

第 11 章：消息认证和 Hash 函数

本章要点

- 消息认证是用来验证消息完整性的一种机制或服务。消息认证确保收到的数据确实和发送时的一样（即没有修改、插入、删除或重放），且发送方声称的身份是真实有效的。
- 对称密码在那些相互共享密钥的用户间提供认证。用消息发送方的私钥加密消息也可提供一种形式的认证。
- 用于消息认证的最常见的密码技术是消息认证码和安全 Hash 函数。
- MAC 是一种需要使用秘密密钥的算法，以可变长的消息和秘密密钥作为输入，产生一个认证码。拥有秘密密钥的接收方产生一个认证码来验证消息的完整性。
- Hash 函数将可变长度的消息映射为固定长度的哈希值，或叫消息摘要。对于消息认证码，安全 Hash 函数必须以某种方式和秘密密钥捆绑起来。

主要内容

1. 对消息认证的要求
2. 消息加密
3. 消息认证码
4. 安全 Hash 函数
5. 安全 Hash 算法
6. HMAC 算法

主要内容

1. 对消息认证的要求

2. 消息加密

3. 消息认证码

4. 安全 Hash 函数

5. 安全 Hash 算法

6. HMAC 算法

网络通信环境中的攻击

1. **泄密**：将消息透露给没有密钥的第三方
2. **传输分析**：分析双方通信模式
3. **伪装**：欺诈源向网络中插入一条消息
4. **内容篡改**：对消息内容的修改
5. **顺序篡改**：对消息顺序的修改
6. **计时篡改**：对消息的延时和重放
7. **信源抵赖**：发送方否认发送过某消息
8. **信宿抵赖**：接收方否认接收过某消息

网络通信环境中的攻击

- 1. **泄密**：将消息透露给没有密钥的第三方
 - 2. **传输分析**：分析双方通信模式
 - 3. **伪装**：欺诈源向网络中插入一条消息
 - 4. **内容篡改**：对消息内容的修改
 - 5. **顺序篡改**：对消息顺序的修改
 - 6. **计时篡改**：对消息的延时和重放
 - 7. **信源抵赖**：发送方否认发送过某消息
 - 8. **信宿抵赖**：接收方否认接收过某消息
- } 消息保密

网络通信环境中的攻击

- 1. **泄密**：将消息透露给没有密钥的第三方
 - 2. **传输分析**：分析双方通信模式
 - 3. **伪装**：欺诈源向网络中插入一条消息
 - 4. **内容篡改**：对消息内容的修改
 - 5. **顺序篡改**：对消息顺序的修改
 - 6. **计时篡改**：对消息的延时和重放
 - 7. **信源抵赖**：发送方否认发送过某消息
 - 8. **信宿抵赖**：接收方否认接收过某消息
- 消息保密
- 消息认证


网络通信环境中的攻击

- 1. 泄密：将消息透露给没有密钥的第三方
 - 2. 传输分析：分析双方通信模式
 - 3. 伪装：欺诈源向网络中插入一条消息
 - 4. 内容篡改：对消息内容的修改
 - 5. 顺序篡改：对消息顺序的修改
 - 6. 计时篡改：对消息的延时和重放
 - 7. 信源抵赖：发送方否认发送过某消息
 - 8. 信宿抵赖：接收方否认接收过某消息
- 消息保密
- 消息认证
- 数字签名

网络通信环境中的攻击




- 1. 泄密：将消息透露给没有密钥的第三方
 - 2. 传输分析：分析双方通信模式
 - 3. 伪装：欺诈源向网络中插入一条消息
 - 4. 内容篡改：对消息内容的修改
 - 5. 顺序篡改：对消息顺序的修改
 - 6. 计时篡改：对消息的延时和重放
 - 7. 信源抵赖：发送方否认发送过某消息
 - 8. 信宿抵赖：接收方否认接收过某消息
- 消息保密
- 消息认证
- 数字签名
- 数字签名 + 其他手段

举例：Linux Mint 下载页面

 **Linux Mint**
from freedom came elegance

Home **Download** Project About Links

Linux Mint 20 LMDE 4 All versions Documentation

  
Donate Participate Download

How to verify ISO images

This page explains how to verify their integrity and authenticity.

It is important to verify the integrity and authenticity of your ISO image.

The integrity check confirms that your ISO image was properly downloaded and that your local file is an exact copy of the file present on the download servers. An error during the download could result in a corrupted file and trigger random issues during the installation.

The authenticity check confirms that the ISO image you downloaded was signed by Linux Mint, and thus that it isn't a modified or malicious copy made by somebody else.


Note: If you are using Windows, following this tutorial: [How to verify the ISO image on Windows](#) .

Preparation

1. Create a directory called "ISO" in your home directory.
2. Move the ISO image you downloaded in this directory.
3. Download the following files (right-click -> Save As...) and move them into the "ISO" directory.

FILE	DESCRIPTION
sha256sum.txt	Contains the SHA256 sums to check the integrity of the ISO images.
sha256sum.txt.gpg	Signed by the Linux Mint team to check the authenticity of the sha256sum.txt file.

Follow us



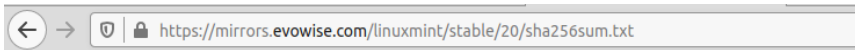
Sponsors

15% OFF Everything

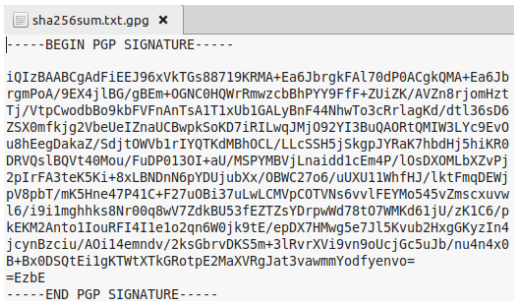
forkfest.party: a decentralized camping festival for freedom

Advertisements

举例：Linux Mint 下载页面



```
2f6ae466ec9b7c6255e997b82f162ae88bfe640a8df16d3e2f495b6281120af9 *linuxmint-20-cinnamon-64bit.iso
42fd764b3a3544a36d820f4164bb64aa5a6d982073e6dlafdea4853d3858fc98 *linuxmint-20-mate-64bit.iso
761fb276da9746a068f4c8aa42e8d4981f352db92babe0ef8a08713eeb38246f *linuxmint-20-xfce-64bit.iso
```



举例：Linux Mint 下载页面



Warning — Linux Mint Website Hacked and ISOs replaced with Backdoored Operating System

February 21, 2016 Swati Khandelwal



Popular This Week

Google Discloses Poorly-Patched, Now Unpatched, Windows 0-Day Bug

A New SolarWinds Flaw Likely Had Let Hackers Install SUPERNova Malware

Microsoft Warns CrowdStrike of Hackers Targeting Azure Cloud Customers

Police Arrest 21 WeLeakInfo

消息认证 (Message Authentication)

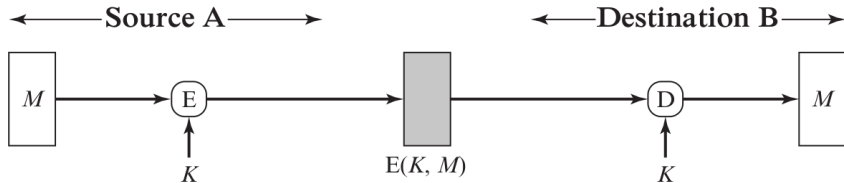
- 消息认证就是验证所收到的消息确实来自真正的发送方且未被修改，也可以验证消息的顺序和时效。
- 消息认证（报文认证）关心的问题是：
 - 保护消息的完整性
 - 验证发起方身份
 - 消息源的不可否认（解决分歧）
- 三种消息认证的方法：
 - 消息加密
 - 消息认证码
 - Hash 函数

主要内容

1. 对消息认证的要求
2. 消息加密
3. 消息认证码
4. 安全 Hash 函数
5. 安全 Hash 算法
6. HMAC 算法

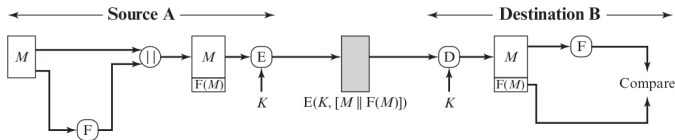
对称加密作为认证手段

- 接收方可以确信消息是由发送方产生的，因为除了接收方以外只有发送方拥有加密密钥，产生出用此密钥可以解密的密文。
- 如果消息可以是任意的位模式，接收方无法确定收到的消息是合法明文的密文。因此，通常不管密文的价值是什么，如果解密后得到的明文有合法明文的位模式，接收方都会作为真实的密文接收。

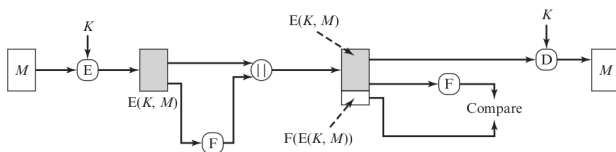


解决解密所得消息是否具有可读性问题

- 要求明文具有某种易于识别的结构，如在加密前对每个消息附加一个**帧校验序列 FCS**；
- **内部差错控制**：由于攻击者很难产生密文，使得解密后其 FCS 是正确的，因此可提供认证；
- **外部差错控制**：攻击者可构造具有正确 FCS 的消息，虽然攻击者不知道解密后的明文是什么，但他可破坏通信。



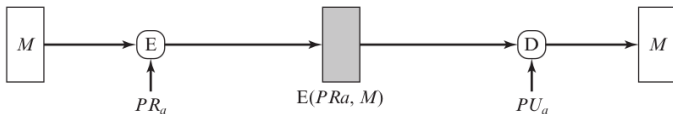
(a) Internal error control



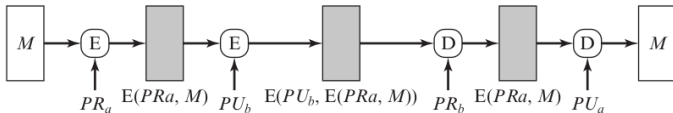
(b) External error control

公钥加密作为认证手段

- 若要提供认证，发送方用自己的私钥对消息加密，接收方用发送方的公钥解密 (验证)，就提供了认证功能。
- 如果发送方用私钥加密消息，再用接收方的公钥加密，就实现了既保密又认证的通信。
- 既保密又认证的通信的代价是一次通信需要执行四次复杂的公钥算法而不是两次。



(c) Public-key encryption: authentication and signature



(d) Public-key encryption: confidentiality, authentication, and signature

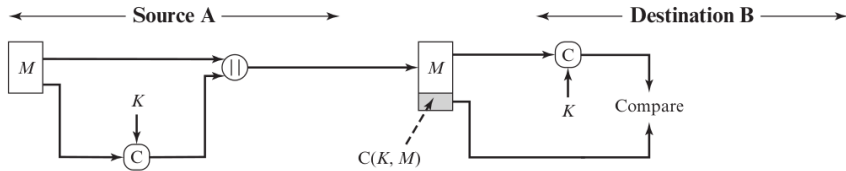
主要内容

1. 对消息认证的要求
2. 消息加密
3. 消息认证码
4. 安全 Hash 函数
5. 安全 Hash 算法
6. HMAC 算法

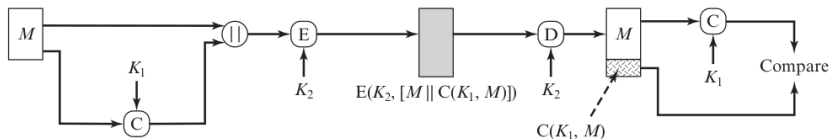
消息认证码 MAC

- 又称**密码校验和**或 MAC：利用密钥 K 生成消息 M 的一个定长短数据块 $MAC = C(K, M)$ ，并将 MAC 附加在消息后。
- 接收方对收到的消息重新计算 MAC，并与接收到的 MAC 比较。因为只有双方知道密钥，如果两个 MAC 匹配，则：
 - 接收方可以确信报文未被更改；
 - 接收方可以确信报文来自声称的发送者；
 - 接收方可以确信报文序号正确，如果有的话。
- MAC 函数类似加密，但非数字签名，也无需可逆；
- 将 MAC 直接与明文并置，然后加密传输比较常用。

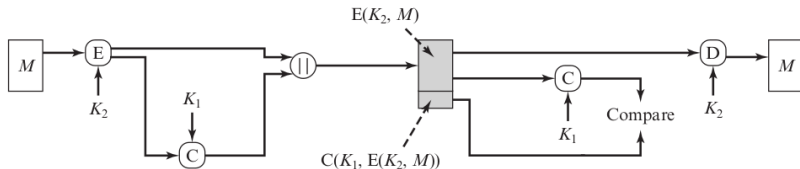
消息认证码的几种工作方式



(a) Message authentication



(b) Message authentication and confidentiality; authentication tied to plaintext



(c) Message authentication and confidentiality; authentication tied to ciphertext

必须使用消息认证码的几种情形

- 相同消息进行多点广播，以明文加对应 MAC 的形式进行广播，只要一个接收者负责鉴别（该接收者拥有密钥），不正确时发出告警；
- 接收方负荷很大，无法对所有收到的消息进行解密工作，则可以进行有选择地鉴别，对消息作随机检查；
- 对明文计算机程序进行鉴别，检查完整性；
- 某些应用不关注消息的保密而更重视鉴别消息的真实性，如 SNMPv3，将保密与鉴别分开；
- 保密函数与鉴别函数的分离能提供结构上的灵活性，如在应用层完成鉴别而在较低层加密；
- 在超过接收时间后继续延长保护期限，确保消息在接收方保存期间不被修改。

消息认证码的特点

- MAC 是一种多对一函数，类似于加密函数，但不要求可逆。
- 定义域由任意长的消息组成，值域由所有可能的 MAC 组成。
- 若使用 n 位长的 MAC，则有 2^n 个可能的 MAC。有 N 条可能的消息， $N \gg 2^n$ 。若密钥长度为 k ，则有 2^k 种可能的密钥，即 2^k 种可能的映射关系。
- 如 N 为 2^{100} ， n 为 10，共有 2^{10} 种不同的 MAC，平均每一个 MAC 可由 $2^{100}/2^{10} = 2^{90}$ 条不同的消息产生。若密钥长度为 5，则从消息集合到 MAC 值的集合有 $2^5 = 32$ 种不同映射。
- 与加密相比，认证函数更不荣易被攻破。

对消息认证码的要求

- 若攻击者已知 M 和 $C(K, M)$ ，则构造满足 $C(K, M') = C(K, M)$ 的消息 M' 在计算上是不可行的；
- $C(K, M)$ 应该是均匀分布的，即对任何随机选择的消息 M 和 M' ， $C(K, M') = C(K, M)$ 的概率是 2^{-n} ，其中 n 是 MAC 的位数；
- 设 M' 是 M 的某个已知的变换，即 $M' = f(M)$ ，如 f 可能表示逆转 M 的一位或多位，那么

$$Pr[C(K, M) = C(K, M')] = 2^{-n}$$

直观理解：只要两消息不相同，其 MAC 值相同的概率都只是 2^{-n} 。

MAC 的安全性

- 攻击者如何用穷举的方法找到密钥？
- 假设攻击者可以访问明文及其 MAC；
- k 为密钥长度， n 为 MAC 长度， $k > n$ ；
- 对满足 $T_1 = C(K, M_1)$ 的 M_1 和 T_1 ，攻击者需要穷举所有可能的密钥值 K_i ，计算 $T_i = C(K_i, M_1)$ ，那么至少会找到一个密钥使得 $T_i = T_1$ 。
 - 这里一共产生 2^k 个 MAC，只有 2^n 个不同 MAC；
 - 许多密钥会产生相同 MAC，而攻击者不知道哪个是正确密钥；
 - 平均来说，有 $2^k/2^n = 2^{k-n}$ 个密钥会产生正确 MAC，因此攻击者对这些密钥必须做重复攻击；
 - 攻击者需要用另一个（消息-MAC）对： $T_2 = C(K, M_2)$ ，对找到的 2^{k-n} 个密钥计算 $T_i = C(K_i, M_2)$ ，得到 2^{k-2n} 个密钥。
- 若 $k = \alpha n$ ，则需 α 次循环，才能找到正确密钥。

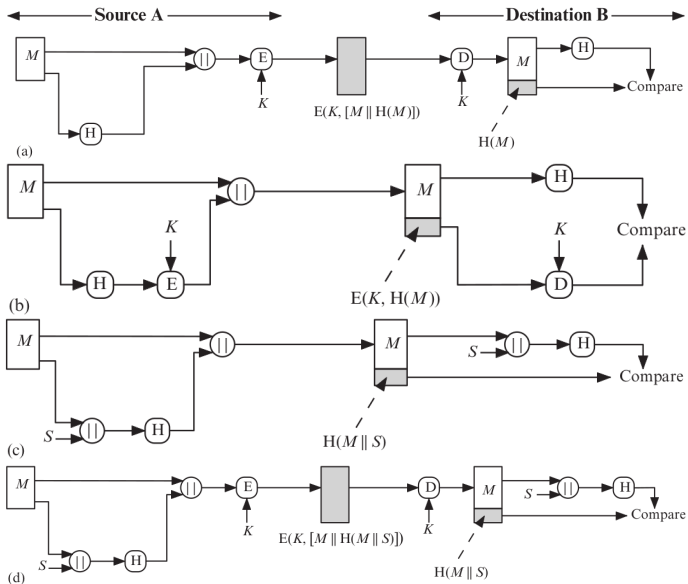
主要内容

1. 对消息认证的要求
2. 消息加密
3. 消息认证码
4. 安全 Hash 函数
5. 安全 Hash 算法
6. HMAC 算法

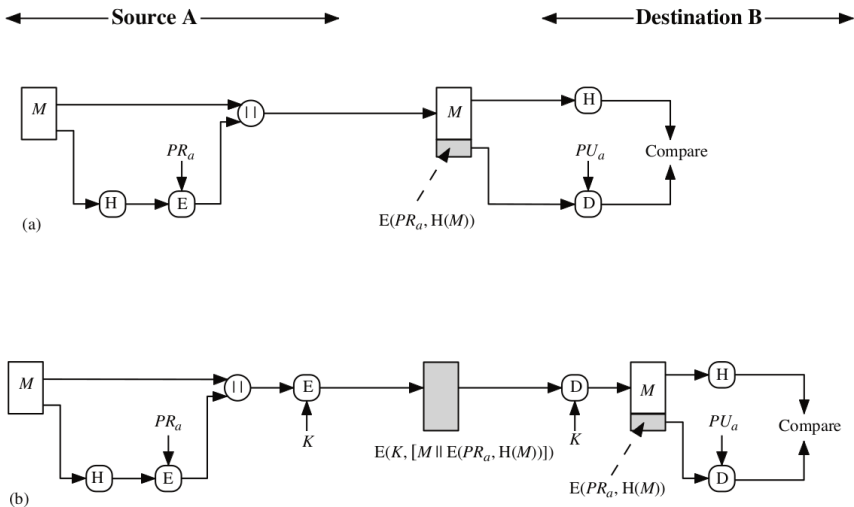
密码学 Hash 函数

- 一个 Hash 函数以变长的消息 M 作为输入，产生定长的哈希码 $H(M)$ 作为输出，哈希码 $H(M)$ 亦称作**报文摘要**。
- 哈希码是消息所有比特的函数值，具有差错检测能力，消息任意一比特的改变都将引起哈希码的改变。
- 不同的哈希码使用方式：
 - 对附加了哈希码的消息进行加密
 - 使用常规加密方法仅对哈希码加密
 - 使用公开密钥方法仅对哈希码加密，提供数字签名
 - 同时提供保密和签名，可以分别使用常规方法加密消息及使用公开密钥方法加密哈希码
- 不希望使用加密函数的理由：加密过程很慢、硬件开销大、初始化的开销、专利问题、出口限制

Hash 函数用于消息认证



Hash 函数用于数字签名



密码学 Hash 函数的安全性要求

1. H 可以应用于任意大小的数据块；
2. H 产生固定长度的输出；
3. 对任意给定的明文 x ，计算 $H(x)$ 容易；
4. **单向性**：对任意给定的哈希码 h ，找到满足 $H(x) = h$ 的 x ，在计算上不可行（即由哈希码找消息不可行）；
5. **抗弱碰撞性**：对任何给定的消息 x ，找到满足 $y \neq x$ 且 $H(x) = H(y)$ 的消息 y ，在计算上不可行（即给定一个消息，找另外一个消息，使它们有相同的哈希码不可行）；
6. **抗强碰撞性**：找到任何满足 $H(x) = H(y)$ 的消息对 (x, y) ，在计算上不可行。

弱 Hash 函数 vs. 强 Hash 函数

- 条件 1, 2, 3, 4 是所谓单向性问题 (One-way);
- 条件 5, 6 是对使用的哈希值的数字签名方法所做的安全保障, 否则攻击者可由已知的明文及相关的数字签名任意伪造对其他明文的签名;
- 条件 6 主要用于防范所谓的生日攻击法;
- 能满足条件 1-5 的, 称为弱 Hash 函数;
- 能同时满足条件 6 的, 称为强 Hash 函数;
- 应用在数字签名上的 Hash 函数必须是强 Hash 函数。

对 Hash 函数的原像攻击

- **原像攻击**：攻击者对给定的 Hash 值 h ，试图找到满足 $H(y) = h$ 的消息 y 。
- **穷举攻击法**：攻击者随机选择 y ，计算其 Hash 值，直到碰撞出现。
- 考虑一般情况：
 - 若一个函数可能有 n 个函数值，且已知一个函数值 $H(x)$ ，任选 k 个任意数作为函数输入值。
 - 问题： k 必须多大才能保证至少找到一个输入值 y ，且 $H(x) = H(y)$ 的概率大于 $1/2$ ？
 - 结论：当 $k > n/2$ 时，碰撞概率将超过 $1/2$ 。

对 Hash 函数的原像攻击

- 对于任意 y ，满足 $H(x) = H(y)$ 的概率是 $1/n$ ，不满足的概率是 $1 - 1/n$ 。 k 个任意输入没有一个满足 $H(x) = H(y)$ 的概率是 $(1 - 1/n)^k$ ，至少有一个满足的概率是 $1 - (1 - 1/n)^k$ 。根据二项式定理

$$(1 - a)^k = 1 - ka + \frac{k(k-1)}{2!}a^2 - \frac{k(k-1)(k-2)}{3!}a^3 \dots$$

当 $a \rightarrow 0$ 时， $(1 - a)^k \rightarrow 1 - ka$ 。

- 所以，至少有一个 y 满足 $H(x) = H(y)$ 的概率几乎等于 $1 - (1 - k/n) = k/n$ 。当 $k > n/2$ 时，这个概率将超过 $1/2$ 。
- Robin 攻击方法：64 位的 Hash 函数，有 2^{64} 种组合，攻击者只要尝试 $k > 2^{64}/2 = 2^{63}$ 个消息，就有可能获得超过 $1/2$ 的成功机会。

对 Hash 函数的碰撞攻击

- 碰撞攻击：攻击者试图找到两个消息 x 和 y ，满足 $H(x) = H(y)$ ，也称为生日攻击（Birthday Attack）。
- 可以证明，碰撞攻击的穷举规模要比原像攻击的穷举规模小很多。
- 生日悖论（Birthday Paradox）：
 - 23 个人当中，存在具有相同生日的两个人的概率大于 50%；
 - 50 个人当中，存在具有相同生日的两个人的概率大于 96%。
- Yuval 提出用生日悖论对 Hash 函数进行碰撞攻击，证明 64 位 Hash 函数只需要 2^{32} 次运算就有可能获得超过 1/2 的成功机会。

生日攻击的数学背景

在 1 到 n 范围内有放回的随机取出 k 个整数，当 k 为多大时可以保证取出的 k 个数有重复的概率大于 $1/2$ ？

从 $[1, n]$ 范围中随机取出 k 个整数， $k \leq n$ ，数字互不相同的概率为

$$\frac{n(n-1)\dots(n-k+1)}{n^k} = \prod_{i=0}^{k-1} \left(1 - \frac{i}{n}\right) \leq \prod_{i=0}^{k-1} e^{-i/n} = e^{-\frac{k(k-1)}{2n}}$$

利用了不等式 $1 - x \leq e^{-x}$, $x \in [0, 1]$ 。则至少两个数相同的概率为：

$$p(n, k) \geq 1 - e^{-\frac{k(k-1)}{2n}}$$

现在希望上述概率不小于 0.5，所以

$$1 - e^{-\frac{k(k-1)}{2n}} \geq 0.5$$

得出

$$k^2 > k(k-1) \geq -2n \ln 0.5$$

所以

$$k > 1.17\sqrt{n}.$$

生日悖论分析

- 问题：一共有 k 个人，希望在其中找到至少两个具有相同生日的人。当 k 多大时，这个概率大于 $1/2$?
- k 个人的生日总排列数是 365^k ， k 个人有不同生日的总排列数为： $365!/(365 - k)!$ 。
 - 第一个人有 365 种生日选择，第二个人有 364 种选择，...
- k 个人有不同生日的概率为

$$Q(365, k) = \frac{365!/(365 - k)!}{365^k}$$

- k 个人至少有两个人具有相同生日的概率为

$$P(365, k) = 1 - Q(365, k)$$

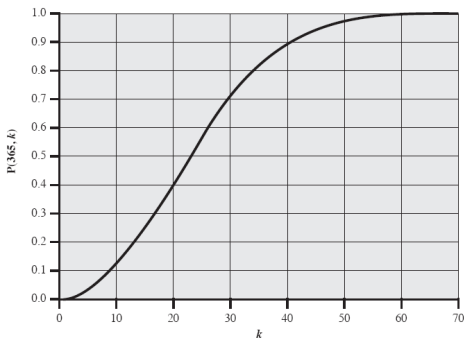
- 计算得

$$P(365, 23) = 0.5073$$

$$P(365, 50) = 0.9704$$

$$P(365, 100) = 0.999997$$

生日悖论分析



- 在 23 个人当中，考虑某一个人的特定生日，在剩下的 22 个人中能找到相同特定生日的概率是很小的，即使剩余每个人都有不同的生日，也只有 $22/365$ 的概率；
- 但是只考虑同一天出生，23 个人会产生 $\binom{23}{2} = 253$ 种不同的组合。所以只要 23 个人，找到两人同一天生的概率 $253/365 > 1/2$ 。

主要内容

1. 对消息认证的要求
2. 消息加密
3. 消息认证码
4. 安全 Hash 函数
5. 安全 Hash 算法
6. HMAC 算法

安全哈希算法 (SHA)

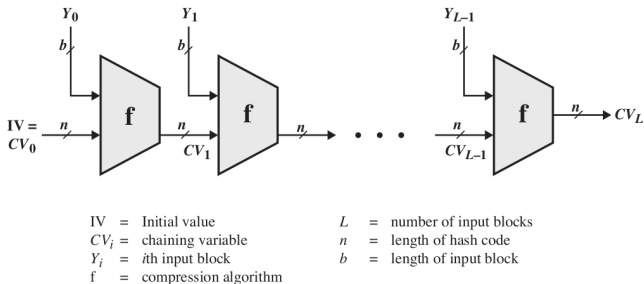
- SHA 是由美国标准与技术协会 NIST 设计，并于 1993 作为联邦标准 FIPS 180 发布，1995 年修订为 FIPS 180-1，即 SHA-1。
- SHA 算法建立在 MD4 算法之上，基本框架与 MD4 类似。
- SHA-1 产生 160 位哈希值，以后的修订版分别为 SHA-256，SHA-384，SHA-512，与 SHA-1 有相同的基础结构。
- 2005 年 NIST 宣布了逐步废弃 SHA-1 的意图；一周后，王小云在 Crypto 2005 会议上发表 Finding Collisions in the Full SHA-1，宣布了一种攻击方法，用 2^{69} 次操作找到两个独立的消息具有相同哈希值（此前认为需要 2^{80} 次操作）。这一结果加速了向 SHA 其他版本的过渡。

SHA 参数比较

Algorithm	Message Size	Block Size	Word Size	Message Digest Size
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

Note: All sizes are measured in bits.

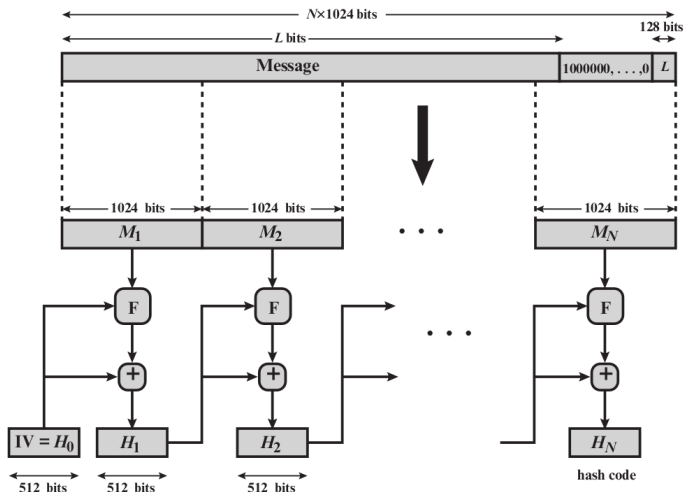
典型的安全 Hash 函数的总体结构



- 这种结构称为**迭代 Hash 函数**，由 Merkle 提出；
- 包括 SHA 在内的大多数 Hash 函数都是基于这种结构；
- 将输入消息分为 L 个固定长度的分组，每组 b 位；
- 最后一个分组不足 b 位要填充，最后一个分组包含总长度；
- 重复使用压缩函数 f ，其输入包括：前一步得出的 n 位结构（称为链接变量 CV）和一个 b 位分组，输出为一个 n 位分组。

SHA-512 逻辑

- 算法的输入是最大长度小于 2^{128} 位的消息，输出是 512 位的消息摘要，输入消息以 1024 位的分组为单位处理。



SHA-512 处理步骤

1. **附加填充位** 使消息长度 $\equiv 896(\text{mod } 1024)$ ，填充位数在 1 到 1024 之间，由 1 和后续的 0 组成；
2. **附加长度** 在消息后附加 128 位的块，将其看作 128 位无符号整数，代表填充前消息的长度；
3. **初始化哈希缓冲区** 8 个 64 位寄存器 (8 个字)

$a = 6A09E667F3BCC908$ $e = 510E527FADE682D1$
 $b = BB67AE8584CAA73B$ $f = 9B05688C2B3E6C1F$
 $c = 3C6EF372FE94F82B$ $g = 1F83D9ABFB41BD6B$
 $d = A54FF53AF1D336F1$ $h = 5BE0CD19137E2179$

4. **以 1024 比特的分组 (16 个字) 为单位处理消息** 核心是具有 80 轮运算的模块，每一轮都把 512 位缓冲区的值 $abcdefg$ 作为输入，并更新缓冲区的值；
5. **输出** 所有 N 个 1024 比特分组都处理完后从第 N 阶段输出的是 512 比特的消息摘要。

SHA-512 的运算

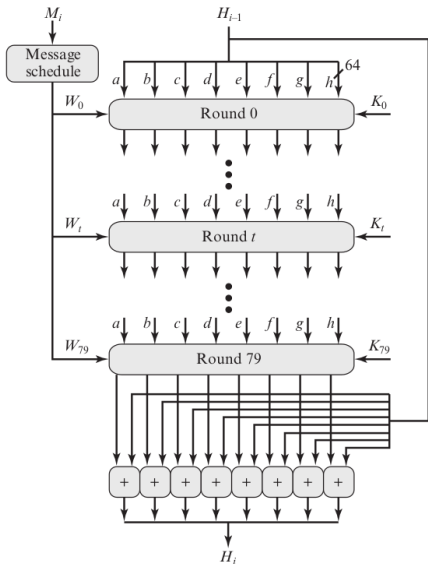
$$H_0 = IV$$

$$H_i = \text{SUM64}(H_{i-1}, abcdefgh_i)$$

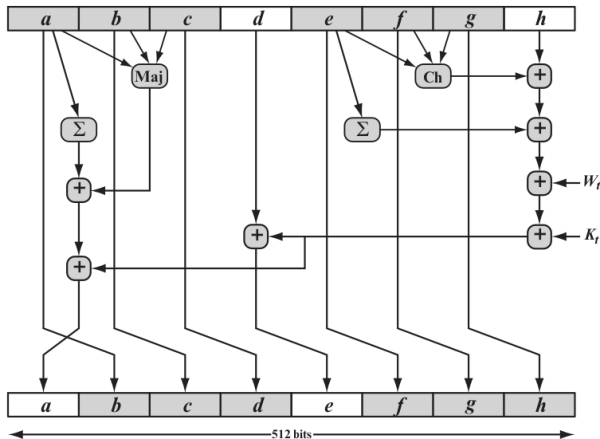
$$MD = H_N$$

其中：

- IV ：缓冲区初值，
- $abcdefgh_i$ ：第 i 个消息分组处理的输出，
- N ：消息中的分组数，
- SUM64 ：模 2^{64} 加，
- MD ：最后的消息摘要，
- $K_t, t = 0, \dots, 79$ 为轮常数，
用来使每一轮的运算不同，
- $W_t, t = 0, \dots, 79$ 由消息导出。

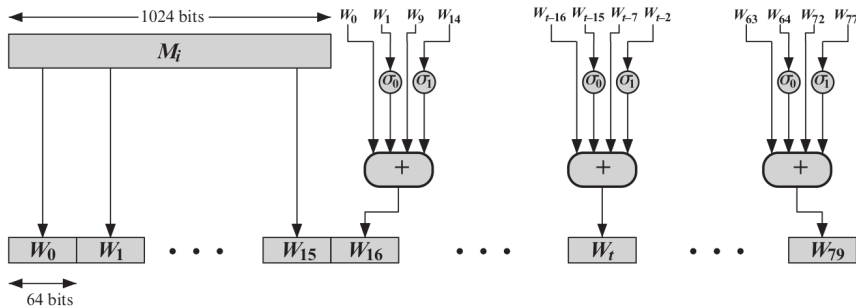


SHA-512 轮函数



- 输出 8 个字中的 6 个是简单的轮转置换（阴影部分）；
- 输出中只有 2 个字是通过替代置换产生的；
- Σ , Maj 和 Ch 为复杂二进制运算。

W_t 的导出



- 前 16 个 W_t 直接取自当前分组的 16 个字，
- 余下的 64 个 W_t 由前面的 4 个 W_t 推导得出，
- σ_0, σ_1 为复杂二进制运算。

SHA-512 的特点

- Hash 码的每一位都是全部输入位的函数，
- 基本函数 F 多次复杂重复运算使得结果充分混淆，
- 除非 SHA-512 中存在尚未公开的隐藏缺陷：
 - 找到两个具有相同 Hash 码的消息的复杂度为 2^{256} 次操作
 - 给定 Hash 码，寻找消息的复杂度为 2^{512} 次操作

主要内容

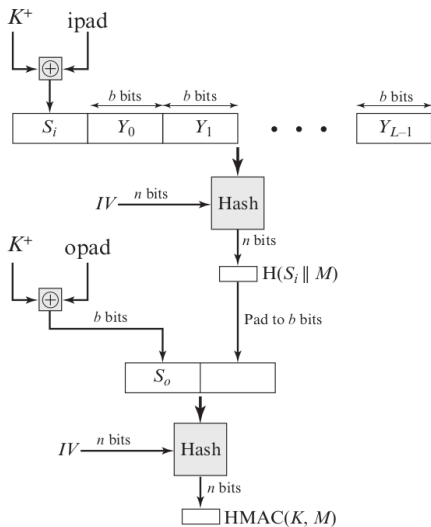
1. 对消息认证的要求
2. 消息加密
3. 消息认证码
4. 安全 Hash 函数
5. 安全 Hash 算法
6. HMAC 算法

基于 Hash 函数的 MAC: HMAC

- 人们越来越感兴趣利用密码学 Hash 函数设计 MAC，因为：
 - Hash 函数软件执行速度比对称分组密码快；
 - 有许多密码学 Hash 函数代码库。
- HMAC (RFC2104) 给出 HMAC 的设计目标是：
 - 不必修改而直接使用现有的 Hash 函数；
 - 用新 Hash 函数替代原来嵌入的 Hash 函数很容易；
 - 能保持 Hash 函数原有特性，不过份降低其性能；
 - 对密钥的使用和处理应较简单；
 - 如果已知嵌入的 Hash 函数的强度，则完全可知认证机制抗密码分析的强度。

HMAC 算法

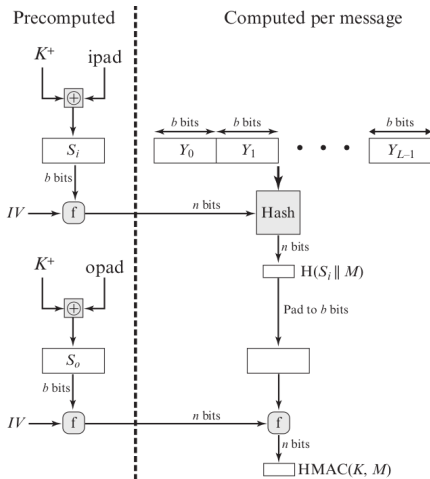
1. 在 K 左边填充 0，得到 b 位的 K^+ ；
2. K^+ 与 ipad 执行异或，产生 b 位分组 S_i ；
3. M 附于 S_i 后；
4. 将 H 作用于步骤 3 所得的结果；
5. K^+ 与 opad 执行异或，产生 b 位分组 S_o ；
6. 将步骤 4 中的哈希码附于 S_o 后；
7. 将 H 作用于步骤 6 所得的结果并输出该函数值：



$$\text{HMAC}(K, M) = H \left[(K^+ \oplus \text{opad}) \parallel H[(K^+ \oplus \text{ipad}) \parallel M] \right]$$

通过预计算提高 HMAC 效率

- 相比于只对消息分组 Hash, HMAC 多执行了三次压缩函数 f (对 S_i , S_o 和内部 Hash 产生的分组);
- 对于长消息, HMAC 和嵌入的 Hash 函数的执行时间大致相同;
- 对于短消息, 可以通过预计算其中的两个压缩函数来提高效率;
- 这样只多执行了一次压缩函数。



HMAC 的安全性

- 任何建立在嵌入 Hash 函数基础上的 MAC，其安全性在某种程度上依赖于该 Hash 函数的强度；
- 攻击 HMAC 意味着：
 - 对密钥的穷举攻击；
 - 生日攻击。
- M.Bellar 等已经证明，如果攻击者已知若干（时间，消息-MAC）对，则成功根据 HMAC 的概率等价于对嵌入 Hash 函数的下列攻击之一：
 - 即使 IV 是随机、秘密和未知的，攻击者也能够计算压缩函数的输出；
 - 即使 IV 是随机、秘密和未知的，攻击者也能够找到 Hash 函数中的碰撞。

小结

1. 对消息认证的要求
2. 消息加密
3. 消息认证码
4. 安全 Hash 函数
5. 安全 Hash 算法
6. HMAC 算法