第六章

散列函数与消息认证码

- 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;

Message Authentication Code（MAC）又称为:
Hash value
Message digest
Digital fingerprint

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

- 散列函数散列函数 *HH* 是一个是一个**公开的函数公开的函数**，，它将任意长度的消息它将任意长度的消息MM变换为固定长变换为固定长度的散列码 *h* 。 *如MD5输出128bits;SHA-1输出为160bits*
- ***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。
-

- 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);
- 找到任何满足 $H(x) = H(y)$的一对数 (x, 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

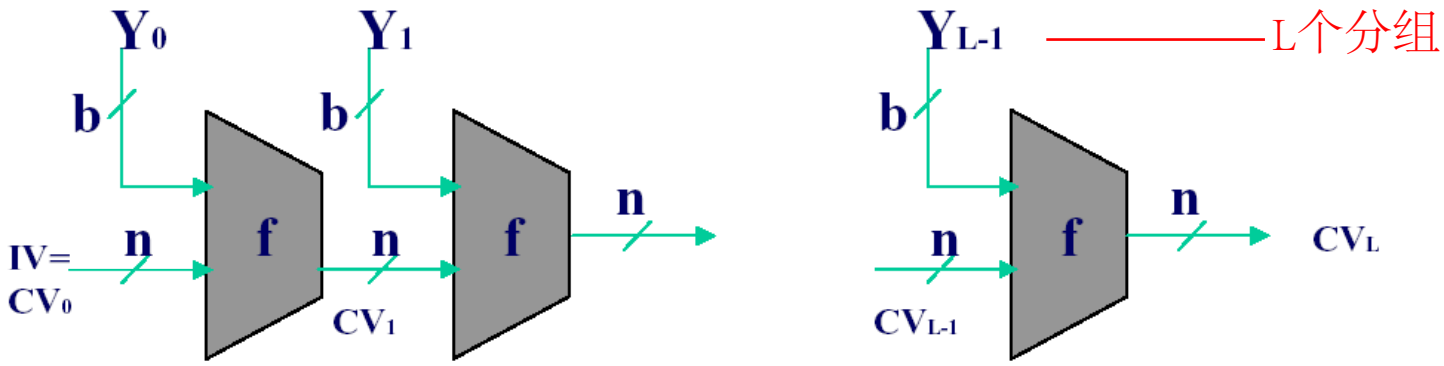
2.2 Hash函数的通用结构

- 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, $CV_i = f(CV_{i-1}, Y_{i-1})$
  - 最后一个 CV_i 为Hash值

CBC模式(Cipher Block Chaining)

填充100...0，使其<长度> ≡448 mod 512,然后再加上以64bits表示的Message长度。如例 message len=13096bits, 则填充串“100...0”，长度为448-mod(13096,512)=152，即1后面跟151个0，然后再填充13096的64bits表示(长度): 0...0 11001100101000

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)

n 可为128/160/256/512

b = length of input block(输入块的长度)

b分组长度一般为512，SHA512为1024

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

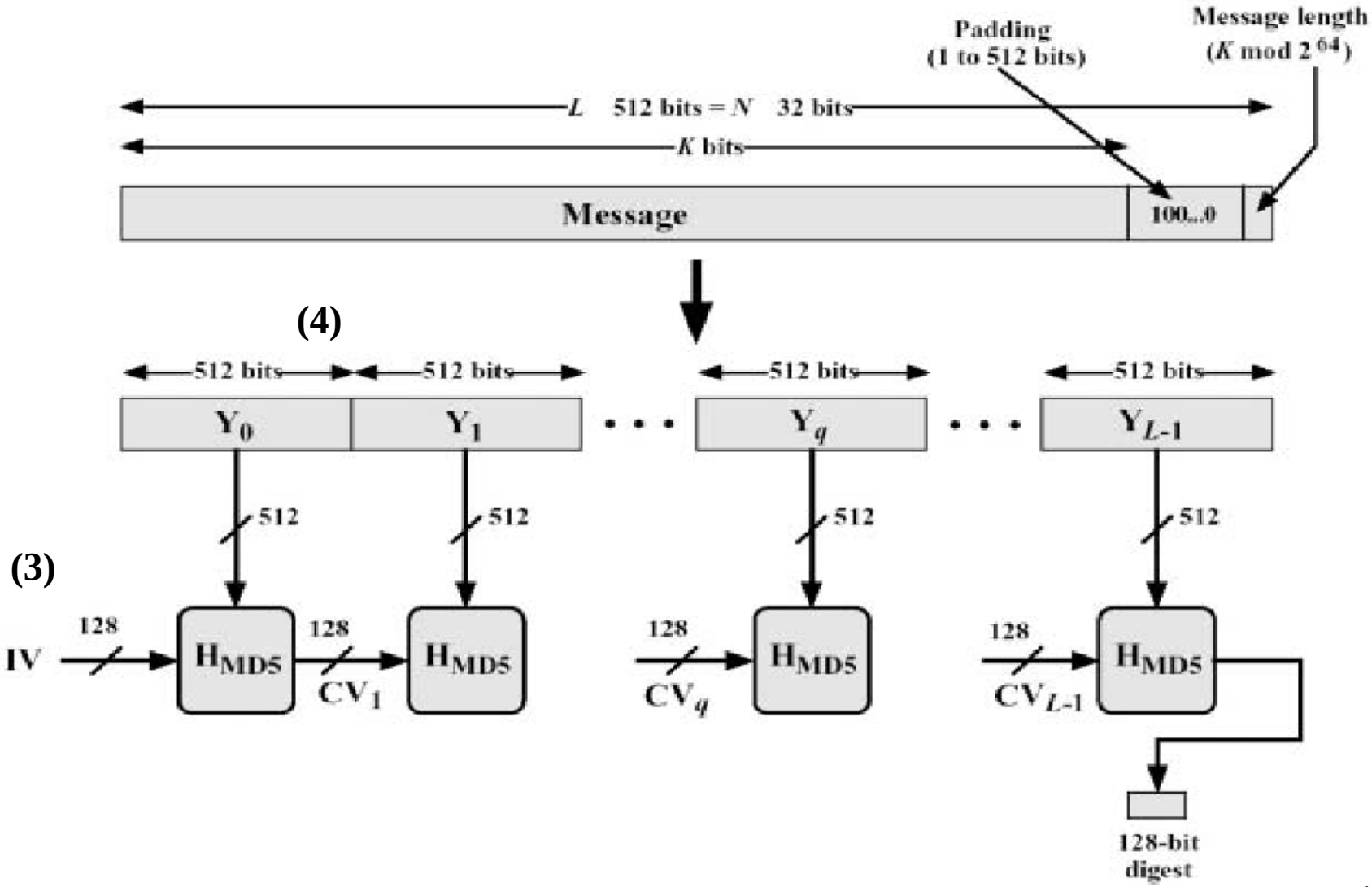
- 1989年,Merkle提出Hash function模型
- 1990年,Ron Rivest提出MD4
- **1992年, Ron Rivest 完成MD5 (RFC 1321)**
  - **http://www.faqs.org/rfcs/rfc1321.html**
- 在最近数年之前，MD5是最主要的Hash算法
- 现行美国标准SHA-1以MD5的前身MD4为基础

- 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**

长度不超过2^64

MD5: 示意图

若K的长度>2^64，则只取K mod 2^64作为长度填充在末尾



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）。

$$M_1 = M \| Padding$$

消息长度正好是512的整数倍

MD5 描述－（2）

- **附加消息长度值**

消息长度大约为2.305E+18 个字节

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

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

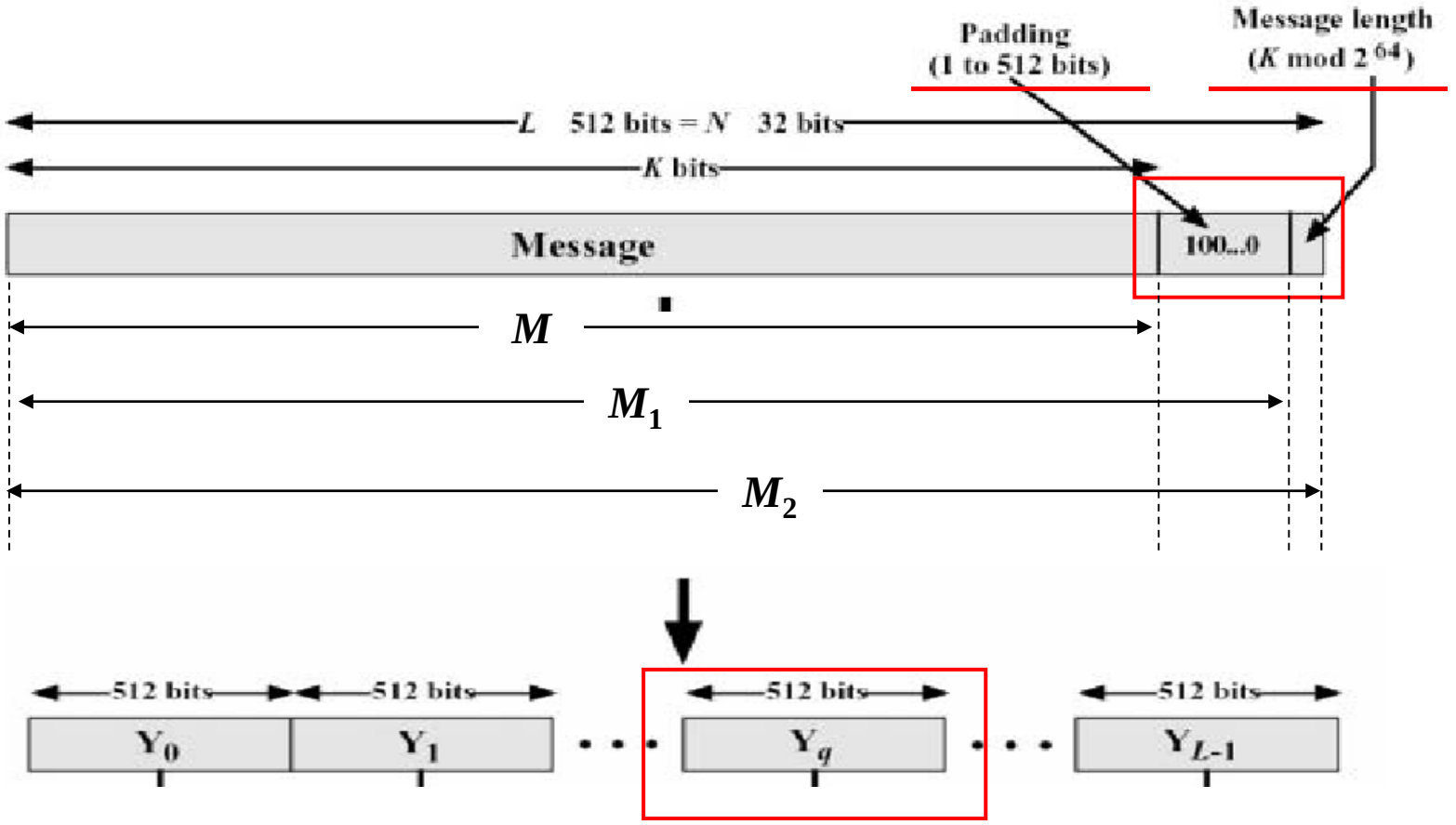
若初始长度大于2^{64}，仅使用该长度的低64bit

- （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 $M_2[0 \ldots N\text{-}1]$，$N = L \times 16$。

每个分组用16个字表示

M2=M1|Length of M=M|Padding|Length of M, 则M2 mod 512=0

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

RISC
CISC

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：

存储方式　　实际值

内存存储单元地址 →

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

每个寄存器以little endian方式存储，即低字节存于低地址。相反的存储方式称为big endian方式

Little-Endian就是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。
Big-Endian就是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。
到底是哪种存储与CPU体系结构有关(PowerPC/ARM/MIPS系列CPU和Intel的x86系列CPU)。PowerPC系列采用big endian方式存储数据，而x86系列则采用little endian方式存储数据

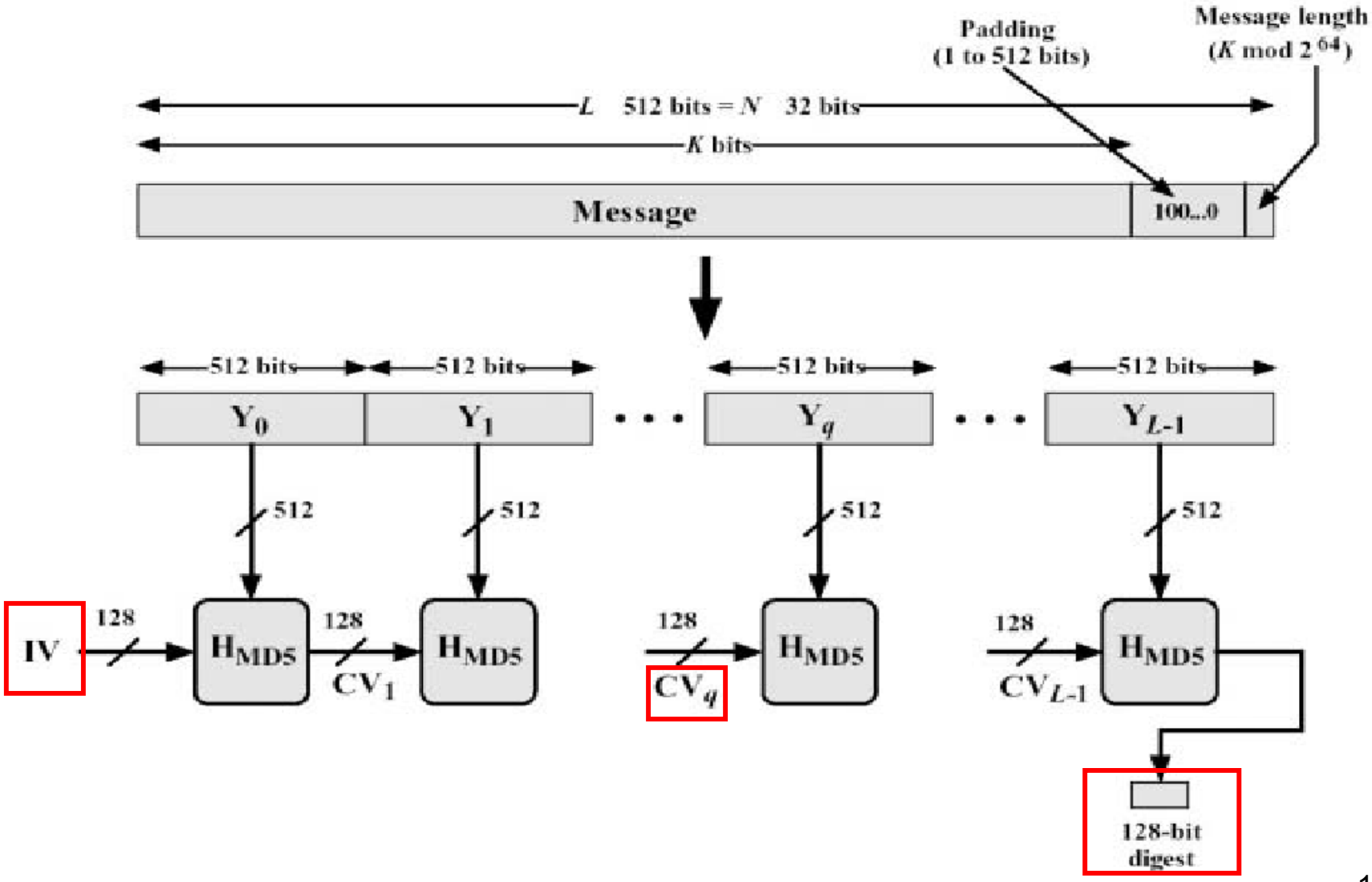
MD5 描述－（4）

- 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：

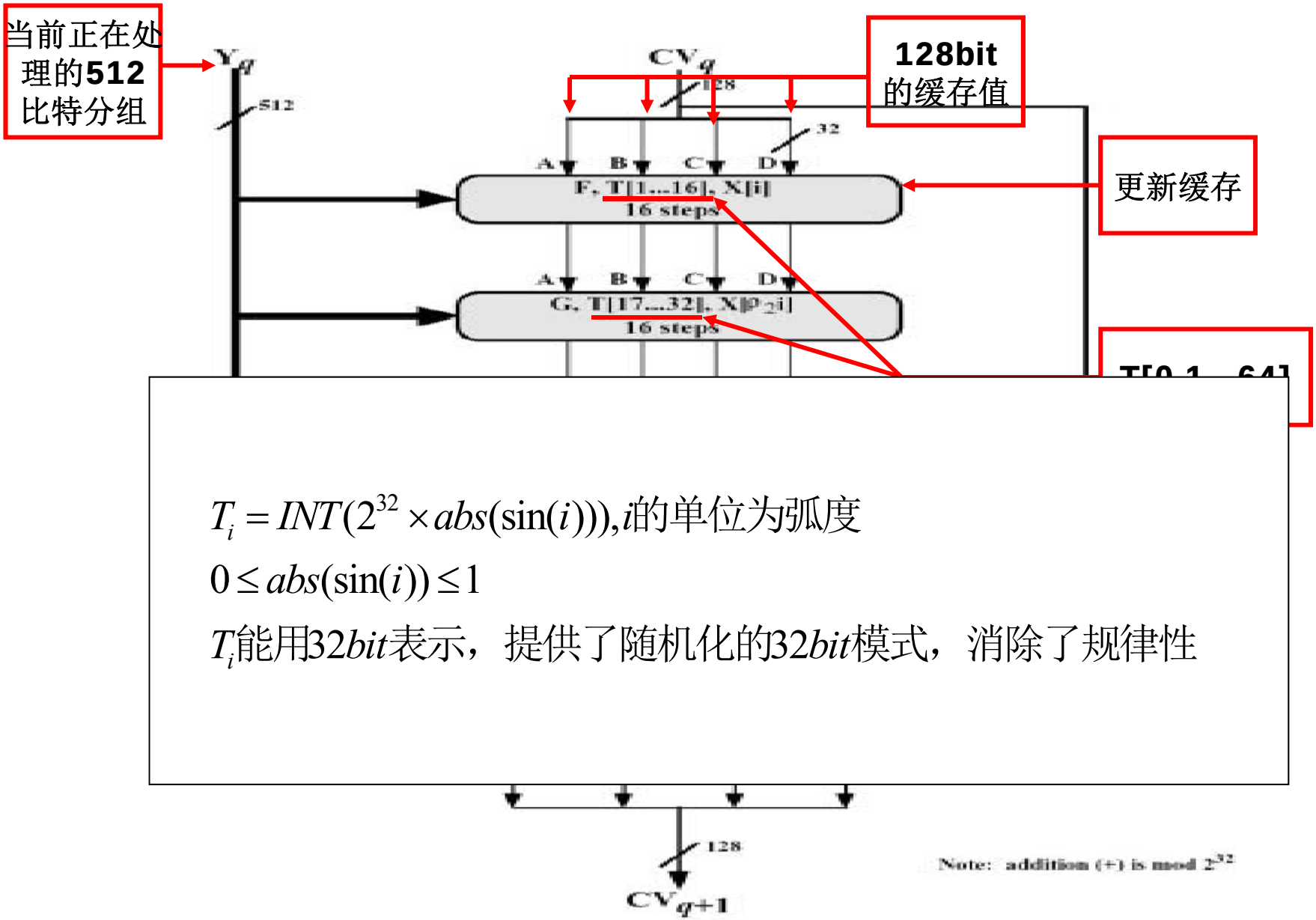
$$\begin{cases} CV_0 = IV \\ CV_i = H_{MD5}(CV_{i-1}, Y_i) \end{cases}$$

- 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；　四个轮函数

<<<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；

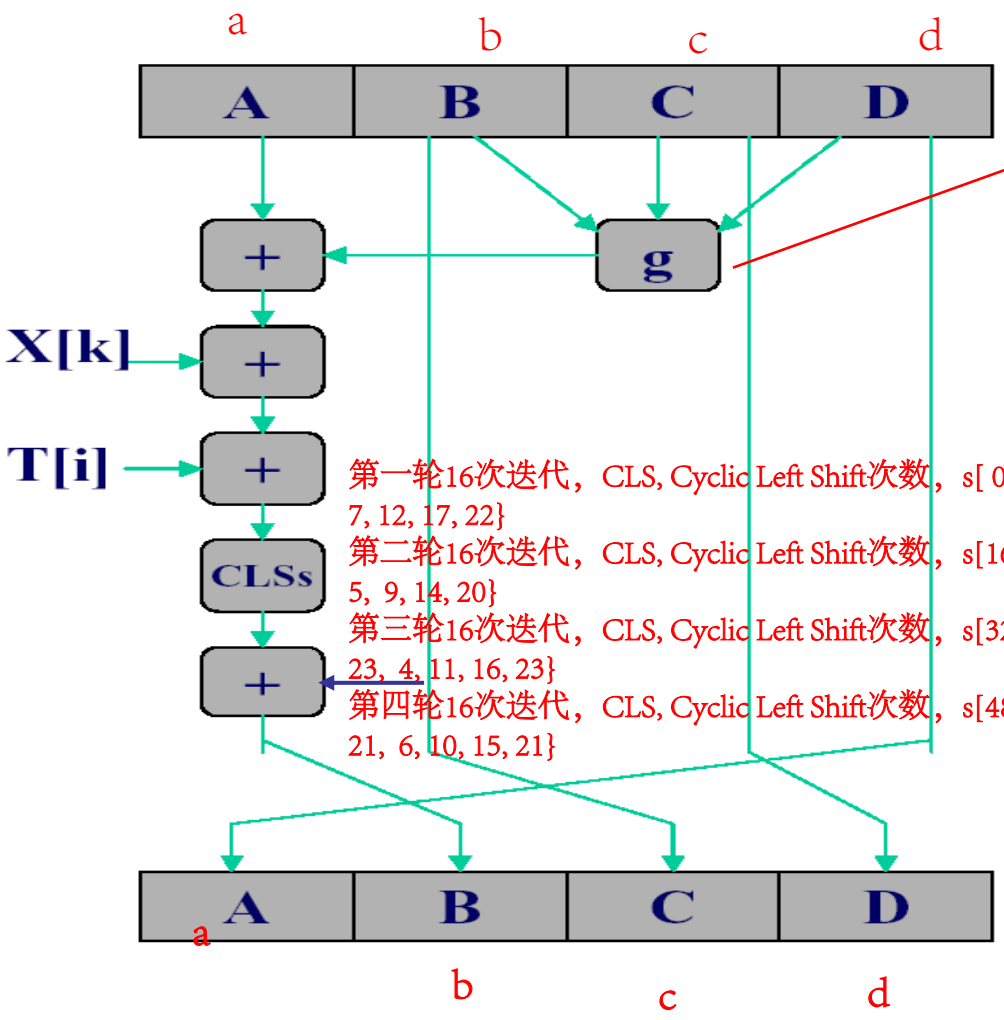
+ ＝模 2^{32}的加；

常数T[i]是4294967296*abs(sin(i)) 的整数部分，i取值从1到64，单位是弧度,4294967296是2的32次方

X[k]表示将当前分组512bits分成16个子分组（32bits)，每轮(共4轮)的16个子分组的顺序不一样。

+表示mod 2^32加法运算

g( )称为轮函数，在4轮中分别表示F,G,H,I函数



Function g g(b,c,d)

1 F(b,c,d) = (b∧c)∨(b∧d)

2 G(b,c,d) = (b∧d)∨(c∧d)

3 H(b,c,d) = b⊕c⊕d

4 I(b,c,d) = c⊕(b∨d)

第一轮16次迭代，CLS, Cyclic Left Shift次数，s[0..15]：= {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22}

第二轮16次迭代，CLS, Cyclic Left Shift次数，s[16..31]：= {5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20}

第三轮16次迭代，CLS, Cyclic Left Shift次数，s[32..47]：= {4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23}

第四轮16次迭代，CLS, Cyclic Left Shift次数，s[48..63]：= {6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21}

a='A'
b='B'
c='C'
d='D'
for i=1 to 16
temp=d
d=c
c=b
b=b + ((a + g(b,c,d)+X[k]+T[i]) << s)
a=temp
end for

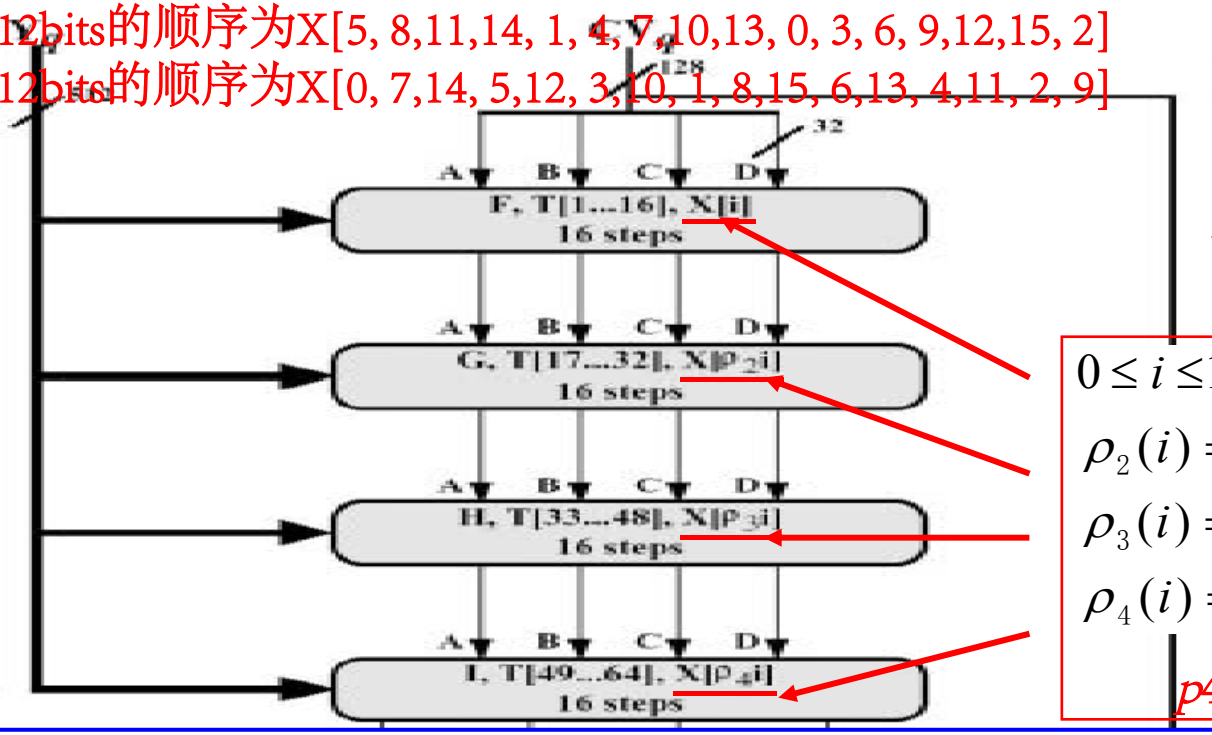
在某1轮16次迭代中参加运算的寄存器的顺序为

移位s不定数

1，ABCD
2，DABC
3，CDAB
4，BCDA
5，ABCD
6，DABC
7，CDAB
8，BCDA
9，ABCD
10,DABC
11,CDAB
12,BCDA
13,ABCD
14,DABC
15,CDAB
16,BCDA

第1轮使用当前分组512bits的顺序为X[0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15]
第2轮使用当前分组512bits的顺序为X[1, 6,11, 0, 5,10,15, 4, 9,14, 3, 8,13, 2, 7,12,]
第3轮使用当前分组512bits的顺序为X[5, 8,11,14, 1, 4, 7,10,13, 0, 3, 6, 9,12,15, 2]
第3轮使用当前分组512bits的顺序为X[0, 7,14, 5,12, 3,10, 1, 8,15, 6,13, 4,11, 2, 9]

四个轮函数F,G,H,I
T[1]到T[64]常量，
每轮16步，每步1个32字节的分组参加运算
（16*32=512bits）
4轮处理中，每轮以不同的次序使用16个字(分组)



$0 \le i \le 15$ 原始的明文分组对应的16个字

$\rho_2(i) = (1 + 5i) \bmod 16$

$\rho_3(i) = (5 + 3i) \bmod 16$

$\rho_4(i) = 5i \bmod 16$

$\rho 4(i)=7*i \bmod 16$

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

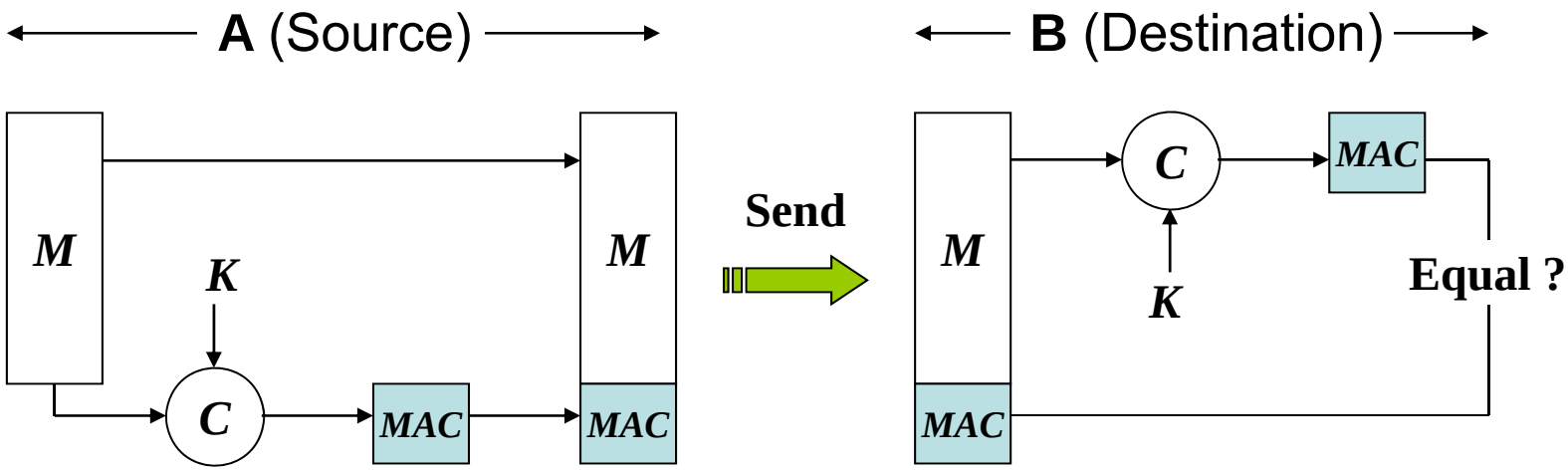
Hash小结

- 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

6.4 消息认证

- 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。

（一）MAC函数用于消息认证

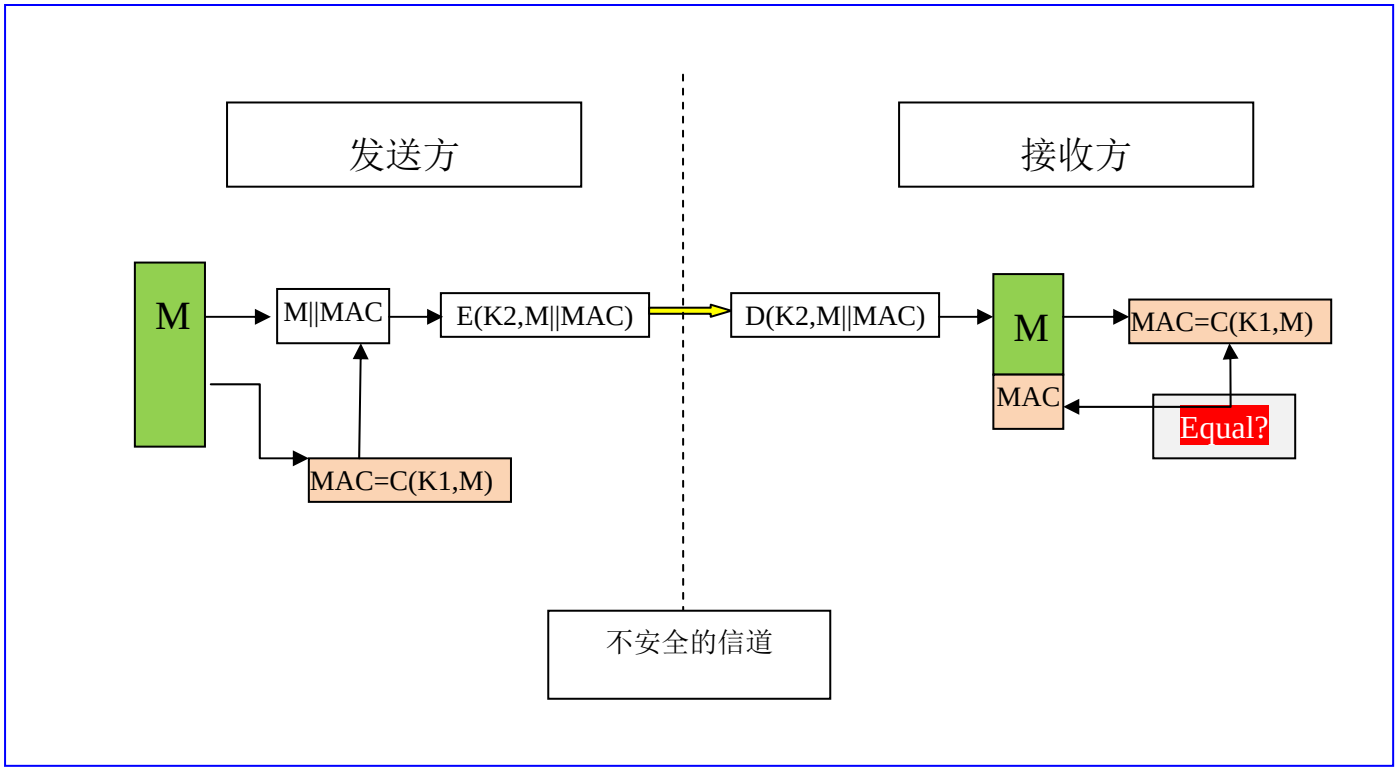


C为加密函数，K为通信双方共享密钥，即用K对hash值进行加密后，附加在消息M之后进行传送。

- 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；

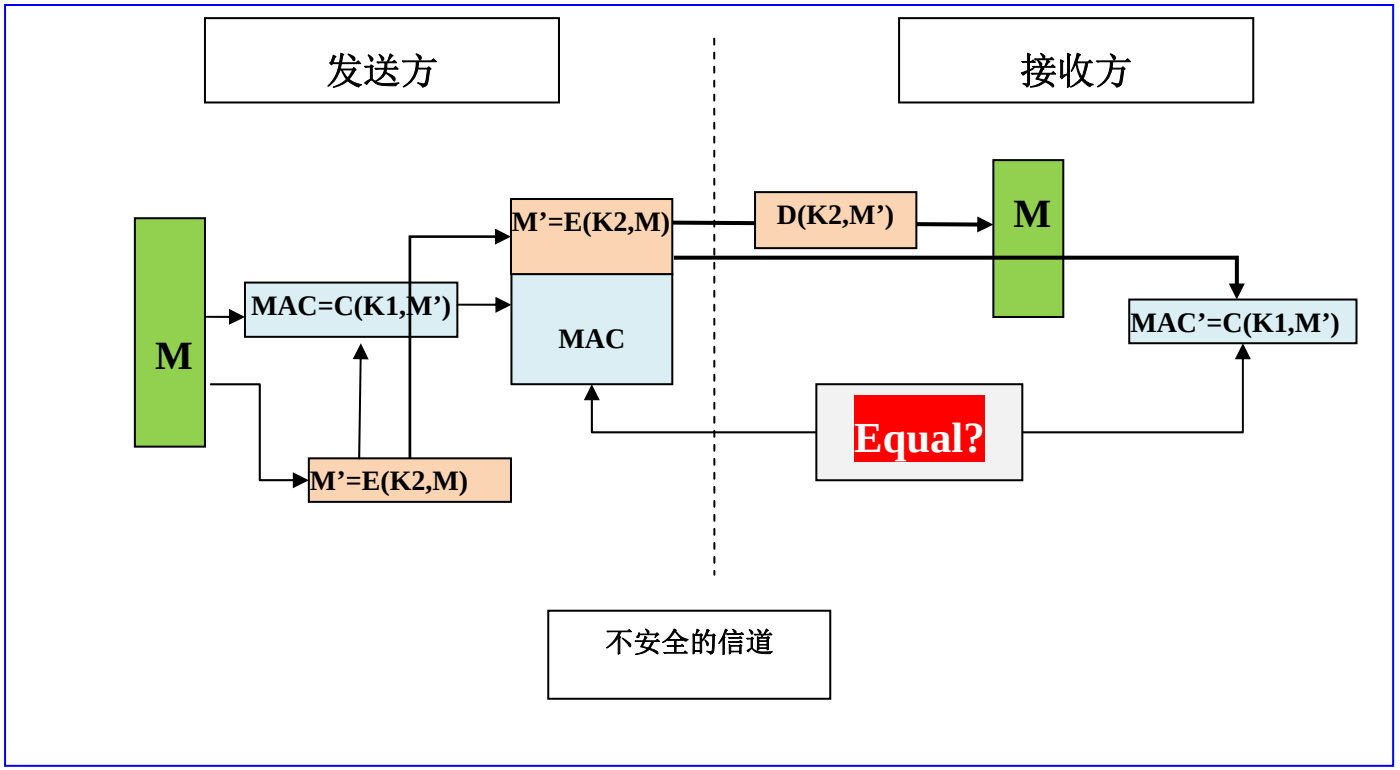
特点：该模型只提供消息认证，没有提供机密性

（二）消息认证和保密性：与明文有关的认证



消息认证与明文有关

（三）消息认证和保密性：与密文有关的认证



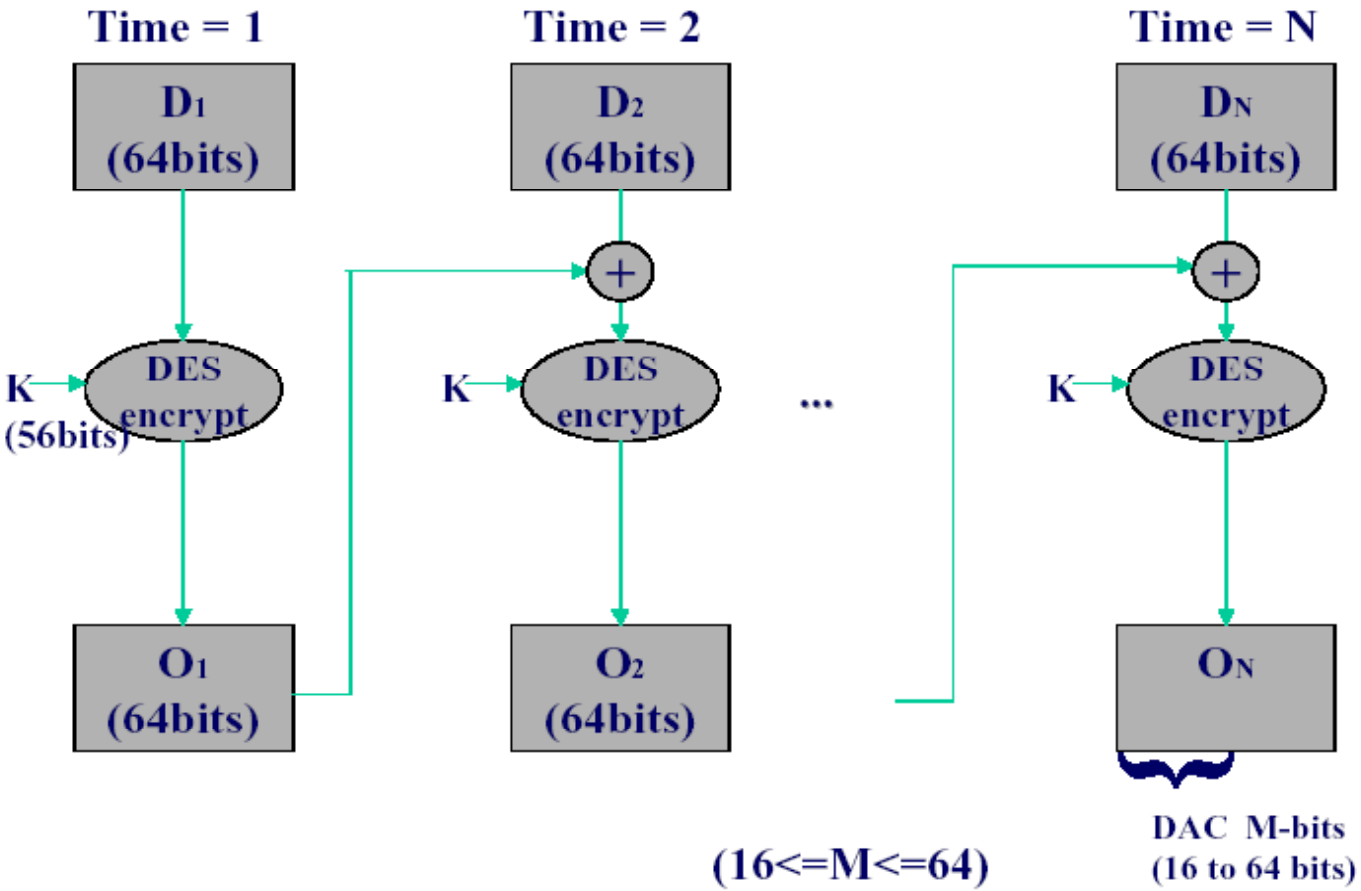
消息认证与密文相关

（四）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：

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



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

生日攻击

- （第I类生日攻击）H有n个输出，H(x)是一个特定的输出，即已知x和H(x)。如果对H随机取k个输入，至少有一个y使H(y)=H(x)的概率为0.5时，k有多大?

  H(y)=H(x)的概率为1/n，不等的概率为1-1/n。取k个值都不等的为$[1-1/n]^k$。至少有一个相等的概率为1－ $[1-1/n]^k$。

生日攻击

由于

$$(1+x)^k \approx 1+kx, \text{其中} |x| << 1, \text{ 可得}$$

$$1-\left(1-\frac{1}{\mathrm{n}}\right)^k \approx 1-(1+k*\frac{-1}{n}) = 1-1+\frac{k}{n} = \frac{k}{n}$$

若使上述概率=0.5，则k=n/2。特别地，如果H的输出为m bits长，则输出的个数n=2^m, 此时k=n/2=2^{m-1}。

生日悖论

- 在一个会场参加会议的人中，问使参会人员中至少有两个同日生的概率超过0.5的参会人数仅为23人。

k个人所有的生日可能性为:

$$X = 365^{k}$$

k个人的生日都不相同的可能性为:

$$Y = P_{365}^{k} = \frac{365!}{(365-k)!}$$

k个人的生日都不相同的概率为:

$$\frac{X}{Y} = \frac{365!}{(365-k)!365^{k}}$$

生日悖论

k个人的生日中，至少有2个人的生日相同的概率为：

$$P = 1 - \frac{X}{Y} = 1 - \frac{365!}{(365-k)!365^k}$$

当k=23时，P=0.5073，即只要有23个人，则至少有2个人的生日相同的概率大于0.5。当k=100时，P=0.9999997。

这种情况与我们想象的有很大不同，所以称为生日悖论。

生日悖论

将生日悖论推广为下述问题：已知一个在1~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2个取值相同的概率大于0.5，则k至少多大?

$$P(n,k)=1-\frac{n!}{(n-k)!n^k}$$，令$P(n,k)>0.5$可得

$$k=1.18\sqrt{n}\approx\sqrt{n}$$

若n=365, 则

$$k=1.18\sqrt{365}=22.54$$

生日悖论

对MD5算法，其输出摘要为128bits，共有2^128可能性，相当于n=2^128，则要找出至少2个消息的HASH值相同的概率大于0.5，则利用生日攻击需要进行的计算量k或称为时间复杂度为:

$$k \approx \sqrt{n} = \sqrt{2^{128}} = 2^{64}$$

对SHA-1算法，其输出摘要为160bits, 则利用生日攻击需要进行的计算量k或称为时间复杂度为:

$$k \approx \sqrt{n} = \sqrt{2^{160}} = 2^{80}$$

生日悖论

王小云所提的杂凑冲撞演算法只需少于2^69步骤，远少于一直以为所需的2^80步。王小云教授又破解了另一国际密码SHA－1。因为SHA－1在美国等国际社会有更加广泛的应用

2005年8月，王小云、姚期智，以及姚期智妻子姚储枫联手于国际密码讨论年会尾声部份提出SHA-1杂凑函数杂凑冲撞演算法的改良版。此改良版使破解SHA-1时间缩短为2^63步（原来需要2^69步）。

2006年6月8日，于中国科学院第13次院士大会和中国工程院第8次院士大会上以**“国际通用Hash函数的破解”**获颁陈嘉庚科学奖信息技术科学奖。