密 码 学

第八讲　中国商用密码HASH函数SM3

王后珍

武汉大学国家网络安全学院

空天信息安全与可信计算教育部重点实验室

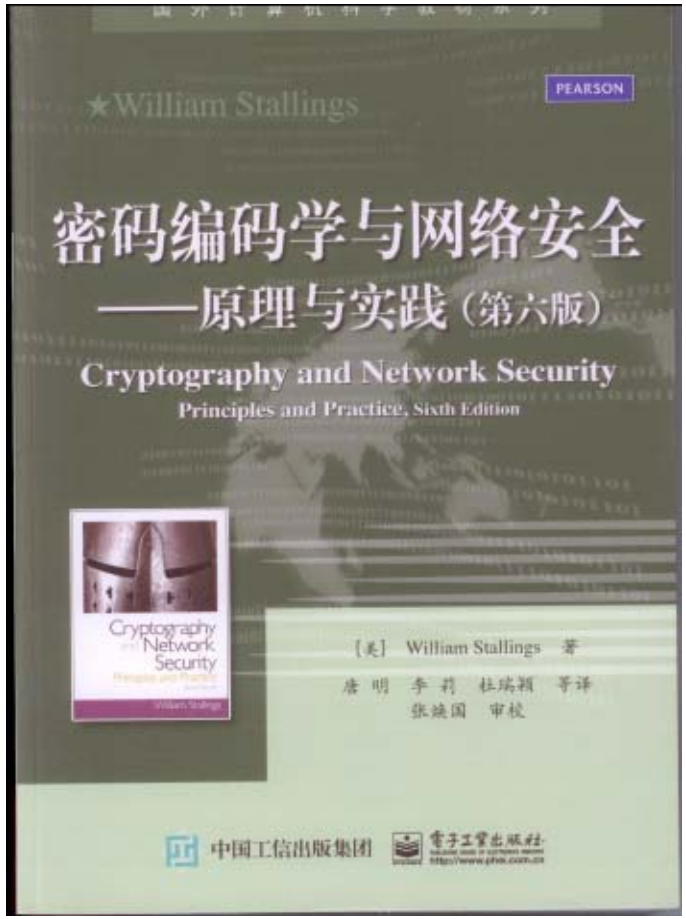
目 录

目 录

教材与主要参考书

教材

参考书



武漢大学

本讲内容

一、Hash函数的概念

二、中国商用密码Hash函数SM3

三、美国NIST的Hash函数SHA-3

一、HASH函数的概念

- Hash函数与加密函数不同的是，Hash函数并不直接使用密钥，它仅是输入消息的函数。

  h=H(M)。

- Hash函数H的输入是可变大小的消息M，输出h是固定大小的**Hash码**。
- **Hash码**有时也称为**消息(报文)摘要**，或**Hash值**。
  - Hash Code
  - Message digest
  - Hash function value

一、Hash函数的概念

1、Hash函数的定义

① Hash函数将任意长的数据*M*变换为定长的码*h*，记为：

$$h=Hash(M)\text{或 }h=H(M)。$$

- 一般，*h*的长度小于*M*的长度，因此HASH函数是一种压缩变换。

②实用性：对于给定的数据*M*，计算*h=Hash (M)*是高效的。

③安全性：

- 单向性：对于给定的$h=H(x)$，由*h*求出*x*在计算上是不可行的。
  - 称*x*是*h*的原像。由*h=H(x)*求出*x*，称为原像攻击。如果Hash函数具有单向性，则称其为是抗原像攻击的。
  - 设*h*码长度为*n*，且Hash函数是等概分布的，则对任意输入*x*产生的*H(x)*恰好为*h*的概率是$1/2^n$。因此穷举攻击对于单向性求解的复杂度为$O(2^n)$。

一、Hash函数的概念

1、Hash函数的定义

③安全性：

- **抗弱碰撞性：**对任何给定的x，找到$y \neq x$且$H(x)=H(y)$的y在计算上是不可行的。
  - 否则，攻击者可以截获报文M及其$H(M)$，并找出另一报文M'使得$H(M')=H(M)$。这样攻击者可用M'去冒充M，而收方不能发现。
  - 抗弱碰撞又称为抗求第二原像。
- **抗强碰撞性：**找到任何满足$H(x)=H(y)$的偶对(x, y)在计算上是不可行的。
- **随机性：**Hash函数的输出具有伪随机性。
  - 应当通过我国国家密码管理局颁布的《随机性测试规范》的测试。
  - 也可参考美国NIST的随机性测试标准。

二、HASH函数的安全性

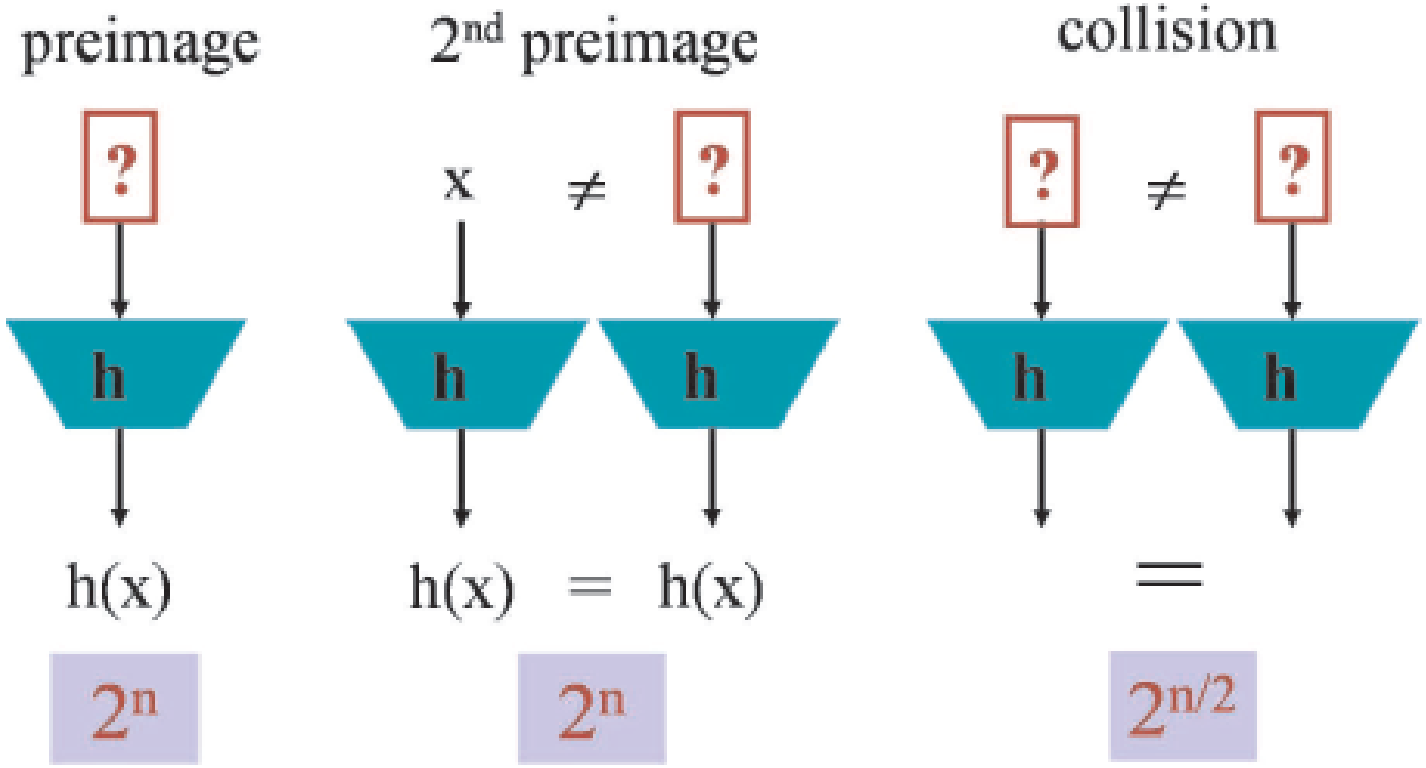
- **单向性**：如果攻击者已经掌握任意随机消息M摘要h=H(M)上的签名S=D(H(M),K_d)。如果hash函数不满足单向性，那么攻击者就能够找到消息x使得H(M)=h=H(x)，那么(x, S)就是伪造消息x的合法签名。
- **抗弱碰撞**：如果对于消息摘要签名，攻击者看到签名者A在x的消息摘要h(x)上的签名S以后，能够寻找到与x不同的x’，使得H (x)=H (x’)，那么（x’, S）就是伪造的合法签名。
- **抗强碰撞**：如果攻击者能够自己选择消息给签名者A签名，并能够找到一对不同的消息x,x’，使得H (x)=H (x’)，那么攻击者先让A对x签名，然后就伪造出x’的合法签名。

二、HASH函数的安全性

- 上述针对数字签名的攻击特点比较

	单向性	抗弱碰撞	抗强碰撞
实现前提条件	截获任意签名	看到某特定消息的签名	说服A对某消息签名
(攻击环境)	易 →		难
计算复杂度	$O(2^n)$	$O(2^n)$	$O(2^{n/2})$
(攻击消耗时间)	难 ←		易

二、HASH函数的安全性



二、HASH函数的安全性

- 三类攻击的复杂度比较

	单向性	抗弱碰撞	抗强碰撞
攻击类型	原像攻击	第二原像攻击	碰撞攻击
q次尝试后的成功概率	$\varepsilon = 1-(1-\frac{1}{N})^q$	$\varepsilon = 1-(1-\frac{1}{N})^q$	$\varepsilon = 1-\prod_{i=1}^{q-1}(\frac{N-i}{N})$
计算复杂度	$O(2^n)$	$O(2^n)$	$O(2^{n/2})$

- 注：假设输出hash码的取值空间为$N=2^n$

HASH函数的安全性——生日攻击

- 假定使用64比特的Hash码,是否安全?
- 如果采用传输加密的散列码和不加密的报文M,对手需要找到M',使得H(M') = H(M),以便使用替代报文来欺骗接收者。
- 一种基于生日悖论的攻击可能做到这一点。

HASH函数的安全性——生日攻击

生日悖论

- 生日问题：一个教室中，最少应有多少学生，才使至少有两人具有相同生日的概率不小于1/2？
- 概率结果与人的直觉是相违背的.
- 实际上只需23人,即任找23人，从中选出两人具有相同生日的成功概率至少为0.5。

HASH函数的安全性——生日攻击

Birthday Paradox:

If there are 23 or more people in a room, the chances are

better than 50% that two of them will have the same birthday !

HASH函数的安全性——生日攻击

- 假设你和其余k个人一起在一个屋子中。那么至少找到一个人同你的生日相同的话，需要k为多大？
- 存在有人同你生日相同的**概率超过1/2**的话，需要k为多大？

HASH函数的安全性——生日攻击

- 假设你和其余k个人一起在一个屋子中。那么至少找到一个人同你的生日相同的话，需要k为多大？
- 存在有人同你生日相同的**概率超过1/2**的话，需要k为多大？
- $$P_k = 1-(1-\frac{1}{365})(1-\frac{1}{365})...(1-\frac{1}{365}) = 1-(\frac{364}{365})^k$$
- 取P>0.5时，得k=253

HASH函数的安全性——生日攻击

- 假设有N个人在一个屋子中，如果任何两个或两个以上人的生日相同的概率超过1/2的话，需要N为多大？

HASH函数的安全性——生日攻击

- 假设有N个人在一个屋子中，如果任何两个或两个以上人的生日相同的概率超过1/2的话，需要N为多大？
- 在N个人中，不存在两个或多个人同一天生日的概率是：

$$\Pr\{E_n\} = P_n = \frac{365 \times 364 \times \ldots \times (365-(n-1))}{365^n}$$

$$= (1-\frac{1}{365})(1-\frac{2}{365})\ldots(1-\frac{n-1}{365})$$

HASH函数的安全性——生日攻击

n	生日不重复的概率	n	生日不重复的概率	n	生日不重复的概率	n	生日不重复的概率	n	生日不重复的概率
1	1.000000	11	0.858859	21	0.556312	31	0.269545	41	0.096848
2	0.997260	12	0.832975	22	0.524305	32	0.246652	42	0.085970
3	0.991796	13	0.805590	23	0.492703	33	0.225028	43	0.076077
4	0.983644	14	0.776897	24	0.461656	34	0.204683	44	0.067115
5	0.972864	15	0.747099	25	0.431300	35	0.185617	45	0.059024
6	0.959538	16	0.716396	26	0.401759	36	0.167818	46	0.051747
7	0.943764	17	0.684992	27	0.373141	37	0.151266	47	0.045226
8	0.925665	18	0.653089	28	0.345539	38	0.135932	48	0.039402
9	0.905376	19	0.620881	29	0.319031	39	0.121780	49	0.034220
10	0.883052	20	0.588562	30	0.293684	40	0.108768	50	0.029626

一、Hash函数的概念

2、Hash函数的类型

根据Hash函数的压缩函数的结构不同，可将其分为以下三种类型。

① 压缩函数迭代型的Hash函数

- 压缩函数由一些简单的非线性函数和线性函数组成。
- 对一个数据分组的处理中用压缩函数多次迭代压缩。
- 如果输入数据很长，将其分成一系列的数据分组，在各个数据分组之间再进行迭代处理。
- 优点是数据处理速度快。
- 目前广泛应用的主要是这种类型的Hash函数。中国商用密码SM3和美国NIST的SHA-3，都是这种类型的Hash函数。

一、Hash函数的概念

2、Hash函数的类型

② 基于对称密码的Hash函数

- 对称密码已经十分成熟，可以利用它设计Hash函数。
- 成功实例
  - 欧洲的Whirlpool算法
  - 俄罗斯的Hash函数标准算法GOST R34.11-94
  - ANSI和ISO的基于分组密码的Hash函数标准
- 安全性和数据处理速度取决于所使用的对称密码。
- 目前有许多强分组密码，都可设计Hash函数。
- 基于序列密码设计Hash函数较少。ZUC的MAC是一个例子。

一、Hash函数的概念

2、Hash函数的类型

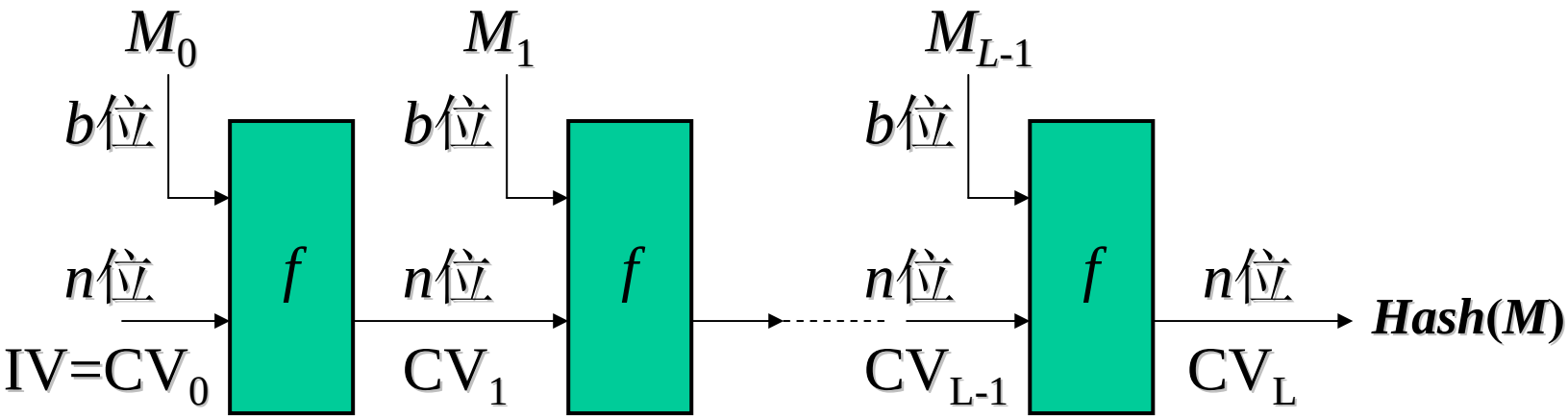
③ 基于数学困难问题的Hash函数

- 已经找到大量的困难问题，NP问题和NPC问题。
- 基于困难问题设计出许多公钥密码，也可以设计Hash函数。
- 具体地：
  - 基于困难问题设计构成压缩函数
  - 对输入数据进行迭代压缩处理
  - 最后产生出Hash码
- 安全性取决于所基于的困难问题。
- 优点：安全性容易确保，长度易调整。
- 缺点：速度慢，工程实现复杂。

一、Hash函数的概念

3、Hash函数的一般模型

- **Merkle提出了用Hash函数处理数据*M*的一般模型。**



***b*位分组，*f*为压缩函数，*L*轮链接迭代，*n*位输出。**

一、Hash函数的概念

3、Hash函数的一般模型

- 分组：将输入M分为L-1个大小为b位的分组：

$$M_0，M_1，M_2，\ldots，M_{L-2}$$

- 填充：若第L-1个分组不足b位，则将其填充为b位。
- 附加：再附加上一个表示输入长度的分组。
- 填充和附加之后，共L个大小为b位的分组。

$$M_0，M_1，M_2，\ldots，M_{L-2}，M_{L-1}$$

- 由于输入中包含长度，所以攻击者必须找出具有相同Hash值且长度相等的两条报文，或者找出两条长度不等但加入报文长度后Hash值相同的报文，从而增加了攻击的难度。

一、Hash函数的概念

3、Hash函数的一般模型

- 压缩函数由一些简单的非线性和线性函数组成。
- 如果输入数据很长，将其分成一系列的数据分组M_0，M_1，M_2，…，M_{L-1}，对数据进行两层迭代压缩处理：
  - 对一个数据分组M_i的处理中用压缩函数多次迭代压缩。
  - 在各个数据分组之间再进行迭代压缩处理。

- 目前大多数Hash函数均采用这种模型，如中国的SM3、美国的SHA-3。

一、Hash函数的概念

4、 Hash函数的作用

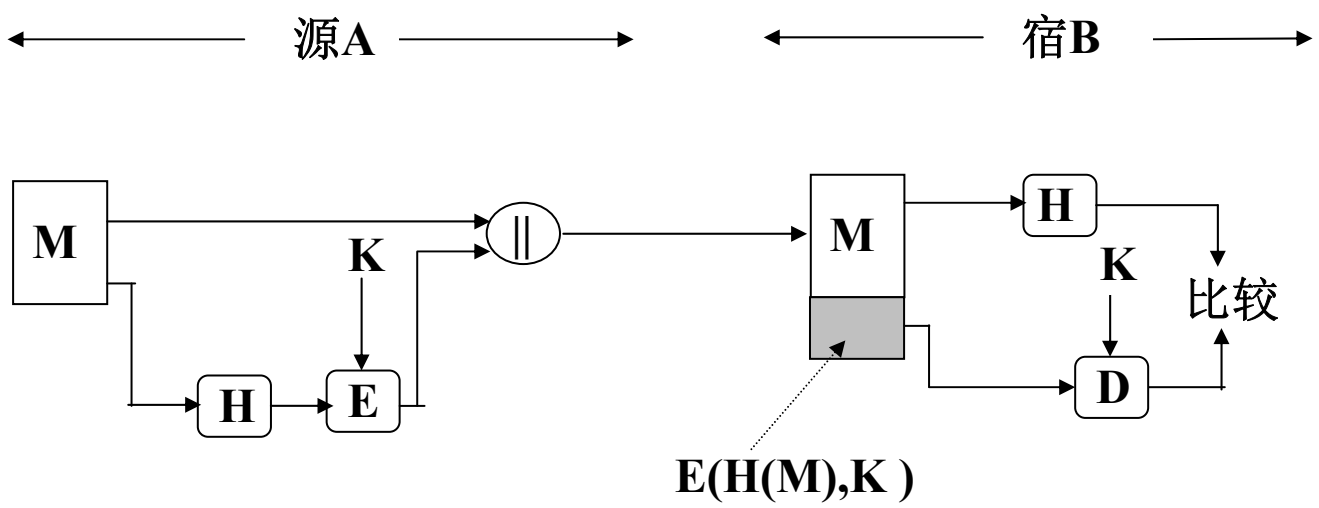
- Hash码也称为数据摘要、数据指纹。
- 具有极强的错误检测能力：
  - 输入有很小的变化，输出将有很大的不同！
  - 检测错误，检测篡改。
- 用Hash码作消息认证码（MAC），可用于认证。
- 用Hash码辅助数字签名：
  - 缩短签名长度
  - 增强签名安全
- Hash函数还可用于伪随机数产生。

一、HASH函数的概念

- 安全Hash函数的应用1：
- 报文认证（方案a）
  - A→B：＜M || E（H（M）,K）＞
  - 发方生成报文M的hash码H(M)并使用传统密码对其加密，将加密后的结果附于消M之后发送给接收方。
  - 由于H(M)受密码保护，所以B通过比较H(M)可认证报文的真实性和完整性。

一、HASH函数的概念

- 安全Hash函数的应用——报文认证
- (1.a)加密**hash**码（使用共享的密钥）
  - 提供认证 ： H(M)受密码保护

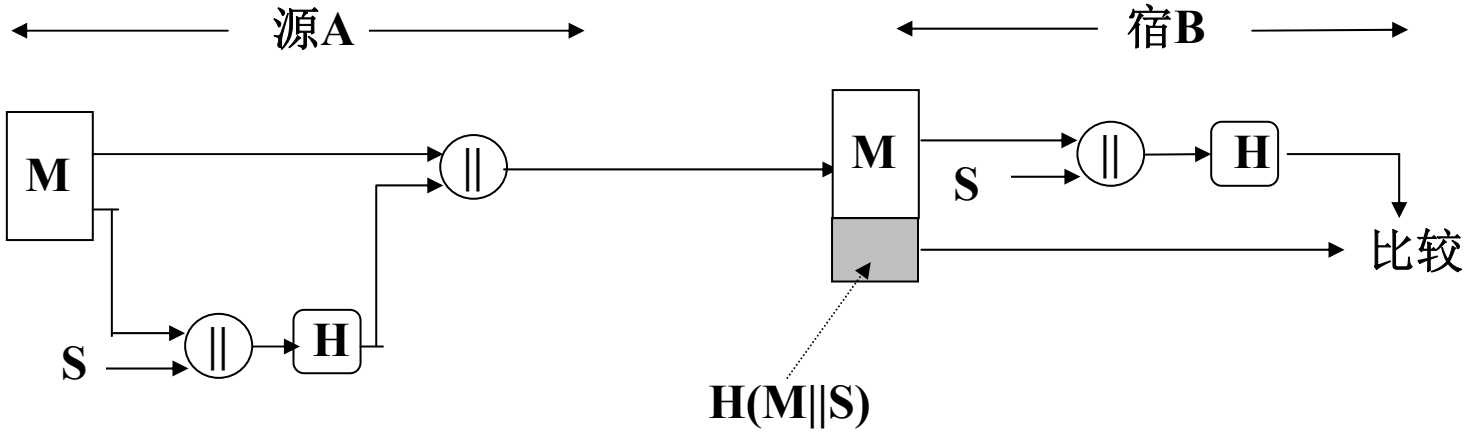


一、HASH函数的概念

- 安全Hash函数的应用1：
- 报文认证（方案b）
  - A→B：＜M || H（M || S）＞
  - 发方生成报文M和秘密值S的Hash码，并将其附于报文M之后发送给接收方。
  - 假定通信双方AB共享公共的秘密值S，B可以通过验证Hash码来认证数据的真实性和完整性。
  - 该方案无需加密，因为秘密值S参与Hash的计算。

一、HASH函数的概念

- 安全Hash函数的应用——报文认证
- (1.b)计算消息和秘密值的**hash**码
  - 提供认证：只有A和B共享秘密值S



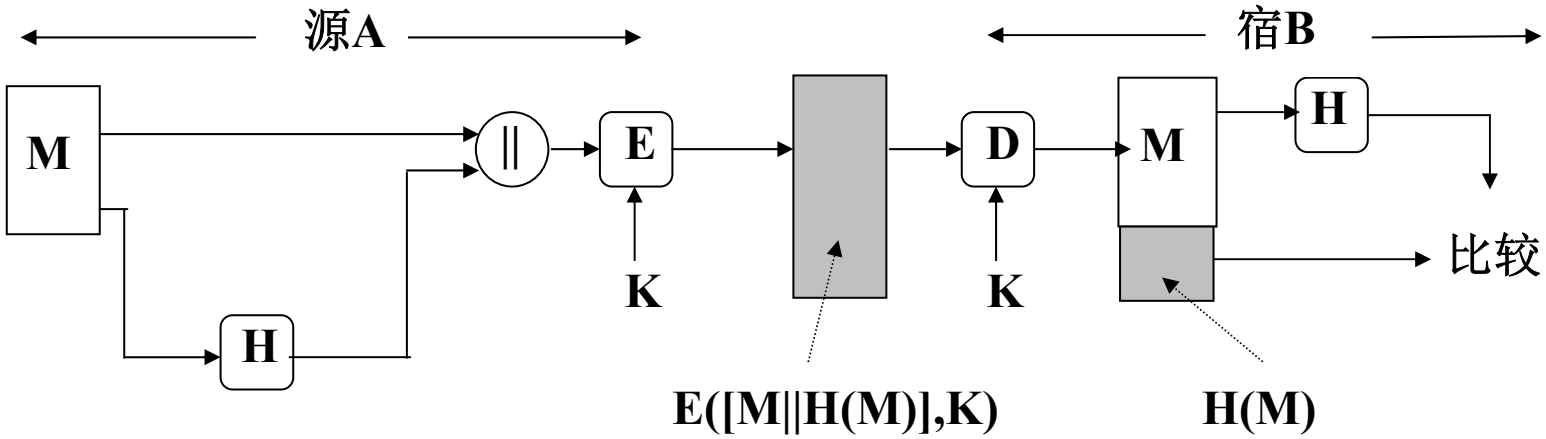
思考：这种方法有什么优点？

一、HASH函数的概念

- 安全Hash函数的应用2：
- 认证和保密（方案a）
  - A→B：＜E（M || H（M）,K）＞
  - 由于只有A和B共享秘密钥，所以B通过比较H(M)可认证报文源和报文的真实性。由于该方法是对整个报文M和Hash码加密，所以也提供了保密性。

一、HASH函数的概念

- 安全Hash函数的应用——认证和保密
- (2.a)加密消息及**hash**码（使用共享的密钥）
  - 提供保密性：只有A和B共享K
  - 提供认证 ： H(M)受密码保护

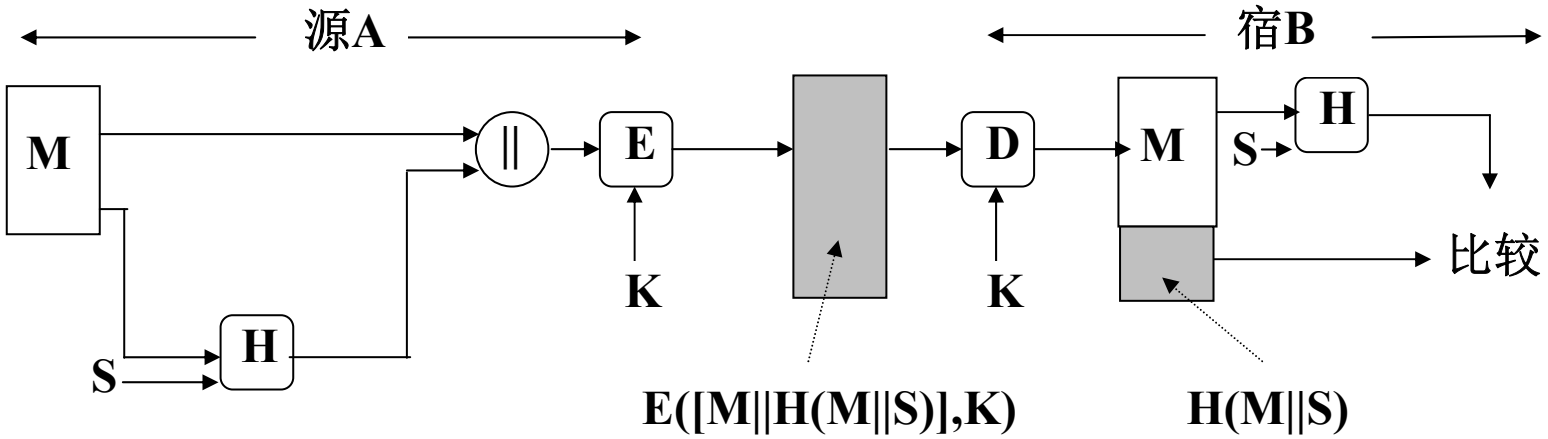


一、HASH函数的概念

- 安全Hash函数的应用2：
- 认证和保密（方案b）
  - A→B：＜E（M || H（M || S ）,K）＞
  - 由于只有A和B共享秘密钥K以及秘密值S，所以B通过比较H(M || S)可认证报文源和报文的真实性。
  - 由于该方法是对整个报文M和Hash码加密，所以也提供了保密性。

一、HASH函数的概念

- 安全Hash函数的应用——认证和保密
- (2.b)加密(1.b)的结果（使用共享的密钥）
  - 提供保密性：只有A和B共享K
  - 提供认证 ：只有A和B共享秘密值S

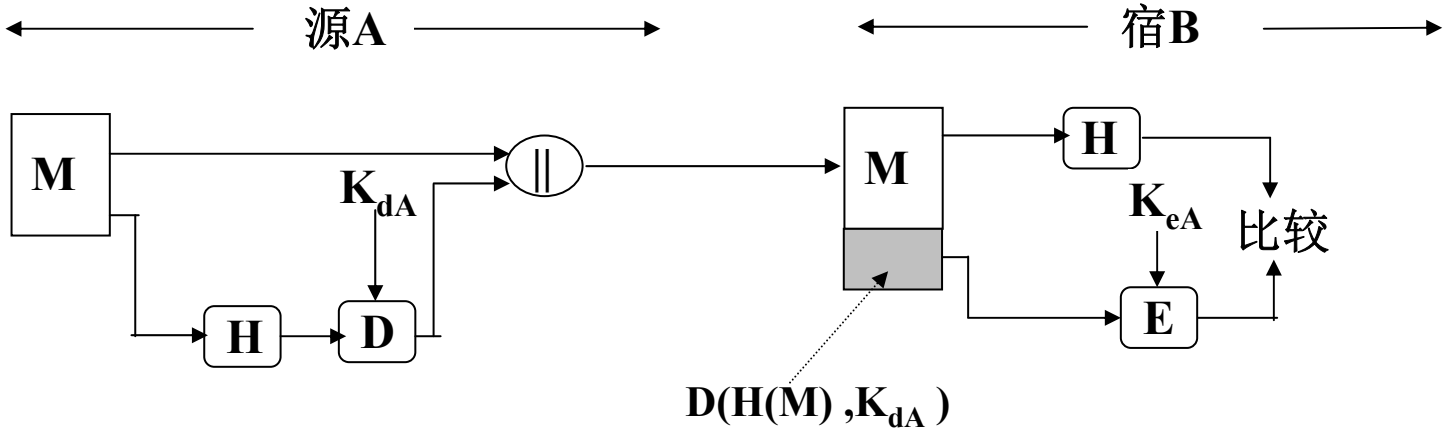


一、HASH函数的概念

- 安全Hash函数的应用3：
- 认证和数字签名
  - A→B：＜M || D（H（M）,K_{dA}）＞
  - 发方使用公钥密码用其私钥K_{dA}对消息M的hash码签名，并将其附于报文M之后发送给收方。
  - B可以验证Hash值来认证报文的真实性，因此该方法可提供认证；
  - 由于只有发方可以进行签名，所以也提供了数字签名

一、HASH函数的概念

- 安全Hash函数的应用——认证和数字签名
- (3)加密**hash**码（使用发送方私钥）
  - 提供数字签名：只有A能产生D(H(M) ,K_{dA})
  - 提供认证： H(M)受密码保护

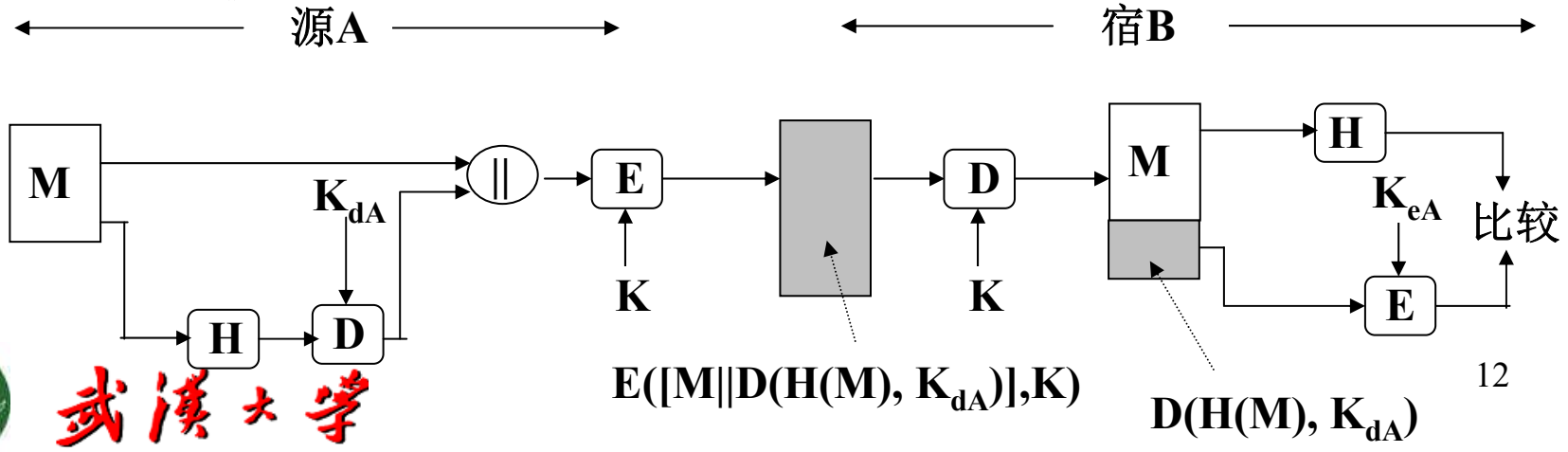


一、HASH函数的概念

- 安全Hash函数的应用4：
- 认证、数字签名和保密
  - A→B：<E（ M || D（H（M）, K_{dA}）, K>
  - 发方使用公钥密码用其私钥K_{dA}对消息M的hash码签名，并将其附于报文M之后用传统密码对报文M和签名进行加密，发送给收方。
  - 由前述讨论可知，该方法可提供认证、数字签名和保密性。

一、HASH函数的概念

- 安全Hash函数的应用——认证、数字签名和保密
- (4)加密(3)的结果（使用共享的密钥）
  - 提供数字签名：只有A能产生D(H(M) ,K_{dA})
  - 提供认证： H(M)受密码保护
  - 提供保密性：只有A和B共享K



二、中国商用密码Hash函数SM3

1. 概况

- SM3是我国密码管理局颁布的商用密码Hash函数
- 用途广泛
  - 商用密码应用中的辅助数字签名和验证
  - 消息认证码的生成与验证
  - 随机数的生成
- SM3在结构上属于基本压缩函数迭代型的Hash函数。

二、中国商用密码Hash函数SM3

2. SM3算法

- 输入数据长度为l比特，$1 \leqslant l \leqslant 2^{64}-1$
- 输出Hash值的长度为256比特。

(1) 常量与函数

SM3密码Hash函数使用以下常数与函数：

① 常量

初始值IV =7380166f 4914b2b9 172442d7 da8a0600
a96f30bc 163138aa e38dee4d b0fb0e4e

$$\text{常量}T=\begin{cases}79cc4519 & 0 \leqslant j \leqslant 15\\ 7a879d8a & 16 \leqslant j \leqslant 63\end{cases}$$

二、中国商用密码Hash函数SM3

② 函数

- 布尔函数：

$$FF_j(X,Y,Z)=\begin{cases}X\oplus Y\oplus Z & 0\leqslant j\leqslant 15\\ (X\wedge Y)\vee(X\wedge Z)\vee(Y\wedge Z) & 16\leqslant j\leqslant 63\end{cases}$$

$$GG_j(X,Y,Z)=\begin{cases}X\oplus Y\oplus Z & 0\leqslant j\leqslant 15\\ (X\wedge Y)\vee(\neg X\wedge Z) & 16\leqslant j\leqslant 63\end{cases}$$

式中X、Y、Z 为32位字。

- 置换函数：

$P_0(X)=X\oplus(X<<<9)\oplus(X<<<17)$

$P_1(X)=X\oplus(X<<<15)\oplus(X<<<23)$

式中X为32位字，符号 $a<<<n$表示把a循环左移n位。

二、中国商用密码Hash函数SM3

(2) 算法描述

SM3 对数据进行填充和扩展，然后迭代压缩生成Hash值。

① 填充

- 对数据填充的目的是使填充后的数据长度为512的整数倍。因为迭代压缩是对512位数据块进行的，如果数据的长度不是512的整数倍，最后一块数据将是短块，这将无法处理。
- 设消息m长度为l 比特。首先将比特“1”添加到m的末尾，再添加 k个“0”，其中，k是满足下式的最小非负整数。

$$l + 1 + k = 448 \bmod 512$$

- 然后再添加一个64位比特串，该比特串是长度l 的二进制表示。填充后的消息 m 的比特长度一定为512的倍数。

二、中国商用密码Hash函数SM3

(2) 算法描述

① 填充

- 举例：对消息01100001 01100010 01100011，其长度*l*=24，经填充得到比特串：

*l*的二进制表示64比特

423比特0

01100001 01100010 01100011 100 …… 00 00 … 011000

二、中国商用密码Hash函数SM3

② 迭代压缩处理

- 将填充后的消息m′按512比特分组：$m' = B^{(0)}B^{(1)} \ldots B^{(n-1)}$

  其中：$n = (l+k+65)/512$。
- 对m'按下列方式迭代压缩：外层迭代

  ① FOR I = 0 TO n-1

  ② $V^{(I+1)} = CF(V^{(I)}, B^{(I)})$

  ③ ENDFOR

  其中CF是压缩函数，$V^{(0)}$为256比特初始值IV，$B^{(I)}$为填充后的消息分组，
- 迭代压缩的结果为$V^{(n)}$，它为消息m的Hash值。

二、中国商用密码Hash函数SM3

③ 消息扩展

- 对一个消息分组$B^{(i)}$迭代压缩之前，首先进行消息扩展
  - 将16个字的消息分组$B^{(i)}$扩展生成如下的132个字，供压缩函数CF使用

    $W_0, W_1, \ldots, W_{67}, W'_0, W'_1, \ldots, W'_{63}$
  - 消息扩展把原消息位打乱，隐蔽原消息位之间的关联，增强了安全性
- 消息扩展的步骤如下：

  ① 将消息分组$B^{(i)}$划分为16个字$W_0, W_1, \ldots, W_{15}$。

  ② **FOR j=16 TO 67**

  $W_j \leftarrow P_1(W_{j-16} \oplus W_{j-9} \oplus (W_{j-3} <<< 15)) \oplus (W_{j-13} <<< 7) \oplus W_{j-6}$

  ENDFOR

  ③ **FOR j=0 TO 63**

  $W_j' = W_j \oplus W_{j+4}$

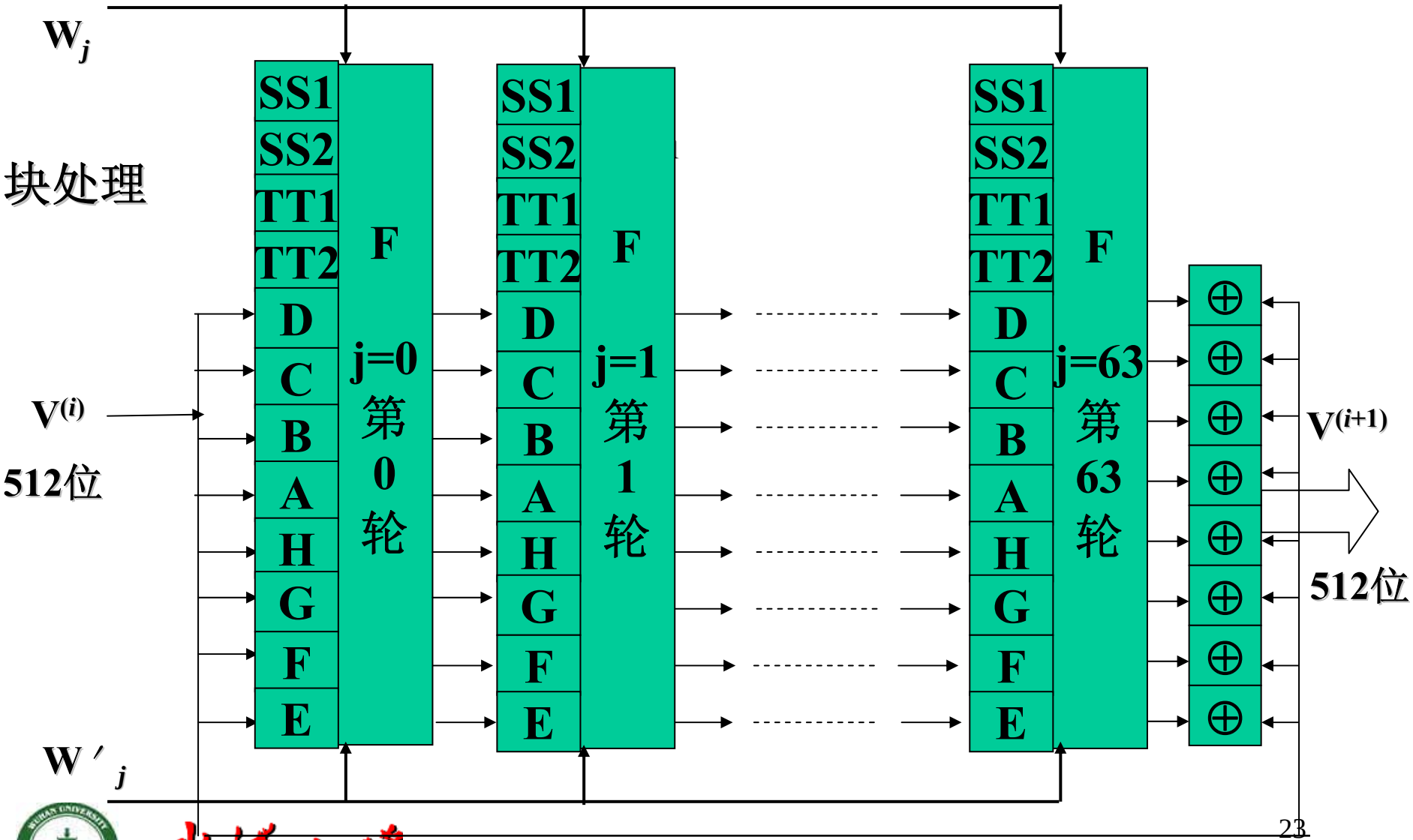
  ENDFOR

二、中国商用密码Hash函数SM3

④ 压缩函数

- 压缩函数是SM3的核心
- 令A,B,C,D,E,F,G,H为字寄存器，SS1,SS2,TT1,TT2为中间变量。
- 压缩函数：$V^{(i+1)} = CF(V^{(i)}, B^{(i)}), 0 \leqslant i \leqslant n-1$。
- 压缩函数CF的压缩处理：内层迭代
  - ① FOR j=0 TO 63
  - ② CF= F(SS1，SS2，TT1，TT2，A，B，C，D，E，F，G，W_j，W'_j)　　/*基本压缩函数 /
  - ③ ENDFOR

●压缩函数CF


W_j
块处理
SS1
SS2
TT1
TT2
D
C
B
A
H
G
F
E
F
j=0
第
0
轮
F
j=1
第
1
轮
F
j=63
第
63
轮
$V^{(i)}$
512位
$V^{(i+1)}$
512位
W'_j

武漢大学

二、中国商用密码Hash函数SM3

⑤ 基本压缩函数F：

$SS1 \leftarrow ((A <<< 12) + E + (T_j <<< j)) <<< 7$;

$SS2 \leftarrow SS1 \oplus (A <<< 12)$;

$TT1 \leftarrow FF_j(A, B, C) + D + SS2 + W_j'$;

$TT2 \leftarrow GG_j(E, F, G) + H + SS1 + W_j$;

$D \leftarrow C$;

$C \leftarrow B <<< 9$;

$B \leftarrow A$;

$A \leftarrow TT1$;

$H \leftarrow G$

$G \leftarrow F <<< 19$

$F \leftarrow E$

$E \leftarrow P_0(TT2)$

说明：

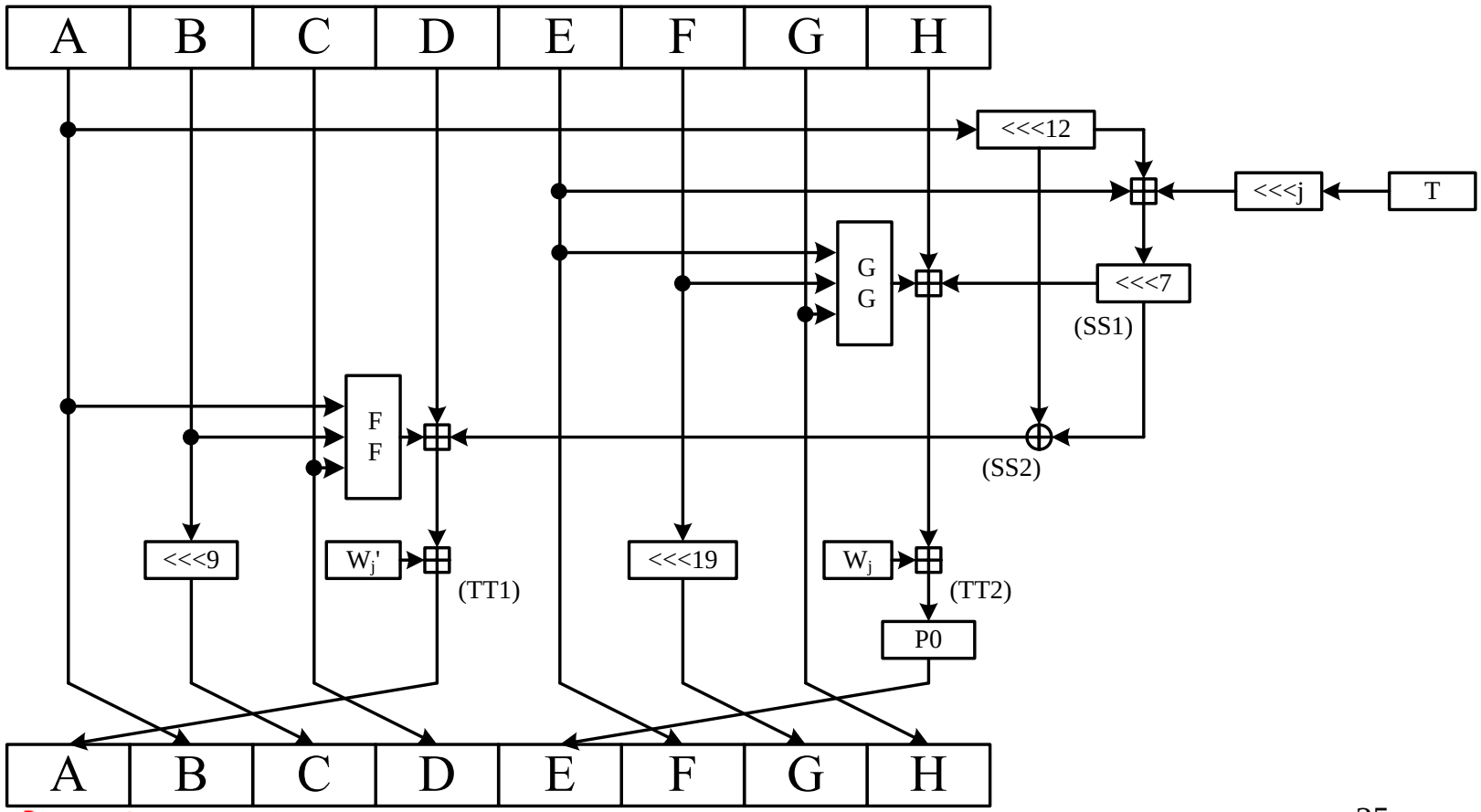
●A,B,C,D,E,F,G,H为字寄存器，SS1,SS2,TT1,TT2为中间变量

●+运算为 mod 2^{32} 算术加运算

●字的存储为大端(big-endian)格式。即，左边为高有效位，右边为低有效位。数的高位字节放在存储器的低地址，低位字节放在存储器的高地址。

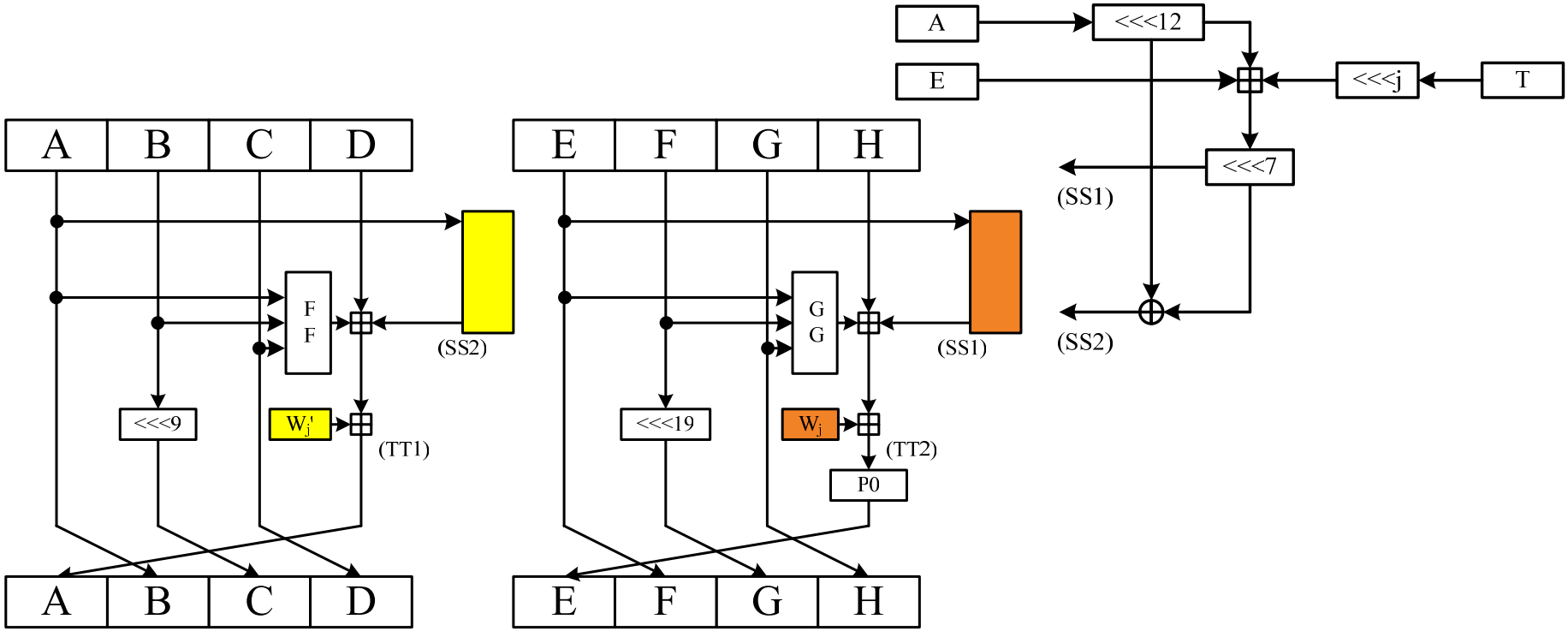
二、中国商用密码Hash函数SMS3

●基本压缩函数



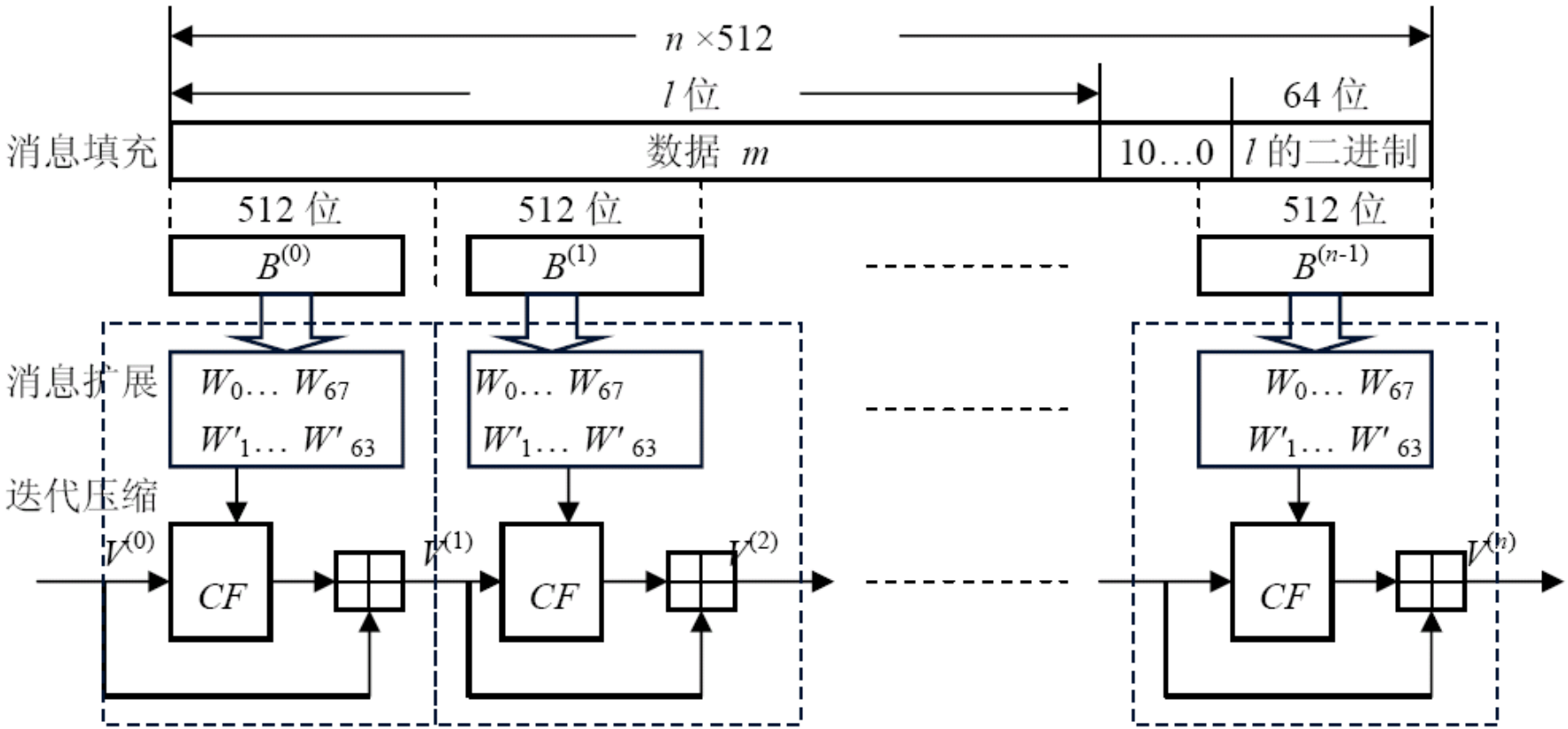
二、中国商用密码Hash函数SM3

- 基本压缩函数分解结构



二、中国商用密码Hash函数SM3

⑥ SM3工作全过程



武漢大学

二、中国商用密码Hash函数SM3

⑦ 压缩函数的作用

- 压缩函数是SM3安全的关键
- 第一个作用是数据压缩。SM3的压缩函数*CF*把每一个512位的消息分组$B^{(i)}$压缩成256位。经过各数据分组之间的迭代处理后把*l*位的消息压缩成256位的Hash值。
- 第二个作用是提供安全性。 在SM3的压缩函数*CF*中，布尔函数$FF_j(X,Y,Z)$和$GG_j(X,Y,Z)$是非线性函数，经过循环迭代后提供混淆作用。置换函数$P_0(X)$和$P_1(X)$是线性函数，经过循环迭代后提供扩散作用。加上压缩函数*CF*中的其它运算的共同作用，压缩函数*CF*具有很高的安全性，从而确保SM3具有很高的安全性。

二、中国商用密码Hash函数SM3

(3) 安全性

- 专业机构设计，经过充分测试和论证
- 安全性可满足上述应用的安全需求
- 学者已开展对SM3的安全分析（如缩减轮的分析），尚未发现本质的缺陷

三、美国NIST的Hash函数

1、 SHA系列Hash函数

- SHA系列标准Hash函数是由美国标准与技术研究所(NIST)组织制定的。
- 1993年公布了SHA-0 (FIPS PUB 180)，后发现不安全。
- 1995年又公布了SHA-1（FIPS PUB 180-1）。
- 2002年又公布了SHA-2（FIPS PUB 180-2）：
  - SHA-256
  - SHA-384
  - SHA-512
- 2005年中国王小云给出一种攻击SHA-1的方法，用2^{69}操作找到一个强碰撞，以前认为是2^{80}。

三、美国NIST的Hash函数

1、 SHA系列Hash函数

SHA参数比较

	SHA-1	SHA-256	SHA-384	SHA-512
Hash码长度	160	256	384	512
消息长度	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
分组长度	512	512	1024	1024
字长度	32	32	64	64
迭代步骤数	80	64	80	80
安全性	80	128	192	256

注:1、所有的长度以比特为单位。

2、安全性是指对输出长度为n比特hash函数的生日攻击产生碰撞的工作量大约为$2^{n/2}$

三、美国NIST的Hash函数

1、 SHA系列Hash函数

- NIST于2007年公开征集SHA-3，要求：

① 能够直接替代SHA-2。这要求SHA-3必须也能够产生224，256，384，512比特的Hash码。

② 必须保持SHA-2的在线处理能力。这要求SHA-3必须能处理小的数据块（如512或1024比特）。

③ 安全性：能够抵抗原像和碰撞攻击的能力，能够抵抗已有的或潜在的对于SHA-2的攻击。

④ 效率：可在各种硬件平台上的实现，且是高效的和存储节省的。

⑤ 灵活性：可设置可选参数以提供安全性与效率折中的选择，便于并行计算等。

三、美国NIST的Hash函数

1、 SHA系列Hash函数

- NIST共征集到64个应征算法：
  - 2009年7月24日宣布，其中14个算法通过第一轮评审进入第二轮。
  - 2010年12月9日宣布，其中5个算法通过第二轮评审进入第三轮。
  - 2012年10月2日NIST公布了最终的优胜者。它就是由意法半导体公司的Guido Bertoai、Jean Daemen、Gilles Van Assche与恩智半导体公司的Michaël Peeters联合设计的Keccak算法。
  - SHA-3成为NIST的新Hash函数标准算法（FIPS PUB 180-5）。尚未公布算法细节。

三、美国NIST的Hash函数

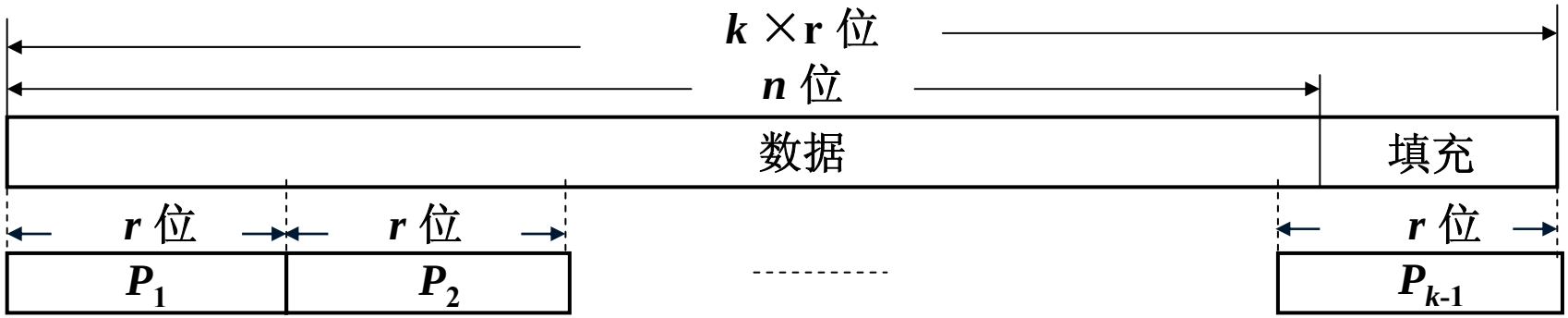
2、 SHA-3 Hash函数概述

- SHA-3的结构仍属于Merkle提出的一般结构。
- 最大的创新点是采用了一种被称为海绵结构的新的迭代结构。海绵结构又称为海绵函数。
- 在海绵函数中，输入数据被分为固定长度的数据分组。每个分组逐次作为迭代的输入，同时上轮迭代的输出也反馈至下轮的迭代中，最终产生输出Hash码。
- 海绵函数允许输入长度和输出长度都可变，具有灵活的性。
- 海绵函数能够用于设计Hash函数（固定输出长度）、伪随机数发生器，以及其他密码函数。

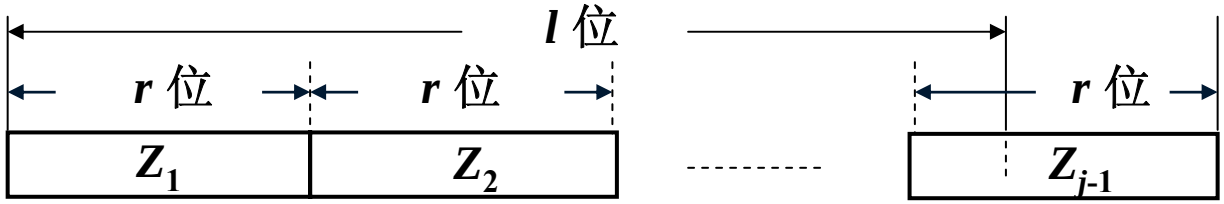
三、美国NIST的Hash函数

2、 SHA-3 Hash函数概述

- 输入结构



- 输出结构



三、美国NIST的Hash函数

2、 SHA-3 Hash函数概述

(1) 填充

- 首先进行数据填充。填充后它的长度为r的整数倍。
- 设输入数据的长度为n位，经填充后它被分为k个分组长度为r位的数据分组P_0，P_1，…，P_{k-1}。
- 对任意消息都需要进行填充。即使输入数据的长度是r的整数倍，也需要填充，此时将填充一个r位的完整块。

● 填充方法

- 简单填充：用一个1后面跟若干个0进行填充，0的个数是使得总长度为分组长度r整倍数的最小值。
- 多重位速率填充：用一个1后面跟若干个0，再跟一个1进行填充，0的个数是使得总长度为分组长度r整倍数的最小值。

三、美国NIST的Hash函数

2、 SHA-3 Hash函数概述

(2) 输出 数据处理完毕，输出Hash码。

- 在对所有的数据分组处理完毕后，海绵函数产生一组输出块 Z_0，Z_1，…，Z_{j-1}。产生的输出数据块的个数由需要的输出位数 l 所决定。
- 如果需要输出 l 位的Hash码，则产生 j 个输出块，

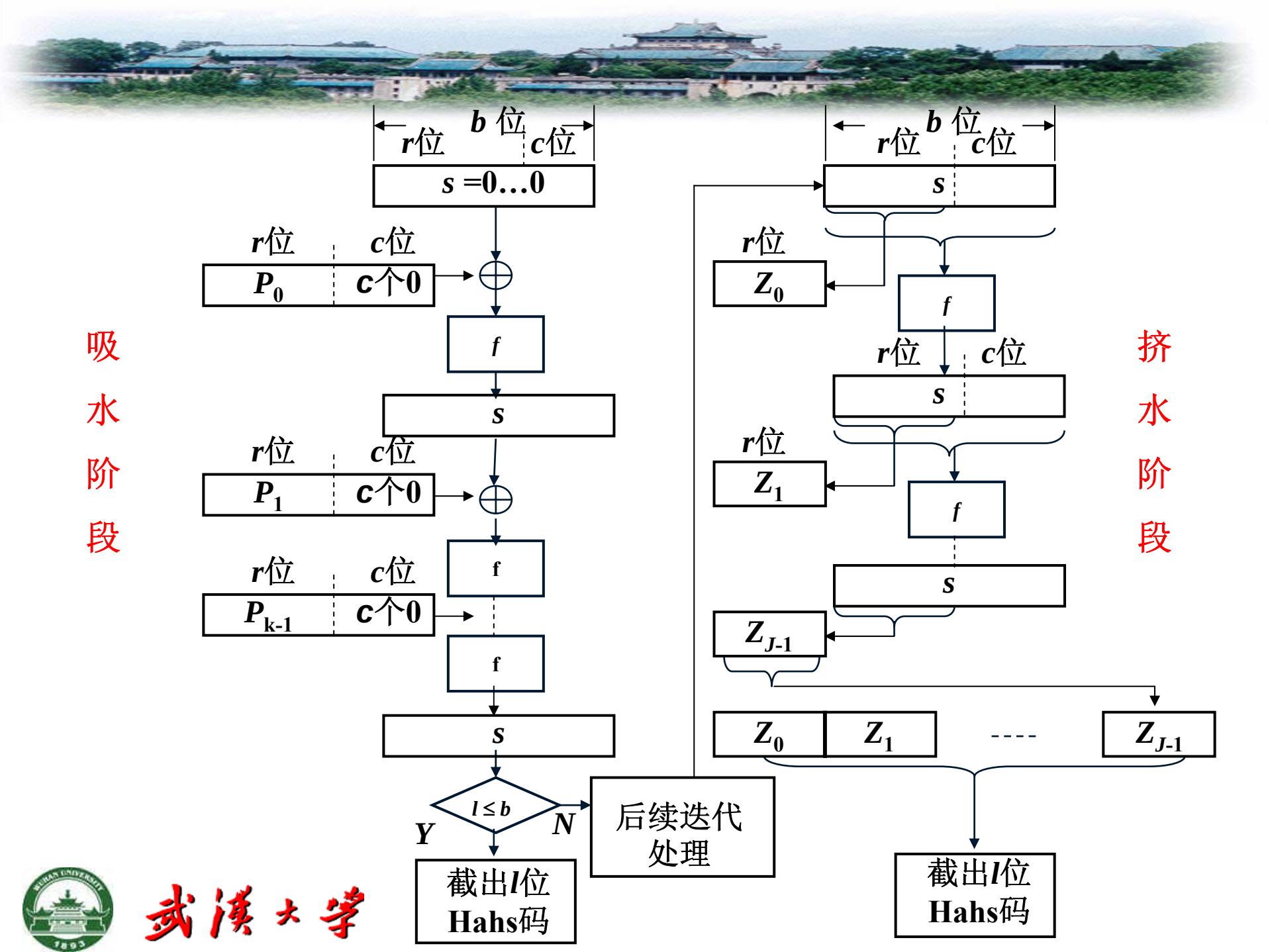
  其中$(j-1)\times r < l < j\times r$

三、美国NIST的Hash函数

2、 SHA-3 Hash函数概述

(3) 迭代处理

- 迭代处理是Hash函数的核心，海绵结构也采用迭代处理。


吸水阶段
挤水阶段
r位
b 位
c位
s =0…0
P0
c个0
f
s
P1
Pk-1
Z0
Z1
ZJ-1
l ≤ b
Y
N
后续迭代处理
截出l位Hahs码
截出l位Hahs码
武漢大学

三、美国NIST的Hash函数

2、 SHA-3 Hash函数概述

(3) 迭代处理

- 迭代处理的核心是压缩函数*f*，它决定着Hash函数的安全和效率。
- 在迭代处理过程中，用压缩函数*f*对数据分组及状态变量进行迭代处理。
- 状态变量s的长度为$b = r+c$位，其初值置为全0，其取值在每轮迭代中更新。数值*r*称为位速率，它是输入数据分组的长度。
  - 位速率*r*越大，海绵结构处理数据的速度就越快。
  - 数值*c*称为容量。它是度量海绵结构能够达到的复杂程度和安全度。
- 在应用中可以通过降低速率来提高安全性。例如，增大容量*c*，减小位速率*r*来提高安全性，反之亦然。默认值是c =1024位，r =576位，于是b =1600位。

三、美国NIST的Hash函数

2、 SHA-3 Hash函数概述

海绵结构包括两个阶段，吸水阶段和挤水阶段。

① 吸水阶段

- 每轮的迭代处理，对数据分组填充c个0，使数据块的长度从r位扩展为b位，$b=r+c$；然后将扩展后的数据分组P和状态变量s（初值为0）进行异或得到b位的结果，并作为函数f的输入。函数f的输出作为下一轮迭代中的状态变量s。
- 如果需要的Hash码的长度 $l \leq b$，那么在吸水阶段完成后，返回s的前l位作为Hash码。否则，海绵结构进入挤水阶段。
- 每一轮迭代都要给数据分组填充c个0，使数据块长度变成b位。这一过程很像海绵吸水，这里的“水”就是填充的c个0。

三、美国NIST的Hash函数

2、 SHA-3 Hash函数概述

② 挤水阶段

- 把s的前r位被保留作为输出分组Z_0，然后在每轮迭代中通过重复执行f函数来更新s的值，s的前r位被依次保留作为输出分组Z_i，并与前面已生成的各分组连接起来。
- 共进行（j-1）次迭代，直到满足$(j-1)\times r < l \leq j \times r$。得到$Z= Z_0 \| Z_1 \| \ldots \| Z_j-1 \|$。最后，输出Z的前$l$位作为Hash码。
- 每一轮迭代都要从长度为b的状态变量s中取出r位的分组，并丢弃其余的c位分组。这一过程很像海绵挤水，这里的“水”就是丢弃的c位分组。

武汉大学

三、美国NIST的Hash函数

2、 SHA-3 Hash函数概述

SHA-3参数表

Hash码长度	224	256	384	512
输入数据长度	没有限制	没有限制	没有限制	没有限制
数据分组长度（位速率r）	1152	1088	832	576
字长度	64	64	64	64
迭代圈数	24	24	24	24
容量c	448	512	768	1024
抗强碰撞攻击强度	2^{112}	2^{128}	2^{192}	2^{256}
抗原像和第二原像攻击强度	2^{224}	2^{256}	2^{384}	2^{512}

三、美国NIST的Hash函数

2、 SHA-3 Hash函数概述

- 安全性
  - 可以抵御对Hash函数的现有攻击。
  - 到目前为止，没有发现它有严重的安全弱点。
- 灵活性
  - 可选参数配置，能够适应Hash函数的各种应用。
- 高效性
  - 设计简单，软硬件实现方便。在效率方面，它是高效的。
- 尚未广泛应用，需要经过实践检验！

四 多变量Hash函数的构造

4.1、多变量Hash函数(MPH)的结构

- **Hash函数的设计方法：**
  - 采用大量逻辑运算直接构造。如MD5、SHA-1等
  - 基于分组密码设计（一般为MDC）。
  - 基于难解问题构造。
- **MQ问题与Hash函数**

定理 5.1.1　对于给定的任意值 $\delta = (\delta_1, \cdots, \delta_n)$, 可以在多项式时间 $O(mn^2)$ 内, 寻找到一对变量 $x = (x_1, \cdots, x_n)$, $y = (y_1, \cdots, y_m)$ 满足 $y - x = \delta$, 使得 $P(x) = P(y)$.

$$p_i(x_1, \cdots, x_n) = \sum_{1 \le j \le k \le n} a_{i,j,k} x_j x_k + \sum_{1 \le j \le n} b_{i,j} x_j + c_i$$

- 结论：MQ问题不能直接用于设计Hash函数

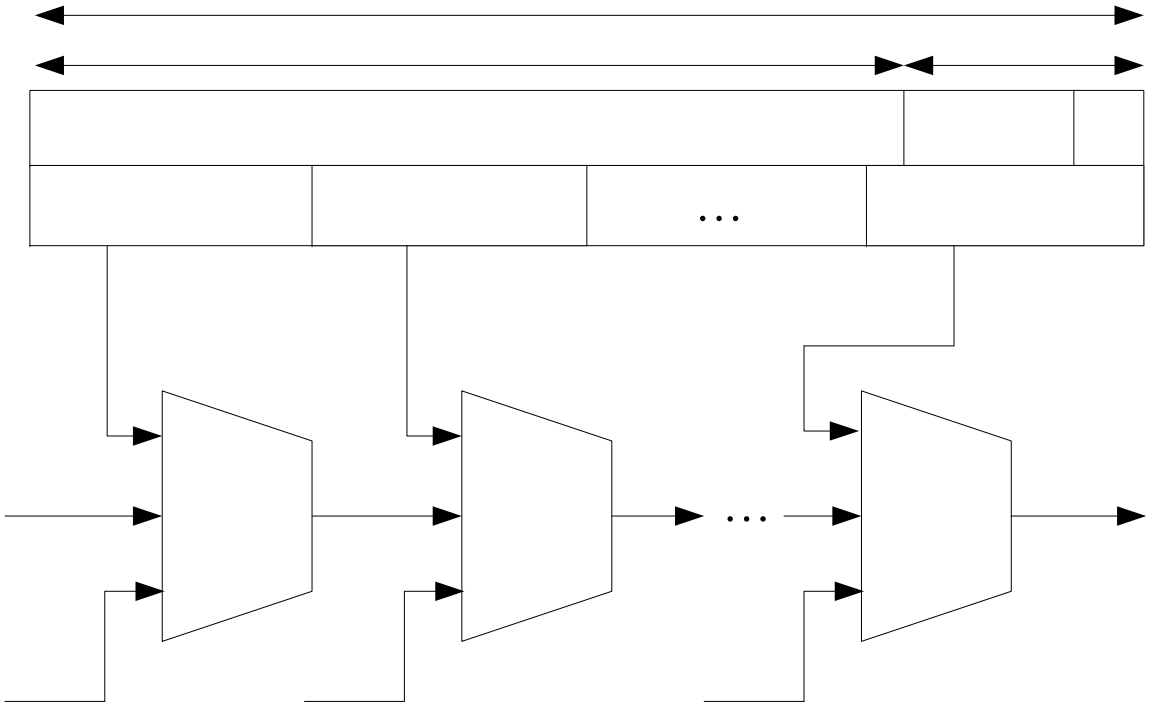
四 多变量Hash函数的构造

4.1、多变量Hash函数(MPH)的结构

压缩函数

$$\begin{cases} CF : F_q^n \mapsto F_q^m,\ CF(x) = (f_1(x_1, \cdots, x_n), \cdots, f_m(x_1, \cdots, x_n)) \\ f_i(x) = \sum_{1 \le j \le k \le t \le n} a_{i,j,k,t} x_j x_k x_t + \sum_{1 \le j \le k = 1} b_{i,j,k} x_j x_k + \sum_{1 \le j \le n} c_{i,j} x_j + d_i \end{cases}$$

HAIFA 结构



算法 5.1: MPH 算法

Input: 任意报文 M, MPH 的基本域 $GF(q)$ 参数 $q = 2^s$, $s \in Z^+$.

Output: h 比特 Hash 值 $H(M)$.

1 报文填充. 填充方式与 SHA 系列类似. 填充后的报文分组为 $M_1, \cdots, M_L$, 其中 M_i 的长度为 h 比特, $h = ns/2$. 再将每个分组 M_i 编码为一个 $n/2$ 维向量;

2 初始化 $n/2$ 维向量 IV, 盐变量 $salt$ 和计数器 $counter$ 向量的初值均为 0;

3 $CV_0 = IV$;

4 **for** $i = 1; i \leq L; i++$ **do**

5 $\quad CV_i = CF((CV_{i-1} \wedge salt_{i-1})||(M_i \wedge counter_{i-1}))$;

6 $\quad$ **for** $j = 1; j \leq n/2; j++$ **do**

7 $\quad\quad$ **if** $j \leq n/4$ **then**

8 $\quad\quad\quad salt_i[j] = CV_{i-1}[j] \wedge CV_i[n/2 + 1 - j]$;

9 $\quad\quad\quad$ **else**

10 $\quad\quad\quad\quad salt_i[j] = CV_{i-1}[j] \wedge CV_i[j - n/4]$;

11 $\quad\quad\quad$ **end**

12 $\quad\quad$ **end**

13 $\quad\quad counter_i[n/2 + 1 - j] = (i >> 4(j-1))\&0\text{x}0\text{F}$;

14 $\quad$ **end**

15 **end**

16 **Return** 返回 h 比特位 Hash 值 $H(M) = encodebits(CV_L)$.

四 多变量Hash函数的构造

4.2、多变量Hash函数(MPH)的安全性分析

- **MPH算法的基本安全性：**
  - 原像攻击。
  - 第二原像攻击。
  - 碰撞攻击。

MQ问题

$$f_i(x) = \sum_{1\le j\le k\le t\le n} a_{i,j,k,t}x_jx_kx_t + \sum_{1\le j\le k=1} b_{i,j,k}x_jx_k + \sum_{1\le j\le n} c_{i,j}x_j + d_i$$

差分函数：

$$DCF(X) = CF(X+\Delta X) - CF(X) = 0$$

$$DCF(X):\ df_i(x_1,\cdots,x_n) = \sum_{1\le j\le k\le n} a_{i,j,k}x_jx_k + \sum_{1\le j\le n} b_{i,j}x_j + c_i$$

4.3、 MPH算法的优化设计

❖ **MPH算法的稀疏加速：**

➢ $$f_i(x)=\underbrace{\sum_{1\le j\le k\le t\le n} a_{i,j,k,t}x_jx_kx_t}_{\text{稀疏部分，稠密度 }\rho_3}+\underbrace{\sum_{1\le j\le k=1} b_{i,j,k}x_jx_k+\sum_{1\le j\le n} c_{i,j}x_j+d_i}_{\text{全部保留}}$$

➢ 差分函数：

$$DF(X):\ df_i(x_1,\cdots,x_n)=\underbrace{\sum_{1\le j\le k\le n} a_{i,j,k}x_jx_k}_{\text{稠密度 }\rho_2}+\sum_{1\le j\le n} b_{i,j}x_j+c_i$$

❖ 稠密度ρ_2与ρ_3之间的关系如下图：

4.3、 MPH算法的优化设计

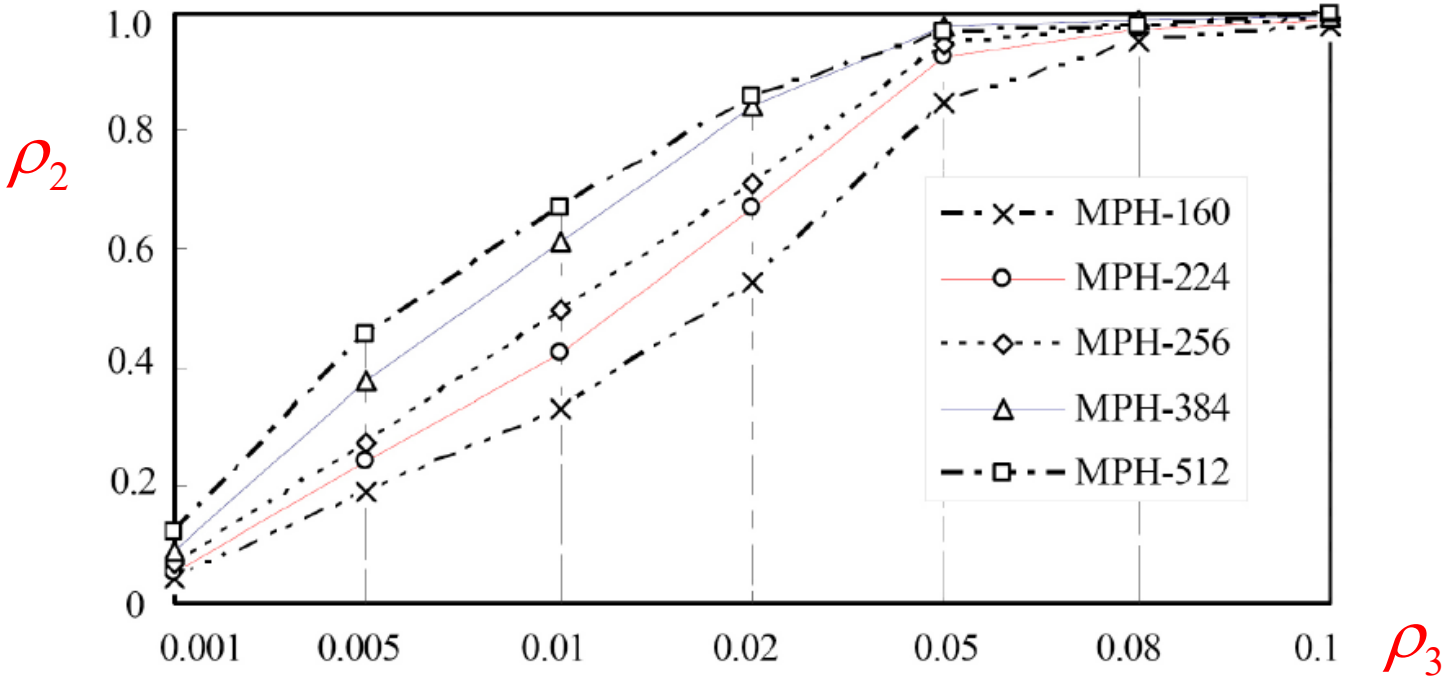


图 5.2: $CF(x)$ 的稠密度 ρ_3 与 $DCF(x)$ 的稠密度 ρ_2 之间的关系

4.3.1、 MPH算法的存储空间优化设计

$$f_i(x) = \underbrace{\sum_{1\le j\le k\le t\le n} a_{i,j,k,t}x_jx_kx_t}_{\text{稀疏加速部分}} + \underbrace{\sum_{1\le j\le k=1} b_{i,j,k}x_jx_k + \sum_{1\le j\le n} c_{i,j}x_j + d_i}_{\text{全部保留}}$$

❖ 例如 $ax_ix_jx_k \Longleftrightarrow (a,i,j,k)$

$$(i,j,k) \Rightarrow R \in [0, n(n+1)(n+2)/6)$$

❖ 后面的多项式由第一个多项式结合一组随机因子生成，因此只需存储第一个多项式及一组随机因子即可。

5.3.1、 MPH算法的存储空间优化设计

算法 5.2: 压缩函数 $CF(x)$ 的生成算法

Input: 有限域 $GF(q)$ 的参数 q, Hash 值长度 h 及三次项项数 $Terms$.

Output: 压缩函数 $CF(x)$.

验证参数 $q, h, Terms$ 是否满足 $q = 2^s$, $s|h$ 以及 $Terms \in [1, N]$. 这里 $s, h \in Z^+$, $N = n(n+1)(n+2)/6$, $s, h \in Z^+$, $n = 2h/s$;

生成 $n/2-1$ 个随机因子 $RF[i] \in (1, N)$, 且满足 $GCD(RF[i], N) = 1$, $RF[i]\%GCD(RF[i]\%q, q) = 1$ 不等于零以及数组 $RF, RF\%q$ 的中没有重复元素;

for $j = 1; j \leq Terms; j++$ **do**

  $Term3[1][j] = get\_random()\%N$;

end

for $j = 1; j \leq (n+1)(n+2)/2; j++$ **do**

  $Term210[1][j] = get\_random()\%q$;

end

for $i = 2; i \leq n/2; i++$

  for $j = 1; j \leq Terms; j++$ **do**

    $Term3[i][j] = (Terms[j] \times RF[j])\%N$;

  end

  for $j = 1; j \leq (n+1)(n+2)/2; j++$ **do**

    $Term210[i][j] = (Term210[1][(n+1)(n+2)/2 - j + 1] \times RF[j])\%q$;

  end

end

Return: $CF(x) = \{Term3, Term210\}$.

算法 5.3: 压缩函数 $CF(x)$ 三次项的解码算法

Input: 参数 $R \in [0, N)$.

Output: 三次项 $x_i x_j x_k$ 的分量 $\{i, j, k\}$.

定义函数 $N[y] = y(y+1)(y+2)/6$; $t = n-1$;

while $N[n] - N[t] < R$ **do**

  $i = n - t - 1$; $M = N[n] - N[i]$; $R = R - M - 1$; $t--$;

end

$r = R\%(n-i)$; $l = (R-r)/(n-i)$;

if $r < \ell$ **then**

  $j = n - \ell; k = n - r - 1$;

else

  $j = \ell + i$; $k = r + i$;

end

Return: 返回 $x_i x_j x_k$ 的下标分量 $\{i, j, k\}$.

压缩函数CF(x)的生成算法及解码算法

第五章、多变量Hash函数的构造

5.3.2、 MPH算法的实现效率

表 1　MPH算法的复杂性

运算类型	运算次数
查表运算	$n(3Terms + n^2 + 2n)/2$
逐位异或运算 “∧”	$(Terms + n^2 + 3n + 12)n\log_2^q/4$
逻辑与运算“&”	$n\log_2^q/2$
右移运算“≫”	$n\log_2^q/2$

表 2　MPH-x与SHA-x系列算法的效率比较

	MPH-160		MPH-224		MPH-256		MPH-384		MPH-512	
ρ_2	ρ_3	η	ρ_3	η	ρ_3	η	ρ_3	η	ρ_3	η
0. 50	0.018	21	0.013	-	0.011	61	0.007	97	0.006	238
0.60	0.024	24	0.017	-	0.015	66	0.010	112	0.008	259
0.70	0.031	27	0.022	-	0.018	72	0.013	128	0.011	286
0.80	0.042	29	0.030	-	0.026	80	0.017	143	0.015	345
0.90	0.065	34	0.045	-	0.037	98	0.024	171	0.022	426
>0.96	0.092	42	0.076	-	0.055	121	0.046	231	0.043	699

4.3.2、 MPH算法的实现效率

表 1　MPH算法的复杂性

运算类型	运算次数
查表运算	$n(3Terms + n^2 + 2n)/2$
逐位异或运算 “∧”	$(Terms + n^2 + 3n + 12)n\log_2^q/4$
逻辑与运算“&”	$n\log_2^q/2$
右移运算“≫”	$n\log_2^q/2$

表 2　MPH-x与SHA-x系列算法的效率比较

	MPH-160		MPH-224		MPH-256		MPH-384		MPH-512	
ρ_2	ρ_3	η	ρ_3	η	ρ_3	η	ρ_3	η	ρ_3	η
0. 50	0.018	21	0.013	-	0.011	61	0.007	97	0.006	238
0.60	0.024	24	0.017	-	0.015	66	0.010	112	0.008	259
0.70	0.031	27	0.022	-	0.018	72	0.013	128	0.011	286
0.80	0.042	29	0.030	-	0.026	80	0.017	143	0.015	345
0.90	0.065	34	0.045	-	0.037	98	0.024	171	0.022	426
>0.96	0.092	42	0.076	-	0.055	121	0.046	231	0.043	699

4.4.1、 MPH算法的伪随机性测试

表 3　MPH-160实例的随机性测试结果

序号	测试项目	测试次数	$P-value_{\tau}$ 最小值	最小通过率
1	Rrequency	1	0.125200	0.9910
2	Block-frequency	1	0.259616	0.9910
3	Cumulative-sums	2	0.267573	0.9900
4	Runs	1	0.030197	0.9920
5	Longest-run	1	0.911413	0.9880
6	Rank	1	0.564639	0.9900
7	DFT	1	0. 036592	0.9920
8	Nonperiodic-templates	148	0.000991	0.9810
9	Overlapping-templates	1	0.733899	0.9890
10	Universal	1	0.542101	0.9880
11	Approximate entropy	1	0.990138	0.9870
12	Random-excursions	8	0.226559	0.9822
13	Random-excursions-variant	18	0.053823	0.9838
14	Serial	2	0.684890	0.9910
15	Linear-complexity	1	0.440975	0.9910

4.4.2、 MPH算法的雪崩性测试

表 4　GF(256)上MPH-160的雪崩性测试结果

三次项项数	$\bar{B}$	ΔB	P	ΔP
10	79.828000	6.893035	0.498925	0.499175
50	79.921000	6.712214	0.499506	0.499756
100	79.795000	6.729656	0.498719	0.498968
200(实例)	80.007100	6.901999	0.500416	0.500612
500	80.041000	6.831121	0.500507	0.501157
1000	79.145000	6.887264	0.500969	0.501219
全选	80.081200	6.665199	0.501020	0.503121

四 多变量Hash函数的构造

4.6、 MPH算法的评价

- 在安全性方面，可转化为MQ难题。
- 采用了HAIFA迭代框架结构, 故可避免传统基于传统Merkle-Damgård结构的攻击方法。
- 输出长度可变, 支持224比特、256比特、384比特、512比特以及更长输出的消息摘要。
- 具有设计自动化的优点, 算法设计简洁灵活。
- 可根据实际应用背景, 通过控制压缩函数的稀疏度以进行安全性和性能的折中。

❖ 缺点：实现效率较SHA系列低，但可考虑并行加速。

谢　谢！