现代密码学

第五章　消息认证和哈希函数

信息与软件工程学院

主要内容

5.1　消息认证码

5.2　哈希函数

5.3　SHA-1哈希算法

5.4　SHA-3哈希算法

5.5　SM3 哈希算法

5.6　HMAC算法

现代密码学

消息认证码

信息与软件工程学院

消息认证

- 抗击被动攻击：加密
- 抗击主动攻击：消息认证

- 消息认证是一个过程，用以验证接收消息的真实性和完整性，同时还用于验证消息的顺序性和时间性。

消息认证

•消息认证机制需要产生认证符。

•认证符：用于认证消息的数值。

•认证符的产生方法：消息认证码MAC（message authentication code）和杂凑函数（hash function）两大类。

消息认证码

消息认证码的定义及使用方式

•消息认证码：指消息被一密钥控制的公开函数作用后产生的用作认证符的固定长度的数值，或称为密码校验和。

$$M||C_K(M)$$

•此时需要通信双方A和B共享一密钥K。

• 如果仅收发双方知道K，且B计算得到的MAC与接收到的MAC一致，则这一就实现了以下功能：

•① 接收方相信发送方发来的消息未被篡改。

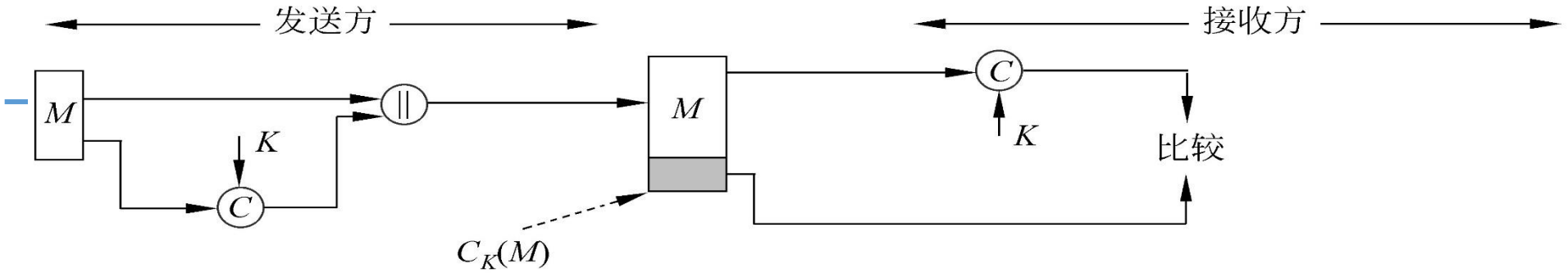
•② 接收方相信发送方不是冒充的。

接收方能向第3方证明消息的来源吗？

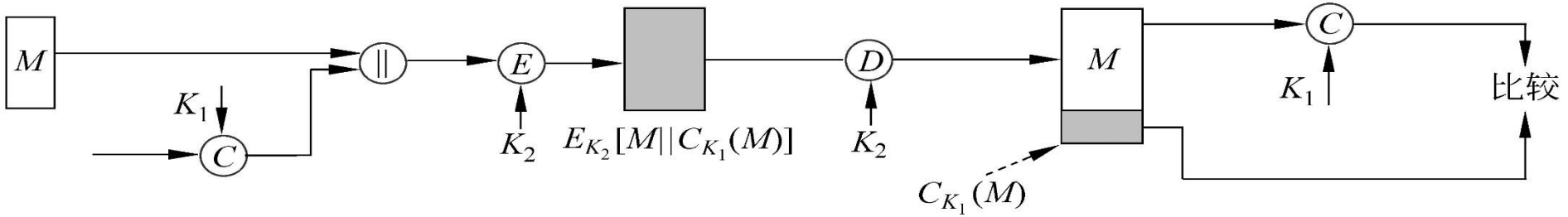
消息认证码

•MAC函数与加密算法类似，不同之处为MAC函数不必是可逆的，因此与加密算法相比更不易被攻破。

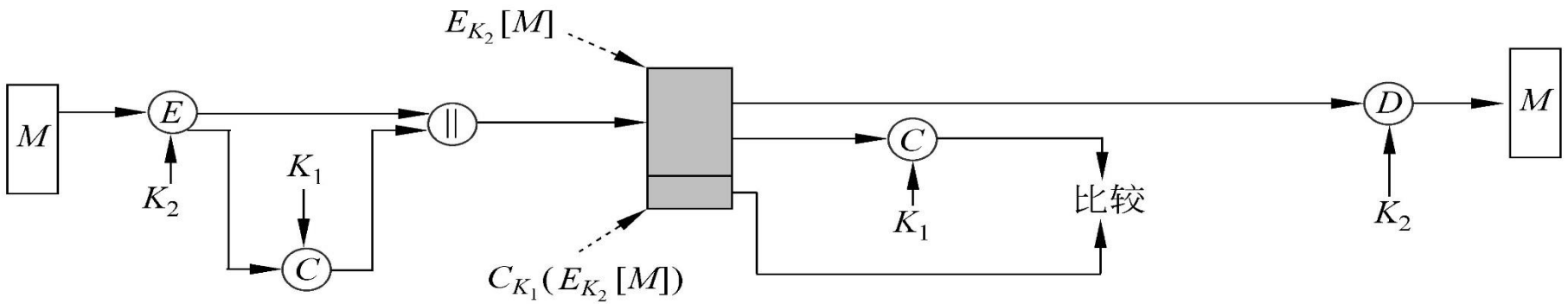
•上述过程中，由于消息本身在发送过程中是明文形式，所以这一过程只提供认证性而未提供保密性。



(a) 消息认证



(b) 认证性和保密性：对明文认证



(c) 认证性和保密性：对密文认证

图5.1 MAC的基本使用方式

产生MAC的函数应满足的要求

(1) 消息认证码的穷搜索攻击的代价比使用相同长度密钥的加密算法的穷搜索攻击还要大。

已知 $M \| C_K(M) = MAC$，找 K。

- 第1轮 已知M_1、MAC_1，其中$MAC_1=C_K(M_1)$。对所有2^k个可能的密钥计算$MAC_i=C_{Ki}(M_1)$，得2^{k-n}个可能的密钥。
- 第2轮 已知M_2、MAC_2，其中$MAC_2=C_K(M_2)$。对上一轮得到的2^{k-n}个可能的密钥计算$MAC_i=C_{Ki}(M_2)$，得$2^{k-2\times n}$个可能的密钥。
- 如此下去，如果$k=\alpha n$，则上述攻击方式平均需要α轮。例如，密钥长为80比特，MAC长为32比特，则第1轮将产生大约2^{48}个可能密钥，第2轮将产生2^{16}个可能的密钥，第3轮即可找出正确的密钥。

产生MAC的函数应满足的要求

(2)有些攻击法却不需要寻找产生MAC所使用的密钥。攻击者可假冒发假消息 $M' \| C_K(M')$ **给接收方。**

产生MAC的函数应满足的要求

•例1：设M=($X_1 \| X_2 \| \ldots \| X_m$)是由64比特长的分组$X_i$(i=1,...,m)链接得到的，其消息认证码由以下方式得到：

$$\triangle(M) = X_1 \oplus X_2 \oplus \cdots \oplus X_m$$

$$C_K(M) = E_K[\triangle(M)]$$

•其中 ⊕ 表示异或运算，加密算法是电码本模式的DES。因此，密钥长为56比特，MAC长为64比特，如果敌手得到$M \| C_K(M)$，那么敌手使用穷搜索攻击寻找K将需做2^{55}次加密。

•然而敌手还可用以下方式攻击系统：随机选用X_1，…，X_{m-1}

$$X_1, \cdots, X_{m-1} \rightarrow Y_1, \cdots, Y_{m-1}$$

$$Y_m = Y_1 \oplus Y_2 \oplus \cdots \oplus Y_{m-1} \oplus \Delta(M)$$

$$X_m \rightarrow Y_m$$

伪造一新消息：

$$M' = Y_1 \| Y_2 \| \cdots \| Y_{m-1} \| Y_m$$

且M'的MAC与原消息M的MAC相同。

产生MAC的函数应满足的要求

•考虑到MAC所存在的以上攻击类型，可知它应满足以下要求，其中假定敌手知道函数C，但不知道密钥K：

•① 如果敌手得到M和$C_K(M)$，则构造一满足$C_K(M')=C_K(M)$的新消息M′在计算上是不可行的。

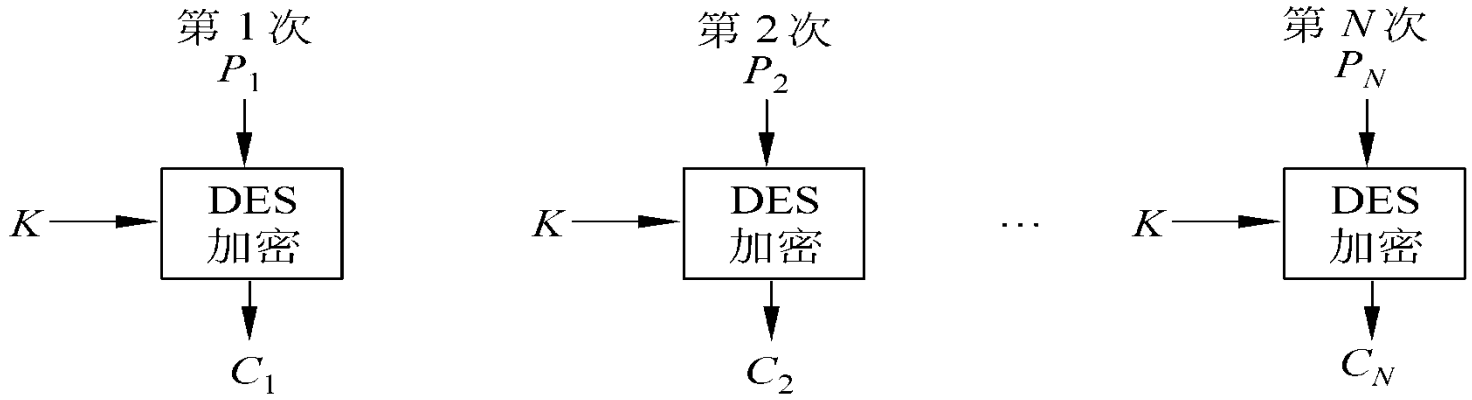
•② $C_K(M)$在以下意义下是均匀分布的： 随机选取两个消息M、M′，$Pr[C_K(M)=C_K(M')]=2^{-n}$，其中n为MAC的长。

•③ 若M′是M的某个变换，即M′=f(M)，例如f为插入一个或多个比特，那么$P_r[C_K(M)=C_K(M')]= 2^{-n}$。

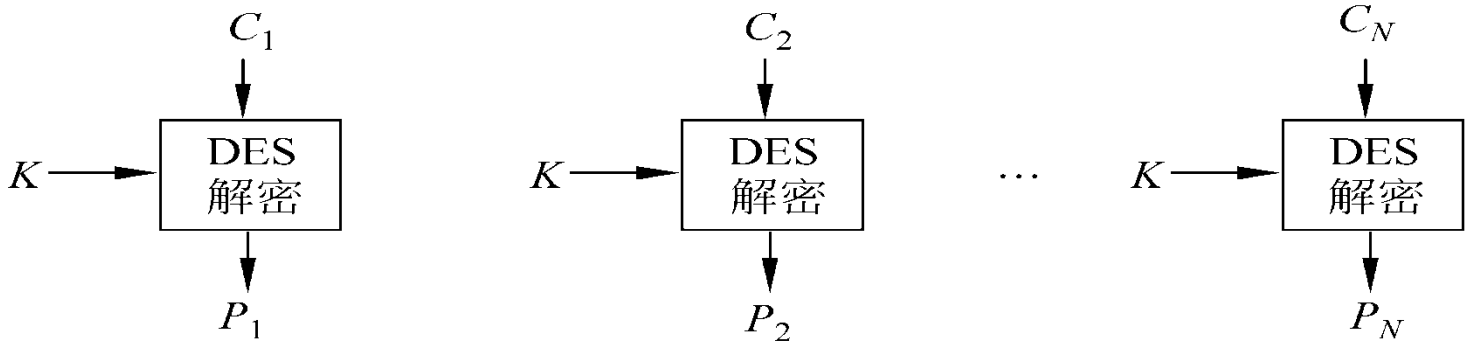
使用DES的工作模式设计MAC

1. 电码本(ECB)模式
2. 密码分组链接(CBC)模式
3. 密码反馈(CFB)模式
4. 输出反馈(OFB)模式
5. 计数器模式

电码本ECB（Electronic Code Book）模式

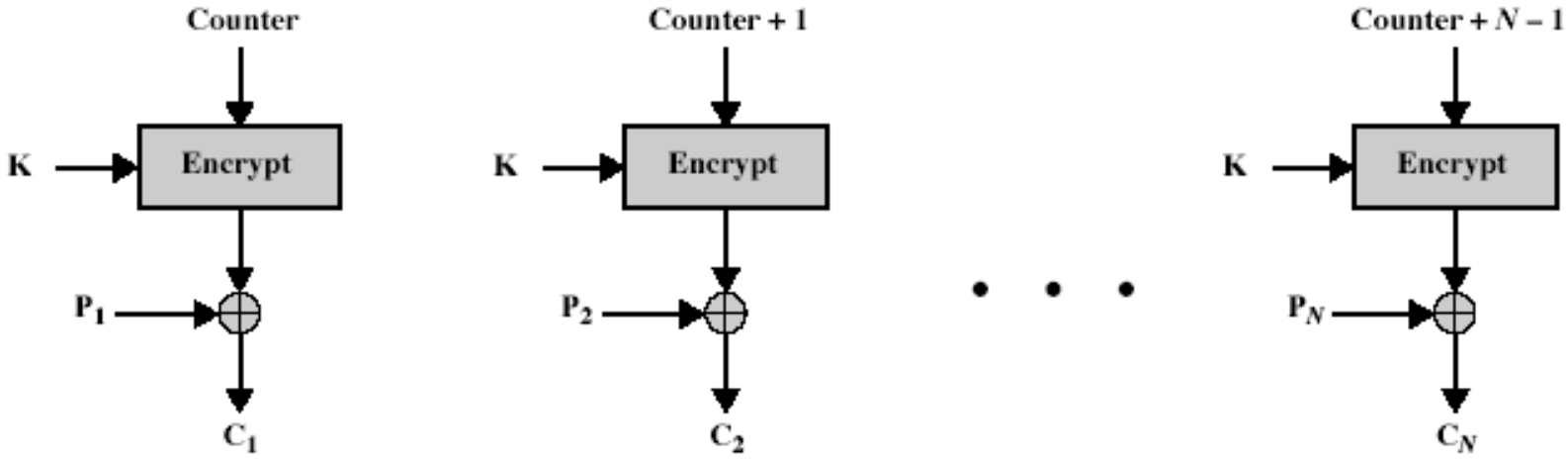


(a) 加密

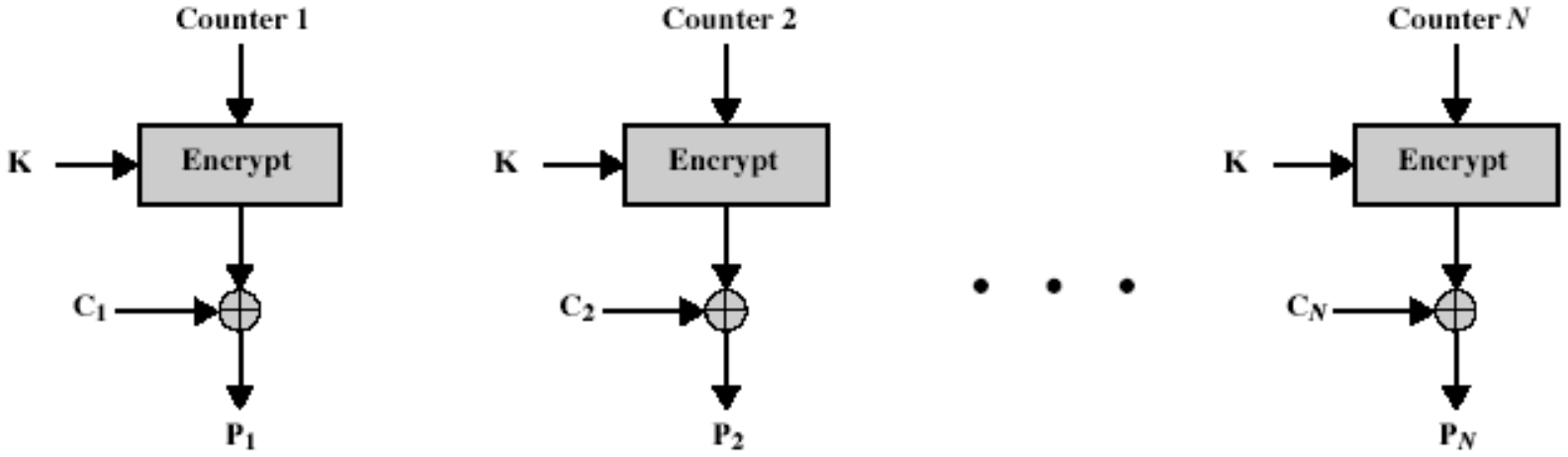


(b) 解密

计数器模式的结构

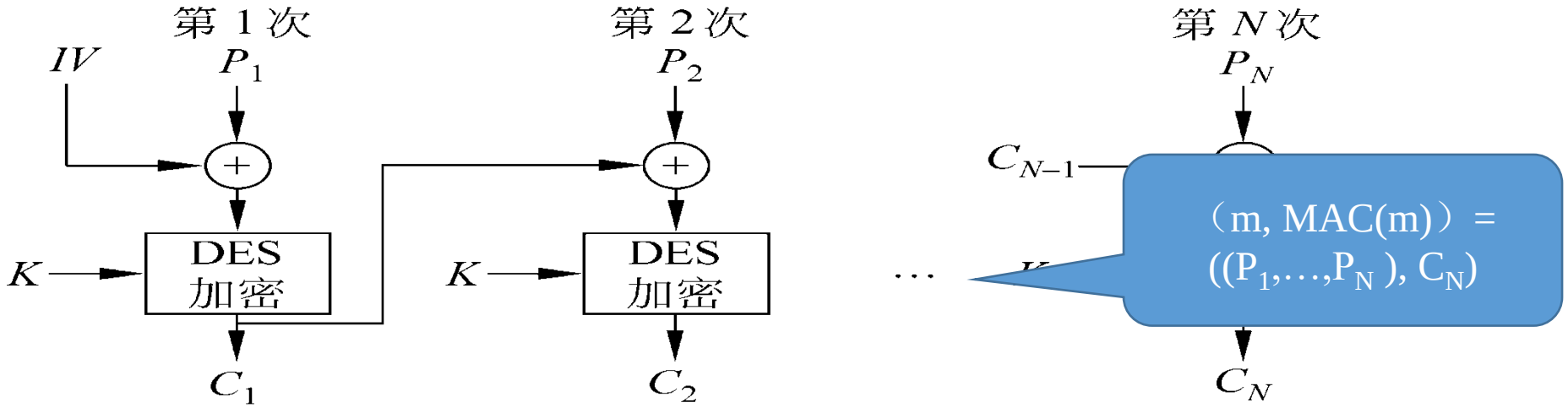


(a) Encryption

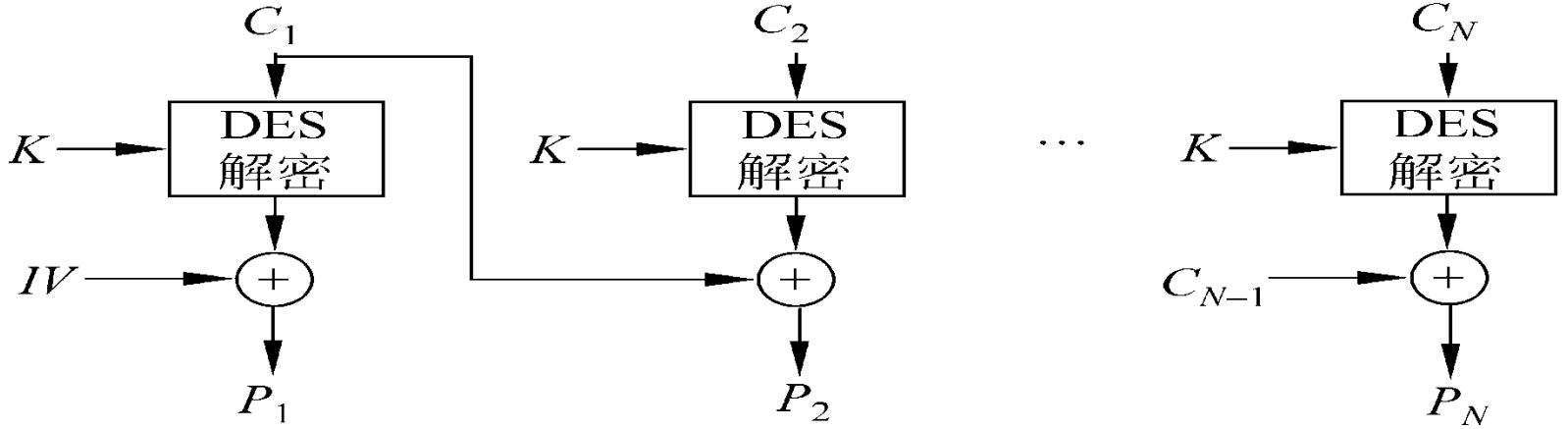


(b) Decryption

密码分组链接CBC（Cipher Block Chaining）模式



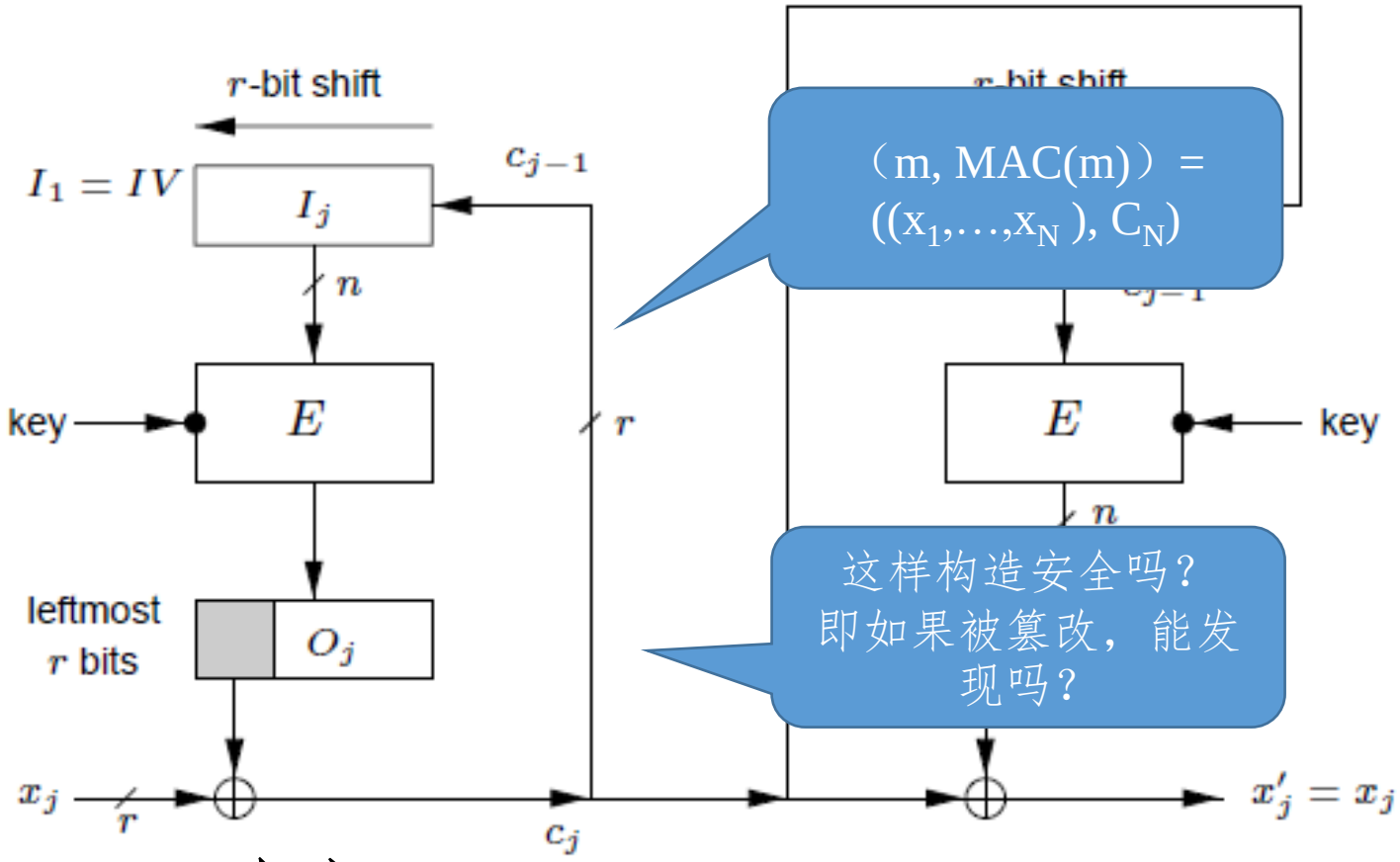
(a) 加密



(b) 解密

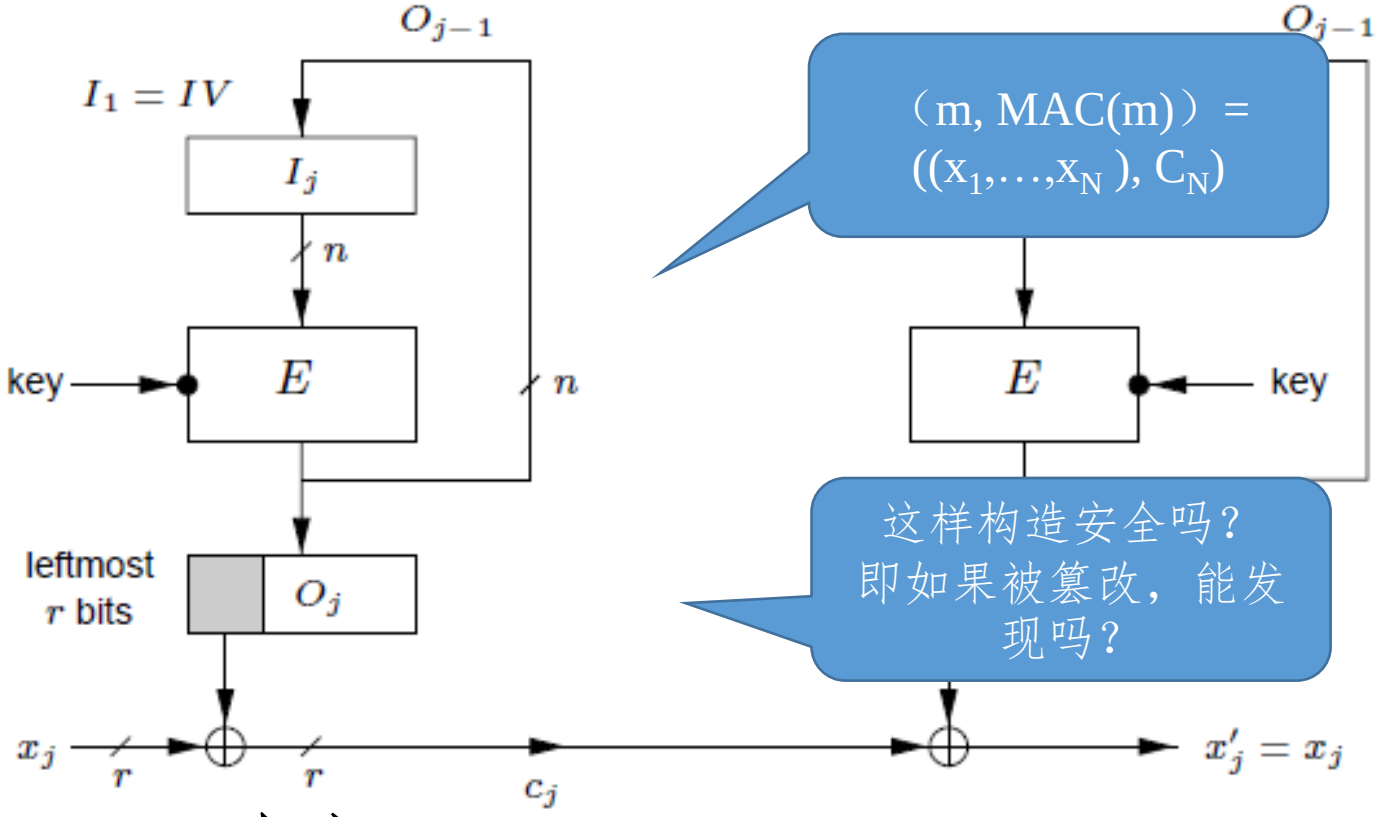
CFB的加密解密

- 若记$IV=c_{-l+1}\ldots c_{-1}\,c_0$，$|c_i|=r$，则加密过程可表示为：$c_i = x_i \oplus left_r(E_k(c_{i-l}\cdots c_{i-2}c_{i-1}))$



输出反馈OFB(Output Feedback)模式

- OFB模式在结构上类似于CFB模式，但反馈的内容是DES的输出而不是密文！



现代密码学

哈希函数的基本概念

信息与软件工程学院

Hash函数的定义

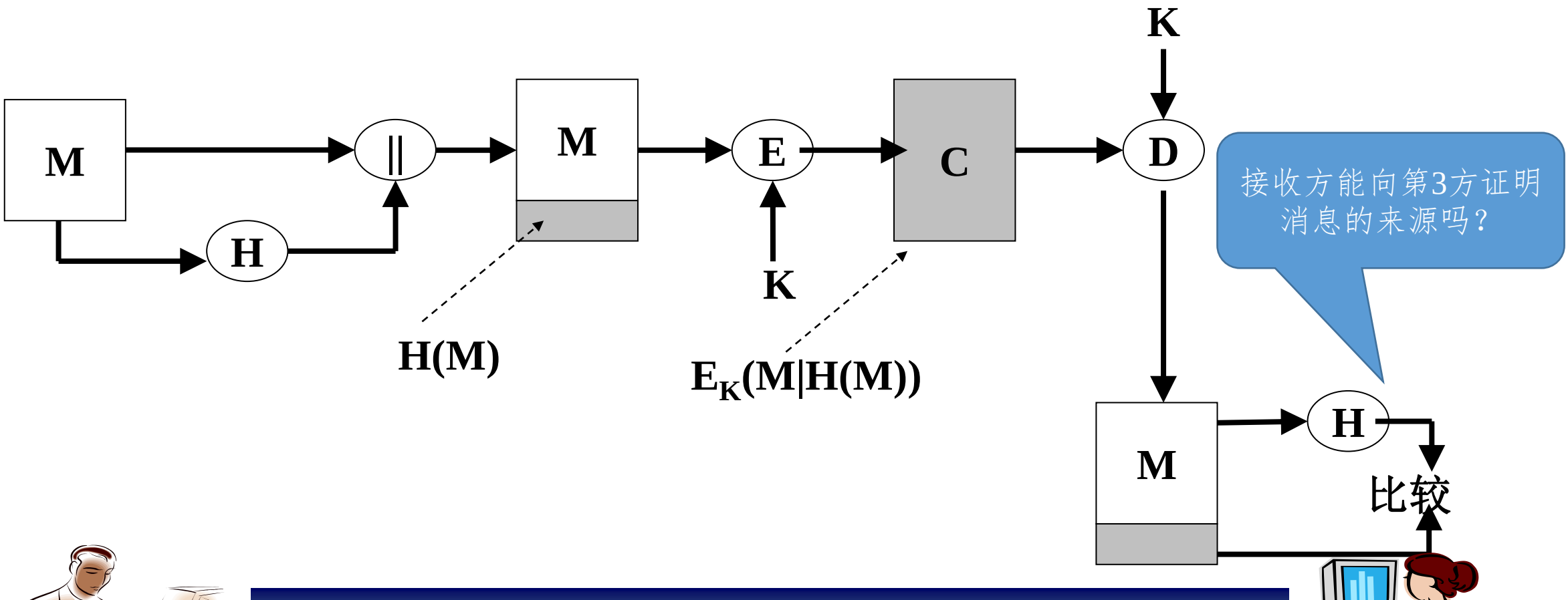
- **Hash函数的定义**
  - 将任意长的消息M映射为较短的、固定长度的一个值H(M)。
  - **Hash函数也称为哈希函数、散列函数、压缩函数、杂凑函数、指纹函数等。其函数值H(M)为哈希值、散列值、杂凑码、指纹、消息摘要等。**
  - **Hash函数H一般是公开的。**

例2

M 是一个长消息，设$M=(M_1, M_2, \cdots\cdots, M_k)$，其中$M_i$为$l$长的比特串，定义函数$H$如下：

- $H(M) = M_1 \oplus M_2 \oplus \cdots\cdots \oplus M_k$

认证函数：Hash函数的基本用法



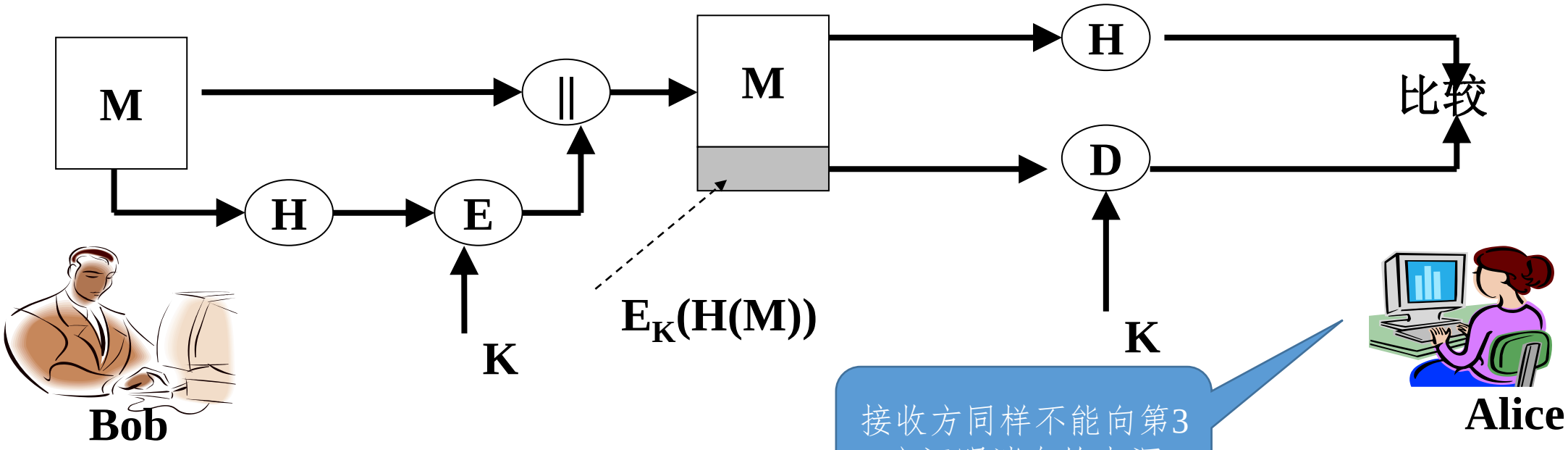
Bob

提供认证
提供保密

Alice

认证函数：Hash函数（续）

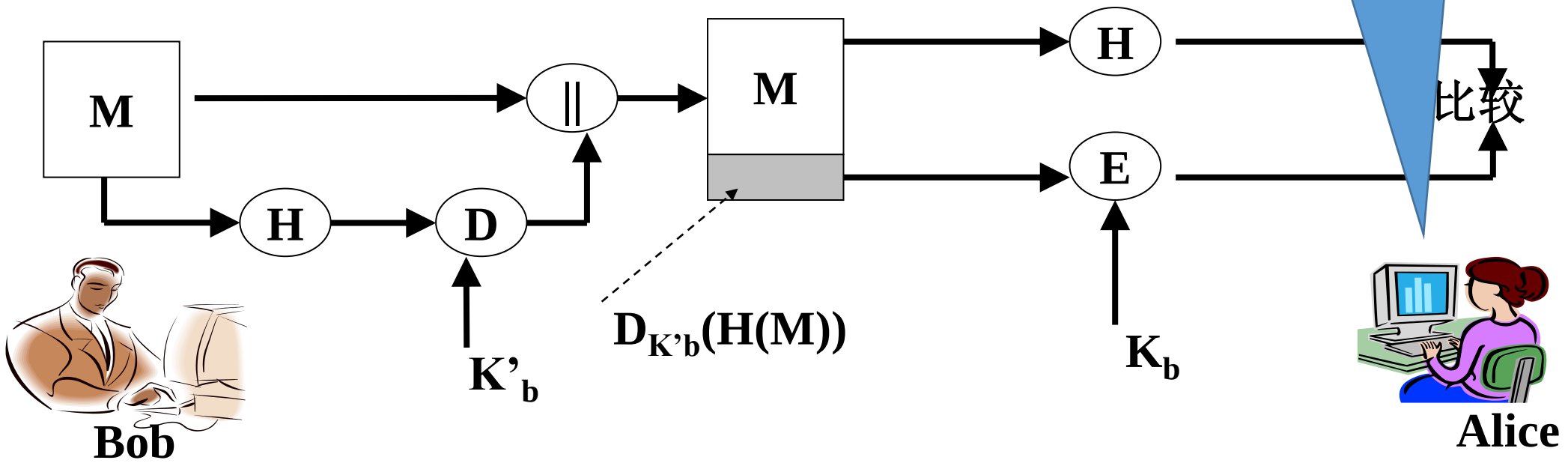
哈希函数的基本用法（b）



提供认证

认证函数：Hash函数（续）

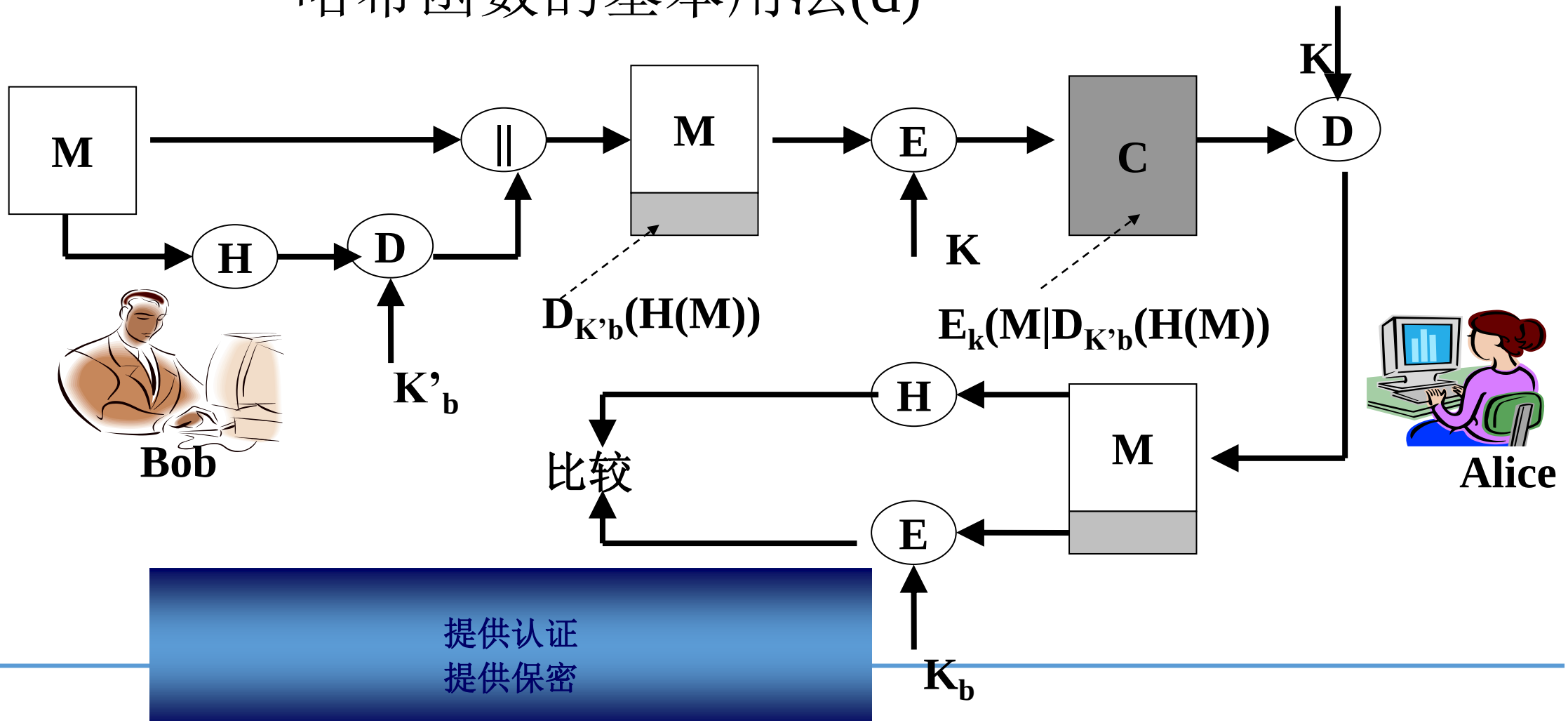
哈希函数的基本用法（c）



提供认证

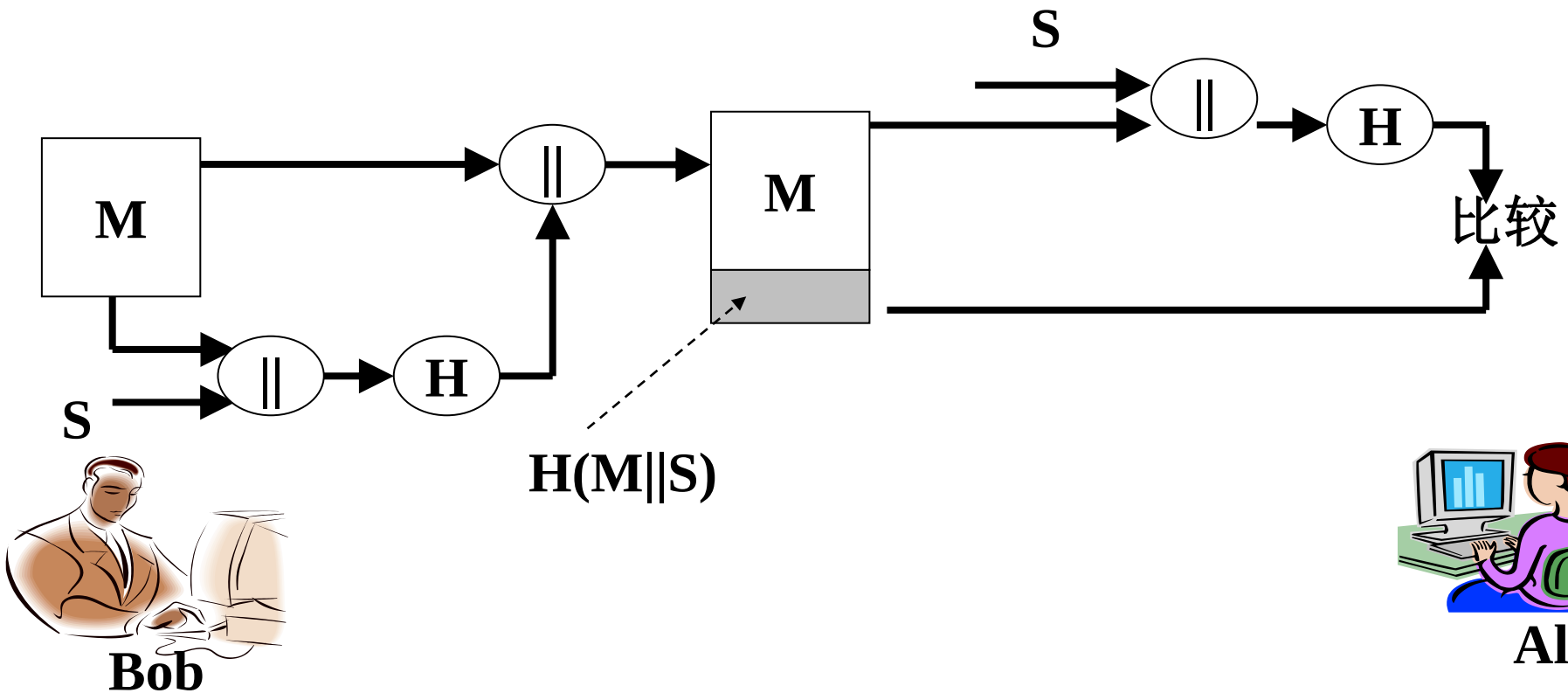
认证函数：Hash函数（续）

哈希函数的基本用法(d)



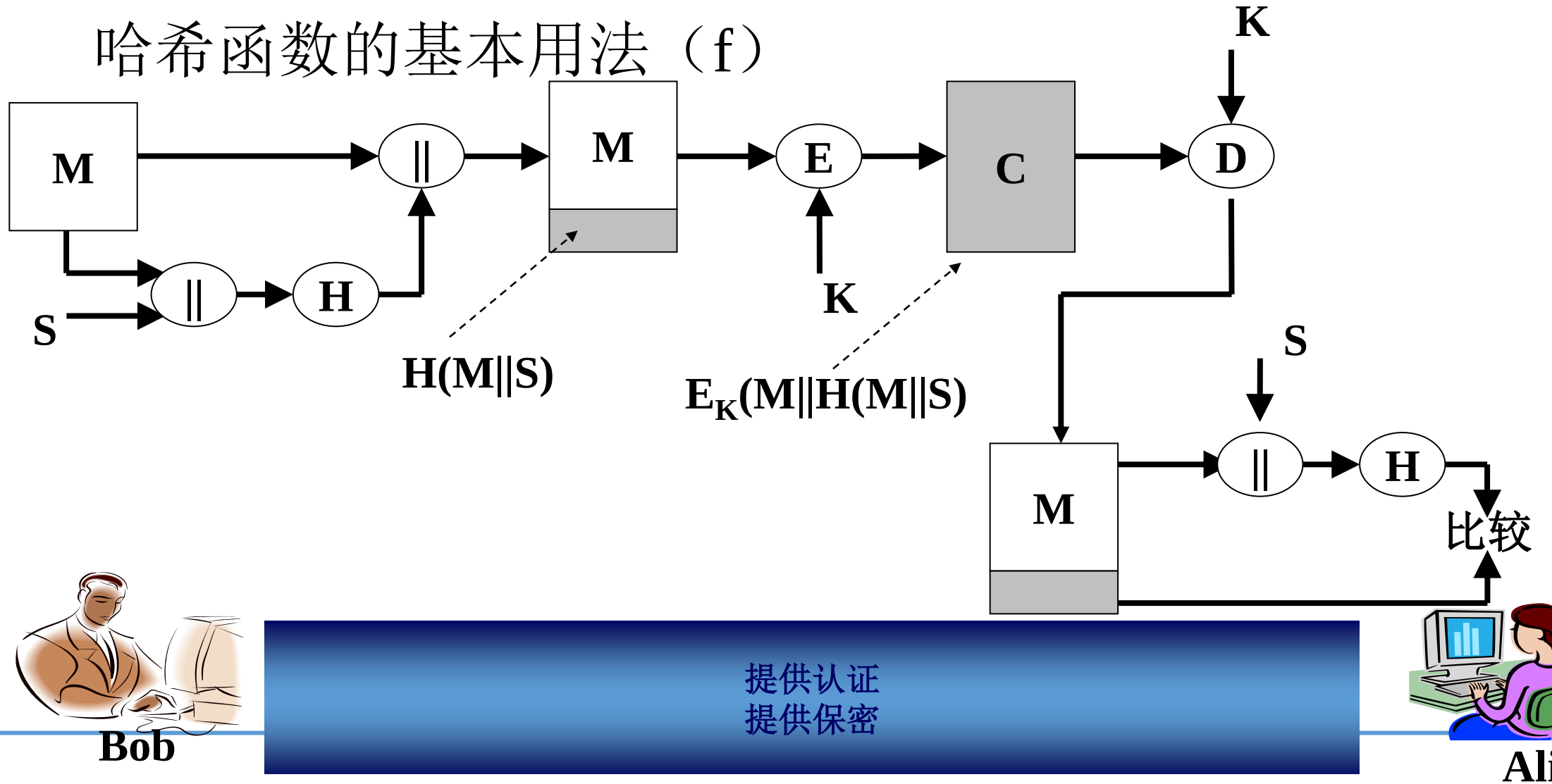
认证函数：Hash函数（续）

哈希函数的基本用法（e）



提供认证

认证函数：Hash函数（续）



Hash函数满足条件

- **Hash函数的目的是为需认证的数据产生一个“指纹”，Hash函数应满足以下条件：**
  - **Hash函数函数的输入可以是任意长**
  - **Hash函数函数的输出是固定长**
  - 易于在软件和硬件实现

Hash函数满足的安全条件

- 同时，Hash函数为了实现安全认证，需要满足如下安全条件：
  - 单向性：已知x，求H(x)较为容易；但是，已知h，求使得H(x)=h的x在计算上是不可行的。
  - 抗弱碰撞性：已知x，找出y(y≠x)使得H(y)=H(x)在计算上是不可行的。
  - 抗强碰撞性：找出任意两个不同的输入x、y，使得H(y)=H(x)在计算上是不可行的。

Hash函数满足的安全条件

- 同时，Hash函数为了实现安全认证，需要满足如下安全条件：
  - 单向性：已知x，求H(x)较为容易；但是，已知h，求使得H(x)=h的x在计算上是不可行的。
  - 单向性问题：已知H(x)，求x
  - 抗弱碰撞性：已知x，找出y(y≠x)使得H(y)=H(x)在计算上是不可行的。
  - 抗弱碰撞性问题：已知H(x)，x，求y (y≠x)使得H(y)=H(x)
  - 抗强碰撞性：找出任意两个不同的输入x、y，使得H(y)=H(x)在计算上是不可行的。
  - 抗强碰撞性问题：已知H(x)，求x、y，使得H(y)=H(x)

例2（续）

M 是一个长消息，设$M=(M_1, M_2, \cdots\cdots, M_k)$，其中$M_i$为$l$长的比特串，定义函数$H$如下：

- $H(M) = M_1 \oplus M_2 \oplus \cdots\cdots \oplus M_k$

单选题

- 在Hash函数中，已知x，找到$y(y\neq x)$满足$h(y)=h(x)$在计算上是不可行的，这一性质称为_____。

  A、完整性　B、单向性　C、抗弱碰撞性　D、抗强碰撞性

- 在Hash函数中，找到任意两个不同输入$y\neq x$，满足$h(y)=h(x)$在计算上是不可行的，这一性质称为_____。

  A、完整性　B、单向性　C、抗弱碰撞性　D、抗强碰撞性

- 已知x，求H(x)较为容易；但是，已知h，求使得H(x)=h的x在计算上是不可行的，这一性质称为_____。

  A、完整性　B、单向性　C、抗弱碰撞性　D、抗强碰撞性

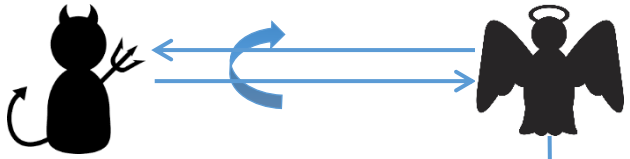
3个安全性的关系

密码游戏与归约

假设

G_1的可证明安全性：基于G_2证明G_1的安全性

A (敌手)　　CH (挑战者)



敌手能攻破密码游戏⇔挑战者输出1

1/0

Pr[CH 输出 1]=negl=>密码游戏是安全的

可忽略的概率

G_2安全=====> G_1安全

几乎所有的计算性的假设和安全性都能用密码游戏来表示

G_1不安全=====> G_2不安全

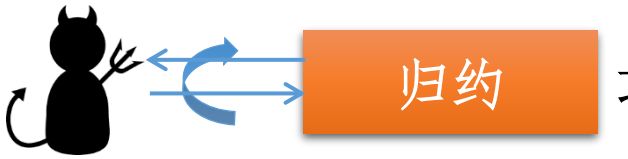
存在敌手攻破G_1=====>存在敌手攻破 G_2

G_1到G_2的归约

攻破G_1的敌手

G_2的挑战者

1/0

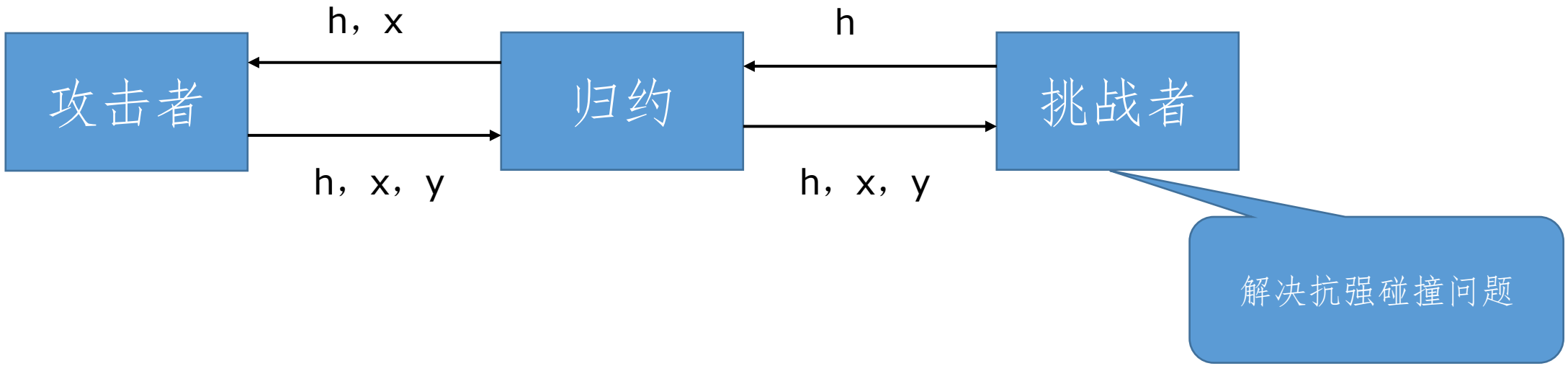


攻破G_2

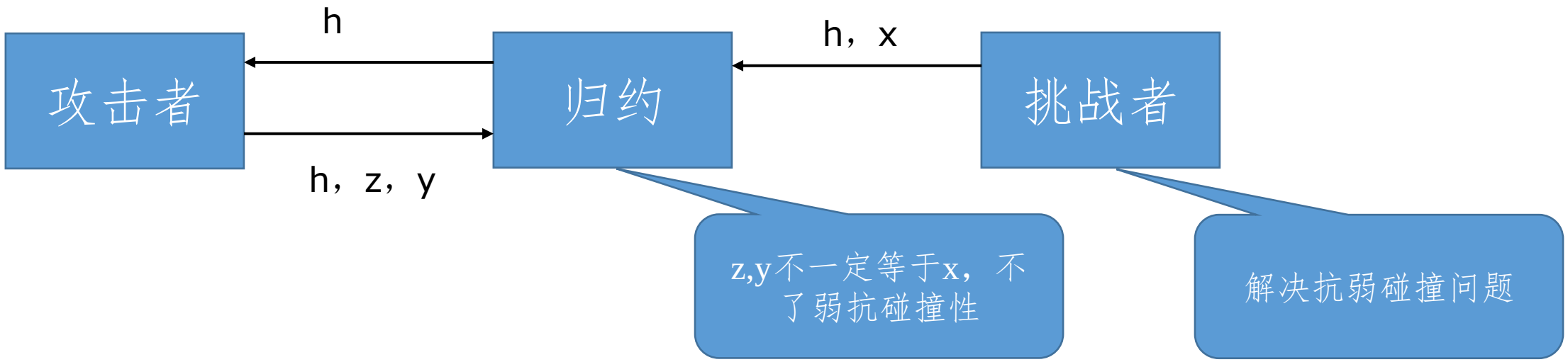
=====> G_1是安全的

3个安全性的关系

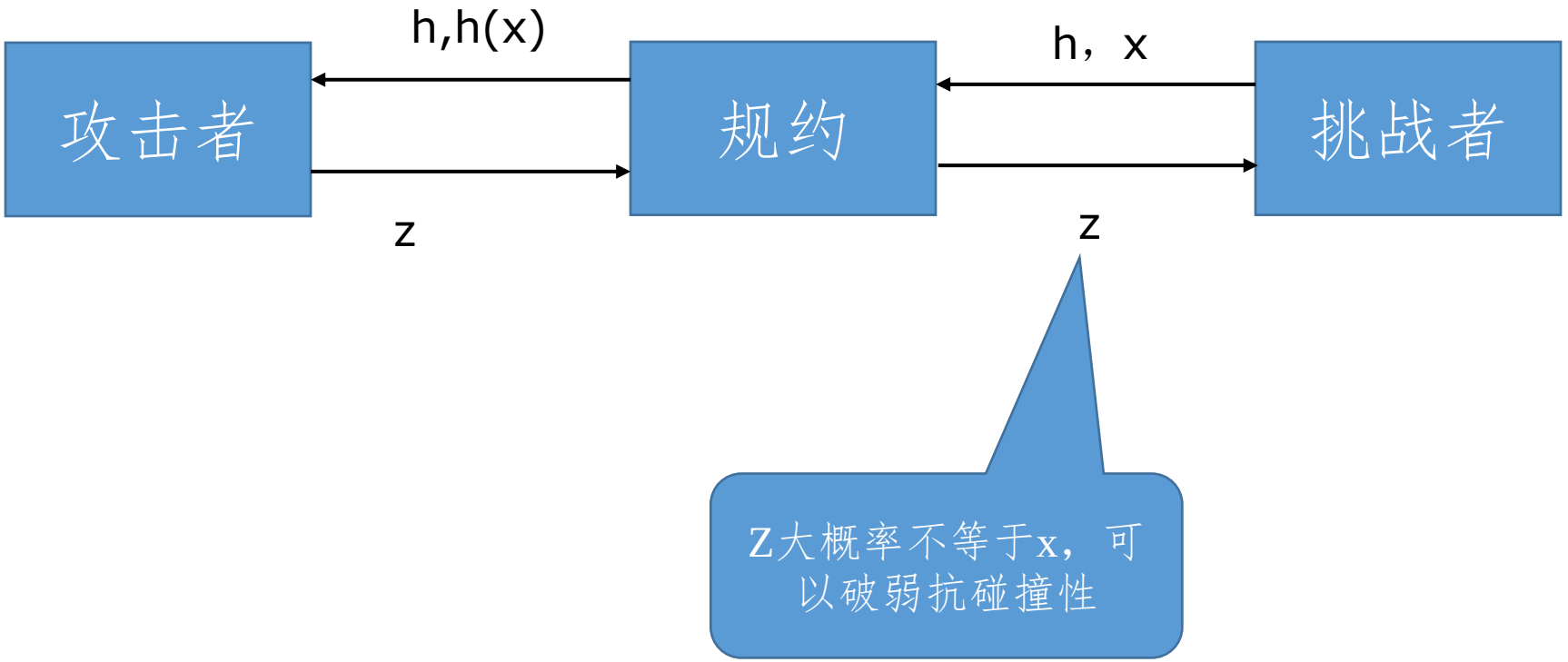
强抗碰撞性到弱抗碰撞性



弱抗碰撞性到强抗碰撞性？



弱抗碰撞性到单向性



Hash函数满足的安全条件

- 同时，Hash函数为了实现安全认证，需要满足如下安全条件：
  - 单向性：已知x，求H(x)较为容易；但是，已知h，求使得H(x)=h的x在计算上是不可行的。
  - 抗弱碰撞性：已知x，找出y(y≠x)使得H(y)=H(x)在计算上是不可行的。
  - 抗强碰撞性：找出任意两个不同的输入x、y，使得H(y)=H(x)在计算上是不可行的。

生日攻击-相关问题

- 问题1---第Ⅰ类生日攻击问题
  - 已知一杂凑函数H有n个可能的输出，H(x)是一个特定的输出，如果对H随机取k个输入，则至少有一个输入y使得H(y)=H(x)的概率为0.5时，k有多大？

  - 以后为叙述方便，称对杂凑函数H寻找上述y的攻击为第I类生日攻击。

第一类生日攻击

- 因为H有*n*个可能的输出，所以输入y产生的输出H(y)等于特定输出H(x)的概率是1/*n*，反过来说H(y)≠H(x)的概率是1-1/n。
- y取k个随机值而函数的k个输出中没有一个等于H(x)，其概率等于每个输出都不等于H(x)的概率之积，为$[1-1/n]^k$，所以y取k个随机值得到函数的k个输出中至少有一个等于H(x)的概率为$1-[1-1/n]^k$。
- 由当$|x|<<1$，$(1+x)^k \approx 1+kx$，可得

$$1-[1-1/n]^k \approx 1-[1-k/n]=k/n$$

- 若使上述概率等于0.5，则k=n/2。特别地，如果H的输出为m比特长，即可能的输出个数$n=2^m$，则

$$k=2^{m-1}$$

生日攻击-相关问题

- 问题2---生日悖论
  - 在k个人中至少有两个人的生日相同的概率大于0.5时，k至少多大？

生日攻击-相关问题

- 问题2---生日悖论
  - 设有k个整数项，每一项都在1到n之间等可能地取值。P(n, k)：k个整数项中至少有两个取值相同的概率
  - 生日悖论就是求使得P(365,k)≥0.5的最小k。
    Q(365, k)：k个数据项中任意两个取值都不同的概率。

生日攻击-相关问题

- 问题2---生日悖论
  - 如果**k>365**，则不可能使得任意两个数据都不相同，因此假定**k≤365**。**k**个数据项中任意两个都不相同的所有取值方式数为

$$365 \times 364 \times \cdots \times (365 - k + 1) = \frac{365!}{(365 - k)!}$$

  - 如果去掉任意两个都不相同这一限制条件，可得**k**个数据项中所有取值方式数为$\mathbf{365^k}$。所以可得

$$Q(365, k) = \frac{365!}{(365 - k)!\ 365^k}$$

$$P(365, k) = 1 - Q(365, k) = 1 - \frac{365!}{(365 - k)!\ 365^k}$$

生日攻击-相关问题

- 问题2---生日悖论
  - 当k=23时，P(365,23)=0.5073，即上述问题只需23人，人数如此之少。
  - 若k=100，则P(365,100)=0.9999997，即获得如此大的概率。
  - 之所以称这一问题是悖论是因为当人数k给定时，得到的至少有两个人的生日相同的概率比想象的要大得多。

生日攻击-相关问题

- 问题---生日悖论推广问题
  - 已知一个在**1**到***n***之间均匀分布的整数型随机变量，若该变量的***k***个取值中至少有两个取值相同的概率大于**0.5**，则***k***至少多大？

$$P(n,k)=1-\frac{n!}{(n-k)!n^k}$$

  - 与上类似，$k=1.18\sqrt{n}\approx\sqrt{n}$ ，令***P*(*n*, *k*)>0.5**，可得

  - 若取***n*=365**，则 $k=1.18\sqrt{365}=22.54$。

生日攻击

- 生日攻击
  - 设杂凑函数H有2^m个可能的输出（即输出长m比特），如果H的k个随机输入中至少有两个产生相同输出的概率大于0.5，则

$$k \approx \sqrt{2^m} = 2^{m/2}$$

  - 第II类生日攻击：寻找函数H的具有相同输出的两个任意输入的攻击方式。

第Ⅱ类生日攻击

$$A \xrightarrow{M \| E_{sk_A}(H(M))} B$$

敌手对M产生出$2^{m/2}$个变形消息：$\tilde{M}$

$$H(\tilde{M}) = H(\tilde{\tilde{M}})$$

敌手还准备一个假冒的消息M′，
产生出$2^{m/2}$个变形的消息：$\tilde{\tilde{M}}$

将$\tilde{M}$提交给A请求签名 $E_{sk_A}(H(\tilde{M}))$

$$A \xrightarrow{\tilde{\tilde{M}} \| E_{sk_A}(H(\tilde{M}))} B$$

安全应用

- 输出长度与碰撞
  - 这种生日攻击给出了消息摘要长度的下界，一个40比特的消息摘要非常不安全的。
  - 因为在大约2^{20}个（大约100万）个随机值中就能以1/2的概率找到一个碰撞。
  - 通常建议消息摘要的最小可接受的长度为160比特，在DSS签名标准中使用160比特的消息摘要就是基于这个考虑。

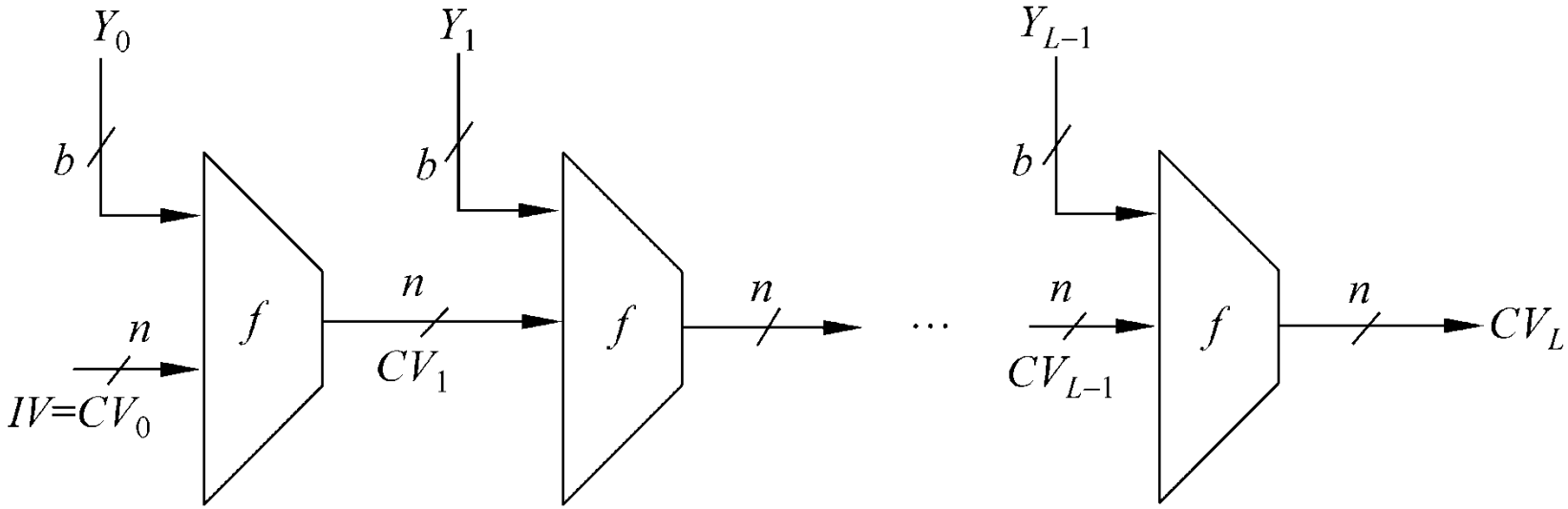
现代密码学

SHA-1密码杂凑函数

信息与软件工程学院

迭代型哈希函数的一般结构

- 迭代



CV_0=IV=n比特长的初值；
$CV_i=f(CV_{i-1},Y_{i-1})1\leq i\leq L$；
$H(M)=CV_L$

CV_{i-1}，称为链接变量
通常b>n，称函数f为压缩函数
分析时需要先找出f 的碰撞

MD5哈希算法

- **MD4是MD5哈希算法的前身，由Ron Rivest于1990年10月作为RFC提出，1992年4月公布的MD4的改进（RFC 1320，1321）称为MD5。**
- 输入：任意长的消息（图中为K比特）
- 分组：512比特
- 输出：128比特的消息摘要。

CRYPTO'04 会议中,我国学者王小云
在Aug.17 (19:00pm—24:00pm)

Rump Session Program

给出15分钟的演讲,
对 MD5 给出了有效地攻击!

MD5的安全性

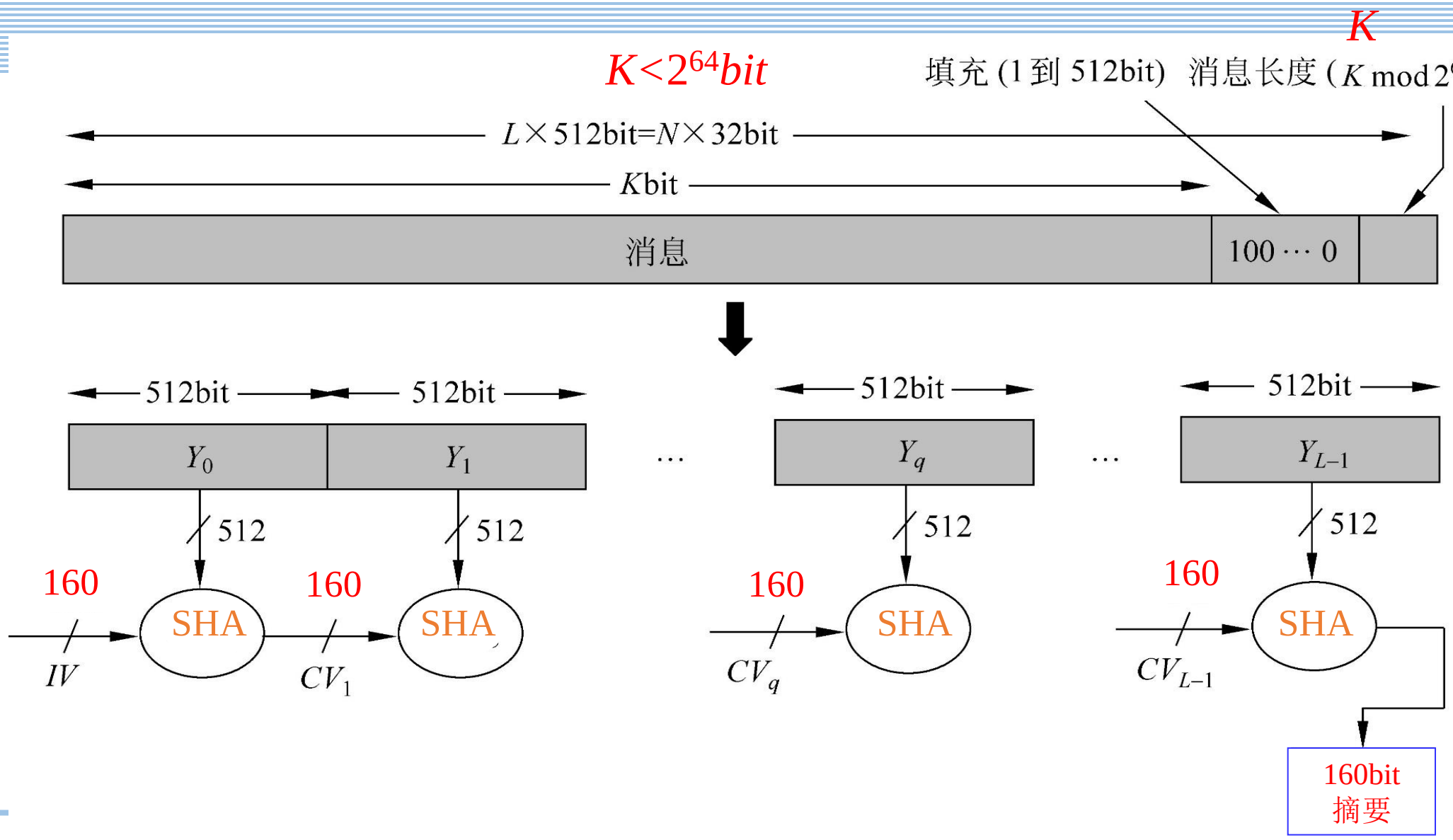
- Rivest猜想作为128比特长的哈希值来说，MD5的强度达到了最大，比如说找出具有相同哈希值的两个消息需执行$O(2^{64})$次运算，而寻找具有给定哈希值的一个消息需要执行$O(2^{128})$次运算。
- 然而，2004年，山东大学王小云等成功找出了MD5的碰撞，发生碰撞的消息是由两个1024比特长的串 M 、N_i 构成，设消息 $M\|N_i$ 的碰撞是 $M'\|N_i'$ ，在IBM P690上找 M 和 M' 花费时间大约一小时，找出 M 和 M' 后，则只需15秒至5分钟就可找出 N_i 和 N_i' 。

安全杂凑算法SHA-1

- 安全哈希算法(Secure Hash Algorithm, SHA)由美国NIST设计，于1993年作为联邦信息处理标准公布。SHA是基于MD4的算法，其结构与MD4非常类似。
- SHA是基于MD4的算法，其结构与MD4非常类似
  - SHA在发布之后很快就被 NSA 撤回，并且以 1995年发布的修订版本 FIPS PUB 180-1（通常称为 “SHA-1”）取代
  - 根据 NSA 的说法，它修正了一个在原始算法中会降低密码安全性的错误。然而 NSA 并没有提供任何进一步的解释或证明该错误已被修正
    - 1998年，在一次对 SHA-0 的攻击中发现这次攻击并不能适用于 SHA-1 — 我们不知道这是否就是 NSA 所发现的错误，但这或许暗示我们这次修正已经提升了安全性。
    - SHA-1 已经被公众密码社群做了非常严密的检验而还没发现到有不安全的地方，
    - 但现在理论上已经被王小云破译了

算法描述

- 算法的输入：小于2^{64}比特长的任意消息，分为512比特长的分组。
- 算法的输出：160比特长的消息摘要。
- 算法的框图与MD5一样，但杂凑值的长度和链接变量的长度为160比特


K
K<2^64bit
填充 (1 到 512bit)
消息长度 (K mod 2^64)
L×512bit=N×32bit
Kbit
消息
100 … 0
512bit
512bit
512bit
512bit
Y0
Y1
Yq
YL–1
512
512
512
512
160
160
160
160
SHA
SHA
SHA
SHA
IV
CV1
CVq
CVL–1
160bit
摘要

算法处理步骤---①对消息填充

- 对消息填充，使得其比特长在模512下为448，即填充后消息的长度为512的某一倍数减64，留出的64比特备第② 步使用。
- 步骤①是必需的，即使消息长度已满足要求，仍需填充。例如，消息长为448比特，则需填充512比特，使其长度变为960，因此填充的比特数大于等于1而小于等于512。
- 填充方式是:第1位为1，其后各位皆为0。

算法处理步骤---②附加消息的长度

- 留出的64比特以来表示消息被填充前的长度。如果消息长度大于2^{64}，则以2^{64}为模数取模。

- big-endian模式是指数据的高字节保存在内存的低地址中，反之为little-endian模式

算法处理步骤-- ③对MD缓冲区初始化

- 使用160比特长的缓冲区存储中间结果和最终杂凑值。

缓冲区为5个32比特以big-endian方式存储数据的寄存器(A，B，C，D，E)初始值分别为A=67452301，

B=EFCDAB89，

C=98BADCFB，

D=10325476，

E=C3D2E1F0。

算法处理步骤--④以分组为单位对消息进行处理

- 每一分组Y_q都经一压缩函数处理，压缩函数由4轮处理过程构成，每一轮又由20步迭代组成。
- 第4轮的输出（即第80步迭代的输出）再与第1轮的输入CV_q相加，以产生CV_{q+1}，其中加法是缓冲区5个字中的每一个字与CV_q中相应的字模2^{32}相加。

算法处理步骤--⑤ 输出消息

- 输出消息的L个分组都被处理完后，最后一个分组的输出即为160比特的消息摘要。

总结---③到⑤的处理过程

$CV_0=IV;$

$CV_{q+1}=SUM_{32}(CV_q,\ ABCDE_q);$

$MD=CV_L$

IV：缓冲区ABCDE的初值。

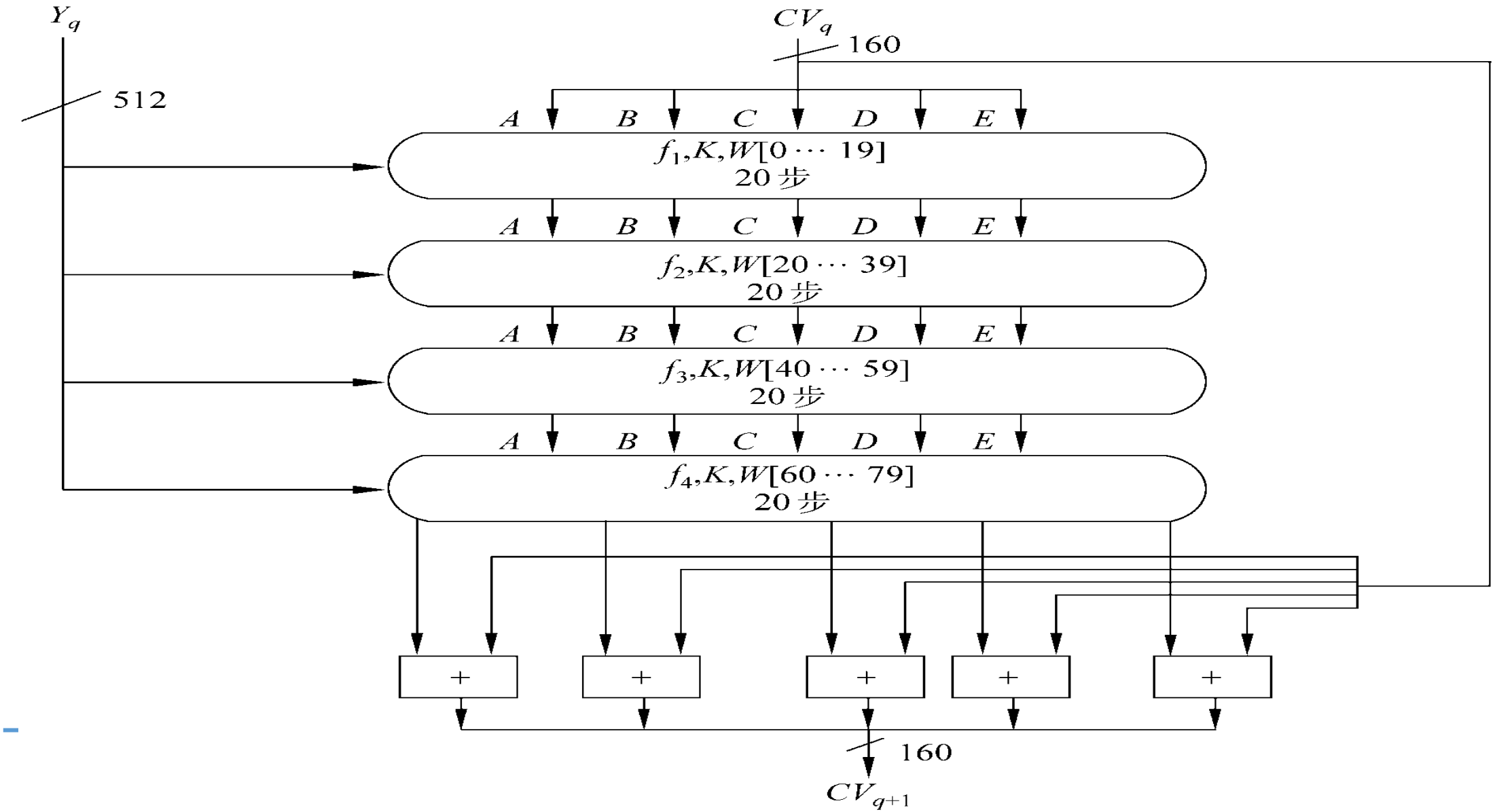
$ABCDE_q$：第q个消息分组经最后一轮处理过程处理后的输出

L：消息(包括填充位和长度字段)的分组数

SUM_{32}：对应字的模2^{32}加法

MD：最终的摘要值。

SHA的分组处理框图


Y_q
512
CV_q
160
A B C D E
$f_1, K, W[0 \cdots 19]$
20 步
A B C D E
$f_2, K, W[20 \cdots 39]$
20 步
A B C D E
$f_3, K, W[40 \cdots 59]$
20 步
A B C D E
$f_4, K, W[60 \cdots 79]$
20 步
+
+
+
+
+
160
CV_{q+1}

SHA的压缩函数

- SHA的压缩函数由4轮处理过程组成，每轮处理过程20步迭代运算组成，每一步迭代运算的形式为

$$A,B,C,D,E \leftarrow (E + f_t(B,C,D) + CLS_5(A) + W_t + K_t),\ A,\ CLS_{30}(B),\ C,\ D$$

A，B，C，D，E：缓冲区的5个字

t：迭代的步数（0≤t≤79）

$f_t(B,C,D)$：第t步迭代使用的基本逻辑函数

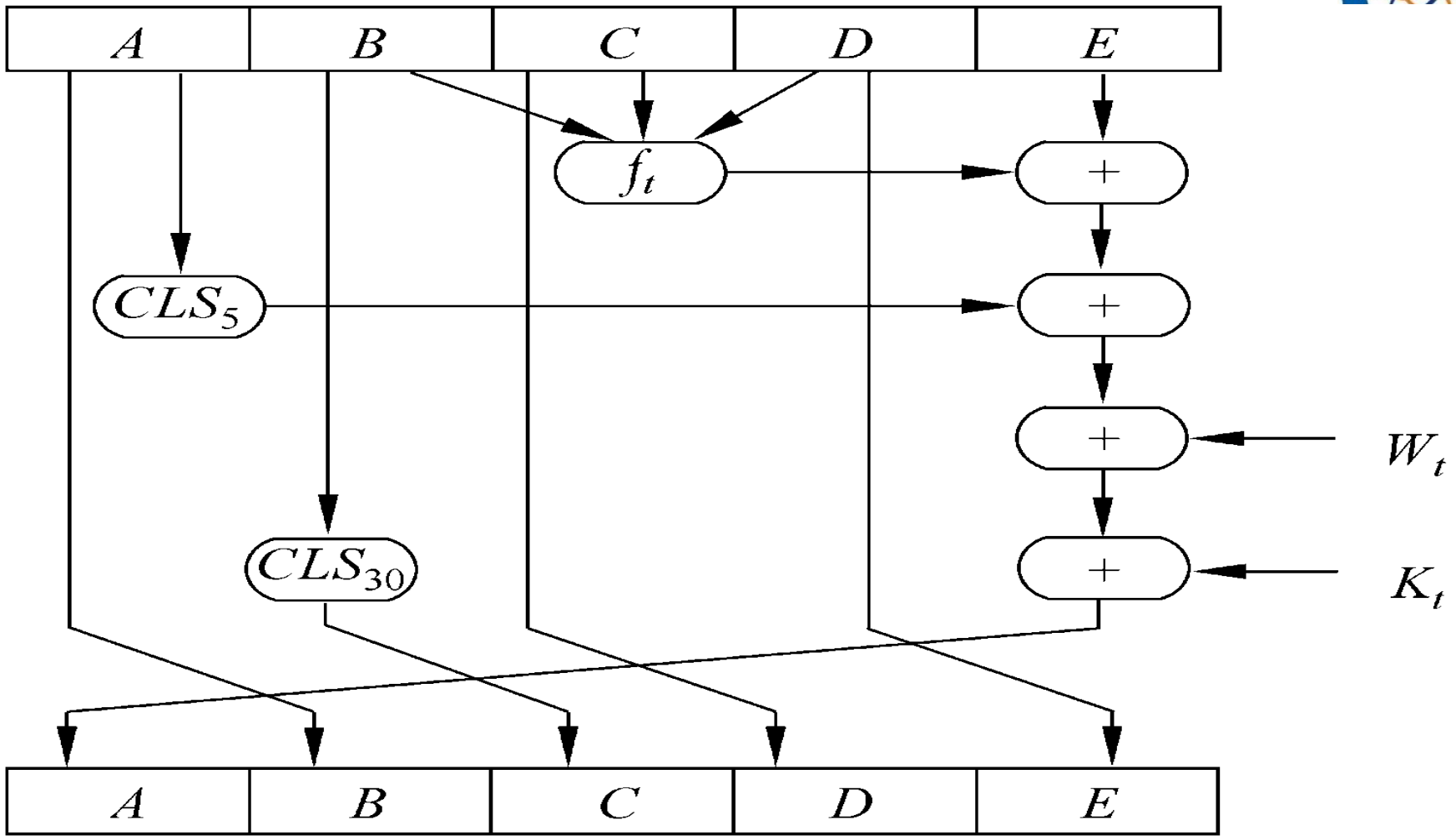
CLSs：左循环移s位

W_t：由当前512比特长的分组导出的一个32比特长的字

K_t：加法常量

+：模2^{32}加法。

一步迭代示意图



$$A, B, C, D, E \leftarrow (E + f_t(B, C, D) + CLS_5(A) + W_t + K_t),\ A,\ CLS_{30}(B),\ C,\ D$$

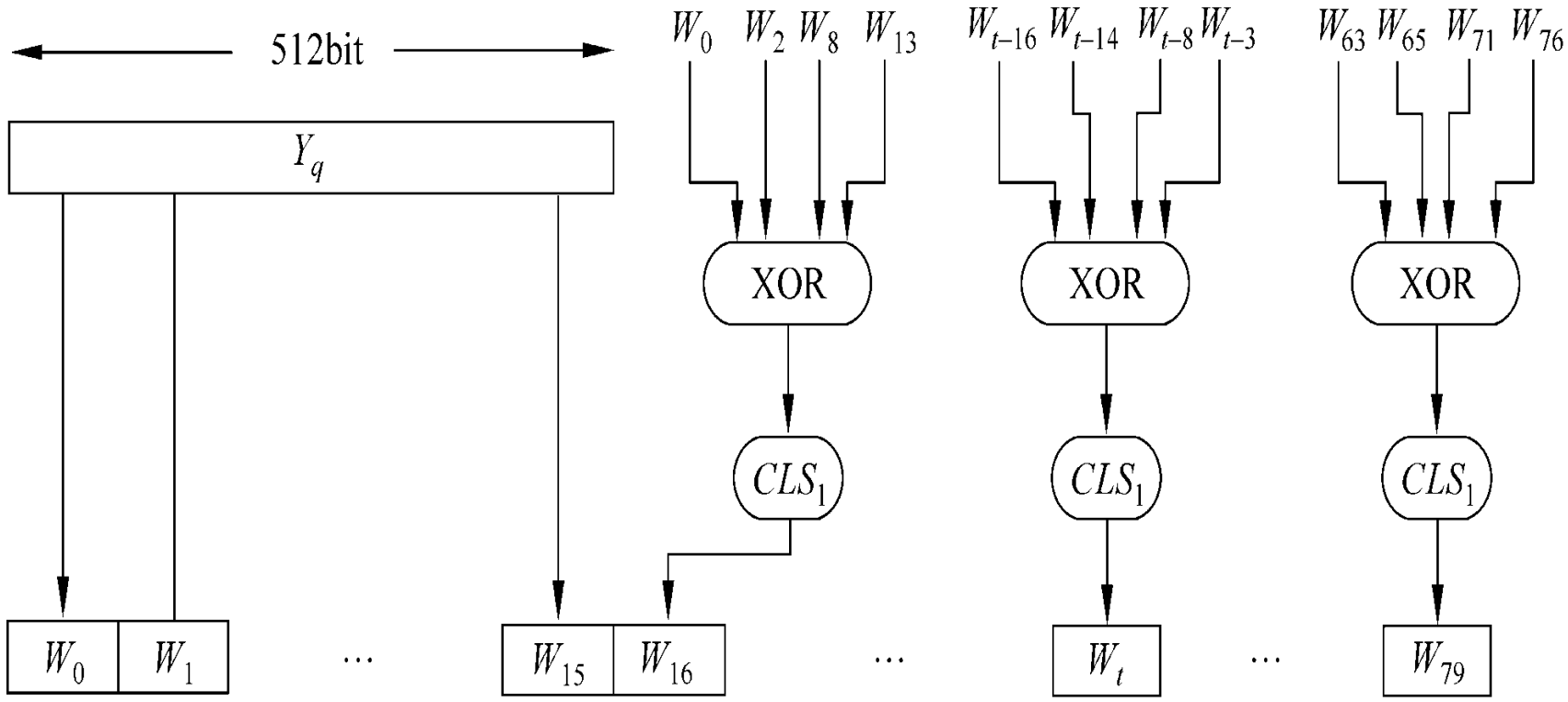
基本逻辑函数f_t

迭代的步数	函数名	定义
$0 \le t \le 19$	$f_1 = f_t(B,C,D)$	$(B \wedge C) \vee (\overline{B} \wedge D)$
$20 \le t \le 39$	$f_2 = f_t(B,C,D)$	$B \oplus C \oplus D$
$40 \le t \le 59$	$f_3 = f_t(B,C,D)$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
$60 \le t \le 79$	$f_4 = f_t(B,C,D)$	$B \oplus C \oplus D$

- 基本逻辑函数的输入为3个32比特的字，输出是一个32比特的字。表中$\wedge$，$\vee$，$-$，$\oplus$分别是与、或、非、异或4个逻辑运算。

SHA分组处理所需的80个字的产生过程

$$W_t = CLS_1(W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3})$$



常量字K_t

- 常量字K_0, K_1,...,K_{79}, 如果以16进制给出。它们如下：

$$K_t = 0x5A827999 \quad (0 <= t <= 19)$$
$$K_t = 0x6ED9EBA1 \quad (20 <= t <= 39)$$
$$K_t = 0x8F1BBCDC \quad (40 <= t <= 59)$$
$$K_t = 0xCA62C1D6 \quad (60 <= t <= 79).$$

SHA系列

1、 SHA系列Hash函数

- SHA系列标准Hash函数是由美国标准与技术研究所(NIST)组织制定的。
- 1993年公布了SHA-0 (FIPS PUB 180)，后发现不安全。
- 1995年又公布了SHA-1（FIPS PUB 180-1）。
- 2002年又公布了SHA-2（FIPS PUB 180-2）：
  - SHA-256
  - SHA-384
  - SHA-512
- 2005年中国王小云给出一种攻击SHA-1的方法，用2^{69}操作找到一个强碰撞，以前认为是2^{80}。

SHA参数比较

SHA参数比较

	SHA-1	SHA-256	SHA-384	SHA-512
Hash码长度	160	256	384	512
消息长度	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
分组长度	512	512	1024	1024
字长度	32	32	64	64
迭代步骤数	80	64	80	80
安全性	80	128	192	256

注:1、所有的长度以比特为单位。

2、安全性是指对输出长度为n比特hash函数的生日攻击产生碰撞的工作量大约为$2^{n/2}$

王小云院士

- 王小云，清华大学高等研究院“杨振宁讲座”教授，中国科学院院士，国际密码协会会士(IACR Fellow)，中国密码学会副理事长。
- 国际密码学会（IACR）公布了2020年“最具时间价值奖”（Test-of-Time Awards）的获奖论文，王小云院士以第一作者身份在2005年发表于美密会（Crypto）上的论文，因在哈希函数分析上取得的突破性成果而获奖。
  - “最具时间价值奖”是国际密码协会（IACR）自2019年起设立的，该奖项每年会从15年前的三大密码学会会议欧密会（Eurocrypt）、美密会（Crypto）和亚密会（Asiacrypt）中各选出一篇论文，以表彰他们对密码学领域产生的持久影响和重要贡献。
- 王小云院士今年1月还获得了“真实世界密码学奖”
- （The Levchin Prize for Real-World Cryptography）“真实世界密码学奖”是2015年互联网企业家马克斯·莱文奇恩（Max Levchin）创立的。该奖项旨在表彰对真实世界密码学作出重大贡献，和在密码学实践及系统应用中取得了具有重大影响的研究进展。每年最多颁发两个奖项。

ICS 35.040
L 80
备案号:36831—2012

GM

中华人民共和国密码行业标准

GM/T 0004—2012

SM3 密码杂凑算法

SM3 cryptographic hash algorithm

2012-03-21 发布　　2012-03-21 实施

国家密码管理局　　发布

SM3密码杂凑算法

- SM3是中国国家密码管理局颁布的中国商用公钥密码标准算法，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/T 0004-2012）
- 2016年8月，成为中国国家密码标准（GB/T 32905-2016）
- 2018年11月22日，含有我国SM3杂凑密码算法的ISO/IEC 10118 – 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密码杂凑算法的描述

算法的输入数据长度为*l*比特，$l < 2^{64}$,输出哈希值长度为256比特。

1. 常数与函数

(1) 常数

初始值

$$IV = 7380166F4914B2B9$$
$$172442D7DA8A0600$$
$$A96F30BC163138AA$$
$$E38DEE4DB0FB0E4E$$

常量

$$T_j = \begin{cases} 79CC4519, & 0 \le j \le 15 \\ 7A879D8A, & 16 \le j \le 63 \end{cases}$$

SM3密码杂凑算法的描述

(2) 函数

式中 X,Y,Z 为**32**位字，$\wedge,\vee,-,\oplus$ 分别是逻辑与、逻辑或、逻辑非和逐比特异或运算

布尔函数：

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

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

混淆

置换函数：

$$P_0(X)=X\oplus(X<<<9)\oplus(X<<<17);$$

$$P_1(X)=X\oplus(X<<<15)\oplus(X<<<23).$$

扩散

式中 X 为**32**位字，符号 $a<<<n$ 表示把 a 循环左移 n 位。

SM3密码杂凑算法的描述

2. 算法描述

算法对数据首先进行填充，再进行迭代压缩后生成哈希值。

(1) 填充并附加消息的长度

- 对消息填充的目的是使填充后的数据长度为512的整数倍。
- 设消息m的长度为l比特。首先将比特“1”添加到m的末尾，再添加k个“0”，其中k满足

$$l+1+k=448 \bmod 512$$

- 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。

举例：消息为01100001011000100110001 1，其长度为l=24，填充后的比特串为

$$01100001011000100110001\underbrace{1\,00\cdots00}_{423\text{个}0}\underbrace{00\cdots011000}_{64\text{比特}}$$

SM3密码杂凑算法的描述

(2) 迭代压缩

将填充后的消息 m' 按512比特进行分组得 $m' = B^{(0)}B^{(1)}\cdots B^{(L-1)}$ 对 m'按下列方式迭代压缩：

$$\text{FOR } i=0 \text{ to } L-1 \quad V^{(i+1)} = CF(V^{(i)}, B^{(i)})$$

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 $V^{(L)}$，$V^{(L)}$ 即为消息 m 的哈希值。

SM3密码杂凑算法的描述

(3) 消息扩展

在对消息分组 $B^{(i)}$ 进行迭代压缩之前，首先对其进行消息扩展，步骤如下：

① 消息分组 $B^{(i)}$ 划分为**16**个字 $W_0, W_1, \cdots, W_{15}$。

② FOR j=16 to 67

$$W_j = P_1\left(W_{j-16} \oplus W_{j-9} \oplus \left(W_{j-3} <<< 15\right)\right) \oplus \left(W_{j-13} <<< 7\right) \oplus W_{j-6}$$

③ FOR j=0 to 63

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

$B^{(i)}$ 经消息扩展后得到**132**个字 $W_0, W_1, \cdots, W_{67}, W'_0, W'_1, \cdots, W'_{63}$。

SM3密码杂凑算法的描述

(4) 压缩函数

设 A,B,C,D,E,F,G,H 为字寄存器，SS_1,SS_2,TT_1,TT_2 为中间变量，压缩函数 $V^{(i+1)}=CF(V^{(i)},B^{(i)})\ (0\le i\le n-1)$ 的计算过程如下：

$ABCDEFGH=V^{(i)}$

FOR j=0 to 63

$SS_1\leftarrow\left((A<<<12)+E+\left(T_j<<<j\right)\right)<<<7;$

$SS_2=SS_1\oplus(A<<<12);$

$TT_1=FF_j(A,B,C)+D+SS_2+W_j';$

$TT_2=GG_j(E,F,G)+H+SS_1+W_j;$

$D=C;$

$C=B<<<9;$

$B=A;$

$A=TT_1;$

$H=G;$

$G=F<<<19;$

$F=E;$

$E=P_0(TT_2)$

ENDFOR

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

其中 + 为模 2^{32} 加运算，字的存储为大端格式

SM3密码杂

$ABCDEFGH = V^{(i)}$

FOR j=0 to 63

$SS_1 \leftarrow \left((A <<< 12) + E + \left(T_j <<< j\right)\right) <<< 7;$

$SS_2 = SS_1 \oplus (A <<< 12);$

$TT_1 = FF_j(A,B,C) + D + SS_2 + W'_j;$

$TT_2 = GG_j(E,F,G) + H + SS_1 + W_j;$

$D = C;$

$C = B <<< 9;$

$B = A;$

$A = TT_1;$

$H = G;$

$G = F <<< 19;$

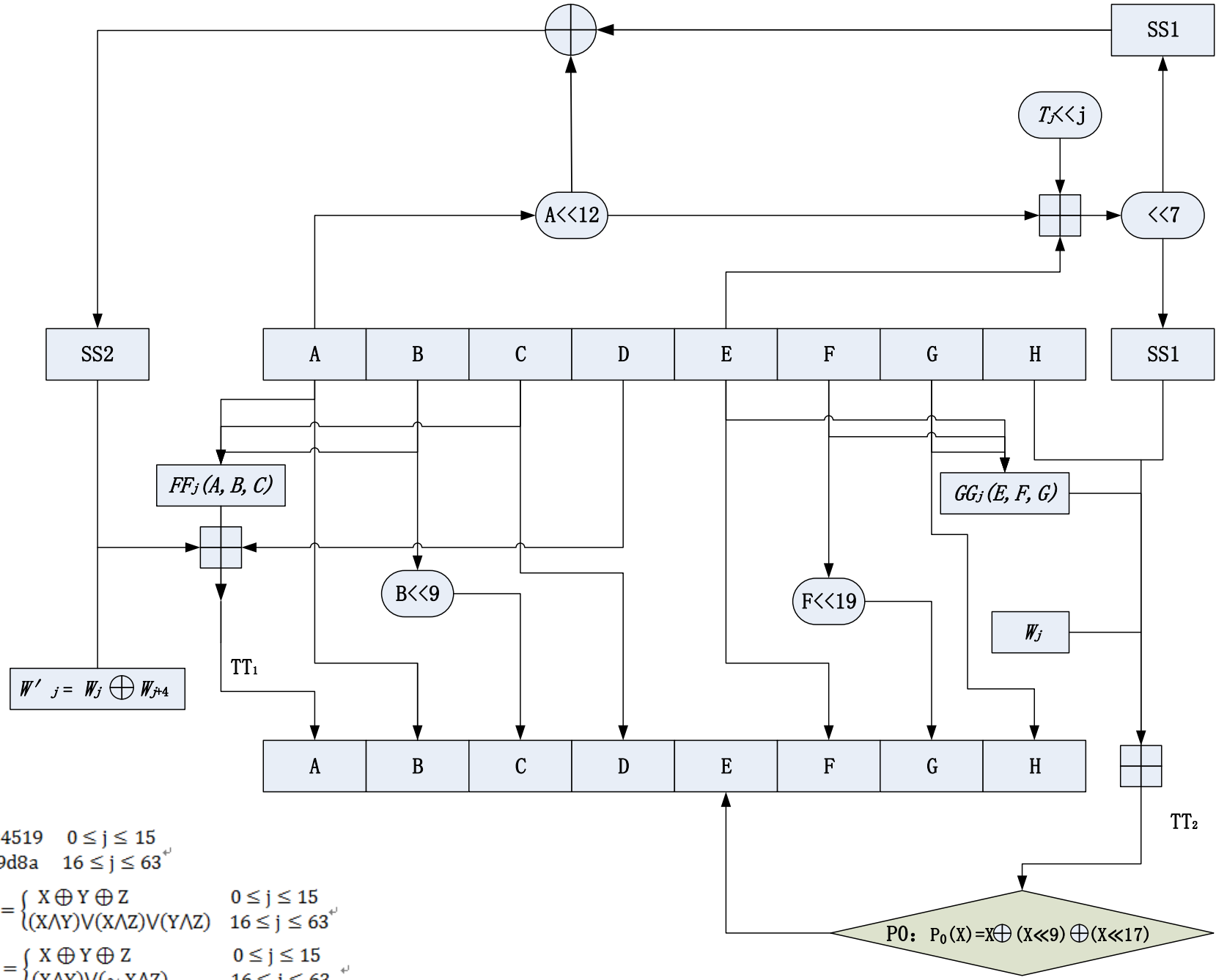
$F = E;$

$E = P_0(TT_2)$

$ENDFOR$

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

SM3算法第j轮迭代压缩流程图



其中：

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

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

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

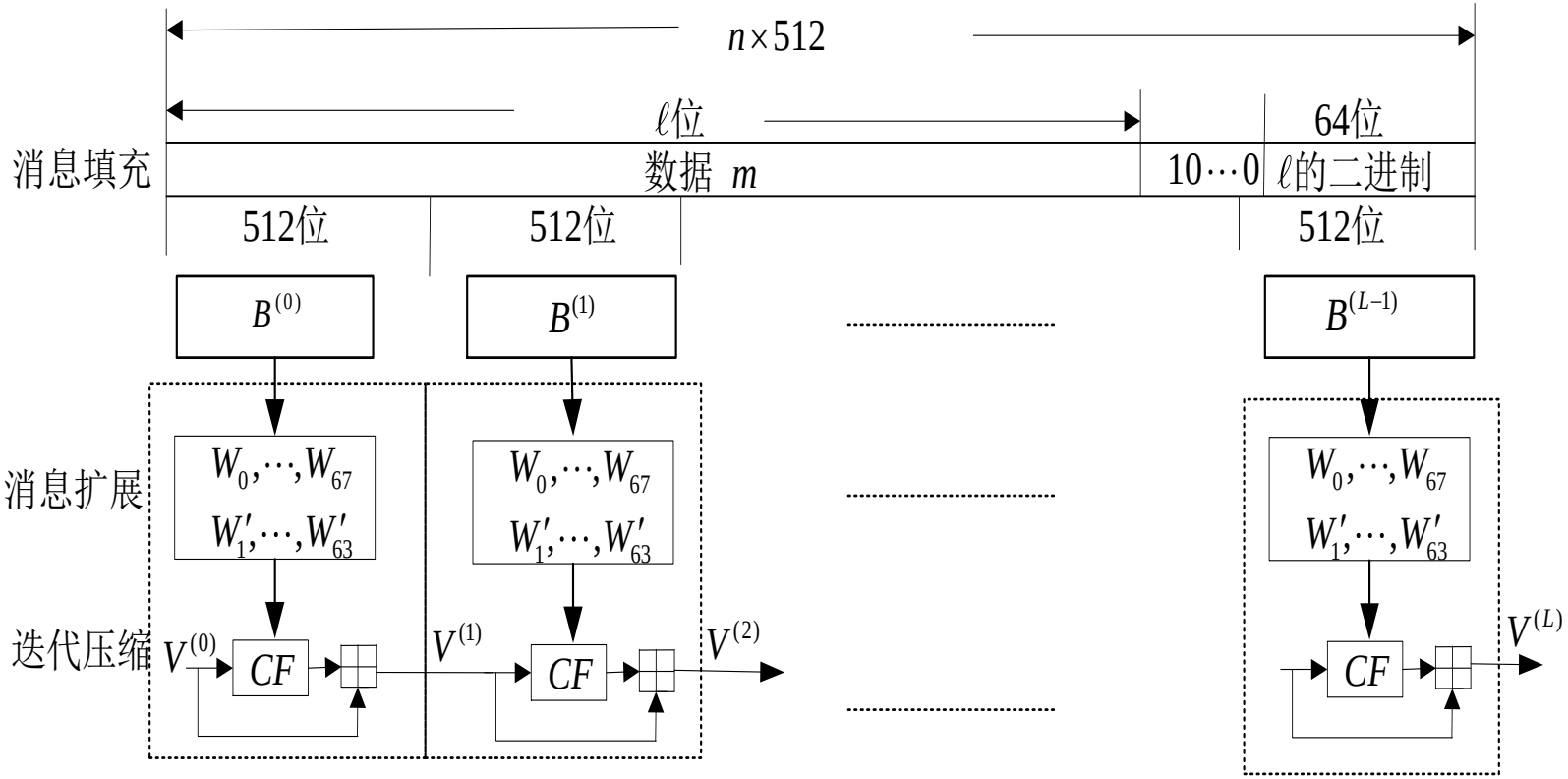
SM3密码杂凑算法的描述

(5) 输出哈希值

$$ABCDEFGH = V^{(L)}$$

输出 256 比特的哈希值 $y = ABCDEFGH$。

SM3密码杂凑算法的描述



SM3 产生消息哈希值的处理过程

SM3哈希算法的安全性

- 压缩函数是哈希函数安全的关键
  - SM3的压缩函数 CF 中的布尔函数 $FF_j(X,Y,Z)$ 和 $GG_j(X,Y,Z)$ 是非线性函数，经过循环迭代后提供混淆作用
  - 置换函数 $P_0(X)$ 和 $P_1(X)$ 是线性函数，经过循环迭代后提供扩散作用。
  - 再加上 CF 中的其他运算的共同作用，压缩函数 CF 具有很高的安全性，从而确保SM3具有很高的安全性。

算法比较

王小云, 于红波. SM3密码杂凑算法[J]. 信息安全研究, 2016(11).

表 2　SM3 密码杂凑算法和其他标准的 ASIC 实现

算法	面积 (gates)	时钟 /MHz	吞吐量 /Mbps	吞吐量面积比 /(Kbps/gate)
SM3[10]	11 068	216.00	1 619	146.28
SHA-256[11]	15 400	189.75	1 349	87.60
SHA-512[11]	30 747	169.20	1 969	64.04
Whirlpool[11]	38 911	101.94	2 485	63.86
SHA-3[12]	56 320	487.80	21 229	376.94

表 5　SM3 密码杂凑算法和其他杂凑标准的最好分析结果

算法	攻击类型	步(轮)数	百分比/%	文献
SM3	碰撞攻击	20	31	[18]
	原像攻击	30	47	[24-25]
	区分器攻击	37	58	[27]
SHA-1	碰撞攻击	80	100	[4,28-29]
	原像攻击	62	77.5	[30]
RIPEMD-128	碰撞攻击	40	62.5	[31]
	原像攻击	36	56.25	[32]
	区分器攻击	64	100	[33]
RIPEMD-160	原像攻击	34	53.12	[34]
	区分器攻击	51	79.68	[35]
SHA-256	碰撞攻击	31	48.4	[36]
	原像攻击	45	70.3	[23]
	区分器攻击	47	73.4	[37]
Whirlpool	碰撞攻击	8	80	[38]
	原像攻击	6	60	[38]
	区分器攻击	10	100	[39]
Stribog	碰撞攻击	7.5	62.5	[40]
	原像攻击	6	50	[41]
KECCAK-256	碰撞攻击	5	20.8	[42]
	原像攻击	2	8	[43]
	区分器攻击	24	100	[44]
KECCAK-512	碰撞攻击	3	12.5	[42]
	区分器攻击	24	100	[44]

现代密码学

HMAC算法

信息与软件工程学院

HMAC算法

- 以前曾介绍过一个MAC的例子——数据认证算法
  - 该算法反映了传统上构造MAC最为普遍使用的方法，即基于分组密码的构造方法
- 近年来研究构造MAC的兴趣已转移到基于密码哈希函数的构造方法，这是因为：
  - 哈希函数(如MD5,SHA)软件实现快于分组密码(如DES)的软件实现
  - 哈希函数的库代码来源广泛
  - 哈希函数没有出口限制，而分组密码即使用于MAC也有出口限制
- 哈希函数并不是为用于MAC而设计的，由于哈希函数不使用密钥，因此不能直接用于MAC
  - 目前已提出了很多将哈希函数用于构造MAC的方法，其中HMAC就是其中之一，已作为RFC2104被公布，并在IPSec和其他网络协议(如SSL)中得以应用

HMAC的设计目标

- RFC2104列举了HMAC的以下设计目标：
  - ① 可不经修改而使用现有的哈希函数，特别是那些易于软件实现的、源代码可方便获取且免费使用的哈希函数。
  - ② 其中镶嵌的哈希函数可易于替换为更快或更安全的哈希函数。
  - ③ 保持镶嵌的哈希函数的最初性能，不因用于HMAC而使其性能降低。
  - ④ 以简单方式使用和处理密钥。
  - ⑤ 在对镶嵌的哈希函数合理假设的基础上，易于分析HMAC用于认证时的密码强度。

- 前两个目标是HMAC被公众普遍接受的主要原因，这两个目标是将哈希函数当作一个黑盒使用，这种方式有两个优点：
  - 第一，哈希函数的实现可作为实现HMAC的一个模块，这样一来，HMAC代码中很大一块就可事先准备好，无需修改就可使用；
  - 第二，如果HMAC要求使用更快或更安全的哈希函数，则只需用新模块代替旧模块，例如用实现SHA的模块代替MD5的模块。
- 最后一条设计目标则是HMAC优于其他基于哈希函数的MAC的一个主要方面，HMAC在其镶嵌的哈希函数具有合理密码强度假设下，可证明是安全的

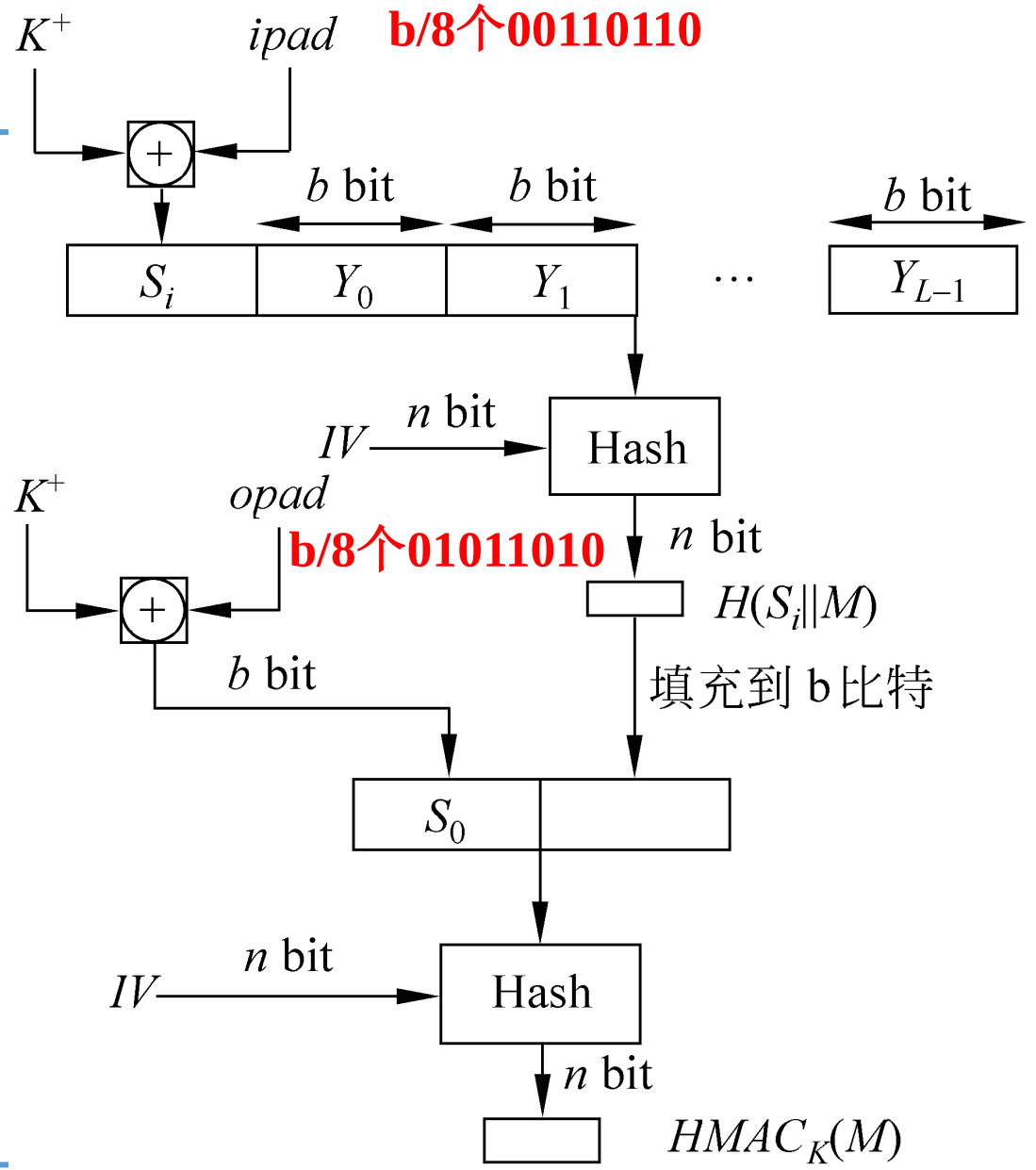
- 算法的输出可如下表示：

$$HMAC_k = H[(K^+ \oplus opad) || H[(K^+ \oplus ipad) || \mathbf{M}]]$$

HMAC算法描述

后面经填充0后的
K，长度为b比特

- 右图是HMAC算法的运行框图
  - H为嵌入的哈希函数(如MD5、SHA)，
  - M为HMAC的输入消息(包括哈希函数所要求的填充位)，
  - $Y_i(0 \le i \le L-1)$是M的第i个分组，
  - L是M的分组数，
  - b是一个分组中的比特数，
  - n为由嵌入的哈希函数所产生的哈希值的长度，
  - K为密钥，如果密钥长度大于b，则将密钥输入到哈希函数中产生一个n比特长的密钥，
  - K^+是后面经填充0后的K，K^+的长度为b比特，
  - $ipad$为b/8个00110110，
  - $opad$为b/8个01011010。

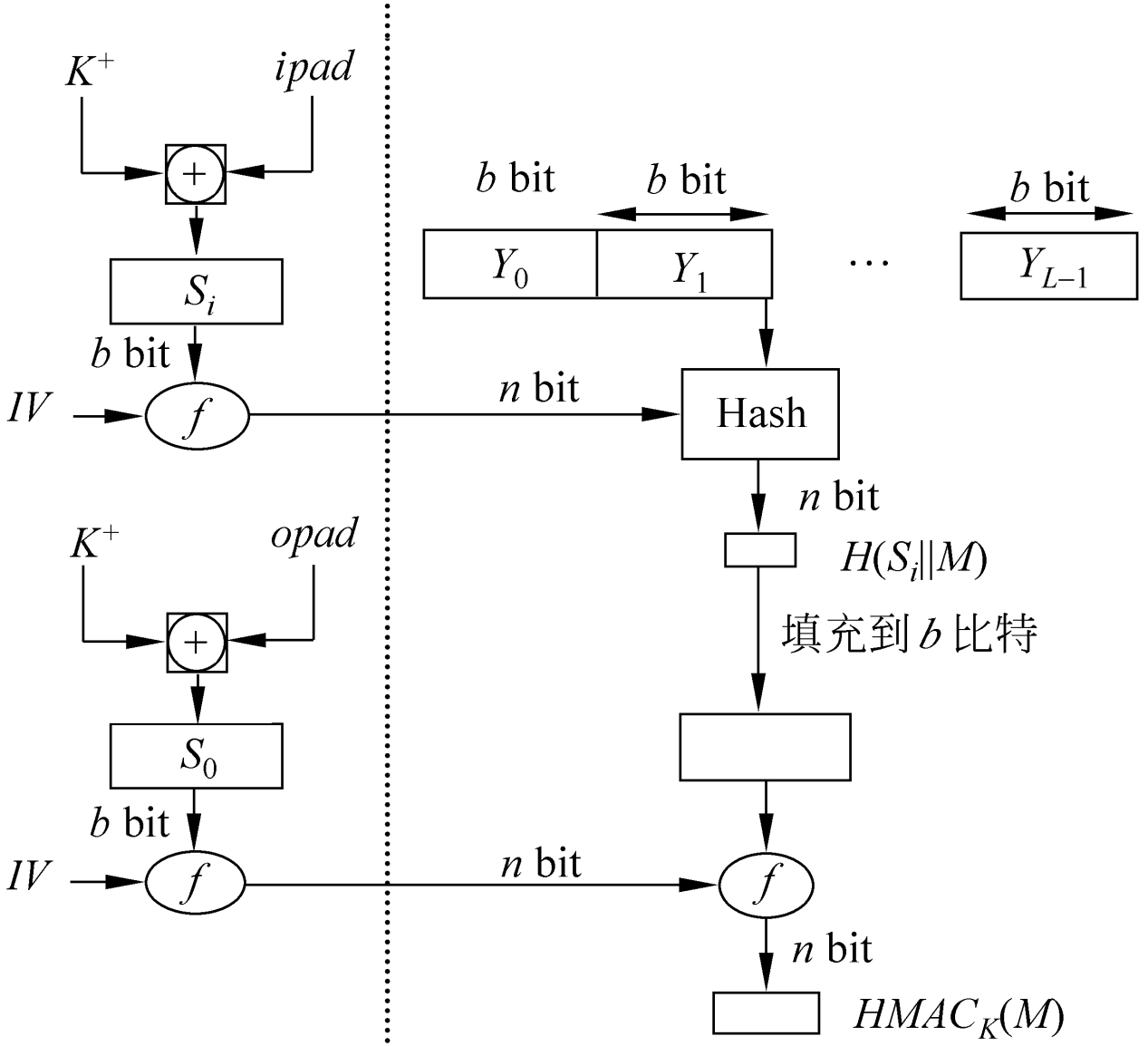


- 算法的输出可如下表示：
  - $HMAC_k = H[(K^+ \oplus opad) || H[(K^+ \oplus ipad) || M]]$
- 算法的运行过程可描述如下：
  - ① K的后面填充0以产生一个b比特长的K^+（例如K的长为160比特，b=512，则需填充　　个零字节0x00）。

- 算法的输出可如下表示：
  - $HMAC_k = H[(K^+ \oplus opad) || H[(K^+ \oplus ipad) || M]]$
- 算法的运行过程可描述如下：
  - ① K的左边填充0以产生一个b比特长的K^+（例如K的长为160比特，b=512，则需填充44个零字节0x00）。
  - ② K^+与$ipad$逐比特异或以产生b比特的分组S_i。
  - ③ 将M链接到S_i后。
  - ④ 将H作用于步骤③产生的数据流。
  - ⑤ K^+与$opad$逐比特异或，以产生b比特长的分组S_0。
  - ⑥ 将步骤④得到的哈希值链接在S_0后。
  - ⑦ 将H作用于步骤⑥产生的数据流并输出最终结果。

- 注意
  - K^+与*ipad*逐比特异或以及K^+与*opad*逐比特异或的结果是将K中的一半比特取反，但两次取反的比特的位置不同
  - 而S_i和S_0通过哈希函数中压缩函数的处理，则相当于以伪随机方式从K产生两个密钥

- 在实现HMAC时，可预先求出下面两个量（见图，虚线以左为预计算）：
  - ***f*(*IV*, (*K*⁺⊕*ipad*))**
  - ***f*(*IV*, (*K*⁺⊕*opad*))**
  - 其中***f*(*cv*，*block*)**是哈希函数中的压缩函数，
    - 其输入是***n***比特的链接变量和***b***比特的分组
    - 输出是***n***比特的链接变量。
  - 这两个量的预先计算只在每次更改密钥时才需进行。事实上这两个预先计算的量用于作为哈希函数的初值**IV**。



HMAC的安全性

- 基于密码哈希函数构造的MAC的安全性取决于镶嵌的哈希函数的安全性
- HMAC最吸引人的地方是
  - 其设计者已证明了算法强度和嵌入的散列函数强度之间的确切关系
  - 即对HMAC的攻击等价于对内嵌哈希函数的下述两种攻击之一：
- ① 攻击者能够计算压缩函数的一个输出，即使IV是随机的和秘密的
  - 在第一种攻击中，可将压缩函数视为与哈希函数等价，而哈希函数的n比特长IV可视为HMAC的密钥。
  - 对这一散列函数的攻击可通过对密钥的穷搜索来进行
  - 攻击的复杂度为$O(2^n)$

- ② 攻击者能够找出哈希函数的碰撞，即使IV是随机的和秘密的
  - 第二种攻击指攻击者寻找具有相同哈希值的两个消息，因此就是第II类生日攻击。
  - 对哈希值长度为n的哈希函数来说，攻击的复杂度为$O(2^{n/2})$。
  - 因此第二种攻击对MD5的攻击复杂度为$O(2^{64})$
  - 就现在的技术来说，这种攻击是可行的。但这是否意味着MD5不适合用于HMAC？

- 回答是否定的，原因如下：
  - 攻击者在攻击**MD5**时，可选择任何消息集合后离线寻找碰撞。由于攻击者知道哈希算法和默认的**IV**，因此能为自己产生的每个消息求出哈希值。
  - 然而，在攻击**HMAC**时，由于攻击者不知道密钥***K***，从而不能离线产生消息和认证码对
- 所以攻击者必须得到HMAC在同一密钥下产生的一系列消息，并对得到的消息序列进行攻击。
  - 对长**128**比特的哈希值来说，需要得到用同一密钥产生的2^{64}个分组（2^{73}比特）。在**1Gbit/s**的链路上，需**250000**年，因此**MD5**完全适合于**HMAC**，而且就速度而言，**MD5**要快于**SHA**作为内嵌哈希函数的**HMAC**。

感 谢 聆 听!

liaoyj@uestc.edu.cn