第四章　Hash 函数

张磊

华东师范大学 • 软件学院

本章内容

1. Hash 函数与数据完整性
2. Hash 函数的安全性
3. 迭代 Hash 函数
4. 消息认证码
5. 无条件安全消息认证码 ($\times$)

§ 4.1 Hash 函数与数据完整性

Hash 函数基本要求

1. Hash 函数的安全作用: 提供数据完整性保护, 是数据的 "指纹", 也称为消息摘要. Hash 函数也称为哈希函数, 单向散列函数, 杂凑函数等.
2. 假定 h 是一个 Hash 函数. Alice 通过对数据 x 计算它的 Hash 函数值 $y = h(x)$, 然后把 x (通过不安全信道) 和 y (通过安全信道) 发送给 Bob. Bob 通过如下验证所接收到的数据 x' 是否被篡改过:

$$\begin{cases} y = h(x'), & \text{数据没有被篡改;} \\ y \neq h(x'), & \text{数据被篡改过.} \end{cases}$$

Hash 函数基本要求

1. Hash 函数的安全作用: 提供数据完整性保护, 是数据的 “指纹”, 也称为消息摘要. Hash 函数也称为哈希函数, 单向散列函数, 杂凑函数等.
2. 假定 h 是一个 Hash 函数. Alice 通过对数据 x 计算它的 Hash 函数值 $y = h(x)$, 然后把 x (通过不安全信道) 和 y (通过安全信道) 发送给 Bob. Bob 通过如下验证所接收到的数据 x' 是否被篡改过:

$$\begin{cases} y = h(x'), & \text{数据没有被篡改;} \\ y \neq h(x'), & \text{数据被篡改过.} \end{cases}$$

Hash 函数 h 基本要求

1. 由 x 计算 $h(x)$ 是容易的;
2. 为方便存储, Hash 函数值 $h(x)$ 必须是一个较短的数值;
3. Hash 函数应该对输入数据的长度没有限制.

更进一步问题

如果 Alice 把 x 和 $y = h(x)$ 都通过不安全信道发送给 Bob, 则 Bob 将无法按照上述方法验证数据是否被篡改, 这是因为敌手可以使用修改过的 x' 代替 x, 用 $y' = h(x')$ 代替 y.

更进一步问题

如果 Alice 把 x 和 $y = h(x)$ 都通过不安全信道发送给 Bob, 则 Bob 将无法按照上述方法验证数据是否被篡改, 这是因为敌手可以使用修改过的 x' 代替 x, 用 $y' = h(x')$ 代替 y.

解决方案

Alice 用密钥 K 控制的 Hash 函数 h_K 计算

$$y = h_K(x),$$

K 为 Alice 和 Bob 所共知.

Bob 通过如下验证所接收到的数据 x', y' 是否被篡改过:

$$\begin{cases} y' = h_K(x'), & \text{数据没有被篡改;} \\ y' \neq h_K(x'), & \text{数据被篡改过.} \end{cases}$$

更进一步问题

如果 Alice 把 x 和 $y = h(x)$ 都通过不安全信道发送给 Bob, 则 Bob 将无法按照上述方法验证数据是否被篡改, 这是因为敌手可以使用修改过的 x' 代替 x, 用 $y' = h(x')$ 代替 y.

解决方案

Alice 用密钥 K 控制的 Hash 函数 h_K 计算

$$y = h_K(x),$$

K 为 Alice 和 Bob 所共知.

Bob 通过如下验证所接收到的数据 x', y' 是否被篡改过:

$$\begin{cases} y' = h_K(x'), & \text{数据没有被篡改;} \\ y' \neq h_K(x'), & \text{数据被篡改过.} \end{cases}$$

消息认证码: 带密钥的 Hash 函数值.

Hash 函数簇定义

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

1. $\mathcal{X}$ 是所有可能的消息的集合;
2. $\mathcal{Y}$ 是所有可能的消息摘要或认证标签构成的有限集;
3. $\mathcal{K}$ 密钥空间, 是所有可能的密钥构成的有限集;
4. 对每一个 $K \in \mathcal{K}$, 存在一个 Hash 函数 $h_K \in \mathcal{H}$, $h_K : \mathcal{X} \to \mathcal{Y}$.

Hash 函数簇定义

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

1. $\mathcal{X}$ 是所有可能的消息的集合;
2. $\mathcal{Y}$ 是所有可能的消息摘要或认证标签构成的有限集;
3. $\mathcal{K}$ 密钥空间, 是所有可能的密钥构成的有限集;
4. 对每一个 $K \in \mathcal{K}$, 存在一个 Hash 函数 $h_K \in \mathcal{H}$, $h_K : \mathcal{X} \to \mathcal{Y}$.

不带密钥的 Hash 函数可看成密钥个数为 1, 即

$$|\mathcal{K}| = 1.$$

§ 4.2 Hash 函数的安全性

有效对定义

假定 $h : \mathcal{X} \to \mathcal{Y}$ 是一个不带密钥的 Hash 函数.

1. $(x, y) \in \mathcal{X} \times \mathcal{Y}$ 称为 h 的一个有效对, 如果 $y = h(x)$.

有效对定义

假定 $h: \mathcal{X} \to \mathcal{Y}$ 是一个不带密钥的 Hash 函数.

1. $(x, y) \in \mathcal{X} \times \mathcal{Y}$ 称为 h 的一个有效对, 如果 $y = h(x)$.
2. 许多 Hash 函数的应用都希望仅有一种方法产生有效对 (x, y), 那就是首先选择 x, 再把 h 作用于 x, 计算出 $y = h(x)$.

Hash 函数的安全性问题

问题 1　原象问题:

实例:　Hash 函数 $h : \mathcal{X} \to \mathcal{Y}$ 和 $y \in \mathcal{Y}$.

找出:　$x \in \mathcal{X}$, 使得 $h(x) = y$.

Hash 函数的安全性问题

问题 1　原象问题:

实例:　Hash 函数 $h : \mathcal{X} \to \mathcal{Y}$ 和 $y \in \mathcal{Y}$.

找出:　$x \in \mathcal{X}$, 使得 $h(x) = y$.

问题 2　第二原象问题:

实例:　Hash 函数 $h : \mathcal{X} \to \mathcal{Y}$ 和 $x \in \mathcal{X}$.

找出:　$x' \in \mathcal{X}$, 使得 $x' \neq x$ 且 $h(x') = h(x)$.

Hash 函数的安全性问题

问题 1　原象问题:

实例:　Hash 函数 $h: \mathcal{X} \to \mathcal{Y}$ 和 $y \in \mathcal{Y}$.

找出:　$x \in \mathcal{X}$, 使得 $h(x) = y$.

问题 2　第二原象问题:

实例:　Hash 函数 $h: \mathcal{X} \to \mathcal{Y}$ 和 $x \in \mathcal{X}$.

找出:　$x' \in \mathcal{X}$, 使得 $x' \neq x$ 且 $h(x') = h(x)$.

问题 3　碰撞问题:

实例:　Hash 函数 $h: \mathcal{X} \to \mathcal{Y}$.

找出:　$x, x' \in \mathcal{X}$, 使得 $x' \neq x$ 且 $h(x') = h(x)$.

Hash 函数的安全性问题

问题 1　原象问题:

实例:　Hash 函数 $h : \mathcal{X} \to \mathcal{Y}$ 和 $y \in \mathcal{Y}$.

找出:　$x \in \mathcal{X}$, 使得 $h(x) = y$.

问题 2　第二原象问题:

实例:　Hash 函数 $h : \mathcal{X} \to \mathcal{Y}$ 和 $x \in \mathcal{X}$.

找出:　$x' \in \mathcal{X}$, 使得 $x' \neq x$ 且 $h(x') = h(x)$.

问题 3　碰撞问题:

实例:　Hash 函数 $h : \mathcal{X} \to \mathcal{Y}$.

找出:　$x, x' \in \mathcal{X}$, 使得 $x' \neq x$ 且 $h(x') = h(x)$.

不能有效解决原象问题、第二原象问题和碰撞问题的 Hash 函数分别称为原象稳固的、第二原象稳固的和碰撞稳固的.

碰撞问题规约为第二原象问题:

Collision-To-Second-Preimage(h)

external Oracle-2nd-Preimage

均匀地随机选择 $x \in \mathcal{X}$

if Oracle-2nd-Preimage$(h, x) = x'$

  then return (x, x')

  else return (failure)

安全问题规约 (一)

碰撞问题规约为第二原象问题:

Collision-To-Second-Preimage(h)

external Oracle-2nd-Preimage

均匀地随机选择 $x \in \mathcal{X}$

if Oracle-2nd-Preimage$(h, x) = x'$

then return (x, x')

else return (failure)

上述规约表明 Hash 函数的碰撞稳固意味着第二原象稳固.

碰撞问题规约为原象问题:

Collision-To-Preimage(h)

external Oracle-Preimage

均匀地随机选择 $x \in \mathcal{X}$

$y \leftarrow h(x)$

if (Oracle-Preimage$(h, y) = x'$) 且 $(x' \neq x)$

then return (x, x')

else return (failure)

定理

假定 $h : \mathcal{X} \to \mathcal{Y}$ 是一个 Hash 函数, $|\mathcal{X}|$ 和 $|\mathcal{Y}|$ 是有限的并且 $|\mathcal{X}| \geq 2|\mathcal{Y}|$. 假定 Oracle-Preimage 对固定的 h 是原象问题的一个 $(1, Q)$ 算法, 则上述 Collision-To-Preimage 对固定的 h 是碰撞问题的一个 $(1/2, Q+1)$ 算法.

定理

假定 $h: \mathcal{X} \to \mathcal{Y}$ 是一个 Hash 函数, $|\mathcal{X}|$ 和 $|\mathcal{Y}|$ 是有限的并且 $|\mathcal{X}| \geq 2|\mathcal{Y}|$. 假定 Oracle-Preimage 对固定的 h 是原象问题的一个 $(1, Q)$ 算法, 则上述 Collision-To-Preimage 对固定的 h 是碰撞问题的一个 $(1/2, Q+1)$ 算法.

补充概念

1. (ϵ, Q) 算法: 具有 (平均情况) 成功率 ϵ 的 Las Vegas 算法, 其中该算法向谕示 (预言) 器查询 (也就是求 h 的值) 的次数最多为 Q 次.
2. Las Vegas 算法: 一个不一定给出答案的随机算法, 但是一旦返回一个答案, 那么这个答案肯定就是正确的.

如果 Hash 函数 h 的消息摘要长度为 n, 我们通过生日悖论来考虑产生一对碰撞的可能性有多大.

生日悖论 (驳论)

如果 Hash 函数 h 的消息摘要长度为 n, 我们通过生日悖论来考虑产生一对碰撞的可能性有多大.

生日悖论 (驳论)

随机选取的 23 个人中, 至少有两人生日相同的概率至少为 $1/2$.

如果 Hash 函数 h 的消息摘要长度为 n, 我们通过生日悖论来考虑产生一对碰撞的可能性有多大.

生日悖论 (驳论)

随机选取的 23 个人中, 至少有两人生日相同的概率至少为 $1/2$.

上述结论表明, 如果 h 的输出可能值个数少于 365 个, 那么随机选择 23 个消息, 它们之中产生碰撞的概率大于 $1/2$.

生日悖论的一般形式

Theorem

如果集合 $\mathcal{Y}$ 的大小为 M, 那么从 $\mathcal{Y}$ 中随机独立选取的 Q 个元素

$$y_1, y_2, \cdots, y_Q$$

中至少有两个相同的概率为

$$1 - \frac{C_M^Q}{M^Q}.$$

生日悖论的一般形式

Theorem

如果集合 $\mathcal{Y}$ 的大小为 M, 那么从 $\mathcal{Y}$ 中随机独立选取的 Q 个元素

$$y_1, y_2, \cdots, y_Q$$

中至少有两个相同的概率为

$$1 - \frac{C_M^Q}{M^Q}.$$

生日攻击

如果 h 的输出可能值个数为 M 个, 那么随机选择 $Q \approx 1.17\sqrt{M}$ 个消息, 它们之中产生碰撞的概率大于 $1/2$.

生日悖论的一般形式

Theorem

如果集合 $\mathcal{Y}$ 的大小为 M, 那么从 $\mathcal{Y}$ 中随机独立选取的 Q 个元素

$$y_1, y_2, \cdots, y_Q$$

中至少有两个相同的概率为

$$1 - \frac{C_M^Q}{M^Q}.$$

生日攻击

如果 h 的输出可能值个数为 M 个, 那么随机选择 $Q \approx 1.17\sqrt{M}$ 个消息, 它们之中产生碰撞的概率大于 $1/2$.

正因为此, 通常建议消息摘要可接受的最小长度为 128 比特, 这时生日攻击需要超过 2^{64} 次随机操作才能以概率 $1/2$ 获得成功.

碰撞的真实例子: SHA (Secure Hash Algorithm, 安全 Hash 算法)

SHA 是美国 NIST 发布的一系列密码 Hash 函数, 其消息摘要长度为 160 比特. SHA 共有三个版本: SHA-0, SHA-1 和 SHA-2, 其中 SHA-0 为最早版本, 应用较少, SHA-1 应用最为广泛.

SHA 是美国 NIST 发布的一系列密码 Hash 函数, 其消息摘要长度为 160 比特. SHA 共有三个版本: SHA-0, SHA-1 和 SHA-2, 其中 SHA-0 为最早版本, 应用较少, SHA-1 应用最为广泛.

针对 SHA-0 的攻击

1. 在美密 Crypto’98上, Chabaud 和 Joux 提出了计算复杂度为 2^{61} 的碰撞攻击.
2. 2004 年, Biham 和 Chen 发现一个近似碰撞: 找到了两条消息, 它们的输出中有 142 比特相同. 同年 8 月 12 日, Joux 等人宣布了复杂度为 2^{51} 的碰撞攻击, 8 月 14 日, 王小云等人给出了另一个碰撞攻击, 计算复杂度为 2^{40}.
3. 2005 年 2 月, 王小云等人宣布只需要 2^{39} 次运算就可找到碰撞.
4. 书中表 4.1 给出了一个碰撞, 这两个消息都是 2048 比特长.

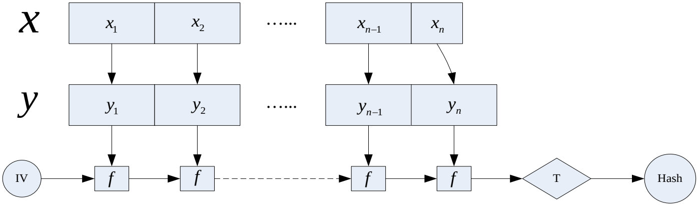
§ 4.3　迭代 Hash 函数

迭代 Hash 函数定义

1. 压缩函数: 有限定义域的 Hash 函数;
2. 迭代 Hash 函数: 将一个压缩函数延拓为具有无限定义域的 Hash 函数, 称之为迭代 Hash 函数.

1. 压缩函数: 有限定义域的 Hash 函数;
2. 迭代 Hash 函数: 将一个压缩函数延拓为具有无限定义域的 Hash 函数, 称之为迭代 Hash 函数.

由压缩函数 f 迭代生成 Hash 函数的通用方法:



迭代 Hash 函数的通用构造

由压缩函数 compress : $\{0,1\}^{m+t} \to \{0,1\}^m$ $(t \geq 1)$ 构造 Hash 函数 $h : \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \to \{0,1\}^l$ 的主要步骤:

由压缩函数 compress : $\{0,1\}^{m+t} \to \{0,1\}^m$ $(t \geq 1)$ 构造 Hash 函数 $h : \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \to \{0,1\}^l$ 的主要步骤:

预处理: 给定一个输入比特串 x ($|x| \geq m+t+1$), 用一个公开算法构造一个串 y, 使得 $|y| \equiv 0 \pmod{t}$. 记

$$y = y_1 || y_2 || \cdots || y_r, \quad |y_i| = t.$$

迭代 Hash 函数的通用构造

由压缩函数 compress : $\{0,1\}^{m+t} \to \{0,1\}^m$ $(t \geq 1)$ 构造 Hash 函数 $h: \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \to \{0,1\}^l$ 的主要步骤:

预处理: 给定一个输入比特串 x $(|x| \geq m+t+1)$, 用一个公开算法构造一个串 y, 使得 $|y| \equiv 0 \pmod t$. 记

$$y = y_1 || y_2 || \cdots || y_r, \quad |y_i| = t.$$

处理: 设 IV 是长度为 m 的公开的初始值比特串。则迭代计算

$$\begin{array}{ccl} z_0 & \leftarrow & \mathrm{IV} \\ z_1 & \leftarrow & \mathrm{compress}(z_0 || y_1) \\ z_2 & \leftarrow & \mathrm{compress}(z_1 || y_2) \\ \vdots & \vdots & \vdots \\ z_r & \leftarrow & \mathrm{compress}(z_{r-1} || y_r) \end{array}$$

迭代 Hash 函数的通用构造

由压缩函数 compress : $\{0,1\}^{m+t} \to \{0,1\}^m$ $(t \geq 1)$ 构造 Hash 函数 $h : \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \to \{0,1\}^l$ 的主要步骤:

预处理: 给定一个输入比特串 x $(|x| \geq m+t+1)$, 用一个公开算法构造一个串 y, 使得 $|y| \equiv 0 \pmod t$. 记

$$y = y_1 || y_2 || \cdots || y_r, \quad |y_i| = t.$$

处理: 设 IV 是长度为 m 的公开的初始值比特串。则迭代计算

$$\begin{array}{lll} z_0 & \leftarrow & \text{IV} \\ z_1 & \leftarrow & \text{compress}(z_0 || y_1) \\ z_2 & \leftarrow & \text{compress}(z_1 || y_2) \\ \vdots & \vdots & \vdots \\ z_r & \leftarrow & \text{compress}(z_{r-1} || y_r) \end{array}$$

可选输出变换: 设 $g : \{0,1\}^m \to \{0,1\}^l$ 是一个公开函数, 定义

$$h(x) = g(z_r).$$

Merkle-Damgård 结构

Merkle-Damgård 结构是一种由压缩函数

$$\text{compress} : \{0,1\}^{m+t} \to \{0,1\}^m$$

构造 Hash 函数

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

的方法.

Merkle-Damgård 结构

Merkle-Damgård 结构是一种由压缩函数

$$\text{compress} : \{0,1\}^{m+t} \to \{0,1\}^m$$

构造 Hash 函数

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

的方法.

特点

1. 如果压缩函数 compress 是碰撞稳固的, 则构造的 Hash 函数 h 也是碰撞稳固的;
2. 分 $t = 1$ 和 $t \geq 2$ 两种情况构造.

Merkle-Damgård(x) ($t \geq 2$)

external compress : $\{0,1\}^{m+t} \to \{0,1\}^m$, 这里 $t \geq 2$

$n \leftarrow |x|$

$k \leftarrow \lceil n/(t-1) \rceil$ // $x = x_1||x_2||\cdots||x_{k-1}||x_k$, $|x_{i \leq k-1}| = t-1$

$d \leftarrow k(t-1) - n$

for $i \leftarrow 1$ to $k-1$ do $y_i \leftarrow x_i$

$y_k \leftarrow x_k||0^d$

$y_{k+1} \leftarrow d$ 的二进制表示 // $|y_i| = t-1$

$z_1 \leftarrow 0^{m+1}||y_1$

$g_1 \leftarrow \text{compress}(z_1)$

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

$h(x) \leftarrow g_{k+1}$

return $(h(x))$

Merkle-Damgård2(x) $(t = 1)$

external compress : $\{0, l\}^{m+1} \to \{0,1\}^m$, $\begin{cases} f(0) = 0, \\ f(1) = 01. \end{cases}$

$n \leftarrow |x|$

$y \leftarrow 11||f(x_1)||f(x_2)||\cdots||f(x_n)$

令 $y = y_1||y_2||\cdots||y_k$, 其中 $y_i \in \{0,1\}, 1 < i < k$

$g_1 \leftarrow \text{compress}(0^m||y_1)$

for $i \leftarrow 1$ to $k-1$

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

return (g_k)

SHA-1 结构

1. 输入长度 $|x| < 2^{64} - 1$;
2. 输出长度: 160 比特
3. 每个运算都是面向字 (32 比特) 的操作
4. 填充方案: SHA-1-PAD(x) 把消息 x 转化为 512 倍数长的串

   $$y \leftarrow x||1||0^d||l,$$

   这里 $d \leftarrow (447 - |x|) \bmod 512$, $l \leftarrow |x|$ 的二进制表示, 而且长度为 64 比特 (必要时左边填充 0).
5. 所用操作为:

$X \wedge Y$	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} 的整数和
$\mathrm{ROTL}^s(X)$	X 循环左移 s 个位置 $(0 \leq s \leq 31)$

函数 f_t 和常数 K_t

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

函数 f_t 和常数 K_t

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

$$K_t = \begin{cases} \text{5A827999}, & 0 \leq t \leq 19 \\ \text{6ED9EBA1}, & 20 \leq t \leq 39 \\ \text{8F1BBCDC}, & 40 \leq t \leq 59 \\ \text{CA62C1D6}, & 60 \leq t \leq 79 \end{cases}$$

算法: SHA-1(x)

external SHA-1-PAD

global $K_0, K_1, \cdots, K_{79}$

$y \leftarrow \text{SHA-1-PAD}(x)$

令 $y = M_1||M_2||\cdots||M_n$, 这里每个 $|M_i| = 512$

$H_0||H_1||H_2||H_3||H_4 \leftarrow 67452301||\text{EFCDAB89}||\text{98BADCFE}||10325476||\text{C3D2E1F0}$

for $i \leftarrow 1$ to n

$$
\text{do} \begin{cases}
\text{令 } M_i = W_0||W_1||\cdots||W_{15} \quad // \ |W_i| = 32 \\
\text{for } t \leftarrow 16 \text{ to } 79 \\
\quad \text{do } W_t \leftarrow \text{ROTL}^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \\
A \leftarrow H_0, \quad B \leftarrow H_1, \quad C \leftarrow H_2, \quad D \leftarrow H_3, \quad E \leftarrow H_4 \\
\text{for } t \leftarrow 0 \text{ to } 79 \\
\quad \text{do} \begin{cases}
\text{temp} \leftarrow \text{ROTL}^5(A) + f_t(B, C, D) + E + W_t + K_t \\
E \leftarrow D \\
D \leftarrow C \\
C \leftarrow \text{ROTL}^{30}(B) \\
B \leftarrow A \\
A \leftarrow \text{temp}
\end{cases} \\
H_0 \leftarrow H_0 + A \quad H_1 \leftarrow H_1 + B \quad H_2 \leftarrow H_2 + C \quad H_3 \leftarrow H_3 + D \quad H_4 \leftarrow H_4 + E
\end{cases}
$$

return $(H_0||H_1||H_2||H_3||H_4)$

§4.4 消息认证码

消息认证码 (Message Authentication Code, 简写为 MAC):

1. 定义: 带密钥的 Hash 函数.
2. 用途: 用于消息完整性的认证, 因为只有密钥的拥有者才能生成消息认证码.
3. 构造方法:
    1. 由不带密钥的 Hash 函数构造, 如 HMAC;
    2. 由对称密码构造得到, 如 CBC-MAC.

MAC 算法的安全性要求与不带密钥的 Hash 函数要求不同, 通常使用假冒 (Forgery) 来描述其安全性.

(ϵ, Q) 假冒者

如果攻击者在获得 Q 个消息 $x_1, x_2, \cdots, x_Q$ 的消息认证码 $y_1, y_2, \cdots, y_Q$ 后, 能够至少以 ϵ 的成功概率得到一个有效对 $(x, h_K(x))$, $x \notin \{x_1, x_2, \cdots, x_Q\}$, 则称攻击者为一个 (对 h_K 的) (ϵ, Q) 假冒者, 有效对 $(x, h_K(x))$ 称为一个假冒.

MAC 算法的安全性要求与不带密钥的 Hash 函数要求不同, 通常使用假冒 (Forgery) 来描述其安全性.

(ϵ, Q) 假冒者

如果攻击者在获得 Q 个消息 $x_1, x_2, \cdots, x_Q$ 的消息认证码 $y_1, y_2, \cdots, y_Q$ 后, 能够至少以 ϵ 的成功概率得到一个有效对 $(x, h_K(x))$, $x \notin \{x_1, x_2, \cdots, x_Q\}$, 则称攻击者为一个 (对 h_K 的) (ϵ, Q) 假冒者, 有效对 $(x, h_K(x))$ 称为一个假冒.

显然, 一个好的 MAC 函数, 如果存在 (ϵ, Q) 假冒者, 当然希望成功概率 ϵ 越小越好, Q 越大越好.

(1, 1) 假冒者例子

假设带密钥的 Hash 函数 h_K 由压缩函数

$$\text{Compress}: \{0,1\}^{m+t} \rightarrow \{0,1\}^m$$

通过一般的迭代方法构造而成, 并且初始向量 $\text{IV} = K$ (为描述简单, 假定构造中没有预处理和输出变换步骤). 如果攻击者能够得到一有效对 $(x, h_K(x))$, 则对任意长度 t 的序列 x', 根据迭代计算过程知道, 攻击者可以计算

$$h_K(x \| x') = \text{compress}(h_K(x) \| x'),$$

即攻击者可以在不知道密钥 K 的情形下, 很容易得到有效对 $(x \| x', h_K(x \| x'))$.

$(1,1)$ 假冒者例子

假设带密钥的 Hash 函数 h_K 由压缩函数

$$\text{Compress}: \{0,1\}^{m+t} \to \{0,1\}^m$$

通过一般的迭代方法构造而成, 并且初始向量 $\text{IV} = K$ (为描述简单, 假定构造中没有预处理和输出变换步骤). 如果攻击者能够得到一有效对 $(x, h_K(x))$, 则对任意长度 t 的序列 x', 根据迭代计算过程知道, 攻击者可以计算

$$h_K(x||x') = \text{compress}(h_K(x)||x'),$$

即攻击者可以在不知道密钥 K 的情形下, 很容易得到有效对 $(x||x', h_K(x||x'))$.

这个例子表明在用不带密钥的 Hash 函数构造 MAC 函数时, 必须要非常小心.

直观地讲, 嵌套 MAC 的思想是通过一个安全的“小 MAC”和一个碰撞稳固的带密钥的 Hash 簇的复合来构建一个安全的“大 MAC”, 具体定义如下:

复合 Hash 簇定义

假设 $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{G})$ 和 $(\mathcal{Y}, \mathcal{Z}, \mathcal{L}, \mathcal{H})$ 是两个 Hash 簇, 则它们的复合是指 Hash 簇 $(\mathcal{X}, \mathcal{Z}, \mathcal{M}, \mathcal{G} \circ \mathcal{H})$, 这里 $\mathcal{M} = \mathcal{K} \times \mathcal{L}$, 并且

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

之中对 $\forall\, x \in \mathcal{X}$, 有

$$g \circ h_{(K,L)}(x) = h_L(g_K(x)).$$

嵌套 MAC 的例子: HMAC

1. HMAC 是一个于 2002 年 3 月被提议作为 FIPS 标准的嵌套 MAC 算法;
2. HMAC 通过不带密钥的 Hash 函数来构造 MAC;
3. 基于 SHA-1 的 HMAC 算法描述如下:
    1. 密钥 K 长度: 512 比特
    2. 输出长度: 160 比特
    3. 2 个 512 比特常数: $\text{ipad} = 3636\cdots36$, $\text{opad} = 5C5C\cdots5C$

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

CBC-MAC 给出了由分组密码构造 MAC 的一种方法.

CBC-MAC 给出了由分组密码构造 MAC 的一种方法.

假设 $(\mathcal{P},\mathcal{P},\mathcal{K},\mathcal{E},\mathcal{D})$ 是一个内嵌式密码体制, 其中 $\mathcal{P}=\{0,1\}^t$, $K\in\mathcal{K}$ 为秘密密钥, 则 CBC-MAC(x,K) 的计算如下:

CBC-MAC(x,K)

令 $x = x_1||x_2||\cdots||x_n \quad // \ \ |x_i| = t$

IV$\leftarrow 00\cdots0$

$y_0 \leftarrow$ IV

for $i \leftarrow 1$ to n

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

return (y_n)

6.1. 试归纳密码学应用中 Hash 函数需满足的要求.

6.2. **(1)** 考虑下列 Hash 函数. 假设消息是十进制序列 $M = (a_1, a_2, \cdots, a_t)$, 对于某个预定义的 n, 计算 Hash 值

$$h_1(M) = \left(\sum_{i=1}^{t} a_i\right) \mod n.$$

问如上定义的 h_1 满足 Hash 函数的要求吗? 请给出解释.

(2) 对于 Hash 函数

$$h_2(M) = \left(\sum_{i=1}^{t} a_i^2\right) \mod n,$$

按 (1) 的要求做题.

(3) 当 $M = (189, 632, 900, 722, 349)$, $n = 989$ 时, 计算 $h_2(M)$.