第5章

物联网安全基础——数据完整性检验

无论是对称密码方案或非对称密码方案，都是保护传输、存储数据的机密性，即实现安全通信。然而在现实中，仅仅保护数据的机密性是不够的。使用加密方案，可以保护通信的内容不被攻击者获知。但对于接收者，如何验证所接收到的数据是指定的发送方发送的呢？在实际应用中，攻击者可能对传输中的加密信息进行篡改，令接收方得到错误的加密信息。因此，如何让接收者验证接收到的消息确实是指定的发送方产生的，在实际的应用中与安全通信同样重要。由于数据加密与数据完整性检验所要达到的目标是不同的，因此要采用不同的技术手段。在本章中，先介绍广泛使用在数据完整性检验及其他领域中的哈希函数，再介绍两种数据完整性检验的方法，即基于对称密钥的消息认证码和基于非对称密钥的数字签名。这两种方法也刚好与数据加密技术中的对称加密和公钥加密相对应。

5.1 哈希函数与伪随机函数

5.1.1 哈希函数

哈希函数(Hash Function)也称散列函数，是一种将任意长度的字符串压缩成为固定长度的短字符串的函数，有时这个短字符串也称消息的摘要。哈希函数广泛使用在数据结构中用于构造哈希表(Hash Table)等。哈希函数的一个重要性质就是抗碰撞性。在哈希表中，希望两个不同的值落在同一个位置的概率尽可能小，即尽量避免碰撞的发生。而密码学中的哈希函数则希望碰撞发生的概率是可忽略的，即需要更强的抗碰撞性要求。非正式地讲，一个安全的哈希函数应当满足几乎不可能找到不同的两个值 x_1, x_2，使它们的哈希结果相等。安全的哈希函数对构造安全的数字签名方案至关重要，几乎所有安全的数字签名方案都需要用到哈希函数。

定义 5.1 哈希函数。一个哈希函数由一对概率多项式算法(HGen, H)组成，并且满足下面两个条件。

(1) HGen 是一个概率算法，其输入为安全参数 λ，输出一个密钥 s。

(2) H 是一个确定性算法，对于确定的 s 和输入 $x\in\{0,1\}^*$，输出 $H^s(x)=\{0,1\}^{l(\lambda)}$ 的字符串。其中，$l(\lambda)$是一个 λ 的多项式。

需要注意的是，s 虽然称为密钥，但是在一般的应用中由于不需要保密的，因此 s 经常省略，直接将 H 称为哈希算法。

在密码学中，使用以下 3 种安全定义来表示一个哈希函数所达到的安全强度。

(1) 单向性(One-Way)：给定哈希函数 H 和一个哈希值 y，如果寻找一个原像 x，使 $H(x)=y$ 在计算上是不可行的，则称 H 是单向的，也称抗原像(Preimage Resistant)攻击的。

(2) 弱抗碰撞性(Weak Collision-Resistant)：给定哈希函数 H 和一个原像 x，找到另一个 x'，使 $H(x')=H(x)$在计算上是不可行的，则称 H 具有弱抗碰撞性，也称可抵抗第二原像(Second Preimage Resistant)攻击。

(3) 强抗碰撞性(Strong Collision-Resistant)：给定哈希函数 H，找到任意两个不同的 x、x'，使 $H(x')=H(x)$在计算上是不可行的，则称 H 具有强抗碰撞性。

实际上，当约定哈希函数中输入的长度大于输出的长度时(即哈希函数是一个压缩算法)，任何一个强抗碰撞性的哈希函数都具有弱抗碰撞性和单向性，而任何一个弱抗碰撞性的哈希函数都是单向的，即抗原像攻击的。这 3 种安全定义的强度逐级增强，其中强抗碰撞性是最强的安全定义。

哈希函数的通用攻击方法是生日攻击。在不考虑构造安全性的情况下，哈希函数的安全强度与输出的长度密切相关。常见的安全哈希函数有 MD 家族(MD4、MD5)与 SHA 家族(SHA1、SHA2、SHA3)。其中 MD4、MD5 以及 SHA1 已经被认为是不安全的哈希函数，且无法保证抗碰撞性。现代的攻击方法可以在几分钟之内找到 MD4、MD5 中有效的碰撞。如何构造和攻击哈希函数是属于密码学领域的研究方向，这部分内容超出了本书的范围。基于目前的研究，普遍认为 SHA2(SHA256、SHA384、SHA512)、SHA3 哈希函数是安全的，而 MD 家族及 SHA1 哈希函数是不安全的。

5.1.2　伪随机函数

在 4.2.3 节，我们讲到使用伪随机生成器来产生安全的序列密码。与伪随机生成器，伪随机函数(Pseudorandom Function)也是一种产生随机字符串的函数。伪随机函数可以接收任意长度的输入，并且输出一个随机的字符串。一个安全的伪随机函数，其输出应当与真正的随机字符串是不可区分的。伪随机函数与哈希函数类似，都是确定性的算法。对于确定性的算法，讨论其伪随机性是无意义的，实际上伪随机函数输出的伪随机性其实是针对函数输出的分布。因此伪随机函数的输入还需要包括一个密钥。

定义 5.2　伪随机函数。一个伪随机函数 F 是一个确定性的函数，对于确定的输入 $x\in\{0,1\}^*$ 和密钥 $k\in\{0,1\}^*$，其输出为固定的值 $y\in\{0,1\}^*$，即

$$F:\{0,1\}^*\times\{0,1\}^*\to\{0,1\}^*$$

上述对伪随机函数的定义是一个一般形式的定义。而在实际使用中，密钥的长度往往会与安全参数相关，并且其输入和输出均为固定长度。则对于一个确定性的密钥 k，伪随机函数可以定义为 $F_k:\{0,1\}^\lambda\to\{0,1\}^{l(\lambda)}$，此处 $l(\cdot)$为关于输入长度 λ 的多项式。

伪随机函数作为一种基本原语，在密码学中具有十分重要的作用。基于伪随机函数，可以构造安全的加密方案、消息认证码方案等。在4.2节，介绍了对称密码和一些常见的密码方案。实际上，利用一个安全的伪随机函数可以一般化地构造一个安全的对称加密方案。方便起见，假设 F 为一个输入、密钥和输出长度均为 λ 的定长伪随机函数，则基于 F 构造的对称加密方案如下。

(1) Gen(λ)：密钥生成算法。算法输入安全参数 λ，然后输出一个随机密钥 $k \in \{0,1\}^{\lambda}$。

(2) Enc(k,m)：加密算法。算法输入对称密钥 $k \in \{0,1\}^{\lambda}$ 和加密消息 $m \in \{0,1\}^{\lambda}$，输出密文：

$$c = (r, F_k(r) \oplus m)$$

其中，r 是长度为 λ 的随机字符串。

(3) Dec(k,c)：解密算法。算法输入对称密钥 $k \in \{0,1\}^{\lambda}$ 和一个密文 $c = (c_1, c_2)$，输出明文：

$$m = F_k(c_1) \oplus c_2$$

在上述加密构造中，消息和一个随机值 r 的伪随机函数输出 $F_k(r)$ 进行异或，从而被隐藏起来。在每次加密过程中，随机值 r 都是均匀随机从 $\{0,1\}^{\lambda}$ 空间中选取的。因此，即使加密同一个消息，也会得到完全不同的密文。实际上，如果伪随机函数是安全的，这种加密方法其实达到了选择明文攻击安全。

5.2 消息认证码

5.2.1 消息认证码的基本概念

正如前面所提到的，加密并不能保证数据的完整性。利用加密方案，可以使通信双方在公开信道上进行消息的传输，并且不用担心通信内容被监听者获取。然而这并不能保证恶意的攻击者对通信的内容进行篡改(攻击者不需要知道通信的内容是什么就可以篡改通信内容)。例如，在使用公钥密码的通信系统中，攻击者可以简单地拦截发送方发送的密文，并替换为自己想要接收者收到的内容，从而使通信双方遭受巨大的损失。即使使用对称加密方案，攻击者仍然可以通过翻转密文中的某些位，使解密的结果完全不同。因此，加密无法保证数据的完整性，并且所有的加密方案单独使用都无法保证数据的完整性。

为了实现数据完整性验证，就需要额外的验证机制来检查通信的消息是否被篡改。消息认证码可以实现：如果有攻击者在数据传输过程中修改了数据，接收者能够以极大的概率检测出来。消息认证码与对称加密相似，都假设通信的双方掌握了其他人不知道的秘密信息，即共享一个密钥。在这个前提下，发送消息的一方使用这个共享密钥，对消息产生认证标签，然后将消息和标签一起发送给接收方(为了保证机密性，可以对消息和标签进行加密)。消息接收方收到消息和标签后，使用共享的公私钥对消息和标签进行验证，从而得出该消息的来源是否可靠。

定义 5.3　消息认证码。一个消息认证码(Message Authentication Code,MAC)由一组概率多项式算法(Gen,Mac,Verify)组成,并满足下面 3 个条件。

(1) Gen 算法。密钥生成算法是一个概率算法,其输入为安全参数 λ,输出一个密钥 k。

(2) Mac 算法。标签生成算法同样是一个概率算法,其输入为密钥 k 和一个消息 $m\in\{0,1\}^*$,输出一个关于消息 m 的标签 t。

(3) Verify 算法。验证算法是一个确定性算法,其输入为密钥 k、消息 m 和消息的标签 t,输出为 1 位,意味着标签 t 是否有效。

消息认证码的正确性:对于任意的安全参数 λ、任意长度的消息 $m\in\{0,1\}^*$ 和密钥 $k\leftarrow\mathrm{Gen}(\lambda)$,都有 $\mathrm{Verify}(k,m,\mathrm{Mac}(k,m))=1$。

类似加密方案的安全性定义,消息认证码的安全性也从方案所要达到的目标以及攻击者的能力两方面来定义。对于攻击者的能力,一般总是约束攻击者为概率多项式时间(PPT)的攻击者。考虑消息认证码的目的是保证消息来源总是可靠的,因此一个攻击者的目标则是伪造一个消息 m^* 和相应的标签 t^*,使它们能够通过 Verify 算法的检验。为此,允许攻击者适应性地选择消息,并获得对应的标签帮助其攻击消息认证码。这种攻击者称为适应性概率多项式攻击者,而这种攻击方式则称为适应性选择消息攻击。而消息认证码所要达到的目标则是让攻击者无法伪造一对有效的消息和标签,即达到存在性不可伪造。因此,一个安全的消息认证码的安全需求是在适应性选择消息攻击下达到存在性不可伪造。

重放攻击。当攻击者获取到消息和相应的标签后,可以重新将这组消息和标签发送给接收方。在重放过程中,攻击者没有对消息或标签进行任何的修改,因此这组消息和标签可以合法地通过 Verify 算法的检验。对于没有合理设计的系统,重放攻击是一种有效且直接的攻击方式,往往会对系统造成极大影响。消息认证码无法抵抗重放攻击。这是因为消息认证码只对消息本身进行认证,而不考虑其状态。

抵抗重放攻击的方法一般有两种,即在消息中加入序列号或时间戳。序列号技术的基本思路是对每个消息都进行编号,任何被认证过的消息序号都会被接收者存储。当接收者收到的消息包含已经认证过的序号时,就可以认为该消息被重放了。使用序列号方式的缺点是接收者必须存储消息序列,会产生额外的存储开销。因此在实际使用中,往往会在消息中加入时间戳。此时,发送者和接收者之间需要同步一个时钟。发送者发送消息时会将当前时间附加在消息中,进行认证产生标签。而接收者在收到消息时,除了验证标签的有效性外,还要验证消息中的时间是否在一个合理的时间区间中;否则就认为这是一个不合法的消息。时间戳也只能在一定程度上抵抗重放攻击。当攻击者的重放速度很快时(在合法的时间区间内进行重放),攻击依然可能有效。

5.2.2　常见的消息认证码方案

1. CBC-MAC 方案

在 4.2 节对称密码部分介绍了密码分组链接(Cipher-Block Chaining,CBC)模式,

CBC-MAC 方案与其构造十分相似，主要用于处理较长消息时使用。设消息 m 的长度为 $l\lambda$，其中 λ 是每个消息分组的长度，并且 MAC 标签的长度只有 λ。CBC-MAC 方案基于带密钥的伪随机函数。在 5.1.2 节中，介绍了如何使用伪随机函数构造安全的对称加密方案，因此很多时候，在构造 CBC-MAC 方案时也可以使用对称加密方案来代替伪随机函数。不失一般性，我们假设 F 为一个输入、密钥和输出长度均为 λ 的定长伪随机函数。则基于伪随机函数 F 构造的 CBC-MAC 方案标签生成过程如图 5.1 所示，CBC-MAC 构造方法如下。

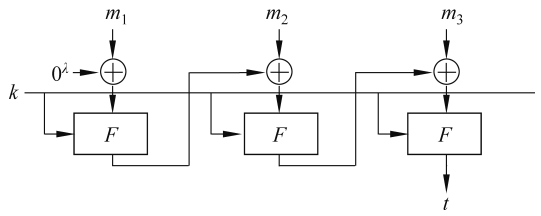

图 5.1　CBC-MAC 方案标签生成过程

(1) Gen：算法输入安全参数 λ，然后输出一个随机密钥 $k\in\{0,1\}^\lambda$。

(2) Mac：对于一个长度为 $l\lambda$ 的消息 m，将消息分为 l 份，每份消息 m_i 长度均为 λ，并令 $t_0=0^\lambda$，迭代计算 $t_i=F_k(t_{i-1}\oplus m_i)$，$1\leqslant i\leqslant l$。输出 $t=t_l$。

(3) Verify：对于输入密钥 k、消息 m 和标签 t，当且仅当 $t=\mathrm{Mac}(k,m)$ 时输出 1，否则输出 0。

在 CBC-MAC 方案中，使用了伪随机函数作为构造组件。标签生成算法最终的输出只有最后一个区块的标签 t_l，前面 $l-1$ 个区块的标签都不需要发送给接收者。这是因为接收者不需要中间的标签，即可验证消息的有效性。此外，这部分标签如果被中间的攻击者拦截，反而会导致方案不安全。例如，攻击者通过获取标签 t_{l-1}，可以让接收者通过消息前 $(l-1)\lambda$ 位的验证，即产生了对新消息的合法标签(虽然新消息为原始消息的一部分)。

如果底层的伪随机函数(或对称加密方案)是安全的，那么上述所构造的 CBC-MAC 方案对于固定长度的消息也是安全的，但是对于不固定长度的消息却是不安全的。也就是说，对于任意一个密钥 k 只能对固定长度的消息或已知长度的消息产生消息认证码。当上述 CBC-MAC 方案在处理变长消息时，攻击者可以利用两个消息-认证码对 (m_1,t_1) 和 (m_2,t_2) 构造一个新的消息 m^*，使它的消息认证码为 t_2。不失一般性，假设消息 m_1 和 m_2 的长度为 $l_1\lambda$ 位和 $l_2\lambda$ 位，m_b^n 为 m_b 的第 n 个 λ 位。攻击者将构造的新消息 m^* 设置为 $m_1\,|\,(t_1\oplus m_2^1)\,|\,m_2^2\,|\cdots|\,m_2^{l_2}$。当验证者去验证新的消息-认证码对 (m^*,t_2) 时，m^* 在输入 CBC-MAC 方案的第 l_1 个块后会得到中间标签 t_1。这是因为 m^* 的前 $l_1\lambda$ 位刚好为 m_1。当中间标签 t_1 与 m^* 的第 l_1+1 个块进行异或时，由于 $m^{*l_1+1}=t_1\oplus m_2^1$，因此有 $m^{*l_1+1}\oplus t_1=m_2^1$。所以，计算 m^* 的认证码时的第 l_1+1 个块输入伪随机函数的值与计算 m_2 的认证码时的第一个块输入伪随机函数的值相同。最终得到 m^* 的标签也与 m_2 的标签相同，即为 t_2。

将 CBC-MAC 方案扩展到变长消息场景下的方法主要有 3 种：Input-length key

separation，Length-prepending 和 Encrypt last block。在这 3 种方法中，Length-prepending 和 Encrypt last block 方法被讨论和使用的场景最多。如图 5.2 所示，使用 Length-prepending 可以保证 CBC-MAC 方案的安全性。在这种方法中，想要对一个消息产生消息认证码，首先将消息的长度作为一个单独的区块附在消息的第一个块中，然后再执行标准的 CBC-MAC 方法。这种方法要求消息认证码的产生要预先知道消息的长度。这就要求用户需要先存储消息，再进行消息认证。因此，有不少开发者在实际的使用中采取将消息的长度附在消息最后来产生消息认证码，但这种做法是不安全的。在这里不再展开如何攻击一个将消息的长度作为后缀的 CBC-MAC 变形，而是将它作为读者课后的扩展学习和调研。再次强调，使用不经验证的密码方案或对原有的密码方案进行改变，在实际应用中是十分危险的行为，可能给系统引入十分严重的安全漏洞。

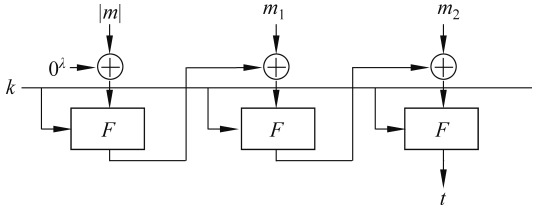

图 5.2　Length-prepending CBC-MAC 方案标签生成过程

加密密码分组链接（Encrypt last block CBC-MAC，ECBC-MAC）方案可以使消息认证码产生方在不知道消息长度的情况下产生消息认证码，并且保证方案是安全的。如图 5.3 所示，在 ECBC-MAC 方案中，密钥由两个不等的部分组成，即k_0和k_1，其中k_0为伪随机函数密钥，k_1则为一个对称加密密钥。在对一个消息 m 计算消息认证码时，首先使用密钥 m 经过原始的 CBC-MAC 方法计算 m 的一个临时标签 t，再使用k_1将 t 进行加密得到消息 m 的最终标签 t'。即表示为

$$\text{ECBC-MAC}(k_0,k_1,m)=\text{Enc}(k_1,\text{CBC-MAC}(k_0,m))$$

其中，Enc 为对称加密方案的加密算法。在收到一组消息-认证码对(m,t)之后，验证者可以通过两种方法验证消息的有效性：①同样使用的 ECBC-MAC 方法计算出另一个标签 t'，使之与 t 进行对比；②先将 t 进行解密，再将解密结果与通过 CBC-MAC 方法计算出来的标签对比。

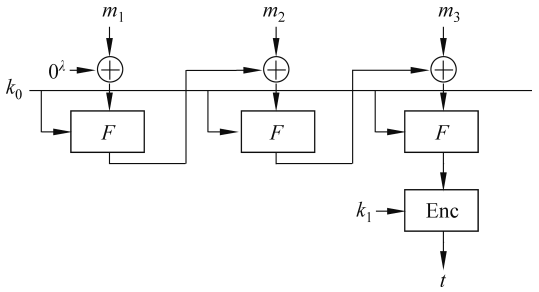

图 5.3　ECBC-MAC 方案标签生成过程

细心的读者可能也会发现 CBC-MAC 方案与 CBC 模式相比，初始向量也是不同的。在 CBC-MAC 方案中，初始向量始终为全 0 的位串，而 CBC 模式则要求使用一个随机的初始向量。这是因为在 CBC-MAC 方案中使用随机的初始向量是不安全的。这个问题也留给读者进行思考。

2. 从哈希函数构造 MAC 方案

留意到哈希函数可以将任意长度的消息压缩为一个短的随机的摘要信息，因此从哈希函数构造 MAC 方案是一种很直觉的想法，事实上也有一些方案是这样做的，如基于哈希函数的消息认证码（Hash-based Message Authentication Code，HMAC）。在 HMAC 出现之前，由于 CBC-MAC 方案是基于对称加密方法或伪随机函数的，不少工业界开发者认为它太慢而拒绝使用，反而采取基于哈希函数的构造方法。一个典型的例子就是将 MAC 方案的密钥作为消息的一部分输入哈希函数中，并产生摘要信息作为该消息的标签。然而，在使用这类构造时，必须十分小心。在介绍 HMAC 前，先给出一个基于该思路来构造的 MAC 例子，并指出它是不安全的。假设$H^s(\cdot)$是一个强抗碰撞性的哈希函数，则基于该哈希函数的 MAC 方案构造如下。

（1）Gen：算法输入安全参数 λ，然后输出一个随机密钥 $k\in\{0,1\}^\lambda$。

（2）Mac：对于一个消息 $m\in\{0,1\}^*$ 和密钥 $k\in\{0,1\}^\lambda$，算法输出该消息的标签 $t=H^s(k\parallel m)$。

（3）Verify：对于输入密钥 k、消息 m 和标签 t，当且仅当 $t=\mathrm{Mac}(k,m)$时输出 1，否则输出 0。

从直觉上看，由于密钥 k 是通信双方秘密保存的，因此对于一个消息 m，任何人都无法计算出它的标签$H^s(k\parallel m)$。然而事实却是上述的构造方法在采用某些哈希函数时，是极不安全的，如基于 Merkle-Damgard 变换的哈希函数。这部分内容由于涉及哈希函数的构造方法，超出了本书的讲解范围，因此不再进行详细解释。不过仍然提醒读者在实际使用密码方案时，不经验证地改动密码算法或使用拍脑袋密码算法可能会使系统暴露在不可知的安全威胁之下。

3. HMAC 方案

基于哈希函数的消息认证码（Hash-based Message Authentication Code，HMAC）是结合了哈希函数的消息认证码，并且克服了上述使用哈希函数构造 MAC 时的缺陷。设 H 为一个具有强抗碰撞性质的哈希函数。HMAC 方案标签生成过程如图 5.4 所示，HMAC 方案的构造方法如下。

（1）Gen：算法随机选取一个密钥 $k\in\{0,1\}^\lambda$。

（2）Mac：对于一个任意长度的消息 $m\in\{0,1\}^*$，算法输出标签

$$t=H((k\oplus\mathrm{opad})\parallel H((k\oplus\mathrm{ipad})\parallel m))$$

其中，opad 和 ipad 为两个常量，分别为外部填充和内部填充。

（3）Verify：对于输入密钥 k、消息 m 和标签 t，当且仅当 $t=\mathrm{Mac}(k,m)$时算法输出 1，否则输出 0。

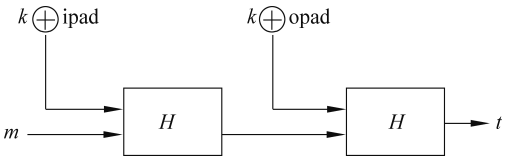

图 5.4　HMAC 方案标签生成过程

HMAC 中使用的哈希函数必须要求是抗碰撞的，这样才能保证其安全性。需要注意的是，由于密钥 k 的长度，有可能超过一次哈希函数所能处理的块的长度，因此在这种情况下，实际输入 Mac 算法中与 opad 和 ipad 进行异或的值为 k 的哈希值，即 $k'=H(k)$。另外 opad 和 ipad 分别为不同的常量，即重复 0x36 和 0x5C 这两个十六进制值。HMAC 本身是工业标准，并且广泛使用在各类系统的实现中。HMAC 中的操作只涉及哈希运算、字符串拼接与异或操作，因此它十分高效。同时它也保证了极高的安全性，即当哈希函数满足强抗碰撞性时，在适应性选择消息攻击下达到存在性不可伪造。

5.3　数字签名

5.3.1　数字签名的基本概念

数字签名(Digital Signature)与消息认证码的作用类似，都可以对数据的完整性进行检验。不同的是，数字签名允许公开检验，即检验者不需要共享数字签名产生者的秘密信息，而是基于公开信息就可以对数据的完整性和来源有效性进行检验。数字签名产生者首先会产生一对密钥，这与非对称加密方案有些类似。这组密钥中由数字签名者保存的称为签名密钥(私钥)，而另一个用于验证签名有效性的密钥则会被公开，称为验证密钥(公钥)。一旦签名者产生了上述的公私钥对后，可以对任意的消息签署一个签名，并允许任何拥有公钥的人来验证该签名的有效性。如果对某个消息的签名是合法的，则证实该签名确实是由签名者认证签发的，即消息来自签名者。除此之外，数字签名还提供了不可抵赖的性质，即数字签名的产生者一旦对某个消息进行签名后，无法否认这个签名的有效性(有时也称该性质为不可否认性)。公开验证和不可抵赖性是数字签名的重要性质，也正是由于这两个性质，数字签名可以用于签署网络合同、文档等，并在很多国家中具有法律效力。

定义 5.4　数字签名。一个数字签名方案由一组概率多项式算法(Gen，Sign，Verify)组成，并满足以下 3 个条件。

(1) Gen 是一个概率算法，其输入为安全参数 λ，输出一对密钥(vk，sk)，分别为验证密钥和签名密钥，其中 vk 公开，sk 由签名者秘密保存。

(2) Sign 是一个概率算法，其输入为签名密钥 sk 和一个消息 $m\in\{0,1\}^*$，输出一个关于消息 m 的签名 σ。

(3) Verify 是一个确定性算法，其输入为验证密钥 vk、消息 m 和消息的签名 σ，输出为 1 位，意味着签名 σ 是否有效。

数字签名的正确性：对于任意的安全参数 λ、消息 $m\in\{0,1\}^*$ 和密钥(vk,sk)←Gen(λ)，都有 Verify(vk,m,Sign(sk,m))=1。数字签名的安全性要求与消息认证码类似，也是需要达到适应性选择消息攻击下的存在性不可伪造。

5.3.2 常见的数字签名方案

1. RSA 签名方案

初学者往往会认为 RSA 签名方案是 RSA 加密方案的逆向操作，即使用 RSA 加密方案的私钥加密，公钥解密。实际上这种认识是错误的，并且构造是不安全的。类似 RSA 加密，先从不安全的教科书式 RSA 签名方案出发来讲述安全的签名方案的构造。RSA 签名方案同样是基于 RSA 假设的，具体构造如下。

(1) Gen(λ)：算法输入安全参数 λ，然后选取两个长度为 λ 的大素数 p 和 q，$n=pq$，$\phi(n)=(p-1)(q-1)$。随机选取一个 e，使 $\gcd(e,\phi(n))=1$，并计算 d，使 $de=1 \bmod \phi(n)$。则 RSA 签名的验证密钥为 vk=(n,e)，签名密钥为 sk=(n,d)。

(2) Sign(sk,m)：算法输入 RSA 签名密钥 sk=(n,d)和消息 $m\in\mathbf{Z}_N^*$，输出签名 $\sigma=m^d \bmod n$。

(3) Verify(vk,m,σ)：算法输入 RSA 签名验证密钥 vk=(n,e)，消息 $m\in\mathbf{Z}_N^*$ 和一个签名 $\sigma\in\mathbf{Z}_N^*$。当且仅当 $m=\sigma^e \bmod n$ 时输出 1，否则输出 0。

从上述的构造可以看出，教科书式 RSA 签名方案与教科书式 RSA 加密方案的计算过程十分相似，其正确性也很容易验证。但是这个方案是不安全的。

在数字签名(消息认证码)的安全定义中，允许攻击者进行适应性地选择消息，从而产生签名(标记)。基于这种操作，攻击者可以对任意的消息产生有效的签名。假设攻击者想要产生的签名的消息为 m。攻击者可以通过以下步骤产生对消息 m 的有效签名 σ。

(1) 攻击者随机产生m_1,m_2，使 $m_1m_2=m \bmod n$。

(2) 攻击者分别对m_1和m_2进行签名询问，得到签名σ_1和σ_2。

(3) 计算 $\sigma=\sigma_1\sigma_2 \bmod n$。

其中，σ 为消息 m 的有效签名

$$\sigma^e=(\sigma_1\sigma_2)^e=\sigma_1^e\sigma_2^e=m_1m_2=m$$

即 σ 可以通过对 m 的有效性验证。

从上述的攻击中可以发现，教科书式的 RSA 签名方案不安全的原因是不同消息的签名可以进行结合，从而产生一个有效的对另一个消息的签名。为了实现安全的 RSA 签名，打破消息和签名之间的关联性是关键。因此，改进的 RSA 签名方案的核心思想是对需要签名的消息进行哈希，然后对哈希的结果进行签名：

$$\sigma=H(m)^d \bmod n$$

由于哈希函数是公开的，因此任何人都可以重新计算消息的哈希值，并验证签名的有效性。由于哈希函数的单向性和强抗碰撞性，因此对于攻击者，找到 3 个消息 m,m_1,m_2，使 $H(m)=H(m_1)H(m_2)$是困难的。

上述的哈希后签名过程实际上是构造安全签名方案的常用方法。当然在实际使用

中，并不是简单地对消息进行哈希，而是会使用一些填充技术加入随机数或固定填充等方法对消息进行处理后，再进行哈希运算。这种方法在很多签名方案中除了可以打破消息和签名之间的关联性外，还可以使签名方案对任意长度或任意类型的消息进行签名，或使签名方案变成概率性方案。

2. ElGamal 签名方案

ElGamal 签名方案很少在实际中使用，但是它的一个变形方案作为数字签名标准算法(Digital Signature Algorithm，DSA)被广泛使用。下面先介绍 ElGamal 签名方案，再介绍 DSA 签名方案。

假设 p 是一个大素数，使离散对数问题在 $\mathbf{Z}_p^*$ 上是困难的，g 是 $\mathbf{Z}_p^*$ 中的一个随机生成元，H 是一个密码学哈希函数 $H:\{0,1\}^* \rightarrow \mathbf{Z}_p$。则 ElGamal 签名方案的构造如下。

(1) Gen(λ)：算法输入安全参数 λ，然后随机选取一个长度为 λ 的大素数 p。随机选取一个生成元 $g \in \mathbf{Z}_p^*$。随机选取 $x \in \mathbf{Z}_p$，计算 $y = g^x \bmod p$。ElGamal 签名方案的签名密钥为 $\mathrm{sk}=(x,g,p)$，验证密钥为 $\mathrm{vk}=(y,g,p)$。

(2) Sign(sk，m)：算法输入签名密钥 $\mathrm{sk}=(x,g,p)$和消息 $m \in \{0,1\}^*$，随机从[1，$p-2$]区间中选取 k，使 $\gcd(k,p-1)=1$。计算 $r = g^k \bmod p$，并计算 $s=(H(m)-xr)k^{-1} \bmod p-1$。如果 $s=0$，则重新随机产生 k 并计算 s，直至 s 不等于 0。算法输出 $\sigma=(r,s)$。

(3) Verify(vk，m，σ)：算法输入签名验证密钥 $\mathrm{vk}=(y,g,p)$，消息 $m \in \{0,1\}^*$ 和一个签名 $\sigma=(r,s)$，当且仅当 $g^{H(m)} = r^s y^r \bmod p$ 时，算法输出 1，否则输出 0。

ElGamal 签名算法的正确性：在签名过程中，$H(m)=sk+xr \bmod p-1$。因此有

$$\begin{aligned} g^{H(m)} &= g^{sk+xr} \\ &= g^{sk}\, g^{xr} \\ &= (g^k)^s (g^x)^r \\ &= r^s y^r \bmod p \end{aligned}$$

则可证明 ElGamal 签名算法的正确性。

3. Schnorr 签名方案

DSA 签名方案同时借鉴了 ElGamal 签名方案和 Schnorr 签名方案。因此，在正式介绍 DSA 签名方案之前，还要介绍 Schnorr 签名方案。Schnorr 签名方案是 Claus-Peter Schnorr 在 1989 年提出的。与 ElGamal 签名方案基于素数阶整数群不同，Schnorr 签名方案是基于任意素数阶循环群的(仍然要求离散对数问题在该群上是安全的)。这就允许 Schnorr 签名方案在长度更小的椭圆曲线群上实现。

假设 G 是一个阶为 p 的循环群，其中 p 是一个大素数，使离散对数问题在 G 上是困难的。令 H 是一个强抗碰撞性的哈希函数 $H:\{0,1\}^* \rightarrow \mathbf{Z}_p^*$。则 Schnorr 签名方案的构造如下。

(1) Gen(λ)：算法输入安全参数 λ，产生满足上述条件的群 G。随机选取一个生成元 $g \in G$。随机选取 $x \in \mathbf{Z}_p^*$，计算 $y = g^x$。Schnorr 签名方案的签名密钥为 $\mathrm{sk}=x$，验证密