

# Lec 09. 消息认证和 Hash 函数

**认证：**是防止主动攻击的重要技术，对开发系统安全性有重要作用。认证的主要目的包括：

- 实体认证（发送者非冒充）；
- 消息认证（验证信息的完整性）。

**消息认证：**验证所收到的消息确实来自真正的发送方，且未被修改。可以看作两层

- 产生消息认证符的函数；
- 使用认证函数作为原语来验证消息真实性的协议。

**保密和认证**是信息系统安全的两个方面，但他们是两个不同属性的问题，认证不能自动提供保密性，而保密性也不能自然提供认证功能。

**认证函数：**三类产生认证符的函数：

- 报文加密：以整个报文的密文为认证码（背后思想：消息可读）；
- 报文认证码 (MAC)：以一个报文的公共函数和用于产生一个定长值的密钥作为认证符；
- 散列函数：一个将任意长度的报文映射为定长的散列值的公共函数，以散列值作为认证符。

**消息加密提供认证：对称加密**

- 对称加密提供认证：用户 A 为发信方，用户 B 接受到信息后，通过解密来判决信息是否来自 A、信息是否是完整的、有无窜扰。
- A 向 B 发送  $E_K(M)$  即可。
- 提供一定程度的认证：
  - 仅来自 A；
  - 传输过程不会被更改（需要加密方式有 CCA 安全性）；
  - 需要某种结构或冗余，以识别合法明文。
- 不提供签名
  - 接受者可以伪造密文；
  - 发送者可以否认报文（发送者声称是接受者伪造）。
  - **原因：收发双方地位完全对等。**

**消息加密提供认证：公钥加密**

1. 发送方用对方的公钥  $PK$  加密消息  $M$ ，接收方用自己的私钥  $SK$  解密消息。提供了保密性但不提供认证性。
  - 完全没有消息源的认证——任何人都能发送；
  - 没有消息完整性的认证——任何人能对任意消息加密发送。
2. 发送方用自己的私钥  $SK$  签名消息  $M$   $\text{Sig}(M, SK)$ ，接收方用发送方的公钥  $PK$  验证签名  $\text{ver}(C, PK)$ （数字签名）。提供了认证性但不保密性。
3. 发送方先用自己的私钥签名消息以提供认证，然后使用接收方公钥加密消息提供保密性（先签名，再用公钥加密）。（选择正确加密算法即可）既提供认证性也提供保密性。缺点是效率不高，一种解决方式为“签密”。

**散列函数**（又称为杂凑函数、消息摘要函数、哈希函数等）：将任意长度的消息映射成一个较短的定长输出消息的函数，即  $h = H(M)$ ,  $M$  是变长消息,  $h$  是定长的散列值； $H(\cdot)$  为公开函数；散列函数的目的是为文件、报文或其他的分组数据产生“数字指纹”。

**散列函数实现消息认证**：以下协议都有一定依赖于  $H$  的安全前提。

1. 假设 A 和 B 共享密钥 K。A 向 B 发送  $E(M \parallel H(M), K)$ 。
  - 提供保密性（仅 A 与 B 共享 K）；
  - 提供认证性（加密保护  $H(M)$ ）。
2. 假设 A 和 B 共享密钥 K。A 向 B 发送  $M \parallel E(H(M), K)$ 。
  - 提供认证性（加密保护  $H(M)$ ），但不提供保密性；
  - 可以看做是一个 MAC。
3. 假设 A 的公私钥对为  $PK, SK$ 。A 向 B 发送  $M \parallel \text{Sig}(H(M), SK)$ 。
  - 提供消息认证和数字签名（加密保护  $H(M)$ ，且仅 A 能生成  $\text{Sig}(H(M), SK)$ ），但不提供保密性；
  - 即对“指纹”签名。
4. 假设 A 的公私钥对为  $PK, SK$ 。A 和 B 共享会话密钥 K。A 向 B 发送  $E(M \parallel \text{Sig}(H(M), SK), K)$ 。
  - 提供消息认证和数字签名；
  - 提供保密性（仅 A 和 B 共享 K）。
5. 该方法不使用加密函数，但假设 A 和 B 共享某种秘密 S，则 A 向 B 发送  $M \parallel H(M \parallel S)$ 。
  - 提供消息认证（仅 A 和 B 共享 S）。
6. 假设 A 和 B 共享会话密钥 K，同时 A 和 B 共享某种秘密 S，则 A 向 B 发送  $E(M \parallel H(M \parallel S), K)$ 。
  - 提供消息认证（仅 A 和 B 共享 S）；
  - 提供保密（仅 A 和 B 共享 K）。

## 散列函数的特点

- 适用范围： $H$  能用于任何大小的数据分组； $H$  能产生定长输出；
- 数学性质：
  - 对于任意给定的  $x$ ,  $H(x)$  要相对易于计算，使得软硬件实现都实际可行；
  - **单向性**：对任意给定的码  $h$ , 寻求  $x$  使得  $H(x) = h$  在计算上不可行；
  - **弱抗碰撞性**：任意给定消息  $x$ , 寻求不等于  $x$  的  $y$ , 使得  $H(y) = H(x)$  在计算上不可行。
  - **强抗碰撞性**：寻求  $(x, y)$  对使得  $H(x) = H(y)$  在计算上不可行。

## 散列函数算法

- 简单的散列函数算法：很容易找到碰撞，但在模式匹配中有用。
- 安全的散列算法：MD 系列、SHA 系列。

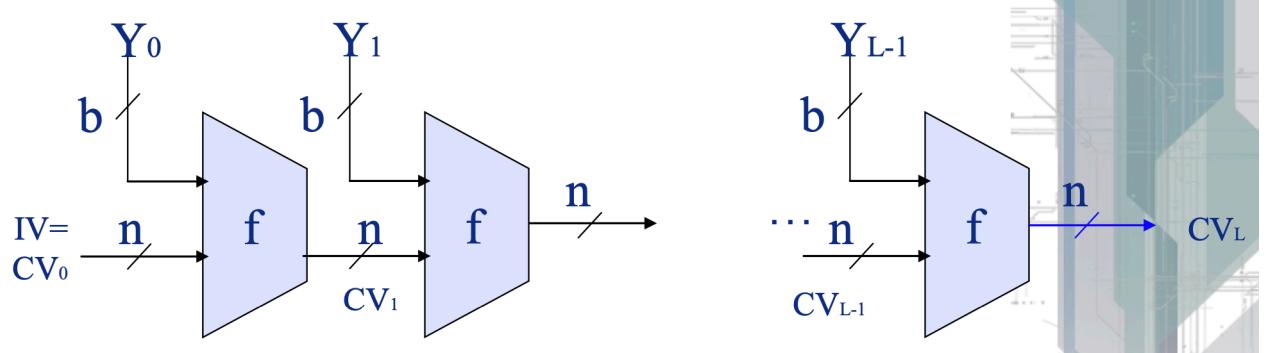
## 简单的散列函数算法

- 简单的杂凑函数：
  - 将明文  $M$  进行分组得到  $B_1, B_2, \dots, B_m$ , 每一分组为  $n$  比特,  $B_i = \{b_{i1}, b_{i2}, \dots, b_{in}\}$ ；每个分组按比特异或，得到  $n$  位的哈希值，其中第  $i$  位为  $C_i = b_{i1} \oplus b_{i2} \oplus \dots \oplus b_{im}$ ；是简单的奇偶校验。不具有哈希函数的三个算法性质（末尾添 0）。

- 改进：将  $n$  比特的散列值先预置为 0；按如下方式依次将数据分组：先将当前的散列值循环左移一位，再将数据分组与散列值异或形成新的散列值。将输入数据完全随机化，并且将输入中的数据格式掩盖掉。**不具有哈希函数的三个算法性质**（起始添 0）。

**Merkle-Damgard 结构：**设  $Y_0 Y_1 \dots Y_{L-1}$  为输入消息； $Y_i$  表示第  $i$  个数据块，长度为  $b$  比特； $IV$  是长度  $n$  比特的初始值； $CV_i$  表示第  $i$  次迭代的链接值。 $f$  是压缩算法， $n$  是散列码的长度， $b$  是输入分块的长度。于是，满足 Merkle-Damgard 结构的散列函数可定义为：

$$\begin{aligned} CV_0 &= IV \\ CV_i &= f(CV_{i-1}, Y_{i-1}) \quad (1 \leq i \leq L) \\ H(M) &= CV_L \end{aligned}$$



- Merkle 与 Damgard 在 1989 年证明，如果压缩函数  $f$  具有抗碰撞能力，那么 MD 结构的迭代 Hash 函数也具有抗碰撞能力；因此现有的 Hash 函数大多使用上述迭代结构。
- 于是，设计安全的散列函数可以归纳为设计输入定长，且具有抗碰撞能力的压缩函数  $f$ ；
- 对 Hash 的密码分析集中在对  $f$  的内部结构分析，并试图找到  $f$  的碰撞，且需要分析轮与伦之间信息变换的规律；
- 但在第三代的安全 Hash 算法 (SHA-3) 中，人们倾向于放弃 MD 结构。

### 对散列函数的生日攻击

- 弱抗碰撞性：**给定一个 Hash 函数  $H$  和一个 Hash 值  $H(x)$ ，设  $H$  的输出长度为  $m$  比特，即共有  $2^m$  个 Hash 码。将  $H$  作用于  $k$  个随机的输入，那么满足这些输入中至少有一个  $y$ ，使得  $H(y) = H(x)$  的概率为 0.5 的  $k$  值为  $2^{m-1}$ 。
- 强抗碰撞性：**给定一个 Hash 函数  $H$  和一个 Hash 值  $H(x)$ ，设  $H$  的输出长度为  $m$  比特，即共有  $2^m$  个 Hash 码。将  $H$  作用于  $k$  个随机的输入得到集合  $X$ ，将  $H$  作用于另外  $k$  个随机输入得到集合  $Y$ 。则两个集合至少有一个匹配 ( $\exists x \in X, y \in Y$  使得  $H(y) = H(x)$ ) 的概率为 0.5 的  $k$  值为  $2^{m/2}$ 。
- 简单的说，对输出长度为  $m$  的 Hash 函数，值  $2^{m/2}$  决定了该 Hash 函数抗穷举攻击的能力。
- 一个密码学上安全的散列函数，输出长度一定是长的！

**MD5 散列算法：**“目前”最为广泛使用的“安全”散列函数；在 2005 年王小云教授发现找到 MD5 碰撞的快速算法。算法输入任意长度的消息，输出 128 位哈希输出；以 512 位输入数据块为单元，使用了 Merkle-Damgard 结构。

$$\begin{aligned} CV_0 &= IV \\ CV_{q+1} &= f(CV_q, Y_q) = \text{SUM}_{32}(CV_q, RF_I[Y_q, RF_H[Y_q, RF_G[Y_q, RF_F[Y_q, CV_q]]]]) \\ MD &= CV_L \end{aligned}$$

其中， $IV$  为  $ABCD$  的初始值， $Y_q$  表示消息的第  $q$  个 512 位数据块， $CV_q$  表示链接变量，用于第  $q$  个数据块的处理； $RF_x$  使用基本逻辑函数  $x$  一轮的功能函数； $\text{SUM}_{32}$  表示分别按 32 位字计算的模  $2^{32}$  加法结果； $MD$  表示最终结果。

## ● 具体步骤

1. **添加填充位** (1个1与若干个0)。在消息的最后添加适当的填充位使数据位的长度满足与448在模512下同余(剩下64位需要记录长度)。
2. **添加长度**: 原始消息长度(二进制位的个数), 用64位表示。如果长度超过 $2^{64}$ 位, 则仅取长度的最低64位。至此, 我们得到了一个长度为512位的整倍数的消息。可以表示成 $L$ 个512位的数据块 $Y_0, Y_1, \dots, Y_{L-1}$ , 其长度为 $L \times 512$ 比特。令 $N = 16L$ , 则长度为 $N$ 个32位的字。令 $M[0, \dots, N-1]$ 表示以字为单位的消息表示。
3. **初始化MD缓冲区**: 一个128位MD缓冲区用以保存中间和最终散列函数的结果。它可以表示为4个32位的寄存器( $A, B, C, D$ ), 寄存器初始为固定的16进制值(如下所示)并使用低端形式(little-endian)存储。

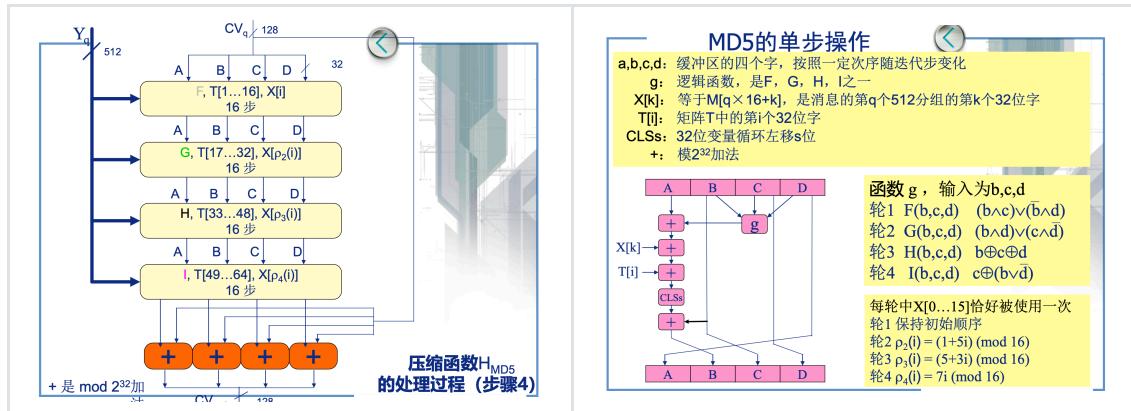
```

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

```

4. **处理消息块**: 一个压缩函数是本算法的核心, 其包括4轮处理。四轮处理具有相似的结构, 但每次使用不同的基本逻辑函数, 记为 $F, G, H, I$ 。每一轮以当前的512位数据块 $Y_q$ 和128位缓冲值 $A, B, C, D$ 作为输入, 并修改缓冲值的内容。每次使用64元素表 $T[1, \dots, 64]$ 的四分之一。

- **$T$ 表**: 由 $\sin$ 函数构造而来, 第*i*个元素表示为 $T[i]$ , 其值等于 $2^{32}|\sin(i)|$ , 其中*i*是弧度。因此 $T$ 表提供了随机化的32位模板, 消除了在输入数据中的任何规律性的特征。



5. **输出结果**: 所有 $L$ 个512位数据块处理完毕后, 最后结果就是128位的消息摘要。

- $H_{MD5}$ 压缩函数共有64步, 每16步为1轮, 每步处理32比特。

## MD4 散列算法

- 设计目标: 安全性、速度、简明与紧凑、有利的小数在前的结构。
- MD4与MD5的区别:
  - MD4用3轮, 每轮16步; MD5用4轮, 每轮16步,
  - MD4中第一轮没有常量加; MD5的64步中的每一步用了不同的常量 $T_i$ 。
  - MD5中用了4个基本逻辑函数, 每轮1个; MD4用了三个;
  - MD5每轮加上前一步的结果, MD4没有。
- 现状:

- 经过  $2^6$  次运算就能找到 MD4 的压缩函数的碰撞；攻击 MD4 大约为  $2^{20}$  数量级；王小云攻击 MD5 的强抗碰撞性（大约  $2^{33}$  次找到碰撞）。对 MD5 的原像攻击目前仍没有使用的结果。
- 2008年出现 MD6（放弃 MD 结构），正参与竞选 SHA-3。

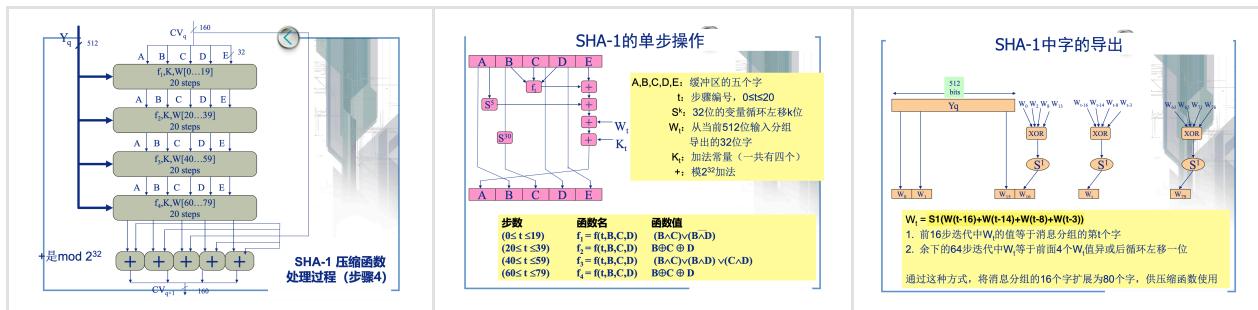
**SHA-1：**SHA 标准“安全哈希算法”由 NIST 开发，基于 MD4 设计。其输入最大长度为  $2^{64}$  位的消息（大于此无法证明安全）；输出 160 位消息摘要；输入以 512 位数据块为单位处理。2005 年王小云找到了 SHA-1 的碰撞（理论攻击）。

$$CV_0 = IV$$

$$CV_{q+1} = f(CV_q, Y_q) = \text{SUM}_{32}(CV_q, ABCDE_q)$$

$$MD = CV_L$$

其中， $IV$  为  $ABCD$  的初始值， $Y_q$  表示消息的第  $q$  个 512 位数据块， $CV_q$  表示链接变量，用于第  $q$  个数据块的处理； $ABCDE_q$  表示处理第  $q$  个消息分组时最后一轮的输出； $\text{SUM}_{32}$  表示分别按 32 位字计算的模  $2^{32}$  加法结果； $MD$  表示最终结果。



### ● 具体步骤

- 添加填充位：与 MD5 完全相同；
- 添加长度：与 MD5 相同，由于限制长度，所以 64 位一定可以存下；
- 初始化 MD 缓冲区：一个 160 位 MD 缓冲区用于保存中间和最终散列函数的结果，它可以表示为 5 个 32 位的寄存器 ( $A, B, C, D, E$ )。寄存器初始为固定的 16 进制值（如下所示，前四个与 MD5 相同）并使用高端形式 (big-endian) 存储。

```

A = 67 45 23 01
B = EF CD AB 89
C = 98 BA DC FE
D = 10 32 54 76
E = C3 D2 E1 F0

```

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

- 输出：全部  $L$  个 512 位数据块处理完毕后，输出 160 位消息摘要。

- 与 MD5 不相同的还有字的导出，MD5 每轮使用的是相同的字，而 SHA1 将字导出进行了进一步的复杂化。

### MD4, MD5, SHA-1 比较

	<b>MD4</b>	<b>MD5</b>	<b>SHA-1</b>
hash 值	128bit	128bit	160bit
分组处理长度	512bit	512bit	512bit
基本字长	32bit	32bit	32bit
步数	48 (3*16)	64 (4*16)	80 (4*20)
消息长	$\leq 2^{64}$	无	$\leq 2^{64}$
基本逻辑函数	3	4	4
常数个数	3	64 (T 表)	4
速度	-	1/7 MD4	3/4 MD4
缓冲区字数	4	4	5

**SHA-2 系列：**包括 SHA-256, SHA-384, SHA-512, 通称为 SHA-2，是 MD 系列。没有接受公众密码学研究者的检验，所以安全性的信任度低于 SHA-1，但人们对 SHA-1 也开始质疑；目前并没有找到 SHA-2 的弱点。

**SHA-3 系列：**最终选中基于 Sponge 函数的 Keccak 算法，输出长度分别是 224、256、384、512。

```

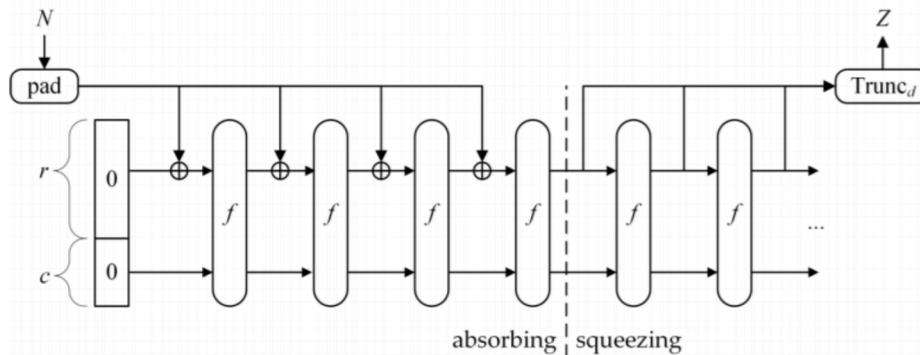
SHA3-224(M)=Keccak[448](M||01, 224)
SHA3-256(M)=Keccak[512](M||01, 256)
SHA3-384(M)=Keccak[768](M||01, 384)
SHA3-512(M)=Keccak[1024](M||01, 512)

```

其中， $[c]$  为 Keccak 容量，哈希算法中  $c = 2d$ ,  $d$  为输出长度。

- Keccak 的优势在于它与 SHA-2 设计上存在极大差异，适用于 SHA-2 的攻击方法将不能作用于 Keccak。

## Sponge 函数



其中， $N$  是输入， $r$  是一个比例， $c$  是安全级别，为  $d$  的两倍；pad 是一个填充策略，需要将输入填充为  $r$  的整数倍； $Z$  是输出， $d$  是输出长度， $f$  是输入输出均为定长  $b$  的函数。

**消息认证码 (MAC)**: 对选定报文  $M$ , 使用一个密钥  $K$ , 产生一个短小的定长数据分组, 称为认证码  $MAC$ , 并将它附加在报文中, 提供认证功能。 $MAC = C_k(M)$ , 其中  $M$  是可变长的报文,  $K$  是通信双方共享密钥,  $C_k(M)$  是定长的认证码。

- 应用认证码: 假设只有收发方知道密钥, 同时收到的 MAC 与计算得出的 MAC 匹配, 则有:
  - 确认报文未被更改;
  - 确信报文来自所谓的发送者;
  - 如果报文包含序号, 可确信该序号的正确性。
- 与对称加密的区别和联系:
  - 认证函数类似加密函数, 但它是不可逆的, 这个性质使其比加密函数更难破解。若 MAC 密钥长度  $k$  而输出长度  $n$ , 且  $k > n$ ; 如果攻击者穷举所有可能的 MAC 密钥, 共有  $2^k$  个结果; 但其中只有  $2^n$  个结果不同; 平均而言共有  $2^{k-n}$  个密钥都会产生正确的 MAC。因此攻击者为唯一确定密钥必须反复穷举缩小密钥空间。
  - MAC 不能用于实现保密性;
  - 认证函数并不提供数字签名。
- 性质: MAC 函数应由如下性质 (假定攻击者没有 K) :
  - 有  $M$  和  $C_K(M)$  试图生成  $M'$  使得  $C_K(M') = C_K(M)$ , 这在计算上不可行;
  - $C_K(M)$  应能均匀分布; 对于随机选取的报文  $M, M'$ ,  $C_K(M) = C_K(M')$  的概率为  $2^{-n}$ , 其中  $n$  为 MAC 的比特长度;
  - 报文  $M'$  为  $M$  的某种已知代换, 即  $M' = f(M)$ , 则  $C_K(M) = C_K(M')$  的概率为  $2^{-n}$ 。
- 为什么使用消息认证?
  - 适用于广播 (并不需要每个点都有密钥, 仅一个接受者验证消息真实性);
  - 报文加密解密工作量比较大;
  - 某些应用不关心报文的保密性而只关心报文的真实性;
  - 认证函数与保密函数的分离能提供结构上的灵活性 (可在网络协议的不同层次进行) ;
  - 认证码可延长报文的保护期限, 同时能处理报文内容 (使用加密时, 当报文解密后保护失效) 。
- 基本用途
  - 消息认证: A 向 B 发送  $M||C_K(M)$ ; 可提供认证, 因为只有 A、B 共享密钥  $K$ ;
  - 与对称加密结合, 实现保密性和认证性: A 向 B 发送  $E_{K_2}(M||C_{K_1}(M))$ ; 可提供认证性, 因为只有 A、B 共享密钥  $K_1$ ; 也可提供保密性, 因为只有 A、B 共享密钥  $K_2$ 。这里的认证与明文相关, 先对明文计算 MAC 再连同明文一起加密。
  - 与对称加密结合, 实现保密性和认证性: A 向 B 发送  $E_{K_2}(M)||C_{K_1}(E_{K_2}(M))$ ; 可提供认证性, 因为只有 A、B 共享密钥  $K_1$ ; 也可提供保密性, 因为只有 A、B 共享密钥  $K_2$ 。这里的认证与密文相关, 先加密, 再对密文计算 MAC。

**数据认证码 DAC (⚠ 不安全! )**: 基于 DES 的消息认证码。是使用最广泛的 MAC 算法之一; 使用了 DES 的 CBC 模式, 即

$$\begin{aligned} C_1 &= E_K(M_1) \\ C_2 &= E_K(M_2 \oplus C_1) \\ &\dots \\ DAC &= C_n = E_k(M_n \oplus C_{n-1}) \end{aligned}$$

- 局限性: 要求输入长度必须为  $mn$ , 其中  $m$  为固定值,  $n$  为分组加密的分组长度; 否则很容易出现攻击:

假设  $T = MAC(K, X)$  为先前的一次消息-tag 对，则  $T = MAC(K, X||(X \oplus T))$  为新的消息-tag 对。

- **改进：**基本思想：使用三个密钥。后来演化为 CMAC，可用于处理不定长的输入。
- **CBC-MAC** 在政府和工业界广泛采用，其限制有：长度限制（长度为定长）、初始向量限制（必须为全 0，否则可被攻破）。

**基于密文的消息认证码 CMAC：**使用三个密钥来克服，这些密钥可通过一个密钥导出，同样**不使用初始向量**！令  $L$  为 MAC 的比特长度，若消息长度是块的整数倍，则

$$\begin{aligned} C_1 &= E_K(M_1) \\ C_2 &= E_K(M_2 \oplus C_1) \\ &\dots \\ C_n &= E_K(M_n \oplus C_{n-1} \oplus K_1) \\ MAC &= MSB_L(C_n) \end{aligned}$$

否则，

$$\begin{aligned} C_1 &= E_K(M_1) \\ C_2 &= E_K(M_2 \oplus C_1) \\ &\dots \\ C_n &= E_K((M_n || 0111\dots) \oplus C_{n-1} \oplus K_2) \\ MAC &= MSB_L(C_n) \end{aligned}$$

其中， $MSB_L(X)$  表示  $X$  最左边的  $s$  位。

- 可支持任意长度——必须分长度是否为块的整数倍使用不同的密钥  $K_1$  或  $K_2$ ！否则将会通过填充攻破。
- 不使用初始向量！否则将会通过第一块攻破。

**带密钥的 Hash 函数 HMAC：**消息认证码 MAC 是基于分组加密算法定义的；应用中，由于散列函数的速度快而且代码易于获取，也希望基于散列函数设计的 MAC。

- **要求**

- 不必修改而直接使用现有的散列函数；
- 容易替代；
- 应保持散列函数的原有性能；
- 对密钥的使用和处理简单；
- 安全长度依赖于使用散列函数的强度。

最原始的想法是直接将  $K$  与  $M$  拼接（直接将初始向量  $IV$  看成  $K$ ），得到

$$HMAC = H(K||M)$$

由于发现其中弱点（由于 Merkle-Damgard 结构，攻击者直接在收集到的消息-tag 对的最后加入若干消息块并计算出 tag 即可）。因此，现有的 HMAC 结构为：

$$HMAC_K = H[(K^+ \oplus opad) || H[(K^+ \oplus ipad) || M]]$$

其中， $K^+$  是左边填充若干 0 至  $b$  比特长； $opad$  为特殊填充常数 00110110 重复  $b/8$  次； $ipad$  为特殊填充常数 01011100 重复  $b/8$  次。

- **正确性：**证明破解 HMAC 与破解 H 等价。

## 代表性 MAC 构造方式

- ECBC-MAC、CMAC：一般用 AES 实现；
- NMAC：是 HMAC 的基础；
- PMAC：并行 MAC；
- Carter-Wegman MAC：基于快速 one time mac。

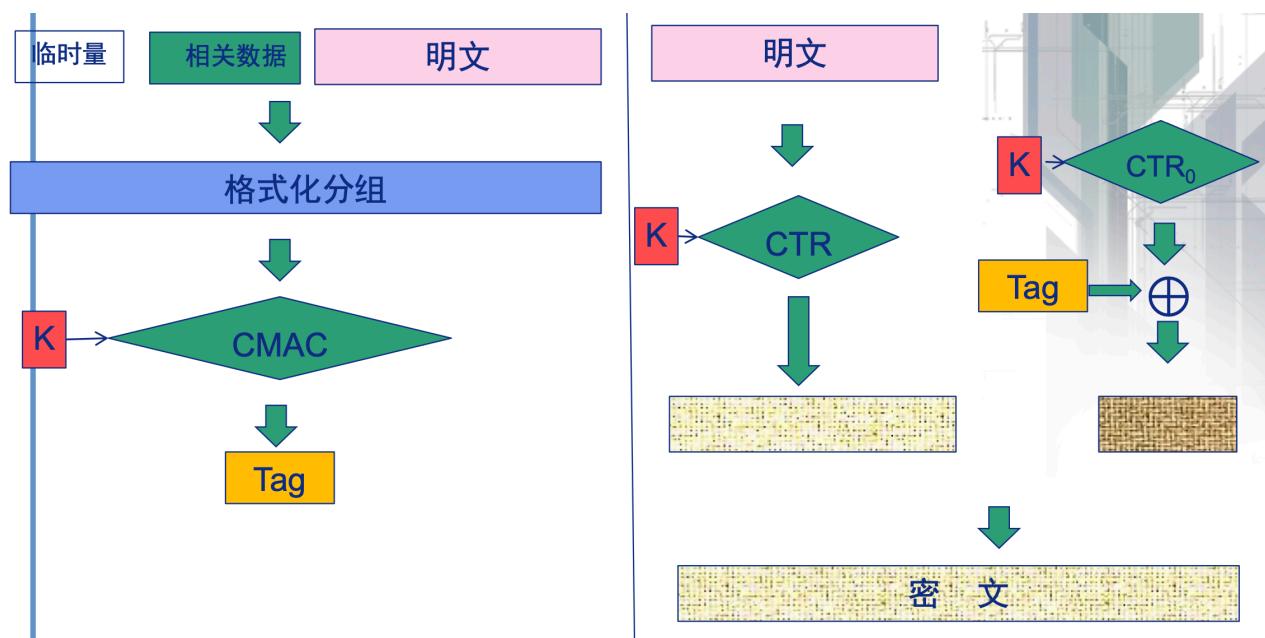
同时实现认证性与保密性：认证加密

- (HtE)：先哈希，再加密： $h = H(M), E_K(M||h)$ ；不安全！
- (MtE)：先 MAC，再加密： $T = MAC_{K_1}(M), E_{K_2}(M||T)$ ，用于 SSL/TLS 协议；不具有通用安全性。
- (E&M)：加密并且 MAC： $C = E_{K_2}(M), T = MAC_{K_1}(M)$ ，用于 SSH 协议；不具有通用安全性。
- (EtM)：先加密再 MAC： $C = E_{K_2}(M), T = MAC_{K_2}(C)$ ，用于 IPsec 协议；具有通用安全性。

认证性攻击模型：从以往消息-tag 对复原新消息-tag 对；

保密性攻击模型：选择明文攻击。

CCM 模式：属于 E&M，使用 AES、CTR、CMAC。



GCM 模式（属于 EtM）并行化的设计，高吞吐率、低成本、低延迟，基于变形的 CTR。（参考作业）