第4章 Hash函数

杨礼珍

提交作业Email：yanglizhen_exe@163.com
课件下载Email：yanglizhen_course@163.com，密码：tongjics

同济大学计算机科学与技术系, 2017

Outline

本章作业

课本习题4.6、4.7、4.12

思考题：课本习题4.10、习题4.13

4.1 Hash函数与数据完整性

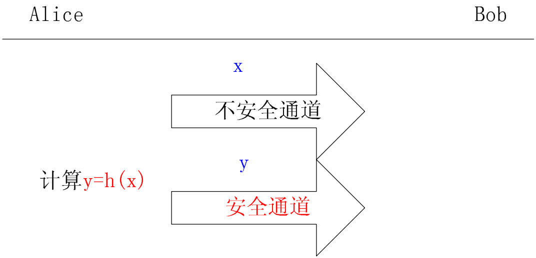
消息安全的基本需求：

- **机密性**：即不被许可的人无法知道消息的内容，由加密函数完成。
- **完整性**：即消息如被修改能够被发现，由hash函数完成。

Hash函数：提供数据完整性保护的密码函数，是消息的“指纹”，输出结果也称为消息摘要。Hash函数也称为单向散列函数。

4.1 Hash函数与数据完整性

Hash函数是如何工作的：假定h是一个hash函数。



$$y = h(x)?\begin{cases} \text{是：数据没修改过} \\ \text{否：数据被修改过} \end{cases}$$

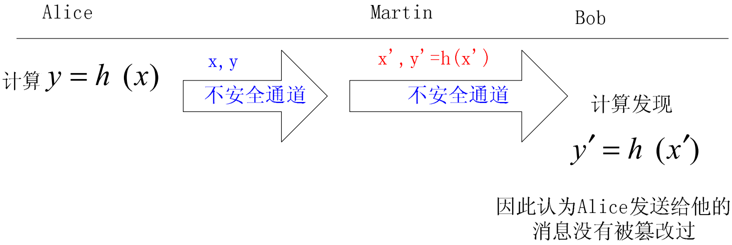
4.1 Hash函数与数据完整性

hash函数的基本要求：

- 为了方便存储，Hash值必须是一个较短的数值，通常为160比特。
- 对输入数据长度无限制。

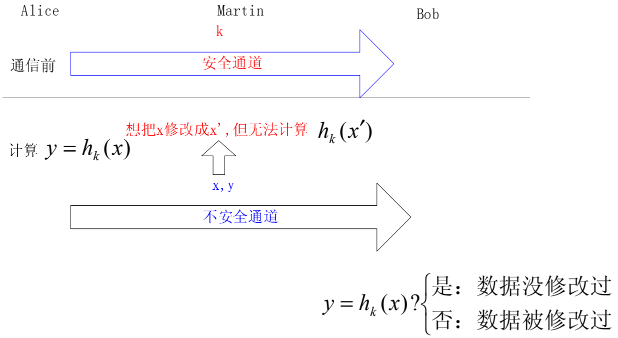
4.1 Hash函数与数据完整性

- 更进一步的问题：如果Alice同时把x和$y = h(x)$通过不安全通道发送给Bob，很多情况下需要这么做。那么Bob将无法按照上述方法验证数据是否被修改。



4.1 Hash函数与数据完整性

- 解决方案：Alice使用密钥控制的Hash函数h计算$y = h_K(x)$



消息认证码（MAC）：带密钥的hash函数。

4.1 Hash函数与数据完整性

定义4.1

一个Hash族是满足下列条件的四元组$(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$：

1. $\mathcal{X}$：消息的集合。
2. $\mathcal{Y}$：消息摘要或认证标签的集合
3. $\mathcal{K}$：密钥空间
4. 对每个$K \in \mathcal{K}$，存在一个Hash函数$h_K \in \mathcal{H}$，$h_K : \mathcal{X} \to \mathcal{Y}$。

注：

- 不带密钥的Hash函数可看成密钥个数为1
- $y = h_k(x)$,$x \in \mathcal{X}, y \in \mathcal{Y}, k \in \mathcal{K}, h_k \in \mathcal{H}$

4.1 Hash函数与数据完整性

本章学习内容：

- Hash函数的安全性(重点)
- Hash函数的迭代构造方法和SHA-1算法
- 消息认证码
- 无条件安全消息认证码（略）

4.2 Hash函数的安全性

问题：什么样的Hash函数h才是安全的？

我们考虑不带密钥的hash函数h：

- 问题4.1（原像(Preimage)）：已知y，求x满足$y = h(x)$

如该问题易解：大部分电脑并不直接存储用户的口令字，而是口令字的hash值。有如下攻击：

1. Oscar获得合法用户Alice的口令字hash值y(很多情况下容易获得)
2. Oscar计算k'，有$y = h(k')$
3. Oscar用口令字k'冒充Alice的身份登陆电脑。

结论：hash函数需对该问题难解，也称为原像稳固或单向。

4.2 Hash函数的安全性

- 问题4.2(第二原像)： 已知x，求x' 有$x' \neq x, h(x) = h(x')$。

若该问题易解：在数字签名中，一般是先计算hash值，后对hash值签名。有如下攻击：

1. Alice对她的一份声明x做数字签名：先计算得到x的hash值$y = h(x)$，然后对y计算签名得到s。
2. Alice把她的签名(x, s)发布在公共网络上，谁都可以验证s是Alice对x的签名。（细节见第7章）
3. Oscar通过求第二原像问题，计算出另一个消息x'有$y = h(x')$，这样他伪造出了Alice对x'的签名s。

结论：hash函数对该问题需难解，也称为第二原像问题稳固。

4.2 Hash函数的安全性

- 问题4.3(碰撞)： 找出两个随机的消息x和x'，使得$h(x) = h(x')$。

若该问题易解：Alice可以欺骗Bob：

1. Alice准备一份合同的两种版本x_1, x_2，x_1对Bob有利，x_2将使他破产。
2. Alice对x_1, x_2都做不影响其内容的细微修改（如加空格、退格等），并分别计算散列值。
3. Alice找出x_1, x_2的修改版本中hash值一致的两个版本x_1', x_2'，即满足$h(x_1') = h(x_2')$。
4. Alice让Bob对x_1'的hash值$h(x_1')$进行数字签名。
5. Alice向法庭证明Bob对不利于他的合同x_2'签名，因为$h(x_1') = h(x_2')$。

结论：hash函数需对该问题难解，也称为碰撞稳固。

分析以下例子的Hash函数是不安全(掌握重点):
例：假定Hash函数$h : \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{Z}_n$如下定义：

$$h(x, y) = ax + by \bmod n$$

对任意一对消息$(x_1, y_1), (x_2, y_2)$可计算它们的Hash值：

$$z_1 = h(x_1, y_1), z_2 = h(x_2, y_2)$$

对任意$r, s \in \mathbb{Z}_n$我们有：

$$\begin{aligned} & h(rx_1 + sx_2 \bmod n, ry_1 + sy_2 \bmod n) \\ = & a(rx_1 + sx_2) + b(ry_1 + sy_2) \bmod n \\ = & r(ax_1 + by_2) + s(ax_1 + by_2) \bmod n \\ = & r(h(x_1, y_1)) + sh(x_2, y_2)) \bmod n \\ = & rz_1 + sz_2 \bmod n \end{aligned}$$

特别的，若z_1, z_2互素，那么取尽r, s的所有可能值，我们就可以通过上述式子对所有摘要值求逆，从而不满足单向性！

MD5算法的攻击例子：MD5是一个应用广泛的著名Hash算法，摘要长度为128位。

- 2004年，山东大学的王小云教授证明MD5数字签名算法可以产生碰撞。
- 2007年，Marc Stevens，Arjen K. Lenstra和Benne de Weger进一步指出通过伪造软件签名，可重复性攻击MD5算法。研究者使用前缀碰撞法（chosen-prefix collision），使程序前端包含恶意程序，利用后面的空间添上垃圾代码凑出同样的MD5 Hash值。
- 2008年，荷兰埃因霍芬技术大学科学家成功把2个可执行文件进行了MD5碰撞，使得这两个运行结果不同的程序被计算出同一个MD5。
- 2008年12月一组科研人员通过MD5碰撞成功生成了伪造的SSL证书，这使得在https协议中服务器可以伪造一些根CA的签名。

SHA-0算法的碰撞攻击例子：SHA (Secure Hash Algorithm，译为安全Hash算法) 是美国国家安全局(NSA) 设计，美国国家标准与技术研究院(NIST) 发布的一系列密码散列函数。SHA共有三个版本：SHA-0、SHA-1和SHA-2，SHA-3正在研发之中。其中SHA-0是最早发布的版本，应用较少，SHA-1应用最为广泛。

- 1998年的美密会CRYPTO’98上，Florent Chabaud 和Antoine Joux 提出了找出了计算复杂度为2^{61}的SHA-0 碰撞的攻击，而在同等输出长度下理想hash函数的计算复杂度为2^{81}。
- 2004年，Biham和Chen发现了SHA-0的近似碰撞：找到两条不同的消息，其160比特的SHA-0输出中有142比特相同。

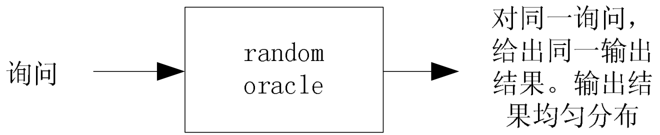
SHA-0算法的碰撞攻击例子：（续）

- 2004年8月12日，Joux, Carribault, Lemuet 和Jalby 宣布了SHA-0 算法的散列碰撞，他们的攻击复杂度为2^{51}。
- 2004年8月14日，在CRYPTO 2004的自由发言会议上，王小云等人给出了SHA-0的另一个碰撞攻击，计算复杂度为2^{40}。
- 2005年2月，王小云等人宣布了只需要2^{39}次运算就可找到SHA-0的碰撞。
- 课本中的表4.1给出了SHA-0的一个碰撞。

4.2.1 随机谕示模型

本节解决：如果不考虑算法的具体细节，最基本的搜索攻击对三个安全问题的成功概率。

随机谕示(random oracle)模型：random oracle是一个黑盒子，对每个询问都可以输出结果，输出服从均匀分布，且对同一咨询给出同一个结果。



以下把Hash函数假设成random oracle，且设$|\mathcal{Y}| = M$，那么对任意$x \in \mathcal{X}, y \in \mathcal{Y}$有$Pr[h(x) = y] = 1/M$。

4.2.2 随机谕示模型下的攻击算法

算法4.1（对原像问题的攻击）：Find-Preimage(h, y, Q)

1. 选择任意$\mathcal{X}_0 = \{x_1, x_2, \ldots, x_Q\} \subseteq \mathcal{X}$
2. 计算$y_1 \leftarrow h(x_1), y_2 \leftarrow h(x_2), \ldots, y_Q \leftarrow h(x_Q)$
3. 若存在$y_i = y$：　输出x_i

   否则失败

定理4.2：算法4.1的平均成功概率$\epsilon = 1 - (1 - 1/M)^Q$。

Proof.

令E_i表示事件“$h(x_i) = y$”，那么E_i是独立事件且由random oracle假设有$Pr[E_i] = 1/M$。因而

$$\begin{aligned}
\epsilon &= Pr(E_1 \cup E_2 \cup \ldots \cup E_Q) \\
&= 1 - Pr(\overline{E_1} \cap \overline{E_2} \cap \ldots \cap \overline{E_Q}) \\
&= 1 - \left(1 - \frac{1}{M}\right)^Q
\end{aligned}$$

$\square$

算法4.2（对第二原像问题的攻击）：
Find-Second-Preimage(h, x, Q)

1. $y \leftarrow h(x)$
2. 选择任意$\mathcal{X}_0 = \{x_1, x_2, \ldots, x_Q\} \subseteq \mathcal{X} \setminus \{x\}$
3. 计算$y_1 \leftarrow h(x_1), y_2 \leftarrow h(x_2), \ldots, y_Q \leftarrow h(x_Q)$
4. 若存在$y_i = y$： 输出x_i
   否则失败

定理4.3：算法4.2的平均成功概率$\epsilon = 1 - (1 - 1/M)^Q$。

Proof.
类似于定理4.2的证明。 □

注意：课本对应结论中Q改成$Q - 1$，和上述结论实质一致。

算法4.3：Find-collision(h, Q)

1. 选择任意$\mathcal{X}_0 = \{x_1, x_2, \ldots, x_Q\} \subseteq \mathcal{X}$
2. 计算$y_1 \leftarrow h(x_1), y_2 \leftarrow h(x_2), \ldots, y_Q \leftarrow h(x_Q)$
3. 若存在$x_i \neq x_j$有$y_i = y_j$： 输出(x_i, x_j)
   否则失败

定理4.4：算法4.3的平均成功概率

$$
\begin{aligned}
\epsilon &= 1 - \frac{\binom{M}{Q}}{M^Q} \\
&= 1 - \left(\frac{M-1}{M}\right)\left(\frac{M-2}{M}\right)\cdots\left(\frac{M-Q+1}{M}\right) \\
&\approx 1 - e^{\frac{-Q(Q-1)}{2M}} (\text{由} e^{-x} \text{展开式})
\end{aligned}
$$

证明：

$$
\begin{aligned}
1-\epsilon &= \text{失败概率} \\
&= Pr(y_1, y_2, \ldots, y_Q \text{为不同值}) = \frac{\binom{M}{Q}}{M^Q}
\end{aligned}
$$

- 当$M = 365, Q = 23$时，即为生日“驳论”：23个人中至少有两人同一天生日的概率$\epsilon \approx 0.5$。因结论违背直觉，而称为“驳论”，实则不是。
- 当摘要=40bits，$Q = 2^{20} \approx$一百万，$\epsilon \approx 0.5$.
- 一般取摘要$\geq$ 128bits，若要$\epsilon \approx 0.5$，需$Q \approx 2^{64}$.

对Hash函数的三个问题的难度比较

总结

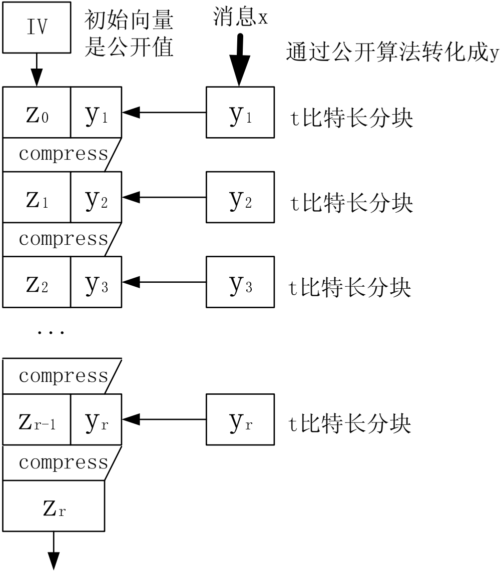
计算过程： 随机产生Q个不同的消息$x_1, \ldots, x_Q$（在第二原像问题中，还要求这些消息不同于给定消息x），分别计算它们的Hash值$h(x_1), \ldots, h(x_Q)$。

问题	成功条件	成功概率
原像问题	某个x_i有$h(x_i) = y$	$1-(1-1/M)^Q$
第二原像	某个x_i有$h(x_i) = h(x)$	$1-(1-1/M)^Q$
碰撞	某2个x_i, x_j有$h(x_i) = h(x_j)$	$1-\frac{\binom{M}{Q}}{M^Q}$

总结： 碰撞问题的成功概率是最高的。

- 输入空间有限的Hash函数称为压缩函数。
- 输入空间无限的Hash函数通常由压缩函数迭代生成，类似于分组加密的迭代生成思想。

Figure: 由压缩函数 $compress : \{0,1\}^{m+t} \to \{0,1\}^m$ 构造Hash函数



预处理： 把输入比特串x（x的长度$\geq m+t+1$）通过一个公开的算法转化成比特串y，并且y的长度为t的倍数：

$$x \quad \Longrightarrow \quad y = y_1 \| y_2 \| \dots \| y_r, \text{每一分块} y_i \text{的比特长度为} t$$

如： $y = x \| pad(x)$ $pad(\cdot)$表示填充函数，

如填充0比特使得y的长度为t的倍数

计算过程： 设IV是一个长度为m的公开的初始值比特串，如下迭代计算：

$$\text{Hash函数的一般迭代过程} \begin{cases} z_0 & \leftarrow & IV \\ z_1 & \leftarrow & compress(z_0 \| y_1) \\ z_2 & \leftarrow & compress(z_1 \| y_2) \\ \vdots & \vdots & \vdots \\ z_r & \leftarrow & compress(z_{r-1} \| y_r) \end{cases} \tag{1}$$

输出变换： 假设$g : \{0,1\}^m \to \{0,1\}^l$是一个公开函数，$h(x) \leftarrow g(z_r)$。输出变换是可选的，如无输出变换则定义：$h(x) = z_r$

注意事项：

- 预处理中$x \rightarrow y$必须是单射
- 如果$x \rightarrow y$不是单射，即存在$x \neq x'$，$x \rightarrow y$和$x' \rightarrow y'$，有$y = y'$，那么$h(x) = h(x') \Longrightarrow h$ 不是碰撞稳固的。

下面我们学习一种由压缩函数构造Hash函数的方法，称为Merkle-Damgård结构，它有以下特点：

- 如果压缩函数是碰撞稳固的，那么所构造的Hash函数也是碰撞稳固的。

4.3.1 Merkle-Damgård结构

算法4.6：

当$t \geq 2$时，Merkle-Damgård构造方法如下：

预处理为：

$$
\begin{aligned}
x &= x_1||x_2||x_3||\ldots||x_k, x_i\text{长度为}t-1, 1 \leq i \leq k-1, \\
IV &= 0^m \qquad\qquad\qquad x_k\text{长度为}t-1-d \\
\mathbf{y_1} &= 0||x_1 \\
\mathbf{y_2} &= 1||x_2 \\
\vdots\ &\ \vdots\ \ \vdots \\
\mathbf{y_k} &= 1||x_k||0^d \\
\mathbf{y_{k+1}} &= 1||0\ldots0||d\text{的二进制表示}, |y_{k+1}| = t
\end{aligned}
$$

计算过程： $z_1 \leftarrow 0^{m+1}||y_1$

$g_1 \leftarrow compress(z_1)$

for $i \leftarrow 1$ to k

$$
\text{do} \begin{cases} z_{i+1} \leftarrow g_i||1||y_{i+1} \\ g_{i+1} \leftarrow compress(z_{i+1}) \end{cases}
$$

输出 $h(x) = g_{k+1}$。

定理4.6：假定$compress : \{0,1\}^{m+t} \to \{0,1\}^m$ 是一个碰撞稳固的压缩函数，其中$t \geq 2$。则按照算法4.6构造的函数

$$h : \cup_{t=m+t+1}^{\infty}\{0,1\}^i \to \{0,1\}^m$$

是一个碰撞稳固的Hash函数。

证明思路： 用反证法证明。 假定可以找到$x \neq x'$使得$h(x) = h(x')$。我们将证明在多项式时间内找到compress的碰撞。

不影响一般性，设$k \geq l$。令

$$y(x) = y_1||y_2||\dots||y_{k+1}$$

和

$$y(x') = y'_1||y'_2||\dots||y'_{l+1}$$

其中x和x'被分别填充了d和d'个0。
用$g_1,\dots,g_{k+1}$和$g'_1,\dots,g'_{l+1}$分别表示算法中计算出的g值；
用$z_1,\dots,z_{k+1}$和$z'_1,\dots,z'_{l+1}$分别表示算法中计算出的z值；
容易证明以下2点：

1. 性质1：若$z_i = z'_j$，则$y_i = y'_j$，$g_{i-1} = g'_{j-1} (i, j > 1)$
2. 性质2：对任意$i \geq 2$，$z_i \neq z'_1$

由$h(x) = h(x')$得到$g_{k+1} = g'_{l+1}$，

$$
若\begin{cases}
z_{k+1} \neq z'_{l+1} \Rightarrow & compress(z_{k+1}) = compress(z'_{l+1}) \\
& \text{则找到compress的一个碰撞} \\
z_{k+1} = z'_{l+1} \Rightarrow & g_k = g'_l，若 \begin{cases}
z_k \neq z'_l \Rightarrow & \text{找到compress的一个碰撞} \\
z_k = z'_l \Rightarrow & g_{k-1} = g'_{l-1} \\
& \vdots \\
& \text{如果一直重复下去，} \\
& \text{最后} g_{k-l+1} = g'_1
\end{cases}
\end{cases}
$$

以上红字结论由性质1得到。最后分2种情况讨论：

1. 若$k > l$，由性质2 $z_{k-l+1} \neq z'_1$，得到compress的一个碰撞；
2. 若$k = l$：若$z_1 \neq z'_1$，找到compress的一个碰撞；若$z_1 = z'_1$，前面已证$z_i = z'_i, 2 \leq i \leq k+1$，那么$y_i = y'_i, 1 \leq i \leq k+1$，所以$y(x) = y(x')$。但因为映射$x :\rightarrow y(x)$是单射，这意味着$x = x'$，与$x \neq x'$矛盾。■

思考：给出不同于Merkle-Damgård的结构，仍然满足："如果compress函数是碰撞稳固的，构造得到的hash函数也是碰撞稳固的"，并给出完整证明。
当$t = 1$时的Merkle-Damgård结构见课本P.104算法4.7，相应的有定理4.7。

4.3.2 安全Hash算法

数据安全算法(SHA)简介：

- SHA是美国国家安全局(NSA) 设计，美国国家标准与技术研究院(NIST) 发布的一系列密码散列函数
- SHA-0发布于1993年。
- SHA-1发布于1995年，它针对SHA-0的一个弱点做了微小修订，输出长度为160比特，是SHA家族中应用最广泛的算法。
- SHA-2包括4个算法：SHA-224，SHA-256，SHA-384，SHA-512，输出长度分别为：224、256、384和512。
- SHA-32012年10月选出Keccak被NIST选为SHA-3。

SHA-1算法结构：

- 输入长度$|x| \leq 2^{64} - 1$
- 输出长度为：160比特
- 面向字的操作，每个字32比特。
- 填充算法：SHA-1-PAD(x)把消息x转化成512倍数长的串

$$y = x||1||0^d||l$$

其中$d \leftarrow (447 - |x|) \bmod 512$,$l \leftarrow |x|$($|x|$表示$x$的长度)的二进制表示，而且长度为64比特，如果达不到64比特则在左边填充0。

- 所使用的操作：

$X \wedge Y$	X和Y的逻辑“和”
$X \vee Y$	X和Y的逻辑“或”
$X \oplus Y$	X和Y的逻辑“异或”
$\neg X$	X的逻辑“补”
$X + Y$	模2^{32}整数和，等于$(X + Y) \wedge (2^{32} - 1)$
$ROTL^s(X)$	X循环左移s个位置$(0 \leq s \leq 31)$

SHA-1算法结构：

- 非线性函数$f_0, f_1, \ldots, f_{79}$：

$$f_t(B, C, D) = \begin{cases} (B \wedge C) \vee (\neg B) \wedge D) & \text{对于} 0 \le t \le 19 \\ B \oplus C \oplus D & \text{对于} 20 \le t \le 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)) & \text{对于} 40 \le t \le 59 \\ B \oplus C \oplus D & \text{对于} 60 \le t \le 79 \end{cases}$$

  输入B, C, D是三个字。

- 常数$K_t, 0 \le t \le 79$

SHA-1计算过程：

输入：　x

填充：　$y \leftarrow SHA - 1 - PAD(x)$

常数：　$K_0, \ldots, K_{79}$

划分：　令$y = M_1 || M_2 || \ldots || M_n$，其中每个$M_i$是一个512比特的分组。

IV值：　$H_0 || H_1 || H_2 || H_3 || H_4 || \leftarrow 67452301 || EFCDADCFE || 10325476 || C3D2E1F0$

SHA-1计算过程（续）：

for $i \leftarrow 1$ to n

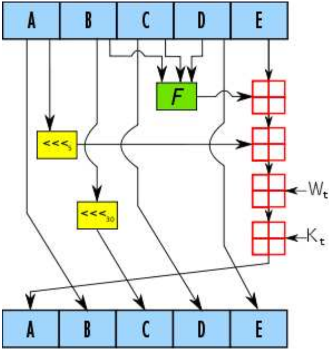
说明：以下括号部分相当于$compress(Z_{i-1}||M_i), Z_{i-1} = H_0||H_1||\ldots||H_4$

do
$$\left\{\begin{array}{l}
\text{令 } M_i = W_0||W_1||\ldots||W_{15}\text{，其中每个}W_i\text{是一个字} \\
\text{for } i \leftarrow 16 \text{ to } 79 \text{ do } W_i \leftarrow ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \\
A \leftarrow H_0, B \leftarrow H_1, C \leftarrow H_2, D \leftarrow H_3, E \leftarrow H_4 \\
\text{for } i \leftarrow 0 \text{ to } 79 \\
\quad \text{do} \left\{\begin{array}{l}
temp \leftarrow ROLT^5(A) + f_t(B, C, D) + E + W_t + K_t \\
E \leftarrow D \\
D \leftarrow C \\
C \leftarrow ROLT^{30}(B) \\
B \leftarrow A \\
A \leftarrow temp
\end{array}\right. \\
H_0 \leftarrow H_0 + A, H_1 \leftarrow H_1 + B, H_2 \leftarrow H_2 + C \\
H_3 \leftarrow H_3 + D, H_4 \leftarrow H_4 + E
\end{array}\right.$$

输出：$(H_0||H_1||H_2||H_3||H_4)$

4.3.2 安全Hash算法

SHA1算法中红色do部分图示（迭代步骤）：



4.3.2 安全Hash算法

SHA的安全性现状：见4.2节关于Hash函数安全性的真实例子2。

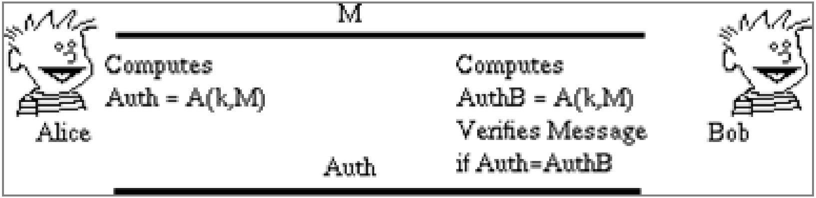
4.4 消息认证码

消息认证码(message authentication code，简写MAC)：

定义： 带密钥的Hash函数。

用途： 用于消息完整性认证。

消息认证过程图示一($A(K, M)$表示密钥为k的消息认证码)：



4.4 消息认证码

MAC的安全功能：

- 确认消息完整性，因为只有密钥的拥有者才能生成消息的认证码。
- 但不能解决消息的不可否认性，即无法证实消息由谁生成（消息生成者和验证者都拥有密钥，都可生成认证码）。
  - 签名方案同时有消息的不可否认性和完整性。

构造方法：

1. 由不带密钥的Hash函数构造得到，如HMAC(见4.4.1 嵌套MAC和HMAC)
2. 由对称密码构造得到，如CBC-MAC(见4.4.2 CBC-MAC)

4.4 消息认证码

衡量MAC算法的安全性(或者安全性定义):

- 和不带密钥的Hash函数要求不同。
- 使用假冒(forger)者来描述MAC算法的安全性:

(ϵ, Q)假冒者

如果攻击者获得$x_1, x_2, \ldots, x_Q$的消息认证码$y_1, y_2, \ldots, y_Q$，对$x \notin \{x_1, x_2, \ldots, x_Q\}$，如果攻击者计算得到有效的对$(x, y)$，即$y = MAC_k(x)$，那么$(x, y)$称为一个假冒，如果成功概率至少为$\epsilon$，则攻击者称为$(\epsilon, Q)$假冒者。
简言之，(ϵ, Q)假冒者是在获得Q个MAC后，可构造出有效MAC的概率为ϵ的攻击者。

显然，一个好的MAC函数，如Q固定，(ϵ, Q)假冒者的成功概率ϵ越小越好。

4.4 消息认证码

(1, 1)假冒者例子：假定$compress : \{0,1\}^{m+t} \rightarrow \{0,1\}^m$是一个压缩函数，使用compress通过一般的迭代的方法生成带密钥的Hash函数h_K，并且：

$$IV = K$$

为简单起见，假定没有预处理步骤和输出变换。
已知消息x和它的hash值$h_K(x)$，对任意t长x'，根据迭代计算过程知道

$$h_K(x||x') = compress(h_K(x)||x')$$

因此，攻击者即使不知道K，也可以根据上式计算出$h_k(x||x')$。

4.4.1 嵌套MAC和HMAC
具体过程不做考试要求

嵌套MAC给出了由两个带密钥的Hash族来建立一个密钥空间更大的MAC算法的方法：

嵌套MAC：

假定$(\mathcal{X},\mathcal{Y},\mathcal{K},\mathcal{G})$和$(\mathcal{Y},\mathcal{Z},\mathcal{L},\mathcal{H})$ 是Hash族，则它们的复合Hash族定义为：$(\mathcal{X},\mathcal{Z},\mathcal{M},\mathcal{G}\circ\mathcal{H})$，其中$\mathcal{M}=\mathcal{K}\times\mathcal{L}$(密钥空间)，并且

$$\mathcal{G}\circ\mathcal{H}=\{g\circ h : g\in\mathcal{G}, h\in\mathcal{H}\}$$

对所有$x\in\mathcal{X}$，有：

$$(g\circ h)_{K,L}(x)=h_L(g_K(x))$$

4.4.1 嵌套MAC和HMAC
HMAC

HMAC介绍：

- HMAC是一个于2002年3月被提议为FIPS标准的嵌套MAC算法。
- HMAC通过不带密钥的Hash函数来构造MAC。
- 基于SHA-1的HMAC算法描述如下：
    - 密钥K长度：512比特
    - 输出长度：160比特
    - 2个512比特常数：ipad$=3636\ldots36$，opad$=5C5C\ldots5C$
    - x的MAC如下计算：

$$HMAC_K(x) = SHA-1(K \oplus \text{opad})||SHA-1((K \oplus \text{ipad}||x))$$

4.4.2 CBC-MAC

CBC-MAC给出了由分组密码构造MAC的方法：

密码体制4.2 CBC-MAC(x, K)

分组密码： e_K

输入： 令$x = x_1||x_2||\ldots||x_n$，每个$x_i$都是$t$长比特串

初始向量： $IV = 00\ldots0$

计算过程：

1. $y_0 \leftarrow IV$
2. for $i \leftarrow 1$ to n

   do $y_i \leftarrow e_K(y_{i-1} \oplus x_i)$

输出： y_n

说明：CBC-MAC的结果为以K为密钥，CBC分组加密模式下对x加密的最后一个密文分组。

4.4.2 CBC-MAC
CBC-MAC的生日攻击

Oscar产生如下形式的q个消息并请求它们的MAC：

$$x^i = x_1^i || x_2^i || x_3 || x_4 || \dots || x_n \quad (i = 1, \dots, q)$$

其中x_1^i, x_2^i随机产生，且$x_1^i (i = 1, \dots, n)$互不相同。
下面分析x^i, x^j产生碰撞的概率和充要条件。
假设CBC-MAC(x^i, K)中间计算得到$y_0^i, y_1^i, \dots, y_n^i$。
假设CBC-MAC(x^j, K)中间计算得到$y_0^j, y_1^j, \dots, y_n^j$。
x^i, x^j产生碰撞当且仅当

$$\begin{aligned} y_n^i = y_n^j \quad & \Leftrightarrow \quad y_{n-1}^i \oplus x_n = y_{n-1}^j \oplus x_n \quad (e_K\text{是双射}) \\ & \Leftrightarrow \quad y_{n-1}^i = y_{n-1}^j \Leftrightarrow y_{n-2}^i = y_{n-2}^j \Leftrightarrow \dots \\ & \Leftrightarrow \quad y_2^i = y_2^j \Leftrightarrow y_1^i \oplus x_2^i = y_1^j \oplus x_2^j \quad (*) \end{aligned}$$

所以x^i, x^j产生碰撞的概率为

$$Pr(y_n^i = y_n^j) = Pr(y_1^i \oplus x_2^i = y_1^j \oplus x_2^j) = Pr(x_2^i \oplus x_2^j = y_1^i \oplus y_1^j) = 1/2^t$$

4.4.2 CBC-MAC
CBC-MAC的生日攻击

因此从定理4.4得到，若要碰撞成功率$\epsilon = 1/2$，需取$q = O(2^{t/2})$ (相当于算法4.2中$M = 2^t$)。
若x^i, x^j产生碰撞，Oscar计算：

$$\begin{aligned} v &= x_1^i || x_2^i \oplus x_\delta || x_3 || x_4 || \ldots || x_n \\ w &= x_1^j || x_2^j \oplus x_\delta || x_3 || x_4 || \ldots || x_n \end{aligned}$$

易验证v, w满足产生碰撞的充要条件(*)，因此Oscar只要请求得到v的MAC，也就知道了w的MAC，虽然他不知道密钥K。
总结：Oscar是一个$(1/2, O(2^{t/2}))$假冒者

总结

本章学习要点（标蓝部分为主要考察点）：

- Hash函数和消息认证码(MAC)的概念、安全作用。
- Hash函数的安全性定义的理解（即原像问题、第二原像问题和碰撞问题）
- 对给定的简单Hash函数，找到其不安全的地方（如习题4.5、4.6，相关例子）
- Hash函数的迭代构造原理
- MAC的安全性衡量方法（即(ε, Q)假冒者定义）
- 对给定的简单的MAC算法，给出不符合安全性的地方（如习题4.12、4.13，相关例子）