第3章 单向散列函数

散列值像指纹

- 世界上很难找到两个相同指纹的人，我们可用指纹代表一个人。
- 如果把一条消息看成一个人，消息的散列值就是指纹，由此我们可用散列值代表一条消息。
- 单向散列函数是数字签名中的一个关键环节, 可以大大缩短签名时间并提高安全性, 另外在消息完整性检测, 内存的散布分配, 软件系统中帐号口令的安全存储，单向散列函数也有重要应用。

3.1 单向散列函数概述

- 单向散列函数是将一个消息以不可逆的方式将它转换成一段（通常更小）密文; 也可以简单的理解为取一串输入码（称为消息），并把它们转化为长度较短、位数固定的输出序列即散列值（也称为消息摘要）的过程。
- 散列函数值可以说是对明文的一种“指纹”或是“摘要”，是明文的压缩版，是明文的映射，可看成是明文的代表，就是用小的散列值代表大的明文。
- 一小代大也有例外！像口令的散列存储。

- 所谓的单向散列函数(Hash Function，又称哈希函数、杂凑函数)，是将任意长度的消息M映射成一个固定长度散列值h(设长度为m)的函数H：
  - h=H(M)
- 散列函数要具有单向性，则必须满足如下特性：
  - 给定M，很容易计算h。
  - 给定h，根据H(M)=h反推M很难。
  - 给定M，要找到另一消息M'并满足H(M)=H(M')很难。
- 在某些应用中，单向散列函数还需要满足抗碰撞(Collision)的条件：要找到两个随机的消息M和M'，使H(M)=H(M')很难。

散列函数工作模式

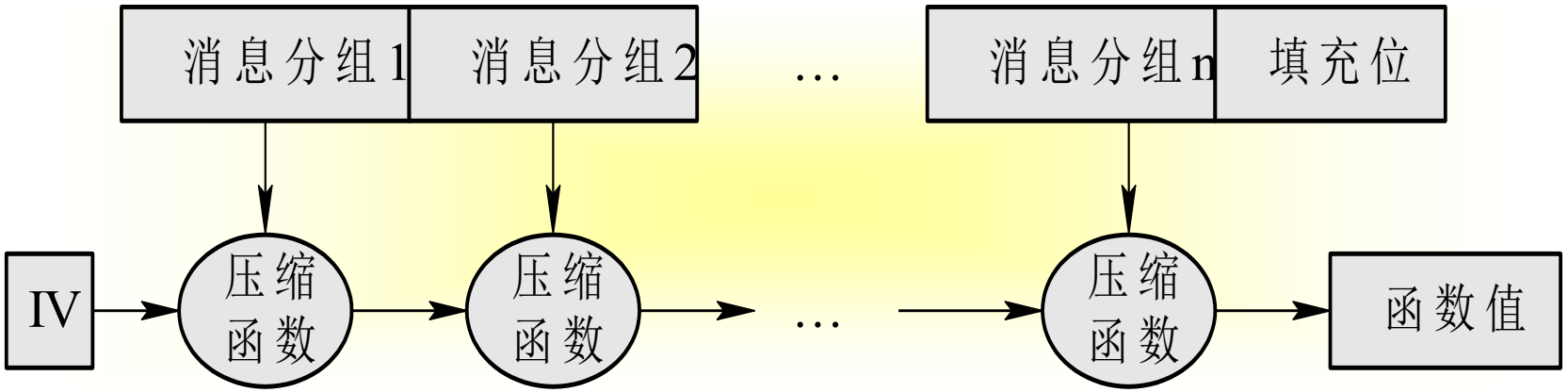


图3-1 单向散列函数工作模式

3.2 MD5 算 法

3.2.1 算法

MD表示消息摘要(Message Digest)。MD5是MD4的改进版，该算法对输入的任意长度消息产生128位散列值(或消息摘要)。MD5算法可用图3-2表示。

1) 按512位的分组处理输入消息

2) 附加填充位

3) 消息长度64

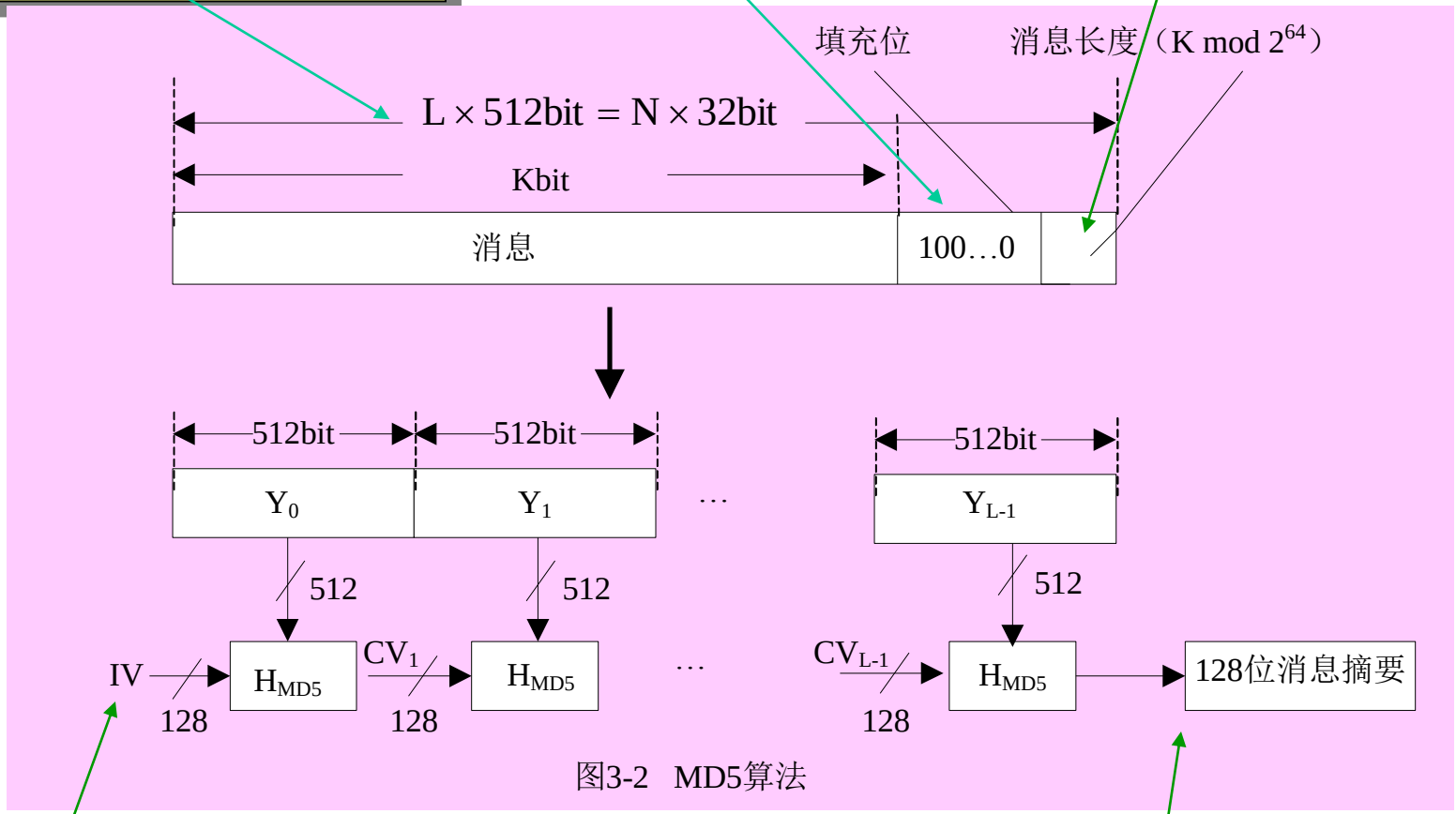


图3-2 MD5算法

4) 初始化MD缓冲区

图3-2 MD5算法

5) 输出

由上图可知，MD5算法包括以下五个步骤。

1） 消息长度

将原消息长度的64位表示附加在填充后的消息后面。当原消息长度大于2^{64}时，用消息长度mod 2^{64}填充。这时，总长度恰好是512的整数倍。令M[0 1...N-1]为填充后消息的各个字(每字为32位)，N是16的倍数。

2） 附加填充位

首先填充消息，使其长度为一个比512的倍数小64位的数。填充方法：在消息后面填充一位1，然后填充所需数量的0。填充位的位数是1～511。

填充消息长度=512-(k+64) mod 512

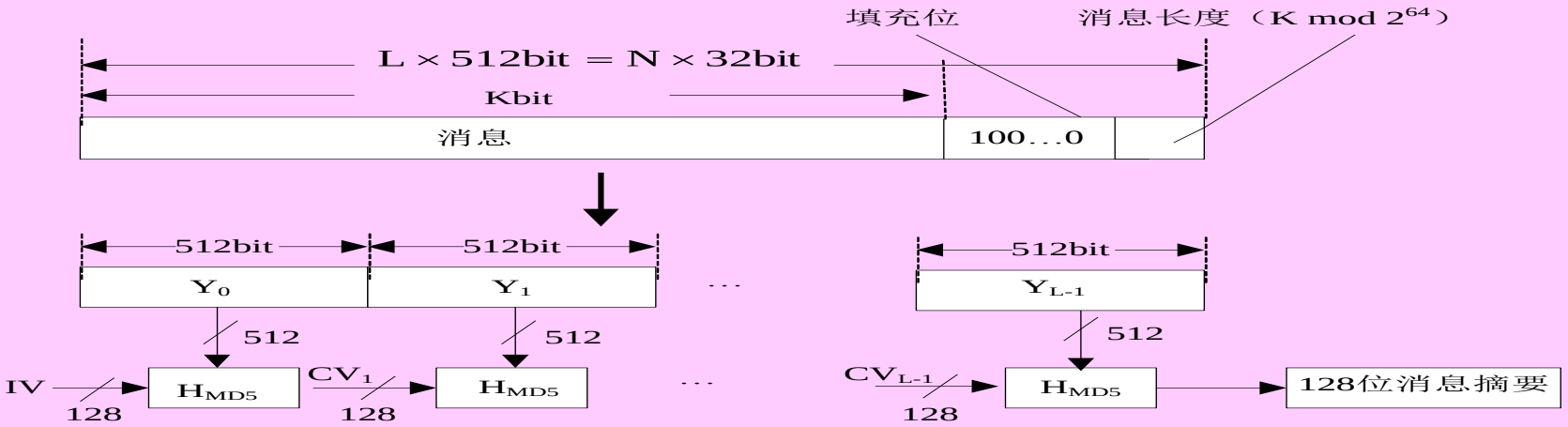


图3-2 MD5算法

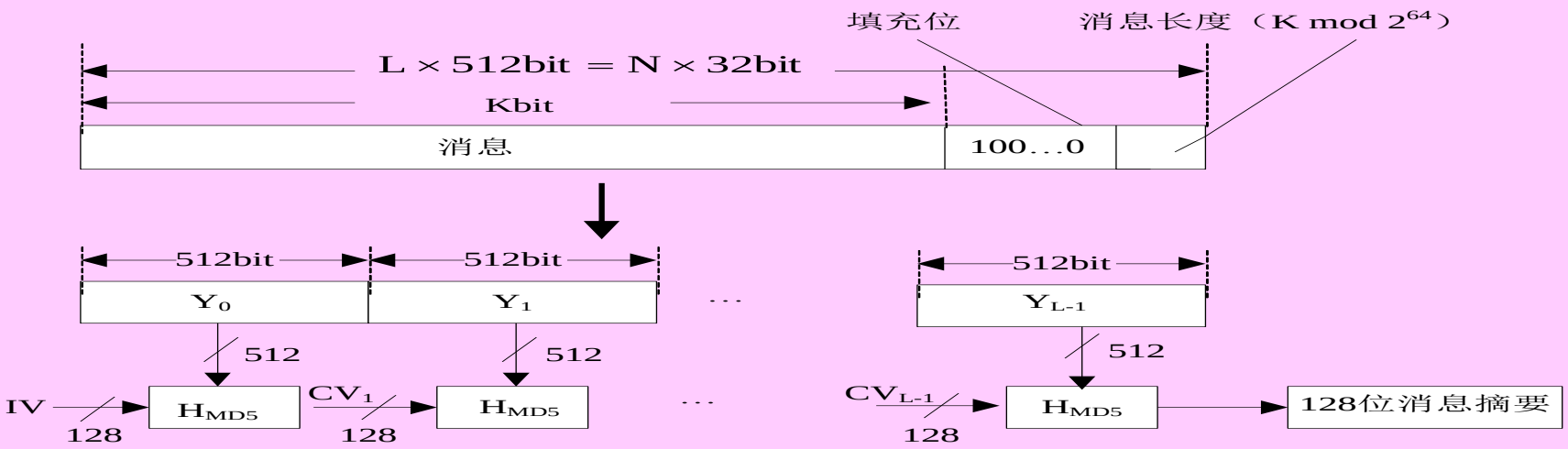


图3-2　MD5算法

3) 初始化MD缓冲区

初始化用于计算消息摘要的128位缓冲区。这个缓冲区由四个32位寄存器A、B、C、D表示。寄存器的初始化值为(按低位字节在前的顺序存放)：

A: 01　23　45　67

B: 89　ab　cd　ef

C: fe　dc　ba　98

D: 76　54　32　10

4) 按512位的分组处理输入消息

这一步为MD5的主循环，包括四轮，如图3-3所示。每个循环都以当前的正在处理的512比特分组Y_q和128比特缓冲值ABCD为输入，然后更新缓冲内容。

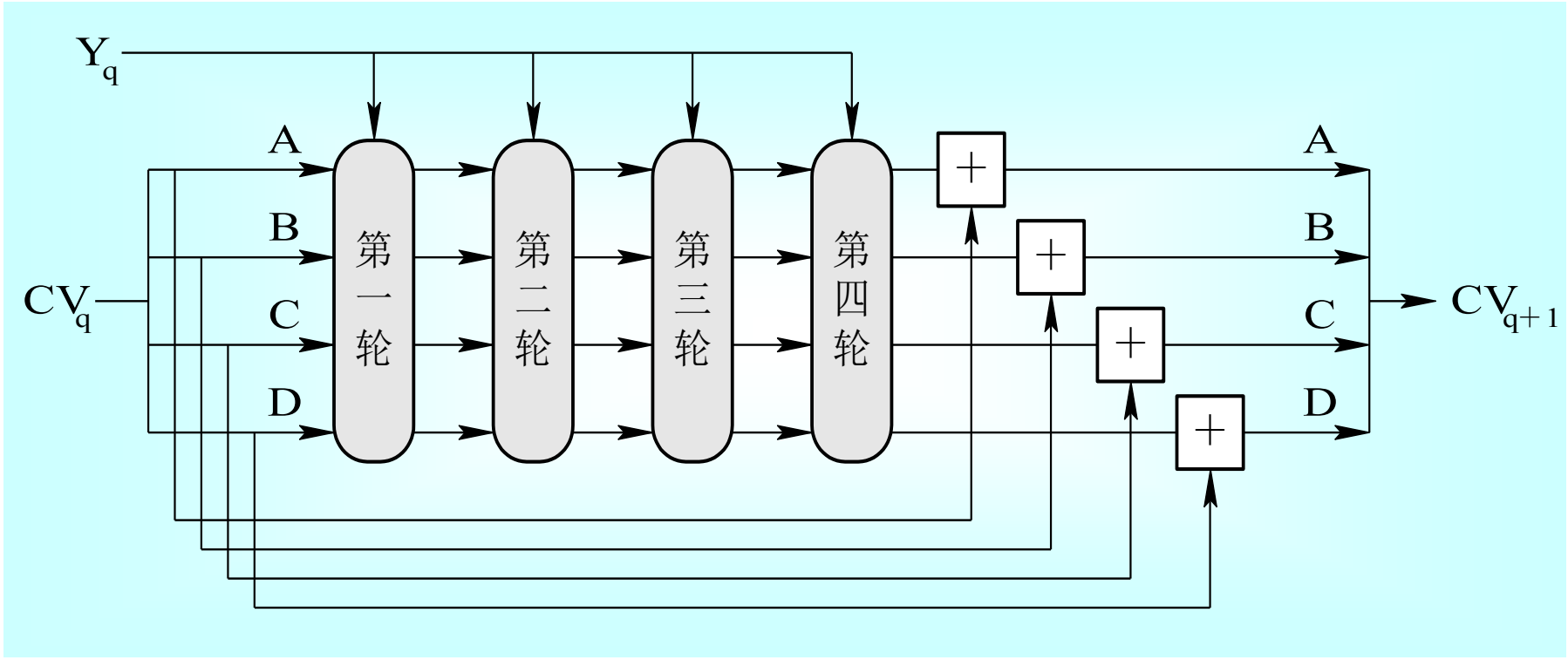
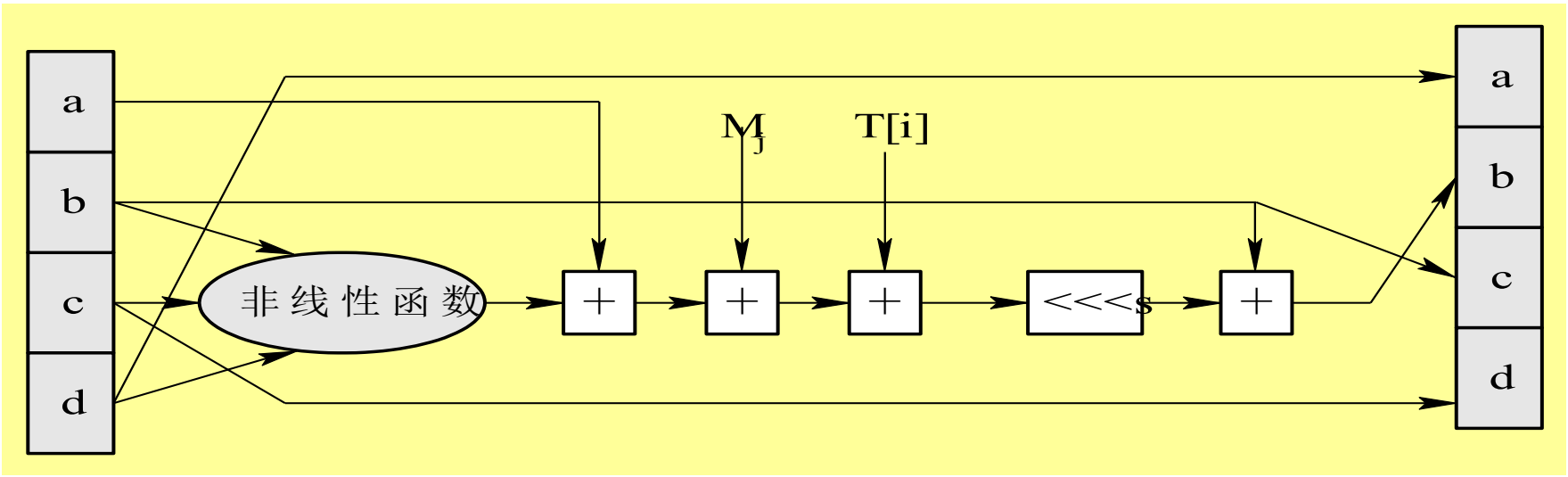


图3-3 单个512比特分组的MD5主循环处理

图3-3中，四轮的操作类似，每一轮进行16次操作。各轮的操作过程如图3-4所示。



j:0—15, i:1—64(共四轮， 每一次用的都不同)

图3-4 MD5某一轮的1次执行过程

四轮操作的不同之处在于每轮使用的非线性函数不同，在第一轮操作之前，首先把A、B、C、D复制到另外的变量a、b、c、d中。这四个非线性函数分别为(其输入/输出均为32位字)：

$F(X,Y,Z) = (X \wedge Y) \vee ((\sim X) \wedge Z)$

$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge (\sim Z))$

$H(X,Y,Z) = X \oplus Y \oplus Z$

$I(X,Y,Z) = Y \oplus (X \vee (\sim Z))$

其中，$\wedge$表示按位与；$\vee$表示按位或；$\sim$表示按位反；$\oplus$表示按位异或。

- 此外，由图3-4可知，这一步中还用到了一个有64个元素的表T[1..64]，$T[i]=2^{32}\times abs(sin(i))$，i的单位为弧度。
- 根据以上描述，将这一步骤的处理过程归纳如下：
- for i = 0 to N/16−1 do
- /* 每次循环处理16个字，即512字节的消息分组*/
- /*把A存为AA，B存为BB，C存为CC，D存为DD*/
- **AA = A**
- **BB = B**
- **CC = C**
- **DD = D**

/* 第一轮*/

/* 令[abcd k s i]表示操作

$$b = b + ((a + F(b,c,d) + M[k] + T[i]) <<< s)$$

其中，Y<<<s表示Y循环左移s位*/

/* 完成下列16个操作*/

[ABCD 0 7 1]	[DABC 1 12 2]	[CDAB 2 17 3]	[BCDA 3 22 4]
[ABCD 4 7 5]	[DABC 5 12 6]	[CDAB 6 17 7]	[BCDA 7 22 8]
[ABCD 8 7 9]	[DABC 9 12 10]	[CDAB 10 17 11]	[BCDA 11 22 12]
[ABCD 12 7 13]	[DABC 13 12 14]	[CDAB 14 17 15]	[BCDA 15 22 16]

/* 第二轮*/

/*令[abcd k s i]表示操作

b = b + ((a + G(b,c,d) + M[k] + T[i]) <<< s)*/

/*完成下列16个操作*/

[ABCD 1 5 17] [DABC 6 9 18] [CDAB 11 14 19] [BCDA 0 20 20]

[ABCD 5 5 21] [DABC 10 9 22] [CDAB 15 14 23] [BCDA 4 20 24]

[ABCD 9 5 25] [DABC 14 9 26] [CDAB 3 14 27] [BCDA 8 20 28]

[ABCD 13 5 29] [DABC 2 9 30] [CDAB 7 14 31] [BCDA 12 20 32]

/*第三轮*/

/*令[abcd k s t]表示操作

b= b + ((a + H(b,c,d) + M[k] + T[i]) <<< s)*/

/*完成以下16个操作*/

[ABCD 5 4 33] [DABC 8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]

[ABCD 1 4 37] [DABC 4 11 38] [CDAB 7 16 39] [BCDA 10 23 40]

[ABCD 13 4 41] [DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]

[ABCD 9 4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA 2 23 48]

/*第四轮*/

/*令[abcd k s t]表示操作

b = b + ((a + I(b,c,d) +M[k] + T[i]) <<< s) */

/*完成以下16个操作*/

[ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21 52]

[ABCD 12 6 53] [DABC 3 10 54] [CDAB 10 15 55] [BCDA 1 21 56]

[ABCD 8 6 57] [DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]

[ABCD 4 6 61] [DABC 11 10 62] [CDAB 2 15 63] [BCDA 9 21 64]

A = A + AA

B = B + BB

C = C + CC

D = D + DD

end /*i循环*/

5) 输出

由A、B、C、D四个寄存器的输出按低位字节在前的顺序(即以A的低字节开始、D的高字节结束)得到128位的消息摘要。

以上就是对MD5算法的描述。MD5算法的运算均为基本运算，比较容易实现且速度很快。

3.2.2 举例

我们以求字符串“abc”的MD5散列值为例来说明上面描述的过程。“abc”的二进制表示为01100001 01100010 01100011。

1. 填充消息

消息长24，先填充1位1，然后填充423位0，再用消息长24，即0x00000000 00000018填充，则：

M[0]=61626380 M[1]=00000000 M[2]=00000000 M[3]=00000000

M[4]=00000000 M[5]=00000000 M[6]=00000000 M[7]=00000000

M[8]=00000000 M[9]=00000000 M[10]=00000000 M[11]=00000000

M[12]=00000000 M[13]=00000000 M[14]=00000000 M[15]=00000018

2. 初始化

A: 01	23	45	67
B: 89	ab	cd	ef
C: fe	dc	ba	98
D: 76	54	32	10

3. 主循环

利用3.2.1节中描述的过程对字块1（本例只有一个字块）进行处理。变量a、b、c、d每一次计算后的中间值不再详细列出。

4. 输出

消息摘要=90015098 3cd24fb0 d6963f7d 28e17f72

问题

- 计算Hash的自变量由3部分组成：
- 消息本身+填充+消息本身长度
- 为什么不是：
- 消息本身+消息本身长度+填充

3.3 安全散列函数(SHA-1)

3.3.1 SHA家族

- SHA (Secure Hash Algorithm，安全散列算法) 家族是美国国家安全局 (NSA) 设计，美国国家标准与技术研究院 (NIST) 发布的一系列密码散列函数。
- 1993年发布SHA 家族的第一个成员（SHA-0），1995年发布了SHA-1。
- SHA-1在许多安全协定中广为使用，包括TLS和SSL、PGP、SSH(Secure Shell)、S/MIME和IPsec，曾被视为是MD5的后继者。

SHA-0和SHA-1相继被攻破和攻击，NIST发布了另外四种变体

SHA算法参数比较

算法名称		输出长度(bits)	分块长度（bits）	最大消息长度(bits)	字长(bits)	步数
SHA-0		160	512	$2^{64}-1$	32	80
SHA-1						
SHA-2	SHA-256/224	256/224	512	$2^{64}-1$	32	64
	SHA-512/384	512/384	1024	$2^{128}-1$	64	80

3.3.2 SHA-1算法

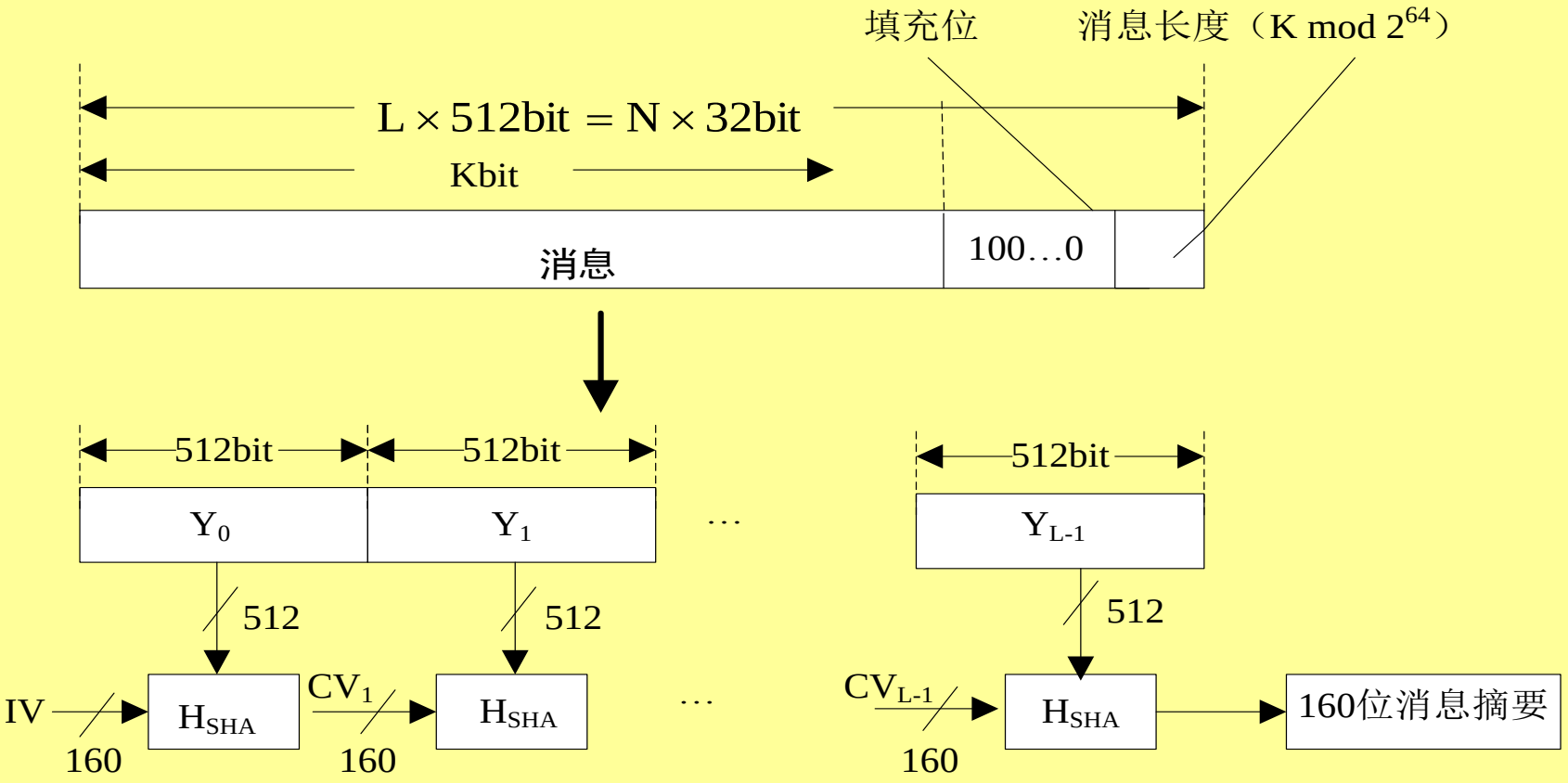


图3-5 SHA-1算法

SHA－1的输入为长度小于2^{64}位的消息（若大于，用mod 即可），输出为160位的消息摘要。具体过程如下。

1) 填充消息

首先将消息填充为512位的整数倍，填充方法和MD5完全相同：先填充一个1，然后填充一定数量的0，使其长度比512的倍数少64位；接下来用原消息长度的64位表示填充。这样，消息长度就成为512的整数倍。以M_0、M_1、…、M_{n-1}表示填充后消息的各个字块(每字块为16个32位字)。

2) 初始化缓冲区

在运算过程中，SHA要用到两个缓冲区，两个缓冲区均有五个32位的寄存器。第一个缓冲区标记为A、B、C、D、E；第二个缓冲区标记为H_0、H_1、H_2、H_3、H_4。此外，运算过程中还用到一个标记为W_0、W_1、…、W_{79}的80个32位字序列和一个单字的缓冲区TEMP。在运算之前，初始化$\{H_j\}$：

H_0 = 0x67452301

H_1 = 0xEFCDAB89

H_2 = 0x98BADCFE

H_3 = 0x10325476

H_4 = 0xC3D2E1F0

3) 按512位的分组处理输入消息

SHA运算主循环包括四轮，每轮20次操作。SHA用到一个逻辑函数序列f_0、f_1、…、f_{79}。每个逻辑函数的输入为三个32位字，输出为一个32位字。定义如下(B、C、D均为32位字)：

$$f_t(B,C,D) = (B \wedge C) \vee (\sim B \wedge D) \qquad (0 \le t \le 19)$$

$$f_t(B,C,D) = B \oplus C \oplus D \qquad (20 \le t \le 39)$$

$$f_t(B,C,D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) \qquad (40 \le t \le 59)$$

$$f_t(B,C,D) = B \oplus C \oplus D \qquad (60 \le t \le 79)$$

其中，运算符的定义与3.1节中MD5运算中的相同。

SHA运算中还用到了常数字序列K_0、K_1、…、K_{79}，其值为

K_t = 0x5A827999 $(0 \le t \le 19)$

K_t = 0x6ED9EBA1 $(20 \le t \le 39)$

K_t = 0x8F1BBCDC $(40 \le t \le 59)$

K_t = 0xCA62C1D6 $(60 \le t \le 79)$

SHA算法按如下步骤处理每个字块M_i:

(1) 把M_i分为16个字W_0、W_1、…、W_{15}。

(2) for t =16 to 79 do

let $W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) <<< 1$

(3) Let $A = H_0$，$B = H_1$，$C = H_2$，$D = H_3$，$E = H_4$

(4) for t =0 to 79 do

$TEMP = (A <<< 5) + f_t(B, C, D) + E + W_t + K_t$;

$E = D$; $D = C$; $C = (B <<< 30)$; $B = A$; $A = TEMP$;

(5) Let $H_0 = H_0 + A$, $H_1 = H_1 + B$, $H_2 = H_2 + C$, $H_3 = H_3 + D$, $H_4 = H_4 + E$

4) **输出**-- 在处理完M_n后，160位的消息摘要为H_0、H_1、H_2、H_3、H_4级联的结果。

3.3.3 举例

我们以求字符串‘abc’的SHA–1散列值为例来说明上面描述的过程。‘abc’的二进制表示为

a b c

01100001 01100010 01100011。

(1) 填充消息：消息长l=24，先填充1位1，然后填充423位0，再用消息长24，即0x00000000 00000018（64位）填充。

(2) 初始化：

H_0 = 0x67452301 H_1 = 0xEFCDAB89

H_2 = 0x98BADCFE H_3 = 0x10325476

H_4 = 0xC3D2E1F0

(3) 主循环：处理消息字块1(本例中只有1个字块)，分成16个字：（01100001 01100010 01100011 24）

W[0]=61626380 W[1]=00000000 W[2]=00000000 W[3]=00000000

W[4]=00000000 W[5]=00000000 W[6]=00000000 W[7]=00000000

W[8]=00000000 W[9]=00000000 W[10]=00000000 W[11]=00000000

W[12]=00000000 W[13]=00000000 W[14]=00000000 W[15]=00000018

然后根据3.2.1节中描述的过程计算，其中，循环“for t = 0 to 79”中，各步A、B、C、D、E 的值如下：

	A	B	C	D	E
t = 0:	0116FC33	67452301	7BF36AE2	98BADCFE	10325476
t = 1:	8990536D	0116FC33	59D148C0	7BF36AE2	98BADCFE
t = 2:	A1390F08	8990536D	C045BF0C	59D148C0	7BF36AE2
t = 3:	CDD8E11B	A1390F08	626414DB	C045BF0C	59D148C0
t = 4:	CFD499DE	CDD8E11B	284E43C2	626414DB	C045BF0C
t = 5:	3FC7CA40	CFD499DE	F3763846	284E43C2	626414DB
t = 6:	993E30C1	3FC7CA40	B3F52677	F3763846	284E43C2
t = 7:	9E8C07D4	993E30C1	0FF1F290	B3F52677	F3763846
t = 8:	4B6AE328	9E8C07D4	664F8C30	0FF1F290	B3F52677
t = 9:	8351F929	4B6AE328	27A301F5	664F8C30	0FF1F290
t =10:	FBDA9E89	8351F929	12DAB8CA	27A301F5	664F8C30

字块1处理完后，$\{H_i\}$的值为：

H_0 = 67452301 + 42541B35 = A9993E36

H_1 = EFCDAB89 + 5738D5E1 = 4706816A

H_2 = 98BADCFE + 21834873 = BA3E2571

H_3 = 10325476 + 681E6DF6 = 7850C26C

H_4 = C3D2E1F0 + D8FDF6AD = 9CD0D89D

(4) 输出：消息摘要 =

A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D

也就是说abc的散列函数值为：

3.3.4 SHA–1与MD5的比较

SHA–1与MD5的比较如表 所示。

	SHA-1	MD5
Hash值长度		
分组处理长度		
步数		
最大消息长		
非线性函数		
常数个数		

Back

3.3.3 SHA–1与MD5的比较

SHA–1与MD5的比较如表3-1所示。

	SHA-1	MD5
Hash值长度	160 bit	128 bit
分组处理长度		
步数		
最大消息长		
非线性函数		
常数个数		

表3-1 SHA-1与MD5的比较

3.3.3 SHA–1与MD5的比较

SHA–1与MD5的比较如表3-1所示。

	SHA-1	MD5
Hash值长度	160 bit	128 bit
分组处理长度	512 bit	512 bit
步数		
最大消息长		
非线性函数		
常数个数		

表3-1 SHA-1与MD5的比较

3.3.3 SHA–1与MD5的比较

SHA–1与MD5的比较如表3-1所示。

	SHA-1	MD5
Hash值长度	160 bit	128 bit
分组处理长度	512 bit	512 bit
步数	80(4×20)	64(4×16)
最大消息长		
非线性函数		
常数个数		

表3-1 SHA-1与MD5的比较

3.3.3 SHA–1与MD5的比较

SHA–1与MD5的比较如表3-1所示。

	SHA-1	MD5
Hash值长度	160 bit	128 bit
分组处理长度	512 bit	512 bit
步数	80(4×20)	64(4×16)
最大消息长	$\leq 2^{64}$ bit	不限
非线性函数		
常数个数		

表3-1 SHA-1与MD5的比较

3.3.4 SHA–1与MD5的比较

SHA–1与MD5的比较如表3-1所示。

ft (B,C,D) = (B ∧ C) ∨(~B ∧ D)　　(0≤t≤19)
ft (B,C,D) = B ⊕ C ⊕ D　　(20≤t≤39)
ft (B,C,D) = (B ∧ C) ∨(B ∧ D) ∨(C ∧ D)　　(40≤t≤59)
ft (B,C,D) = B ⊕ C ⊕ D　　(60≤t≤79)

分组处理长	512 bit	512 bit
步数	80(4×20)	64(4×16)
最大消息长	<2^{64} bit	不限
非线性函数	3(第2、4轮相同)	4
常数个数		

F(X,Y,Z) = (X ∧ Y)∨ ((～X) ∧ Z)
G(X,Y,Z) = (X ∧ Z) ∨(Y ∧ (～Z))
H(X,Y,Z) = X ⊕ Y ⊕Z
I(X,Y,Z) = Y ⊕(X ∨(～Z))

表3

Back

3.3.4 SHA–1与MD5的比较

SHA–1与MD5的比较如表3-1所示。

	SHA-1	MD5
Hash值长度	160 bit	128 bit
分组处理长度	512 bit	512 bit
步数	80(4×20)	64(4×16)
最大消息长	$\leq 2^{64}$ bit	不限
非线性函数	**3(第2、4轮相同)**	4
常数个数	4	64

表3-1 SHA-1与MD5的比较

根据各项特征，简要地说明它们间的不同。

（1）安全性：SHA-1所产生的摘要较MD5长32位。若两种散列函数在结构上没有任何问题的话，SHA-1比MD5更安全。

（2）速度：两种方法都考虑了以32位处理器为基础的系统结构，但SHA-1的运算步骤较MD5多了16步，而且SHA-1记录单元的长度较MD5多了32位。因此若是以硬件来实现SHA-1，其速度大约较MD5慢25%。

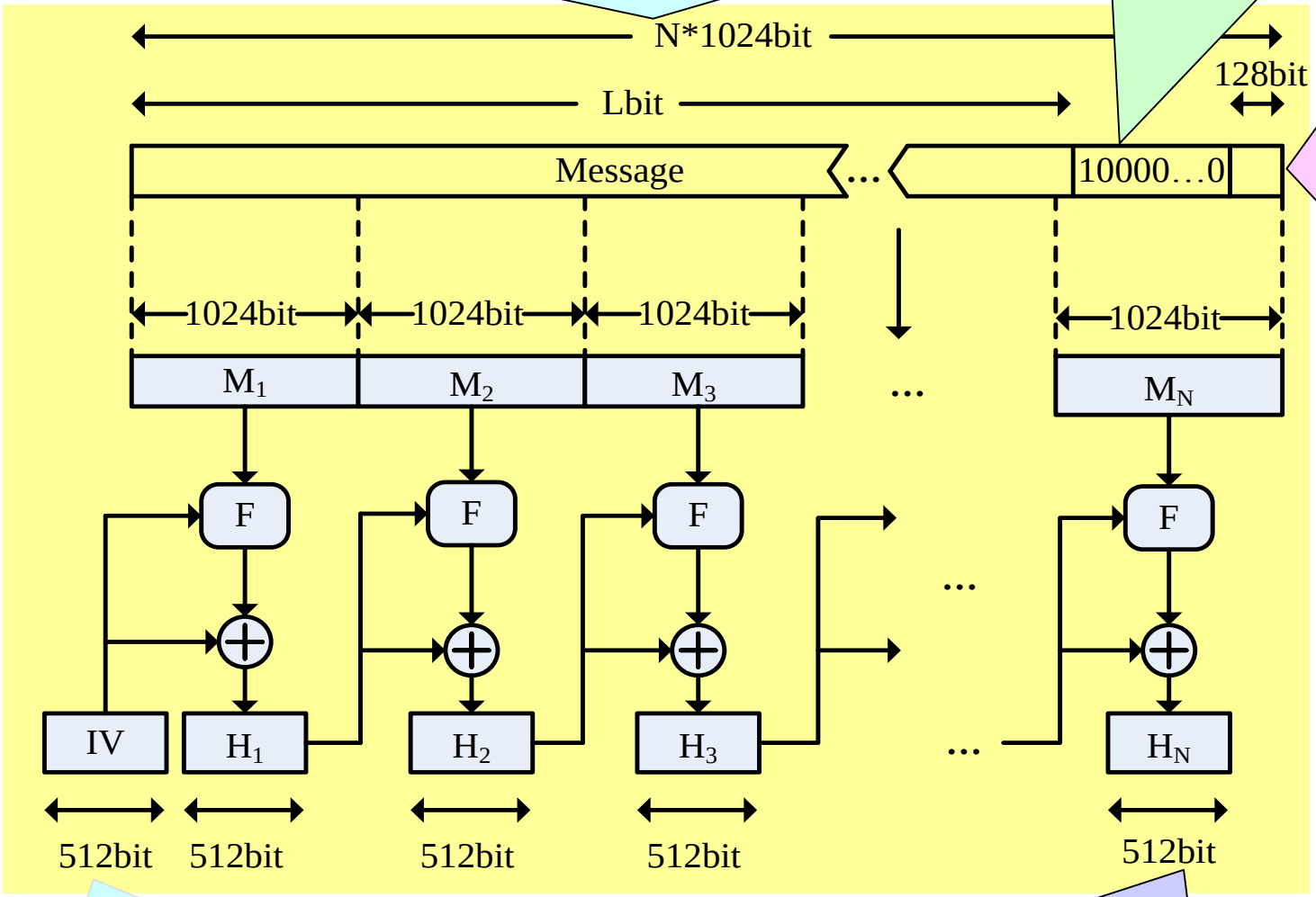
（3）简易性：两种方法都相当的简单，在实现上不需要很复杂的程序或是大量的存储空间。然而总体上来讲，SHA-1每一步的操作都较MD5简单。

3.3.5 SHA-512算法

- **SHA-512输出消息摘要的长度为512位。输入被处理成1024位的块，处理过程由以下步骤组成：**
- 针对64bit的寄存器

（4）以1024bit的块(16个字)为单位处理消息，算法的核心是具有80轮运算的模块

（2）填加长度。 填充后 消息的长度恰为1024bit的整数倍。



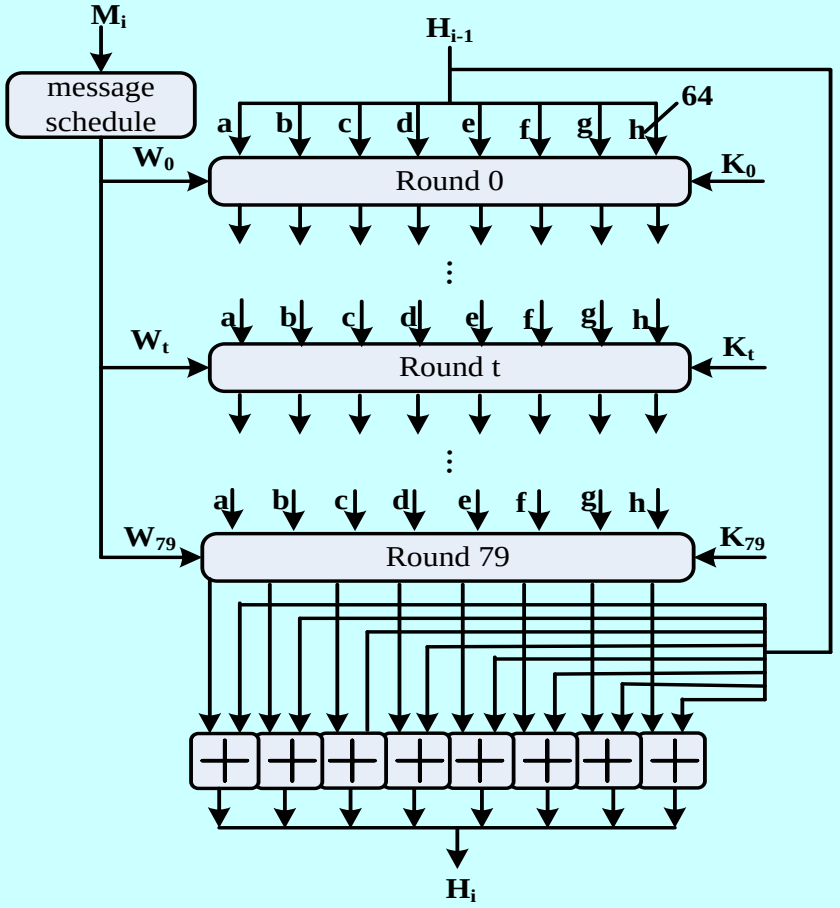
（1）填充。若k是消息长度，则计算填充消息长度的公式是：1024-（k+128）mod 1024，128位用来放消息长度。

（3）初始Hash缓冲区。一个512bit的块用于保存Hash函数的中间值和最终结果。该缓冲区用8个64bit的寄存器

（5）输出。所有的N个1024bit分组都处理完以后，从第N阶段输出的是512bit的消息摘要。

（4）以1024bit块(16个字)为单位处理消息，算法的核心是具有80轮运算的模块 。

- 第一轮，缓冲区里的值是中间的散列值H_{i-1}。
- 每一轮，如t轮，使用一个64bit的值W_t，其中0≤t≤79表示轮数，该值由当前被处理的1024bit消息分组M_i导出。
- 每一轮还将使用附加的常数K_t。
  常数获得：前80个素数取三次方根，取小数部分的前64bit。
- 第80轮的输出和第一轮输入H_{i-1}相加产生H_i。
  缓冲区里的8个字和H_{i-1}里的相应字独立进行模2^{64}的加法运算。



Hash函数在Web信息系统中的应用—身份认证

- 用户注册时第一次设置密码（password），密码不直接写数据库。？
- 系统用Hash函数计算用户密码的Hash值，并保存到数据库中。
- 以后用户登录系统时，用户输入密码后， 系统首先计算它的Hash值，然后从数据库中取出对应的Hash值与其比较， 若相同视为合法用户， 否则不合法。
- 合法用户允许访问系统， 不合法不能访问系统。

用户+密码 这种身份认证方式简单、 高效，但安全性不是很高。

3.4 消息认证码(MAC)

- 目前的认证技术有对用户的认证和对消息的认证两种方式。
- 1 用户认证用于鉴别用户的身份是否是合法用户；
- 2 消息认证就是验证所收到的消息确实是来自真正的发送方且未被修改的消息，也可以验证消息的顺序和及时性。
- 消息认证实际上是对消息本身产生一个冗余的信息—MAC（Message Authentication Code，消息认证码）附加在消息后面。
- 其组成是：消息+消息认证码，以认证消息的完整性

3.4.1 消息认证码基本概念

与密钥相关的单向散列函数通常称为MAC，即消息认证码：

$$MAC=C_K(M)$$

其中，M为可变长的消息；K为通信双方共享的密钥；C为单向函数。

MAC可为拥有共享密钥的双方在通信中验证消息的完整性；也可被单个用户用来验证他的文件是否被改动 。

3.4.2 消息的完整性验证

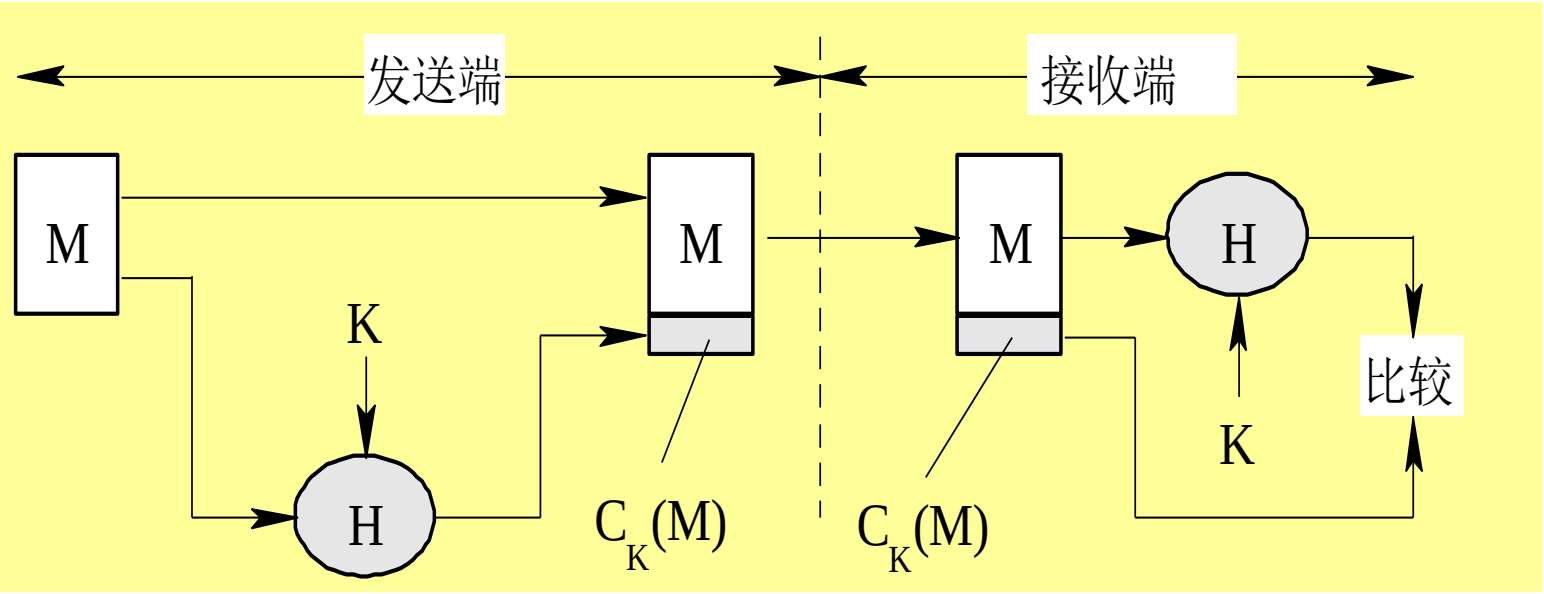


图3-6 MAC应用于消息认证

若没有K，能否验证消息的完整性？

不能

如攻击者将消息M 和摘要都替换了，将不能验证原消息的完整性

3.4.3 HMAC算法

- RFC 2104定义的全称为**Keyed-Hash Message Authentication Code**，它用一个秘密密钥来产生和验证**MAC**
- 小知识：**Request For Comments**（**RFC**）文件是由**Internet Society**（**ISOC**）赞助发行。基本的互联网通信协议在**RFC**文件内都有详细说明。几乎所有的互联网标准都收录在**RFC**文件之中。

将数值0x36重复B次。 B:计算消息摘要时输入块的字节长度

在密钥K的左边项加0

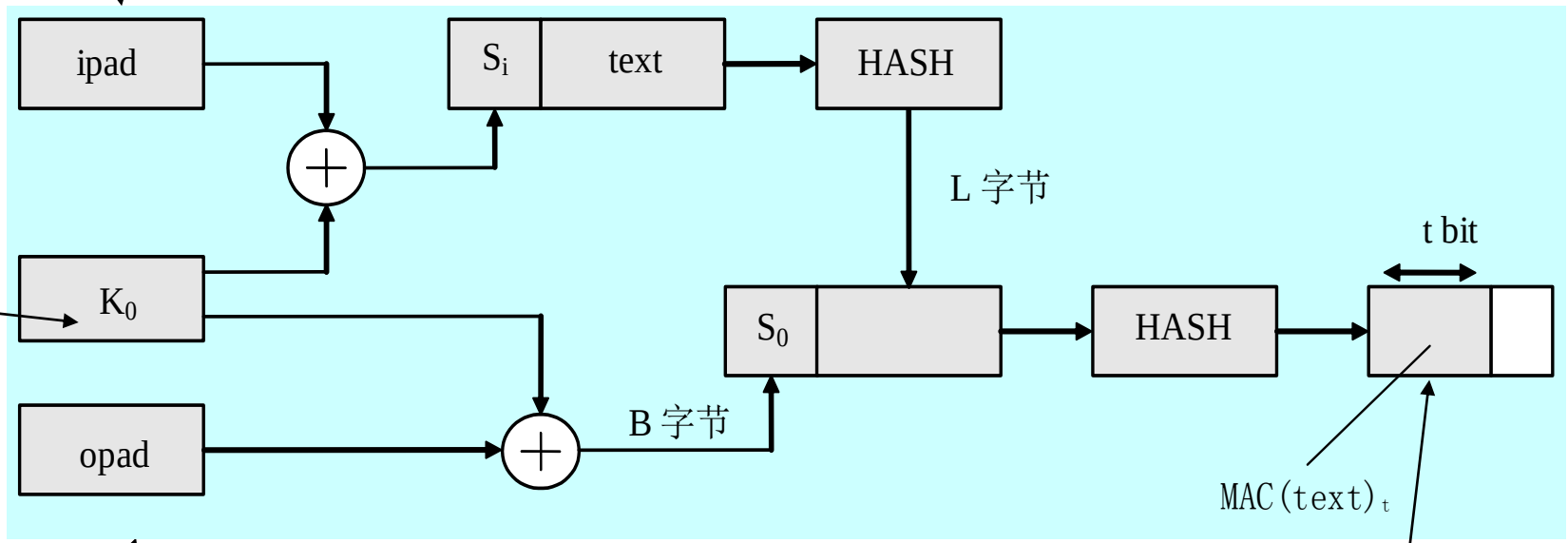
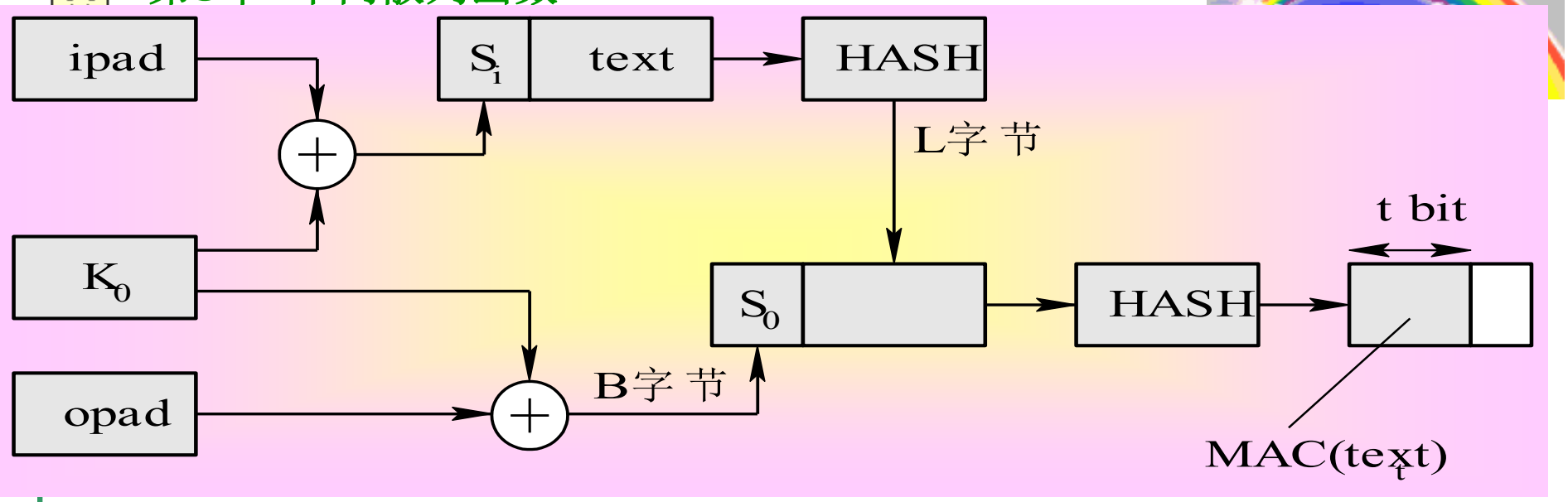


图3-7 HMAC结构

最左边t字节作为MAC

将数值0x5c重复B次



(1) 如果K的长度等于B，设置K_0 = K并跳转到第(4)步。

(2) 如果K的长度大于B，对K求散列值：K_0 = H(K)。

(3) 如果K的长度小于B，在K的左边附加0得到B字节的K_0。

(4) 执行$K_0 \oplus$ ipad。

(5) 将数据text附加在第(4)步结果的后面：

$$(K_0 \oplus \text{ipad}) \,||\, \text{text}$$

(6) 将H应用于第(5)步的结果：

$$H((K_0 \oplus \text{ipad}) \,||\, \text{text})$$

(7) 执行$K_0 \oplus \text{opad}$。

(8) 把第(6)步的结果附加在第(7)步的结果后面：

$$(K_0 \oplus \text{opad}) \,||\, H((K_0 \oplus \text{ipad}) \,||\, \text{text})$$

(9) 将H应用于第(8)步的结果：

$$H((K_0 \oplus opad) \| H((K_0 \oplus ipad) \| text))$$

(10) 选择第(9)步结果的最左边t字节作为MAC。

HMAC算法可以和任何密码散列函数结合使用，而且对HMAC实现作很小的修改就可用一个散列函数H代替原来的散列函数H'。

Back

3.5 对单向散列函数的攻击

- 对单向散列函数攻击的目的在于破坏单向散列函数的某些特性.
- 比如可以根据输出求得输入，找到一条新消息使它的输出与原消息的输出相同，或者找到不同的两个消息，使它们的输出相同。

3.5.1 字典攻击

- 字典攻击，对用单向散列函数加密的口令文件特别有效。
- 攻击者编制含有多达几十万个常用口令的表，然后用单向散列函数对所有口令进行运算，并将结果存储到文件中。
- 攻击者非法获得加密的口令文件后，将比较这两个文件，观察是否有匹配的口令密文。
- 字典式攻击，成功率非常高。这是因为大多数人缺乏安全意识和想象力，选择的口令过于简单。编制巨大的口令表是个问题吗？
- 从网上就能找到，热心的黑客们已经把它完善得很好了。

抵御散列函数的字典攻击所采取策略

- 1、Salt(添加符)是使这种攻击更困难的一种方法。加盐
- Salt是一随机字符串，它与口令连接在一起，再用单向散列函数对其运算。然后将Salt值和单向散列函数运算的结果存入主机数据库中。
  - 攻击者必须对所有可能的Salt值进行计算。如果Salt的长度为64比特，那么攻击者的预计算量就增大了2^{64}倍，同时存储量也增大了2^{64}倍，使得字典攻击几乎不可能。
  - 如果攻击者得知Salt值后进行攻击，那就不得不重新计算所有可能的口令，仍然是比较困难的。
- 2、另一个对策是增加单向散列函数处理次数。比如可以对口令用单向散列函数处理多次，这就大大增加了攻击者的预计算时间，但对正常用户没有明显影响。

3.5.2 生日攻击

- 对单向散列函数有两种穷举攻击的方法。
- 第一种是最明显的：给定消息M的散列值H(M)，破译者逐个生成其他消息M′，使H(M)= H(M′)。
- 第二种攻击方法更巧妙：攻击者寻找两个随机的消息：M和M′，并使H(M)=H(M′)(这称为随机碰撞)，它比第一种方法来得容易。

- 生日悖论是一个标准的统计问题。
- 房子里面应有多少人才能使至少一人与你生日相同的概率大于1/2呢？
- 答案是253。
- 有多少人才能使他们中至少两个人的生日相同的概率大于1/2呢？
- 答案出人意料地低：23人。
- 若一个房子里面的人数有100人，有两人相同生日的概率是：
- 0.9999997

- 寻找特定生日的某人类似于第一种方法；而寻找两个随机的具有相同生日的两个人则是第二种攻击。
- 假设一个单向散列函数是安全的，并且攻击它最好的方法是穷举攻击。
- 假定其输出为m比特，那么寻找一个消息，使其散列值与给定散列值相同则需要计算2^m次；而寻找两个消息具有相同的散列值仅需要试验$2^{m/2}$个随机的消息。
- 每秒能运算一百万次单向散列函数的计算机得花600,000年才能找到一个消息与给定的64比特散列值相匹配。
- 同样的机器可以在大约一个小时里找到一对有相同散列值的消息。

- 这就意味着如果你对生日攻击非常担心，那么你所选择的单向散列函数其输出长度应该是你本以为可以的两倍。
- 例如，如果你想让他们成功破译你的系统的可能低于$1/2^{80}$，那么应该使用输出为160比特的单向散列函数。
- 令我们中国人骄傲的是，山东大学数学院的王小云教授已经在一定程度上“破解”了MD5和SHA-1这两大主流算法。这在国际社会尤其是国际密码学领域引起了极大反响，也敲响了电子商务安全的警钟。

- 2004年8月17日，在美国加州圣巴巴拉召开的国际密码学会议上，王小云教授公布了MD5算法的破解成果。
- 其后，密码学家Arjen Lenstra利用王小云教授的研究成果伪造了符合X.509标准的数字证书。
- 这说明了MD5的破译已经不仅仅是理论破译结果，而是可以导致实际的攻击。MD5的撤出迫在眉睫。
- 王小云教授的攻击方法称为模差分。这种攻击是一种特殊的差分攻击，与其它差分攻击不同，它不使用异或作为差分手段，而使用整模减法差分作为手段。
- 这种方法可以有效地查找碰撞，使用这种攻击可在15分钟至1小时内查找到MD5碰撞。将这种攻击用于MD4,则能在不到一秒钟的时间内找到碰撞。这种攻击也可用于其它单向散列函数，如RIPEMD、HAVAL。

- 更让密码学界震惊的是，2005年2月15日，王小云等人的论文证明SHA-1算法在理论上也被破解。
- 需要指出的是，找到单向散列函数的碰撞并不能证明单向散列函数就彻底失效了。因为产生碰撞的消息可能是随机的，没有什么实际意义。
- 最致命的破解是对给定的消息M，较快地找到另一消息M’并满足H(M)=H(M’)，当然M’应该有意义并最好符合攻击者的意图。

小结

- 单向散列函数在许多方面得到了广泛的应用，特别是在完整性验证和密码的散列值存储方面。
- Hash函数的特点？
- 两种目前普遍使用的单向散列函数：MD5和SHA-1。
- MAC—带密钥的哈希函数， 用于消息认证。
- 对Hash函数的攻击：字典攻击、生日攻击。

Thanks !