第12章 散列算法

1　离散对数的散列函数算法

(Chaum－Van, Heijst－Pfitzmann散列算法)

设P是一个大素数，且q=(p-1)/2也为素数，取定有限域F_P中的一个本原元α，给定一个保密的指数λ，要求（λ，p－1）=1，于是β=α^{λ}也为F_P中的本原元。值$\lambda=\log_{\alpha}\beta$不公开，计算这个对数值是计算上难处理的。

散列算法

$h:\{0,1,\cdots,q-1\}\times\{0,1,\cdots,q-1\}\rightarrow F_p^*$

定义为 $h(x_1,x_2)=\alpha^{x1}\beta^{x2}(\mathrm{mod}\ p)$

下面要证明，散列算法h是强无碰撞的！相当于证明：

定理：若上述算法h的碰撞算法是可行的，那么计算离散对数$\log_\alpha\beta$也是可行的。

证明：

假设我们给了一个碰撞$h(x_1,x_2)=h(x_3,x_4)$其中$(x_1,x_2)\neq(x_3,x_4)$，则有下列同余式

$$\alpha^{x1}\beta^{x2}\equiv\alpha^{x3}\beta^{x4}(\mathrm{mod}p)$$

也即，$\alpha^{x1-x3} \equiv \beta^{x4-x2} \pmod{p}$ 记 $\gcd(x_4-x_2, p-1)=d$，

易见 $d \in \{1, 2, q, p-1\}$ 原因是 $p-1=2q$.

1° 若 $d=1$，此时可设 $y=(x_4-x_2)^{-1} \pmod{p-1}$，

有 $\beta \equiv \beta^{(x4-x2)y} \pmod{p} \equiv \alpha^{(x1-x3)y} \pmod{p}$

于是，计算出离散对数

$$\log_\alpha \beta = (x_1-x_3)\, y$$

$$= (x_1-x_3)\,(x_4-x_2)^{-1} \pmod{p-1}$$

2° 若 $d=2$：由 $p-1=2q$，q 为奇素数，

有 $\gcd(x_4-x_2, q)=1$,

可以设 $y=(x_4-x_2)^{-1} \pmod{q}$

于是 $(x_4-x_2)\, y \equiv 1 \pmod{q}$

也就是存在整数k使得（x_4-x_2）$y=kq+1$

所以 $\beta^{(x4-x2)y} \equiv \beta^{kq+1} \pmod p$

$\equiv (-1)^k \beta \pmod p$　　(因为 $\beta^{(p-1)/2} \equiv -1 \pmod p$)

$\equiv \pm \beta \pmod p$

这样　　$\alpha^{(x1-x3)y} \equiv \beta^{(x4-x2)y} \pmod p \equiv \pm \beta \pmod p$

(i) 若 $\alpha^{(x1-x3)y} \equiv \beta \pmod p$

则 $\log_\alpha \beta = (x_1-x_3)y \pmod{p-1}$

(ii)若 $\alpha^{(x1-x3)y} \equiv -\beta \pmod p \equiv \alpha^{(p-1)/2} \beta \pmod p$

$\Longrightarrow \alpha^{(x1-x3)y} \alpha^{(p-1)/2} \equiv \beta \pmod p$

所以，$\log_\alpha \beta = (x_1-x_3)y+(p-1)/2 \pmod{p-1}$

可见，（i）、（ii）都是易计算的。

3°若d=q:因为$0 \leqslant x_2 \leqslant q-1, 0 \leqslant x_4 \leqslant q-1$
有结果，$\gcd(x_4-x_2, p-1)=q$是不可能的。

4°若d=p-1，此时仅当$x_4=x_2$时发生，此时有

$$\alpha^{x1}\beta^{x2} \equiv \alpha^{x3}\beta^{x2} \pmod p$$

$$\alpha^{x1} \equiv \alpha^{x3} \pmod p \Rightarrow x_1=x_3$$

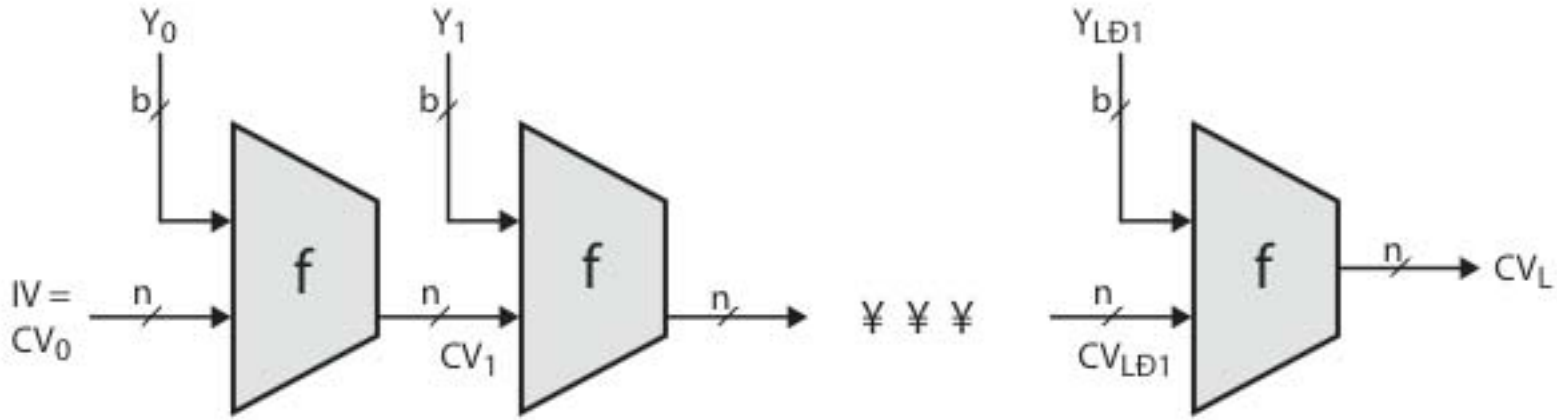
于是，$(x_1,x_2)=(x_3,x_4)$与假设矛盾，此种情况不可能发生。

综上知，如果计算F_P中离散对数$\log_\alpha \beta$是不可行的，那么我们给出的算法h一定是强无碰撞的。

扩展的散列函数想法来自：
Ralph Merkle

我们已研究过具有有限定义域的散列函数。现在研究如何把具有有限定义域的强无碰撞的散列函数扩展为具有无限定义域的强无碰撞散列函数，这将使我们能签名任意长度的消息。

Hash Algorithm Structure



IV = Initial value
CV_i = chaining variable
Y_i = ith input block
f = compression algorithm

L = number of input blocks
n = length of hash code
b = length of input block

2 （扩展的）散列函数

假设h: $(F_2)^m \to (F_2)^t$是一个强无碰撞的散列函数，这里m≥t+1。我们的目的是想从h入手，构造无限定义域的散列函数 $h^*: X \to (F_2)^t$,

其中 $X = \bigcup_{i=m}^{\infty} (F_2)^i$

首先考虑m≥t+2的情况：

符号：|x|表示x的长度。（比特位数目）

x||y表示比特串x和y的并。

(下面给出扩展算法)

假设|x|=n＞m，（x为一段信息bit串）

则先将x表为$x=x_1||x_2||\ldots\ldots||x_k$

这里，

$|x_1|=|x_2|=\ldots=|x_{k-1}|=m-t-1$

且

$|x_k|=m-t-1-d$，易见$0\leqslant d\leqslant m-t-2$

而 $k=[n/(m-t-1)]$ （上整数部分）

下面给出扩展的散列函数h^*的生成算法(Merkle)：

从h到h^*算法，注意$|x|=n>m$，$m\geqslant t+2$情形：

$x=x_1||x_2||\cdots\cdots||x_k$；$|x_1|=|x_2|=\ldots=|x_{k-1}|=m-t-1$;

$|x_k|=m-t-1-d$

1° 对$i=1$到$k-1$做$y_i=x_i$

2° $y_k=x_k||o^d$

3° 设y_{k+1}是d的二进制表示，(右边补0,使$|y_{k+1}|=m-t-1$)

4° $g_1=h(o^{t+1}||y_1)$

5° 对$i=1$到k做$g_{i+1}=h(g_i||1||y_{i+1})$

6° $h^*(x)=g_{k+1}$

注：可以记$y(x)=y_1||y_2||\cdots\cdots||y_k||y_{k+1}$，

其中y_k是由x_k后面（右边）补上d个零来形成的，于是,所有块y_i（$1 \leqslant i \leqslant k$）的长度都为m-t-1；在第3°步中也将在$y_{k+1}$的左边补上零，使得$|y_{k+1}|=m-t-1$。

为了散列x，首先构造y（x），然后以特定方式处理块（分组）y_1，y_2，…，y_{k+1}。对于任意数对x,x′,只要$x \neq x'$，易见 y（x）≠y（x′）。

下面证明，在h是安全的前提下，h^*也是安全的！

定理，假设$h: (F_2)^m \rightarrow (F_2)^t$ 是一个强无碰撞的散列函数，这里$m \geqslant t+2$。那么按上述方式所构造的函数

$$h^*: \bigcup_{i=m}^{\infty} (F_2)^i \rightarrow (F_2)^t$$

也是一个强无碰撞的散列函数。

证明：（假设我们能找到$x \neq x'$满足$h^*(x)=h^*(x')$，那么如果能证明在多项式时间内可找到h的一个碰撞，这就与h假设为强无碰撞的事实矛盾，这样就证明了h^*为强无碰撞的散列函数。）

若有方法找到$x \neq x'$,使$h^*(x)=h^*(x')$：记

$y(x)=y_1\|y_2\|\cdots\cdots\|y_k\|y_{k+1}$和$y(x')=y_1'\|y_2'\|\cdots\cdots\|y_l'\|y_{l+1}'$.

考虑上述算法中，第2°步x和x′分别补上d个和d′个零；记第4°步，第5°步计算出的值分别为$g_1,\cdots, g_{k+1}$和$g'_1,\cdots,g'_{l+1}$。

对于 $|x|?\equiv|x'|\ (\mathrm{mod}\ m-t-1)$,分两种情况：

情况1：若 $|x| \neq |x'| \pmod{m-t-1}$,则有

$d \neq d'$ 及 $y_{k+1} \neq y'_{l+1}$，

我们有 $h(g_k||1||y_{k+1}) = g_{k+1} = h^*(x) =$

$= h^*(x') = g'_{l+1} = h(g'_l||1||y'_{l+1})$,

因为 $y_{k+1} \neq y'_{l+1}$，所以它是h的一个碰撞。

情况2：$|x| \equiv |x'| \pmod{m-t-1}$

(1)若 $|x| = |x'|$,此时 $k=l$ 且 $y_{k+1} = y'_{k+1}$，

$h(g_k||1||y_{k+1}) = g_{k+1} = h^*(x) = h^*(x')$

$= g'_{k+1} = h(g'_k||1||y'_{k+1})$,

若$g_k \neq g'_k$，易见找到了碰撞；若否，即$g_k=g'_k$，则有
$h(g_{k-1}\|1\|y_k)=g_k=g'_k=h(g'_{k-1}\|1\|y'_k)$,
若仍不是碰撞，说明$g_{k-1}=g'_{k-1}, y_k=y'_k$，继续回推下去;如果一直无找到碰撞，说明对$1\leqslant i\leqslant k-1$有$y_i=y'_i$，所以$y(x)=y(x')$，意味着$x=x'$，矛盾！

(2）$|x|\neq|x'|$, 不妨设$|x'|>|x|$，所以$l>k$.用类似（1）的方式进行处理。假设对函数h没有计算出一个碰撞，最终达到如下结果:

$h(O^{t+1}\|y_1)=g_1=g'_{l-k+1}=h(g'_{l-k}\|1\|y'_{l-k+1})$

但是$O^{t+1}\|y_1$的第t+1个比特为0，而$g'_{l-k}\|1\|y'_{l-k+1}$的第t+1个比特为1，所以我们找到了一个h的碰撞.

证毕

上面构造h^*的方法仅当$m \geq t+2$时才使用。对于$m=t+1$时，h^*的构造按下述方法：

仍设$|x|=n>m$，且定义一个函数f：

$f(0)=0$

$f(1)=01$

1) 设$y= y(x)= y_1y_2\ldots y_k=11||f(x_1)||f(x_2)||\ldots||f(x_n)$

2) $g_1=h(O^t||y_1)$

3) 对i=1到k-1做$g_{i+1}=h(g_i||y_{i+1})$

4) $h^*(x)=g_k$

注：定义在上面第1)步中的编码x→y=y(x)有特性：

I) 若x≠x′，则y(x) ≠y(x′)（即为单射）

II) 若x≠x′，则不存在z使y(x)=z||y(x ′)。

（换言之，没有一个编码是另一个编码的后缀，这是因为每个串y(x)都以11开始，且在该串的剩余部分不会存在两个连续的1。）

根据如上的论述，我们有如下的定理——

定理：假设h: $(F_2)^m \to (F_2)^t$ 是一个强无碰撞的散列函数，那么上述构造的函数h*: $\bigcup_{i=t+1}^{\infty} (F_2)^i \to (F_2)^t$ 是一个强无碰撞的散列函数。

3 MD4与MD5散列算法

Rivest(原RSA公司首席科学家，MIT教授)1990年提出MD4，1991年提出增强版MD5;1992年与NSA合作给出SHA-1，公布于1992年1月31日的联帮记录上，并于1993年5月11日采纳作为标准。这些散列函数运算起来非常快，都源于同一思想。它们对于签名非常长的消息是适用的。

为了讲清Rivest的想法，还是从MD4讲起。

MD4：给定一个消息比特串x，使用如下算法来构造M：

1．设$d=447-(|x| \pmod{512})$

2. l表示$|x| \pmod{2^{64}}$的二进制表示,$|l|=64$

3. $M=x||1||O^d||l$

在M的构造中。在x后面附上一个1，然后并上足够多的0使得长度变成模512（$=2^9$）与448用余的数，最后并上64bit的*l*，它是x的长度的二进制表示。（注意：若必要的话，用模2^{64}约简）。易见，512整除 |M| 。

将M表示成阵列形式：

$$M=M[0]M[1]\ldots M[N-1]$$

其中每一个M[i]是一个长度为32bit的串, 并且

$$N\equiv 0\ (\mathrm{mod}\ 16)$$

,我们称每一个M[i]为一个字。

下面从着手，构造关于x的128bit的消息摘要。算法描述如下：

1°给出寄存器A，B，C，D，第一步初始化：

A=67452301　　（16进制）

B=efcdab89　　（16进制）

C=98badcfe　　（16进制）

D=10325476　　（16进制）

2°对i=0到N/16-1，对j=0到15做X[j]=M[16i+j]每次处理阵列M的16个字！

3°AA=A，BB=B，CC=C，DD=D

4°轮1；　　5°轮2；　　6°轮3；

7°A=A+AA，B=B+BB，C=C+CC，D=D+DD

消息摘要为A||B||C||D……128bits!!

解释与说明：

（1）四个寄存器在第一步初始化。从此，每次处理陈列M的16个字。在第2°步的每完成一个循环选代，接着取M的“下一个”16个字，然后把它存贮在陈列X中（第3°步）；接着将4个寄存器的值存贮起来（第4°步）,然后完成三轮散列。每一轮由作用于X中的16个字中的每一个相关运算组成；在三轮中的运算结果产生的新的4个寄存器的值。最后，第8°步中4个寄存器的值通过加上第4°步中存贮的值来更新。这个加法定义为正整数加法，用模2^{32}来约简。

（2）在MD4中三轮是不同的，它们使用的运算描述如下：（注意，X和Y表示输入字，每一个运算产生一个字作为输出。）

X^Y ：X和Y的逐比特“与”；

XVY：X和Y的逐比特“或”；

X⊕Y：X和Y的逐比特“异或”；

$\overline{X}$ ：X的逐比特取补；

X+Y：模2^{32}的整数加法；

X＜＜S：X循环左移S位（O≤S≤31）

MD4的轮1，2，3分别使用了函数f,g和h。

它们的定义如下：注意，取三个字做为输入！

$f(X,Y,Z)=(X\wedge Y)\vee(\overline{X}\wedge Z)$

$g(X,Y,Z)=(X\wedge Y)\vee(X\wedge Z)\vee(Y\wedge Z)$

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

设计的**MD4**是非常快的，且实际上在**Sun SPARC**工作站上软实现可达速度**1.4M bytes/S**。另一方面，具体谈论象**MD4**这样的散列函数的安全性是困难的，因为它不是基于一个已经很好研究过的问题，如大整数分解，离散对数问题。所以，象采用**DES**的情况一样，仅能在一段时间内取得对该体制的安全性的信任。

MD4的增强版本称为**MD5**，于**1991**年提出。**MD5**使用了四轮来代替**MD4**的三轮，且运算速度大约比**MD4**慢**30%**。（在**SPARC**工作站上大约为**0.9Mbytes/S**）。

下面给出MD4中，轮1，2，3的完整描述：

MD4中的第一轮：

1° A=（A+f（B，C，D）+X[0]）＜＜3，

2° D=（D+f（A，B，C）+X[1]）＜＜7，

3° C=（C+f（D，A，B）+X[2]）＜＜11，

4° B=（B+f（C，D，A）+X[3]）＜＜19，

5° A=（A+f（B，C，D）+X[4]）＜＜3，

6° D=（D+f（A，B，C）+X[5]）＜＜7，

7° C=（C+f（D，A，B）+X[6]）＜＜11，

8° B=（B+f（C，D，A）+X[7]）＜＜19；

9° A=（A+f（B，C，D）+X[8]）＜＜3，

10° D=（D+f（A，B，C）+X[9]）＜＜7，

11° C=（C+f（D，A，B）+X[10]）＜＜11，

12° B=（B+f（C，D，A）+X[11]）＜＜19，

13° A=（A+f（B，C，D）+X[12]）＜＜3，

14° D=（D+f（A，B，C）+X[13]）＜＜7，

15° C=（C+f（D，A，B）+X[14]）＜＜11，

16° B=（B+f（C，D，A）X[15]）＜＜19，

MD4中的第二轮：

1° A=（A+g（B，C，D）+X[0]+5a827999）＜＜3，

2° D=（D+g（A，B，C）+X[4]+5a827999）＜＜5，

3° C=（C+g（D，A，B）+X[8]+5a827999）＜＜9，

4° B=（B+g（C，D，A）+X[12]+5a827999）＜＜13；

5° A=（A+g（B，C，D）+X[1]+5a827999）＜＜3，

6° D=（D+g（A，B，C）+X[5]+5a827999）＜＜5，

7° C=（C+g （D，A，B）+ X[9]+5a827999）＜＜9，

8° B=（B+g（C，D，A）+X[13]+5a827999）＜＜13；

9° A=（A+g（B，C，D）+X[2]+5a827999）＜＜3，

10° D=（D+g（A，B，C）+X[6]+5a827999）＜＜5，

11° C=（C+g（D，A，B）+X[10]+5a827999）＜＜9，

12° B=（B+g（C，D，A）+X[14]+5a827999）＜＜13；

13° A=（A+g（B，C，D）+X[3]+5a827999）＜＜3，

14° D=（D+g（A，B，C）+X[7]+5a827999）＜＜5，

15° C=（C+g（D，A，B）+X[11]+5a827999）＜＜9，

16° B=（B+g（C，D，A）+X[15]+5a827999）＜＜13。

MD4中的第三轮：

1° A=（A+h（B，C，D）+X[0]+6ed9ebal）＜＜3，

2° D=（D+h（A，B，C）+X[8] +6ed9ebal）＜＜9，

3° C=（C+h（D，A，B）+X[4] +6ed9ebal）＜＜11，

4° B=（B+h（C，D，A）+X[12] +6ed9ebal）＜＜15，

5° A=（A+h（B，C，D）+X[2] +6ed9ebal）＜＜3，

6° D=（D+h（A，B，C）+X[10] +6ed9ebal）＜＜9，

7° C=（C+h（D，A，B）+X[6] +6ed9ebal）＜＜11，

8° B=（B+h（C，D，A）+X[14] +6ed9ebal）＜＜15；

9° A=（A+h（B，C，D）+X[1] +6ed9ebal）＜＜3，

10° D=（D+h（A，B，C）+X[9] +6ed9ebal）＜＜9，

11° C=（C+h（D，A，B）+X[5] +6ed9ebal）＜＜11，

12° B=（B+h（C，D，A）+X[13] +6ed9ebal）＜＜15；

13° A=（A+h（B，C，D）+X[3] +6ed9ebal）＜＜3，

14° D=（D+h（A，B，C）+X[11] +6ed9ebal）＜＜9，

15° C=（C+h（D，A，B）+X[7] +6ed9ebal）＜＜11，

16° B=（B+h（C，D，A）+X[15] +6ed9ebal）＜＜15。

下面介绍最近关注中的**HASH**算法

- MD5
- SHA-1
- SHA-512
- HMAC

MD5简介

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

- **MD5把数据按512-bit长度进行分组.**
- **MD5的hash值是128-bit.**
- **随着穷举攻击和差分攻击对MD5的威胁,该算法一直没有成为标准算法.见教材[Berson92].**
- **现行美国标准SHA-1以MD5的前身MD4为基础.**

MD5算法步骤

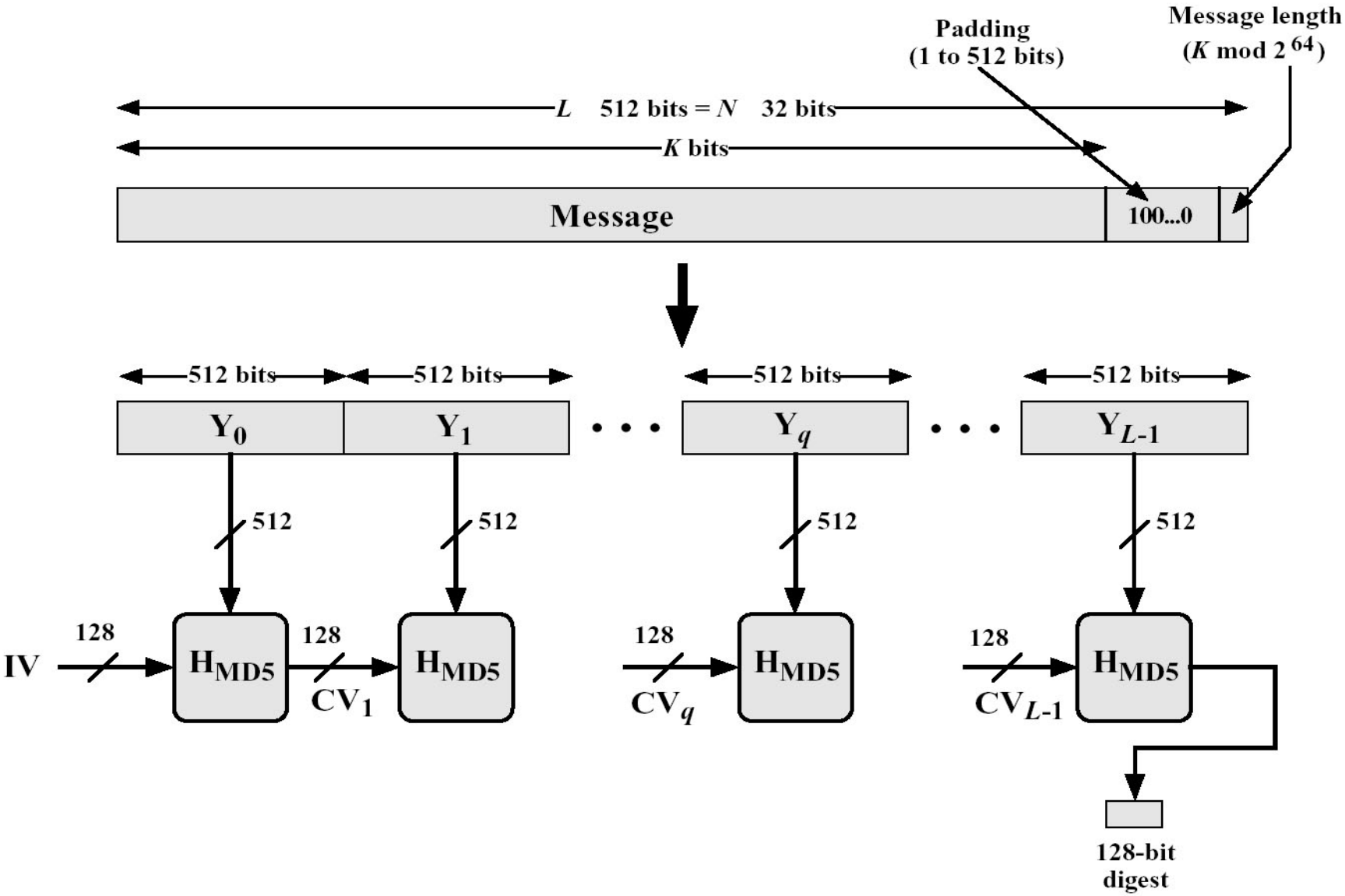
- **Step 1: Padding M ➔ M_1**
  - **$|M_1| \equiv 448 \bmod 512$**
  - **$|M_1| > |M|$ ➡**
  - **如果$|M| \equiv 448 \bmod 512$,则$|M_1| = |M|+512$**
  - **Padding内容: 100…0**
- **Step 2: Append 64-bit length M_1 ➔ M_2**

后缀的64 比特信息是值α的2-进制64位表示,其中

$\alpha \equiv |M| \bmod 2^{64}$,低字节在前 (little-endian).

于是 ,$|M_2|$为512的整倍数,将其512长分组:$Y_0, Y_1, \ldots, Y_{L-1}$.

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
128
128
128
128
IV
H_{MD5}
H_{MD5}
H_{MD5}
H_{MD5}
CV_1
CV_q
CV_{L-1}
128-bit
digest

- **Step 3:**初始和过程结果存入4个32位寄存器,左为16进制初值:

  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:**以512比特(16字)分组处理消息,按4轮压缩运算;各轮使用不同的基本逻辑函数,分别记为F,G,H和I.每轮处理即时的分组Y_q和128比特缓冲区A,B,C,D的内容. 每轮使用T[i]表,$2^{32} \times \text{abs}(\sin(i)) < 1$.

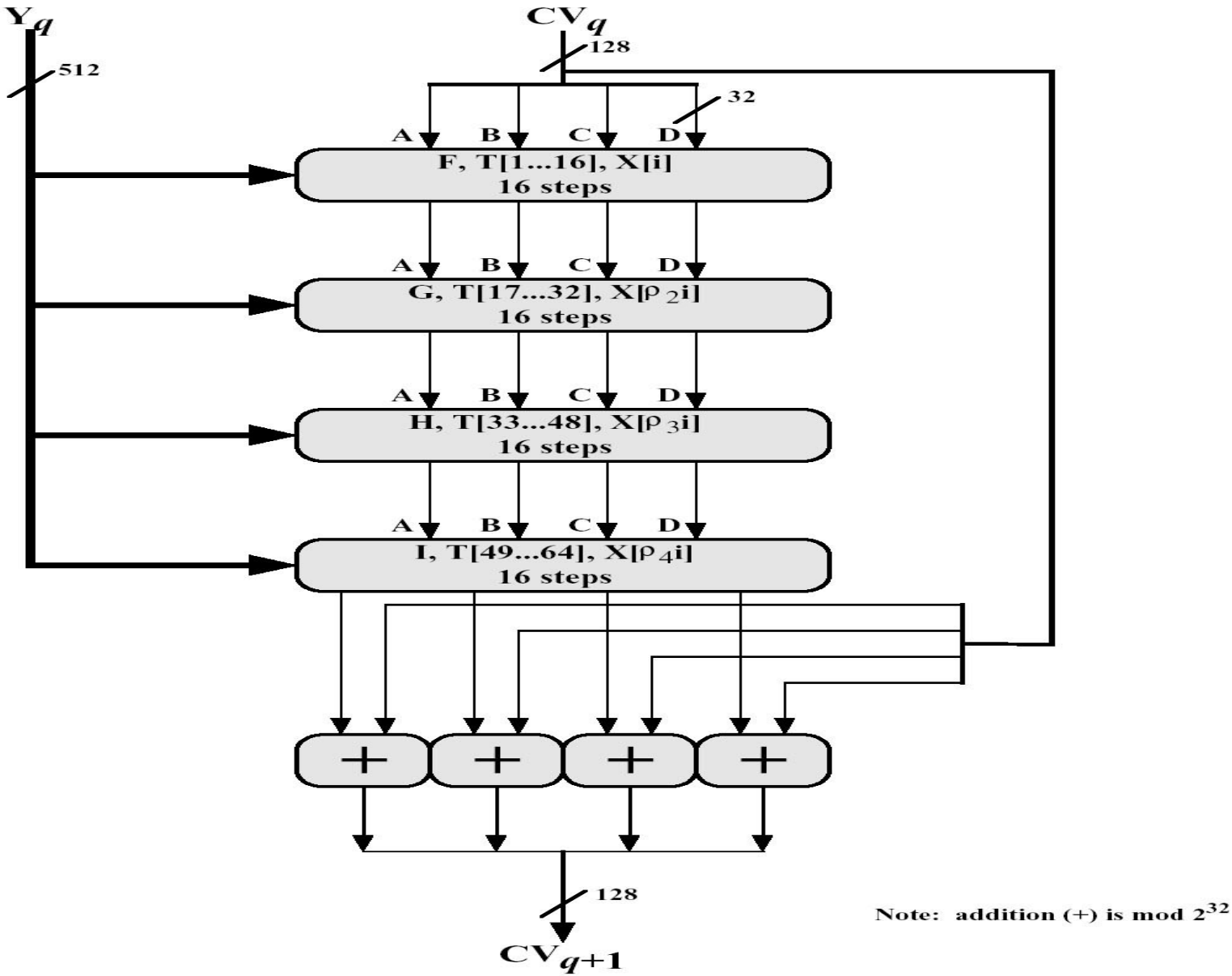
  $CV_0 = IV$

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

- **Step 5: Output**所有512比特分组处理完毕后,第L个分组输出128比特!

  $MD = CV_L$

MD5 Step 4:



Note: addition (+) is mod 2^{32}

注记:对 Step 4的说明:

我们知，$CV_0=IV$，　　$CV_i=H_{MD5}(CV_{i-1},Y_i)$

- (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。

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

其中，

a,b,c,d = 缓冲区的四个字，以一个给定的次序排列；

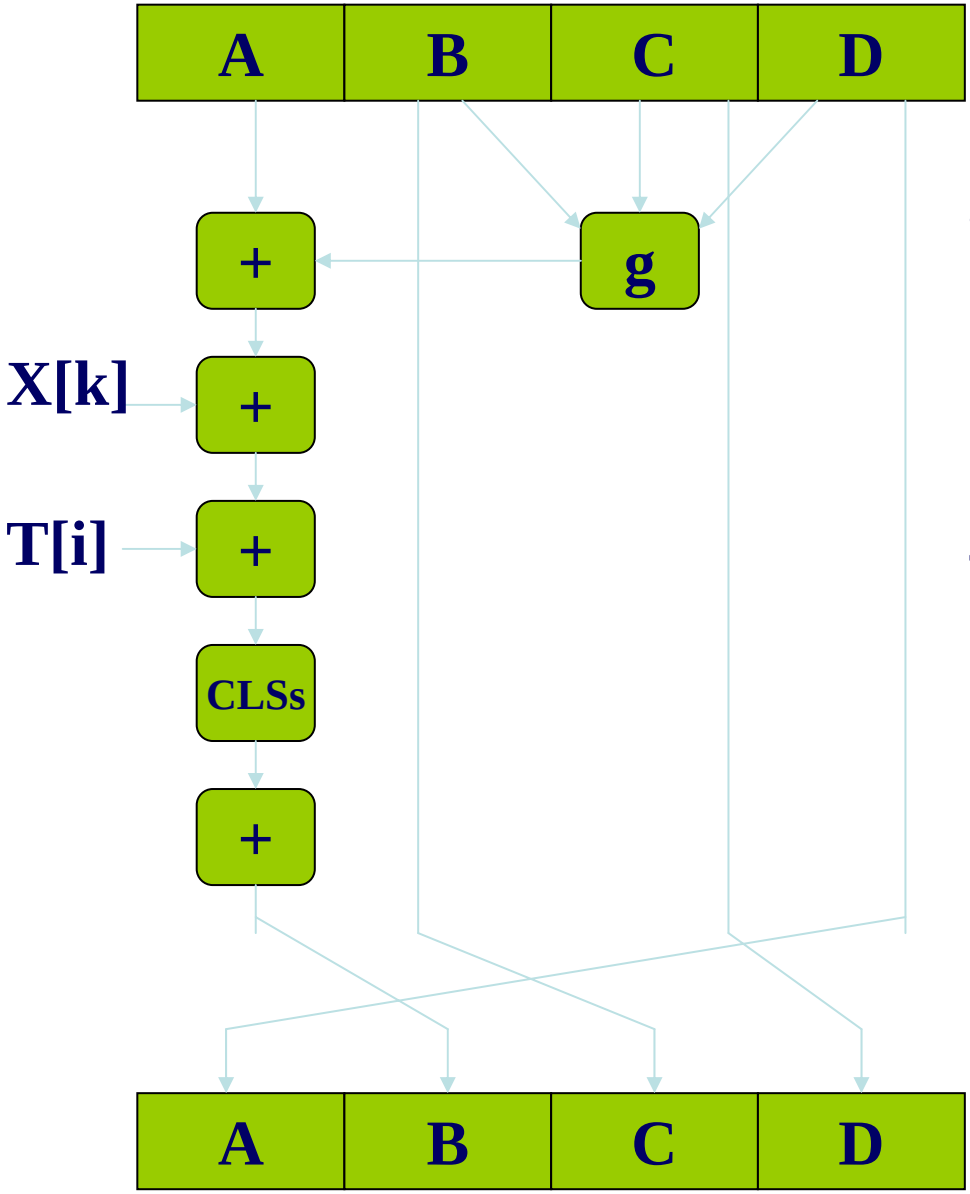
g = 基本逻辑函数F,G,H,I之一；

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

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

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

$\rho_2 i = 7i \bmod 16$

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) {

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

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

}

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

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 在他的第三版 "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

Collisions such that:

$$M' = M + \Delta C_1, \Delta C_1 = (0,0,0,0,2^{31},0000000,2^{15},...2^{31},0)$$

$$N_i' = N_i + \Delta C_2, \Delta C_2 = (0,0,0,0,2^{31},.... - 2^{15},...2^{31},0)$$

$$MD5(M, N_i) = MD5(M', N_i')$$

non-zeros at position 4,11 and 14

Running time: On IBM P690.

1. Finding (M, M') :about one hour (the fastest only needs 15 minutes)

2. Finding (N_i, N_i') : 15 seconds to 5 minutes.

Attack method

- Differential attack
- Find the best differential and determine all the conditions that the differential occurs.

Colliding X.509 Certificates version 1.0, 1st March 2005

Arjen Lenstra1,2, Xiaoyun Wang3, and Benne de Weger2

1 Lucent Technologies, Bell Laboratories, Room 2T-504

600 Mountain Avenue, P.O.Box 636, Murray Hill, NJ 07974-0636, USA

2 Technische Universiteit Eindhoven

P.O.Box 513, 5600 MB Eindhoven, The Netherlands

3 The School of Mathematics and System Science, Shandong University

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(RFC3174)

输入长度不大于2^{64}比特，以512位分组；输出160比特摘。

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，按512分组成: $Y_0, Y_1, \ldots, Y_{L-1}$

- **Step 3:** 初始化 MD 缓冲区（A，B，C，D，E）

```
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)
```

初始化的前4个与MD5的值相同，但SHA-1中这些值以高端格式存储!

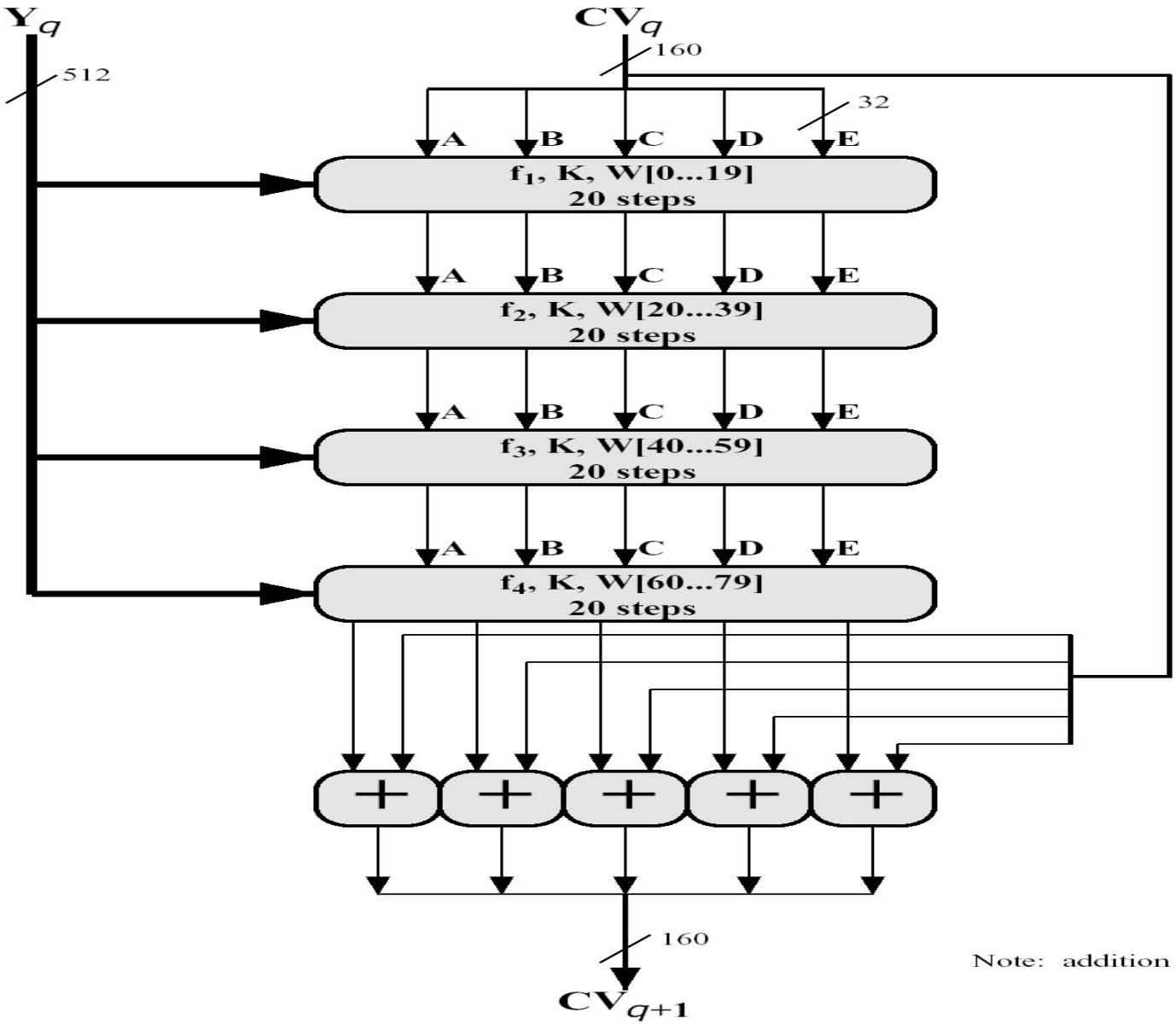
- **Step 4:** 以512位的分组（16字）作为每轮的输入！共有4轮，每轮执行20步迭代。

$$CV_0 = IV$$

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

4轮运算结构相同，但各轮使用不同的逻辑函数：f_i 其中 i=1,2,3,4 . 每轮输入是当前处理的512位分组（Y_q)和160位缓冲区ABCDE的内容。

SHA-1 step 4: 示意图



Note: addition (+) is mod 2^{32}

对 step 4轮函数的解释，见P269：

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

Step 5,Output:所有L个**512**分组处理完后，第L个分组输出就是**160**比特的消息摘要！

整个处理过程：$CV_0 = IV$

$$CV_{q+1} = SUM_{32}(CV_q, ABCDE\ I_q)$$

$$MD = CV_L$$

对SHA-1轮压缩函数的进一步注记：

- **Four rounds:　　$0 \le t < 80$（见P266下）。**

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

$$B \leftarrow B<<<30$$

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

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

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

- $f(t,B,C,D) = B \oplus C \oplus D$　　$20 \le t < 40$

  $K[t] = 2^{30} \times sqrt(3)$

- $f(t,B,C,D) = (B\&C)|(B\&D)|(C\&D)$　　$30 \le t < 60$

  $K[t] = 2^{30} \times sqrt(5)$

- $f(t,B,C,D) = B \oplus C \oplus D$　　$60 \le t < 80$

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

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；

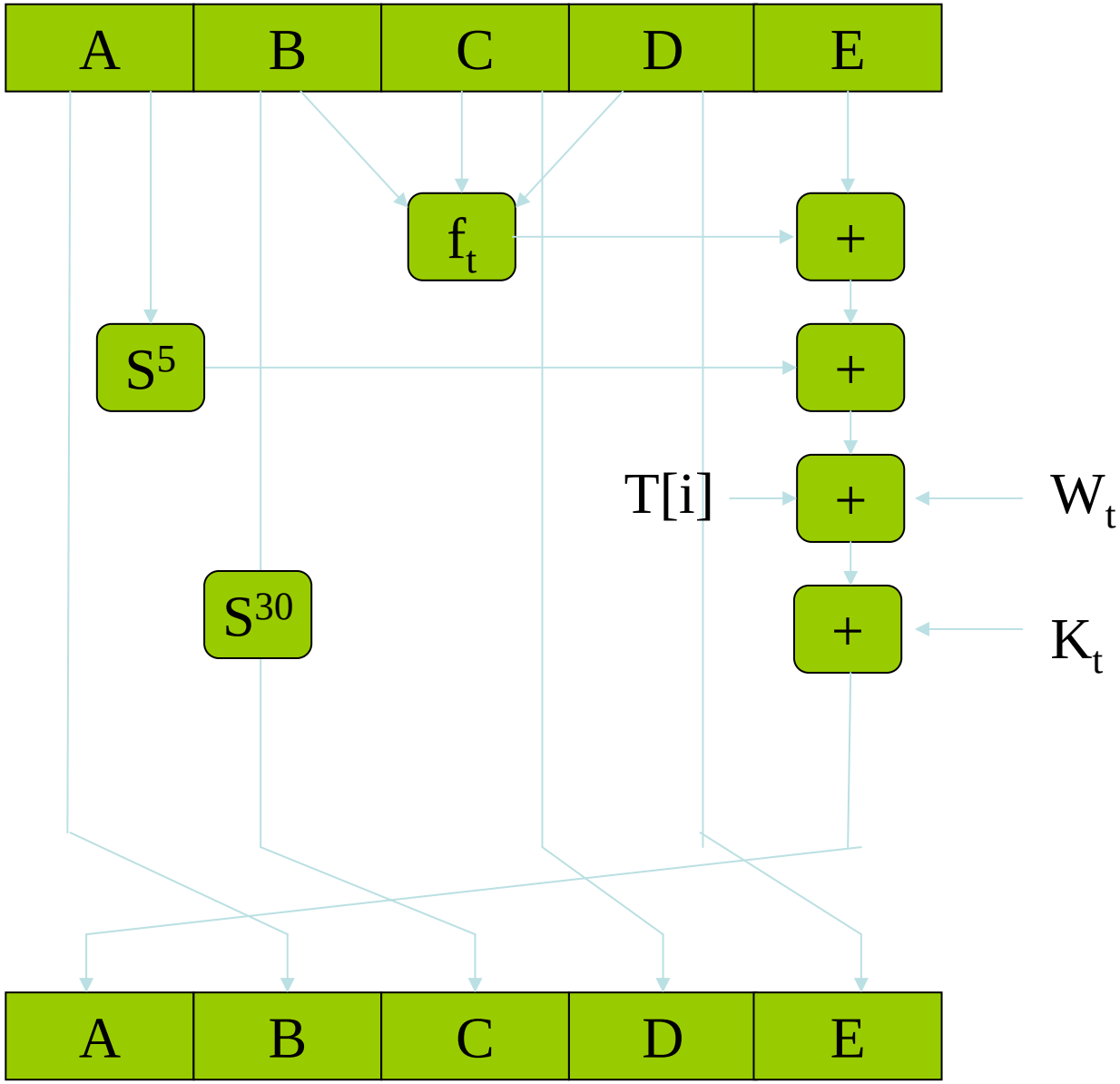
f(t,B,C,D) = 步t的基本逻辑函数；

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

安全散列函数的修定

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

“SHA-1 ”嵌入诸如加密软件(PGP)和安全插座层协定(SSL)等使用广泛的程序中。“SHA-1 ”已获美国标准与技术研究院(NIST)的认证，而且是唯一获准用于美国政府“数字签名标准”的签名演算法。“SHA-1 ”产生160 位的数字与字串，长度比MD5 产生的128 位更长，因此被视为更安全.自2005年以来,对SHA-1攻击代价降至2^{63}.人们逐渐提倡使用SHA-512.

2005年NIST对未来散列函数的探讨:

压缩函数　一定要做到“可证明安全的”,如何进行探索?　Rivest的杂凑方法好吗?

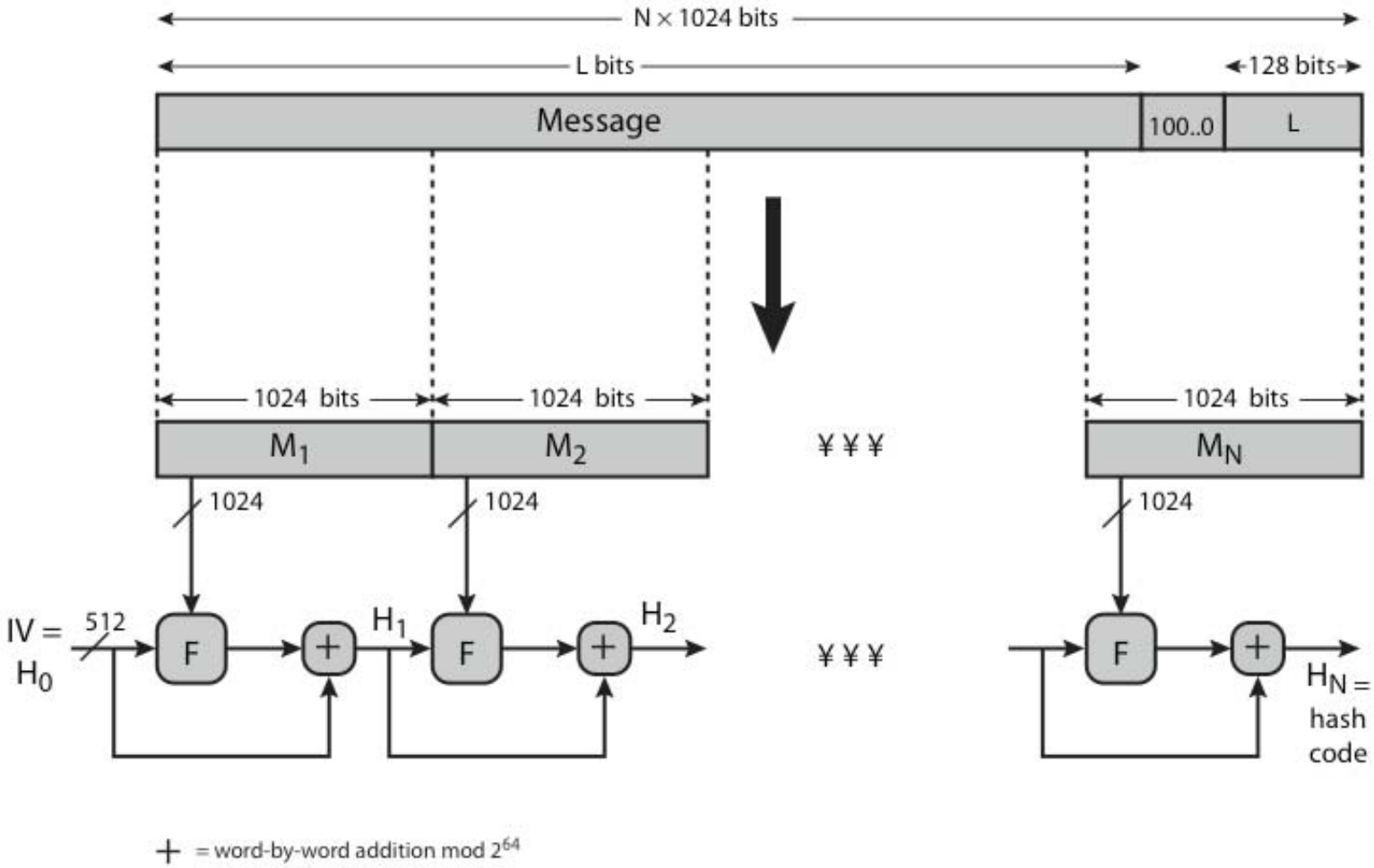
迭代结构　要改进Merkle-Damgard 给出的方法吗?

对策　对来自各方的攻击 要有哪些应对方案?

与AES 比较　对AES的讨论开始于1997年,那时侯我们储备了大量有　关分组密码的知识;然而我们对散列函数的知识很难与其比较.

其它规范　期待的未来散列函数还应有哪些特别的性能?Do we know enough yet?

SHA-512 介绍


N × 1024 bits
L bits
128 bits
Message
100..0
L
1024 bits
1024 bits
1024 bits
M1
M2
MN
1024
1024
1024
IV =
H0
512
F
H1
F
H2
F
HN =
hash
code
+ = word-by-word addition mod 2^64

输入消息长小于2^{128} 比特,按1024比特分组,输出消息摘要为512比特.

对给定文本的比特序列末尾填充10…0,使其长度

$r \equiv 896 \pmod{1024}$;然后在此基础上后续一个128比特的最高有效字节在前的正整数.于是将原始消息扩展成长度为1024整数倍的消息,按1024分组成M_1 , M_2 , …, M_N . 散列函数的初始、中间和最终结果都接连保存于512比特的缓冲区中；缓冲区由8个64比特的寄存器组成

（a, b, c, d, e, f ,g ,h)

用2、3、5、7、11、13、17、19这8个素数的平方根2进制表示的小数部分依次取64比特来初始化这8个寄存器（见p256）

SHA-512单轮运行图

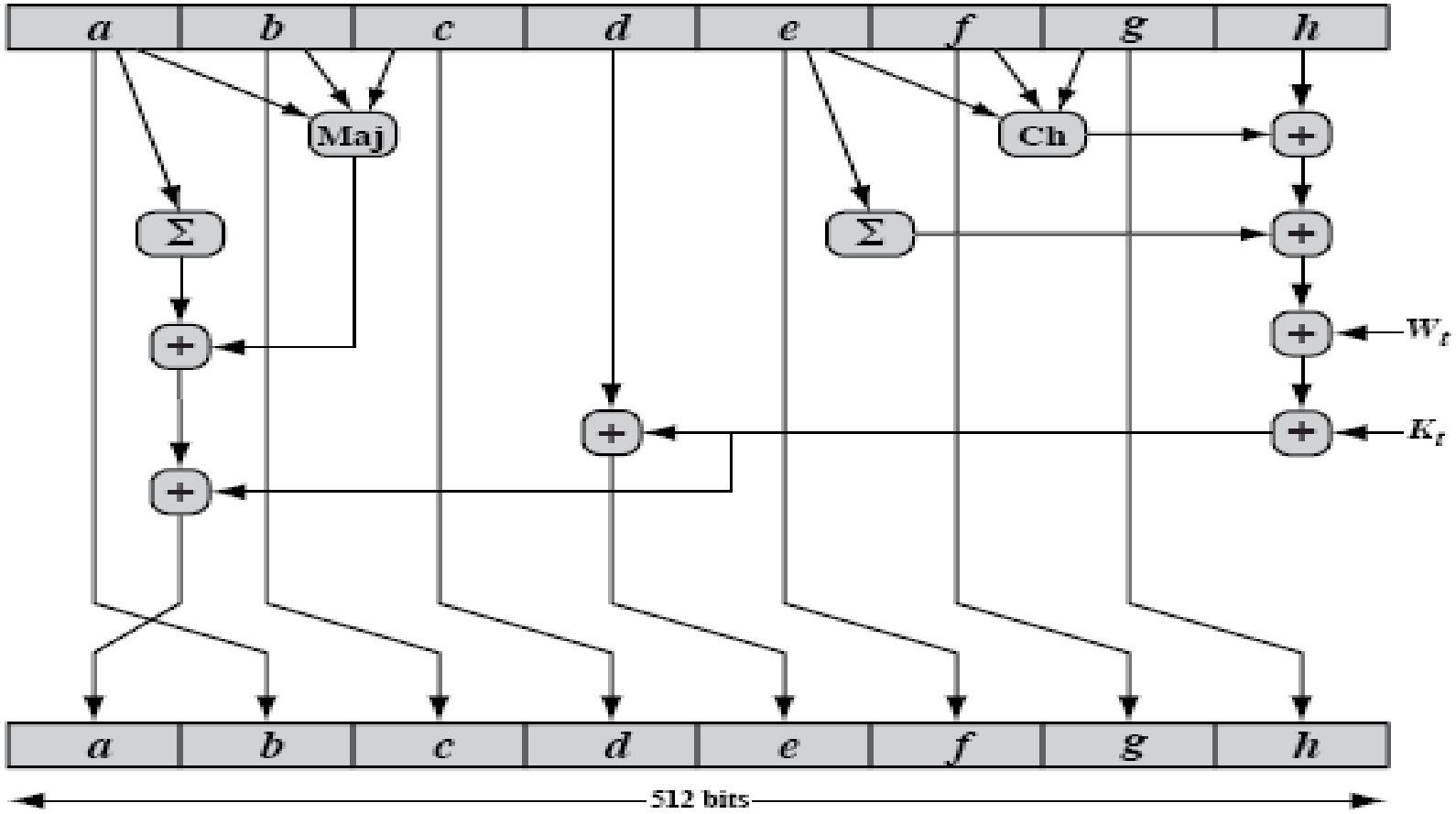
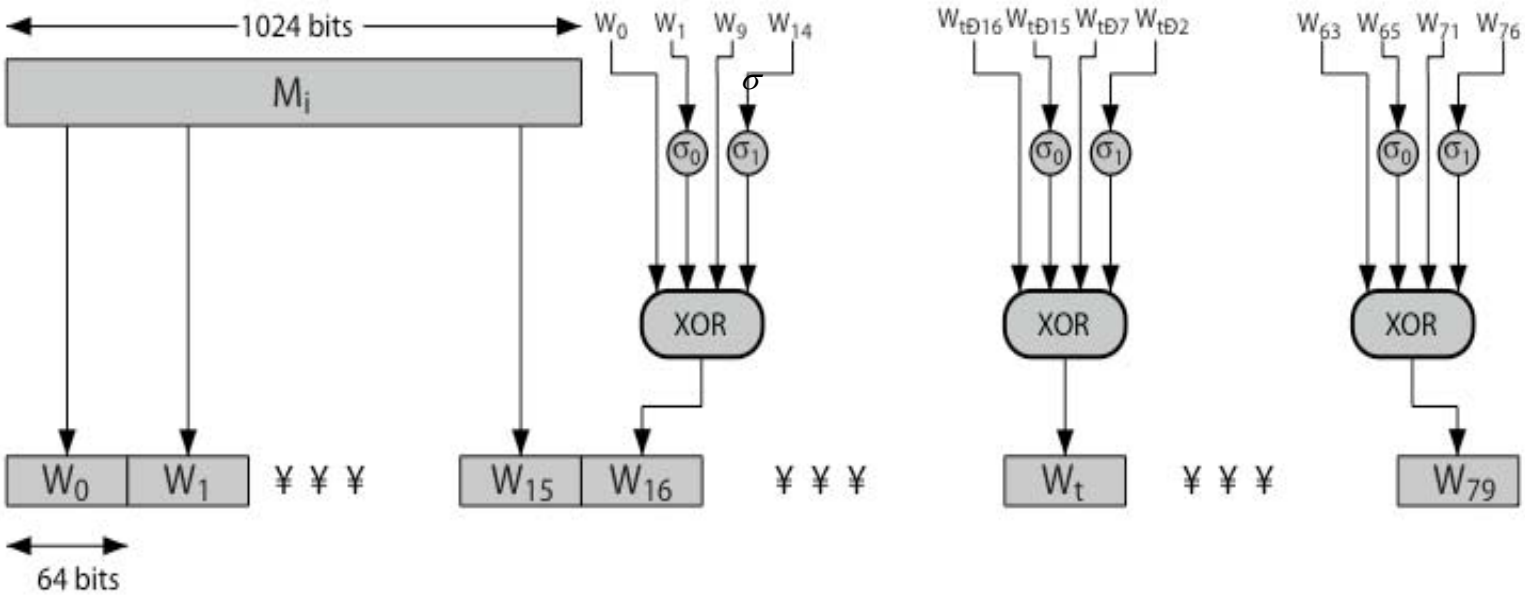


Figure 12.3 Elementary SHA-512 Operation (single round

对于算法的核心压缩函数F由80轮的子函数组成。每一轮，如第t轮，使用一个64比特字Wt和附加常数K_t进行输入，K_t 常数是取前80个素数分别开3次方，取小数部分的前64比特，其中W_t的生成办法:



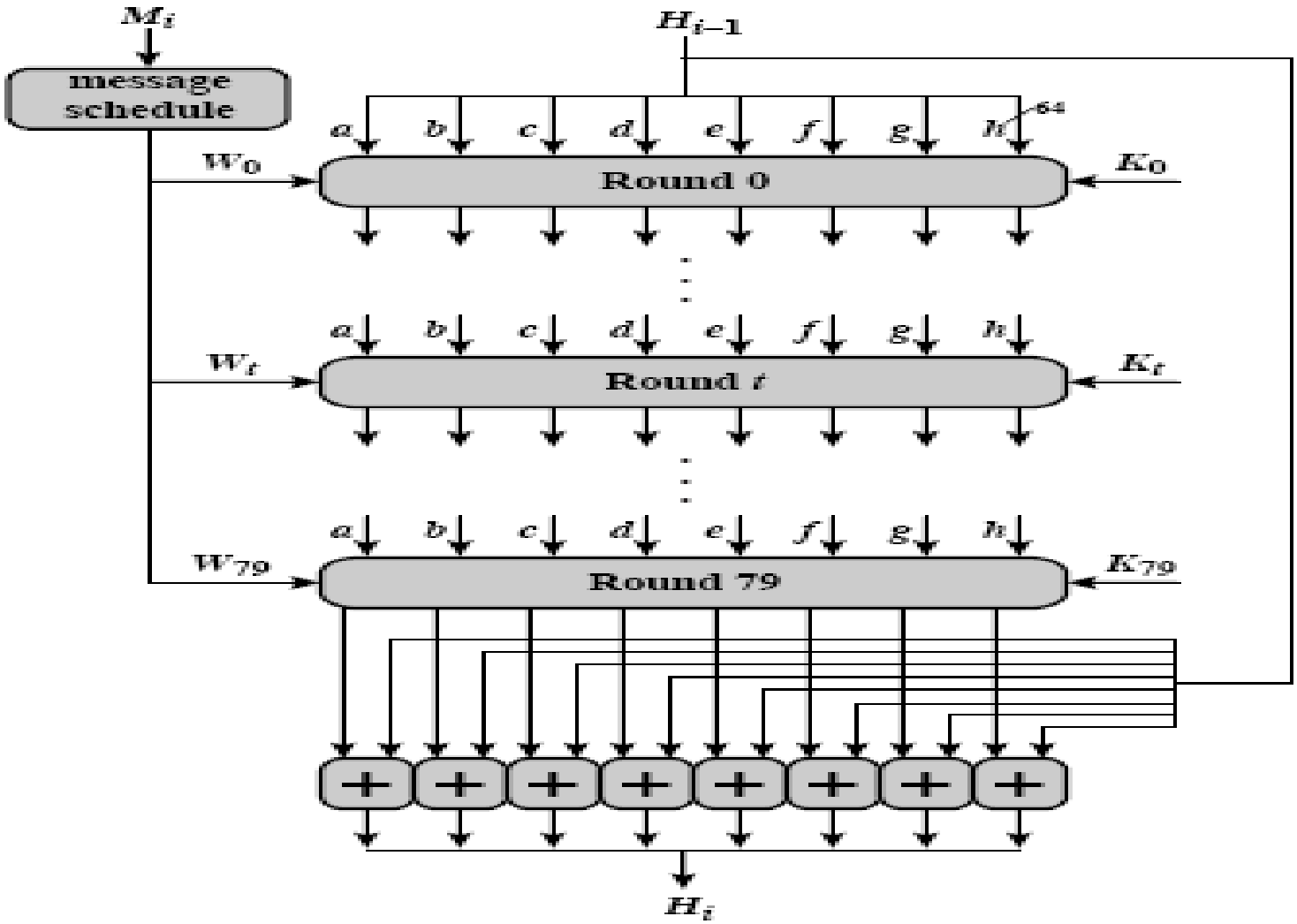
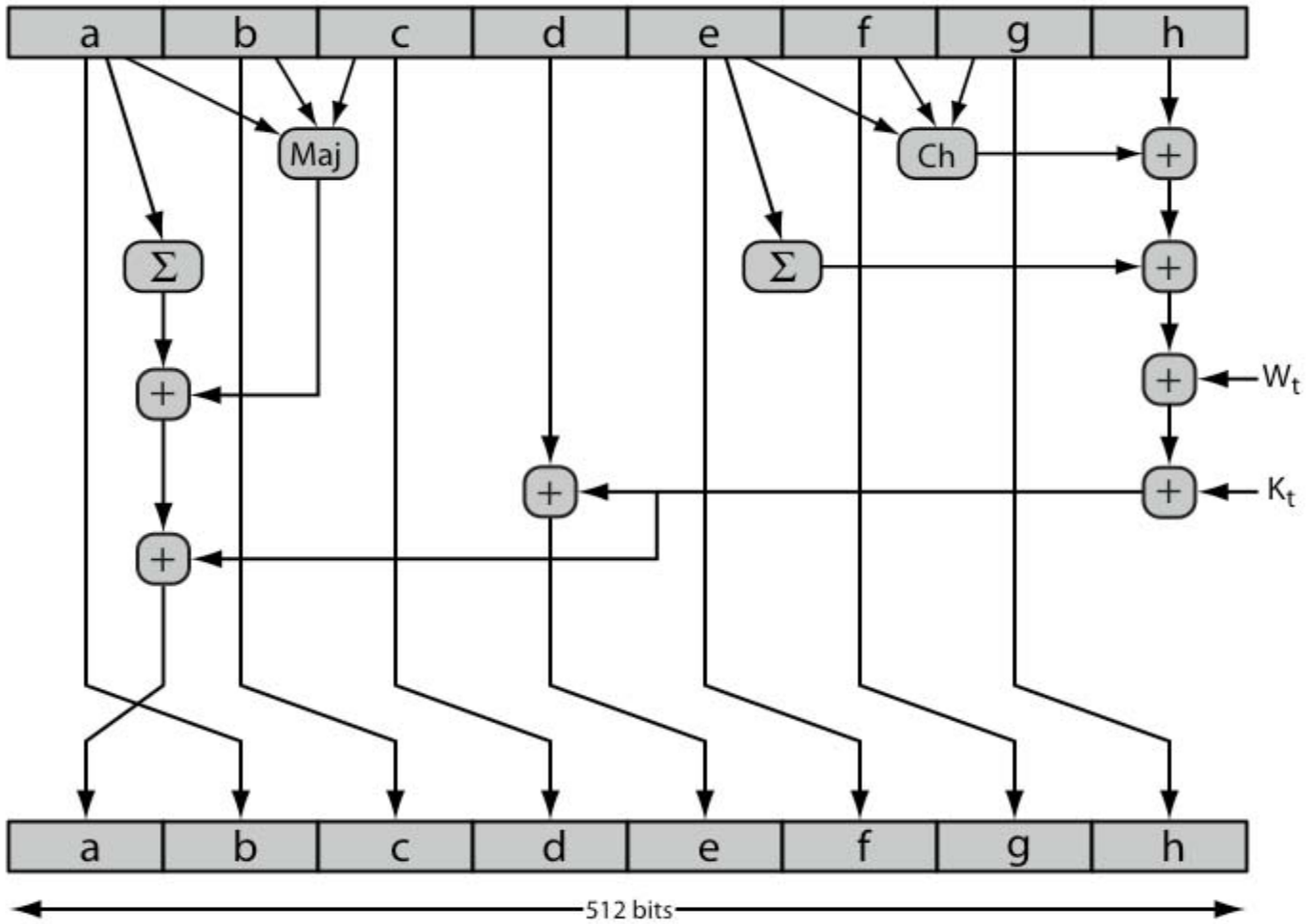


Figure 12.2 SHA-512 Processing of a Single 1024-Bit Block

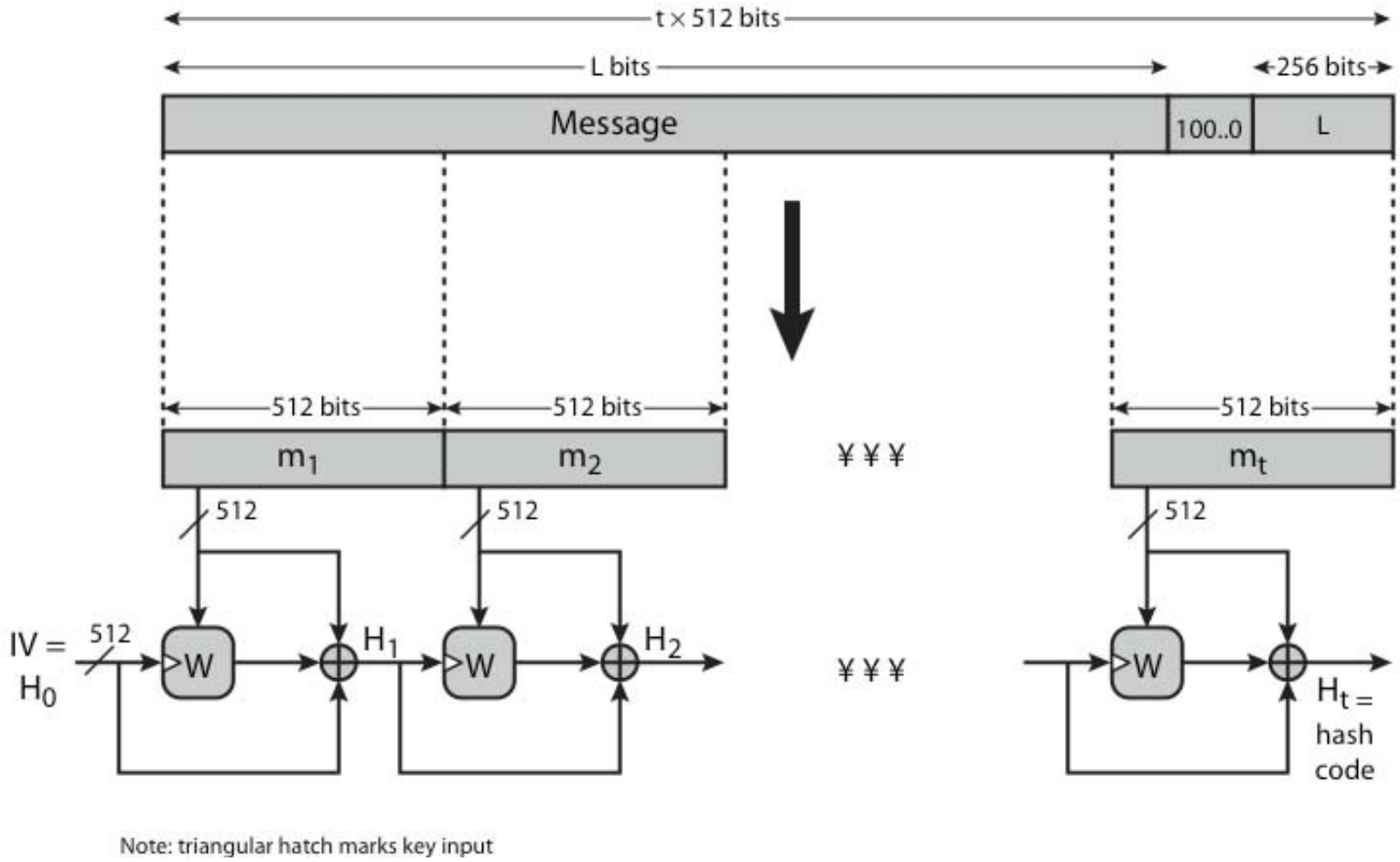
SHA-512轮函数



Whirlpool

- **now examine the Whirlpool hash function**
- **endorsed by European NESSIE project**
- **uses modified AES internals as compression function**
- **addressing concerns on use of block ciphers seen previously**
- **with performance comparable to dedicated algorithms like SHA**

Whirlpool Overview

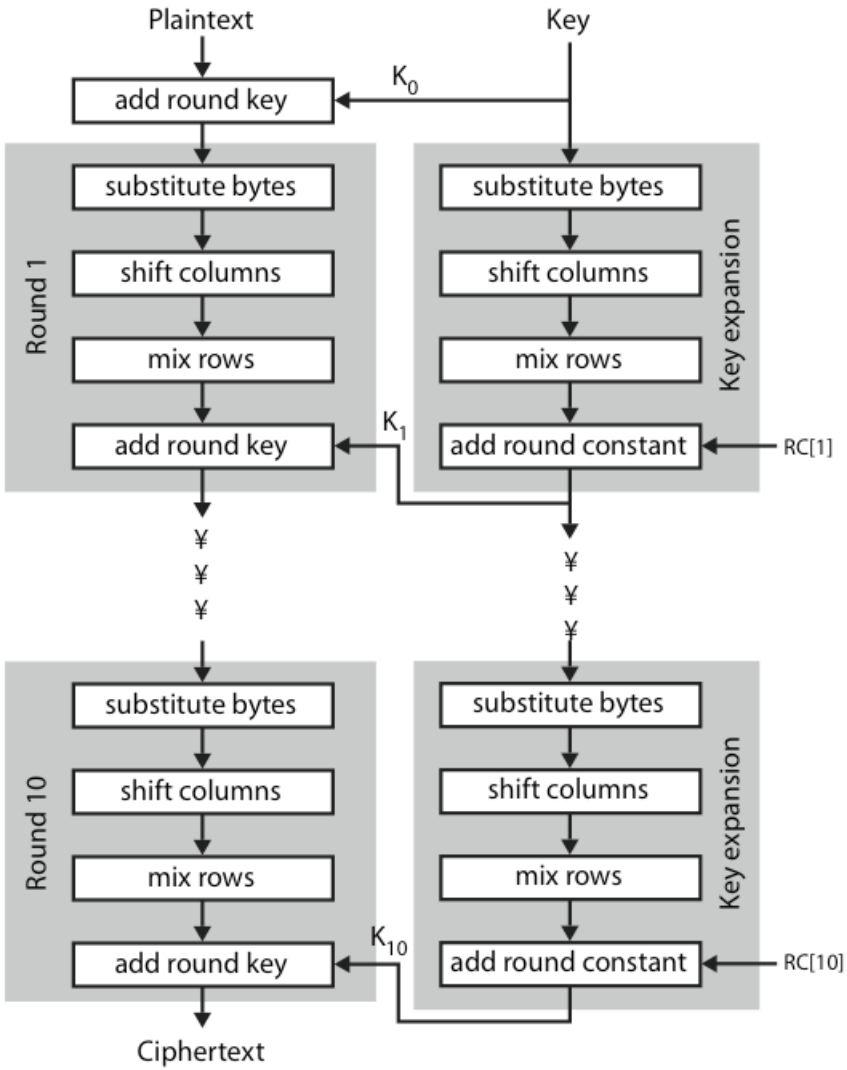


Note: triangular hatch marks key input

Whirlpool Block Cipher W

- designed specifically for hash function use
- with security and efficiency of AES
- but with 512-bit block size and hence hash
- similar structure & functions as AES but
  - input is mapped row wise
  - has 10 rounds
  - a different primitive polynomial for GF(2^8)
  - uses different S-box design & values

Whirlpool Block Cipher W



Whirlpool Performance & Security

- **Whirlpool is a very new proposal**
- **hence little experience with use**
- **but many AES findings should apply**
- **does seem to need more h/w than SHA, but with better resulting performance**

Keyed Hash Functions as MACs

- want a MAC based on a hash function
  - because hash functions are generally faster
  - code for crypto hash functions widely available
- hash includes a key along with message
- original proposal:

  `KeyedHash = Hash(Key|Message)`

  - some weaknesses were found with this
- eventually led to development of HMAC

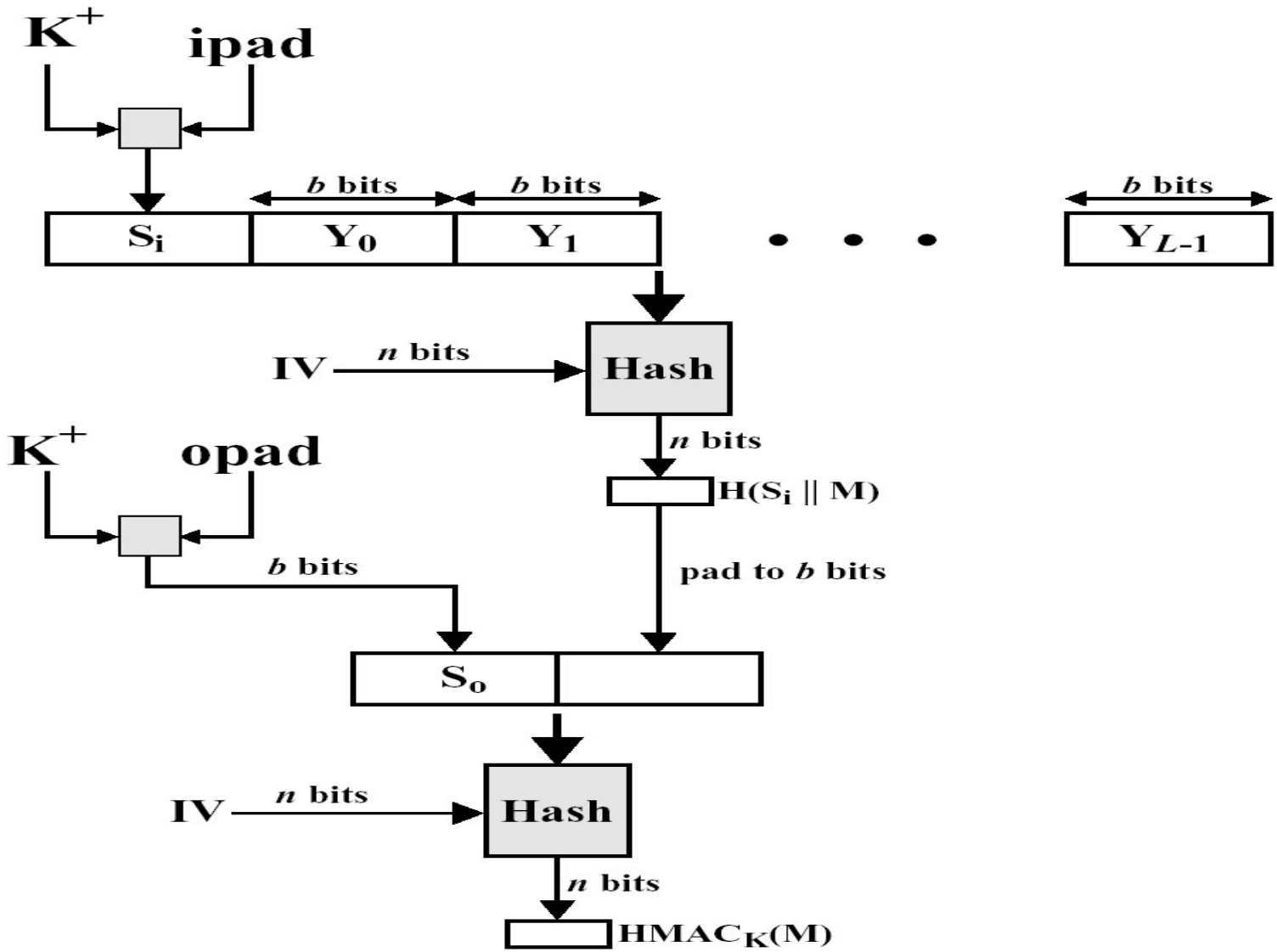
HMAC简介

- 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
n bits
K+
opad
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

❺HMAC=f[IV,S_o||f(IV,S_i||M)]

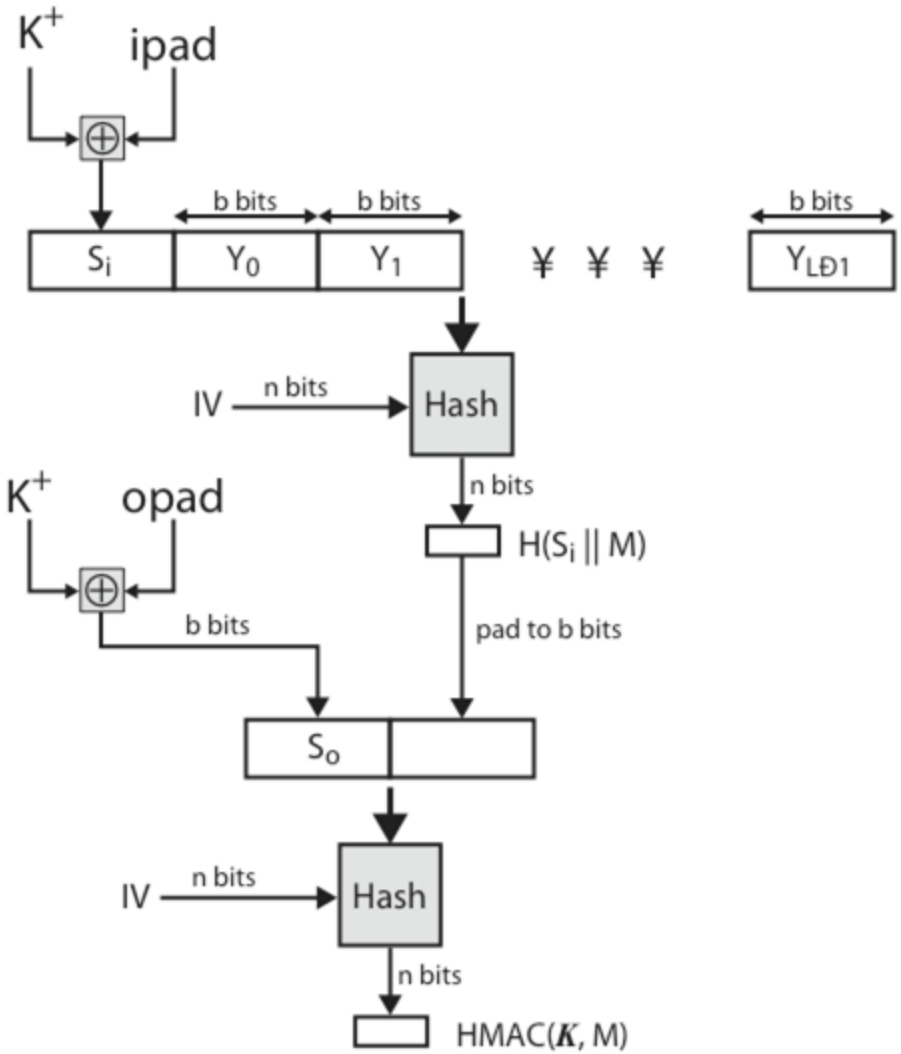
HMAC

- specified as Internet standard RFC2104
- uses hash function on the message:

$$HMAC_K = Hash[(K^+ \text{ XOR opad}) \,||\, Hash[(K^+ \text{ XOR ipad})||M)]]$$

- where K^+ is the key padded out to size
- and opad, ipad are specified padding constants
- overhead is just 3 more hash calculations than the message needs alone
- any hash function can be used
  - eg. MD5, SHA-1, RIPEMD-160, Whirlpool

HMAC Overview



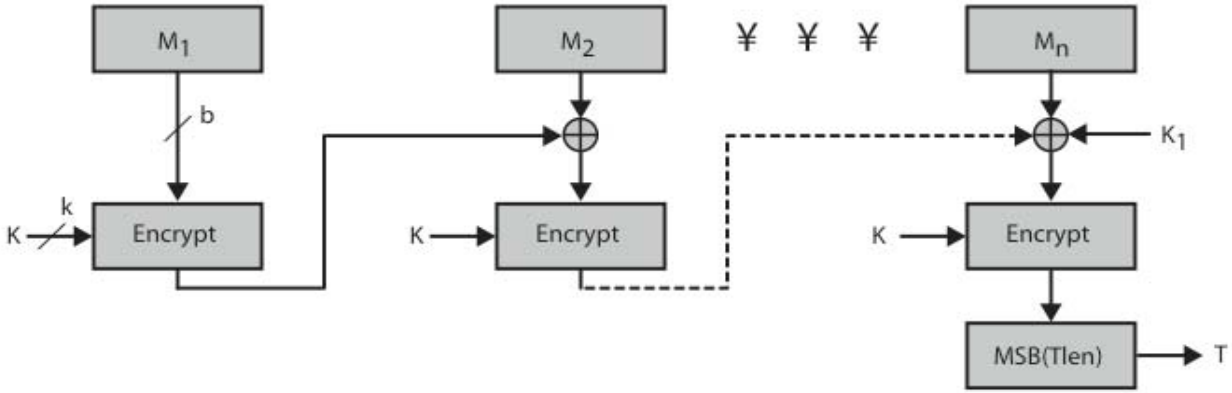
HMAC Security

- proved security of HMAC relates to that of the underlying hash algorithm
- attacking HMAC requires either:
  - brute force attack on key used
  - birthday attack (but since keyed would need to observe a very large number of messages)
- choose hash function used based on speed verses security constraints

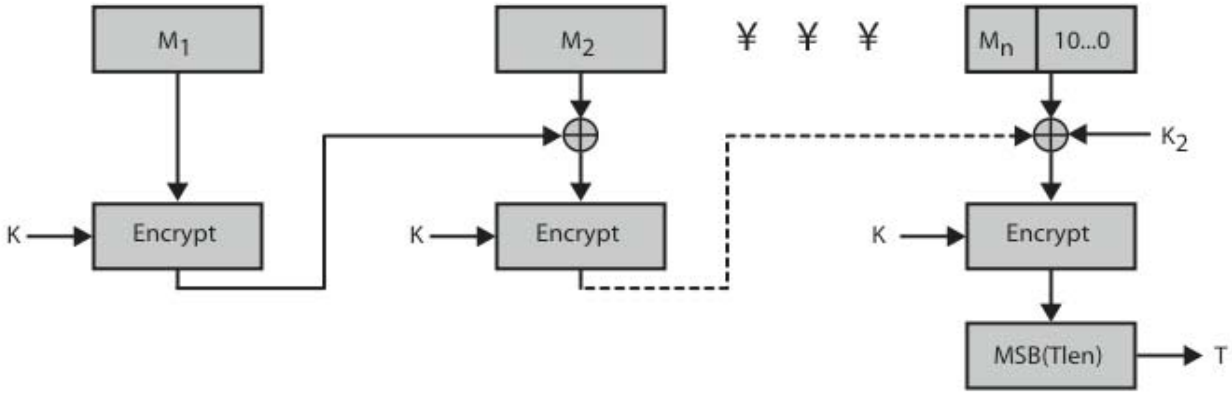
CMAC

- previously saw the DAA (CBC-MAC)
- widely used in govt & industry
- but has message size limitation
- can overcome using 2 keys & padding
- thus forming the Cipher-based Message Authentication Code (CMAC)
- adopted by NIST SP800-38B

CMAC Overview



(a) Message length is integer multiple of block size



(b) Message length is not integer multiple of block size

Figure 12.12 Cipher-Based Message Authentication Code (CMAC)

Summary

- **have considered:**
  - **some current hash algorithms**
    - **SHA-512 & Whirlpool**
  - **HMAC authentication using hash function**
  - **CMAC authentication using a block cipher**

07年11月2日美国NIST公开征集密码散列函数(**cryptographic hash Algorithm**),该函数一旦被选定,将被称之为“SHA-3”,作为联邦信息处理标准(FIPS)180-2、“安全的散列标准”.征收截止日为**2008年10月31日.**

(NIST has opened a public competition to develop a new cryptographic hash algorithm, which converts a variable length message into a short “message digest” that can be used for digital signatures, message authentication and other applications. The competition is NIST’s response to recent advances in the cryptanalysis of hash functions. The new hash algorithm will be called “SHA-3” and will augment the hash algorithms currently specified in FIPS 180-2, Secure Hash Standard. Entries for the competition must be received by October 31, 2008. The competition is announced in the Federal Register Notice published on November 2, 2007; further details of the competition will be available at the specific sites indicated in the menu on the left.)

CRYPTO-08 :

20Aug. 11:10 — 12:10

Rivest 演讲

The MD6 hash function

(戏称为 “南瓜”散列)

(aka “Pumpkin Hash”)

将作为SHA-3的候选算法

MD6 Team

- Dan Bailey
- Sarah Cheng
- Christopher Crutchfield
- Yevgeniy Dodis
- Elliott Fleming
- Asif Khan
- Jayant Krishnamurthy
- Yuncheng Lin
- Leo Reyzin
- Emily Shen
- Jim Sukha
- Eran Tromer
- Yiqun Lisa Yin

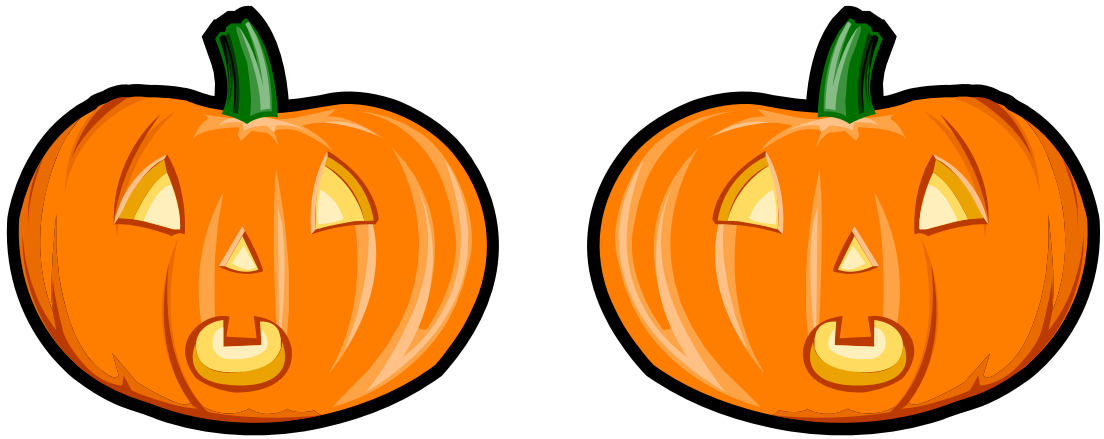
- Juniper Networks
- Cilk Arts
- NSF

Wang et al. break MD5 (2004)

- **Differential cryptanalysis (re)discovered by Biham and Shamir (1990). Considers step-by-step ``difference'' (XOR) between two computations…**
- **Applied first to block ciphers (DES)…**
- **Used by Wang et al. to break collision-resistance of MD5**
- **Many other hash functions broken similarly; others may be vulnerable…**

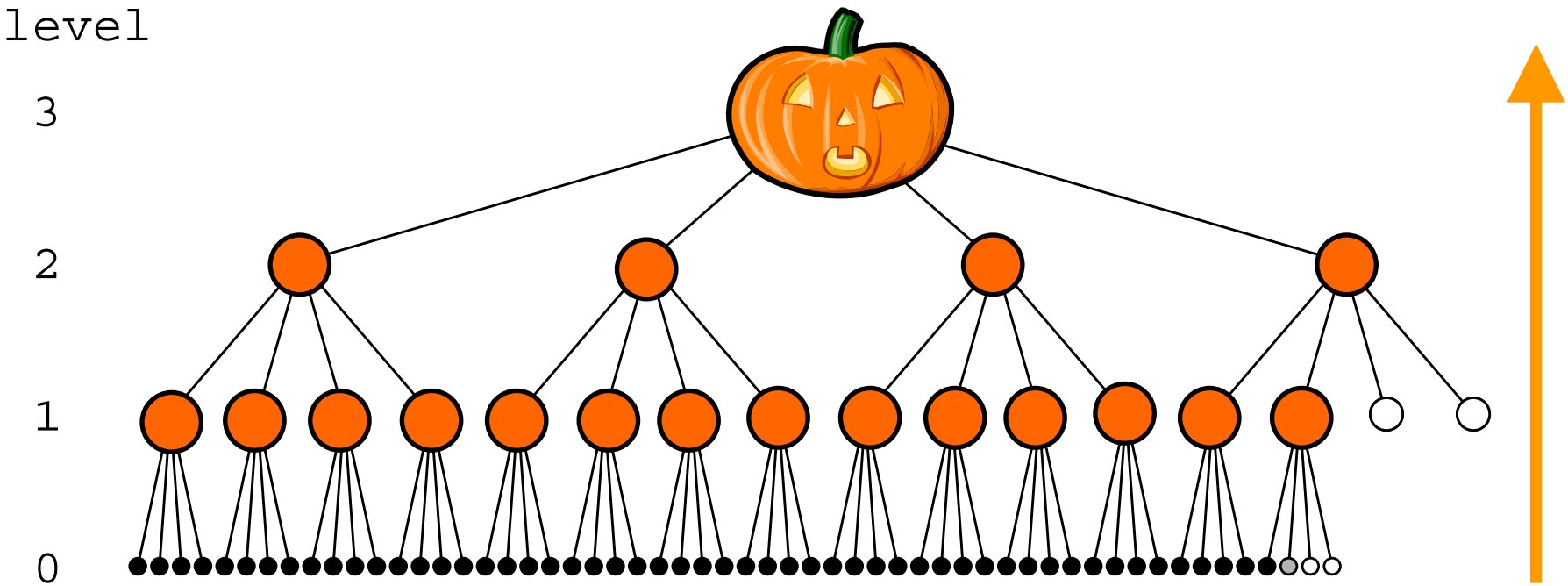
So... MD6 is...

- 可以证明它能够抵御差分攻击的 **(more on this later...)**



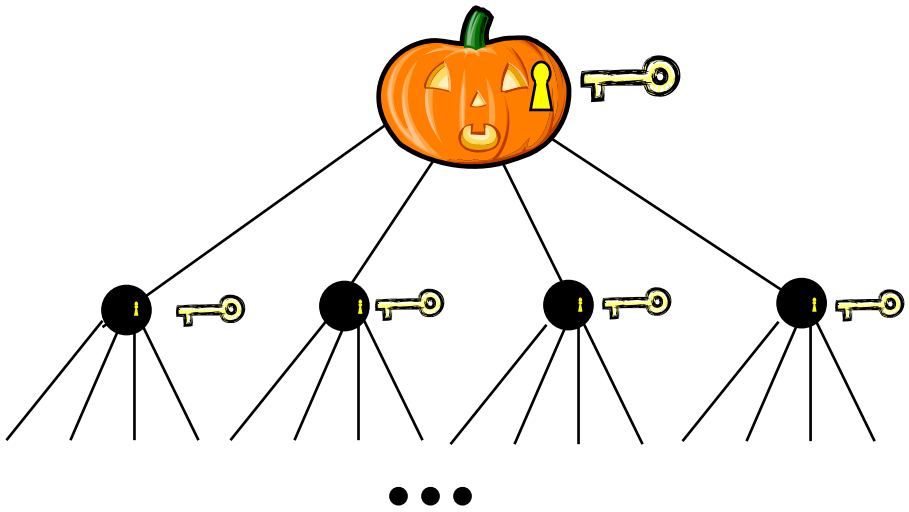
MD6 has...

- 采用从底端向上爬树的计算模型**(like Merkle-tree)**
- 对每一层结点的压缩比例为4:1



MD6 has...

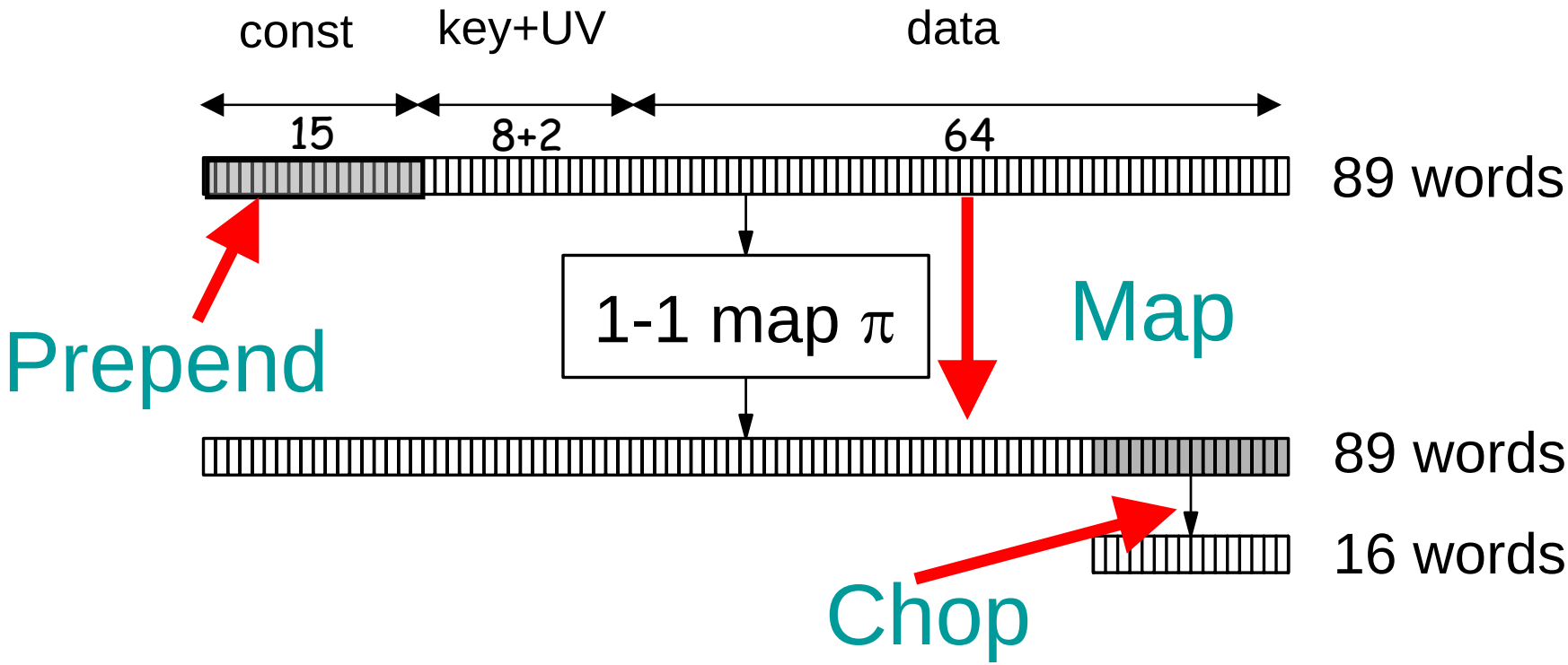
- 输入密钥 ***K***  最大可采用**512 bits**
- **K**是每一个压缩函数的密钥



MD6 uses...

- 整个运算是基于 **64-bit words**
- 所采取的基本运算方式如下：
  - **XOR** $\oplus$
  - **AND** $\wedge$
  - **SHIFT by fixed amounts:**
    - **x >> r** **>>**
    - **x << ℓ** **<<**

预置常数 + 映射 + 削减


const
key+UV
data
15
8+2
64
89 words
Prepend
1-1 map π
Map
89 words
16 words
Chop

详细可以进入**Ron Rivest** 个人网页 ,查找**MD6**文献