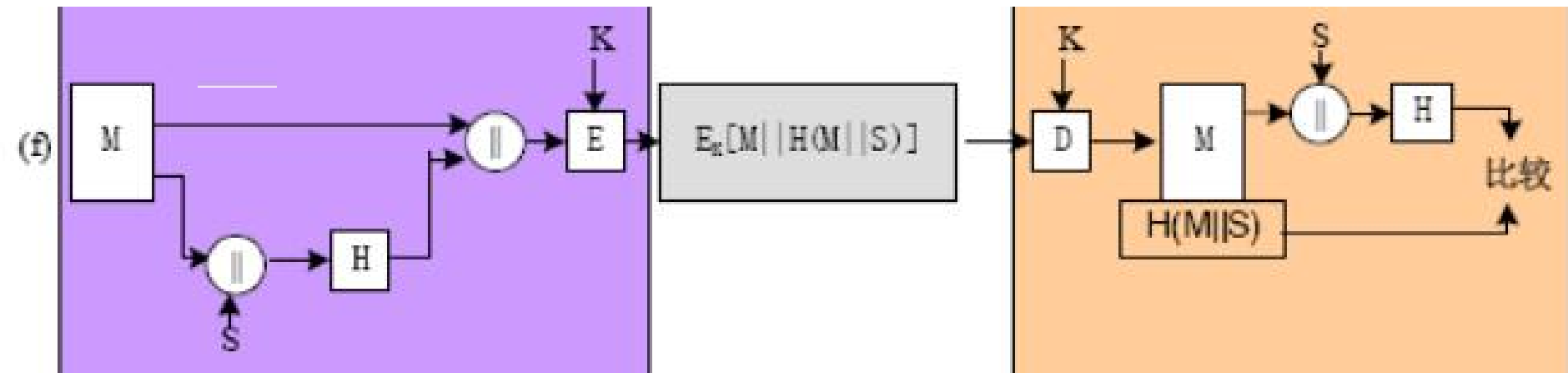


该方案既能提供鉴别和数字签名，也提供机密性，在实际中较常见。



该方案未使用密码体制，为了实现鉴别，要求发送方和接收方共享一个秘密值 $S$ 。

这种方法中，秘密值 $S$ 并不参与传递，因此，可以保证攻击者无法伪造。



该方案除了提供鉴别，还可以提供机密性。

# 消息认证码的功能

- 接收方相信发送方发来的消息未被篡改。这是因为攻击者不知道密钥，所以不能够在篡改消息后相应地篡改MAC，而如果仅篡改消息，则接收方计算的新MAC将与收到的MAC一定是不同的。
- 接收方相信发送方不是冒充的。这是因为除收发双方外再无其他人知道密钥，因此其他人不可能对发送方发送的消息计算出正确的MAC。

# 本章小结

- ◆ Hash函数的定义
- ◆ Hash函数的用途
- ◆ 哈希函数的安全性
- ◆ SHA-1 算法
- ◆ 基于MAC的消息认证



# 作业

7.1-7.4 7.6 7.7

***Thank you!***