第五章、哈希函数及其在区块链中的应用

何德彪

武汉大学

国家网络安全学院

目　录

目　录

5.1 哈希函数的定义

1. 哈希函数的一般定义

哈希函数(Hash Function)是一公开函数，用于将任意长的消息M映射为较短的、固定长度的一个值$H(M)$，又称为散列函数、杂凑函数.我们称函数值$H(M)$为哈希值、杂凑值、杂凑码、或消息摘要.

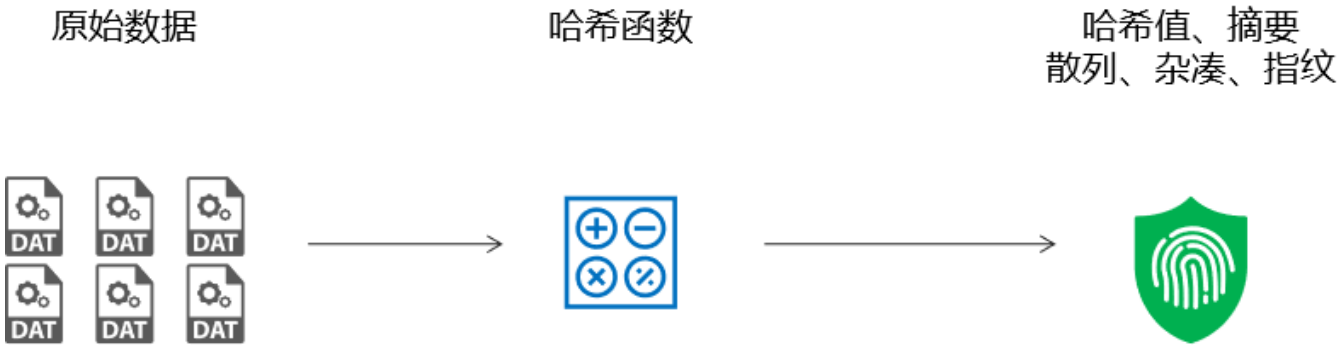


图5.1. 哈希函数功能示意图

- 杂凑值是消息中所有比特的函数，因此提供了一种**错误检测能力**，即改变消息中任何一个比特或几个比特都会使杂凑值发生改变.

5.1 哈希函数的定义

2. 哈希函数的性质

哈希函数具有如下性质：

- H可以作用于一个任意长度的数据块(实际上是不是任意，比如SHA-1要求不超过2^{64})；
- H产生一个固定长度的输出(比如SHA-1的输出是160比特，**SHA-256的输出是256比特**)；
- 对任意给定的x，***H(x)*计算相对容易**，无论是软件还是硬件实现；
- 单向性(抗原像)(One-Way)：对于任意给定的消息，计算其哈希值容易.但是，对于给定的哈希值h，要找到M使得$H(M)=h$在计算上是不可行的.

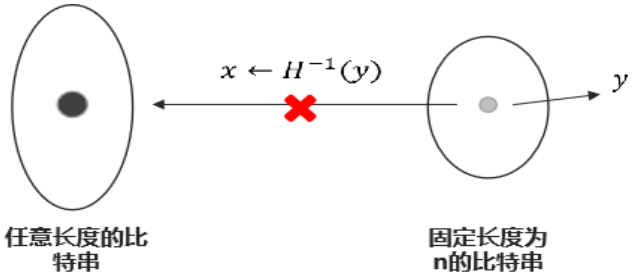


图5.2. 哈希函数单项性示意图

5.1 哈希函数的定义

2. 哈希函数的性质

- **弱抗碰撞**(抗二次原像) (Weakly Collision-Free)：对于给定的消息x，要发现另一个消息y，满足$H(x)=H(y)$在计算上是不可行的.
- **强抗碰撞**(Strongly Collision-Free)：找任意一对不同的消息x，y，使$H(x)=H(y)$在计算上是不可行的.

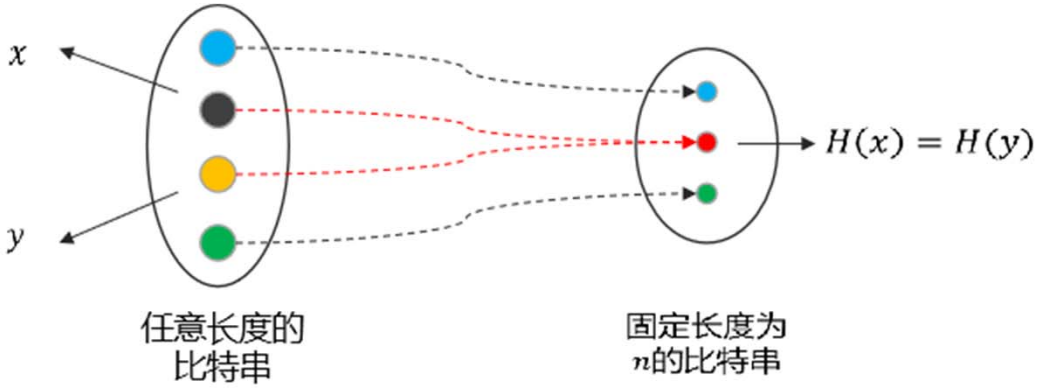


图5.3. 哈希函数强碰撞性示意图

5.1 哈希函数的定义

备注：

① 其中前3条是实用性要求，后3条是安全性要求.

② 单向性：即给定消息可以产生一个哈希值，而给定哈希值不可能产生对应的消息；否则, 设传送数据$C=<M, H(M\|K)>$，K是密钥.攻击者可以截获C，求出哈希函数的逆，从而得出$H^{-1}(C)$，然后从M 和$M\|K$即可得出K.

③ 弱抗碰撞性：是保证一个给定的消息的哈希值不能找到与之相同的另外的消息，即防止伪造.否则，攻击者可以截获报文M及其哈希函数值$H(M)$,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.

④ 强抗碰撞性：是对已知的生日攻击方法的防御能力.

⑤ 强抗碰撞自然包含弱抗碰撞！

5.1 哈希函数的定义

总结：密码学上安全的杂凑函数H应具有以下性质：

① 对于任意的消息x，计算$H(x)$是容易的；

② H是单向的；

③ H是强抗碰撞的.

目　录

5.2 哈希函数的发展

- 1978年，Merkle和Damagad设计MD迭代结构；
- 1993年，来学嘉和Messay改进加强MD结构；
- 在90年代初MIT Laboratory for Computer Science和RSA数据安全公司的Rivest设计了散列算法MD族，MD代表消息摘要.
- MD族中的MD2、MD4和MD5都产生一个128位的信息摘要.

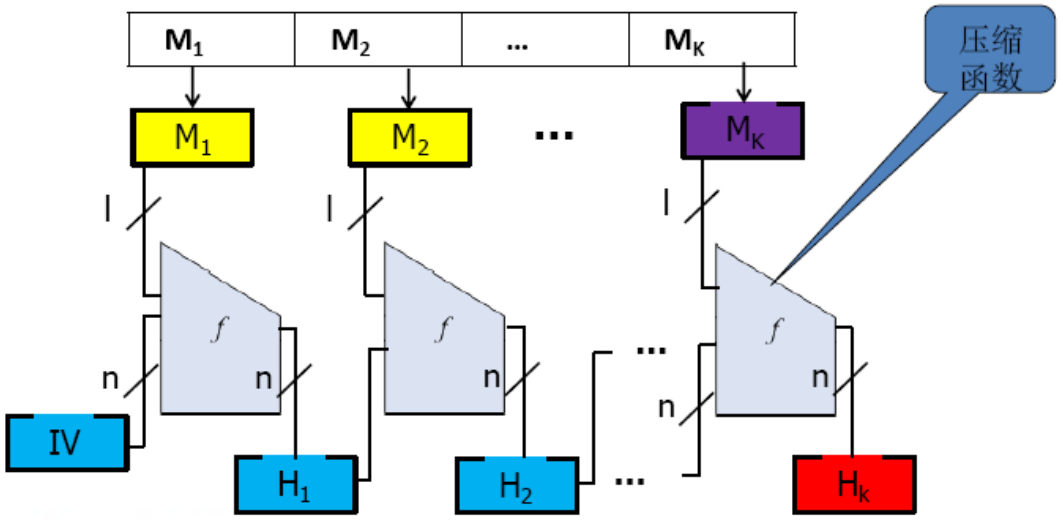


图5.4. 哈希函数结构示意图

5.2 哈希函数的发展

- MD2(1989年)
- MD4(1990年)
- MD5(1991年)：由美国密码学家罗纳德·李维斯特(Ronald Linn Rivest)设计，经MD2、MD3和MD4发展而来，输出的是128位固定长度的字符串.
- RIPEMD-128/160/320：国际著名密码学家Hans Dobbertin的在1996年攻破了MD4算法的同时，也对MD5的安全性产生了质疑，从而促使他设计了一个类MD5的RIPEMD-160. 在结构上，RIPEMD-160可以视为两个并行的类MD5算法，这使得RIPEMD-160的安全性大大提高.
- **值得注意得是，MD4， MD5已经在2004年8月Crypto2004 上，被我国密码学者王小云等破译，即在有效的时间内找到了它们的大量碰撞.**

5.2 哈希函数的发展

SHA系列算法是美国国家标准与技术研究院(National Institute of Standards and Technology，NIST)根据Rivest设计的MD4和MD5开发的算法. 美国国家安全局(National Security Agency，NSA)发布SHA作为美国政府标准.

- **SHA-0**

SHA-0正式地称作SHA，这个版本在发行后不久被指出存在弱点.

- **SHA-1**

SHA-1是NIST于1994年发布的，它与MD4和MD5算法非常相似，被认为是MD4和MD5的后继者.

- **SHA-2**

SHA-2实际上分为SHA-224、**SHA-256**、SHA-384和SHA-512算法.

5.2 哈希函数的发展

算法	输出长度(bits)	初始状态长度(bits)	分块长度(bits)	最大消息长度	字长	圈数	操作	碰撞
SHA-0	160	160	512					
SHA-1	160	160	512	$2^{64}-1$	32	80	+,and,or,xor,rotl	2^{63}攻击
SHA-256/224	256/224	256	512	$2^{64}-1$	32	64	+,and,or,xor,rotl,shr	没有
SHA-512/384	512/384	512	1024	$2^{128}-1$	64	80	+,and,or,xor,rotl,shr	没有

2017年2月23日谷歌宣布，谷歌研究人员和阿姆斯特丹CWI研究所合作发布了一项新的研究，详细描述了成功的SHA1碰撞攻击.

图5.5. SHA系列哈希函数相关参数比较

5.2 哈希函数的发展

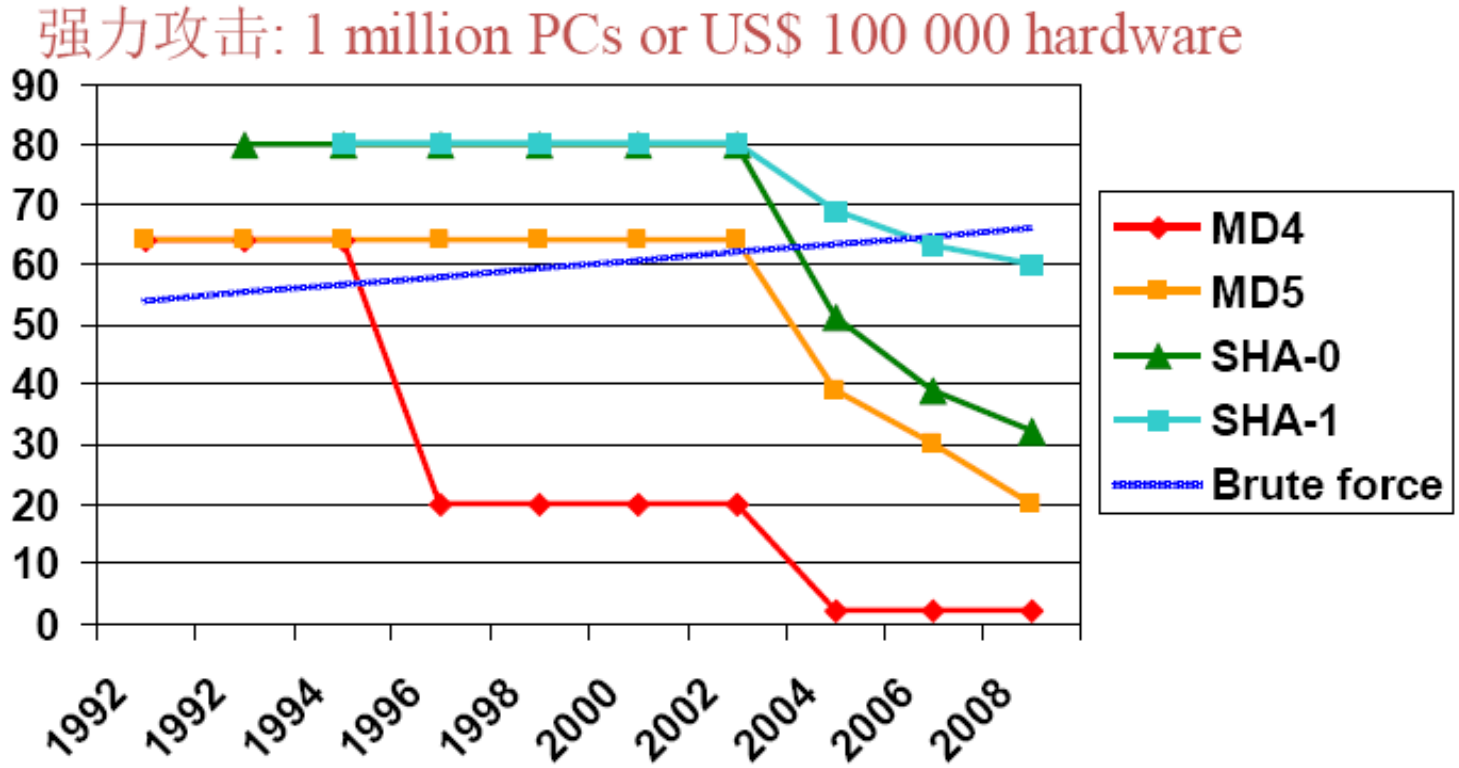


图5.6. 哈希函数碰撞攻击复杂度示意图 （单位：年）

5.2 哈希函数的发展

- NESSIE工程推荐使用的哈希算法有SHA-256/384/512和Whirlpool (Vincent Rijmen和Paulo S. L. M. Barreto在2000年设计，入选国际标准ISO/IEC 10118-3). (NESSIE是欧洲一项为期三年的密码标准计划, 详见： http://www.nessieproject.com/)
- 日本密码研究与评估委员会推荐使用的算法有RIPEMD-160、SHA-256/384/512.
- 此外，NIST于2008年启动新的哈希标准的征集活动.
  - ✓除迭代结构以外的结构
  - ✓适用于任何平台的压缩函数
- 2008年10月提交文档，收到64个算法，公开56个，51个进入第一轮评估.
- 2009年10月，第二轮评估开始，剩余14个算法.
- 第三轮剩余五个算法，2012年10月2日， Keccak被选为SHA-3.

目　录

5.3 哈希函数的常见攻击方法

1. 穷举攻击

① 原像攻击和第二原像攻击(Preimage or Second Preimage attack)

- 对于给定的哈希值h，试图找到满足 $H(x)=h$的x;
- 对于m位的哈希值，穷举规模大约是2^m.

② 碰撞攻击(Collision Resistance)

- 找到两个消息$x \neq y$,满足$H(x) = H(y)$;
- 对于m位的哈希值，预计在$2^{m/2}$次尝试后就将找到两个具有相同哈希值的数据.

③ 因此对于m位的哈希值，抗穷举攻击的强度为$2^{m/2}$

- 目前128比特已经不够, 需要160比特甚至更多.

5.3 哈希函数的常见攻击方法

2. 生日攻击

考虑教室有30位同学，定义函数H: {张三,李四,……|在教室里的同学}→{1,2, ……,365}，如果有两个同学的生日相同，则称为一个“碰撞”. 直观看来，产生碰撞的可能性较小. 但是，对于30个人的人群，这个事情发生的可能性大约为1/2.当人数增加时，这个可能性就增大. 这个事实与我们的直观相悖, 称为”生日悖论”.

① 问题：某个人是10月1日生日的概率为多少？

② 问题：教室里有人是10月1日生日的概率是多少？

③ 问题：某个同学与其他任何一个同学生日相同的概率为多少？

④ 问题：教室有两个同学具有相同生日的概率是多少？

我们对于一般的函数H:$\{0,1\}^m \to \{0,1\}^n$进行回答.

5.3 哈希函数的常见攻击方法

1. 问题: 某个人是10月1日生日的概率为 $1/365$.
   对于一般的函数 $H : \{0,1\}^m \to \{0,1\}^n$ 问题是:
   随机的在 $\{0,1\}^n$ 中取一个元素 y, 问集合 $S \subseteq \{0,1\}^m$ 中的任意元素 $x \in S$ 使得 $H(x) = y$ 的概率是多少?
   答案 $1/2^n$. 前提是每个元素 $y \in \{0,1\}^n$ 的原像个数几乎相等.

- 问题: 教室里有人是 10月1日生日的概率是多少?
- 一般问题是:对于任意集合 $S \subseteq \{0,1\}^m$, 以及任意 $y \in \{0,1\}^n$. 存在 $x \in S$ 使得 $H(x) = y$ 的概率是多少?

5.3 哈希函数的常见攻击方法

- 计算: 对于固定的 y, 令 $S = \{x_1, x_2, \ldots, x_s\}$, $s = |S|$. 用 E_i 记事件 "$H(x_i) = y$". 则 E_i 是相互独立的, 并且根据上面的问题答案, 有

$$\Pr[E_i] = 1/2^n$$

所以 $\Pr[\overline{E}_i] = 1 - \Pr[E_i] = 1 - 1/2^n$. 故有

$$\begin{aligned}
\Pr[E_1 \vee E_2 \vee \cdots \vee E_s] &= 1 - \Pr[\overline{E_1 \vee E_2 \vee \cdots \vee E_s}] \\
&= 1 - \Pr[\overline{E}_1 \wedge \overline{E}_2 \wedge \cdots \wedge \overline{E}_s] \\
&= 1 - \Pr[\overline{E}_1] \cdot \Pr[\overline{E}_2] \cdot \cdots \cdot \Pr[\overline{E}_s] \\
&= 1 - (1 - 1/2^n)^s
\end{aligned}$$

5.3 哈希函数的常见攻击方法

- 问题: 教室有两个同学具有相同生日的概率是多少? (生日攻击: 教室有多少人时, 这个概率超过 $1/2$?)
- 一般问题是:对于任意集合 $S \subseteq \{0,1\}^m$, 存在 $x, x' \in S$ 使得 $H(x) = H(x')$ 的概率是多少?
- 计算: 令 $N := 2^n$, $S = \{x_1, x_2, \ldots, x_s\}$, $s = |S|$. 用 D_i 记事件

$$\text{“}H(x_i) \notin \{H(x_1), \ldots, H(x_{i-1})\}\text{”}$$

使用归纳法: $\Pr[D_1] = 1$ 并且对于 $2 \le i \le s$,

$$\Pr[D_i \mid D_1 \wedge D_2 \wedge \cdots \wedge D_{i-1}] = \frac{N - i + 1}{N},$$

5.3 哈希函数的常见攻击方法

所以 $\Pr[D_1 \wedge D_2 \wedge \cdots \wedge D_s] = \left(\frac{N-1}{N}\right)\left(\frac{N-2}{N}\right)\cdots\left(\frac{N-s+1}{M}\right)$.
至少有一个碰撞是事件“$\overline{D_1 \wedge \cdots \wedge D_s}$”, 其概率是

$$1 - \Pr[D_1 \wedge D_2 \wedge \cdots \wedge D_s] = 1 - \left(\frac{N-1}{N}\right)\left(\frac{N-2}{N}\right)\cdots\left(\frac{N-s+1}{N}\right)$$

使用近似公式: 当 x 较小时, 有 $1 - x \approx e^{-x}$. 上式得到:

$$\begin{aligned} \text{上式} &= 1 - \left(1 - \frac{1}{N}\right)\left(1 - \frac{2}{N}\right)\cdots\left(1 - \frac{s-1}{N}\right) \\ &= 1 - \prod_{i=1}^{s-1}\left(1 - \frac{i}{N}\right) \\ &\approx 1 - \prod_{i=1}^{s-1} e^{\frac{-i}{N}} = 1 - e^{\frac{-s(s-1)}{2N}} \end{aligned}$$

5.3 哈希函数的常见攻击方法

记上式为 ε, 则 $e^{\frac{-s(s-1)}{2N}} \approx 1-\varepsilon$. 解之得 $s \approx \sqrt{2N \ln \frac{1}{\varepsilon}}$.

如果 $\varepsilon = 1/2$, 则 $s \approx 1.17\sqrt{N}$. 当 $N = 365$, 则 $s \approx 22.35 \approx 23$.

如果 $e^{\frac{-s(s-1)}{2N}} \approx 1$

$$s \approx 1.17\, N^{0.5} \approx N^{0.5} = (2^n)^{0.5} = 2^{n/2}$$

即对于***n***位的哈希值，只要尝试$2^{n/2}$次，就至少存在一对***x***和***x'***，使得***H(x)*＝*H(x')***相同.

5.3 哈希函数的常见攻击方法

3. 其它攻击

- 利用算法的某种性质进行攻击；
- 哈希函数使用迭代结构
  - ✓将消息分组后分别进行处理
- 密码分析的攻击集中于压缩函数 f 的碰撞.

目　录

5.4 哈希函数的构造

1. 基于数学难题的构造方法

- MASH-1 (Modular Arithmetic Secure Hash)是一个基于RSA算法的哈希算法，在1995年提出，入选国际标准ISO/IEC 10118-4；
- MASH-2是MASH-1的改进，把第四步中的2换成了2^8+1；
- 由于涉及模乘/平方运算，计算速度慢，非常不实用；

MASH-1(1995 年 11 月的版本)

输入：比特长为 $0 \leqslant b \leqslant 2^{n/2}$ 的数据 x。

输出：x 的 n 比特杂凑(n 接近模数 M 的比特长度)。

1. 系统设置和常数定义。固定比特长度为 m 的类 RSA 的模数 $M=pq$，其中 p 和 q 是随机选择的秘密素数，使得 M 的因子分解是困难的。定义杂凑结果的比特长度 n 是小于 m 的 16 的最大整数倍数(即 $n=16n' < m$)。定义 IV 为 $H_0=0$ 和一个 n 比特的整数 $A=0x80\cdots00$。“∨”表示按比特或，“⊕”表示按比特异或。
2. 填充、分组和 MD 强化。必要时用 0 比特填充 x 以得到比特长度为 $t \cdot n/2$ 的串(取最小的 $t \geqslant 1$)。将填充的文本划分为 $n/2$ 个比特分组 $x_1, \cdots, x_t$，附加最后一个表示 b 的比特分组 x_{t+1}，它的长度也是 $n/2$。
3. 扩充。将 x_i 扩充为 n 比特分组 y_i。方法是将它分割成 4 比特的半字节，在每个半字节前插入四个值为 1 的比特，但对 y_{t+1} 插入的是 1010(不是 1111)。
4. 压缩函数处理。对 $1 \leqslant i \leqslant t+1$，将两个 n 比特的输入 (H_{i-1}, y_i) 按照以下方法映射为一个 n 比特的输出：$H_i \leftarrow ((((H_{i-1} \oplus y_i) \vee A)^2 \bmod M) \dashv n) \oplus H_{i-1}$，其中 $\dashv n$ 表示保留 m 比特结果最右边的 n 比特到结果的左边。
5. 完成。杂凑值是 n 比特分组 H_{t+1}。

5.4 哈希函数的构造

2. 利用对称密码体制来设计哈希函数

分组密码的工作模式是：根据不同的数据格式和安全性要求， 以一个具体的分组密码算法为基础构造一个分组密码系统的方法.基于分组的对称密码算法比如DES/AES算法只是描述如何根据秘钥对一段固定长度(分组块)的数据进行加密，对于比较长的数据，分组密码工作模式描述了如何重复应用某种算法安全地转换大于块的数据量.

简单的说就是，DES/AES算法描述怎么加密一个数据块，分组密码工作模式模式了如果重复加密比较长的多个数据块. 常见的分组密码工作模式有五种：电码本(Electronic Code Book，ECB)模式、密文分组链接(Cipher Block Chaining，CBC)模式、密文反馈(Cipher Feed Back ，CFB)模式、输出反馈(Output Feed Back ， OFB)模式和计数器(Counter, CTR)模式.

大家还记得五种工作模式的工作流程吗？

5.4 哈希函数的构造

(1) ECB工作模式

加密：输入是当前明文分组.

解密：每一个密文分组分别解密.

具体公式为：

$$C_n = E_K[P_n]$$

$$P_n = D_K[C_n]$$

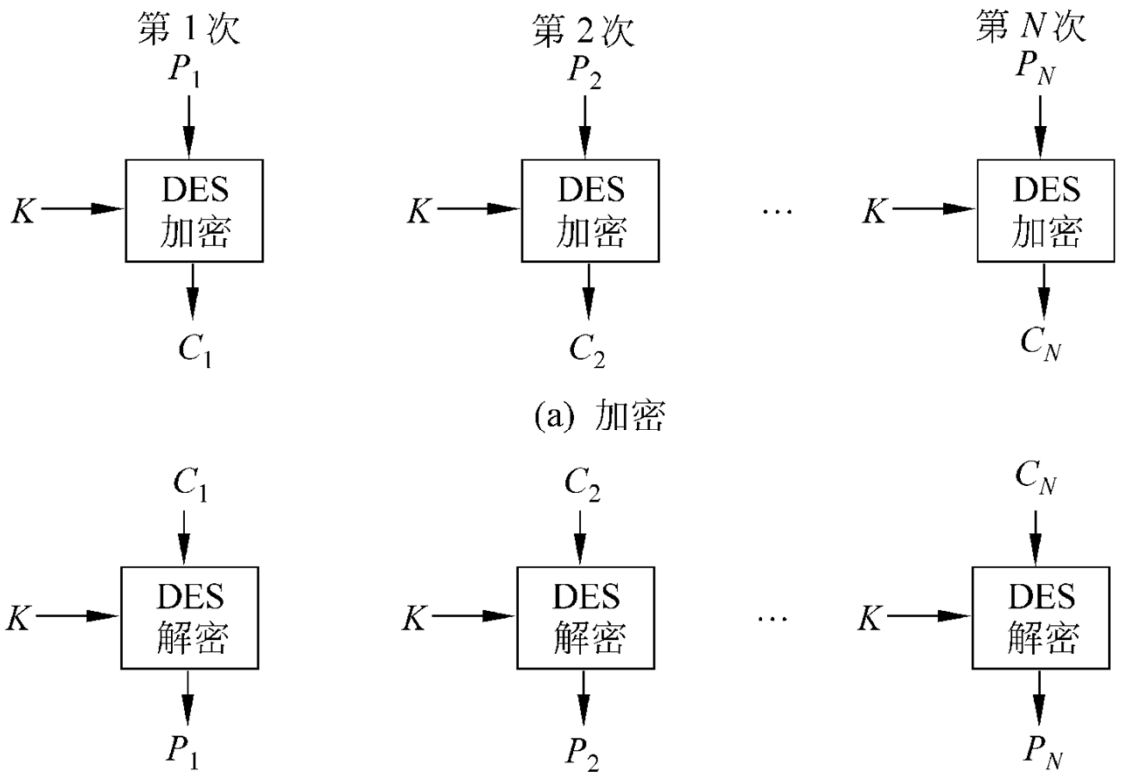


图5.7. ECB工作模式示意图

5.4 哈希函数的构造

(2) CBC工作模式

加密：输入是当前明文分组和前一次密文分组的异或.

解密：每一个密文分组被解密后，再与前一个密文分组异或得明文.

具体公式为：

$$C_n = E_K\left[C_{n-1} \oplus P_n\right]$$

$$P_n = D_K\left[C_n\right] \oplus C_{n-1}$$

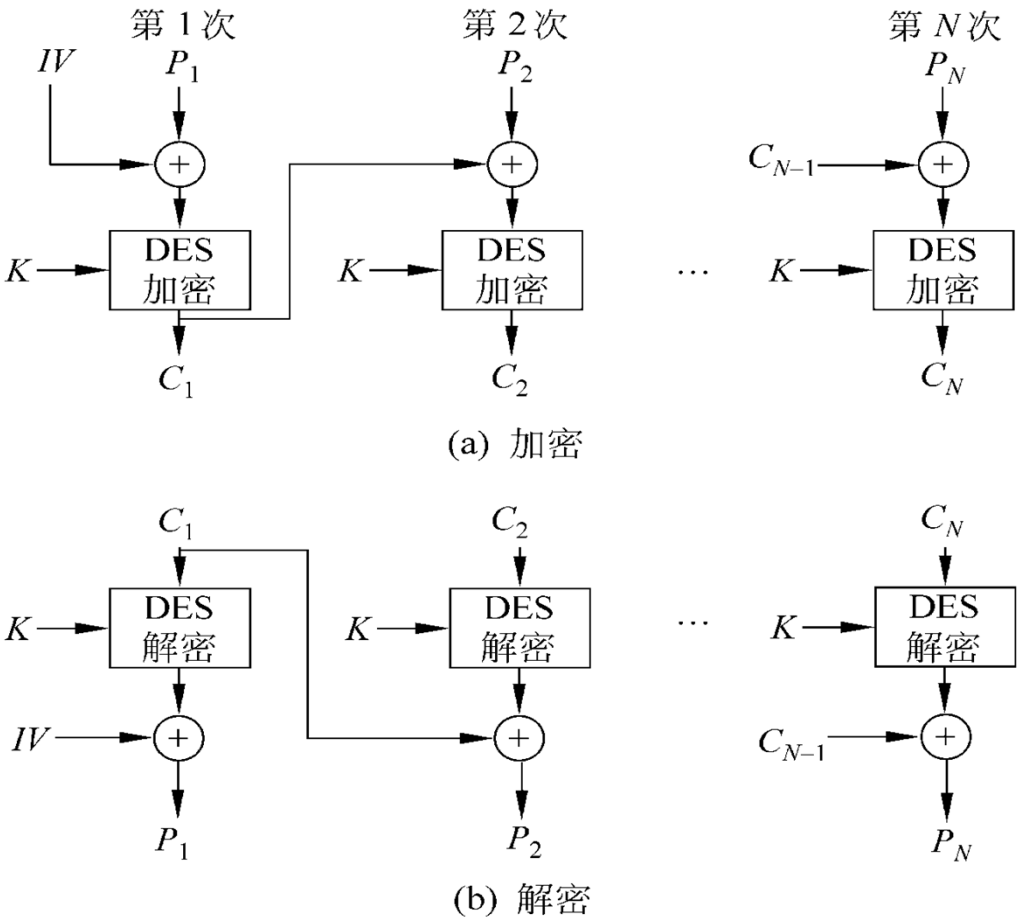


图5.8. CBC工作模式示意图

5.4 哈希函数的构造

(3) CFB工作模式

- 加密算法的输入是64比特移位寄存器，其初值为某个初始向量IV.
- 加密算法输出的最左(最高有效位)j比特与明文的第一个单元P1进行异或，产生出密文的第1个单元C1，并传送该单元.
- 然后将移位寄存器的内容左移j位并将C1送入移位寄存器最右边(最低有效位)j位.
- 这一过程继续到明文的所有单元都被加密为止.

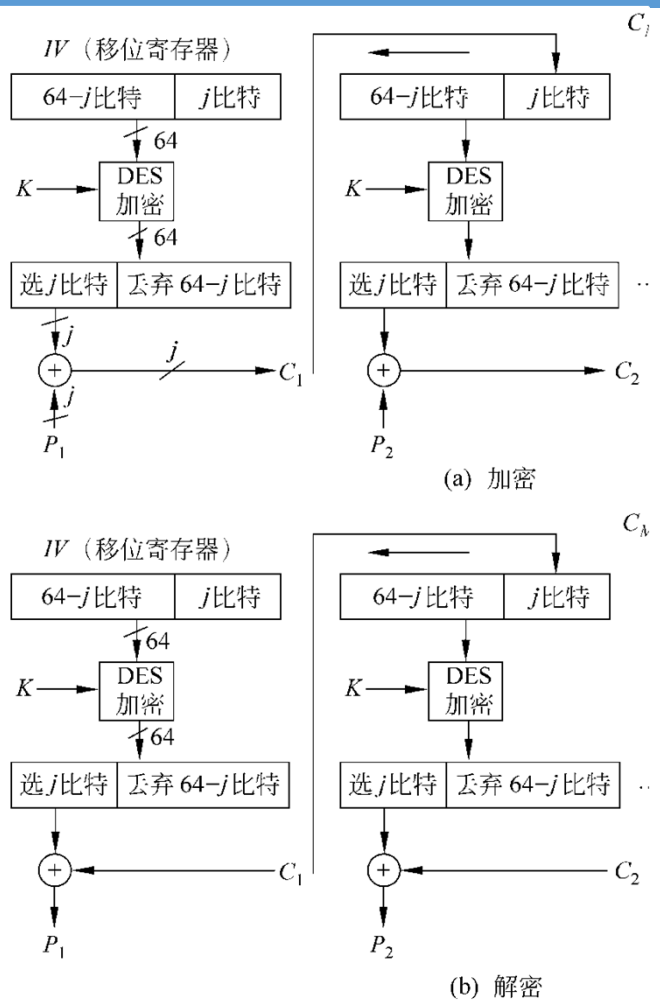


图5.9. CFB工作模式示意图

5.4 哈希函数的构造

(4) OFB工作模式

- OFB模式的结构类似于CFB.
- 不同之处
  - ✓ **OFB模式是将加密算法的输出反馈到移位寄存器**
  - ✓ **CFB模式中是将密文单元反馈到移位寄存器.**

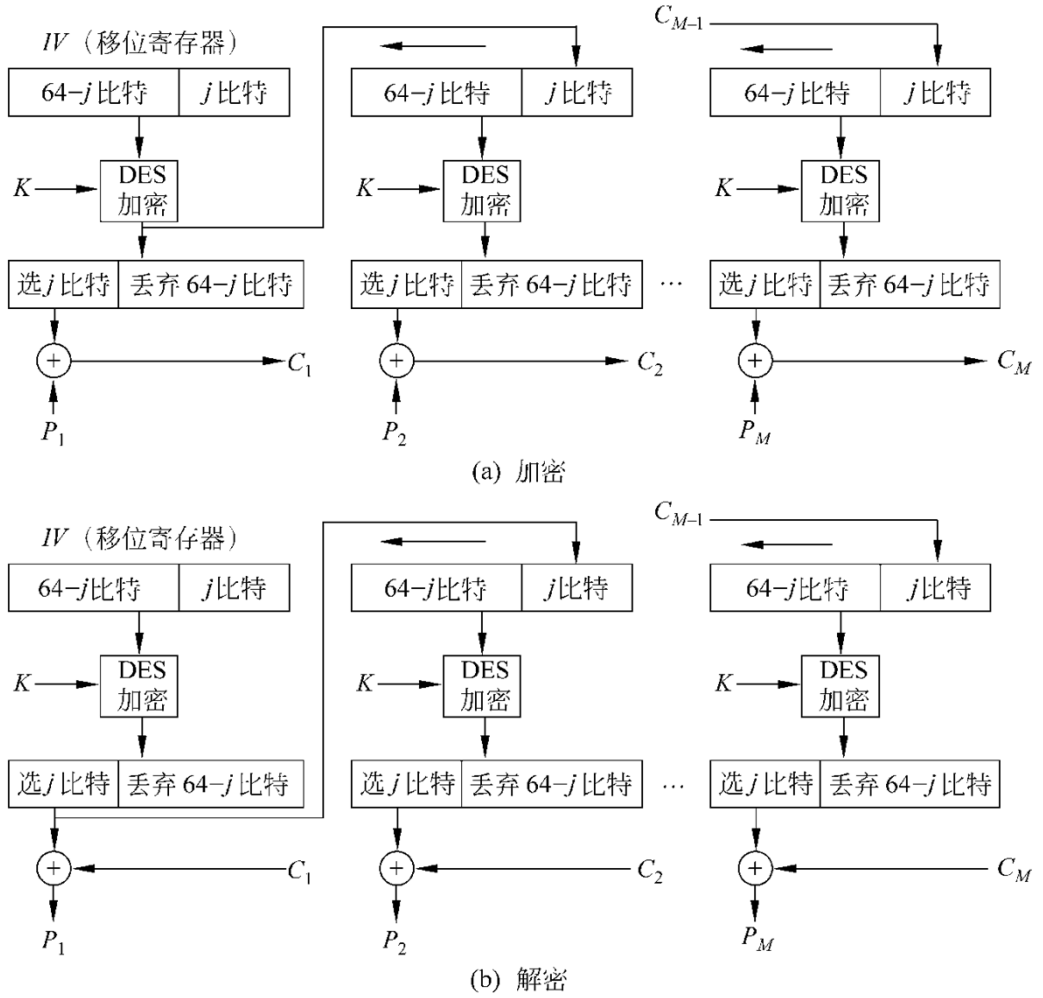


图5.10. OFB工作模式示意图

5.4 哈希函数的构造

(5) CTR工作模式

加密：输入是当前明文分组和计数器密文分组的异或.

解密：每一个密文分组被解密后，再与计数器密文分组异或得明文.

具体公式为：

$$C_i = E_K\left[VI \| CTR_i\right] \oplus P_i$$

$$P_i = E_K\left[VI \| CTR_i\right] \oplus C_i$$

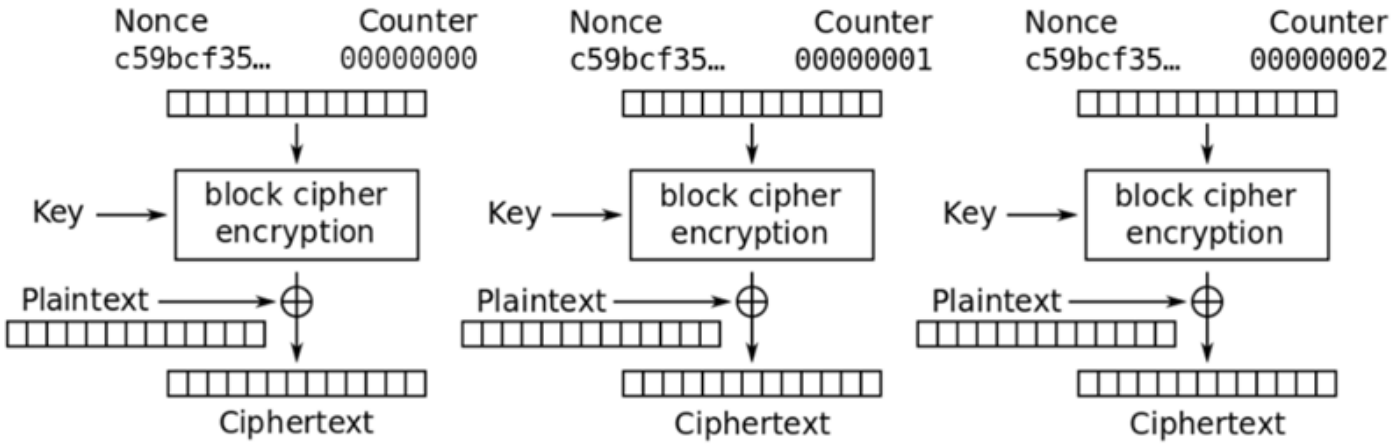


图5.11. CTR工作模式示意图

5.4 哈希函数的构造

工作模式的比较

- ECB模式，简单、高速，但最弱、易受重发攻击，一般不推荐.
- CBC模式适用于文件加密，比ECB模式慢.安全性加强. 当有少量错误时，不会造成同步错误.
- OFB模式和CFB模式较CBC模式慢许多. 每次迭代只有少数比特完成加密. 若可以容忍少量错误扩展，则可换来恢复同步能力，此时用CFB或OFB模式. 在字符为单元的流密码中多选CFB模式.
- CTR模式用于高速同步系统，不容忍差错传播.

下面利用对称密码来构造哈希函数，我们规定：

设 k 为密钥, E_k 为长度为 n 的分组密码 (DES, AES) 加密算法. 对于任意的消息 x, 先对 x 进行适当的填充, 使其长度恰好为 n 的倍数; 再将x 分为长度为 n 的 ℓ 块, 即有 $x = x_1x_2\cdots x_\ell$, 其中 $x_i \in \{0,1\}^n$, $1 \le i \le \ell$.

5.4 哈希函数的构造

2.1. 基于分组密码的CBC工作模式杂凑函数

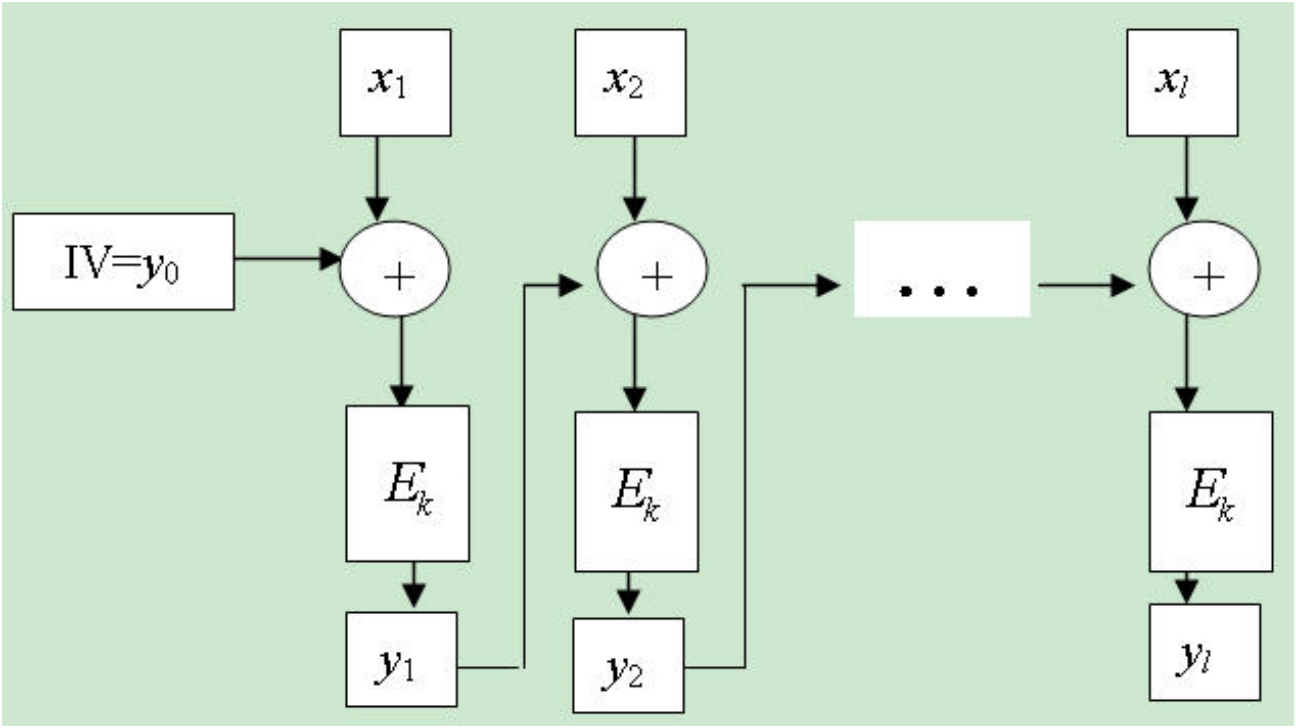


图5.12. 基于CBC工作模式的哈希函数

5.4 哈希函数的构造

2.2. 基于分组密码的CFB工作模式杂凑函数

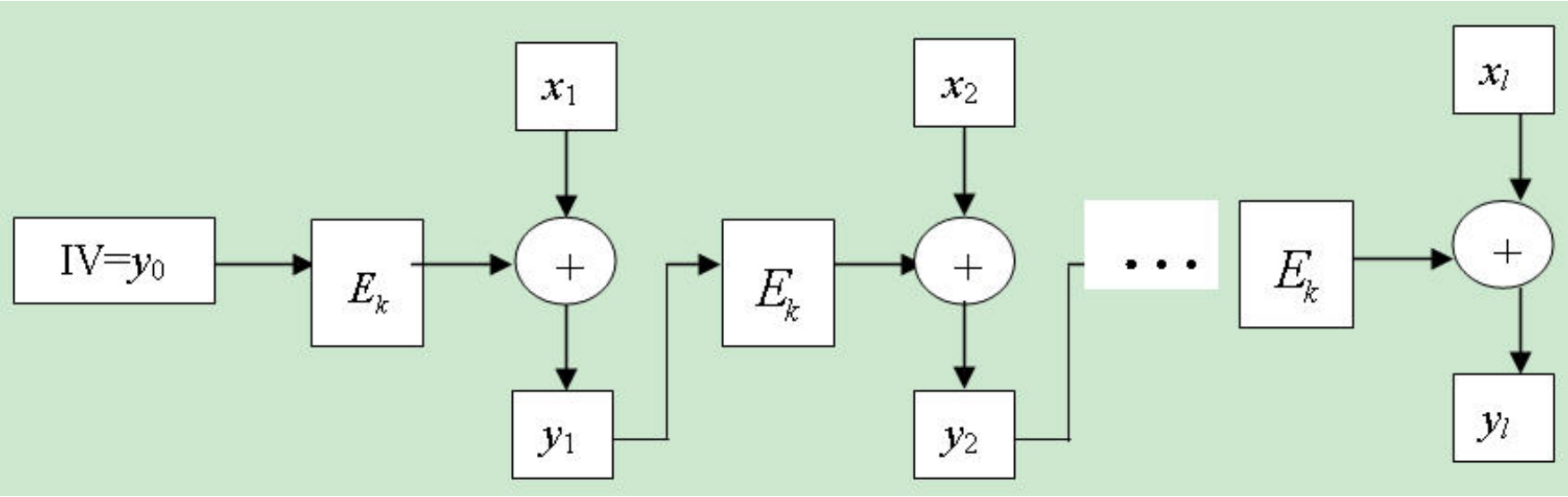


图5.13. 基于CFB工作模式的哈希函数

5.4 哈希函数的构造

- ✓ 上述两种基于分组密码的杂凑函数中，***K***可以公开，称为不带密钥的哈希函数；***K***也可以不公开，称为带密钥的哈希函数(MAC).
- ✓ 在***K***公开的情况下，上述两种构造杂凑函数的方法是不安全的, 即容易找到碰撞.(为什么？)

5.4 哈希函数的构造

3. 直接设计哈希函数

- Merkle在1989年提出迭代型哈希函数的一般结构；(另外一个工作是默克尔哈希树)
- Ron Rivest在1990年利用这种结构提出MD4；(另外一个工作是**RSA算法**)
- 这种结构在几乎所有的哈希函数中使用，具体做法为：

  - ✓ 把所有消息M分成一些固定长度的块Y_i；
  - ✓ 最后一块padding并使其包含消息M的长度；
  - ✓ 设定初始值CV_0；
  - ✓ 循环执行压缩函数f，$CV_i=f(CV_{i-1}||Y_{i-1})$；
  - ✓ 最后一个CV_i为哈希值；

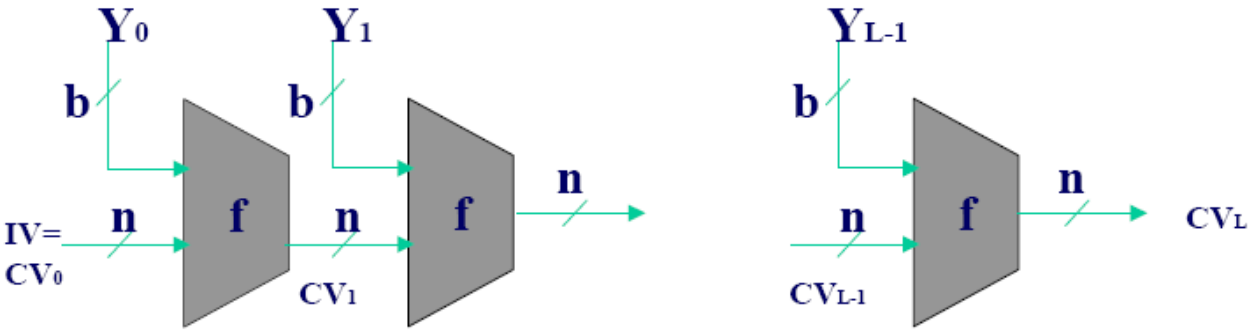


图5.14.迭代型哈希函数的一般结构示意图

5.4 哈希函数的构造

3. 直接设计哈希函数

- 算法中重复使用一个压缩函数f；
- f的输入有两项，一项是上一轮输出的n比特值CV_{i-1}，称为链接变量，另一项是算法在本轮的b比特输入分组Y_{i-1}；
- f的输出为n比特值CV_i，CV_i又作为下一轮的输入；
- 算法开始时还需对链接变量指定一个初值IV，最后一轮输出的链接变量CV_L即为最终产生的杂凑值；
- 通常有$b>n$，因此称函数f为压缩函数.
- 算法可表达如下：CV_0=IV= n比特长的初值；
- $CV_i=f(CV_{i-1},Y_{i-1})$；$1\le i\le L$；
- $H(M)=CV_L$；

5.4 哈希函数的构造

3. 直接设计哈希函数

- 算法的核心技术是设计难以找到碰撞的压缩函数f，而敌手对算法的攻击重点是f的内部结构；
- f和分组密码一样是由若干轮处理过程组成；
- 对f的分析需要找出f的碰撞.由于f是压缩函数，其碰撞是不可避免的，因此在设计f时就应保证找出其碰撞在计算上是困难的；

目　录

5.5 常用哈希函数简介

5.5.1. SHA-256算法简介

1. 概况

SHA系列标准哈希函数是由美国标准与技术研究所(National Institute of Standards and Technology，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)，包括3种算法：SHA-256, SHA-384, SHA-512.
- 2005年王小云院士给出一种攻击SHA-1的方法，用2^{69}操作找到一个强碰撞，以前认为是2^{80}.
- 2017年2月23日，谷歌宣布找到SHA-1碰撞的算法，需要耗费110块GPU一年的运算量.

5.5 常用哈希函数简介

5.5.1. SHA-256算法简介

	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

图5.15. SHA系列哈希函数的参数比较

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比特哈希函数的生日攻击产生碰撞的工作量大约为$2^{n/2}$

5.5 常用哈希函数简介

5.5.1. SHA-256算法简介

2. SHA-256算法描述

- 输入数据长度为l比特，$1 \le l \le 2^{64}-1$；
- 输出哈希值的长度为256比特.

(1) 常量与函数

SHA-256算法使用以下常数与函数：

①常量

初始值IV = 0x6a09e667 bb67ae85 3c6ef372 a54ff53a 510e527f 9b05688c 1f83d9ab 5be0cd19，这些初值是对自然数中前8个质数(2,3,5,7,11,13,17,19)的平方根的小数部分取前32比特而来.

5.5 常用哈希函数简介

5.5.1. SHA-256算法简介

举个例子来说，$\sqrt{2}$小数部分约为0.4142135623730950 48，而0.4142135623730950 48 ≈ $6*16^{-1}+a*16^{-2}+0*16^{-3}+9*16^{-4}+...$于是，质数2的平方根的小数部分取前32比特就对应出0x6a09e667.

另外，SHA-256还用到了64个常数：$K_0, K_1, \ldots, K_{63}$ =

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

和8个初始值类似，这些常量是对自然数中前64个质数(2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97…)的立方根的小数部分取前32比特而来.

5.5 常用哈希函数简介

5.5.1. SHA-256算法简介

②函数

SHA-256用到了以下函数

$$
\begin{aligned}
Ch(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\
Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\
\sum_0^{\{256\}}(x) &= ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \\
\sum_1^{\{256\}}(x) &= ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \\
\sigma_0^{\{256\}}(x) &= ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x) \\
\sigma_1^{\{256\}}(x) &= ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)
\end{aligned}
$$

其中： ∧表示按位“与”； ¬ 表示按位“补”； ⊕表示按位“异或”；

$ROTR^i(x)$表示循环右移i比特； $SHR^i(x)$表示右移i比特；

5.5 常用哈希函数简介

5.5.1. SHA-256算法简介

(2) 算法描述

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.**

5.5 常用哈希函数简介

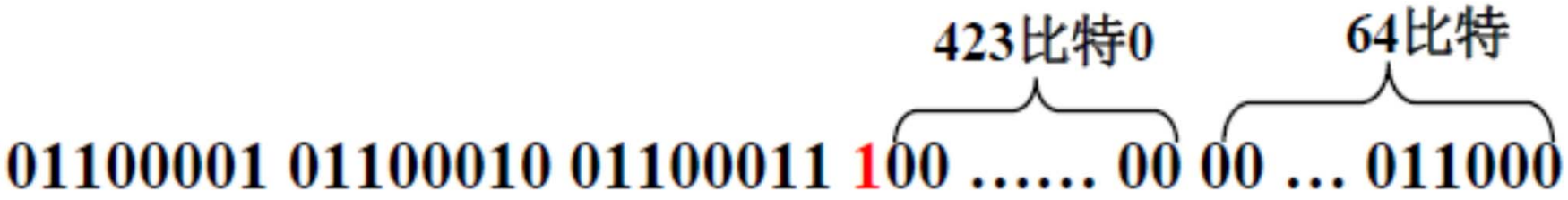
5.5.1. SHA-256算法简介

以信息“abc”为例显示补位的过程.a, b, c对应的ASCII码分别是97, 98, 99；于是原始信息的二进制编码为：01100001 01100010 01100011.

① 补一个“1”： 01100001 01100010 01100011 **1**

② 补423个“0”：01100001 01100010 01100011 **10000000 00000000 … 00000000**

③ 补比特长度24 (64位表示)，得到512比特的数据：



5.5 常用哈希函数简介

5.5.1. SHA-256算法简介

②消息分块

将填充后的消息m'按512比特分成n组：$m'=M^{(0)}||M^{(1)}||\cdots||M^{(n-1)}$，其中：$n=(l+k+65)/512$.

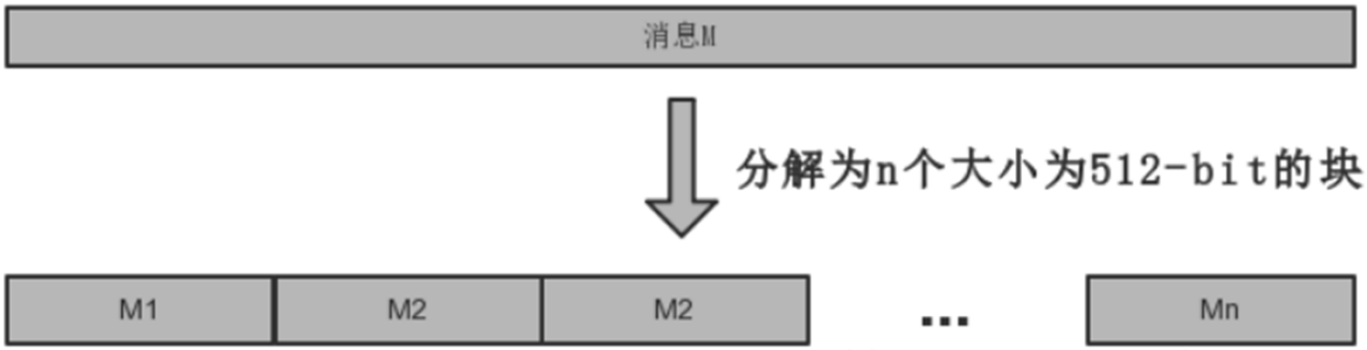


图5.16. SHA256算法的消息分块过程示意图

5.5 常用哈希函数简介

5.5.1. SHA-256算法简介

③消息扩展

➢对一个消息分组$M^{(i)}$迭代压缩之前，首先进行消息扩展

✓将16个字的消息分组M^t扩展生成如下的64个字，供压缩函数使用W_0，W_1，…，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：

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

5.5 常用哈希函数简介

5.5.1. SHA-256算法简介

④压缩函数

➢ 压缩函数是**SHA-256**的核心

➢ 令a, b, c, d, e, f, g, h为字寄存器，T_1, T_2为中间变量.

➢ 压缩函数：$V^{(t+1)}= CF(V^{(t)}, M^{(t)}), 0 \le t \le n-1$.

➢ 压缩函数CF的压缩处理：内层迭代

① FOR $t = 0$ TO 63

② $CF= F(T_1, T_2, a, b, c, d, e, f, g, h, M^{(t)})$

③ END FOR

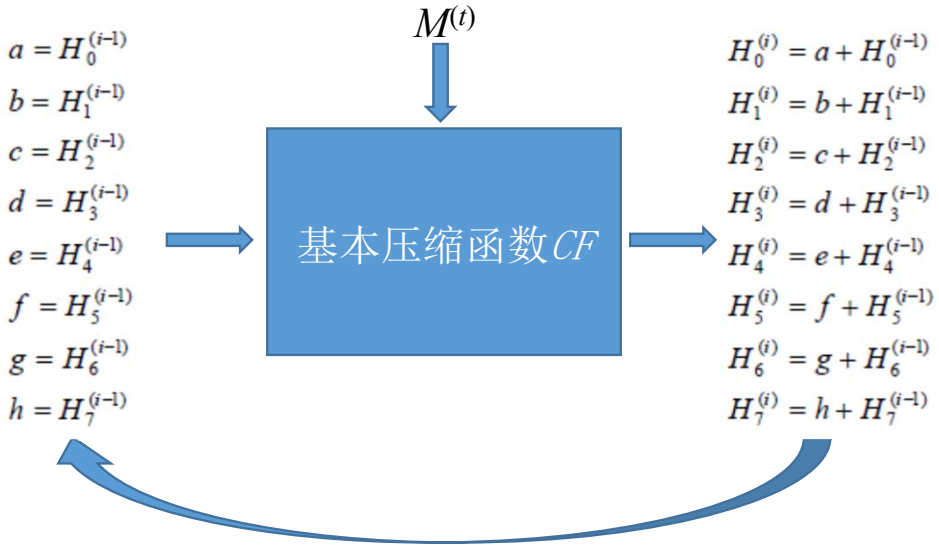


图5.17. SHA-256算法的压缩函数工作流程示意图

5.5 常用哈希函数简介

5.5.1. SHA-256算法简介

⑤基本压缩函数

基本压缩函数的流程如右边公式所述.

$$T_1 = h + \sum_1^{\{256\}}(e) + Ch(e,f,g) + K_t^{\{256\}} + W_t$$

$$T_2 = \sum_0^{\{256\}}(a) + Maj(a,b,c)$$

$$h = g$$

$$g = f$$

$$f = e$$

$$e = d + T_1$$

$$d = c$$

$$c = b$$

$$b = a$$

$$a = T_1 + T_2$$

说明：

- a, b, c, d, e, f, g, h为字寄存器， T_1, T_2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.

5.5 常用哈希函数简介

5.5.1. SHA-256算法简介

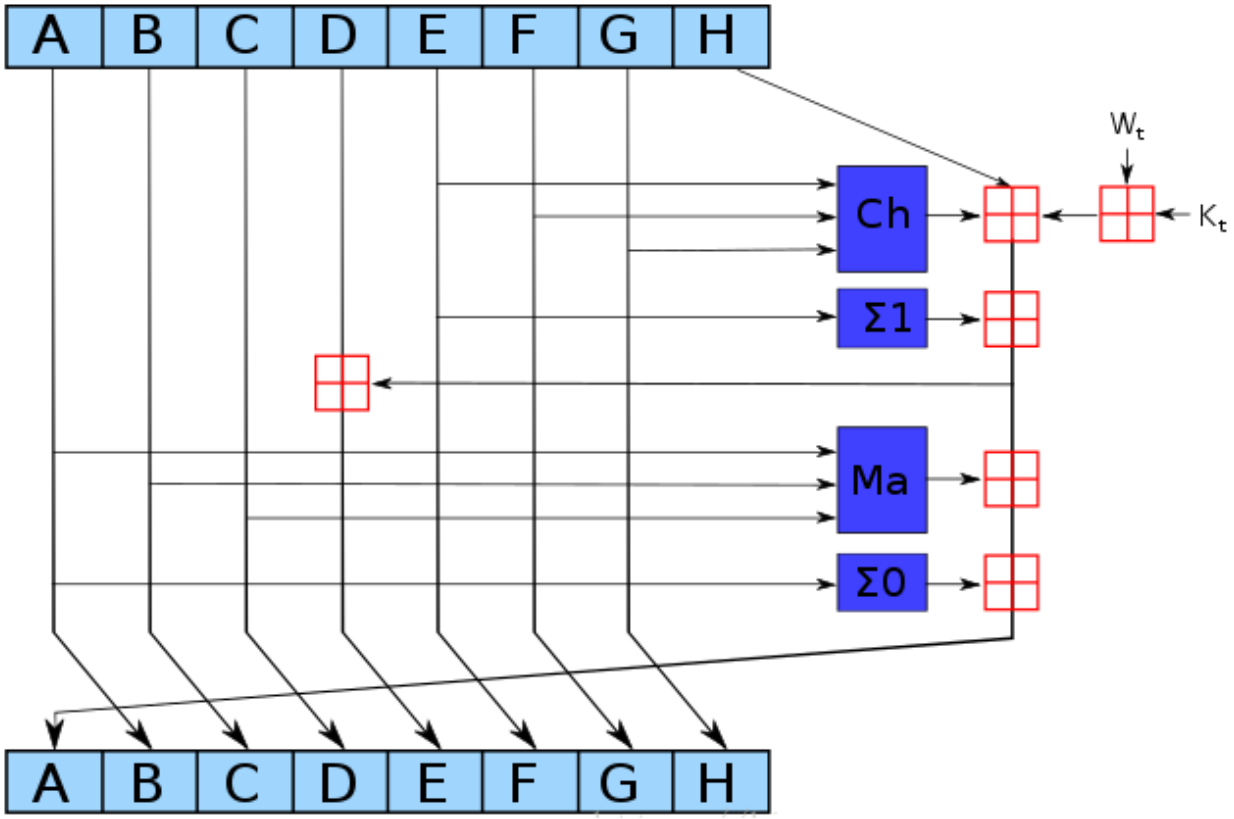


图5.18. SHA-256算法的基本压缩函数示意图

5.5 常用哈希函数简介

5.5.1. SHA-256算法简介

⑥SHA-256工作全过程

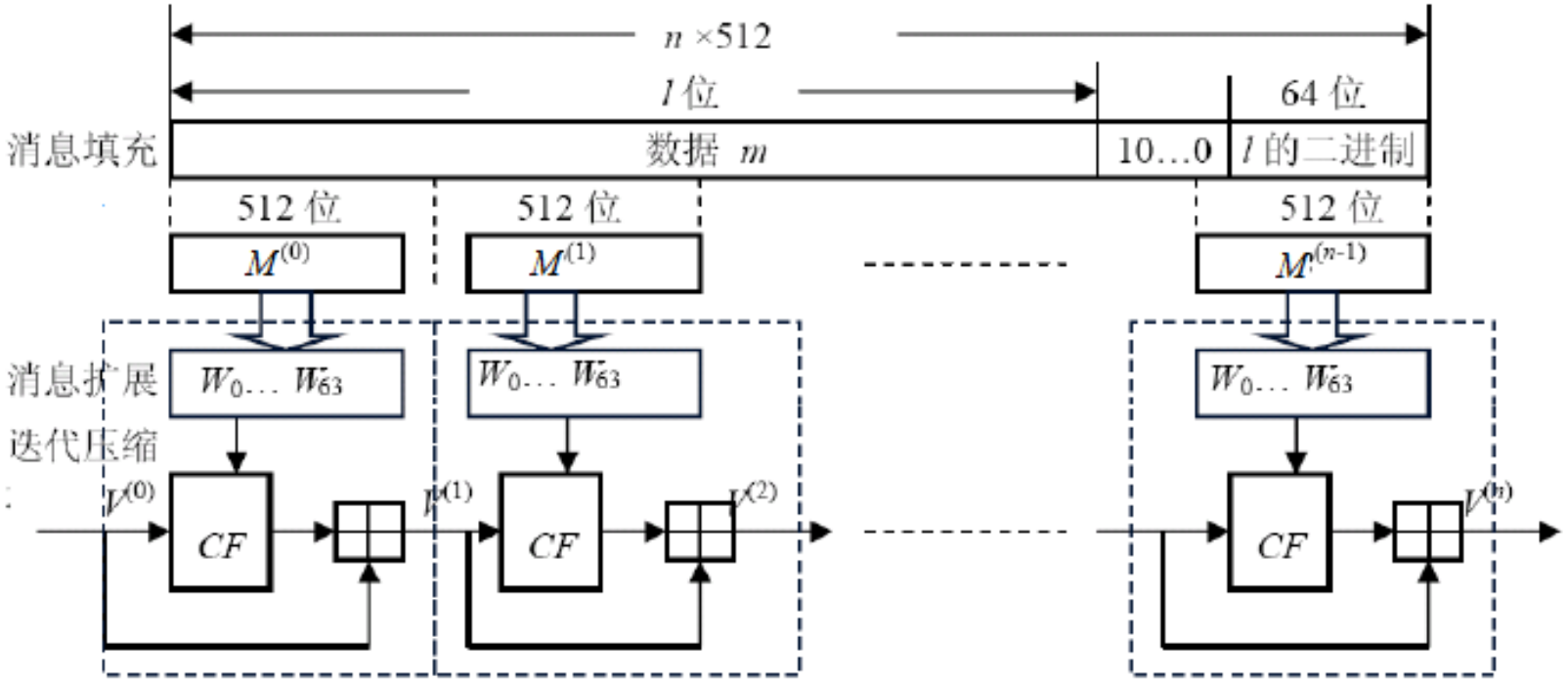


图5.19. SHA-256算法的工作全过程示意图

5.5 常用哈希函数简介

5.5.1. SHA-256算法简介

3. 安全性

- 专业机构设计，经过充分测试和论证；
- 安全性可满足应用的安全需求；
- 学者已开展对SHA-256的安全分析(如缩减轮的分析)，尚未发现本质的缺陷；

5.5 常用哈希函数简介

5.5.1. SHA-256算法简介

4. 程序设计

- typedef.h：定义数据类型；
- sha256.h：定义SHA-256算法相关功能函数、数据接口声明；
- sha256.c：实现SHA-256算法相关功能函数；
- sha256_test.h：定义测试函数、数据接口声明；
- Sha256_test.c：实现测试功能函数；
- main.c：实现main函数，测试程序的正确性、性能等.

```
#ifndef __HEADER_SHA256_H__
#define __HEADER_SHA256_H__

#include <string.h>
#include "typedef.h"

typedef struct
{
    U32 total[2];
    U32 state[8];
    U8 buffer[64];
}
sha256_context;

#ifdef __cplusplus
extern "C"
{
#endif

    void SHA256_Init( sha256_context *ctx );
    void SHA256_Transform( sha256_context *ctx, U8 data[64] );
    void SHA256_Update( sha256_context *ctx, U8 *input, U32 length );
    void SHA256_Final( sha256_context *ctx, U8 digest[32] );

#ifdef __cplusplus
}  /* end extern "C" */
#endif

#endif /* sha256.h */
```

目　录

5.5 常用哈希函数简介

5.5.2. Keccak算法简介

1. 概况

美国国家标准与技术研究院(National Institute of Standards and Technology，NIST)于2007年公开征集SHA-3，要求：

- 能够直接替代SHA-2.这要求SHA--3必须也能够产生224，256，384，512比特的哈希值.
- 保持SHA-2的在线处理能力. 这要求SHA-33必须能处理小的数据块(如512或1024比特).
- 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.
- 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.
- 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.

5.5 常用哈希函数简介

5.5.2. Keccak算法简介

① 2008 年10 月有64 个算法正式向NIST提交了方案，经过初步评价，共有51个算法进入第一轮评估，主要对算法的安全性、消耗、和实现特点等进行分析；

② 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；

③ 2010年12月9日宣布，其中5个算法(JH、Grstl、Blake、Keccak和Skein)通过第二轮评审进入第三轮.

④ 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 Bertoai、Jean Daemen Daemen、Gilles Van Assche Assche与恩智半导体公司的Micha Michaëël Peeters联合设计的**Keccak算法.**

⑤ SHA-3成为NIST的新哈希函数标准算法(FIPS PUB 180--5).

⑥ Keccak算法的分析与实现详见：https://keccak.team/index.html

5.5 常用哈希函数简介

5.5.2. Keccak算法简介

- SHA-3的结构仍属于Merkle提出的迭代型哈希函数结构.
- 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，最终产生输出哈希值.
- 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哈希函数(固定输出长度)、伪随机数发生器，以及其他密码函数.

5.5 常用哈希函数简介

5.5.2. Keccak算法简介

2. Keccak算法描述

➢输入数据没有长度限制；

➢输出哈希值的比特长度分为：224，256，384，512.

b/比特	*r*/比特	*c*/比特	输出长度/比特	安全级别/比特
1600	1152	448	224	112
1600	1088	512	256	128
1600	832	768	384	192
1600	576	1024	512	256

图5.20. Keccak算法的参数定义

(1) 符号与函数

Keccak算法使用以下符号与函数：

①符号

r：比特率(比特 rate)，其值为每个输入块的长度.

c：容量(capacity)，其长度为输出长度的两倍.

b：向量的长度，$b=r+c$，而*b*的值依赖于指数*I*，即$b=25\times2^{I}$.

5.5 常用哈希函数简介

5.5.2. Keccak算法简介

②函数

Keccak算法用到了以下5个函数：**ϑ(theta)、ρ(rho)、π(pi)、χ(chi)、ι(iota)**

$$\theta : a[x][y][z] \leftarrow a[x][y][z] + \sum_{y'=0}^{4} a[x-1][y'][z] + \sum_{y'=0}^{4} a[x+1][y'][z-1]$$

$$\pi : a[x][y] \leftarrow a[x'][y'], \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

$$\chi : a[x] \leftarrow a[x] + (a[x+1]+1)a[x+2]$$

$$\rho : a[x][y][z] \leftarrow a[x][y][z-(t+1)(t+2)/2],$$

with t satisfying $0 \le t \le 24$ and

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^{t} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$$

in $GF(5)^{2\times 2}$, or $t = -1$, if $x = y = 0$

$$\iota : a \leftarrow a + RC[i]$$

5.5 常用哈希函数简介

5.5.2. Keccak算法简介

(2) 算法描述

Keccak算法 对数据进行填充，然后迭代压缩生成哈希值.

①填充

对数据填充的目的是使填充后的数据长度为*r*的整数倍.因为迭代压缩是对***r***位数据块进行的，如果数据的长度不是***r***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：$l + 1 + k = r\text{-}1 \bmod r$；
- 然后再添加比特“1”添加到末尾. **填充后的消息*m* 的比特长度一定为*r*的倍数.**

5.5 常用哈希函数简介

5.5.2. Keccak算法简介

以算法Keccak-256，信息“abc”为例显示补位的过程. a, b, c对应的ASCII码分别是97, 98, 99；于是原始信息的二进制编码为：01100001 01100010 01100011.此时***r*** = 1088.

① 补一个“1”： 01100001011000100 1100011 **1**

② 补**1062**个“0”：01100001 01100010 01100011 **10000000 00000000 … 00000000**

③ 补一个“1”，得到**1088**比特的数据：

5.5 常用哈希函数简介

5.5.2. Keccak算法简介

②整体描述

Keccak算法采用海绵结构(Sponge Construction)，在预处理(padding并分成大小相同的块)后，海绵结构主要分成两部分：

- 吸入阶段(Absorbing Phase)：将块x_i传入算法并处理.
- 挤出阶段(Squeezing Phase)：产生一个固定长度的输出.

Keccak算法的整体结构如下图：

5.5 常用哈希函数简介

5.5.2. Keccak算法简介

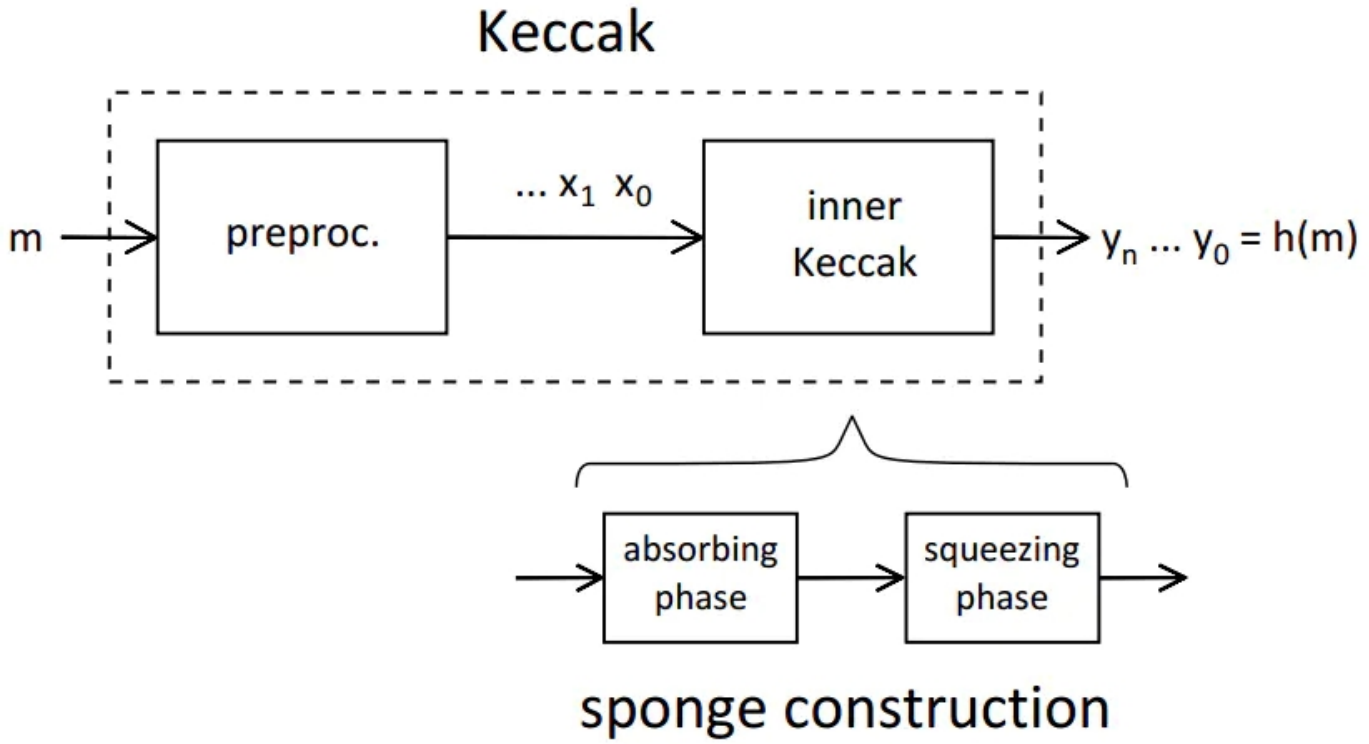


图5.21. Keccak算法的整体运算示意图

5.5 常用哈希函数简介

5.5.2. Keccak算法简介

③吸入与挤出阶段

给定输入串x ，首先对x做padding，使其长度能被***r***整除，将padding后分割成长度为***r***的块，即$x=x_0\|\| x_1\|\| x_2\|\|...\|\| x_{t-1}$.然后执行以下吸入阶段和挤出阶段：

1. 初始化一个长度为***r***+**c** 比特的全零向量.
2. 输入块x_i，将x_i和向量的前***r***个比特做异或运算，然后输入到 ***f*** 函数中处理.
3. 重复上一步，直至处理完x中的每个块.
4. 输出长为***r***的块作为y_0，并将向量输入到 ***f*** 函数中处理，输出y_1，以此类推.得到的哈希序列即为$y= y_0\|\| y_1\|\| y_2\|\|...\|\| y_u$.在Keccak-224/256/384/512中，只需要在y_0中取出**前224/ 256/ 384/ 512**位即可.

5.5 常用哈希函数简介

5.5.2. Keccak算法简介

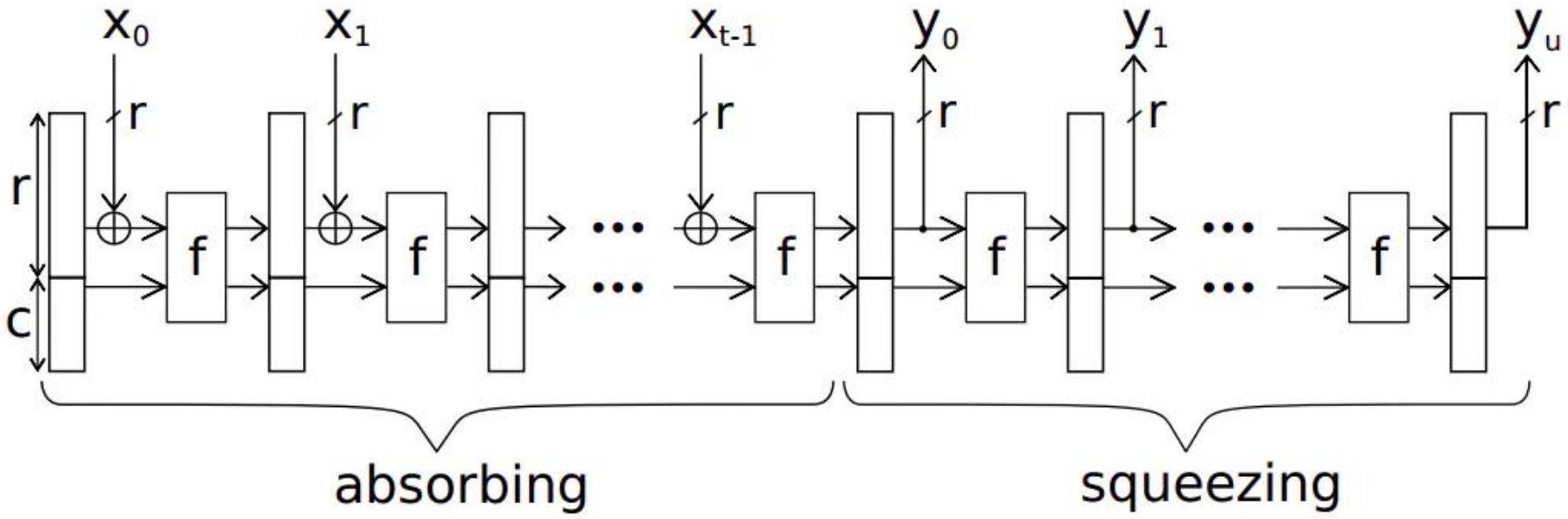


图5.22. Keccak算法的吸入阶段和挤出阶段示意图

5.5 常用哈希函数简介

5.5.2. Keccak算法简介

④压缩函数

- 压缩函数 ***f*** 是Keccak算法的核心，它包含n_r轮.
- n_r的取值与我们之前计算b时用到的指数I ($b=25\times2^I$)有关，具体地， $\boldsymbol{n_r=12+2*I}$.Keccak-224/256/384/512中，取I=6，因此n_r=24.
- 在每一轮中，要以此执行五步，即θ(theta)、ρ(rho)、π(pi)、χ(chi)、ι(iota).
- 在处理过程中，我们把b=1600个比特排列成一个5*5*w的三维数组，其中$w=2^I$=64比特，如右图所示：

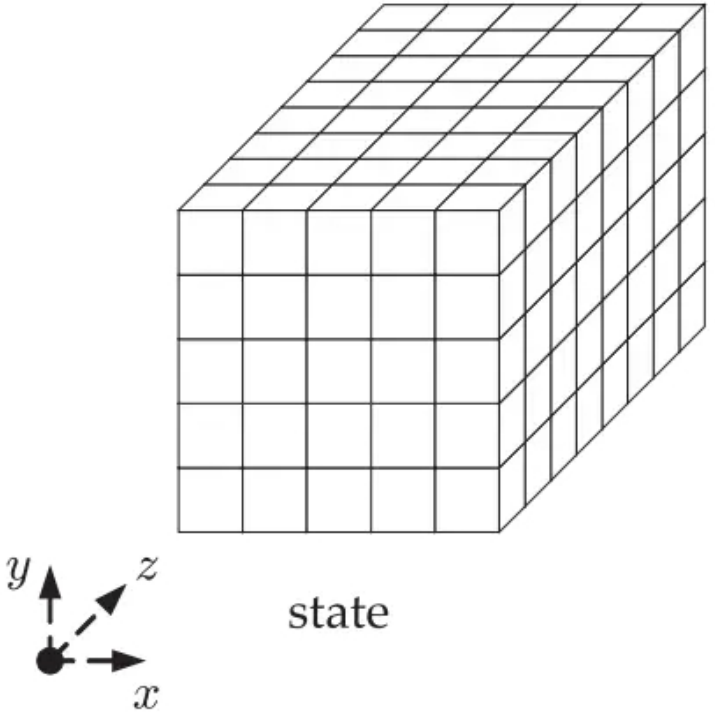


图5.23. Keccak算法的三维数组示意图

5.5 常用哈希函数简介

5.5.2. Keccak算法简介

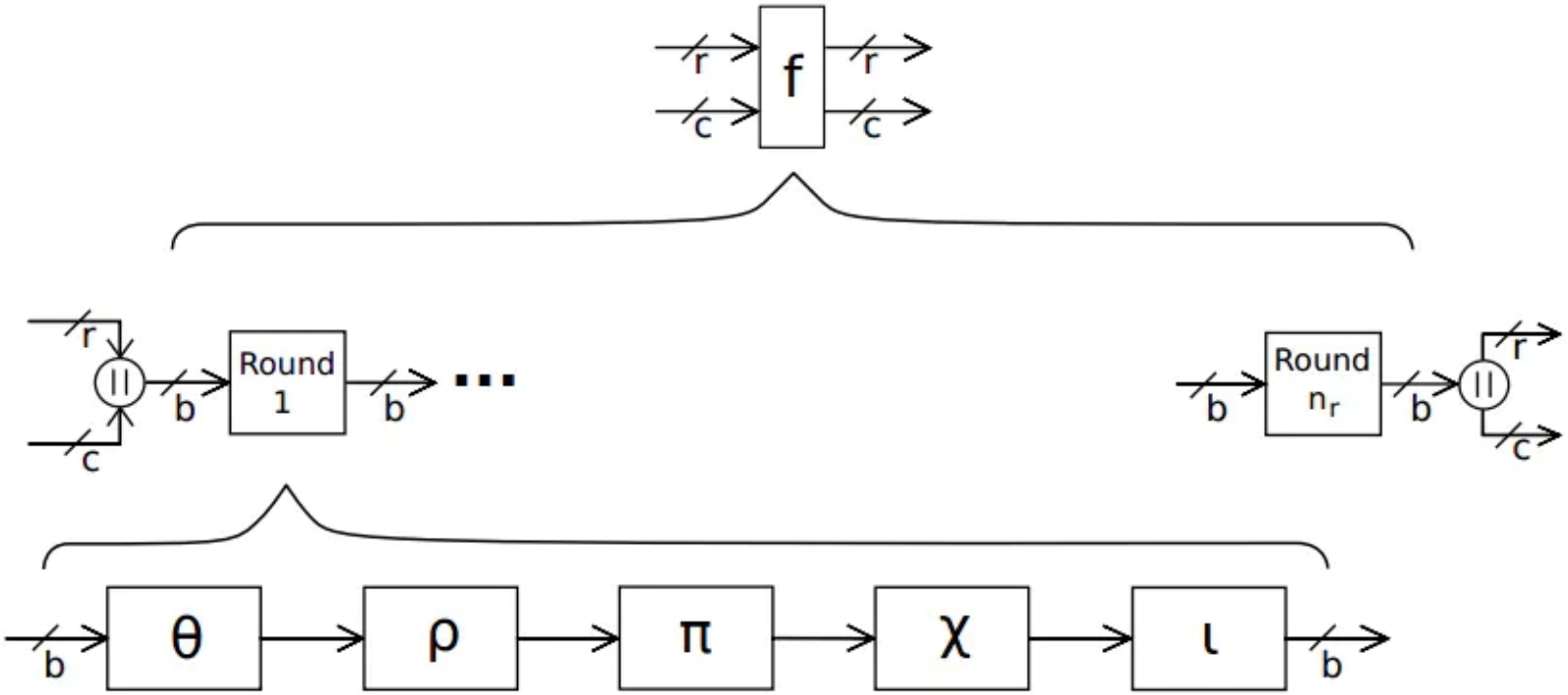


图5.24. Keccak算法的压缩函数结构示意图

5.5 常用哈希函数简介

5.5.2. Keccak算法简介

(3) 安全性与性能

- 安全性
  - ✓ 可以抵御对哈希函数的所有现有攻击.
  - ✓ 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，能够适应哈希函数的各种应用.
- 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！

目　录

5.5 常用哈希函数简介

5.5.3. SM3算法简介

1. 概况

- SM3是我国商用密码管理局颁布的商用密码哈希函数
- 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哈希函数.

5.5 常用哈希函数简介

5.5.3. SM3算法简介

2. SM3算法描述

- 输入数据长度为l比特，$1 \le l \le 2^{64}-1$；
- 输出哈希值的长度为256比特.

(1) 常量与函数

SHA-256算法使用以下常数与函数：

①常量

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

常数$T = \begin{cases} 79cc4519, & 0 \le j \le 15 \\ 7a879d8a, & 16 \le j \le 63 \end{cases}$

5.5 常用哈希函数简介

5.5.3. SM3算法简介

②函数

➢布尔函数

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

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

➢置换函数

$$P_0(X)=X\oplus(X<<<9)\oplus(X<<<17)$$
$$P_1(X)=X\oplus(X<<<15)\oplus(X<<<23)$$

其中：∧表示按位“与”；∨表示按位“或”；¬表示按位“补”；⊕表示按位“异或”；

<<<表示循环左移；

5.5 常用哈希函数简介

5.5.3. SM3算法简介

(2) 算法描述

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.**

5.5 常用哈希函数简介

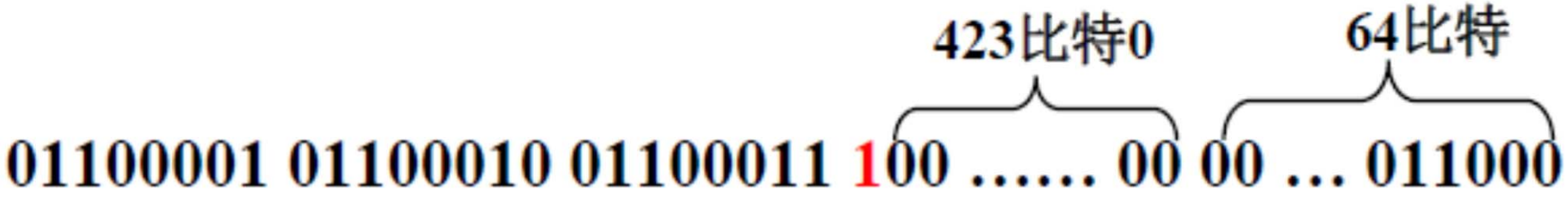
5.5.3. SM3算法简介

以信息“abc”为例显示补位的过程.a, b, c对应的ASCII码分别是97, 98, 99；于是原始信息的二进制编码为：01100001 01100010 01100011.

① 补一个“1”： 01100001 01100010 01100011 **1**

② 补423个“0”：01100001 01100010 01100011 **10000000 00000000 … 00000000**

③ 补比特长度24 (64位表示)，得到512比特的数据：



5.5 常用哈希函数简介

5.5.3. SM3算法简介

②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，…，W_{67}，W'_0，W'_1，…，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：

a) 将消息分组$B^{(i)}$划分为16个字W_0，W_1，…，W_{15}.

b) FOR j=16 TO 67

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

END FOR

c) FOR j=0 TO 63

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

END FOR

5.5 常用哈希函数简介

5.5.3. SM3算法简介

③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

a) FOR $i = 0$ TO n-1

b) $V^{(i+1)}= CF(\ V^{(i)},\ B^{(i)}\ )$

c) 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的哈希值.

5.5 常用哈希函数简介

5.5.3. SM3算法简介

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：$V^{(i+1)} = CF(V^{(i)}, B^{(i)}), 0 \leq i \leq n-1$.
- 压缩函数CF的压缩处理：//内层迭代

a) FOR j=0 TO 63

b) $CF = F(SS1, SS2, TT1, TT2, A, B, C, D, E, F, G, H, W_j, W'_j)$ //基本压缩函数

c) ENDFOR

5.5 常用哈希函数简介

5.5.3. SM3算法简介

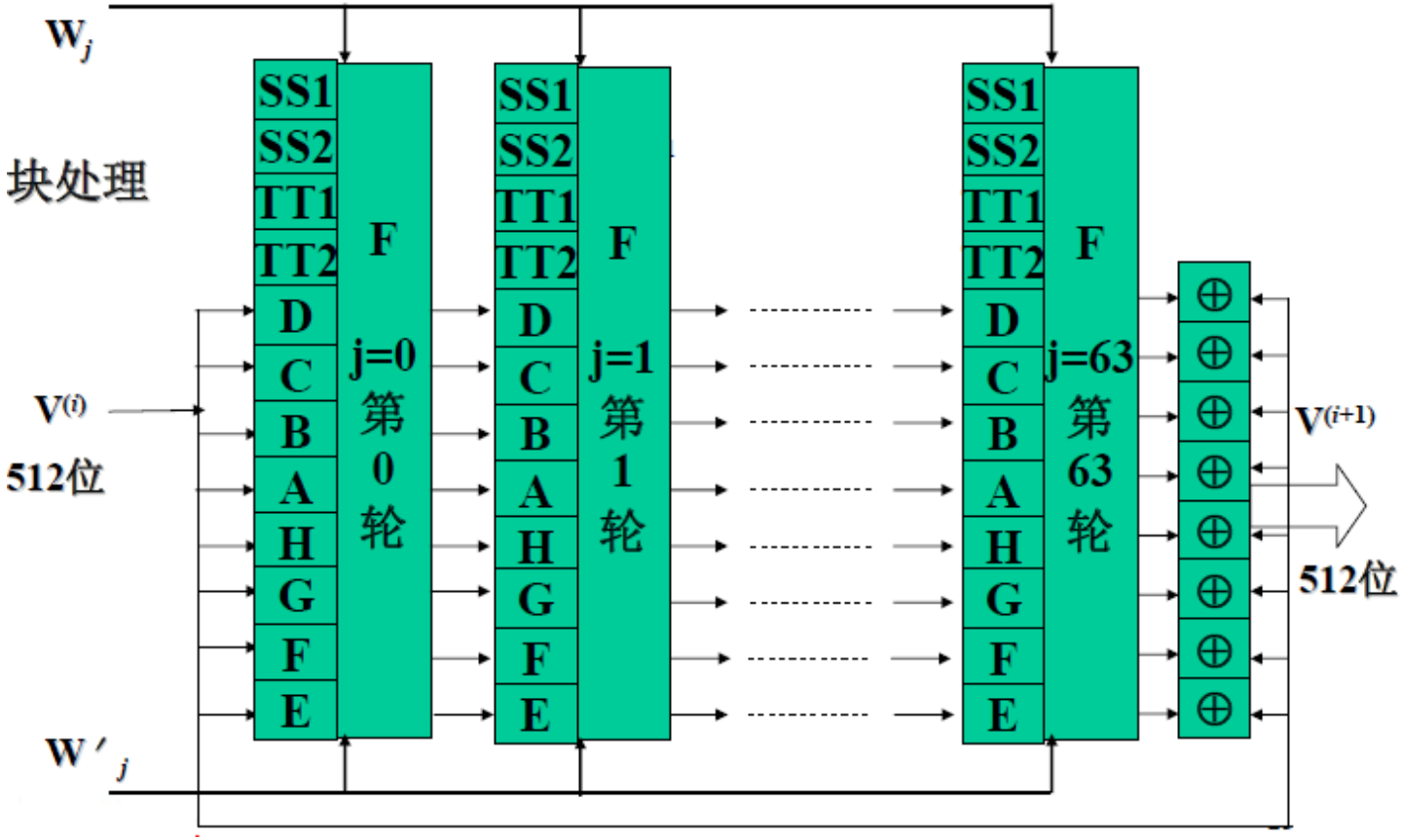


图5.25. SM3算法的迭代压缩流程示意图

5.5 常用哈希函数简介

5.5.3. SM3算法简介

⑤基本压缩函数**F**

1. $SS1 \leftarrow ((A <<< 12) + E + (T_j <<< j)) <<< 7$
2. $SS2 \leftarrow SS1 \oplus (A <<< 12)$
3. $TT1 \leftarrow \mathbf{FF}_j(A, B, C) + D + SS2 + W_j'$
4. $TT2 \leftarrow \mathbf{GG}_j(E, F, G) + H + SS1 + W_j$
5. $D \leftarrow C$
6. $C \leftarrow B <<< 9$
7. $B \leftarrow A$
8. $A \leftarrow TT1$
9. $H \leftarrow G$
10. $G \leftarrow F <<< 19$
11. $F \leftarrow E$
12. $E \leftarrow \mathbf{P0}(TT2)$

注意：

- ***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.**

5.5 常用哈希函数简介

5.5.3. SM3算法简介

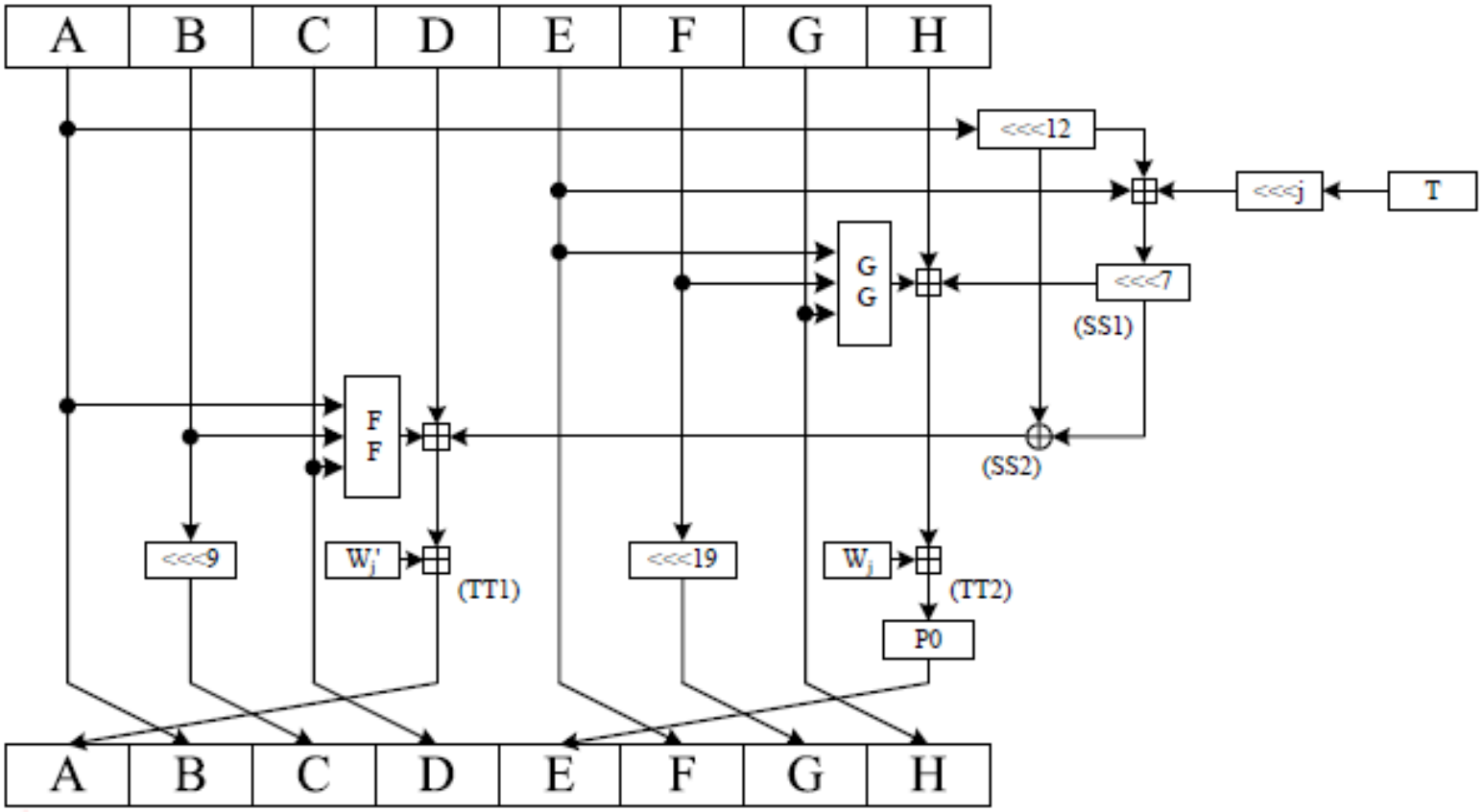


图5.26. SM3算法的基本压缩函数示意图

5.5 常用哈希函数简介

5.5.3. SM3算法简介

⑥SM3工作全过程

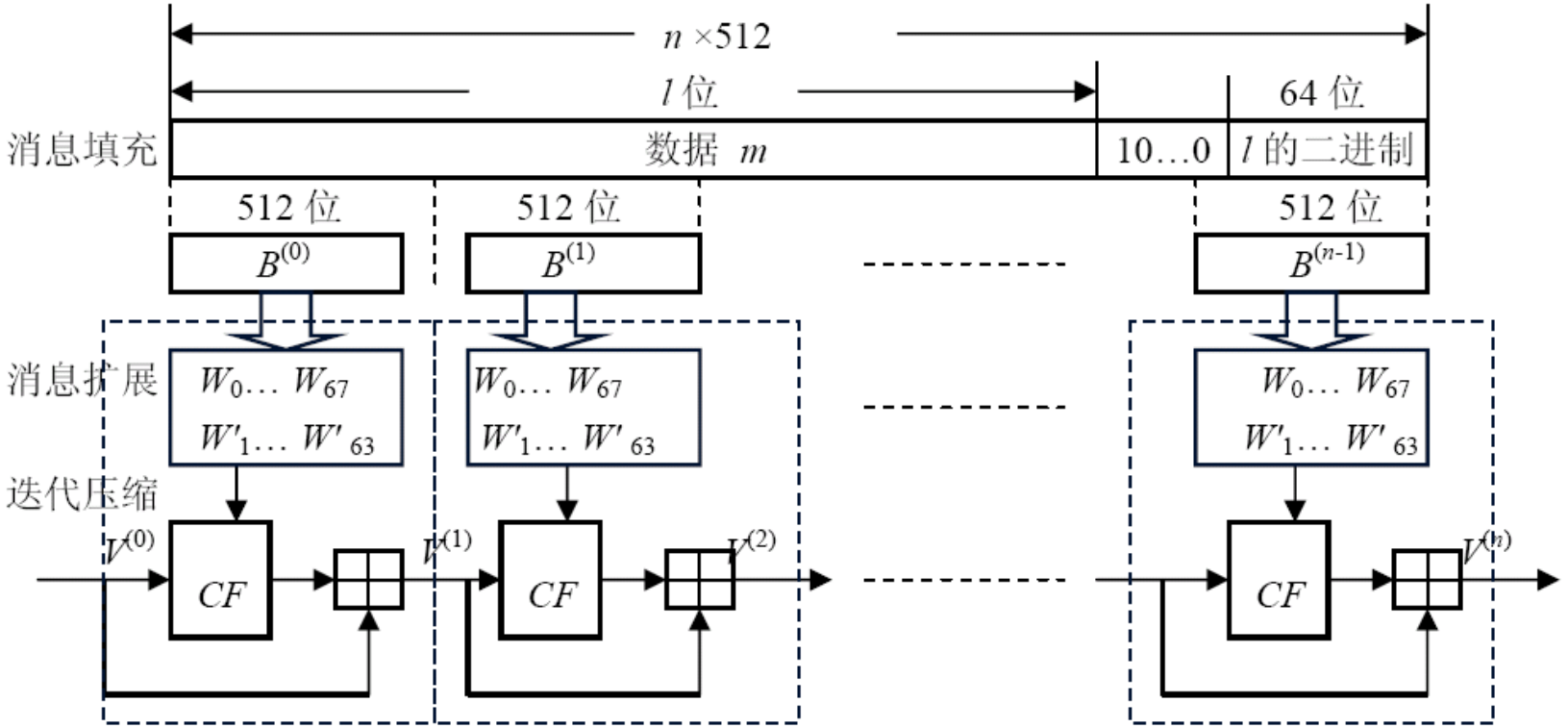


图5.27. SM3算法的工作全过程示意图

5.5 常用哈希函数简介

5.5.3. SM3算法简介

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位的哈希值.
- 第二个作用是提供安全性.在SM3的压缩函数CF中，布尔函数$\mathbf{FF}_j(X,Y,Z)$和$\mathbf{GG}_j(X,Y,Z)$是非线性函数，经过循环迭代后提供混淆作用.置换函数$\mathbf{P0}(X)$和$\mathbf{P1}(X)$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.

5.5 常用哈希函数简介

5.5.3. SM3算法简介

⑶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；

目　录

5.6 哈希函数在区块链中的应用

1. 以太坊用户地址的生成

第一步：生成私钥 (private key)

产生的256比特随机数做为私钥(256比特　16进制32字节)：　18e14a7b 6a307f42 6a94f811 4701e7c8 e774e7f9 a47e2c20 35db29a2 06321725

第二步：生成公钥 (public key)

① 利用将私钥(32字节)和椭圆曲线ECDSA-secp256k1计算公钥(65字节)(前缀04||X公钥||Y公钥)：　04 ||50863ad6 4a87ae8a 2fe83c1a f1a8403c b53f53e4 86d8511d ad8a0488 7e5b2352 || 2cd47024 3453a299 fa9e7723 7716103a bc11a1df 38855ed6 f2ee187e 9c582ba6

② 利用Keccak-256算法计算公钥的 哈希值(32bytes)：　fc12ad81 4631ba68 9f7abe67 1016f75c 54c607f0 82ae6b08 81fac0ab eda21781

③ 取上一步结果取后20bytes即以太坊地址：　1016f75c54c607f082ae6b0881fac0abeda21781

第三步：输地址 (address)

0x1016f75c54c607f082ae6b0881fac0abeda21781

5.6 哈希函数在区块链中的应用

2. 默克尔哈希树

比特币区块包含了区块头部和一些比特币交易. 一个区块上**所有交易的哈希值构成了该区块默克尔哈希树的叶子结点**，默克尔哈希树的根节点保存在区块头里面，因此所有交易与区块头部绑定在了一起.

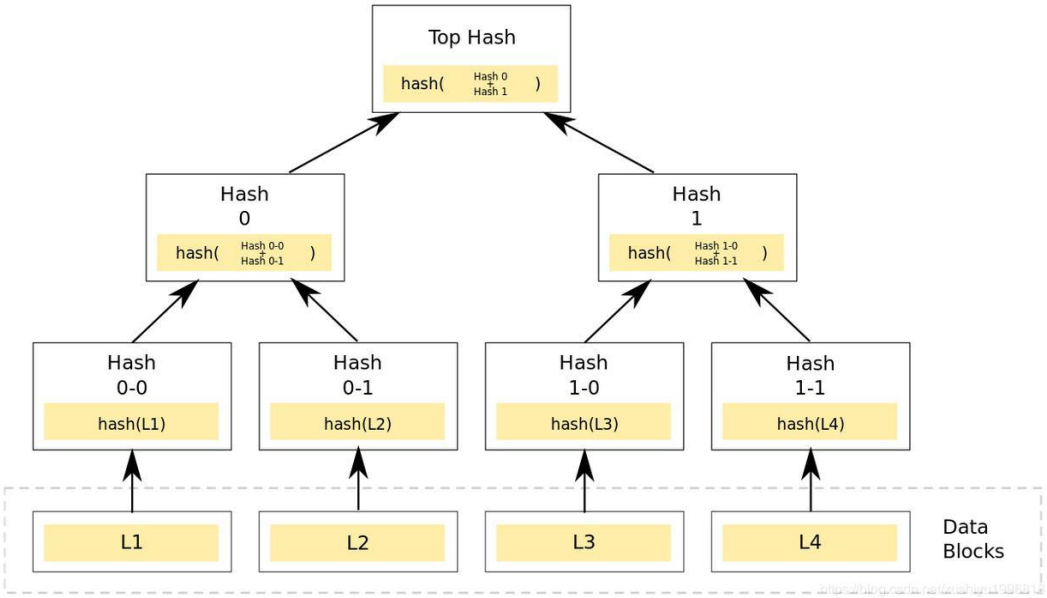


图5.28. 默克尔树结构示意图

5.6 哈希函数在区块链中的应用

3. 挖矿难度的设置

比特币难度是对挖矿困难程度的度量，即指：计算符合给定目标的一个哈希值的困难程度.

difficulty = difficulty_1_target / current_target

difficulty_1_target的长度为256比特， 前32位为0， 后面全部为1 ，一般显示为哈希值，

0x00000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

difficulty_1_target表示btc网络最初的目标哈希. current_target是当前块的目标哈希，先经过压缩然后存储在区块中，区块的哈希值必须小于给定的目标哈希值，表示挖矿成功.

5.6 哈希函数在区块链中的应用

4. 数字签名

比特币需要利用公钥进行加锁，利用私钥签名进行解锁，从而实现数字货币的交易. 解锁过程实际上是利用**ECDSA**算法的产生数字签名. 给定交易信息m，签名过程如下：

①选择一个随机数k;

②计算点$R= k*G= (x_R, y_R)$，计算$r = x_R \bmod n$;

③利用私钥d计算$s = k^{-1} * ((\boldsymbol{H(m)} - d * r)) \bmod n$;

④输入签名(r, s).

5.6 哈希函数在区块链中的应用

5. 软件发布

有关 cgminer_4.7.0-2_armel.deb 的更多信息:

实际大小	443134 字节（432.7 kB）
MD5 校验码	cf91c4ec74a8c6951c822039e667cf6f
SHA1 校验码	638f2cb5e6dc661f03b0de6c61337334f7fb5735
SHA256 校验码	b2f44e5f220de123e7fda4ba50ed0a06fd366fa4acbe104a00daf0355476669a

图5.29. 挖矿软件发布信息示意图

谢谢！