下面介绍最近关注中的**HASH**算法

- MD5
- SHA-1
- RIPEMD-160
- HMAC

MD5简介

1992年, MD5 (RFC 1321) developed by Ron Rivest at MIT

- **MD5把数据分成512-bit块**
- **MD5的hash值是128-bit**
- **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算法步骤

- **Step 1: Padding M ➔ M_1**
  - **$|M_1| \equiv 448 \bmod 512$**
  - **$|M_1| > |M|$ ➡**
  - 如果**$|M| \equiv 448 \bmod 512$**,则**$|M_1| = |M|+512$**
  - **Padding**内容**: 100…0**
- **Step 2: Append 64-bit length M_1 ➔ M_2**
  - 若**$|M| > 2^{64}$**,则仅取低**64**位
  - 低字节在前 **(little-endian)**
  - **$|M_2|$**为**512**的倍数**: $Y_0, Y_1, \ldots, Y_{L-1}$**

MD5: compression

- **Step 3: Initialize MD buffer (little-endian)**

  A = 01 23 45 67 (0x67452301)

  B = 89 AB CD EF (0xEFCDAB89)

  C = FE DC BA 98 (0x98BADCFE)

  D = 76 54 32 10 (0x10325476)

- **Step 4: Compression**

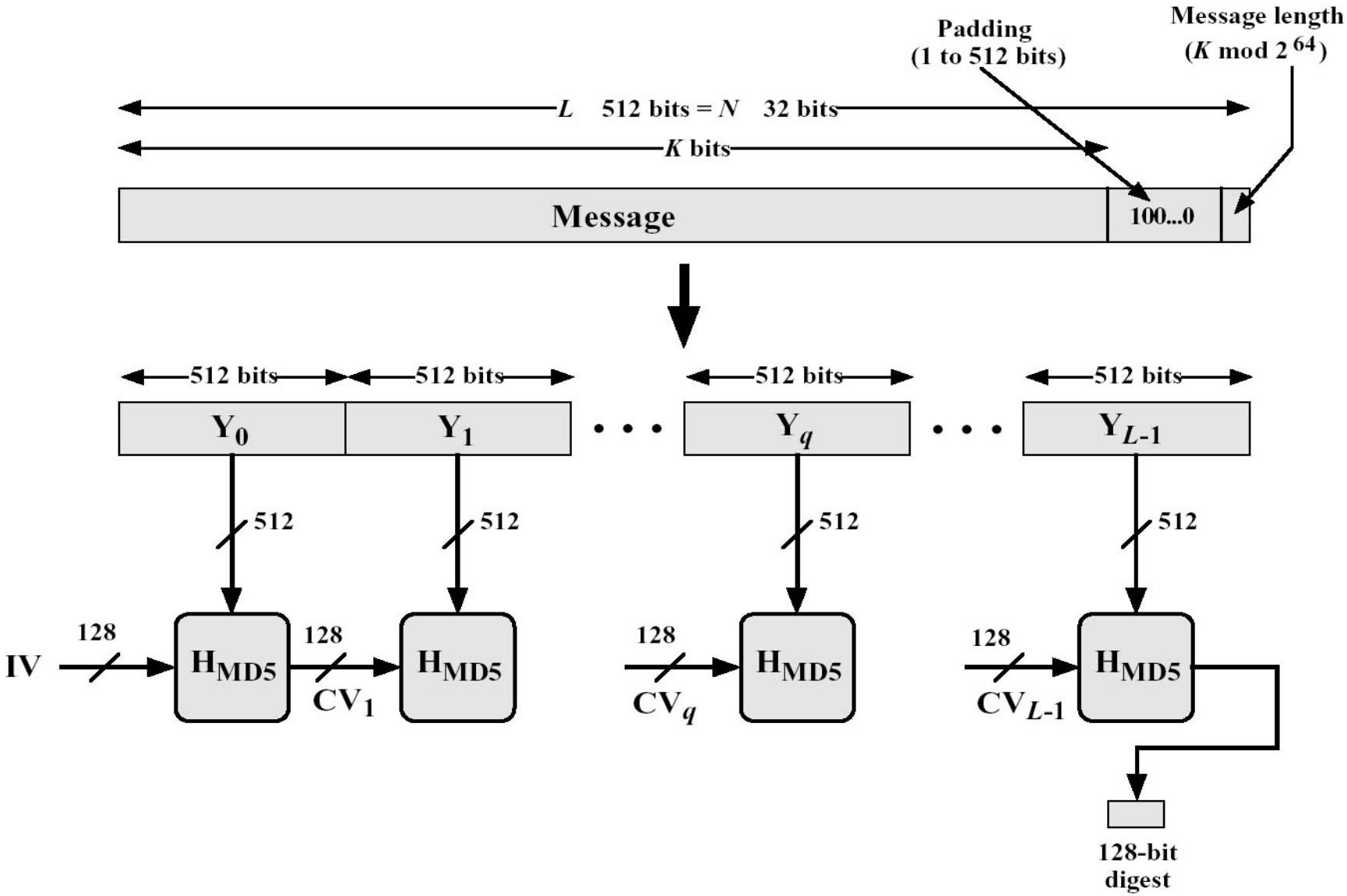
  $CV_0 = IV$

  $CV_i = H_{MD5}(CV_{i-1}, Y_i)$

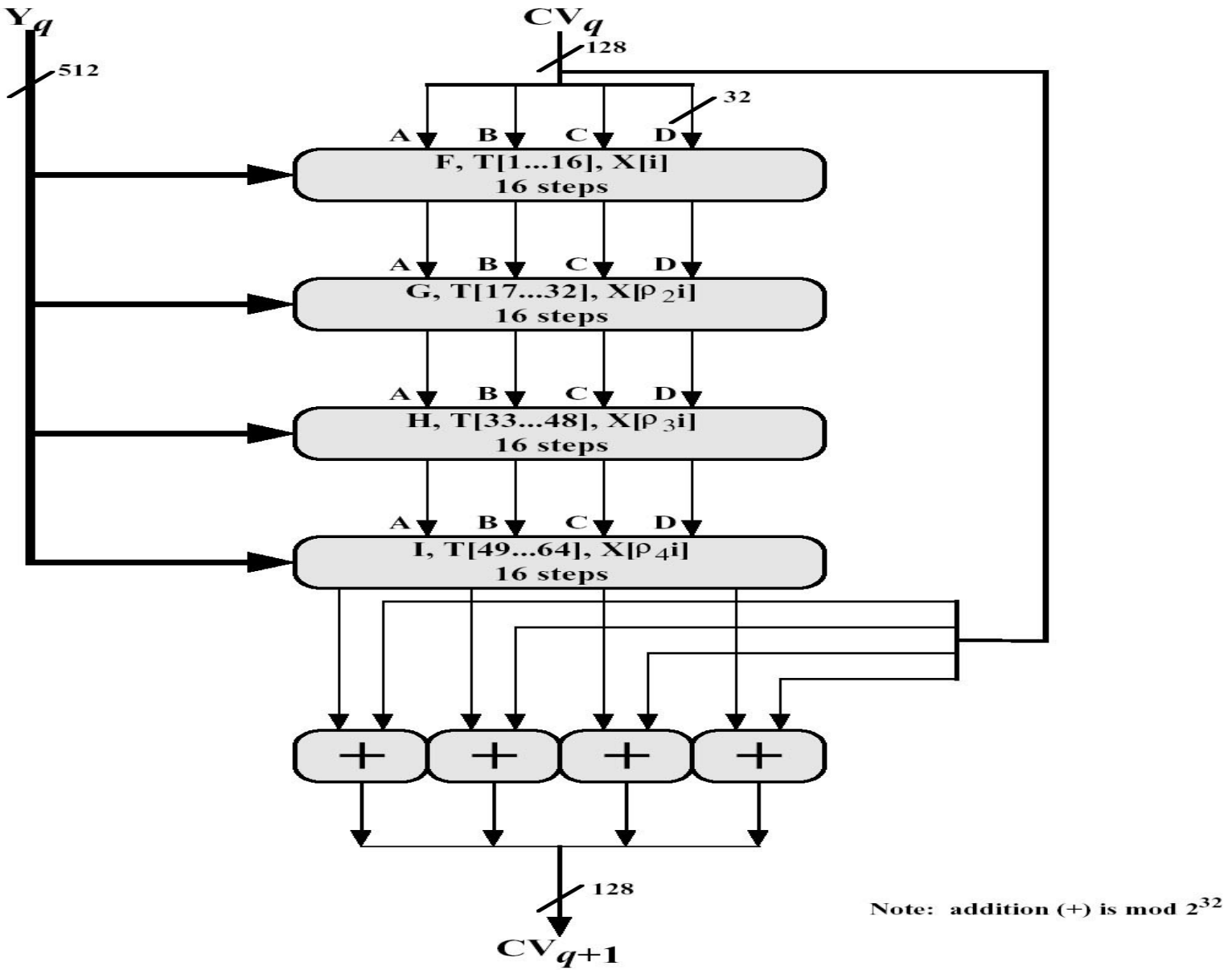
- **Step 5: Output**

  $MD = CV_L$

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
Y0
Y1
Yq
YL-1
512
512
512
512
IV
128
HMD5
128
CV1
128
HMD5
CVq
128
HMD5
CVL-1
HMD5
128-bit
digest

MD5 Step 4: 示意图



MD5 Step 4: overview

- Step 4: CV_0=IV, $CV_i=H_{MD5}(CV_{i-1}, Y_i)$
  - –(A0,B0,C0,D0)←(A,B,C,D)
  - –RoundOne(A,B,C,D,T[1…16],X[0…15])
  - –RoundTwo(A,B,C,D,T[17…32],X[0…15])
  - –RoundThree(A,B,C,D,T[33…48],X[0…15])
  - –RoundFour(A,B,C,D,T[49…64],X[0…15])
  - –(A,B,C,D)←(A+A0,B+B0,C+C0,D+D0)
- 512-bit块(X[…]为32-bit表示)在四个Round使用
- 每个Round包含16次循环,每次处理一个32-bit
- T[j]= [sin(j)*2^{32}]的整数部分, $1 \le j \le 64$

MD5 Compression Function

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；

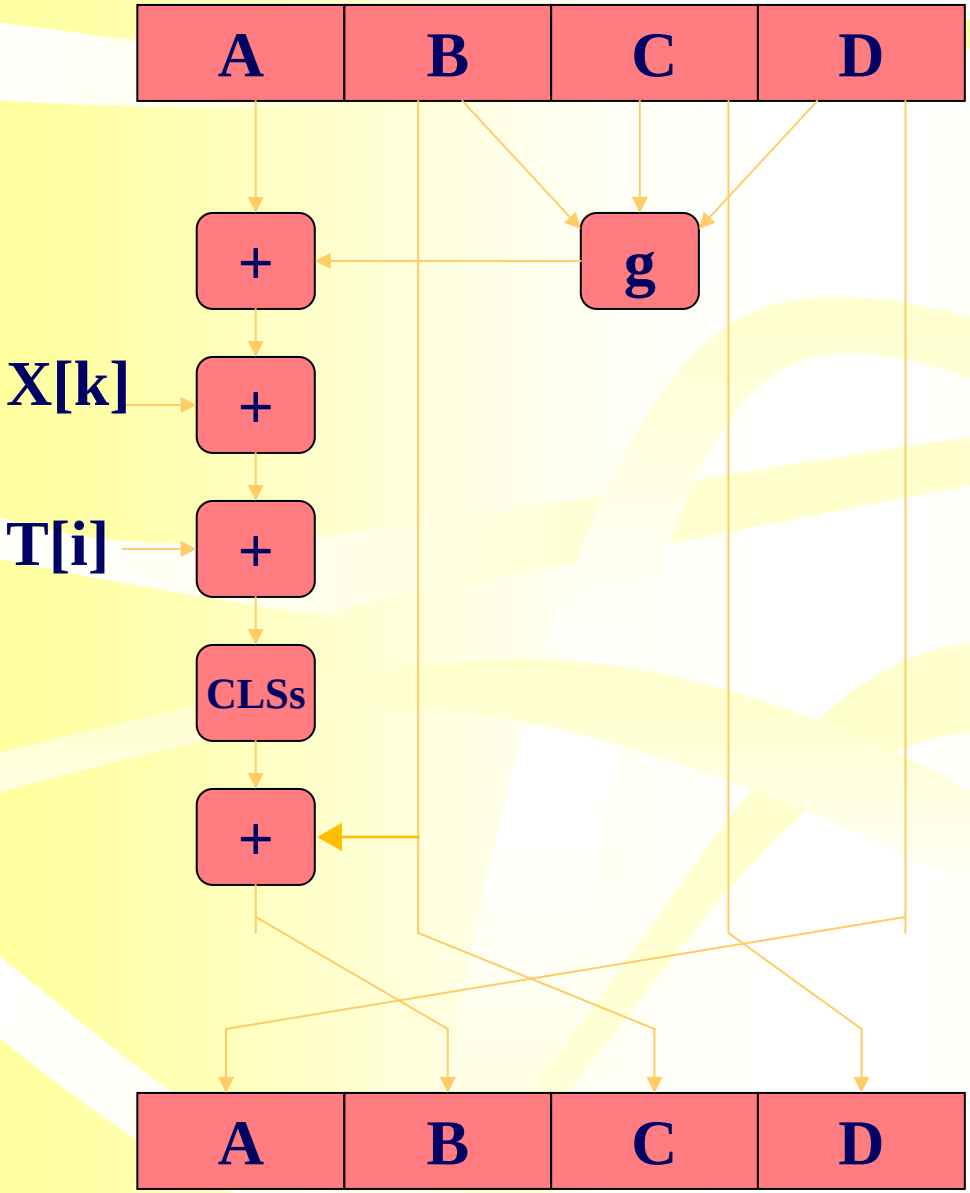
g = 基本逻辑函数**F,G,H,I**之一；

<<<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)	$c \oplus (b \vee \bar{d})$

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

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

$\rho_2 i = 7i \bmod 16$

MD5 Step 4: RoundOne

- For(k = 0; k < 16; ++k) {

$A \leftarrow B + ((A + g_1(B,C,D) + X[\rho_1(k)] + T[16 \times 0 + k + 1]) <<< s_1[k \bmod 4])$

$(A,B,C,D) \leftarrow (A,B,C,D) >>> 32$

}

- $g_1(B,C,D) = (B \,\&\, C) \mid (B \,\&\, D)$
- $\rho_1(k) = k, \qquad 0 \le k < 16$
- $s_1[0 \ldots 3] = [7,12,17,22]$

MD5 Step 4: RoundTwo

- For(k = 0; k < 16; ++k) {

$A \leftarrow B + ((A + g_2(B,C,D) + X[\rho_2(k)] + T[16 \times 1 + k + 1]) <<< s_2[k \bmod 4])$

$(A,B,C,D) \leftarrow (A,B,C,D) >>> 32$

}

- $g_2(B,C,D) = (B \,\&\, D) \mid (C \,\&\, D)$
- $\rho_2(k) = (1 + 5k) \bmod 16, \quad 0 \le k < 16$
- $s_2[0 \ldots 3] = [5,9,14,20]$

MD5 Step 4: RoundThree

- For(k = 0; k < 16; ++k) {

$A \leftarrow B + ((A + g_3(B,C,D) + X[\rho_3(k)] + T[16 \times 2 + k + 1]) <<< s_3[k \bmod 4])$

$(A,B,C,D) \leftarrow (A,B,C,D) >>> 32$

}

- $g_3(B,C,D) = B \oplus C \oplus D$
- $\rho_3(k) = (5 + 3k) \bmod 16, \quad 0 \leq k < 16$
- $s_3[0 \ldots 3] = [4,11,16,23]$

MD5 Step 4: RoundFour

- For(k = 0; k < 16; ++k) {

$A \leftarrow B + ((A + g_4(B,C,D) + X[\rho_4(k)] + T[16 \times 3 + k + 1]) <<< s_4[k \bmod 4])$

$(A,B,C,D) \leftarrow (A,B,C,D) >>> 32$

}

- $g_4(B,C,D) = C \oplus (B \mid D)$
- $\rho_4(k) = 7k \bmod 16, \quad 0 \le k < 16$
- $s_4[0\ldots3] = [6,10,15,21]$

$CV0 = IV$

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

$MD = CV_L$

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。

对MD5的攻击

•**Berson**在**1992**年就已经证明,对单轮的**MD5**算法利用差分密码分析,可以在适当的时间内找到碰撞!然而扩展到四轮就困难了. **Berson. T. "Differential Cryptanalysis Mod $2^{32}$with Applications to MD5." EUROCRYPTO'92 .**

•**Dobbertin** 在**1996**年对**MD5**的强无碰撞现象给出有效攻击. **"The Status of MD5 After a Recent Attack." CRYPTO-Bytes ,Summer 1996.**

William Stallings 在他的 "密码学与网络安全"书中已经表明自**1998**年后,人们已经不太提倡使用**MD5!**

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!

Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD

Xiaoyun Wang, Dengguo Feng, Xuejia Lai, Hongbo Yu

Secure Hash Algorithm 简介

- 1992年NIST制定了SHA(128位)
- 1993年SHA成为标准（FIPS PUB 180）
- 1994年修改产生SHA-1(160位)
- 1995年SHA-1成为新的标准,作为SHA-1(FIPS PUB 180-1)
- SHA-1要求输入消息长度<2^{64}
- 输入按512位的分组进行处理的
- SHA-1的摘要长度为160位
- 基础是MD4

SHA-1: padding

- 与MD5相同
- **Step 1: Padding M ➔ M_1**
  - $|M_1| \equiv 448 \bmod 512$
  - $|M_1| > |M|$ ➡
  - 如果$|M| \equiv 448 \bmod 512$,则$|M_1| = |M|+512$
  - **Padding内容: 100…0**
- **Step 2: Append 64-bit length M_1 ➔ M_2**
  - $|M| < 2^{64}$
  - 高字节在前 **(big-endian)**
  - $|M_2|$为512的倍数: $Y_0, Y_1, \ldots, Y_{L-1}$

SHA-1: compress

- **Step 3: Initialize MD buffer (big-endian)**

  A = 67 45 23 01 (0x67452301)
  B = EF CD AB 89 (0xEFCDAB89)
  C = 98 BA DC FE (0x98BADCFE)
  D = 10 32 54 76 (0x10325476)
  E = C3 D2 E1 F0 (0xC3D2E1F0)

- **Step 4: Compression**

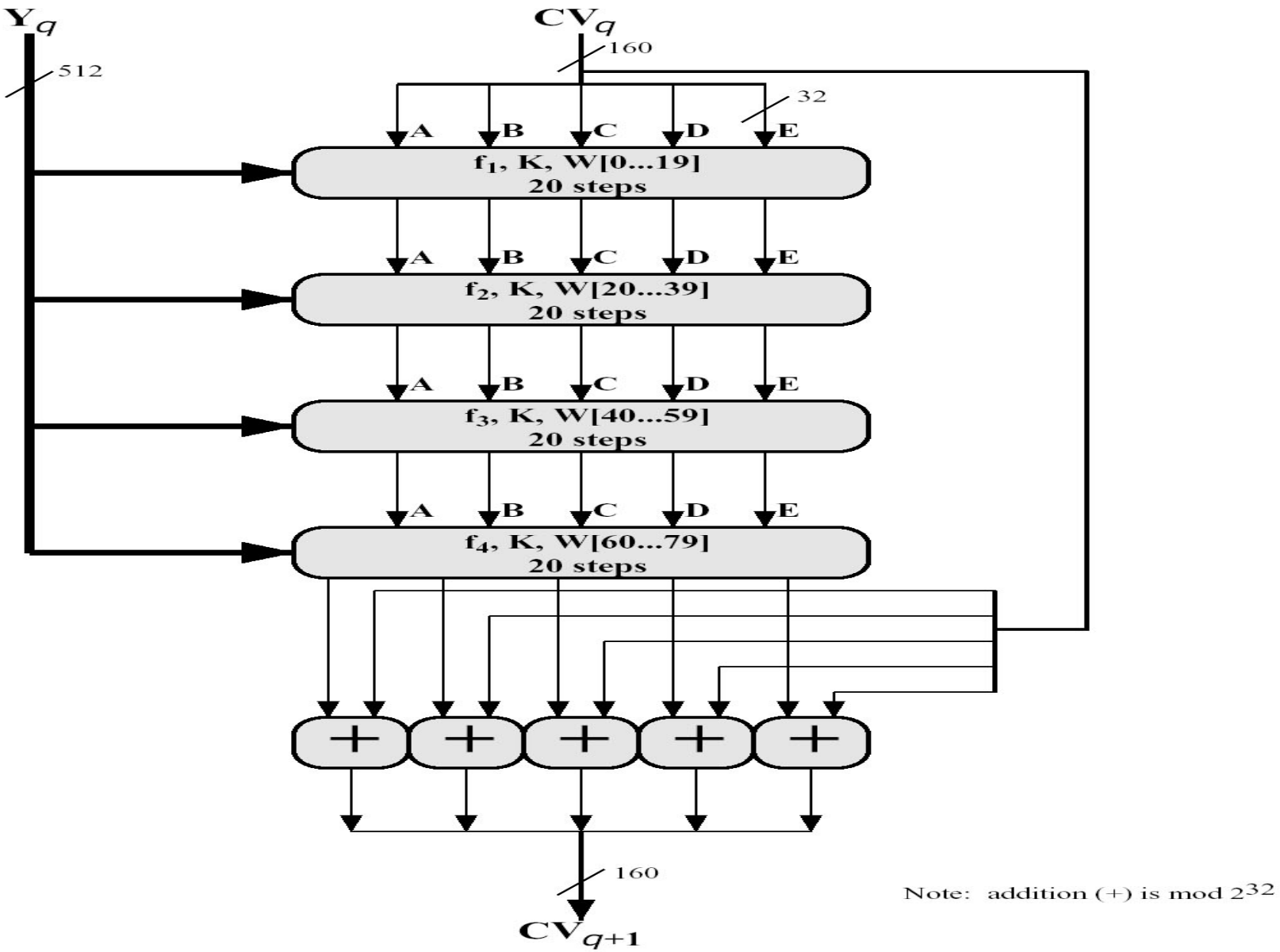
  $CV_0 = IV$

  $CV_i = H_{SHA\text{-}1}(CV_{i-1}, Y_i)$

- **Step 5: Output**

  $MD = CV_L$

SHA-1 step 4: 示意图



SHA-1 step 4: overview

- Step 4: $CV_0=IV$, $CV_i=H_{SHA\text{-}1}(CV_{i-1}, Y_i)$
  (A0,B0,C0,D0,E0)←(A,B,C,D,E), t ← 0
  Round(A,B,C,D,E,K[t],W[t]) $0 \le t < 80$
  (A,B,C,D,E)←(A+A0,B+B0,C+C0,D+D0,E+E0)
- 整个Round包含80次循环,每次处理一个32-bit
  - $W[t] = Y_i[t]$ $0 \le t < 16$
    $W[t] =(W[t-16] \oplus W[t-14] \oplus W[t-8] \oplus W[t-3])<<<1$
    $16 \le t < 80$
  - 每组(16个)W[t]可由前一组W[t]直接计算

SHA-1: compression function

- **Four rounds: $0 \le t < 80$**

  $E \leftarrow E + f(t,B,C,D) + (A <<< 5) + W[t] + K[t]$

  $B \leftarrow B <<< 30$

  $(A,B,C,D,E) \leftarrow (A,B,C,D,E) >>> 32$

  - $f(t,B,C,D) = (B \& C) | (B \& D)$ $\quad 0 \le t < 20$

    $K[t] = 2^{30} \times \text{sqrt}(2)$
  - $f(t,B,C,D) = B \oplus C \oplus D$ $\quad 20 \le t < 40$

    $K[t] = 2^{30} \times \text{sqrt}(3)$
  - $f(t,B,C,D) = (B \& C) | (B \& D) | (C \& D)$ $\quad 30 \le t < 60$

    $K[t] = 2^{30} \times \text{sqrt}(5)$
  - $f(t,B,C,D) = B \oplus C \oplus D$ $\quad 60 \le t < 80$

    $K[t] = 2^{30} \times \text{sqrt}(10)$

SHA-1 压缩函数

$$A,B,C,D,E \leftarrow (E + f(t,B,C,D)+S^5(A) +Wt + Kt),A,S^{30}(B),C,D$$

其中，

A,B,C,D,E = 缓冲区的5个字；

t = 步数，**0<= t <= 79**；

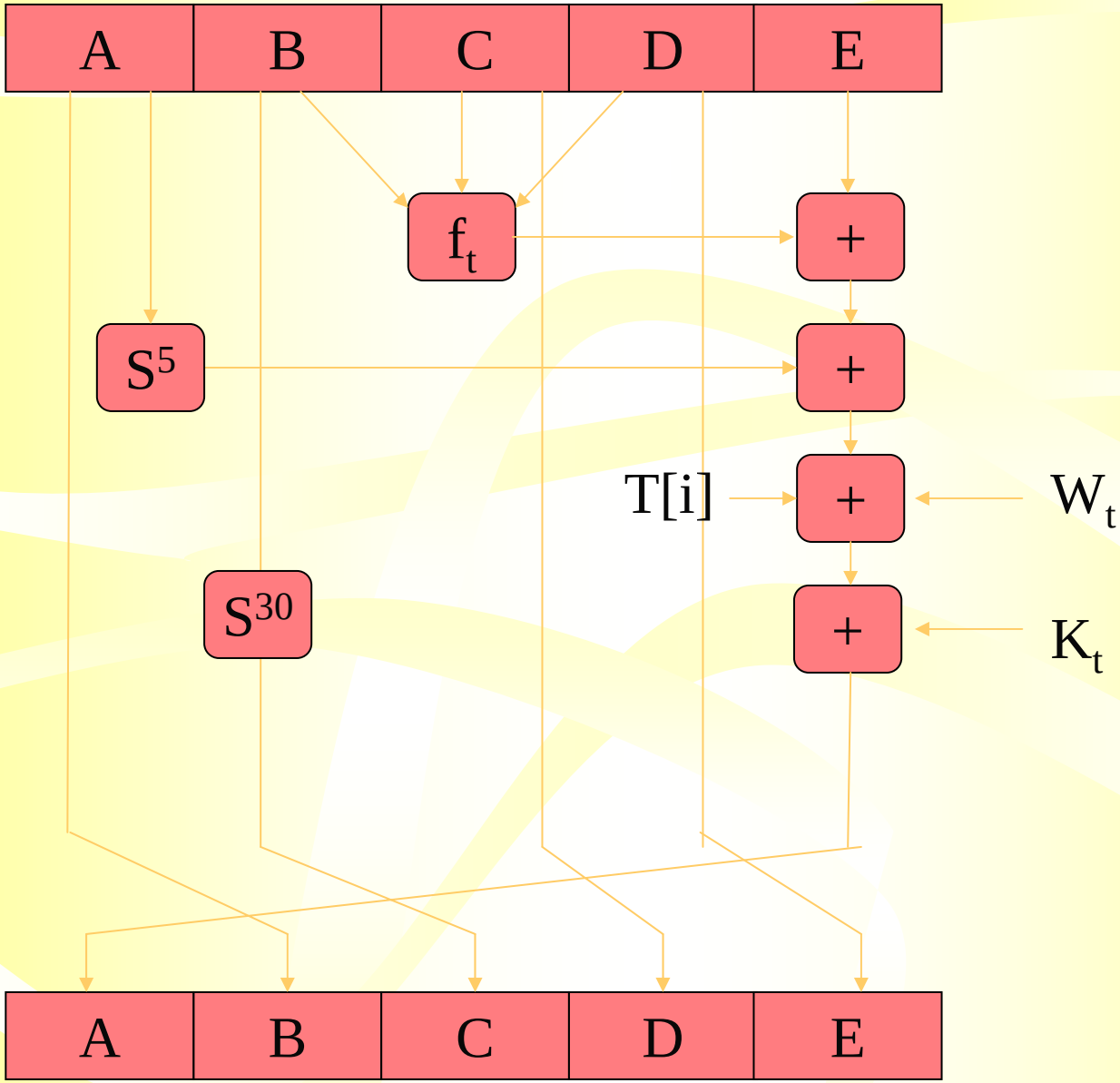
f(t,B,C,D) = 步t的基本逻辑函数；

S^k = 循环左移k位给定的32位字；

Wt = 一个从当前512数据块导出的32位字；

Kt = 一个用于加法的常量，四个不同的值，如前所述；

+ = 加模2^{32}。


A
B
C
D
E
f_t
+
S^5
+
T[i]
+
W_t
S^{30}
+
K_t
A
B
C
D
E

SHA-1总结

- SHA-1使用big-endian
- 抗穷举攻击：
  - 摘要->2^{160}（SHA-1），2^{128}（MD5）
  - 产生两个相同摘要的消息：2^{80}（SHA-1），2^{64}（MD5）
- 抵抗生日攻击: 160位hash值
- 没有发现两个不同的512-bit块,它们在SHA-1计算下产生相同的“hash”
- 速度：慢于MD5
- 安全性优于MD5

安全散列函数的修改

2001年,NIST修改了FIPS 180-1,给出修订版FIPS 180-2,它们是安全的:

	SHA-1	SHA-256	SHA-384	SHA-512
• 消息摘要	160	256	384	512
• 分组大小	512	512	1024	1024
• 字长	32	32	64	64
• 步数	80	80	80	80
• 安全性	80	128	192	256

RIPEMD-160简介

- 欧洲RIPE项目的结果
- RIPEMD为128位
- 更新后成为RIPEMD-160
- 基础是MD5

RIPEMD-160: padding

- **Step 1: Padding M ➔ M_1**
  - **$|M_1| \equiv 448 \bmod 512$**
  - **$|M_1| > |M|$ ➡**
  - **如果$|M| \equiv 448 \bmod 512$,则$|M_1| = |M|+512$**
  - **Padding内容: 100…0**
- **Step 2: Append 64-bit length M_1 ➔ M_2**
  - **$|M| < 2^{64}$**
  - **低字节在前 (little-endian)**
  - **$|M_2|$为512的倍数: $Y_0, Y_1, \ldots, Y_{L-1}$**

RIPEMD-160: compression

- **Step 3: Initialize MD buffer (little-endian)**

  A = 01 23 45 67 (0x67452301)

  B = 89 AB CD EF (0xEFCDAB89)

  C = FE DC BA 98 (0x98BADCFE)

  D = 76 54 32 10 (0x10325476)

  E = F0 E1 D2 C3 (0xC3D2E1F0)

- **Step 4: Compression**

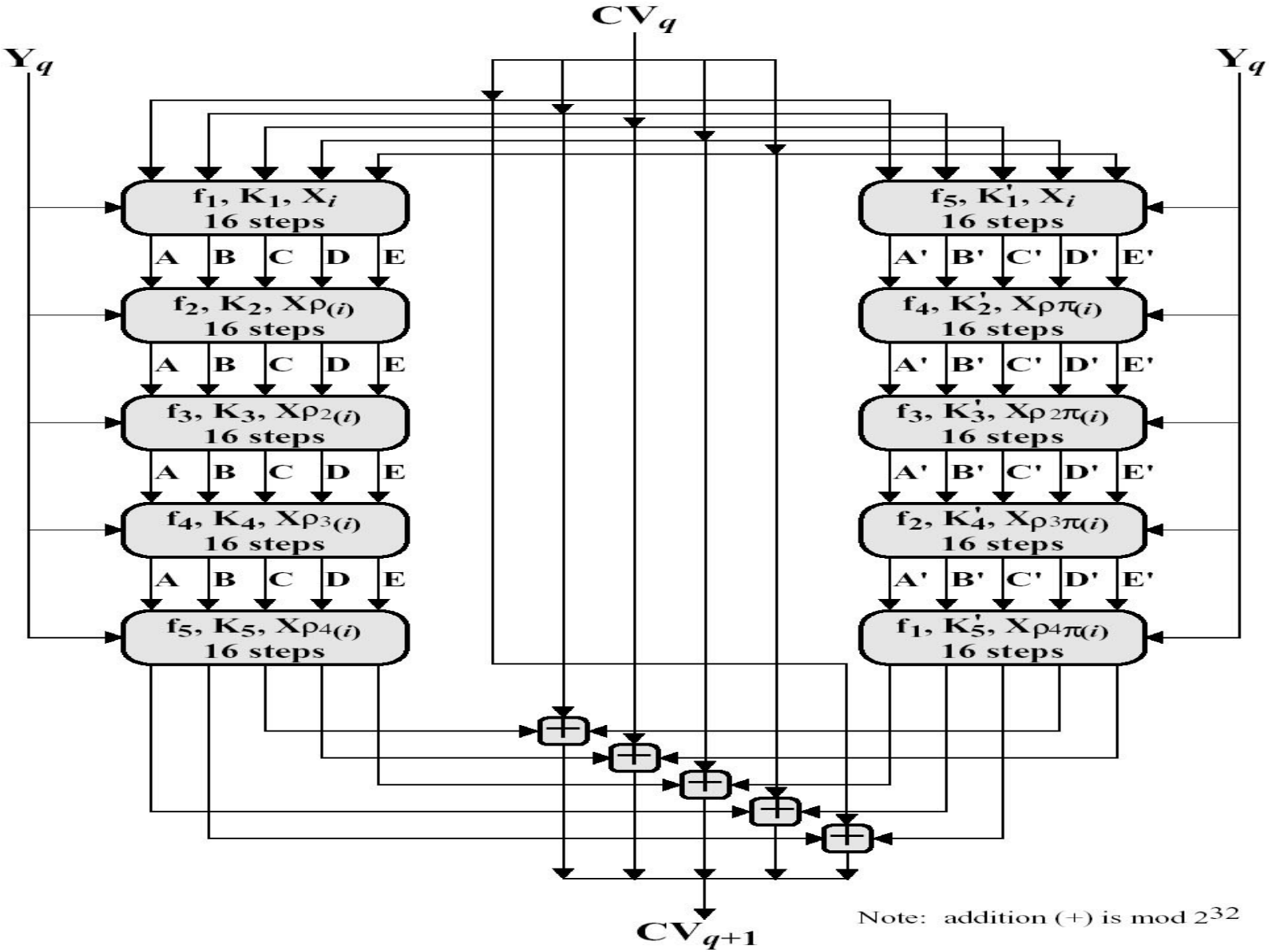
  $CV_0 = IV$

  $CV_i = H_{RIPE}(CV_{i-1}, Y_i)$

- **Step 5: Output**

  $MD = CV_L$

RIPEMD-160 step 4: 示意图



RIPEMD-160: compression function

- **(A0,B0,C0,D0,E0)←(A,B,C,D,E)**
- **Five rounds: $0 \le t < 16$**

  $A \leftarrow ((A+f(B,C,D)+X[p[t]]+K)<<<s)+E$

  $C \leftarrow C<<<10$

  $(A,B,C,D,E) \leftarrow (A,B,C,D,E)>>>32$

  $A' \leftarrow ((A'+f'(B',C',D')+X[p'[t]]+K')<<<s')+E'$

  $C' \leftarrow C'<<<10$

  $(A',B',C',D',E') \leftarrow (A',B',C',D',E')>>>32$

- **$(A,B,C,D,E) \leftarrow (B0+C+D', C0+D+E', D0+E+A', E0+A+B', A0+B+C')$**

RIPEMD-160 step 4: f_i, ρ, π

- **Function f_1,f_2,f_3,f_4,f_5:**
  - **$f_1(B,C,D) = B \oplus C \oplus D$**
  - **$f_2(B,C,D) = (B\&C)|(!B\&D)$**
  - **$f_3(B,C,D) = (B|!C) \oplus D$**
  - **$f_4(B,C,D) = (B\&C)|(C\&!D)$**
  - **$f_5(B,C,D) = B \oplus (C|!D)$**
  - **ρ: 7,4,13,1,10,6,15,3,12,0,9,5,2,14,11,8**
  - **π: 5,14,7,0,9,2,11,4,13,6,15,8,1,10,3,12**

RIPEMD-160总结

- **RIPEMD-160使用little-endian**
- **抵抗生日攻击: 160位hash值**
- **没有发现两个不同的512-bit块,它们在RIPEMD-160计算下产生相同的“hash”**
- **速度略慢于SHA-1**
- **安全性优于MD5**
- **对密码分析的抵抗力好于SHA-1**

比较：

	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

性能	204.6 Mbps	72.6 Mbps	50.5 Mbps

Pentium 4 2.1GHz Windows XP VC++ .NET

hash函数小结

- **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

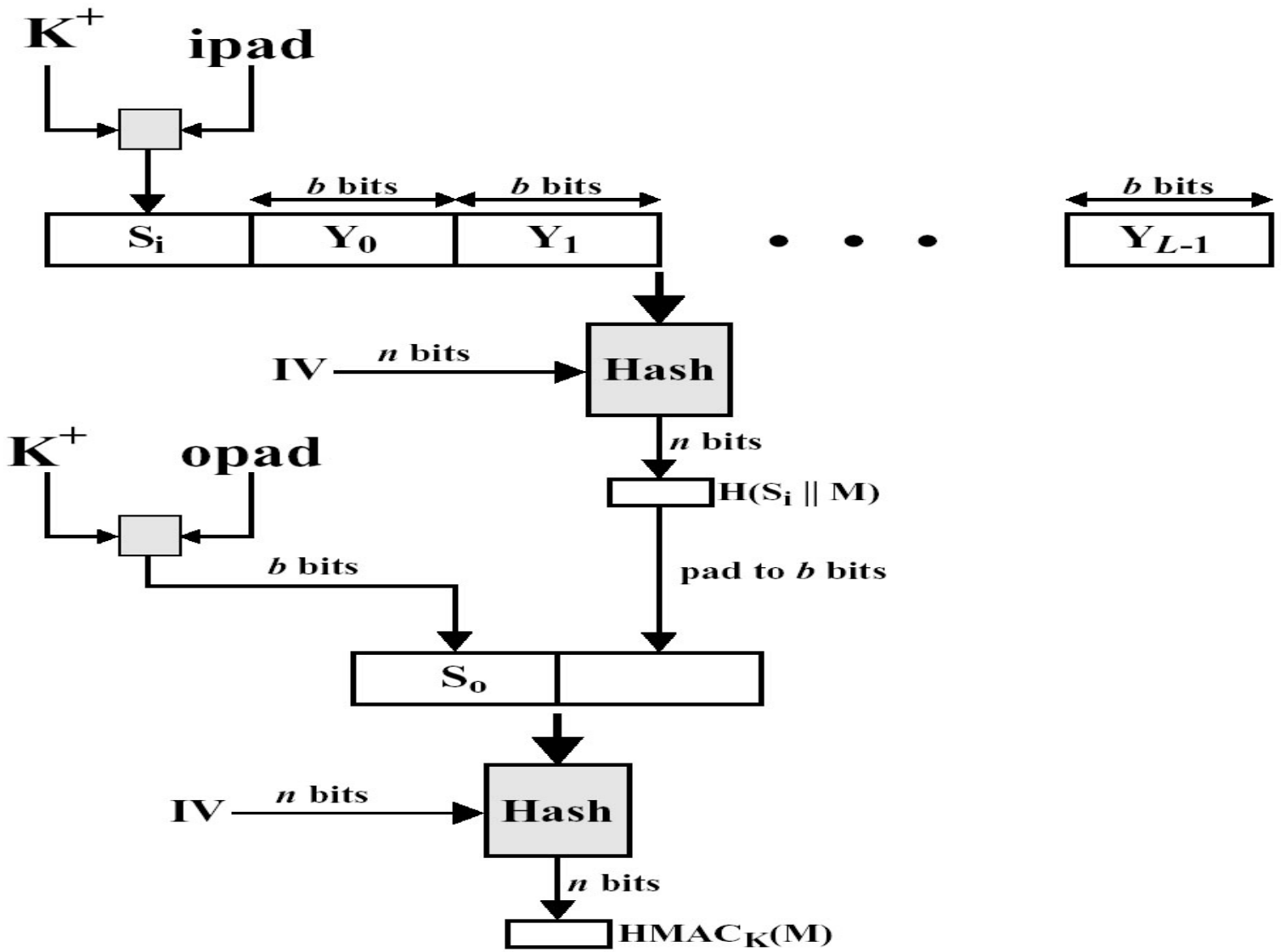
HMAC简介

- **MAC可用分组加密算法产生**
- **ANSI标准(X9.17): $M=(X_1,X_2,\ldots,X_t)$**
- **$\Delta M_1=E_K(X_1)$, $\Delta M_{j+1}=E_K(X_{j+1}\oplus\Delta M_j)$, $1\le j<t$**
  - 速度慢
  - 加密算法出口受限制
- **hash函数可用来构造MAC: HMAC为其中之一**
- **HMAC作为RFC2104并在SSL中使用**

HMAC设计目标

- 无需修改地使用现有的散列函数
- 当出现新的散列函数时,要能轻易地替换
- 保持散列函数的原有性能不会导致算法性能的降低
- 使用和处理密钥的方式简单
- 对鉴别机制的安全强度容易分析,与hash函数有同等的安全性

HMAC示意图


K+
ipad
b bits
b bits
b bits
Si
Y0
Y1
YL-1
IV
n bits
Hash
K+
opad
n bits
H(Si || M)
b bits
pad to b bits
So
IV
n bits
Hash
n bits
HMACK(M)

HMAC的定义与特征

❶对密钥**K**左边补**0**以产生一个**hash**用块K^+

❷K^+每个字节与**ipad(00110110)**作**XOR**以产生S_i

❸对$(S_i||M)$进行**hash**

❹K^+每个字节与**opad(01011010)**作**XOR**以产生S_o

❺$HMAC=f[IV,S_o||f(IV,S_i||M)]$