Section 2.3 Hash and Message Authentication Code

- 网络安全威胁:
  - 被动攻击（Passive Attack）：敌手通过侦听和截获等手段获取数据;
  - 主动攻击（Active Attack）：敌手通过伪造、重放、篡改、乱序等手段改变数据;

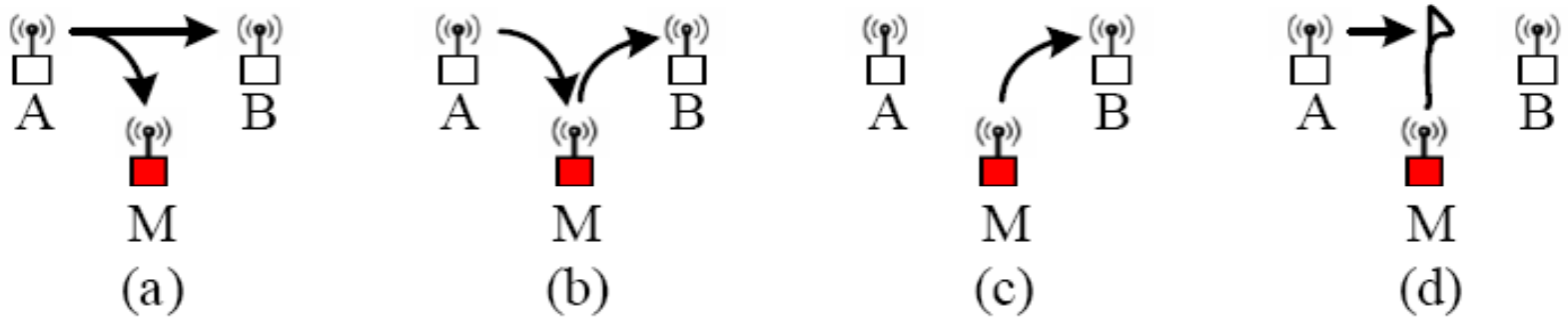


图1 无线网络中的四种通信安全威胁

（a）监听；（b）篡改；（c）伪造；（d）阻断

- 消息认证（Message Authentication）的目的:
  - 验证消息的完整性，确认数据在传送和存储过程中未受到主动攻击。

- 消息认证的方式:
  - 消息**加密函数**: 加密整个消息，以消息的密文文件作为认证，可使用对称密码或公钥密码体制进行加密;
  - **散列函数（Hash）**：将任意长度的消息变换为定长的消息摘要，并加以认证;
  - **消息认证码（MAC）**：依赖公开的函数（密钥控制下）对消息处理，生成定长的认证标识，并加以认证;

提纲

- 1 散列函数
  - 1.1 散列函数的定义
  - 1.2 散列函数的通用结构
  - 1.3 MD5
- 2 消息认证码
  - 2.1 MAC函数
  - 2.2 MAC的安全性
  - 2.3 CBC-MAC

提纲

- **1 散列函数**
  - **1.1 散列函数的定义**
  - 1.2 散列函数的通用结构
  - 1.3 MD5
- 2 消息认证码

散列函数的定义

- 散列函数 H 是一个**公开的函数**，它将任意长度的消息M变换为固定长度的散列码 h 。

- ***h =H (M)***
  - *M:* 变长消息，*H*(*M*): 定长的散列值

- 散列函数是一种算法，算法的输出内容称为**散列码**或者**消息摘要**。
- **消息摘要要唯一地对应原始消息**，如果原始消息改变并且再次通过散列函数，它将生成不同的消息摘要，因此散列函数能用来检测消息地完整性，保证消息从建立开始到收到为止没有被改变和破坏。
- 运行相同算法的接受者应该收到系统的消息摘要，否则保温是不可信的。

- 散列函数又称为：**哈希（ Hash ）函数**、数字指纹（ Digital fingerprint)、压缩（ Compression)函数、数据认证码（ Data Authentication Code ）等

散列函数的要求

- H 能用于任意大小的分组;
- H 能产生定长的输出;

- 对任何给定的x，$H(x)$要相对易于计算，使得硬件和软件实现成为实际可能;

- 对任何给定的码 h ，寻找 x 使得 $H(x) = h$ 在计算上是不可行的，即**单向性**（one-way）;

- 对任意给定的分组 x，寻找不等于x 的 y，使得 $H(x) = H(y)$ 在计算上是不可行的，即**弱抗冲突性**(Weak Collision-free);
- 寻找对任意的 (x, y) 对使得 $H(x) = H(y)$ 在计算上是不可行的，即**强抗冲突性**(Strong Collision-free);

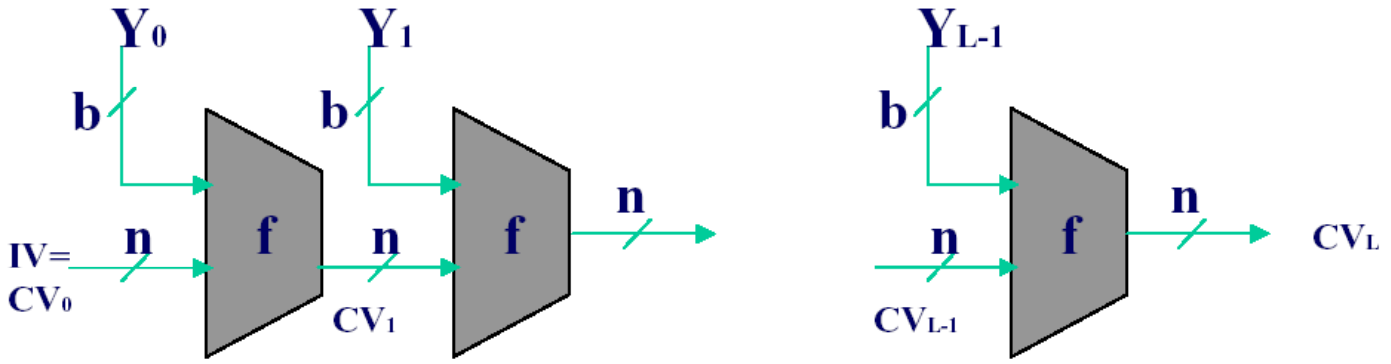
提纲

- **1 散列函数**
  - 1.1 散列函数的定义
  - **1.2 散列函数的通用结构**
  - 1.3 MD5
- 2 消息认证码

2.2 Hash函数的通用结构

- 由Ron Rivest于1990年提出MD4
- 几乎被所有Hash函数使用
- 具体做法:
  - 把原始消息M分成一些固定长度的块Y_i
  - 最后一块 padding 并使其包含消息M长度
  - 设定初始值CV_0
  - 压缩函数f, $CV_i = f(CV_{i\text{-}1}, Y_{i\text{-}1})$
  - 最后一个 CV_i 为Hash值

General Structure of Secure Hash Code



CV_0=IV= initial n-bit value
$CV_i = f(CV_{i-1}, Y_{i-1})$ $(1 \le i \le L)$
$H(M) = CV_L$

IV = initial value 初始值
CV = chaining value 链接值
Yi = ith input block (第i 个输入数据块)
f = compression algorithm (压缩算法）
n = length of hash code (散列码的长度)
b = length of input block(输入块的长度)

提纲

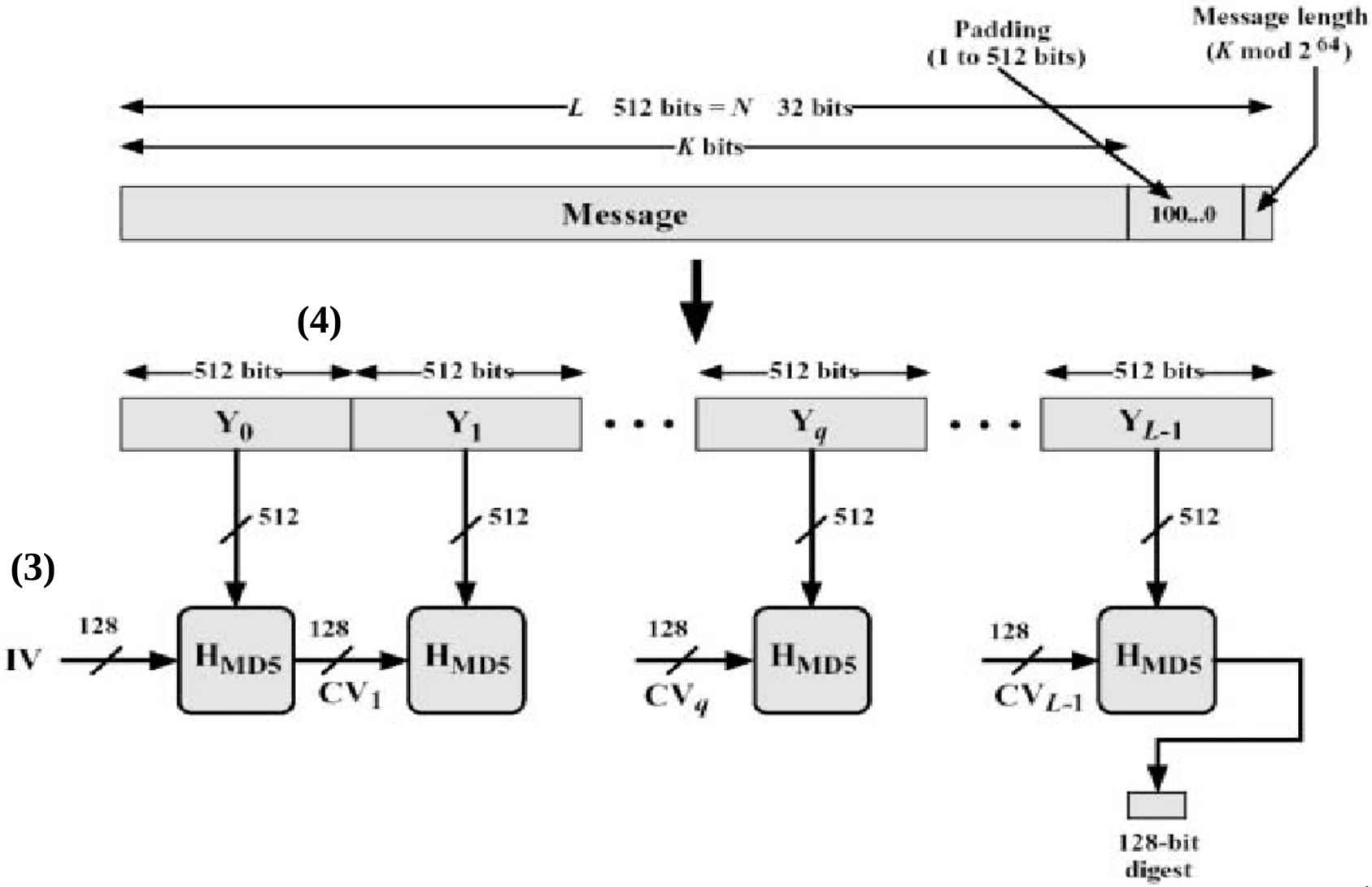
- **1 散列函数**
  - 1.1 散列函数的定义
  - 1.2 散列函数的通用结构
  - **1.3 MD5**
- 2 消息认证码

MD5 描述

- 1989年,Merkle提出Hash function模型
- 1990年,Ron Rivest提出MD4
- **1992年, Ron Rivest 完成MD5 (RFC 1321)**
  - **http://www.faqs.org/rfcs/rfc1321.html**
- 在最近数年之前，MD5是最主要的Hash算法
- 现行美国标准SHA-1以MD5的前身MD4为基础

- MD5（Message Digest）
  - 输入：任意长度的消息
  - 输入分组长度：**512 bit**
  - 输出：**128 bit 消息**

MD5: 示意图


Padding
(1 to 512 bits)
Message length
(K mod 2^{64})
L 512 bits = N 32 bits
K bits
Message
100...0
(4)
512 bits
512 bits
512 bits
512 bits
Y_0
Y_1
Y_q
Y_{L-1}
512
512
512
512
(3)
IV
128
128
128
128
H_{MD5}
H_{MD5}
H_{MD5}
H_{MD5}
CV_1
CV_q
CV_{L-1}
128-bit
digest

MD5 描述－（1）

- 消息填充
- 对消息进行填充，使其比特数与448模512同余，即填充长度为512的整数倍减去64,留出64比特在（2）中使用:

$$|M_1| \equiv 448 \bmod 512$$

$$if\ |M| = 448 \bmod 512, then\ |M_1| = |M| + 512$$

- 填充方法：填充比特串的最高位为1，其余各位均为0。最少要补1比特，最多补512比特（补一个组）。

$$M_1 = M \| Padding$$

MD5 描述－（2）

- 附加消息长度值

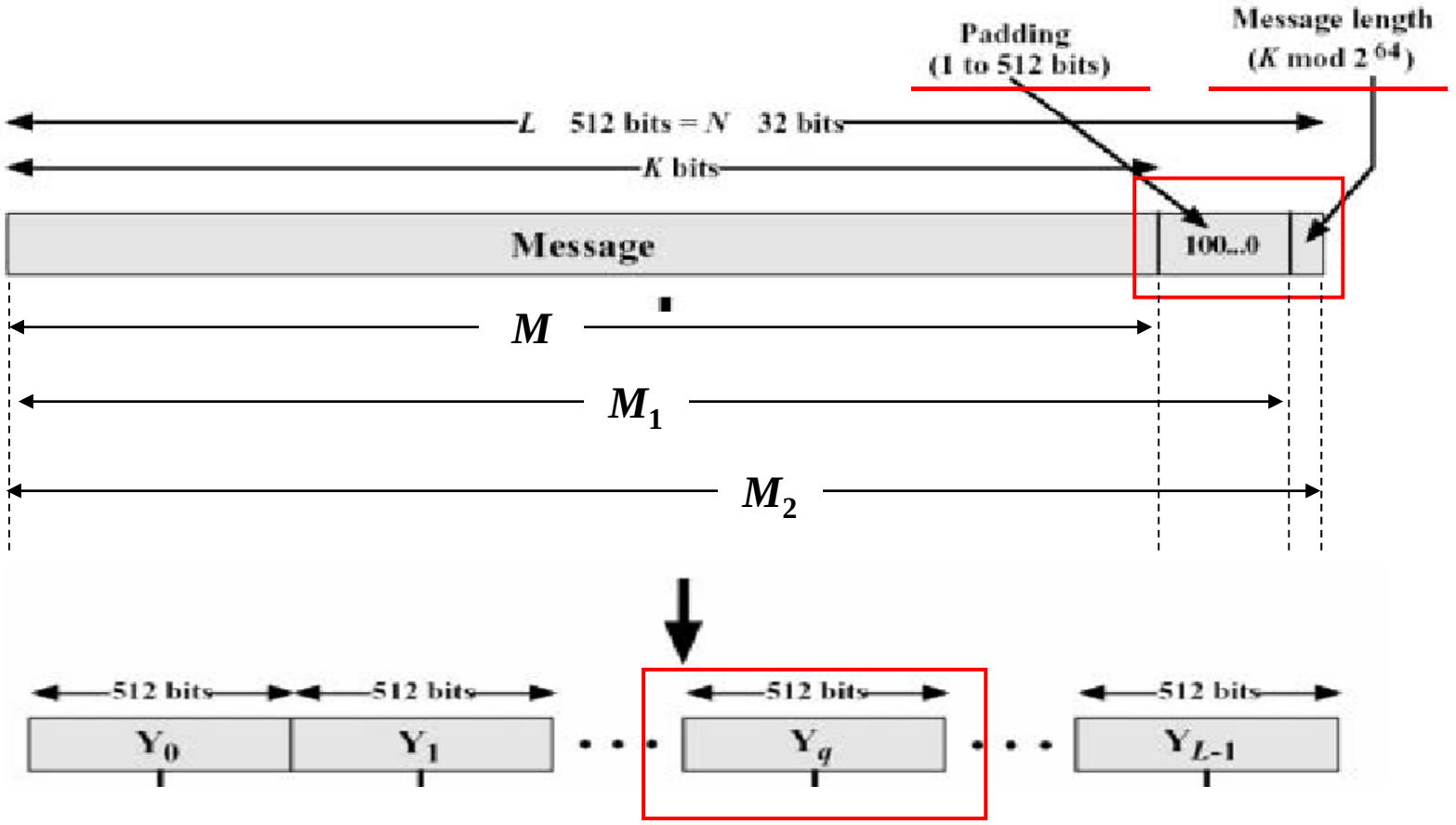
  $M_2 = M_1 \| LengthM = M \| Padding \| Length$

  $Length = | M | \bmod 2^{64}$, 低位字节优先, 表示为64bit长

  若初始长度大于2^{64}，仅使用该长度的低64bit

- （1）和（2）执行后，消息的长度为512的倍数，假设扩展后的长度为 $L \times 512$ 比特，将消息以512比特为单位进行分组，用$Y_0, Y_1, \ldots, Y_{L-1}$表示。
- 同样扩展后的消息以字（32比特）为单位可表示为 $M_2[0 \ldots N\text{-}1]$，$N = L \times 16$。

MD5: 示意图


Padding
(1 to 512 bits)
Message length
(K mod 2^{64})
L 512 bits = N 32 bits
K bits
Message
100...0
M
M_1
M_2
512 bits
512 bits
512 bits
512 bits
Y_0
Y_1
Y_q
Y_{L-1}

一个分组

MD5 描述－（3）

- **初始化MD缓存**
- MD为128bit，记为 CVq，用于存放散列函数的中间及最终结果， $CV_0 = \text{IV}$ 为初始化值。
- MD可表示为4个32bit的寄存器(A,B,C,D)，它们也称为链接变量，其初始化如下:

```
A = 01 23 45 67 (0x67452301)
B = 89 AB CD EF (0xEFCDAB89)
C = FE DC BA 98 (0x98BADCFE)
D = 76 54 32 10 (0x10325476)
```

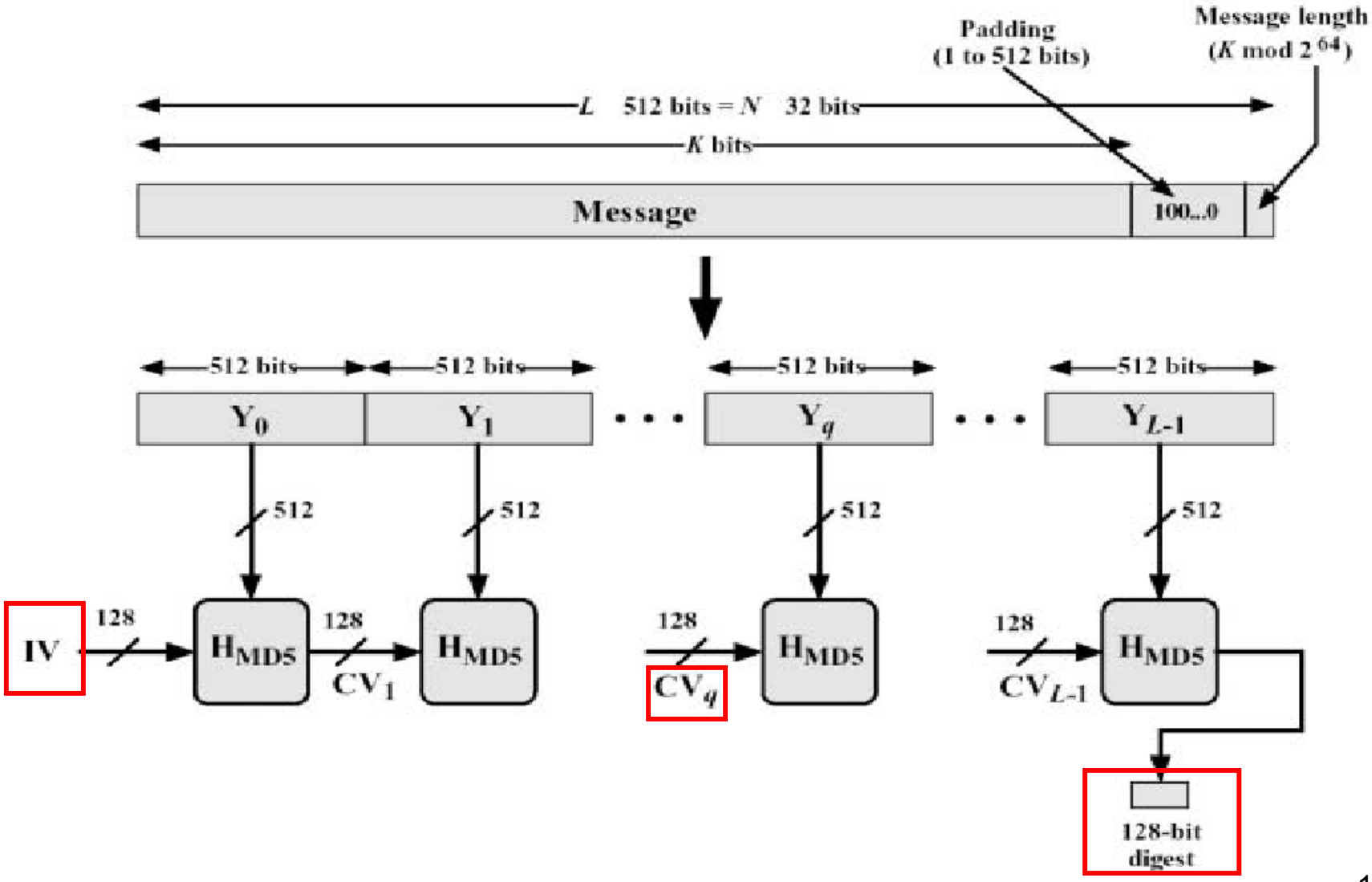
MD5 描述－（4）

- 处理消息分组
- 以消息分组（512bits）为单位，每一分组 Yq ($q = 0,1,\ldots,L\text{-}1$) 经过4个循环的压缩算法，表示为：

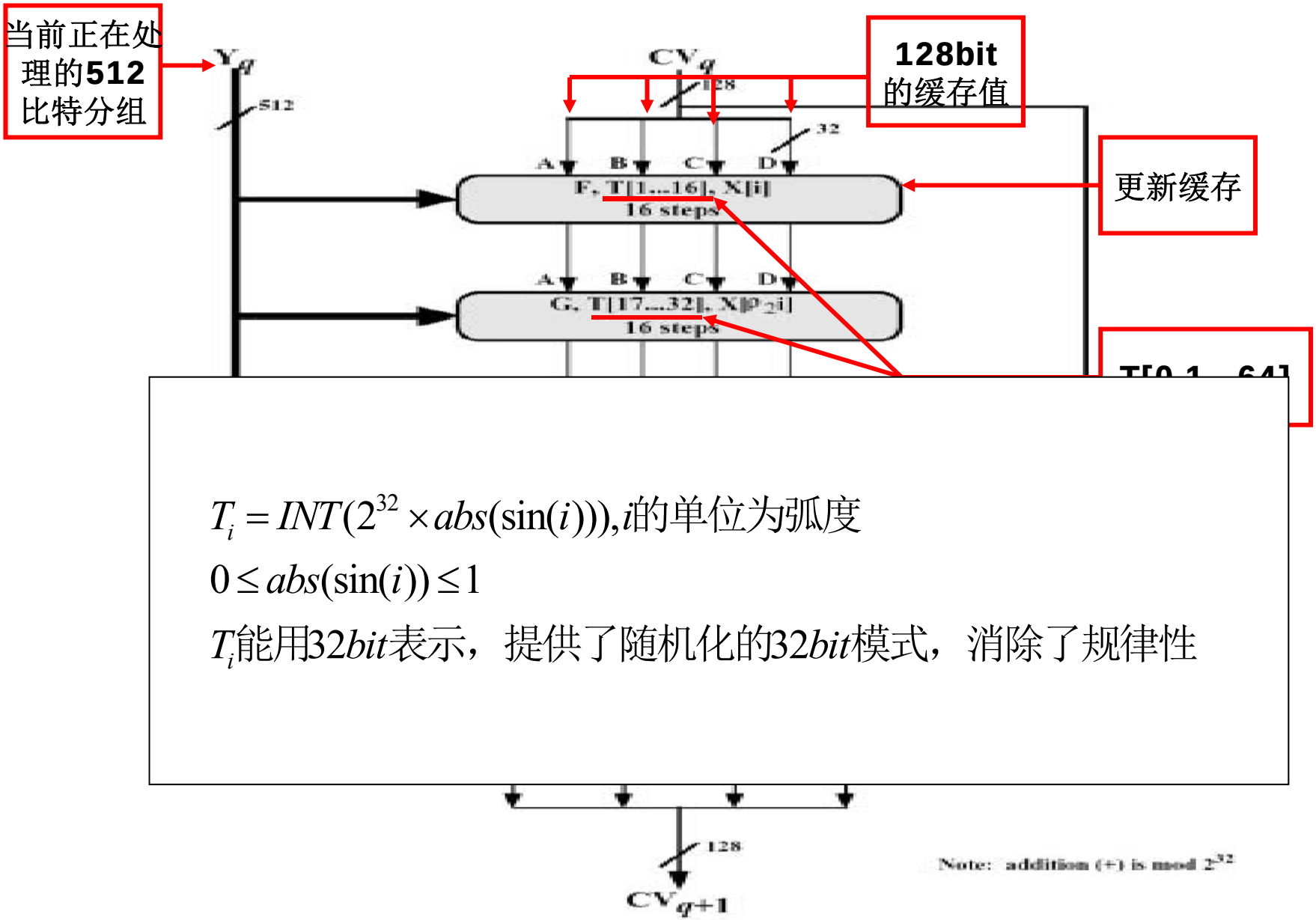
$$\begin{cases} CV_0 = IV \\ CV_i = H_{MD5}(CV_{i-1}, Y_i) \end{cases}$$

- 输出结果：$MD = CV_L$

MD5: 示意图


Padding
(1 to 512 bits)
Message length
(K mod 2^{64})
L 512 bits = N 32 bits
K bits
Message
100...0
512 bits
512 bits
512 bits
512 bits
Y_0
Y_1
Y_q
Y_{L-1}
512
512
512
512
IV
128
H_{MD5}
128
H_{MD5}
CV_1
128
CV_q
H_{MD5}
128
CV_{L-1}
H_{MD5}
128-bit
digest

H_{MD5}



$T_i = INT(2^{32} \times abs(\sin(i))), i$的单位为弧度

$0 \le abs(\sin(i)) \le 1$

T_i能用32*bit*表示，提供了随机化的32*bit*模式，消除了规律性

单个512bit分组的MD5处理过程 (MD5压缩函数H_{MD5})

每步操作形式

每一轮包含对缓冲区**ABCD**的**16**步操作所组成的一个序列。

a←b + ((a + g(b,c,d) + X[k] +T[i])<<<s)

其中，

a,b,c,d ＝缓冲区的四个字，以一个给定的次序排列；

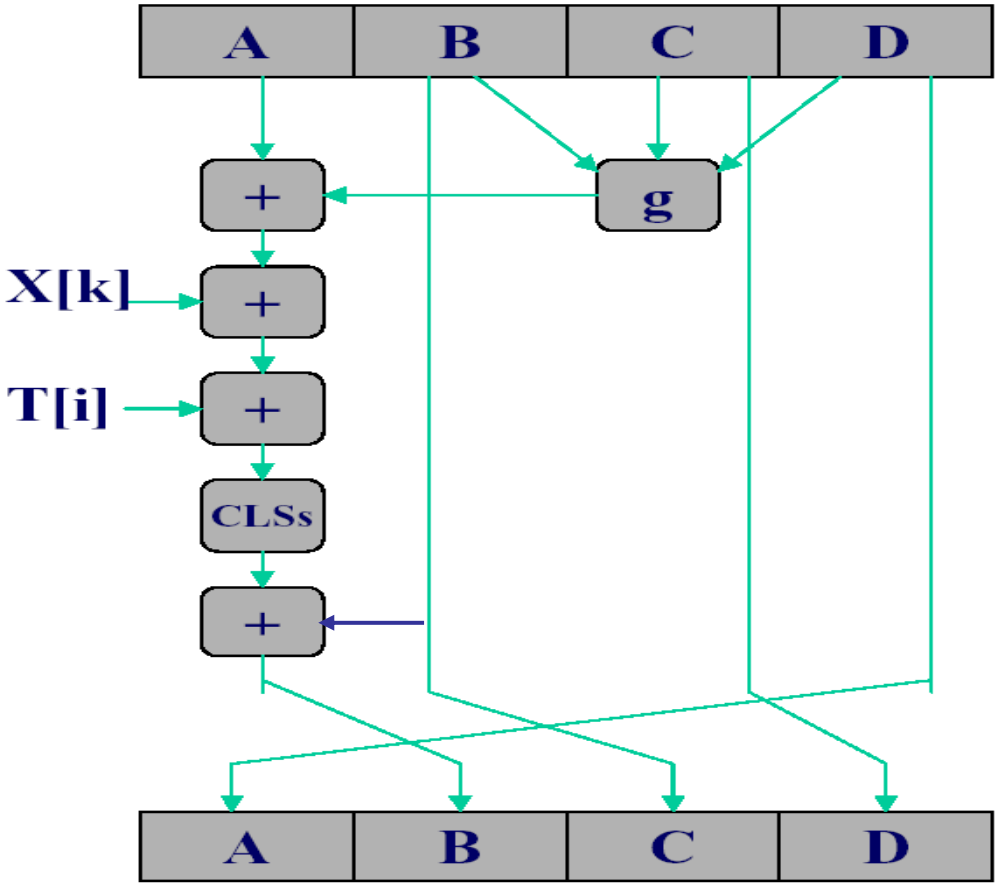
g ＝基本逻辑函数**F,G,H,I**之一；

<<<s ＝对**32**位字循环左移**s**位

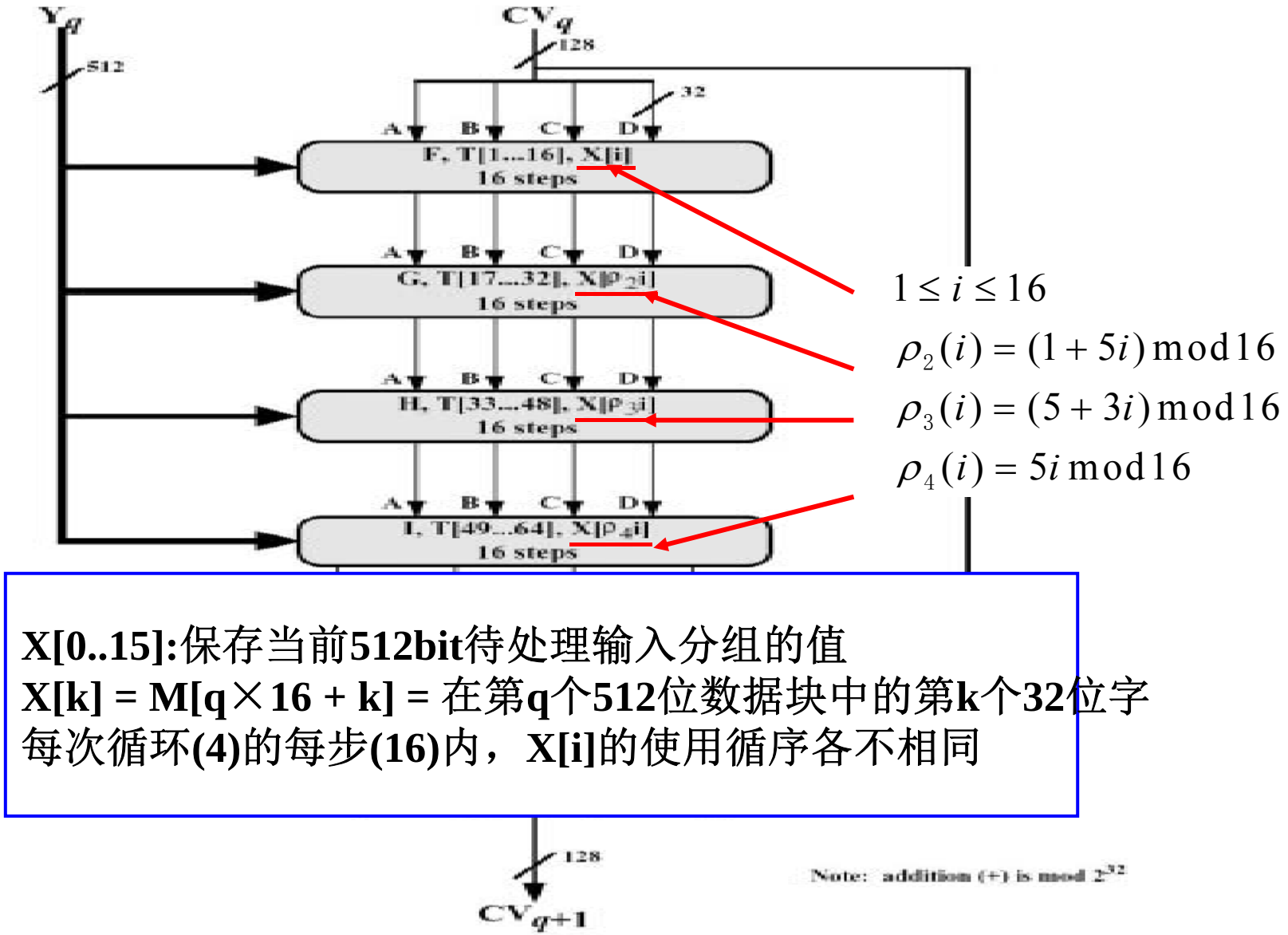
X[k] ＝**M[q×16 + k]** ＝在第**q**个**512**位数据块中的第**k**个**32**位字

T[i] ＝表**T**中的第**i**个**32**位字；

+ ＝模 2^{32}的加；



Function g	g(b,c,d)
1 F(b,c,d)	$(b \wedge c) \vee (b \wedge \bar{d})$
2 G(b,c,d)	$(b \wedge d) \vee (\bar{c} \wedge d)$
3 H(b,c,d)	$b \oplus c \oplus d$
4 I(b,c,d)	$c \oplus (b \vee \bar{d})$



X[0..15]:保存当前512bit待处理输入分组的值
X[k] = M[q×16 + k] = 在第q个512位数据块中的第k个32位字
每次循环(4)的每步(16)内，X[i]的使用循序各不相同

单个512bit分组的MD5处理过程 (MD5压缩函数H_{MD5}）

MD5的安全性

- Berson表明，对单循环MD5，使用不用的密码分析可能在合理的时间内找出能够产生相同摘要的两个消息，这个结果被证明对四个循环中的任意一个循环也成立，但作者没有能够提出如何攻击包含全部4个循环MD5的攻击。
- Boer和Bosselaers显示了即使缓存ABCD不同，MD5对单个512bit分组的执行将得到相同的输出(伪冲突）。
- Dobbertin的攻击技术：使MD5的压缩函数产生冲突，即寻找:

$$\exists x、y, x \neq y,\ H(x) = H(y)$$

- MD5被认为是易受攻击的，逐渐被SHA-1和RIPEMD-160替代。

其他散列函数

	MD5	SHA-1	RIPEMD-160
摘要长度	128位	160位	160位
基本处理单位	512位	512位	512位
步数	64(4 of 16)	80(4 of 20)	160(5 paired of 16)
最大消息长度	无限	2^{64}-1位	2^{64}-1位
基本逻辑函数	4	4	5
加法常数	64	4	9
Endianness	Little-endian	Big-endian	Little-endian
性能	32.4 Mbps	14.4Mbps	13.6Mbps

Http://www.eskimo.com/~weidai/benchmarks.txt, C++ on Pentium 266Mhz

Hash小结

- Hash函数把变长信息映射到定长信息
- Hash函数不具备可逆性
- Hash函数速度较快
- Hash函数与对称密钥加密算法有某种相似性
- 对Hash函数的密码分析比对称密钥密码更困难
- Hash函数可用于消息摘要
- Hash函数可用于数字签名

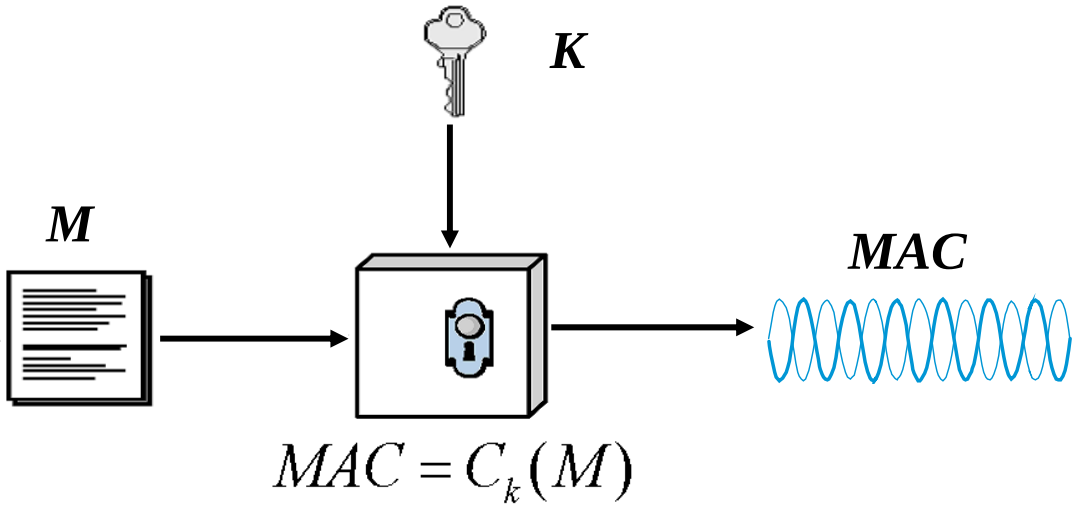
提纲

- 1 散列函数
- **2 消息认证码**
  - **2.1 MAC函数**
  - 2.2 MAC的安全性
  - 2.3 CBC-MAC
  - 2.4 HMAC

2.1 MAC函数

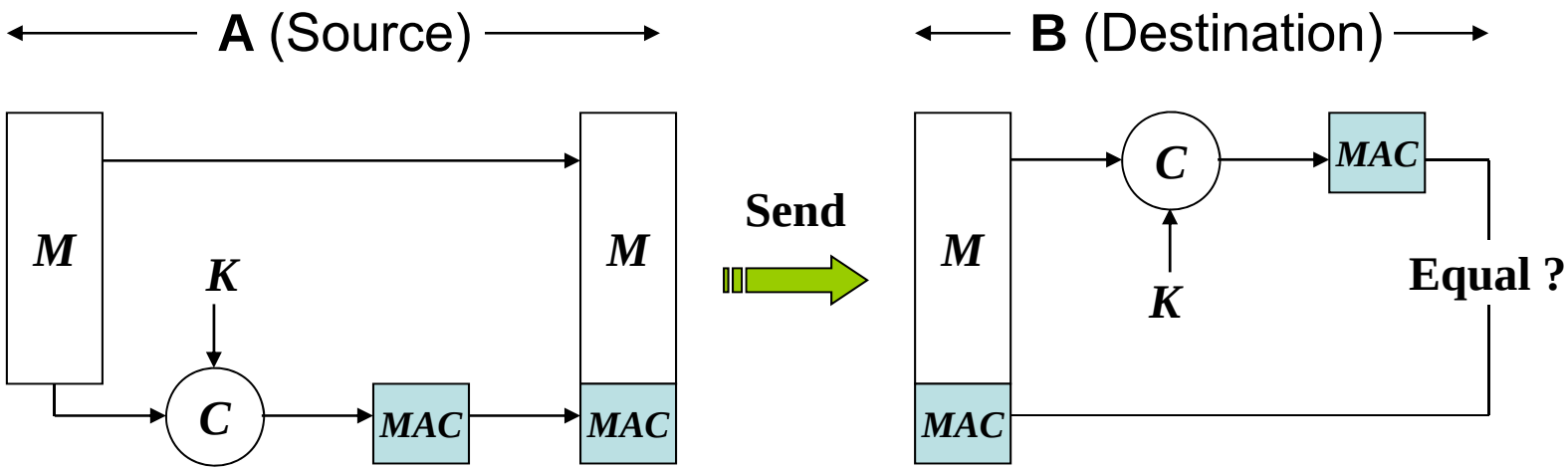
- Message Authentication Code（MAC）
  - 使用一个密钥生成一个固定大小的小数据块，并加入到消息中，称 MAC，或密码校验和（cryptographic checksum）。
- MAC的作用:
  - 接收者可以确信消息M未被改变;
  - 接收者可以确信消息来自所声称的发送者;
  - 如果消息中包含顺序码（如HDLC,X.25,TCP），则接收者可以保证消息的正常顺序;
- MAC函数类似于加密函数，但不需要可逆性。因此在数学上比加密算法被攻击的弱点要少。

MAC函数



- K：双方共享的一个密钥;
- M：变长的消息;
- $MAC = C_K(M)$：定长的认证码;
- MAC 一般为多对一函数;

MAC函数用于消息认证



- MAC函数：$MAC = C_K(M)$
  - K 为通信双方 A 和 B 共享的一个密钥;
  - M 为原始消息，MAC 为消息认证码;

提纲

- 1 散列函数
- **2 消息认证码**
  - 2.1 MAC函数
  - **2.2 MAC的安全性**
  - 2.3 CBC-MAC
  - 2.4 HMAC

对MAC的强行攻击（1）

- 函数域：任意长度的消息。
- 值域：所有可能的MAC和所有可能的密钥。

- 假设：
  - MAC 的长度为 n 比特 $\rightarrow 2^n$ 个可能的MAC
  - N 个消息，$N \geq 2^n$
  - 密钥的长度为$k \rightarrow 2^k$ 个可能的密钥

- 假设攻击者使用强行攻击，且已经获得消息的明文和相应的MAC，即 (M_i, MAC_i) 。

对MAC的强行攻击（2）

- 第1 轮:
  - 给定 M_1 和 $MAC_1 = C_K(M_1)$，对所有 2^k 个可能的密钥计算:
    $$MAC_1 = C_{Ki}(M_1) \qquad i = 1,2,\ldots,2^k$$
  - 匹配数≈ 2^{k-n}，得到 2^{k-n} 个可能的密钥;
- 第2 轮:
  - 给定 M_2 和 $MAC_2 = C_K(M_2)$，对所有 2^{k-n} 个可能的密钥计算:
    $$MAC_2 = C_{Ki}(M_2) \qquad i = 1,2,\ldots,2^{k-n}$$
  - 匹配数≈ 2^{k-2n}，得到 2^{k-2n} 个可能的密钥;
- ……

- 如果$k > n$（设 $k = a \times n$），则这种攻击需要进行a 轮。
  - 共需要 2^k 次以上的 *MAC* 运算。
- 如果 $k \leq n$，则第1轮就可以产生一个唯一对应的密钥，但仍然可能有多于一个密钥产生这一配对（当$k < n$ 时），这时攻击者需对一个新的 (M, MAC)进行相同的测试。
  - 共需要 ≥ 2^k 次的 *MAC* 运算。
- 由此可见，攻击者企图发现 *MAC* 的密钥不小于甚至大于对同样长度的解密密钥的攻击。

对MAC的其它攻击

- 设$M=(X_1\|X_2\|\ldots\|X_m)$是一个由64比特$X_i$数据块连接而成，
  - 定义 $\triangle(M)=X_1\oplus X_2\oplus\ldots\oplus X_m$
    $C_K(M)=E_K[\triangle(M)]$
  - 其中$\oplus$为异或操作，***E* 为ECB工作模式的DES算法**
- 则：
  - 密钥长度为56比特，MAC长度为64比特，
  - 攻击者至少需要2^{56}次加密来决定密钥K。

- 设$M'=(Y_1\|Y_2\|\ldots\|Y_m)$，
  - 其中$Y_1, Y_2,\ldots,Y_{m-1}$是替换$X_1, X_2,\ldots, X_{m-1}$的任意值，而
    $Y_m=Y_1\oplus Y_2\oplus\ldots\oplus Y_{m-1}\oplus\triangle(M)$
  - 则 $\boldsymbol{C_K(M')}=E_K[\triangle(M')]$
    $=E_K[Y_1\oplus Y_2\oplus\ldots\oplus Y_{m-1}\oplus Y_m]$
    $=E_K[Y_1\oplus Y_2\oplus\ldots\oplus Y_{m-1}\oplus Y_1\oplus Y_2\oplus\ldots\oplus Y_{m-1}\oplus\triangle(M)]$
    $=\boldsymbol{E_K[\triangle(M)]}$
- 这时消息M'和$MAC=C_K(M)=E_K[\triangle(M)]$是一对可被接收者认证的消息。
- 用此方法，任何长度为$64\times(m-1)$比特的消息可以被插入任意的欺骗性信息。

MAC应具备的性质

- 如果一个攻击者得到 M 和 $C_K(M)$，则攻击者构造一个消息 M' 使得 $C_K(M') = C_K(M)$ 应具有计算复杂性意义下的不可行性。
- $C_K(M)$应均匀分布，即：随机选择消息 M 和M'，$C_K(M) = C_K(M')$的概率是2^{-n}，其中 n 是 MAC 的位数。
- 令 M' 为M 的某些变换，即：$M' = f(M)$，（例如：f 可以涉及 M 中一个或多个给定位的反转），在这种情况下，$C_K(M) = C_K(M')$的概率是2^{-n}，其中 n 是 MAC 的位数。

提纲

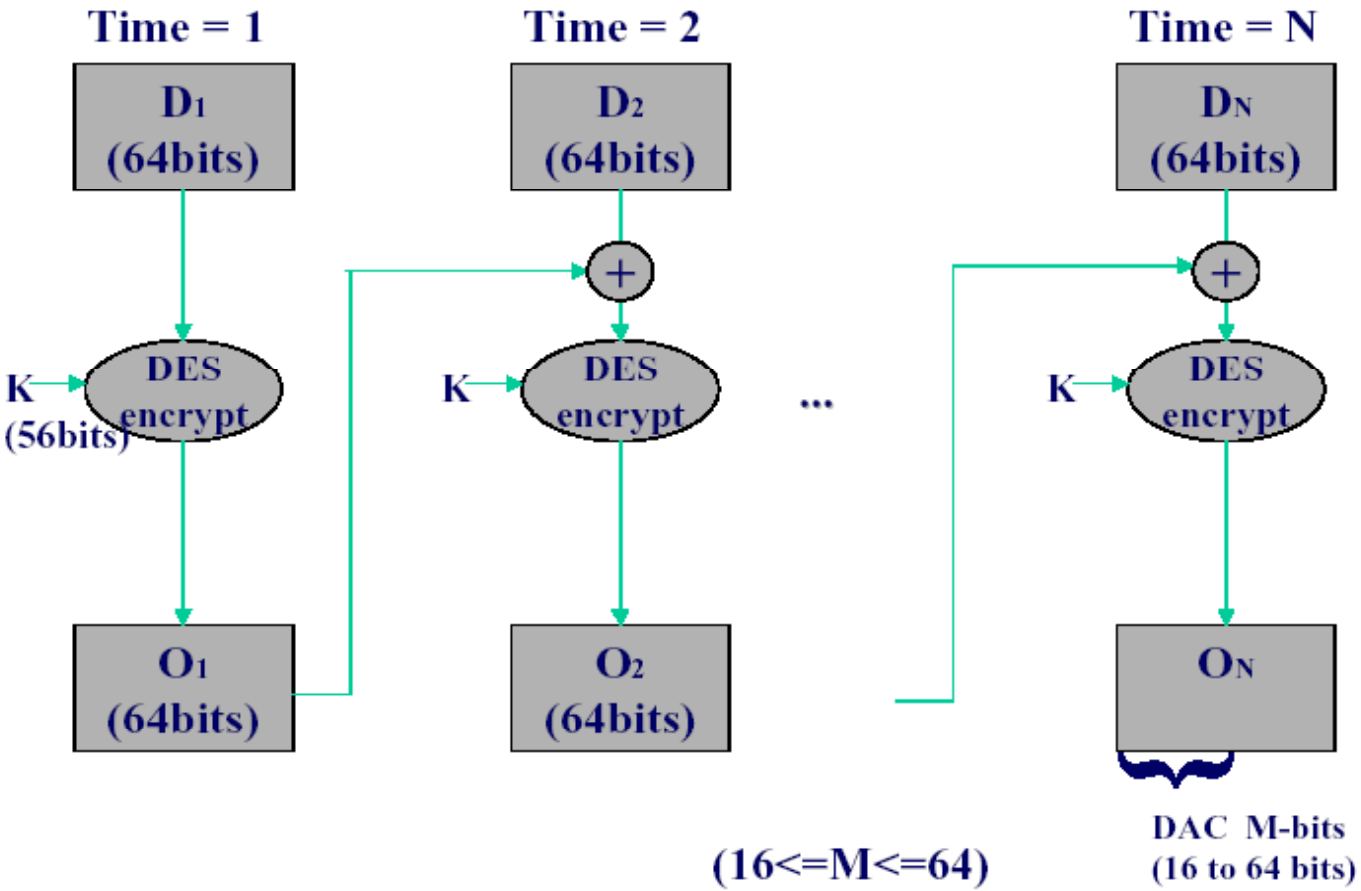
- 1 散列函数
- **2 消息认证码**
  - 2.1 MAC函数
  - 2.2 MAC的安全性
  - **2.3 CBC-MAC**
  - 2.4 HMAC

基于DES的CBC-MAC

- 算法来源
  - FIPS publication (FIPS PUB 113) http://www.itl.nist.gov/fipspubs/fip113.htm
  - ANSI standard (X9.17)
- 使用CBC(Cipher Block Chaining)方式，初始向量为IV=0

- 将数据按64位分组，$D_1, D_2, \ldots, D_N$，必要时最后一个数据块用 0 向右填充。
- 运用DES算法E，密钥为K。
- 数据认证码(DAC)的计算如下:

$$O_1 = E_K(D_1)$$
$$O_2 = E_K(D_2 \oplus O_1)$$
$$O_3 = E_K(D_3 \oplus O_2)$$
$$\ldots$$
$$O_N = E_K(D_N \oplus O_{N-1})$$



$$\begin{cases} O_i = E_K(D_i \oplus O_{i-1}) \\ O_0 = \{0\}^{64} \end{cases} \qquad i = 1,2,\ldots,N$$

The End

machunguang@hrbeu.edu.cn