# 散列算法

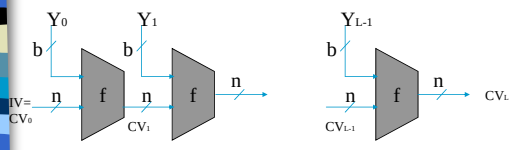
郑燕飞

## 散列算法

- MD5和MD4
- 安全杂凑算法SHA
- RIPEMD-160
- HMAC



## 安全杂凑算法的一般结构




IV = 初始值
CV = 链接值
Yi = 第i 个输入数据块
f = 压缩算法
n = 散列码的长度
b = 输入块的长度

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

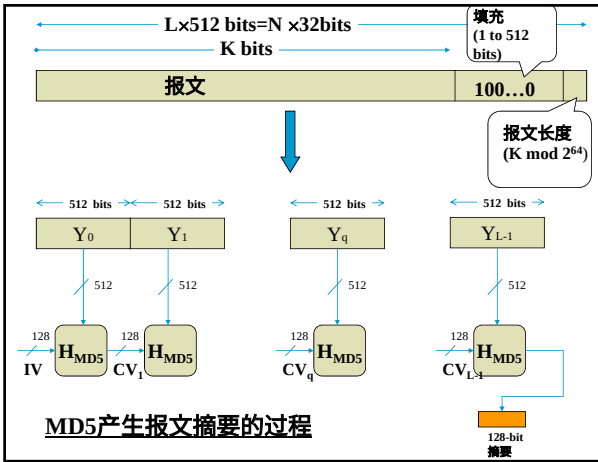


## MD5 算法逻辑

- 输入：任意长度的消息
- 输出：128位消息摘要
- 处理：以512位输入数据块为单位

MD5 (RFC 1321) developed by Ron Rivest ("R" of the RSA )at MIT in 90's.






MD5产生报文摘要的过程

## MD5算法描述

•步骤1：添加填充位(一个1 和若干个0)。在消息的最后添加适当的填充位使得数据位的长度满足length ≡ 448 mod 512。
•步骤2：添加长度。原始消息长度（二进制位的个数），用64位表示。如果长度超过$2^{64}$位，则仅取最低64位，即mod $2^{64}$。

到此为止，我们已经得到一个512位的整倍数长度的新的消息。可以表示为L个512位的数据块：$Y_0,Y_1,\ldots,Y_{L-1}$。其长度为L×512bits。令N=L×16,则长度为N个32位的字。令M[0…N-1]表示以字为单位的消息表示。

## MD5算法描述(Cont.)

- 步骤3：初始化MD缓冲区。一个128位MD缓冲区用以保存中间和最终散列函数的结果。它可以表示为4个32位的寄存器(A,B,C,D)。

  寄存器初始化为以下的16进制值。

  A = 67452301

  B = EFCDAB89

  C = 98BADCFE

  D = 10325476



## MD5算法描述(Cont.)

- 上述值的存储方式为：

```
Word A:  01  23  45  67
Word B:  89  AB  CD  EF
Word C:  FE  DC  BA  98
Word D:  76  54  32  10
```



## MD5算法描述(Cont.)

- 步骤4：处理消息块（512位 = 16个32位字）。一个压缩函数是本算法的核心($H_{MD5}$)。它包括4轮处理。四轮处理具有相似的结构,但每次使用不同的基本逻辑函数，记为F,G,H,I。每一轮以当前的512位数据块($Y_q$)和128位缓冲值ABCD作为输入，并修改缓冲值的内容。每次使用64元素表T[1…64]中的四分之一.



T表，由sin 函数构造而成。T的第i个元素表示为T[i]，其值等于 $2^{32}$×abs(sin(i)),其中i是弧度。由于abs(sin(i))是一个0到1之间的数，T的每一个元素是一个可以表示成32位的整数。T表提供了随机化的32位模板，消除了在输入数据中的任何规律性的特征。

```
T[1]  = D76AA478        T[49] = F4292244
T[2]  = E8C7B756        T[50] = 432AFF97
T[3]  = 242070DB        T[51] = AB9423A7
T[4]  = C1BDCEEE        T[52] = FC93A039
. . .                   . . .
T[16] = 49b40821        T[64] = EB86D391
```

## MD5算法描述(Cont.)

•步骤5：输出结果。所有L个512位数据块处理完毕后，最后的结果就是128位消息摘要。

$CV_0 = IV$

$CV_{q+1} = SUM_{32}(CV_q, RF_I[Y_q, RF_H[Y_q, RF_G[Y_q, RF_F[Y_q, CV_q]]]])$

$MD = CV_L$

其中：IV = ABCD的初始值（见步骤3）

$Y_q$ = 消息的第q个512位数据块

L = 消息中数据块数；

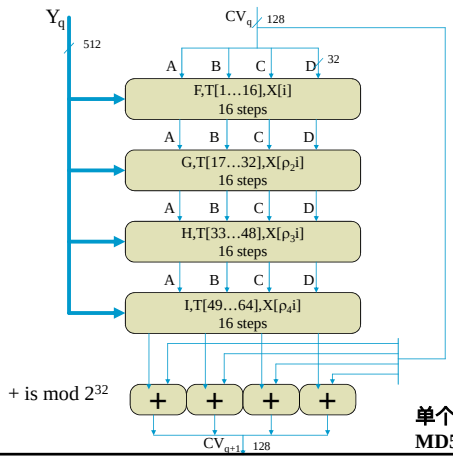
CVq = 链接变量，用于第q个数据块的处理

RFx = 使用基本逻辑函数x的一轮功能函数。

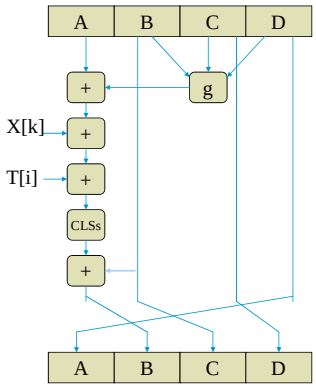
MD = 最终消息摘要结果

$SUM_{32}$=分别按32位字计算的模$2^{32}$加法结果。






单个 512-bit 分组的 MD5 处理过程




**基本MD5操作(单步)**

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

$\rho_2 i = (1+5i) \bmod 16$
$\rho_3 i = (5+3i) \bmod 16$
$\rho_2 i = 7i \bmod 16$

## MD4 (1990年10月作为RFC1320发表) by Ron Rivest at MIT

- **MD4的设计目标**
  - 安全性：寻找两个具有相同报文摘要的报文计算不可行
  - 速度：32位体系结构下计算速度快.
  - 简明与紧凑：易于编程.
  - 有利的小数在前的结构(Intel 80xxx, Pentium )
- **MD4与MD5的区别**
  - MD4用3轮,每轮16 步,MD5用4轮,每轮16步.
  - MD4中第一轮没有常量加；MD5中64步每一步用了一个不同的常量 T[i]；
  - MD5用了四个基本逻辑函数，每轮一个；MD4用了三个
  - MD5每轮加上前一步的结果；MD4没有.



## 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由美国国家标准技术研究所NIST开发，作为联邦信息处理标准
于1993年发表（FIPS PUB 180），1995年修订，作为SHA-1(FIPS PUB 180-1)，SHA-1基于MD4设计。



## SHA-1 算法描述

•步骤1：添加填充位(一个1 和若干个0)。在消息的最后添加适当的填充位使得数据位的长度满足length ≡ 448 mod 512。

•步骤2：添加长度。一个64位块，表示原始消息长度，64位无符号整数。

•步骤3：初始化MD缓冲区。一个160位MD缓冲区用以保存中间和最终散列函数的结果。它可以表示为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
```

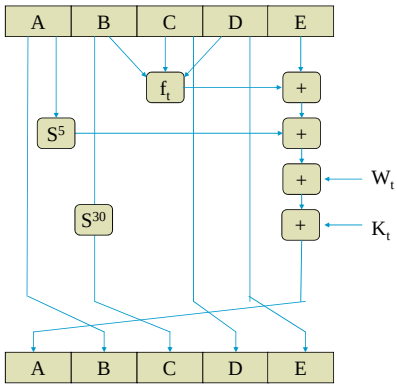
前四个与MD5相同，但存储为大数在前的形式.

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

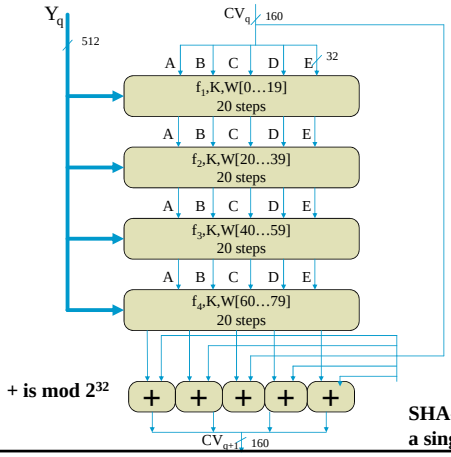
步骤5：输出。全部L个512位数据块处理完毕后，输出160位消息摘要。



$CV_0 = IV$
$CV_{q+1} = SUM_{32}(CV_q, ABCDE_q)$
$MD = CV_L$
其中：IV = ABCDE的初始值；
ABCDEq对第q轮消息数据块处理最后一轮所得的结果；
L = 数据块的个数
$SUM_{32}$ = 对每一个输入对的字求加模$2^{32}$
MD = 最后的消息摘要值。



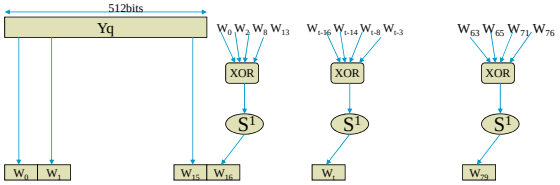

基本SHA操作(单步)




SHA-1 Processing of a single 512-bit block

| Step | Function Name | Function Value |
|---|---|---|
| (0≤ t ≤19) | $f_1 = f(t,B,C,D)$ | $(B \wedge \overline{C}) \vee (B \wedge D)$ |
| (20≤ t ≤39) | $f_2 = f(t,B,C,D)$ | $B \oplus C \oplus D$ |
| (40≤ t ≤59) | $f_3 = f(t,B,C,D)$ | $(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$ |
| (60≤ t ≤79) | $f_4 = f(t,B,C,D)$ | $B \oplus C \oplus D$ |

$Wt = S^1(W_{t-16} \oplus W_{t-14} \oplus W_{t-3})$




# MD4、MD5、SHA比较

| | MD4 | MD5 | SHA |
|---|---|---|---|
| Hash 值 | 128bit | 128bit | 160bit |
| 分组处理长度 | 512bit | 512bit | 512bit |
| 基本字长 | 32bit | 32bit | 32bit |
| 步数 | 48(3*16) | 64(4*16) | 80(4*20) |
| 消息长 | <=$2^{64}$bit | No limit | <=$2^{64}$bit |
| 基本逻辑函数 | 3 | 4 | 3 |
| 常熟个数 | 3 | 64 | 4 |
| 速度 | -- | 1/7 MD4 | 3/4 MD4 |



# HMAC

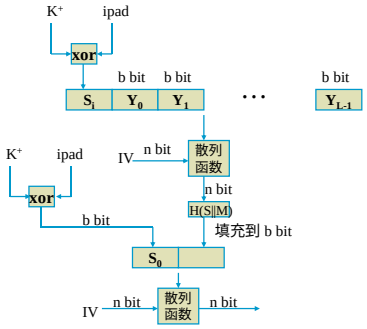
- 设计目标
  - 无需修改的使用现有的散列函数
  - 算法中的散列函数易于替换
  - 保持散列函数的原有性能不会降低
  - 使用和出来密钥的方式简单
  - 易于密码编码分析



# HMAC

- 优点
  - 已有散列函数可作为HMAC中一个模块
  - 替换HMAC中的散列函数非常简单
  - 使用更高安全性的散列函数可以保持HMAC的安全性

# HMAC


K+
ipad
xor
b bit
b bit
b bit
Si
Y0
Y1
YL-1
K+
ipad
IV
n bit
散列函数
n bit
xor
H(S|M)
b bit
填充到 b bit
S0
IV
n bit
散列函数
n bit





- END