第 3 章

CHAPTER 3

数据完整性技术

科技前沿——区块链技术在数据完整性保护中的应用

在当今数字化时代，数据已成为社会和经济的核心驱动力。确保数据在传输、存储和处理过程中的完整性，不仅是信息网络安全的基础，也是保障各行业正常运作的关键。传统的数据完整性技术主要依赖于哈希函数、消息认证码(MAC)和数字签名，虽然能够在一定程度上保护数据的完整性，但随着数据规模的增加和网络攻击的日益复杂，仍然存在一定的局限性。区块链技术的出现，为数据完整性保护带来了革命性的突破。

区块链是一种去中心化的分布式账本技术，通过链式结构和共识机制来确保数据的不可篡改性。其独特的工作原理是将数据记录分成多个区块，每个区块都包含前一个区块的哈希值，这样形成了一条连续不断的“链”。一旦某个区块中的数据被篡改，其哈希值将发生变化，进而影响后续所有区块的哈希链接。因此，任何未授权的修改都会被立即察觉，从而确保数据在整个区块链网络中的完整性和可靠性。

区块链的一个显著特点是分布式存储。数据不再集中于单一服务器，而是同步存储在多个节点上。每个节点都拥有相同的账本副本，通过共识算法来决定数据的有效性，这使得数据即使在一个节点上被篡改，也无法影响整个网络的数据完整性。正因为如此，区块链实现了数据防篡改和高可靠性的特性，这在金融、供应链管理、医疗健康和知识产权保护等领域有着广泛的应用前景。区块链技术的原理如图 3-1 所示。

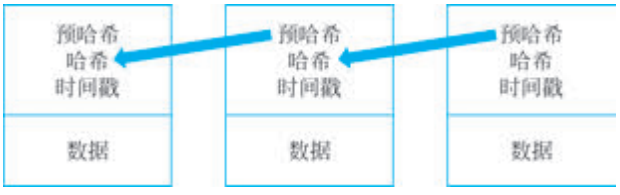

图 3-1　区块链技术的原理

更为重要的是，区块链不仅提供了强大的数据完整性保障，还通过智能合约技术进一步提升了数据处理的自动化和可信度。智能合约是一种基于区块链的自动化执行协议，它能够在满足特定条件时自动执行预设的操作，无须人工干预。这意味着数据在被写入区块链的同时，还能够实现业务逻辑的自动化执行，确保流程的透明性和不可篡改性。例如，在金融交易中，区块链和智能合约的结合可以确保交易记录的真实性和不可逆性，从而防止欺诈行为的发生。

此外，区块链技术的时间戳机制也是确保数据完整性的关键要素。每个区块的生成时间都会被记录下来，使得数据不仅无法被篡改，还可以追溯到其最初的创建时间。通过这种时间戳记录，数据在传输和共享过程中能够始终保持其真实性和完整性。这在医疗记录、电子合同和知识产权保护等需要高度可信数据的场景中尤为重要。

机密性、完整性和可用性是信息安全 CIA 三要素。第 2 章介绍的加密技术是保护机密性的主要手段。本章将探讨数据完整性技术，该技术通过多种手段来确保信息在存储或传输过程中的准确性、完整性和一致性，避免被未经授权的篡改或破坏。

本章首先分析数据完整性保护的重要性，明确数据完整性保护的定义，构建数据完整性系统模型，对比数据完整性与通信领域中常用的检错码技术；然后详细介绍采用对称密码技术和公钥密码技术来提供数据完整性保护的方法。基于对称密码技术的数据完整性保护可以通过密钥哈希（Hash）函数或者分组密码加密算法来实现；基于非对称密码技术的数据完整性保护可以通过数字签名和无源识别的数据完整性算法来实现。

本章学习目标：

（1）掌握数据完整性保护的基本概念；

（2）熟悉哈希函数的性质、特点和应用；

（3）理解数字签名的意义以及对数字签名的存在性伪造攻击原理；

（4）了解无源识别数据完整性算法 RSA-OAEP 的安全特点。

3.1 数据完整性技术概述

由计算机、设备和资源构成的大型网络是典型开放性的，这就意味着一个主体（或实体、代理、用户）能够接入这样的网络，并通过该网络来发送和接收消息，而不需要由“超级”主体授权。这里的主体可以是一台计算机，一个设备，一些资源、服务的提供者，一个人或一个组织等。在这样的开放性环境中，必然会有攻击者，他们会做各种坏事，不仅是被动地窃听消息，还会主动地改变、伪造、复制、删除或注入消息等。注入的消息可能是恶意的并对接收主体产生破坏性的影响。在安全领域，这些坏人叫主动攻击者。本章给攻击者一个特定的名称：Malice（和 Alice、Bob 对应），指经常戴着不同身份面具做坏事或捣蛋的人。Malice 可以是单个的，也可以是相互勾结的攻击团伙，甚至可能是网络中的一个合法主体。

如图 3-2 所示，当 Malice 通过篡改或伪造的消息，试图欺骗接收者，使其相信消息来自某个其他合法的主体时，这种破坏行为在许多应用场合中都会造成严重的后果，如电子支付、网上银行、股票交易等。例如，假设 Malice 在网络上窃听到 Alice 企业和一个 Bob 电子银行之间的一笔资金转账消息：请拨 1 亿

图 3-2 主动攻击者的目的

美元到 Alice 账户，Malice 悄悄地将信息修改为：请拨 1 亿美元到 Malice 账户，如果没有合适的安全机制来帮助 Bob 电子银行验证信息是否是 Alice 企业发的信息，而直接按信息忠实地转账，那么可想而知 Alice 企业的损失是不小的。

所以，需要一种安全机制，使得消息的接收者可以确信消息在传输过程中未受到任何未授权的篡改、毁坏或伪造，从而提供 OSI 安全架构中的数据完整性安全服务。数据完整性服务就是抗击对消息未授权修改的安全服务。

3.1.1 数据完整性保护定义

首先给出数据完整性保护的一个语法定义。

设 Data 为任意信息，Ke 为编码密钥，Kv 为与该编码密钥相匹配的验证密钥。Data 的数据完整性保护包括以下过程。

(1) 篡改检测码的生成：

$$MDC = f(Ke, Data)$$

(2) 篡改检测码的验证：

$$g(Kv, Data, MDC) = \begin{cases} True, & MDC = f(Ke, Data) \\ False, & MDC \neq f(Ke, Data) \end{cases}$$

其中，f 和 g 都是有效的密码变换，f 由编码密钥 Ke 参数化，g 由验证密钥 Kv 参数化。MDC 是 Manipulation Detection Code(篡改检测码)的缩写。

3.1.2 数据完整性保护系统模型

根据数据完整性保护的定义，图 3-3 描绘了数据完整性保护系统模型。

尽管图 3-3 采用了通信场合描绘的数据完整性保护系统模型，但数据完整性保护系统并不局限于通信场合。例如，当将数据存储到不安全的存储器中或者从一个不安全的数据存储器中恢复数据时，也可以产生(Data, MDC)。数据完整性保护并不一定要有发收的通信过程。

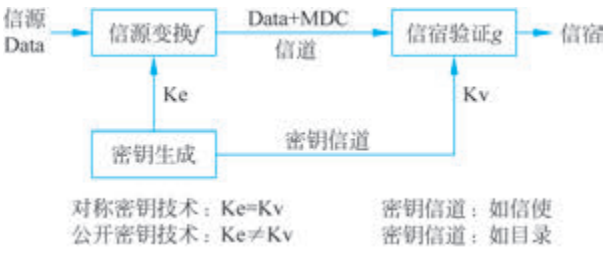

图 3-3 数据完整性保护系统模型

与密码系统的情况类似，数据完整性技术也分为对称密码技术和非对称密码技术。在基于对称密码技术的数据完整性保护系统中，编码密钥 Ke 和验证密钥 Kv 相同，对称密钥可通过双方信赖的信使进行分发，或者利用 DH 密钥交换算法共享一个密钥；在基于对称密码技术的数据完整性保护系统中，编码密钥 Ke 和验证密钥 Kv 不同，公钥(私钥)既可用于编码，又可用于验证。公钥可通过目录形式分发，私钥严格保密。

图 3-4 通信中的检错码技术

3.1.3 检错码技术的借鉴

数据完整性与通信中的经典主题检错码有密切的联系，并由其演变而来。如图 3-4 所示，在远距离通信过程中，为确保消息准确无误地传递，常采用检错码技术来检测并识别由信道噪声

或数据处理不当在接收端可能引发的错误情况。

通常认为，使用被恶意方式修改过的信息和使用由于通信或数据处理不当而导致的错误信息是同样危险的。因此，数据完整性技术的工作原理和检错码技术的工作原理本质上是相同的。在数据完整性技术中，信源端通过编码变换为消息增加一些冗余信息而生成 MDC，并将该值附在消息之后，信宿端根据与变换匹配的验证规则来检验消息的完整性。在检错码技术中，消息发送者通过检错编码为消息增加一些冗余来生成一个“校验和”，并附在消息之后，消息接收者根据与发送者协商好的规则来检验消息的正确性。

数据完整性技术和检错码技术的目的不同。数据完整性技术是检测因攻击者 Malice 的恶意操作所导致的消息错误。检错码技术是检测因信道噪声误差所导致的消息错误。

数据完整性技术和检错码技术所生成的冗余的作用不同。在数据完整性保护中，编码变换生成的冗余使得加入的 MDC 在整个空间中尽可能均匀分布，这使得攻击者伪造一个有效 MDC 的概率达到最小。在检错码中，通过编码加入的冗余度使得消息的接收者可以采用极大似然检测器来判定接收到的码字应该译为哪条消息，即最可能由改动过的码字得到的所发送的消息。

3.2 对称密码数据完整性技术

在实现数据完整性的对称技术中，密码变换 f 和 g 是对称密码算法，这意味着 $f=g$，并且 Ke=Kv。也就是说，对 Data 生成和验证 MDC 采用的是相同的密码操作。

由于基于对称密码技术的数据完整性与消息认证关系密切，因此对称密码技术生成的 MDC 常称为消息认证码(Message Authentication Code，MAC)。

消息认证是验证消息的完整性和真实性的过程，即当接收方收到发送方的报文时，接收方能够验证收到的报文是真实的和未被篡改的。消息认证的目的是防止传输和存储的消息被有意或无意地篡改，确保消息的完整性和真实性。消息认证包含以下几个方面的内容。

- 消息的完整性：验证消息在传输过程中是否保持原样，未被篡改、删除或插入。
- 消息的来源：验证消息的发送者身份，确保消息来自真正的发送者而非冒充者。
- 消息的序号和时间性：验证消息的序号和时间戳，防止消息的重放攻击。

MAC 的生成和验证可以使用密钥哈希函数技术，也可使用分组密码加密算法。

3.2.1 哈希函数

哈希(Hash)函数又称杂凑函数、散列函数，是一种由任意长消息创建小的数字指纹的方法。哈希函数示意图如图 3-5 所示。设 h 表示一个哈希函数，它使用变长消息 M 作为输入，生成定长结果 $h(M)$，即消息 M 的哈希值，M 为 h 的原像。

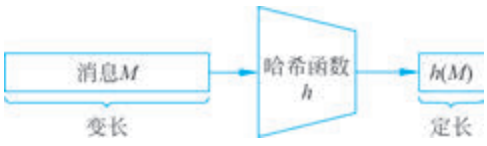

图 3-5 哈希函数示意图

这个定长的哈希值为消息 M 的消息摘要，也称为消息指纹。“指纹”这个名称来自于物理世界上没有两个人的指纹是一模一样的，即使同卵双胞胎的指纹也不一样。

1. 哈希函数特性

哈希函数虽然被称为函数，但实际上没有一个固定的公式，它更像是一种思想，只要符合哈希思想的都可以被称为哈希函数。哈希函数具有以下几个关键特性，这些特性使得它们成为验证消息完整性和真

实性的强大工具。

（1）唯一性。对于给定的哈希函数，不同的输入消息将产生不同的哈希值。尽管哈希函数的输出空间有限（例如，SHA-256 的输出是 256 位，即存在 2^{256} 种可能的哈希值），两个不同的输入理论上可能产生相同输出，但这种概率在实际应用中几乎可以忽略不计。

（2）确定性。相同的输入总是产生相同的输出。这意味着，只要哈希函数和输入消息保持不变，其哈希值就不会改变。

（3）不可逆性。从哈希值几乎不可能恢复出原始消息。这是哈希函数设计的一个重要安全特性，确保了即使哈希值被公开，原始消息的内容也能保持安全。这种转换是一种有损压缩映射，就像一台榨汁机，把香蕉、草莓和橙子等各种水果搅拌加工成一杯果汁，在加工过程中，有果皮、果肉、果核等的损失，故无法通过一杯果汁复原出原来的水果，如图 3-6 所示。

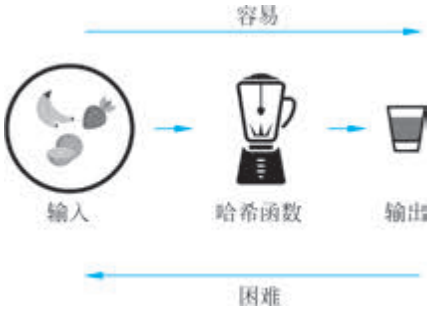

图 3-6　哈希函数的不可逆性

（4）雪崩效应。即使输入消息发生微小变化（例如，一个比特位的翻转），其哈希值也会发生显著变化。这种“雪崩效应”确保了哈希值能够准确地反映消息的任何变动。如图 3-7 所示，对两个仅有 1 个字符不同（2021 改为 2022）的.txt 文件计算 MD5 哈希值，输出的哈希值发生了显著变化。

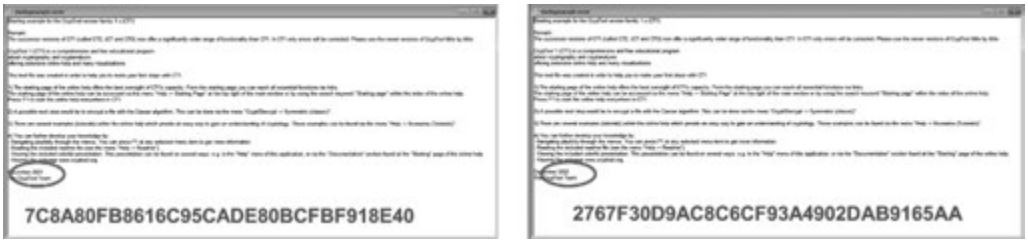

图 3-7　哈希函数的雪崩效应示例

2. 哈希函数安全性质

哈希函数将输入值映射到一个固定长度的输出值，这个输出值通常是一个整数或二进制串。由于哈希函数的输出值范围是有限的，因此存在不同的输入值，它们会被映射到相同的输出值上，即发生了哈希碰撞。哈希碰撞是哈希函数运算中的一个重要现象。如何保证其安全性质呢？

令$|h|$表示哈希函数 h 的固定输出长度。我们希望哈希函数具有以下安全性质。

1）混合变换

定义：对于任意的输入 x，输出的哈希值 $h(x)$和区间$[0, 2^{|h|}]$中的均匀二进制串在计算上是不可区分的（指在给定的计算资源和时间限制下，攻击者无法有效地进行区分）。

混合变换性质意味着，对于一个给定的哈希值，攻击者无法确定它是由哪个原始输入产生的（因为存在哈希碰撞现象）。混合变换性质使得哈希函数的输出与输入的具体内容在统计上无法直接关联，从而隐藏了输入数据的某些特性。这对于保护数据的隐私性和防止通过哈希值推断原始数据非常重要。

混合变换是哈希函数抗碰撞性和抗原像性的基础。由于混合变换的存在，攻击者难以找到两个不同的输入产生相同的哈希值（抗碰撞性），也难以根据哈希值反推出原始输入（抗原像性）。

2）实用有效性

定义：给定一个输入 x，$h(x)$的计算可在关于 x 的长度规模的低阶多项式（理想情况是线性的）时间内完成。

哈希函数的实用有效性是一个关键的性能指标，它直接关系到哈希函数在实际应用中的效率和可行性。输入长度规模是指输入 x 的大小或复杂度，比如字符串的长度、数组的元素数量、图的节点数或边数等。这个规模决定了算法需要处理的数据量。多项式时间复杂度意味着算法的运行时间可以表示为输入长度 n 的某个多项式函数，低阶多项式时间表示随着 n 的增长，算法的运行时间不会迅速增加。理想情况下，算法具有线性时间复杂度 $O(n)$。

在实际应用中，哈希函数需要能够快速计算哈希值，以便在数据检索、加密、验证数据完整性等方面提供高效的解决方案。

3）抗原像攻击

定义：已知一个哈希值 h，找一个输入 x，使得 $h=h(x)$，在计算上是不可行的。为使这个性质成立，要求 h 的输出空间 $2^{|h|}$ 应当足够大。

抗原像攻击性质也被称为抗第一原像性。抗原像攻击性质确保了哈希函数的单向性，即哈希过程是不可逆的。哈希函数的输出长度越长，找到与给定哈希值相匹配的原始输入的难度就越大，因此要求 h 的输出空间足够大。

如果哈希函数不具备抗原像攻击性质，那么攻击者就可能通过已知的哈希值来逆向推导出用户的原始输入，从而造成安全威胁。如图 3-8 所示，合法用户 Alice 尝试登录某个系统时，需要输入其用户名 Alice 和口令 4lice。为了确保口令的安全性，系统通常不会直接存储用户的原始口令，而是存储经过哈希函数处理后的哈希值 h(4lice)。这样，即使系统的口令数据库被泄露，攻击者也无法直接获取用户的原始口令。

然而，如果哈希函数不具备抗原像攻击的性质，那么攻击者就有可能利用已知的哈希值 h(4lice)来尝试逆向推导出用户的原始口令 4lice。一旦攻击者成功推算出 4lice 为 h(4lice)的原像，就可以直接获取用户的原始口令。随后，攻击者便能够以用户名 Alice 和口令 4lice 冒充 Alice 登录系统，进而执行未授权的操作或访问敏感信息，从而对系统安全造成严重威胁。

图 3-8　哈希函数原像攻击示例

4）抗第二原像攻击

定义：对给定的输入 x，找到另外一个不同输入 x'，使得 $h(x')=h(x)$，在计算上是不可行的。为使这个性质成立，同样要求 h 的输出空间 $2^{|h|}$ 应当足够大。

抗第二原像攻击性质保证了哈希函数的输出在给定输入下的唯一性，从而防止了伪造相同哈希值的攻击。哈希函数的输出长度越长，找到具有相同哈希值的不同输入的难度就越大，因此要求 h 的输出空间足够大。

抗第二原像攻击性质对于防止伪造和篡改数据至关重要。它确保攻击者无法伪造与原始消息具有相同哈希值的伪造消息，从而保证了消息的完整性和真实性。在密码存储、文件校验等场景中，抗第二原像攻击性质防止了攻击者通过找到具有相同哈希值的不同输入来绕过安全机制。如图 3-9 所示，合法用户 Alice 尝试登录某个系统时，需要输入其用户名 Alice 和口令 4lice。为了确保口令的安全性，系统不直

接存储口令本身，而是存储经过哈希函数处理后的哈希值 h(4lice)。如果哈希函数不具备抗第二原像攻击性质，那么攻击者就有可能利用已知的哈希值 h(4lice)来找到另一个与原口令 4lice 不同的输入，例如 4lic3，该输入经过哈希函数处理后仍然产生相同的哈希值 h(4lice)。随后，攻击者便能够以用户名 Alice 和这个新找到的口令 4lic3 冒充 Alice 登录系统，进而执行未授权的操作或访问敏感信息，从而对系统安全造成严重威胁。

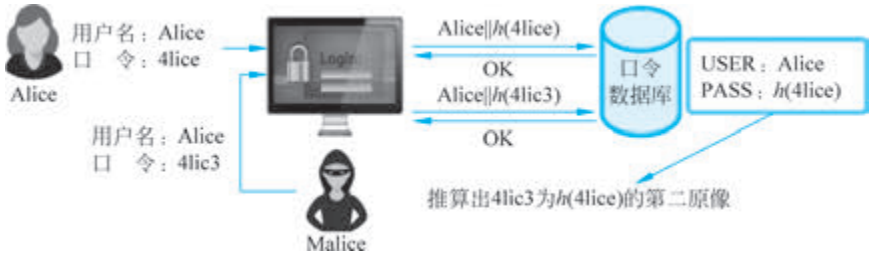

图 3-9　哈希函数第二原像攻击示例

5）抗碰撞攻击

定义：找两个输入 x 和 y，且 $x \neq y$，使得 $h(x)=h(y)$，在计算上是不可行的。为使这个性质成立，同样要求 h 的输出空间 $2^{|h|}$ 应当足够大。

抗碰撞攻击性质确保了哈希函数输出的独特性，降低了数据被篡改或伪造的风险。如果满足 $x \neq y$ 且 $h(x)=h(y)$，则称出现碰撞。碰撞是一定存在的。哈希函数的输出长度越长，找到碰撞的难度就越大，因此要求 h 的输出空间足够大。

抗碰撞攻击性质确保了即使数据在传输或存储过程中被篡改，通过比较原始数据的哈希值与当前数据的哈希值，也能及时发现这种篡改。在数字签名中，哈希函数的抗碰撞攻击性质确保了签名的唯一性和不可伪造性，从而保护了交易双方的权益。如图 3-10 所示，农民 Alice 向地主 Bob 借款。在这个场景中，Bob 试图利用哈希碰撞来欺诈 Alice，他如果事先能找到两个完全不同的借据消息，具有相同的哈希值，其中一个借据消息中要求 Alice 偿还的金额较小（比如 10 美元），而另一个消息中要求的偿还金额较大（比如 1 000 000 美元）。Bob 设法让 Alice 在小额借据上签名认可。Alice 签下小额借据，生成了相应的哈希值并记录在“账本”上之后，Bob 声称大额借据是真实的。

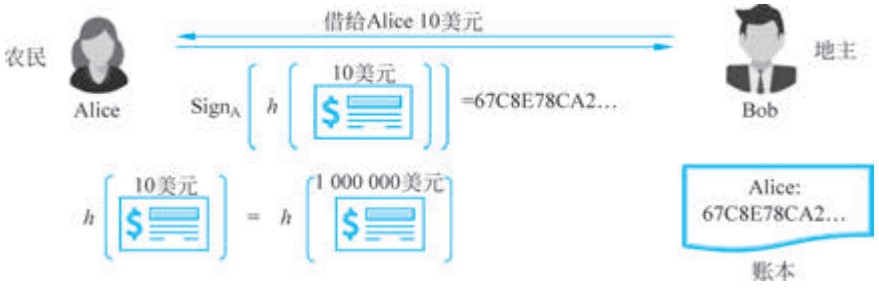

图 3-10　哈希函数碰撞攻击示例

哈希函数的混合变换和抗碰撞攻击性质可以使用分组密码算法设计中所用的类似操作来实现。抗原像攻击性质可以通过一些数据压缩技术实现，数据压缩技术损失一部分输入数据，因而使得该函数不可逆。

图 3-11 给出了哈希函数 3 条性质，即抗原像攻击、抗第二原像攻击、抗碰撞攻击之间的关系。

一个哈希函数如果是抗碰撞攻击的，那么它同时也是抗第二原像攻击的；反之不一定成立。一个哈

希函数可以是抗碰撞攻击的，但不一定是抗原像攻击的；反之也成立。一个哈希函数可以是抗第二原像攻击的，但不一定是抗原像攻击的；反之也成立。

3. 哈希函数应用

哈希函数的应用非常广泛，几乎渗透到了信息网络安全的方方面面。以下是几个典型的应用场景。

1）消息认证

哈希函数在消息认证中发挥着重要作用，通过生成和验证消息摘要来确保消息的完整性和真实性。如图 3-12 为哈希函数在消息认证中的应用。

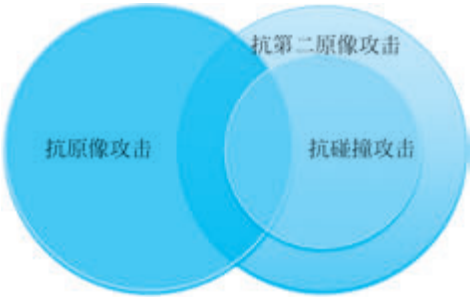

图 3-11　哈希函数的几条性质之间的关系

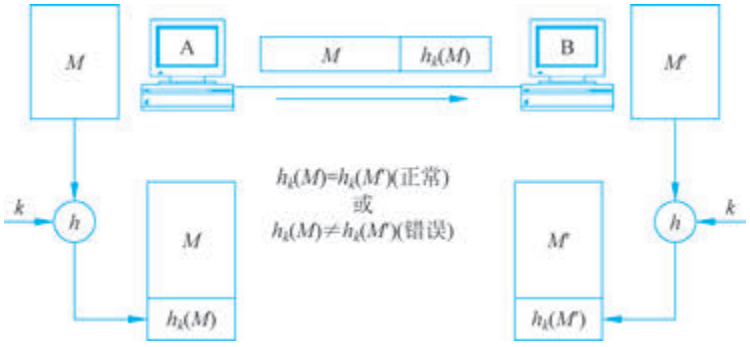

图 3-12　哈希函数在消息认证中的应用

发送者使用哈希函数 h 将密钥 k 和消息 M 作为输入进行哈希计算，生成一个固定长度的输出作为认证码 $h_k(M)$。接收者在验证消息时，同样使用哈希函数 h 对接收到的消息 M' 和密钥 k 进行哈希计算，并将结果 $h_k(M')$ 与接收到的认证码进行对比。如果两者相同，则表明消息在传输过程中未被篡改且确实来自合法的发送者。其中，发送者和接收者的共享密钥 k 是消息认证必需的，以防攻击者伪造有效的哈希值。

2）数字签名

在数字签名中，哈希函数一般用来产生消息摘要或消息指纹，如图 3-13 所示。

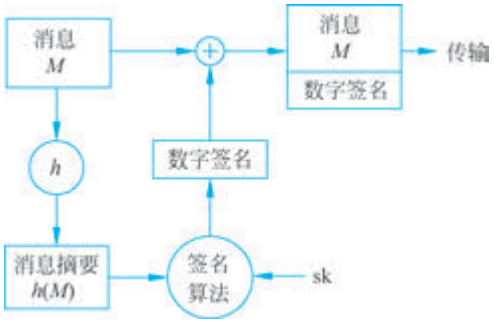

图 3-13　哈希函数在数字签名中的应用

这种用法有两个目的：第一个目的是数字签名是基于公钥密码算法实现的。公钥密码算法的一大特点是适合处理短消息，相对于较长的消息则显得有些吃力。虽然可以将长的消息分成若干小段，然后再分别签名。不过，这样做非常麻烦，而且会使数据在存储和传输中受到威胁的范围更大。合理的做法是利用哈希函数可将任意长度数据映射为固定长度数据的性质，在数字签名前对任意长度的消息 M 进行哈希运算得到固定长度的消息摘要 $h(M)$。然后，发送方用自己的私钥 sk 对消息摘要进行加密，创建发送方的个人签名，和消息一起打包送给接收方。收到消息和签名后，接收方用发送方的公钥解密该签名，以恢复消息摘要，并使用发送方所用的同一哈希函数对该消息进行哈希运算。如果接收方计算的消息摘要与从发送方发来的消息摘要完全匹配，则接收方确认消息是发送方发送的，而且消息是完整的。第二个目的是为将要签名的消息增加一个可验证的冗余，或者说一种可辨识的信息，从而实现数字签名的不可伪造性。关于数字签名的伪造攻击见 3.3.1 节。


视频 15

3）其他应用

哈希函数常用于生成单向口令文件。操作系统在存储用户口令时，直接使用明文存储是极其不安全的。如果数据库被泄露，那么所有用户的口令都将暴露无遗。为了增强安全性，通常使用哈希函数来存储口令的加密形式，而不是原始口令本身。当用户输入口令时，操作系统将对比输入口令的哈希值和存储在口令文件中的哈希值。

哈希函数被广泛地用作实用的伪随机函数。这些应用包括密钥协商（如两个主体将他们自己的随机种子作为哈希函数的输入，得到一个共享的密钥值）、认证协议（如协议双方通过交换某些哈希值来证实协议执行的完整性）、电子商务（如以赌博方式实现小额支付的聚集）、区块链（挖矿过程中使用哈希函数对包含随机数的区块进行哈希处理，生成一个具有必要属性的哈希值）等。密码技术中所使用的随机数需要具备“不可能根据过去的随机数列预测未来的随机数列”这样的性质。哈希函数的雪崩效应就具有这样的性质。

哈希函数还可以用于构建入侵检测和病毒检测系统。通过将每个文件的哈希值存储到系统中并保证其安全，就能通过重新计算哈希值来判断文件是否已经被修改。这种方法可以用于检测病毒和其他恶意软件，保护系统的安全。

4. 对哈希函数的攻击

同加密算法一样，针对哈希函数的攻击也分为两大类：穷举攻击和密码分析。穷举攻击不依赖于算法细节，仅与哈希值的长度有关。密码分析利用哈希函数结构和算法的缺陷进行攻击。

1）穷举攻击

穷举攻击主要包括原像攻击、第二原像攻击和碰撞攻击。

对于原像攻击和第二原像攻击，攻击者的目标是希望找到一个值 y，使得 $h(y)$ 等于给定的哈希值 h。穷举攻击的方法是随机选择各个 y 值，尝试计算其哈希值直至出现碰撞。对于 n 位的哈希值，原像攻击和第二原像攻击的穷举次数为 2^n。举个例子，$n=64$，如果服务器集群每秒验证一亿次，则需要约 5849 年。

对于碰撞攻击，攻击者的目标是希望找到两条消息 x、y，使得 $h(x)=h(y)$。与原像攻击和第二原像攻击相比，碰撞攻击的穷举次数要少很多。可以通过生日悖论证明，对于 n 位的哈希值，碰撞攻击的穷举次数减少为 $2^{n/2}$。这种碰撞攻击方法因此也称为生日攻击。

生日悖论基于概率论中的如下命题：对于任意的函数 $f: X \to Y$，其中，Y 包含 n 个元素。在已知概率 ε 的条件下，为了发生碰撞仅需计算如下 k 个函数值

$$k=\sqrt{2n\ln\frac{1}{1-\varepsilon}}$$

其中，ln 是以自然对数为底的对数函数。

根据以上命题，使 k 个人中至少有两个人生日相同的概率大于 1/2 的最小 k 值为

$$k=\sqrt{2\times 365\times\ln\frac{1}{1-0.5}}\approx 23$$

这就是生日悖论。换个形象的描述，以大于 50%的概率从一个房间中找到有两个人的生日相同，在该房间中只需有 23 人即可。这一结果初看上去似乎不大直观，与人的直觉相抵触。因为大多数人会认为，23 人中有 2 人生日相同的概率应该远远小于 50%。其实对于特定的人，房间里有其他人和他生日相同的概率很小，但我们考虑的是任意两个人生日相同的概率，任意两个人生日相同的概率随着人数变化的曲线如图 3-14 所示。23 人中有 2 人生日相同有 $C_2^{23}=253$ 种组合，因此生日相同的概率较大。（从引起逻

辑矛盾的角度来说，生日悖论并不是一种逻辑悖论，逻辑悖论是表面上同一命题或推理中隐含着两个对立的结论，而这两个结论都能自圆其说。生日悖论是说这个数学事实与一般直觉相抵触。）

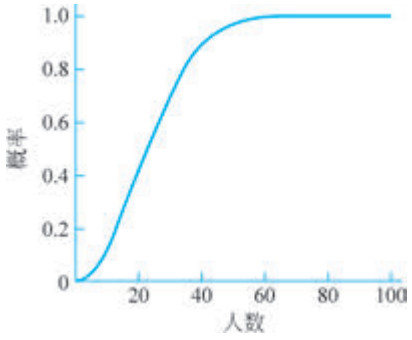

图 3-14 任意两个人生日相同的概率随着人数变化的曲线图

根据生日悖论，对一个输出空间大小为 n 的哈希函数，只需计算大约 $\sqrt{n}$ 个函数值，就能以一个不可忽略的概率 ε 发现一个碰撞，因为 $k=\sqrt{2n\ln(1/1-\varepsilon)}\sim\sqrt{n}$。即使 ε 非常接近于 1，ln 的值仍然比较小，因此通常 k 与 $\sqrt{n}$ 成正比。例如，$\varepsilon=1/2$，$k=1.774\sqrt{n}$。

生日攻击是对哈希函数最经典的穷举攻击法，可用于攻击任何类型的哈希函数。由生日攻击得到以下启示：

- 哈希函数的 $2^{n/2}$ 个哈希值足以使攻击者以不可忽略的概率得到一个碰撞；
- 由于哈希函数不是真正的随机函数，因此实际上所需计算的函数值可能会更少；
- 哈希函数的输出空间大小的下界为 $2^{n/2}$。

在国际上，MD5 和 SHA-1 两大哈希函数算法曾广泛应用于网络安全、数据校验等多个关键领域，它们分别生成 128 位和 160 位的哈希值，并具有抗生日攻击强度为 2^{64} 和 2^{80} 的安全性。2004 年，我国王小云教授领导的研究团队在国际密码学会议上公布了 MD5 的重大漏洞，他们以远低于传统生日攻击理论所需的计算量，以不可忽视的概率实现了 MD5 的碰撞。此发现极大地削弱了 MD5 算法的安全性，导致它逐渐在实际应用中被淘汰。2005 年，王小云教授团队再接再厉，成功地对 SHA-1 进行了破解，使 SHA-1 的安全性也遭受了严峻考验。美国政府随即宣布，在 2010 年前将停止在关键应用中使用 SHA-1。随后，美国国家标准与技术研究院（NIST）启动了为期五年的 SHA-3 国际哈希函数新标准设计项目，目标是开发出更为强大和安全的哈希函数算法，以替代越发脆弱的 SHA-1。

为了实施生日攻击，攻击者应当生成消息-哈希值对：

$$(m_1, h(m_1)), (m_2, h(m_2)), \cdots$$

直到找到两个消息 m 和 m'，满足

$$m\neq m', h(m)=h(m')$$

这样的一对消息称为哈希函数 h 的碰撞。当然，为了使生日攻击对攻击者有用，碰撞消息 m 和 m' 应当包含一些有意义的子消息。例如，一个要进行哈希运算的消息是下面形式的支付授权语句：

$$M=\text{Price}, \text{Goods\_Description}, R$$

其中，R 是随机数，使得该协议消息随机化。

那么，一个有效的生日攻击可以是

$$m=\text{Price\_1}, \text{Goods\_Description}, r$$
$$m'=\text{Price\_2}, \text{Goods\_Description}, r'$$

其中，Price_1≠Price_2 和 Goods_Description 是固定的消息部分，碰撞是指关于随机数 $r\neq r'$ 的碰撞，使得 $h(m)=h(m')$。上述场景的示意图如图 3-15 所示。Malice 是电脑卖家，Bob 是电脑买家。Malice 要找到两份合同，哈希值相同，合同内除价格不同外其他信息相同。应该怎样做呢？在合同文本中，找到若干个要攻击的节点，每个节点通过插入，改变一些符号，找到可能的取值，使得哈希值相同。Malice 将 5000

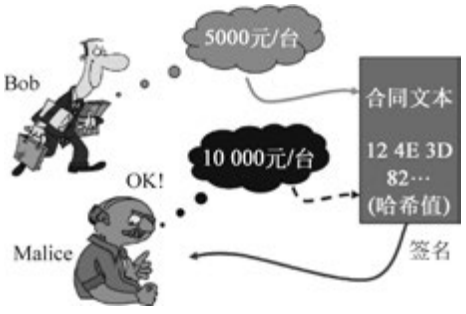

图 3-15 哈希函数生日攻击场景示意图

元/台的合同作为合同样本由 Bob 签名，自己将 10 000 元/台的合同藏起来，以便在将来进行欺诈，声称 10 000 元/台的合同才是真的。


视频 16

2）密码分析

与针对加密算法的攻击类似，针对哈希函数的密码分析攻击也利用算法的某种性质，而不采用穷举搜索方法。评价哈希算法抗密码分析能力的方法是，将其强度与穷举攻击所需的次数进行比较。也就是说，理想的哈希函数算法要求密码分析攻击的尝试次数大于或等于穷举攻击所需的尝试次数。

典型哈希函数的总体结构如图 3-16 所示。这种迭代型结构称为 Merkle-Damgaard(MD)变换，大多数哈希函数都采用这种结构。哈希函数将输入的消息分为 L 个定长的分组，每个分组的长度均为 b 位。若最后一个分组不满足 b 位，则将其填充为 b 位。

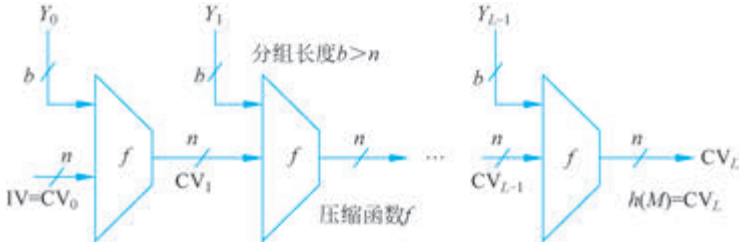

图 3-16　典型哈希函数的总体结构

哈希算法重复使用压缩函数 f，这个函数有两个输入：一个是前一阶段的 n 位输出，另一个来源于消息的 b 位分组，函数产生一个 n 位输出。算法开始时需要有一个 n 位的初始变量 IV，最终的输出结果是一个 n 位哈希值。通常，$b>n$，因此称为压缩。哈希函数迭代过程可以归纳如下：

$$CV_0 = IV$$

$$CV_i = f(CV_{i-1}, Y_{i-1}), \quad 1 \leqslant i \leqslant L$$

$$h(M) = CV_L$$

式中，哈希函数的输入是消息 M，它由分组 Y_0，Y_1，…，Y_{L-1} 组成。

哈希函数的核心就是设计具有抗碰撞能力的压缩函数 f。由于是压缩函数，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不可行的。类似于对称分组密码算法，f 也常采用迭代轮结构，因此相应的攻击方法有中间相遇攻击和差分分析等。

5. 专用哈希函数

哈希函数本质上是一种设计理念，基于这一理念而专门设计的具体算法被称为专用哈希函数，它们是根据特定需求量身定制的。

到目前为止，最流行的专用哈希函数就是 MD4(Message-Digest algorithm 4)家族。MD4 是 Ronald Rivest(RSA 算法设计者之一)在 20 世纪 90 年代初设计的一种消息摘要算法。MD4 的设计初衷是为了在软件实现中取得高效率，它采用了 32 位寄存器，并大量使用位操作(如逻辑与、或、异或、否)来完成数据的处理。后来，MD4 被发现存在安全漏洞，特别是容易受到碰撞攻击，因此逐渐被弃用。

MD5(Message-Digest Algorithm 5)是 MD4 的改进版本，由 Ronald Rivest 在 1991 年与麻省理工学院(MIT)的同事一起开发，并于 1992 年作为 RFC 1321 标准发布。MD5 自发布以来，迅速成为互联网安全领域广泛使用的哈希算法之一，被应用于文件完整性校验、数字签名、安全认证等多个方面。然而，随着计算能力的提升和密码学研究的深入，MD5 算法的安全性也逐渐受到挑战。2004 年，中国密码学家王小云在国际密码学会议上公布了 MD5 的碰撞攻击方法，证明了 MD5 算法无法抵抗碰撞攻击。这一发现引起了广泛关注，并促使密码学领域开始寻找更为安全的哈希算法。

SHA-1(Secure Hash Algorithm 1)最初是由美国国家安全局(NSA)设计,并于 1993 年由美国国家标准与技术研究院(NIST)发布,其最初版本通常称为 SHA-0。SHA-0 发布不久就被发现存在潜在的安全问题,很快被撤回,并由 1995 年发布的 FIPS PUB 180-1 中的修订版本 SHA-1 所取代。SHA-1 的设计原理与 MD5 相似,都使用了迭代结构和位操作来生成哈希值。2005 年,王小云团队发表了对 SHA-1 碰撞运算量的研究结果,揭示了 SHA-1 存在碰撞攻击的可能性。随后,王小云教授与姚期智夫妇合作,进一步提出了将 SHA-1 破解时间缩短的方法。这些发现促使 NIST 在 2005 年宣布逐步废止 SHA-1,并加速了向 SHA-2 版本的过渡。

SHA-2 是 NIST 于 2002 年发布的一系列新哈希算法,包括 SHA-256、SHA-384、SHA-512 以及稍后加入的 SHA-224,这些统称为 SHA-2 家族。SHA-2 在算法结构上与 SHA-1 有着相似之处,都采用了迭代结构,并依赖于模算术运算与二元逻辑操作来生成哈希值。然而,SHA-2 在多个关键方面进行了显著的改进,包括内部状态的大小、消息预处理机制以及消息扩展过程,这些改进旨在增强算法的安全性和性能。尽管截止到目前,SHA-2 尚未暴露出明显的安全弱点,也未有公开报道的有效攻击案例,但 SHA-1 的遭遇为整个哈希函数家族敲响了警钟。2017 年,荷兰密码学研究机构 CWI 与谷歌合作,利用 100 个 GPU 进行为期一年的计算,耗资数十万美元,成功构造了两个内容不同但 SHA-1 哈希值完全相同的 PDF 文件,这标志着 SHA-1 的碰撞攻击已成为现实。SHA-1 碰撞攻击的成功实施让人们开始审视是否应继续将大量资源投入到 SHA 家族的进一步研究和推广上,还是应该转而探索更为先进和安全的哈希算法,如 SHA-3 等。

SHA-3 的推出是哈希函数领域的一个重要里程碑,其开发过程严格遵循高级加密标准(AES)的征集模式,确保了算法的广泛参与和严格筛选。2007 年,NIST 启动了全球范围内的征集活动,旨在寻找一种全新的、不依赖于 MD 结构的哈希算法,以接替并超越 SHA-2,这一新算法最终被命名为 SHA-3。经过激烈的竞争和多轮评估,NIST 在 2009 年和 2010 年分别举办了两次会议,从最初的 51 个候选算法中逐步筛选出 14 个进入第二轮,并最终确定了 5 个算法进入最终角逐。2012 年 10 月,Keccak 算法凭借其独特的海绵函数设计脱颖而出,成为 SHA-3 的获胜者。这一算法由包括 Joan Daemen(Rijndael 算法设计者之一)在内的 5 位专家团队设计,其创新之处在于其核心函数与 MD5、SHA-1 及 SHA-2 在结构上存在显著差异,从而有效抵御了传统攻击手段。为了验证并提升 Keccak 算法的安全性,Keccak 设计团队在互联网上发起了 Keccak 原像挑战和碰撞挑战竞赛,鼓励全球研究人员对算法进行深入的安全性分析。这一举措不仅增强了 Keccak 算法的公信力,也促进了哈希函数安全性研究的进一步发展。

自 2015 年 SHA-3 被正式公布为新的哈希标准以来,其应用逐渐展开,并有望在未来逐步取代 SHA-2。然而,值得注意的是,SHA-3 的普及进程并非一帆风顺,许多主流平台如 macOS X 和 iOS 等仍依赖于仅支持 SHA-1 和 SHA-2 的 libcrypto 库。因此,将 SHA-3 有效部署到这些平台并实现实用化,成为了当前研究的一个热点和难点。随着技术的不断进步和应用的深入,SHA-3 有望在未来成为哈希函数领域的主流标准,为信息网络安全提供更加坚实的保障。

表 3-1 总结了专用哈希函数概况。

表 3-1 专用哈希函数概况

名称	哈希值长度/位	分组长度/位	碰撞情况	颁布时间
MD4	128	512	是	1990 年
MD5	128	512	是	1992 年
SHA-1	160	512	有缺陷	1995 年
SHA-2	256/384/512	512/1024/1024	否	2002 年
	224	512	否	2004 年
SHA-3	224/256/384/512	1152/1088/832/576	否	2015 年

以下介绍最流行的 MD5 算法和新的哈希标准 SHA-3。

1）MD5 算法

尽管 MD5 哈希函数在安全性上已显示出明显的弱点，特别是其易受到碰撞攻击，但由于其广泛的兼容性、在硬软件实现上的高效率，以及免费使用的特点，仍然被广泛应用于一些非安全敏感的场景中，比如快速校验文件在传输过程中是否保持完整且未被篡改。

MD5 算法以 512 位分组为单位处理输入消息，每个分组又划分为 16 个 32 位的子块。算法的输出由 4 个 32 位的块组成，这些块级联成一个 128 位的摘要值。MD5 算法的主要操作包括加法、移位和逻辑运算（如 AND、OR、XOR 和 NOT），且不需要复杂的数据结构（如替换表）来支持其操作，这使得它在各种硬件和软件平台上都能快速执行。如图 3-17 所示，MD5 算法包括以下几个步骤。

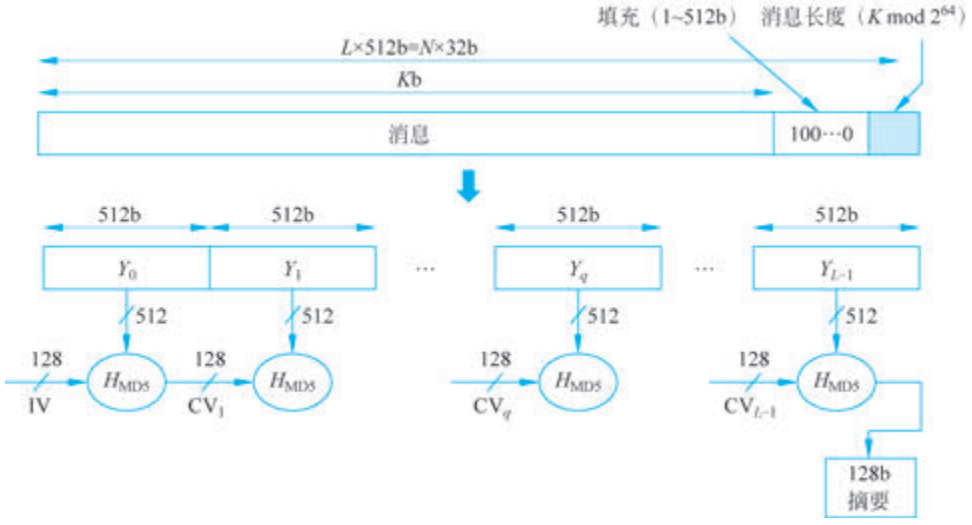

图 3-17　MD5 算法总体结构

（1）填充消息。首先在消息的末尾处附上消息长度为 64b 的二进制表示（最低有效位在前），大小为 $K \bmod 2^{64}$，K 表示消息长度。然后在消息后面填充一个 1 和多个 0，填充位数为 1～512，即使消息长度已满足要求，仍需填充。例如，如果消息长度为 447b，则填充 1b；如果消息长度为 448b，则需填充 512b，使其长度变为 960b（960 = 512 × 2 − 64）；如果消息长度为 512b，也需填充 448b，使其长度变为 960b（960=512×2−64）。

（2）消息分组。填充后的消息恰好是 512b 整数倍（L）长。用 Y_0，Y_1，…，Y_{L-1} 表示不同的 512b 长的消息分组。

（3）初始化缓冲区。算法使用 128b 的缓冲区存储中间结果和最终哈希值。每个缓冲区由 4 个 32b 的寄存器 a、b、c、d 组成，这 4 个寄存器初始化为：

a = 01　23　45　76

b = 89　AB　CD　EF

c = FE　DC　BA　98

d = 76　54　32　10

（4）处理消息分组。算法以 512b 的分组为单位处理，核心是压缩函数 H_{MD5}。压缩函数 H_{MD5} 的结构图 3-18 所示。首先需要有一个 128b 的初始变量 IV，从 Y_0 开始处理，上一循环的输出作为下一循环的输入，直到处理完 Y_{L-1} 为止。压缩函数 H_{MD5} 对当前 512b 的消息分组 Y_q 与 128b 缓冲值 CV_q 进行

处理，处理结果存放在 a、b、c、d 中，再更新 128b 缓冲值 CV_{q+1}。这里 512＞128，故 H_{MD5} 为压缩函数。消息分组 Y_q 的处理包含 4 轮操作，每一轮由 16 次迭代操作组成，上一轮的输出作为下一轮的输入。

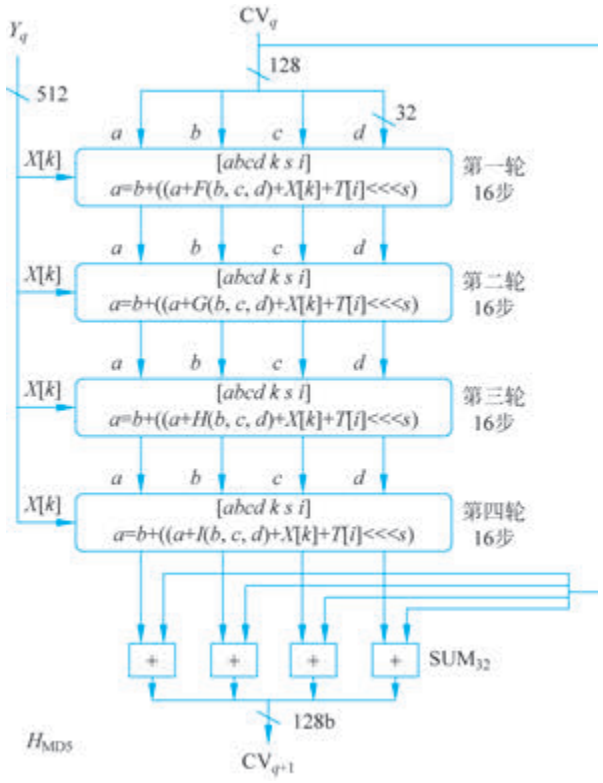

图 3-18　压缩函数 H_{MD5} 的结构图

4 轮处理具有相似的结构：

$$a, b, c, d \leftarrow d, b+((a+G(b,c,d)+X[k]+T[i])<<<s), b, c$$

但每轮处理使用不同的非线性函数 G。函数 G 为以下 4 个非线性函数之一：

$$F(b,c,d)=(b \wedge c) \vee (\sim b \wedge d)$$

$$G(b,c,d)=(b \wedge d) \vee (c \wedge \sim d)$$

$$H(b,c,d)=b \oplus c \oplus d$$

$$I(b,c,d)=c \oplus (b \vee \sim d)$$

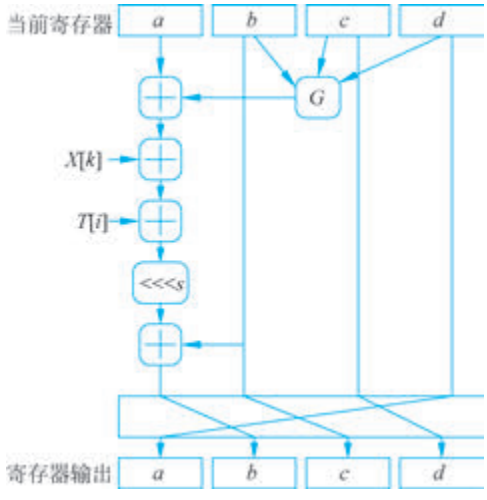

图 3-19　H_{MD5} 轮处理的运算

各种运算符号的含义为："∧"表示逻辑与，"∨"表示逻辑或，"⊕"表示逻辑异或，"～"表示逻辑补，"＋"表示整数模 2^{32} 加法运算，"<<<"表示循环左移 s 位。轮处理的运算如图 3-19 所示。

常数表 $T[i](1 \leqslant i \leqslant 64)$ 由 sin 函数构造，共有 64 个元素，每个元素是 32b 字，$T[i]=2^{32}|\sin(i)|$，其中 i 是弧度，$|x|$ 表示取 x 的绝对值。处理每一个消息分组 Y_q 时，每一轮使用常数表 $T[i]$ 中的 16 个，正好用 4 轮。

$X[k](0 \leqslant k \leqslant 15)$ 保存了当前待处理的 512b 分组 Y_q 的值，分为 16 个 32b 字。每一轮使用 $X[k]$ 中的第 j 字运

算，$X[k]$在每一轮中恰好被使用 1 次。

第一轮使用 $X[j]$，$j=k$；

第二轮使用 $X[j]$，$j=(1+5k) \bmod 16$；

第三轮使用 $X[j]$，$j=(5+3j) \bmod 16$；

第四轮使用 $X[j]$，$j=7j \bmod 16$。

每一轮每一步按表 3-2 所示的规则循环左移 s 位。

表 3-2　H_{MD5} 轮迭代循环左移运算

s	第一轮	第二轮	第三轮	第四轮
第 1,5,9,13 步	7	5	4	6
第 2,6,10,14 步	12	9	11	10
第 3,7,11,15 步	17	14	16	15
第 4,8,12,16 步	22	20	23	21

(5) 输出摘要值。消息的 L 个分组都被处理完后，最后输出的就是运算结束时缓冲区中的 128b 的消息摘要。

2) SHA-3 算法

SHA-3 算法由美国国家标准与技术研究院(NIST)于 2015 年正式发布。作为 SHA-2 算法的后续版本，SHA-3 旨在提供更高的安全性和性能。算法自发布以来，已经在多个领域得到了广泛应用，包括数字签名、数据完整性验证、密码学协议、区块链技术等。特别是在区块链领域，SHA-3 因其安全性和灵活性而受到青睐。

SHA-3 算法提供了多种输出长度，包括 224 位、256 位、384 位和 512 位，可以根据不同的应用场景选择合适的输出长度。SHA-3 算法使用了一种称为“海绵函数”的结构，它允许数据被“吸收”到海绵中，并通过一系列的变换和处理，最终被“挤压”出来生成哈希值。图 3-20 所示为海绵函数的输入和输出，算法包括以下几个步骤。

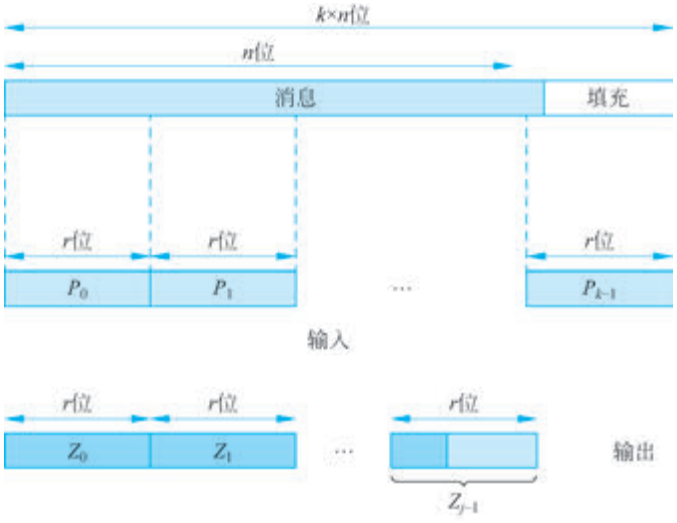

图 3-20　海绵函数的输入和输出

(1) 填充消息。对长度为 n 位的输入消息进行填充，使其为 r 位分组的整数倍长 k。若 $n \bmod r=0$，则添加一个 r 位填充分组。填充方案有两个：

- 简单填充——将第一位填充 1，其他位填充 0；

- 多速率填充——将第一位和最后一位填充 1，其他位填充 0。

（2）消息分组。将填充后的消息分成固定大小的分组，每个分组的大小为 r 位，用 P_0，P_1，…，P_{k-1} 表示。

（3）初始化内部状态。SHA-3 使用一个长度为 $b=r+c$ 位的状态变量 s，在开始哈希计算之前将其初始化为 0，其取值在每轮迭代中更新。状态变量的两个参数定义如下：

- 位速率(r)——输入消息的分组长度。它反映每轮迭代中处理的位数。它决定了算法的效率，因为更大的位速率意味着可以同时处理更多的数据。
- 容量(c)——代表状态变量中不受输入/输出影响的部分，即秘密值。关于容量的含义超出本书讨论的范围。本质上，容量越大，海绵结构的安全性就越高，因为它为攻击者提供了更多的数据需要猜测或破解。

（4）迭代过程。海绵结构由两个阶段组成：吸收阶段和挤压阶段。图 3-21 给出了海绵函数的迭代结构。

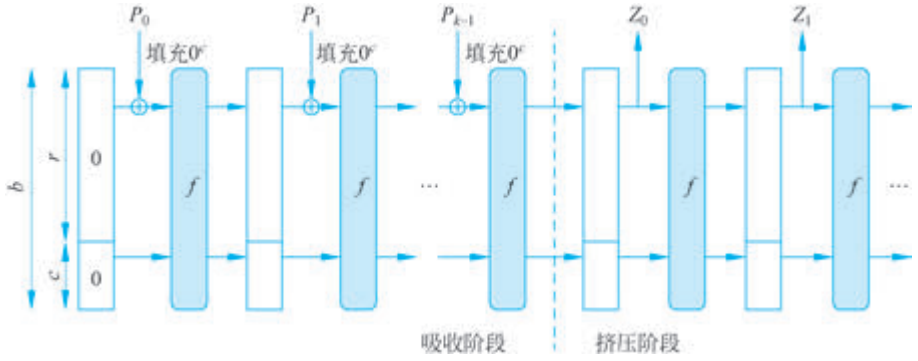

图 3-21　海绵函数的迭代结构

在吸收阶段，对于每轮迭代，通过填充若干 0 将输入分组的长度从 r 位扩展为 b 位；然后对扩展后的分组和 s 进行 XOR 异或运算得到 b 位的结果，将其作为迭代函数 f 的输入；f 函数的输出作为下一轮迭代中 s 的取值。

如果期望的输出长度 l 满足 $l \leqslant b$；那么在吸收阶段完成后，返回 s 的前 r 位，且海绵结构的运行结束；否则，海绵结构进入挤压阶段。首先，保留 s 的前 r 位作为输出分组 Z_0，然后在每轮迭代中通过重复执行 f 函数来更新 s 的值，s 的前 r 位依次保留为输出分组 Z_i，并与前面已生成的分组连接。该处理过程持续 $j-1$ 次，直到满足 $(j-1)\times r<l \leqslant j\times r$。这时，返回连接分组 Z 的前 l 位。

总之，海绵结构是一种简单的迭代结构，基于定长变换或对固定长度 b 位进行操作的置换 f，构造出来的整体函数 F 能够处理可变长度的输入，得到任意长度的输出。基于海绵结构的 SHA-3 的参数取值如表 3-3 所示。

表 3-3　SHA-3 参数

消息摘要长度	224	256	384	512
消息长度	没有限制	没有限制	没有限制	没有限制
分组长度(位速率 r)	1152	1088	832	576
容量 c	448	512	768	1024
抗碰撞攻击	2^{112}	2^{128}	2^{192}	2^{256}
抗第二原像攻击	2^{224}	2^{256}	2^{384}	2^{512}

SHA-3 的迭代函数 f 用于处理每个连续的输入消息分组。f 函数的细节内容超出本书讨论的范围，可参考文献[1]。

(5) 输出摘要值。在所有的消息分组都被处理完后，海绵函数生成一组输出 Z_0，Z_1，…，Z_{j-1}。输出数据分组的数量由所需的输出位数决定。当需要 l 位输出时，生成的 j 个输出分组就是摘要值，其中 $(j-1)\times r<l\leqslant j\times r$。

3.2.2 基于哈希函数的 MAC

用哈希函数如何实现数据完整性保护？哈希函数是直接设计的，对消息进行哈希运算时不需要密钥控制。但数据完整性保护必须要有密钥的参与才能正确实现，否则任何人都可以构造有效的 MDC。因此，在共享密钥的情况下，哈希函数将密钥作为它的一部分输入，另一部分输入为需要认证的消息。

如图 3-12 所示，为了认证一个消息 M，发送者计算 $\mathrm{MAC}=h_k(M)$，其中，k 为发送者和接收者的共享密钥。与发送者共享密钥 k 的接收者由所接收的消息重新计算出 MAC，并检验同所接收的 MAC 是否一致。如果一致，则可以相信该消息未被篡改并来自所声称的发送者。

因为这样的 MAC 是使用哈希函数构造的，因此也称为 HMAC。将密钥加到现有哈希函数中的方法包括：

$$\mathrm{HMAC}=h(k\parallel M)$$
$$\mathrm{HMAC}=h(M\parallel k)$$
$$\mathrm{HMAC}=h(k_1\parallel M\parallel k_2)$$
$$\mathrm{HMAC}=h(k_2\parallel h(k_1\parallel M))$$

其中，“‖”表示比特串的链接，k_1 和 k_2 表示用不同的密钥保护消息的两端，这样做可以防止攻击者利用某些哈希函数的轮函数迭代结构，选择一些数据用作消息前缀或后缀来修改消息。

HMAC 算法是最受支持的一种方案，也是 IP 安全中必须实现的方案，并且其他因特网协议(如 SSL)也使用了 HMAC。HMAC 已经成为互联网标准(RFC 2104)，同时也是 NIST 标准(FIPS 198)。

HMAC 算法对密钥的使用和处理较简单，可以与任何迭代型哈希函数捆绑使用(将哈希函数当作一个黑盒使用)。这样一来，任何哈希函数都可作为实现 HMAC 的一个模块，HMAC 代码中的哈希函数可事先准备好，无须修改就可使用；如果 HMAC 要求使用更快或更安全的哈希函数，则只需用新模块代替旧模块，例如，用实现 SHA 的模块代替 MD5 的模块。若使用的哈希函数具有合理的密码分析强度，则可证明 HMAC 是安全的；即便是被攻破的哈希算法，由于攻击者在线攻击困难，也能产生安全的 HMAC。

图 3-22 给出了 HMAC 算法的总体结构。定义参数和符号如下：

- b 是每一分组的位数；
- n 是嵌入哈希函数所产生的哈希值长度；
- K 为密钥，建议密钥长度≥n。密钥长度大于 b，则输入哈希函数产生一个 n 位密钥；
- M 为输入消息，填充后分为 L 个分组 Y_0，Y_1，…，

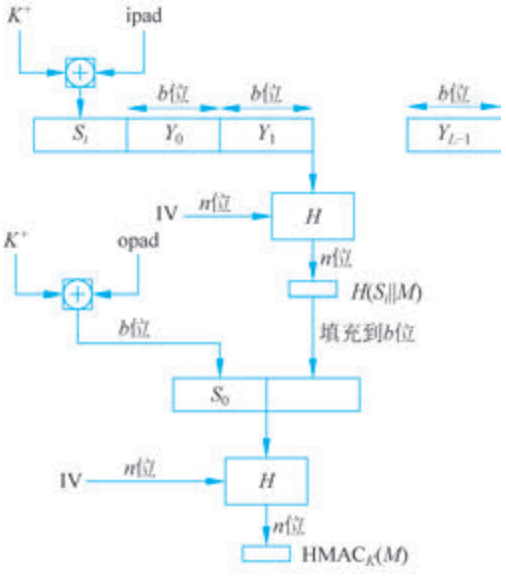

图 3-22 HMAC 的结构

Y_{L-1}；

- H 为嵌入的哈希函数(如 MD5、SHA-1)；
- IV 为输入哈希函数的初始值；
- ipad 为 00110110(十六进制数 36)重复 $b/8$ 次的结果；
- opad 为 01011100(十六进制数 5C)重复 $b/8$ 次的结果。

HMAC 可用下式描述：

$$\mathrm{HMAC}_K(M)=h\left[(K^+\oplus \mathrm{opad})\parallel h\left[(K^+\oplus \mathrm{ipad})\parallel M\right]\right]$$

算法描述如下：

(1) 在 K 的左边填充 0 产生一个 b 位长的 K^+；

(2) K^+ 与 ipad 逐位异或产生 b 位分组 S_i；

(3) S_i 链接 M 后输入哈希函数 H；

(4) K^+ 与 opad 逐位异或产生 b 位分组 S_0；

(5) 将步骤(3)得到的哈希值填充至 b 位，链接在 S_0 后输入哈希函数 H；

(6) 输出 n 位最终结果 $\mathrm{HMAC}_K(M)$。

3.2.3 基于分组密码的 MAC

使用分组密码构造 MAC 是常用的方法。随着 AES 和密码算法代码的普及，这种方法变得更加广泛。

令 $E_k(M)$表示输入消息为 M，密钥为 k 的分组加密算法。为了认证消息 M，发送者首先对 M 进行分组 M_1，M_2，…，M_n。每一个消息组 $M_i(1\leqslant i\leqslant n)$的长度都等于分组加密算法分组长度。若最后一个子消息 M_n 的长度小于分组长度，则必须对其填充一些随机值。

设 C_0=IV 为随机初始值。现在，发送者加密 $M_i(1\leqslant i\leqslant n)$如下：

$$C_i=E_k(M_i\oplus C_{i-1})$$

然后，数值对(IV，C_n)作为 MAC 附在 M 后送出。

与发送者共享密钥 k 的接收者基于接收到的消息和共享密钥 k 用分组加密算法重新加密得到(IV，C_n)，检验与所接收的内容是否一致。如果一致，则可以相信该消息未被篡改并来自所声称的发送者。

很明显，生成数值对(IV，C_n)的计算中包含了不可逆的数据压缩，因此，用分组密码生成 MAC 的计算是单向变换。而且，所用的分组加密算法的混合变换性质为这个单向变换增加了哈希函数的特点(也就是，将 MAC 分布到 MAC 空间与分组加密算法将密文分布到密文空间同样均匀)。所以，用分组密码构造的 MAC 本质上就是整个消息的摘要。

为了提高用分组密码构造 MAC 的安全性，解决消息长度不固定的情况下的安全漏洞，提出了一种基于密码的消息认证码(Cipher-based Message Authentication Code，CMAC)算法。CMAC 适用于 AES 和 3DES 加密算法，并在 NIST SP 800-38B 标准中定义。

首先，当消息长度是分组长度 b 的 n 倍时，对 AES，b=128。这个消息被划分为 n 组 M_1，M_2，…，M_n。算法使用了 k 位的加密密钥 K 和 b 位的常数 K_1。对于 AES，密钥长度 k 为 128 位、192 位或 256 位(见图 3-23(a))，CMAC 按如下方式计算：

$$C_1=E_K(M_1)$$

$$C_i=E_K(M_i\oplus C_{i-1}),\quad 2\leqslant i\leqslant n-1$$

$$C_n=E_K(M_n\oplus C_{n-1}\oplus K_1)$$

$$T = \mathrm{MSB}_{\mathrm{Tlen}}(C_n)$$

其中，T 为 MAC 值，Tlen 为 MAC 值的长度，$\mathrm{MSB}_s(X)$ 表示取位串 X 的高 s 位。

如果消息不是密文分组长度的整数倍（见图 3-23(b)），则最后分组的最低有效位填充一个 1 和若干个 0，使得最后的分组长度为 b。除了使用一个不同的 b 位密钥 K_2 代替 K_1 外，和前面所述的一样进行 CMAC 运算。

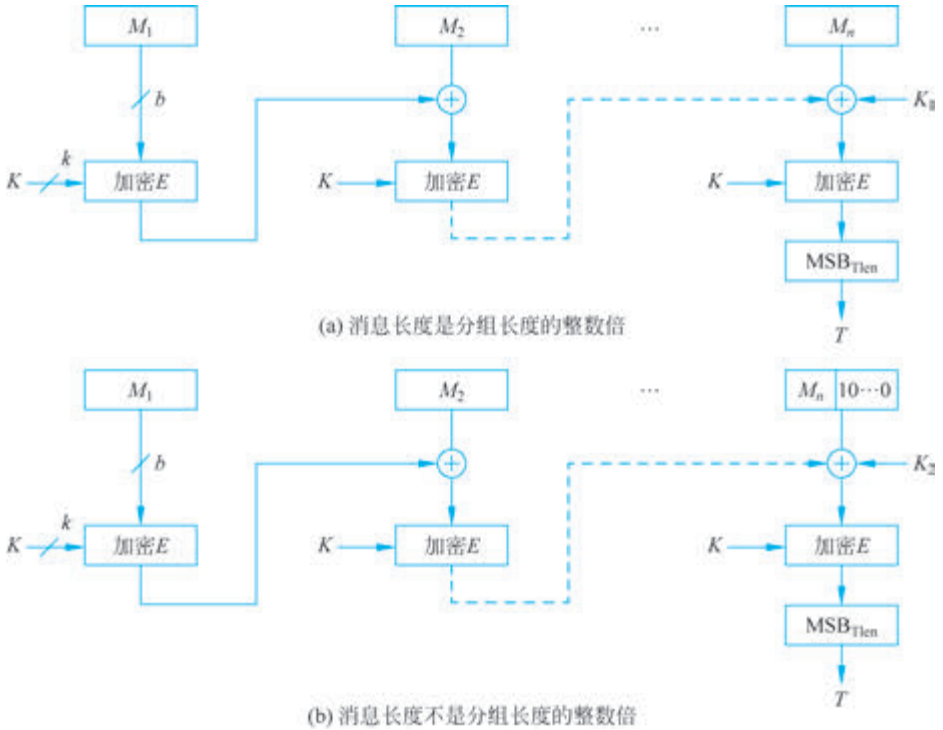

图 3-23 CMAC 算法结构

两个 b 位的密钥 K_1 和 K_2 由 k 位的加密密钥 K 按如下方式导出：

$$L = E_K(0^b)$$

$$K_1 = L \times x$$

$$K_2 = L \times x^2 = (L \times x) \times x$$

其中，0^b 表示 b 位的 0 串，乘法运算“×”在有限域 $\mathrm{GF}(2^b)$ 内进行，x 和 x^2 是 $\mathrm{GF}(2^b)$ 内的 1 次和 2 次多项式。有限域由不可约多项式定义。对于 $b=128$，多项式是 $x^{128}+x^7+x^2+x+1$。

3.3 非对称密码数据完整性技术

在实现数据完整性的非对称密码技术中，密码变换 f 和 g 是公钥密码算法，Ke 和 Kv 为公私钥对。发送方用私钥（公钥）加密消息，消息的密文为 MDC，接收方用公钥（私钥）解密消息，解密是 MDC 验证过程的一个重要步骤。

基于非对称密码技术的数据完整性保护机制，主要依赖两种方法：数字签名算法和无源识别的完整性算法。

3.3.1 数字签名

1. 数字签名的意义

消息认证可以保护信息交换双方不受第三方的攻击，但是不能处理通信双方自身发生的攻击。

如图 3-24 所示，Alice 公司向 Bob 公司进行电子资金转账，并使用了一个共享密钥 K 来生成 MAC 以确保消息的完整性。然而，如果 Bob 是不诚实的，那么他可以修改转账金额（比如将金额从 100 美元增加至 100 000 美元），然后用共享密钥重新计算修改后消息的 MAC，声称这才是来自 Alice 的转账消息。由于 MAC 是基于正确的共享密钥生成的，因此从技术上讲，这个伪造的消息在验证时会被认为是有效的。

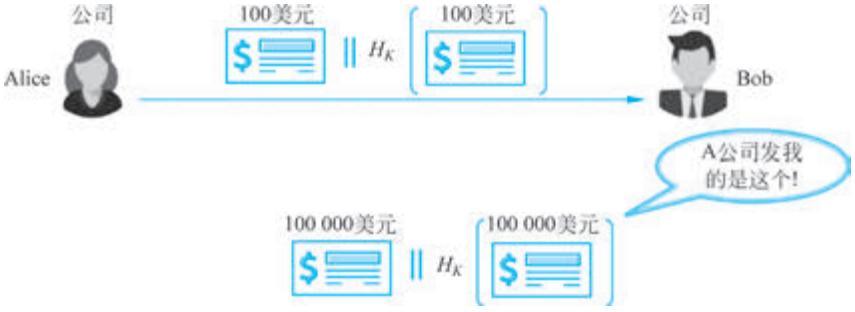

图 3-24 消息认证面临的内部攻击示例一

如图 3-25 所示，股票经纪人 Bob 收到投资用户 Alice 的电子邮件，要他进行一笔三百万股的交易，而这笔交易后来赔钱了，然而，如果 Alice 是不诚实的，她可以否认从未发过这条消息，理由是 Bob 可以伪造。Bob 同样也可否认没有伪造，理由是 Alice 能生成这样的消息。

图 3-25 消息认证面临的内部攻击示例二

其他类似情况还包括，Bob 可以冒充 Alice 发布消息并生成有效 MAC，等等。

前面介绍的对称密码数据完整性技术能防御第三方篡改、伪造，但是无法处理通信双方的内部攻击。因此，在通信双方不能完全信任的情况下，不仅需要数据完整性和认证服务，还需要一种称为不可否认性的安全服务。不可否认性服务防止通信中的任何一方在事后否认曾经进行过的操作或发送过的信息。

数字签名体制能提供以下 3 种安全服务：

- 数据完整性——验证被签名消息的完整性；
- 认证——认证被签名消息的来源；
- 不可否认性——不可否认与一个消息的关系。在电子商务应用中，不可否认性是必不可少的安全要求。

2. 数字签名的基础

数字签名是公钥密码算法的一个重要应用。公钥密码算法作为单向陷门函数，为数字签名提供了坚实的安全基础。

在数字签名过程中，消息的发送者（私钥持有者）使用其私钥对消息（通常是对信息的摘要值）进行加密（或签名）。接收者可以使用发送者的公钥来验证签名的真实性，从而确认消息确实是由私钥持有者发送的，并且在传输过程中没有被篡改。

如图 3-26 所示，只有私钥持有者能用其私钥对消息 Y 加密生成签名 X，签名生成的密码变换在不知私钥时是困难的；已知私钥这个特定信息，签名生成的密码变换是容易的。任何人都可以用公钥对签名 X 进行验证，签名验证的密码变换是容易的。

图 3-26 数字签名的公钥密码算法基础

利用单向陷门函数的特性，数字签名能够确保信息的完整性和来源。能够提供数字签名也是公钥密码相对于对称密码的一个很大的优点。既然只有一个实体可以生成消息的一个数字签名，并可以被任何人验证，所以很容易处理关于是谁生成了该签名的纠纷。

3. 数字签名体制的定义

下面以语法形式给出数字签名体制的定义。一个数字签名体制由以下部分组成。

- 一个明文消息空间 M：某字母表中串的集合；
- 一个签名空间 S：可能的签名集合；
- 一个签名密钥空间 K：用于生成签名的可能密钥集合；
- 一个验证密钥空间 K'：用于验证签名的可能密钥集合；
- 一个有效的密钥生成算法 Gen：$N \to K \times K'$，其中 K 和 K' 分别为私钥和公钥空间；
- 一个有效的签名算法 Sign：$M \times K \to S$；
- 一个有效的验证算法 Verify：$M \times S \times K' \to \{\text{True}, \text{False}\}$；

对任意 $\text{sk} \in K$ 和任意的 $m \in M$，用

$$s = \text{Sign}_{\text{sk}}(m)$$

表示签名变换，读作“s 是用密钥 sk 生成的 m 的签名”。

对任意 $\text{sk} \in K$，用 pk 表示与 sk 相匹配的公钥。对于 $m \in M$ 和 $s \in S$，必有

$$\text{Verify}_{\text{pk}}(m, s) = \begin{cases} \text{True}, & s = \text{Sign}_{\text{sk}}(m) \\ \text{False}, & s \neq \text{Sign}_{\text{sk}}(m) \end{cases}$$

此定义和数据完整性保护定义是相符的，Sign 是 g 变换，sk 是编码密钥，m 是 Data，s 是 MDC，Verify 是 f 变换，pk 是验证密钥。

密钥生成算法 Gen 的输入整数规定了输出签名/验证密钥的规模长度大小。因为密钥生成算法是有效的，其运行时间为输入长度规模的多项式时间，输入的整数值应该是一元编码的。这个整数是数字签名体制的安全参数，定义了签名空间的大小。

香农关于加密算法混合变换的特性同样适用于数字签名体制。算法 Sign 应该是一个很好的混合变换函数：输出的签名值在整个签名空间 S 上分布得相当均匀。这个性质使得如果不适用相应的签名密钥，就不能很容易地生成一个有效的签名。

4. 对数字签名的攻击

分析数字签名的安全性，明确“要保护什么，要防止什么”是至关重要的。对数字签名来说，其核心目标是保护消息的完整性和真实性，即确保消息在传输过程中未被篡改，并且消息的来源是可信的。由于消息本身是公开的，因此保护其机密性不是数字签名的目的。针对数字签名的安全性威胁，攻击者主要试图通过伪造合法签名来达成其不法目的。

所谓合法的签名，就是能够通过验证，并被签名验证者接受的签名。成功的攻击签名方法有完全攻击、选择性伪造攻击、存在性伪造攻击等。

1）完全攻击

这是最严重的攻击类型，意味着攻击者能够完全控制签名过程。他们要么直接获取了签名者的私钥，要么能够构造一个与原签名算法等效但完全由他们控制的算法。在这种攻击下，攻击者能够对任意消息伪造出合法的签名，从而完全破坏数字签名的完整性和认证性。

防止此类攻击的关键在于确保私钥的安全，应采用强加密算法和安全的密钥管理策略。

2）选择性伪造攻击

在这种攻击中，攻击者虽然不能直接获取私钥或完全控制签名算法，但他们能够选择特定的消息，并针对这些消息伪造出合法的签名。尽管攻击者不能随意伪造任意消息的签名，但他们可以针对特定消息（如重要的合同、交易指令等）进行伪造，从而造成严重的后果。

防止此类攻击需要确保签名算法在面对特定消息时仍能保持高度的安全性，同时采用随机性强的签名机制以减少可预测性。

3）存在性伪造攻击

攻击者能够找到一个新的消息（这个消息可能是随机生成的，或者对攻击者具有特定意义的），并针对这个新消息伪造出合法的签名。然而，攻击者通常无法控制这条新消息的具体内容。尽管攻击者不能直接控制伪造签名的消息内容，但他们的成功仍然会破坏数字签名的认证性，因为签名验证者可能无法区分真实签名和伪造签名。需要注意的是，对于存在性伪造攻击，攻击者对新消息伪造出的合法签名是从未被签名者签发过的。

为了抵御此类攻击，需要确保签名算法在生成签名时具有高度的随机性和不可预测性，同时签名验证过程需要严格验证签名的有效性。

5. 数字签名体制

1）RSA 签名体制

RSA 签名体制是继 Diffie 和 Hellman 提出数字签名思想后的第一个数字签名体制，它是由 Rivest、Shamir 和 Adleman 三人实现的。RSA 签名体制的算法描述如下：

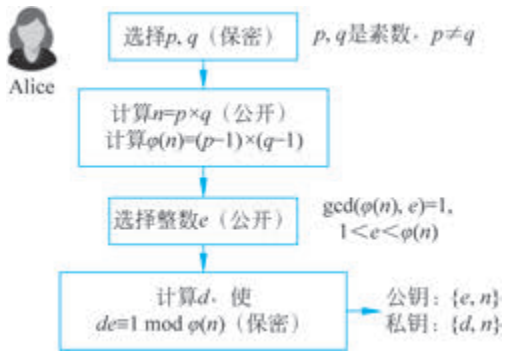

图 3-27 RSA 签名体制的密钥建立过程

（1）密钥建立。密钥建立过程和 RSA 密码系统的密钥建立过程相同。Alice 是签名者，Bob 是验证者，密钥建立过程如图 3-27 所示。因此，用户 Alice 的公钥为(e,n)，私钥为(d,n)。

（2）签名生成。为了生成消息 m（$m<n$）的签名，Alice 计算

$$s=\mathrm{Sign}_{(d,n)}(m)=m^d \bmod n$$

生成消息-签名对(m,s)。

（3）签名验证。Bob 知道公钥(e,n)属于 Alice。给定

一个消息-签名对(m,s)，Bob 的验证过程为

$$\text{Verify}_{(e,n)}(m,s)=\text{True}, \quad \text{若 } m \equiv s^e \bmod n$$

显然，RSA 数字签名过程和 RSA 加解密过程的格式相同，唯一不同的是，现在 Alice 首先用她的私钥进行“加密”，而 Bob(或任何人)再用 Alice 的公钥进行“解密”。有效签名验证一致性与 RSA 加密解密一致性的形式是一样的。

以上描述的 RSA 签名体制提供了数据完整性、认证和不可否认性安全服务。因为验证者 Bob 知道公钥(e, n)属于签名者 Alice，私钥(d, n)只有 Alice 有，这样，Bob(或任何人)收到的签名 s 除了签名者 Alice 外，其他未授权的主体是无法生成的。当 Bob 进行验证，结果为 True 时，表明被签名消息 m 在信道传输时没有受到未授权方式的篡改。数字签名体制对被签名消息 m 进行了数据完整性保护。同时，他能确定消息的确来自所声称的消息源 Alice。再者，只有 Alice 能生成签名，再也不用担心出现无法认定谁生成签名的纠纷，即不可否认性。

以上描述的 RSA 签名体制只能称为“教科书式签名体制”。因为这种签名体制是在非常弱的安全概念下存在的，即假定攻击者是“从零开始”伪造(或生成)一个消息-签名对在计算上是不可行。也就是说，已知公钥和关于签名体制的描述，攻击者需要输出一个有效的消息-签名对，且该消息-签名对从未被目标签名者(即已知公钥的拥有者)所签发。这种假定对于实际应用来说是不适用的，因为它假定攻击者是无能的弱者，或其环境对攻击者是极为苛刻的。攻击者不去想以某些方法使其伪造任务更容易。例如，攻击者可能会利用已经获得的其他有效消息-签名对，或者通过与目标签名者进行交互，诱导其对攻击者精心选择的消息生成有效的签名。

针对“教科书式签名体制”所描述的 RSA 签名算法，任何人伪造 Alice 的签名都不难。例如，Malice 可以选取一个随机数 $s<n$，并用 Alice 的公钥(e, n)计算 $m=s^e \bmod n$。当然，对这样事先准备好的消息-签名对，Bob 用 Alice 的公钥(e, n)验证 $s^e \bmod n=m$，验证结果为 True。Malice 成功地伪造了 Alice 的合法签名。RSA 函数的乘法性也提供了简单的攻击方法，Malice 利用已有的 Alice 的消息-签名对(m_1, s_1)和(m_2, s_2)，计算 $m=m_1\times m_2$，$s=s_1\times s_2$，发送消息-签名对(m, s)。Bob 用 Alice 的公钥(e, N)计算 $s^e=s_1{}^e\times s_2{}^e=m_1\times m_2\equiv m \bmod n$，验证结果为 True。Malice 也成功地伪造了 Alice 的合法签名。

以上攻击都属于存在性伪造攻击，即攻击者能够找到新的消息，并伪造出合法签名。究其原因在于教科书式的 RSA 签名体制中的 m 不可识别，看起来是随机的。数字签名体制的$(\text{Sign}_{sk}, \text{Verify}_{pk})$构成一个单向陷门函数，其中，$\text{Verify}_{pk}$ 是从签名 s 到消息 m 的正向计算，具有混合变换性质，所以从签名 s 生成一个消息 m 是容易的且看起来是随机的，攻击者利用 Verify_{pk} 变换的特点来伪造合法的消息-签名对。

解决办法是为消息增加一些可识别的冗余，使之变得不随机或“有意义”，这样能够使验证者验证消息的非随机分布性。

使被签名消息变得不随机或“有意义”最简单的方法是使消息包含可识别的部分，如 $m=M\parallel l$，其中 M 是要签名的消息，l 为可识别的串，比如签名者的身份。这样，在 RSA 签名生成中，Alice 先对消息 M 格式化 $m=M\parallel l(m<n)$再签名：

$$s=\text{Sign}_{(d,n)}(M\parallel l)=(M\parallel l)^d \bmod n$$

生成消息-签名对(m, s)。Bob 的验证过程为

$$\text{Verify}_{(e,N)}(m, s)=\text{True}, \quad \text{若}(M\parallel l)\equiv s^e \bmod n$$

此时伪造签名不再是容易的，因为很难通过选择 s，使得计算的 $m=M\parallel l$(m 是“有意义的”可识别信息)。

使被签名消息变得不随机或“有意义”的最常用方法是对消息的哈希值 $h(M)$ 进行签名，其中 h 是哈希函数。这样，Alice 先对消息 M 取哈希值 $m=h(M)(m<n)$ 再签名

$$s=\mathrm{Sign}_{(d,n)}[h(M)]=[h(M)]^d \bmod n$$

生成消息-签名对 (m, s)。Bob 的验证过程为

$$\mathrm{Verify}_{(e,n)}(m, s)=\mathrm{True}, \quad \text{若 } h(M)\equiv s^e \bmod n$$

此时伪造签名不再是容易的，因为很难通过选择 s，使得计算的 $m=h(M)$（m 是可识别的消息指纹）。

但是，即便 m 可识别，上述使用哈希函数的 RSA 签名体制也不是“可证明安全”的数字签名体制。在选择性伪造攻击下（攻击者能够选择特定的消息，对所选择的特定消息伪造出合法签名。因为现实中，对于一个给定的公钥签名体制，有大量的消息-签名对可以利用，因为它们并非秘密信息。而且，通常攻击者应当有权要求签名者对攻击者所选取的消息签名），无法证明它是安全的。

根本原因在于使用哈希函数的 RSA 签名体制是确定性算法，对于给定的密钥对(sk, pk)和消息 M，RSA 签名算法输出的签名唯一确定。在密码学中，确定性是我们不希望的性质。使用哈希函数的 RSA 签名体制有一个概率签名方案：RSA-PSS(RSA Probabilistic Signature Scheme)，适于应用，并且可证明安全（伪造签名的困难性同大数因子分解的困难性是联系在一起的），具体可参考文献[11]。

2）ElGamal 签名体制

ElGamal 签名体制是一种基于离散对数问题的非对称数字签名方案，由 Tather ElGamal 在 1985 年提出。

ElGamal 签名体制的安全性可以用形式化证明伪造签名的困难性（包括那些聪明的攻击者）依赖于计算离散对数的困难性。攻击者想要伪造一个有效的签名，必须能够解决离散对数问题，这在计算上是不可行的。此外，ElGamal 签名体制具有非确定性，即对于相同的消息，由于随机数 k 的不同，会产生不同的签名，这进一步增加了伪造签名的难度。如前所述，在密码学中，确定性是我们不希望的性质。

因为设计精巧，许多实用的签名体制都是模仿它设计的，由此产生了类 ElGamal 签名体制族，如 Shnorr、DSS 等签名方案。

ElGamal 签名体制主要由 3 部分组成：密钥建立、签名生成和签名验证。

（1）密钥建立。Alice 是签名者，Bob 是验证者，Alice 执行以下步骤：

- 随机选择大素数 p，并生成有限域 GF(p)的一个随机乘法生成元 r；
- 随机选择一个整数 $a(1<a<p-1)$，计算 $y=r^a \bmod p$。

至此，Alice 就产生了一对公钥 (p, r, y) 和私钥 a。

（2）签名生成。对消息 M 签名，Alice 执行以下步骤：

- 计算消息摘要值 $m=h(M)$；
- 选择一次性随机整数 $k(1<k<p-1, \gcd(k, p-1)=1)$，计算 $s_1\equiv r^k \bmod p$；
- 计算 $s_2\equiv (m-as_1)k^{-1}(\bmod (p-1))$。

生成消息-签名对 $(m, (s_1, s_2))$。其中，a 是 Alice 的永久私钥，k 是 Alice 的一次性临时私钥。Alice 先用临时私钥 k 生成签名的第一部分 s_1。接着，将消息 m 表示为永久私钥 a 和临时私钥 k 的线性组合，组合系数为签名的第一部分 s_1 和第二部分 s_2：

$$m=as_1+ks_2(\bmod (p-1))$$

（3）签名验证。给定消息-签名对 $(m, (s_1, s_2))$，Bob 的验证过程为

$$\mathrm{Verify}_{(p,r,y)}(m,(s_1,s_2))=\mathrm{True} \quad \text{若 } s_1<p, y^{s_1}s_1^{s_2}\equiv r^m \bmod p$$

理由如下：

$$y^{s_1}s_1^{s_2} \equiv r^{as_1}r^{ks_2} \bmod p = r^{(as_1+ks_2)} \bmod p = r^m \bmod p$$

3）其他签名体制

数字签名领域还存在多种实用且可证明安全的标准，其中 DSS(Digital Signature Standard)是由美国国家安全局(NSA)开发并经美国国家标准与技术研究院(NIST)发布的数字签名标准，其安全性基于离散对数问题，目前针对该问题的经典计算机攻击方法仅有亚指数时间算法，该标准广泛应用于美国政府、金融及软件分发等领域。ECDSA(Elliptic Curve Digital Signature Algorithm)是一种基于椭圆曲线的数字签名算法，于 1998 年被 ISO 采纳为标准，后于 1999 年和 2000 年分别成为 ANSI、IEEE 及 NIST 的标准，其特点是密钥长度短且安全性高(256 位 ECDSA 相当于 3072 位 RSA)，主要应用于区块链(如比特币)、移动设备及物联网(IoT)。中国的 SM2 数字签名算法由国家密码管理局发布，安全性基于椭圆曲线离散对数问题，具有密钥长度短(256 位)和计算效率高的优势，支持数字签名、密钥交换和公钥加密，广泛应用于电子政务、金融及电子商务等领域；2018 年 11 月，SM2 成为 ISO/IEC 国际标准，标志着我国自主设计的数字签名算法在国际标准化工作方面实现了重大突破。


视频 17

3.3.2 无源识别的数据完整性

1. 无源识别的含义

在利用数字签名体制实现的数据完整性技术中，密钥参数通常的设置约定为：Ke 为私钥，Kv 为与之匹配的公钥。在这种情况下，一条消息的完整性的正确验证结果向验证者提供了有关消息发送者，即该消息的签名者，亦即公钥 Kv 的拥有者的身份。

尽管这种密钥参数常用的设置方式是实现数字签名体制的一个必要组成部分，但是对于数据完整性系统并不是必需的。事实上，在定义数据完整性保护时，从未对生成和验证 MDC 的两个密钥作任何限制。因此，可以采用与数字签名体制相反的方式设置这两个密钥 Ke 和 Kv，也就是说，设 Ke 为公钥，Kv 为私钥。在这种密钥设置方式下，任何人都可使用公钥 Ke 生成一个一致性的数值对(Data，MDC)，只有私钥 Kv 的拥有者才可验证数值对(Data，MDC)的一致性。

因为数字签名机制在保护数据完整性的同时提供了数据源的主体身份知识，这称有源识别数据完整性。因为任何人都可使用公钥 Ke 生成一个一致性的数值对(Data，MDC)，接收方在判定数据完整性的同时并不知道数据是由哪个主体签发的，这称为无源识别数据完整性。依据对 Malice 的行为的了解，为了方便，甚至可以将这种数据完整性服务称为“来自 Malice 的数据完整性”。

2. RSA-OAEP 算法

下面介绍一个提供无源识别的数据完整性的例子——RSA-OAEP(RSA-Optimal Asymmetric Encryption Padding，RSA 最优非对称加密填充)算法，一种结合 RSA 加密和 OAEP 填充方案的公钥加密算法。

该算法具有这样的性质：任何人(不妨假设 Malice)都可以发给 Alice 一条加密的消息，但对于 Malice 所在团伙中的任何其他人来说，若要修改该密文而不被 Alice 发现在计算上是困难的。

该算法的过程如下：

(1) 密钥建立。设 Alice 的 RSA 公钥和私钥分别为(e, N)和(d, N)，其中 $d = e^{-1} \bmod \phi(N)$，N 的长度为 $k = n + k_0 + k_1$，n 是明文消息长度，2^{-k_0} 和 2^{-k_1} 均为可忽略的量，G 和 H 是两个哈希函数，且满足

$$G:\{0,1\}^{k_0} \mapsto \{0,1\}^{k-k_0}, \quad H:\{0,1\}^{k-k_0} \mapsto \{0,1\}^{k_0}$$

(2) 加密过程。为了发送一个消息 $m \in \{0, 1\}^n$ 给 Alice，Malice 执行以下运算：

① 随机选择 k_0 位长的二进制比特串，计算 $r \leftarrow \{0,1\}^{k_0}$，$s \leftarrow (m \| 0^{k_1}) \oplus G(r)$，$t \leftarrow r \oplus H(s)$；

② 如果($s \| t \geqslant N$)则返回①；

③ $c \leftarrow (s \| t)^e \bmod N$。

所得密文为 c。其中，“‖”表示比特串的链接，“⊕”表示按位 XOR 运算，0^{k_1} 表示 k_1 个 0 组成的串，用作冗余，以便解密时进行数据完整性验证。

(3) 解密过程。收到密文 c 后，Alice 执行以下运算：

① $s \| t \leftarrow c^d \bmod N$ 满足 $|s| = n + k_1 = k - k_0$，$|t| = k_0$；

② $u \leftarrow t \oplus H(s)$；$v \leftarrow s \oplus G(u)$；

③ 输出 $\begin{cases} m, & \text{若 } v = m \| 0^{k_1} \\ \text{拒绝}, & \text{其他} \end{cases}$

当输出为“拒绝”时，就认为密文是无效的。

若密文 c 是完整的，则 Alice 能在解密第①步正确地获得 s 和 t，接着，Alice 能在解密第②步正确地恢复 r，最后，Alice 将在解密第③步看到 k_1 个 0 组成的串，前面即明文消息 m。若篡改密文 c，将使得解密后的($s \| t$)变化，由此使得 r 发生不可控制的变化，从而导致 ($m \| 0^{k_1}$) 发生不可控制的变化。直觉上，这种不可控制的改变是由该体制所用的两个哈希函数的雪崩效应引起的。本质上，这种不可控制的改变的出现是由于破坏了明文中的冗余信息，即 k_1 个，0 组成的串(改变概率至少为 $(1-2^{-k_1}) \to 1$)。所以，对密文的改变将导致解密的明文发生改变，密文被 Alice 认为无效。

RSA-OAEP 算法提供的数据完整性保护很奇特。它为密文提供了数据完整性保护：当 Alice 看到 k_1 个 0 串后，能确信密文没有被修改过，如果 Malice 所在团伙中的任何其他攻击者试图修改密文，那么数据完整性检验无法通过，解密的结果为一条无意义的消息。但 Alice 不知道该消息的发送者是谁，即无源识别的数据完整性。

本章小结

数据完整性保护是指通过一系列技术手段，确保数据在生成、处理、存储、传输直至使用等整个生命周期中保持其原始性和准确性，防止数据被非法篡改或破坏的过程。

本章在详细介绍数据完整性保护原理的基础上，深入探讨了数据完整性保护的对称密码技术和非对称密码技术。对称密码数据完整性技术依赖于加密和解密使用相同密钥的特性，通过密钥哈希函数或分组密码加密算法来生成数据的完整性校验值。发送方将原始数据与校验值一同发送给接收方，接收方使用相同的密钥和算法验证校验值，从而确认数据的完整性。非对称密码数据完整性技术基于公钥密码体系的数学特性，通过数字签名算法或无源识别的数据完整性算法来完成数据的完整性证明。数字签名利用私钥生成唯一性签名，接收方则通过对应公钥进行验证，这一过程不仅能够检测数据是否遭到篡改，同时可准确验证消息来源的真实性。无源识别的数据完整性是一种特别的数据完整性保护方案，接收方在能够判定数据完整性的同时并不知道数据是由哪个主体签发的。