

Chapter 12 数字签名方案

12.1 数字签名 - 概述

在上一章中，我们探讨了如何在公钥设置中使用公钥加密来实现**机密性**。公钥设置中的**完整性**（或**认证性**）是通过**数字签名方案**提供的。这些可以被视为消息认证码的**公钥模拟**，尽管，正如我们将看到的，这些原语之间存在几个重要的区别。签名方案允许一个**签名者** S 建立一个公钥 pk 来“签署”一个消息 m ，使用的相关联的私钥 sk ，其方式是任何知道 pk （并且知道该公钥是由 S 建立的）的人都可以**验证**该消息源自 S 并且在传输过程中未被修改。（请注意，与公钥加密相反，在数字签名的上下文中，公钥的拥有者充当**发送方**）。作为一个典型的应用，考虑一家软件公司想要**以认证的方式**传播软件更新；也就是说，当公司发布更新时，其任何客户端都应该能够验证该更新是**真实的**，并且恶意的第三方绝不应该能够欺骗客户端接受一个并非由该公司实际发布的更新。为此，公司可以生成一个公钥 pk 和一个私钥 sk ，然后以某种可靠的方式向其客户端分发 pk ，同时保守 sk 的秘密。（与公钥加密的情况一样，我们假设公钥的初始分发已正确执行，因此所有客户端都有一个正确的 pk 副本。在当前示例中， pk 可以与客户端购买的原始软件捆绑在一起。）当发布软件更新 m 时，公司使用其私钥 sk 计算 m 的数字签名 σ ，并将 (m, σ) 发送给每个客户端。每个客户端都可以通过检查 σ 是否是相对于公钥 pk 的 m 的正确签名来验证 m 的真实性。恶意方可能会尝试通过向客户端发送 (m', σ') 来发布欺诈性更新，其中 m' 代表公司从未发布的更新。这个 m' 可能是某个先前更新的修改版本，或者它可能是完全新的且与任何先前更新无关。如果签名方案是“安全”的（在某种意义上我们很快会更仔细地定义），那么当客户端尝试验证 σ' 时，它会发现这是一个相对于 pk 的 m' 的**无效**签名，因此会**拒绝**该签名。即使 m' 只是对真实更新 m 稍作修改，客户端也会拒绝。上述内容不仅仅是数字签名的一个理论应用，它还是当今广泛用于分发软件更新的应用。

与消息认证码的比较

消息认证码和数字签名方案都用于确保传输消息的**完整性**。尽管第 10 章中比较公钥和私钥设置的讨论主要集中在加密上，但该讨论也适用于消息完整性。使用数字签名而不是消息认证码可以简化密钥分发和管理，尤其是在发件人需要与多个接收者通信时，如上面的软件更新示例所示。通过使用数字签名方案，发件人避免了与每个潜在接收者建立一个不同的密钥，并避免了对每个这样的密钥计算一个单独的 MAC 标签。相反，发件人只需要计算一个可以被所有接收者验证的**单个**签名。与消息认证码相比，数字签名有一个**定性**优势，即**签名是可公开验证的**。这意味着如果接收者验证了给定消息上的签名是合法的，那么所有其他收到此签名消息的各方也会验证其合法性。如果签名者与每个接收者共享一个单独的密钥，则消息认证码无法实现此功能：在这种设置中，恶意发送者可能会计算一个与其与接收者 A 共享的密钥相关的正确 MAC 标签，但计算一个与其与不同用户 B 共享的密钥相关的不正确 MAC 标签。在这种情况下， A

知道他收到了发件人的真实消息，但无法保证 B 会同意。公开可验证性意味着签名是**可转移的**：签名者 S 对消息 m 的签名 σ 可以展示给第三方，第三方随后可以自行验证 σ 是相对于 S 的公钥（这里，我们假设该第三方也知道 S 的公钥）的 m 的合法签名。通过制作签名的副本，该第三方可以将签名展示给另一方，并说服他们 S 确实认证了 m ，依此类推。公开可验证性和可转移性对于数字签名在**证书和公钥基础设施**中的应用至关重要，我们将在第 12.7 节中讨论。

数字签名方案还提供了一个非常重要的属性，即**不可否认性**。这意味着一旦 S 签署了一条消息，他以后就不能否认这样做（假设 S 的公钥被广泛宣传和分发）。数字签名的这一方面对于法律应用至关重要，在这些应用中，接收者可能需要向第三方（例如法官）证明签名者确实“证明”了特定的消息（例如合同）：假设法官知道 S 的公钥或以其他方式公开可用，消息上的有效签名可作为令人信服的证据，证明 S 确实签署了此消息。消息认证码根本无法提供不可否认性。要理解这一点，假设用户 S 和 R 共享一个密钥 K_{SR} ， S 发送消息 m 给 R ，以及使用该密钥计算的一个（有效）MAC 标签 t 。由于法官不知道 K_{SR} （事实上，该密钥由 S 和 R 保密），法官无法确定 t 是否有效。如果 R 向法官泄露密钥 K_{SR} ，法官也无法知道这是否是 S 和 R 共享的“实际”密钥，或者它是否是 R 制造的某个“假”密钥。最后，即使我们假设法官可以通过某种方式获取双方共享的实际密钥 K_{SR} ，法官也无法区分 t 是由 S 生成的还是由 R 生成的——这源于消息认证码是对称密钥原语的事实；任何 S 可以做的事情， R 也可以做。

与私钥与公钥加密的情况一样，消息认证码的优势是更短，并且生成/验证效率比数字签名高大约 2-3 个数量级。因此，在不需要公开可验证性、可转移性和/或不可否认性，并且发件人主要与单个收件人通信（发件人能够与其共享一个密钥）的情况下，应使用消息认证码。

与公钥加密的关系

数字签名通常被错误地视为公钥加密的“逆”，其发送方和接收方的角色互换。历史上，事实上，有人建议可以通过“反转”公钥加密来获得数字签名，即，通过（使用私钥）**解密**消息 m 来获得 σ ，并通过（使用相应的公钥）**加密** σ 并检查结果是否为 m 来验证签名 σ 。以这种方式构造签名方案的建议是**完全没有根据的**：在大多数情况下，它根本不适用，在适用的情况下，它会导致**不安全**的签名方案。

12.2 定义

数字签名是消息认证码的公钥对应物，它们的语法和安全保证是相似的。发送方应用于消息的算法在这里表示为 Sign（而不是 Mac），并且该算法的输出现在称为**签名**（而不是标签）。接收者应用于消息和签名的算法仍表示为 Vrfy，以便验证有效性。

定义 12.1 一个（数字）签名方案由三个概率多项式时间算法 $(\text{Gen}, \text{Sign}, \text{Vrfy})$ 组成，满足：

1. **密钥生成算法** Gen 将安全参数 1^n 作为输入，并输出一对密钥 (pk, sk) 。这些分别称为**公钥**和**私钥**。我们假设 pk 和 sk 各自的长度至少为 n ，并且 n 可以从 pk 或 sk 确定。
2. **签名算法** Sign 将私钥 sk 和来自某个消息空间（可能取决于 pk ）的消息 m 作为输入。它输出一个签名 σ ，我们将其记为 $\sigma \leftarrow \text{Sign}_{sk}(m)$ 。
3. **确定性验证算法** Vrfy 将公钥 pk 、消息 m 和签名 σ 作为输入。它输出一个比特 b ，其中 $b = 1$ 表示**有效**， $b = 0$ 表示**无效**。我们将其记为 $b := \text{Vrfy}_{pk}(m, \sigma)$ 。

要求，除了由 $\text{Gen}(1^n)$ 输出的 (pk, sk) 具有可忽略的概率之外，对于每个（合法）消息 m ，都满足 $\text{Vrfy}_{pk}(m, \text{Sign}_{sk}(m)) = 1$ 。如果存在一个函数 l 使得对于由 $\text{Gen}(1^n)$ 输出的每个 (pk, sk) ，消息空间是 $\{0, 1\}^{l(n)}$ ，那么我们说 $(\text{Gen}, \text{Sign}, \text{Vrfy})$ 是长度为 $l(n)$ 的消息的签名方案。

如果 $\text{Vrfy}_{pk}(m, \sigma) = 1$ ，我们称 σ 是消息 m 上的**有效签名**（相对于上下文中理解的某个公钥 pk ）。签名方案的使用方式如下。一方 S ，充当**发送方**，运行 $\text{Gen}(1^n)$ 以获取密钥 (pk, sk) 。然后，公钥 pk 被公开为属于 S ；例如， S 可以将公钥放在其网页上或放置在某个公共目录中。与公钥加密的情况一样，我们假设任何其他方都能够获得 S 公钥的合法副本（见下文讨论）。当 S 想要认证消息 m 时，它计算签名 $\sigma \leftarrow \text{Sign}_{sk}(m)$ 并发送 (m, σ) 。接收到 (m, σ) 后，知道 pk 的接收者可以通过检查 $\text{Vrfy}_{pk}(m, \sigma) \stackrel{?}{=} 1$ 来验证 m 的真实性。这既确立了 S 发送了 m ，也确立了 m 在传输过程中未被修改。然而，与消息验证码的情况一样，它没有说明 m 是何时发送的，重放攻击仍然是可能的（见第 4.2 节）。

假定各方能够获得 S 公钥的合法副本，意味着 S 能够以可靠和认证的方式传输**至少一条消息**（即 pk 本身）。然而，如果我们可以可靠地传输消息，那么我们为什么仍然需要签名方案呢？答案是 pk 的可靠分发是一项困难且昂贵的任务，但是使用签名方案意味着**只需要执行一次**，之后就可以可靠地发送**无限数量**的消息。此外，正如我们将在第 12.7 节中讨论的那样，签名方案本身被用作确保其他公钥可靠分发的中央工具。因此，它们是建立“公钥基础设施”以解决密钥分发问题的核心工具。

签名方案的安全性。 对于由签名者 S 生成的固定公钥 pk ，**伪造**是消息 m 以及有效签名 σ ，其中 m 之前未被 S 签署过。签名方案的安全性意味着，即使攻击者获得了它选择的许多其他消息的签名，攻击者也应该无法输出伪造。这与消息验证码的安全定义直接类似，我们请读者参阅第 4.2 节以获取动机和进一步讨论。现在我们提出安全性的正式定义，它本质上与定义 4.2 相同。令 $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$ 为一个签名方案，并考虑以下针对敌手 \mathcal{A} 和参数 n 的实验：

签名实验 $\text{Sig-forge}_{\mathcal{A}, \Pi}(n)$:

1. 运行 $\text{Gen}(1^n)$ 以获得密钥 (pk, sk) 。
2. 敌手 \mathcal{A} 被给予 pk 并可以访问谕示 $\text{Sign}_{sk}(\cdot)$ 。敌手随后输出 (m, σ) 。令 \mathcal{Q} 表示 \mathcal{A} 询问其谕示的所有查询的集合。
3. 当且仅当 (1) $\text{Vrfy}_{pk}(m, \sigma) = 1$ 且 (2) $m \notin \mathcal{Q}$ 时, \mathcal{A} 成功。在这种情况下, 实验的输出定义为 1。

定义 12.2 如果对于所有概率多项式时间敌手 \mathcal{A} , 存在一个**可忽略的函数** negl 使得:

$$\Pr[\text{Sig-forge}_{\mathcal{A}, \Pi}(n) = 1] \leq \text{negl}(n)$$

那么签名方案 $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$ 在**适应性选择消息攻击下是存在不可伪造的**, 或简称**安全的**。

12.3 哈希-然后-签名范例

与公钥与私钥加密的情况一样, “原生”签名方案比消息认证码的效率低几个数量级。幸运的是, 与混合加密一样 (见第 11.3 节), 至少对于足够长的消息, 可以以私钥操作的渐近成本获得数字签名的功能。这可以通过接下来讨论的**哈希-然后-签名方法**来实现。哈希-然后-签名方法背后的直觉很简单。假设我们有一个长度为 l 的消息的签名方案 Π , 并且希望签署一个 (更长) 的消息 $m \in \{0, 1\}^*$ 。与其签署 m 本身, 不如使用一个哈希函数 H 将消息哈希为一个固定长度 l 的**摘要**, 然后签署生成的摘要。这种方法与第 5.3.1 节中讨论的哈希-然后-MAC 方法完全类似。

构造 12.3 令 $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$ 为长度为 $l(n)$ 的消息的签名方案, 并令 $\Pi_H = (\text{Gen}_H, H)$ 为输出长度为 $l(n)$ 的哈希函数。构造签名方案 $\Pi' = (\text{Gen}', \text{Sign}', \text{Vrfy}')$ 如下:

- Gen' : 输入 1^n , 运行 $\text{Gen}(1^n)$ 获得 (pk, sk) , 并运行 $\text{Gen}_H(1^n)$ 获得 s ; 公钥是 (pk, s) , 私钥是 (sk, s) 。
- Sign' : 输入私钥 (sk, s) 和消息 $m \in \{0, 1\}^*$, 输出 $\sigma \leftarrow \text{Sign}_{sk}(H^s(m))$ 。
- Vrfy' : 输入公钥 (pk, s) 、消息 $m \in \{0, 1\}^*$ 和签名 σ , 当且仅当 $\text{Vrfy}_{pk}(H^s(m), \sigma) = 1$ 时输出 1。

哈希-然后-签名范例。

定理 12.4 如果 Π 是长度为 l 的消息的安全签名方案, 并且 Π_H 是**抗碰撞的**, 那么构造 12.3

是一个安全签名方案（适用于任意长度的消息）。

该定理的证明与定理 5.6 的证明几乎相同。

12.4 RSA 签名

我们开始考虑具体的签名方案，并讨论基于 RSA 假设的方案。

12.4.1 朴素 RSA

我们首先描述一个简单、基于 RSA 的签名方案。尽管该方案**不安全**，但它可作为一个有用的起点。

像往常一样，令 GenRSA 是一个 PPT 算法，输入 1^n ，输出一个模数 N ，它是两个 n 比特素数的乘积（除了可忽略的概率之外），以及满足 $ed = 1 \pmod{\phi(N)}$ 的整数 e, d 。朴素 RSA 中的密钥生成仅涉及运行 GenRSA，并输出 (N, e) 作为公钥， (N, d) 作为私钥。要签署消息 $m \in \mathbb{Z}_N^*$ ，签署者计算 $\sigma := [m^d \pmod N]$ 。相对于公钥 (N, e) 对消息 m 的签名 σ 的验证是通过检查 $m \stackrel{?}{=} \sigma^e \pmod N$ 来执行的。见构造 12.5。

构造 12.5 令 GenRSA 如正文所述。定义一个签名方案如下：

- Gen：输入 1^n ，运行 $\text{GenRSA}(1^n)$ 获得 (N, e, d) 。公钥是 (N, e) ，私钥是 (N, d) 。
- Sign：输入私钥 $sk = (N, d)$ 和消息 $m \in \mathbb{Z}_N^*$ ，计算签名

$$\sigma := [m^d \pmod N]$$

- Vrfy：输入公钥 $pk = (N, e)$ 、消息 $m \in \mathbb{Z}_N^*$ 和签名 $\sigma \in \mathbb{Z}_N^*$ ，当且仅当 $m \stackrel{?}{=} [\sigma^e \pmod N]$ 时输出 1。

朴素 RSA 签名方案。

很容易看出，合法生成的签名总是成功的，因为

$$\sigma^e = (m^d)^e = m^{[ed \pmod{\phi(N)}]} = m^1 = m \pmod N$$

人们可能期望这个方案是安全的，因为对于只知道公钥 (N, e) 的敌手来说，计算消息 m 上的有效签名似乎需要解决 RSA 问题（因为签名正好是 m 的 e 次根）。不幸的是，这种推理是**不正确的**。首先，RSA 假设只意味着计算**均匀**消息 m 的签名（即计算 e 次根）是困难的；它没有说明计算**非均匀** m 的签名或攻击者选择的**某些**消息 m 的签名是困难的。此外，RSA 假设没有说明一旦攻击者知道其他消息的签名后，他可能能够做什么。以下示例证明了这两个观察

结果都导致了对朴素 RSA 签名方案的攻击。

无消息攻击。 我们描述的第一个攻击仅使用公钥，无需从合法签名者那里获得任何签名，即可生成伪造。攻击的工作原理如下：给定公钥 $pk = (N, e)$ ，选择一个均匀的 $\sigma \in \mathbb{Z}_N^*$ 并计算 $m := [\sigma^e \bmod N]$ 。然后输出伪造 (m, σ) 。很明显 σ 是 m 上的有效签名，这是一个伪造，因为公钥的拥有者根本没有发布任何签名。我们得出结论，朴素 RSA 签名方案不满足定义 12.2。有人可能会争辩说，这不构成一个“现实的”攻击，因为敌手对它伪造有效签名的消息 m “没有控制权”。就定义 12.2 而言，这无关紧要，而且我们已经讨论过（在第 4 章中），假定将使用任何密码方案进行认证的消息具有任何语义是危险的。此外，敌手确实对 m 有一些控制权：例如，通过选择多个均匀的 σ 值，他可以（以高概率）获得一个具有某些所需设置比特的 m 。通过以某种特定的方式选择 σ ，也可能影响最终输出伪造的消息。

伪造任意消息上的签名。 对朴素 RSA 签名方案更具破坏性的攻击要求敌手从签名者那里获得两个签名，但允许敌手输出它选择的**任何消息上的伪造签名**。假设敌手想要伪造消息 $m \in \mathbb{Z}_N^*$ 上的签名，相对于公钥 $pk = (N, e)$ 。敌手选择不同于 m 的任意 $m_1, m_2 \in \mathbb{Z}_N^*$ ，使得 $m = m_1 \cdot m_2 \bmod N$ 。然后它分别获得 m_1, m_2 上的签名 σ_1, σ_2 。最后，它输出 $\sigma := [\sigma_1 \cdot \sigma_2 \bmod N]$ 作为 m 上的有效签名。之所以有效，是因为

$$\sigma^e = (\sigma_1 \cdot \sigma_2)^e = (\sigma_1^e \cdot \sigma_2^e) = (m_1^d \cdot m_2^d)^e = m_1^{ed} \cdot m_2^{ed} = m_1^1 \cdot m_2^1 = m_1 \cdot m_2 = m \bmod N$$

利用了 σ_1, σ_2 是 m_1, m_2 上的有效签名这一事实。能够伪造任意消息上的签名是具有毁灭性的。然而，有人可能会争辩说，这种攻击是不现实的，因为敌手将无法说服签名者签署精确的消息 m_1 和 m_2 。再次强调，就定义 12.2 而言，这无关紧要。此外，对签名者可能愿意或不愿意签署的消息做出假设是危险的。例如，客户端可以使用签名方案通过签署服务器发送的随机挑战来向服务器进行认证。在这里，恶意服务器将能够获得它选择的任何消息的签名。更一般地，攻击者可能能够选择 m_1 和 m_2 作为签名者将同意签署的“合法”消息。最后，请注意，攻击可以推广：如果敌手获得了 q 个任意消息 $\mathcal{M} = \{m_1, \dots, m_q\}$ 上的有效签名，那么敌手可以输出通过对 \mathcal{M} 的子集（大小不同于 1）进行乘积而获得的任何 $2^q - q$ 个其他消息上的有效签名。

12.4.2 RSA-FDH 和 PKCS #1 v2.1

可以尝试通过在签名消息之前对其应用某种转换来防止上一节中的攻击。也就是说，签名者现在将指定一个具有某些密码属性的（确定性）函数 H 作为其公钥的一部分，将消息映射到 \mathbb{Z}_N^* ；消息 m 上的签名将是 $\sigma := [H(m)^d \bmod N]$ ，并且对消息 m 的签名 σ 的验证将通过检查 $\sigma^e \stackrel{?}{=} H(m) \bmod N$ 来完成。见构造 12.6。

构造 12.6 令 GenRSA 如前几节所述，并构造一个签名方案如下：

- Gen: 输入 1^n , 运行 $\text{GenRSA}(1^n)$ 计算 (N, e, d) 。公钥是 (N, e) , 私钥是 (N, d) 。
作为密钥生成的一部分, 指定了一个函数 $H : \{0, 1\}^* \rightarrow \mathbb{Z}_N^*$, 但我们在此省略。
- Sign: 输入私钥 (N, d) 和消息 $m \in \{0, 1\}^*$, 计算

$$\sigma := [H(m)^d \bmod N]$$

- Vrfy: 输入公钥 (N, e) 、消息 m 和签名 σ , 当且仅当 $\sigma^e \stackrel{?}{=} H(m) \bmod N$ 时输出 1。

RSA-FDH 签名方案。

为了使这种构造安全, 函数 H 需要具备哪些属性? 至少, 为了防止无消息攻击, 攻击者应该无法从 σ 开始, 计算 $m := [\sigma^e \bmod N]$, 然后找到一个消息 m 使得 $H(m) = m$ 。特别是, 这意味着 H 在某种意义上应该**难以反转**。为了防止第二次攻击, 我们需要一个**不具有“乘法关系”的 H** , 即, 难以找到三个消息 m, m_1, m_2 使得 $H(m) = H(m_1) \cdot H(m_2) \bmod N$ 。最后, 在 H 中必须**难以找到碰撞**: 如果 $H(m_1) = H(m_2)$, 则 m_1 和 m_2 具有相同的签名, 伪造变得微不足道。

没有已知的方法可以选择 H 以便证明构造 12.6 是安全的。然而, 如果 H 被建模为将输入均匀映射到 \mathbb{Z}_N^* 的**随机谕示**, 则可以证明安全性; 在这种情况下, 该方案被称为 **RSA 全域哈希 (RSA-FDH)** 签名方案。可以检查这种随机函数满足上一段中讨论的要求: 随机函数 (具有大范围) 难以反转, 没有任何容易找到的乘法关系, 并且是抗碰撞的。当然, 这种非正式的推理并不能排除所有可能的攻击, 但下面的安全性证明可以。

在进行正式证明之前, 我们提供一些直觉。我们的目标是证明, 如果相对于 GenRSA 而言 RSA 问题是困难的, 那么当 H 被建模为随机谕示时, RSA-FDH 是安全的。我们首先考虑针对无消息攻击的安全性, 即, 当敌手 \mathcal{A} 不能请求任何签名时。在这里, 敌手仅限于对随机谕示进行查询, 并且我们假设不失一般性, 如果敌手输出伪造 (m, σ) , 则它之前已经查询了 m 到 H , \mathcal{A} 总是对 H 进行恰好 q 次 (不同的) 查询。

假设存在一个高效的敌手 \mathcal{A} , 它执行无消息攻击并对 H 进行恰好 q 次查询。我们构造一个解决相对于 GenRSA 的 RSA 问题的高效算法 \mathcal{A}' 。给定输入 (N, e, y) , 算法 \mathcal{A}' 在公钥 $pk = (N, e)$ 上运行 \mathcal{A} 。令 m_1, \dots, m_q 表示 \mathcal{A} 对 H 进行的 q 次 (不同的) 查询。我们的算法 \mathcal{A}' 用 \mathbb{Z}_N^* 的均匀元素回答 \mathcal{A} 的这些随机谕示查询, 除了一个查询 (比如第 i 个查询, 从 \mathcal{A} 的谕示查询中均匀选择) —该查询用 y 本身回答。请注意, 从 \mathcal{A} 的角度来看, 其所有随机谕示查询都用 \mathbb{Z}_N^* 的均匀元素回答 (回想一下 y 也是均匀的, 尽管它不是由 \mathcal{A}' 选择的), 因此 \mathcal{A} 对 i 没有任何信息。此外, 当作为 \mathcal{A}' 的子程序运行时, \mathcal{A} 的视图与攻击原始签名方案时 \mathcal{A} 的视图具有相同的分布。

如果 \mathcal{A} 输出伪造 (m, σ) , 那么因为 $m \in \{m_1, \dots, m_q\}$, 以概率 $1/q$ 我们将有 $m = m_i$ 。在这种情况下,

$$\sigma^e = H(m) = H(m_i) = y \bmod N$$

\mathcal{A}' 可以输出 σ 作为其给定 RSA 实例 (N, e, y) 的解。我们得出结论, 如果 \mathcal{A} 以概率 ϵ 输出伪造, 那么 \mathcal{A}' 以概率 ϵ/q 解决 RSA 问题。由于 q 是多项式的, 我们得出结论, 如果相对于 GenRSA 而言 RSA 问题是困难的, 则 ϵ 必须是可忽略的。

处理允许敌手请求它选择的消息的签名的情况更加困难。复杂性在于我们的算法 \mathcal{A}' 不知道解密指数 d , 但现在必须计算 \mathcal{A} 向其签名谕示查询的消息的有效签名。这似乎是不可能的 (甚至可能是矛盾的!), 直到我们意识到 \mathcal{A}' 只要将 $H(m)$ 设置为等于已知值 σ 的 $[\sigma^e \bmod N]$, 就可以正确计算消息 m 上的签名。(这里我们使用随机谕示是“可编程”的事实。) 如果 σ 是均匀的, 那么 $[\sigma^e \bmod N]$ 也是均匀的, 因此随机谕示仍然由 \mathcal{A}' “正确”地模拟。上述直觉在以下证明中得到形式化:

定理 12.7 如果相对于 GenRSA 而言 RSA 问题是困难的, 并且 H 被建模为随机谕示, 那么构造 12.6 是安全的。

证明 令 $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$ 表示构造 12.6, 并令 \mathcal{A} 是一个概率多项式时间敌手。我们假设不失一般性, 如果 \mathcal{A} 请求消息 m 上的签名, 或输出伪造 (m, σ) , 那么它之前已经查询了 m 到 H 。令 $q(n)$ 是 \mathcal{A} 对安全参数 n 对 H 进行的查询次数的多项式上限; 我们假设不失一般性, \mathcal{A} 对 H 进行恰好 $q(n)$ 次不同的查询。

为方便起见, 我们列出实验 $\text{Sig-forge}_{\mathcal{A}, \Pi}(n)$ 的步骤:

1. 运行 $\text{GenRSA}(1^n)$ 获得 (N, e, d) 。选择一个随机函数 $H : \{0, 1\}^* \rightarrow \mathbb{Z}_N^*$ 。
2. 敌手 \mathcal{A} 被给予 $pk = (N, e)$, 并且可以查询 H 以及签名谕示 $\text{Sign}_{(N, d)}(\cdot)$, 该谕示在输入消息 m 时, 返回 $\sigma := [H(m)^d \bmod N]$ 。
3. \mathcal{A} 输出 (m, σ) , 其中它之前没有请求过 m 上的签名。当且仅当 $\sigma^e = H(m) \bmod N$ 时, 实验的输出为 1。

我们定义一个修改后的实验 $\text{Sig-forge}'_{\mathcal{A}, \Pi}(n)$, 在其中我们预先猜测 \mathcal{A} 最终输出的伪造 (如果有) 将对应于哪个消息 (来自 \mathcal{A} 查询 H 的 q 个消息中) :

1. 均匀选择 $j \in \{1, \dots, q\}$ 。
2. 运行 $\text{GenRSA}(1^n)$ 获得 (N, e, d) 。选择一个随机函数 $H : \{0, 1\}^* \rightarrow \mathbb{Z}_N^*$ 。
3. 敌手 \mathcal{A} 被给予 $pk = (N, e)$, 并且可以查询 H 以及签名谕示 $\text{Sign}_{(N, d)}(\cdot)$, 该谕示在输

入消息 m 时, 返回 $\sigma := [H(m)^d \bmod N]$ 。

4. \mathcal{A} 输出 (m, σ) , 其中它之前没有请求过 m 上的签名。令 i 使得 $m = m_i$ 。² 当且仅当 $\sigma^e = H(m) \bmod N$ 且 $j = i$ 时, 实验的输出为 1。

由于 j 是均匀的且独立于其他一切, 因此 $j = i$ 的概率 (即使以 \mathcal{A} 输出伪造的事件为条件) 正好是 $1/q$ 。因此 $\Pr[\text{Sig-forge}'_{\mathcal{A}, \Pi}(n) = 1] = \frac{1}{q(n)} \cdot \Pr[\text{Sig-forge}_{\mathcal{A}, \Pi}(n) = 1]$ 。

现在考虑修改后的实验 $\text{Sig-forge}''_{\mathcal{A}, \Pi}(n)$, 其中如果 \mathcal{A} 曾经请求消息 m_j (其中 m_j 表示 \mathcal{A} 对 H 查询的第 j 个消息, j 是预先选择的均匀值) 上的签名, 则实验中止。这不会改变实验输出 1 的概率, 因为如果 \mathcal{A} 曾经请求 m_j 上的签名, 那么它不可能输出 m_j 上的伪造。用文字表示,

$$\Pr[\text{Sig-forge}''_{\mathcal{A}, \Pi}(n) = 1] = \Pr[\text{Sig-forge}'_{\mathcal{A}, \Pi}(n) = 1] = \frac{\Pr[\text{Sig-forge}_{\mathcal{A}, \Pi}(n) = 1]}{q(n)} \quad (12.1)$$

最后, 考虑以下算法 \mathcal{A}' 解决 RSA 问题:

算法 \mathcal{A}' : 算法被给予 (N, e, y) 作为输入。

1. 均匀选择 $j \in \{1, \dots, q\}$ 。
2. 在输入公钥 $pk = (N, e)$ 上运行 \mathcal{A} 。在一个表中存储三元组 (\cdot, \cdot, \cdot) , 该表最初为空。一个条目 (m_i, σ_i, y_i) 表示 \mathcal{A}' 已设置 $H(m_i) = y_i$, 并且 $\sigma_i^e = y_i \bmod N$ 。
3. 当 \mathcal{A} 进行其第 i 个随机谕示查询 $H(m_i)$ 时, 回答如下:
 - 如果 $i = j$, 返回 y 作为查询的答案。
 - 否则, 选择均匀的 $\sigma_i \in \mathbb{Z}_N^*$, 计算 $y_i := [\sigma_i^e \bmod N]$, 返回 y_i 作为查询的答案, 并在表中存储 (m_i, σ_i, y_i) 。当 \mathcal{A} 请求消息 m 上的签名时, 令 i 使得 $m = m_i$ 并回答查询如下³
 - 如果 $i = j$ 则 \mathcal{A}' 中止。
 - 如果 $i \neq j$ 则表中存在条目 (m_i, σ_i, y_i) 。返回 σ_i 作为查询的答案。
4. 在 \mathcal{A} 执行结束时, 它输出 (m, σ) 。如果 $m = m_j$ 且 $\sigma^e = y \bmod N$, 则输出 σ 。

显然, \mathcal{A}' 在概率多项式时间内运行。假设 \mathcal{A}' 的输入 (N, e, y) 是通过运行 $\text{GenRSA}(1^n)$ 获得 (N, e, d) , 然后选择均匀的 $y \in \mathbb{Z}_N^*$ 生成的。关键的观察是, 当 \mathcal{A} 作为 \mathcal{A}' 的子程序运行时, \mathcal{A} 的视图与 \mathcal{A} 在实验 $\text{Sig-forge}''_{\mathcal{A}, \Pi}(n)$ 中的视图完全相同。特别是, 所有签名谕示查询都得到正确回答, 并且当 \mathcal{A} 作为 \mathcal{A}' 的子程序运行时, \mathcal{A} 的每个随机谕示查询都用 \mathbb{Z}_N^* 的均匀元素回答:

- 查询 $H(m_j)$ 用 y 回答, 它是 \mathbb{Z}_N^* 的均匀元素。

- 查询 $H(m_i)$, $i \neq j$, 用 $y_i = [\sigma_i^e \bmod N]$ 回答, 其中 σ_i 在 \mathbb{Z}_N^* 中是均匀的。由于求 e 次幂是一个一对一函数, y_i 也是均匀分布的。

最后, 观察到每当实验 $\text{Sig-forge}_{\mathcal{A}, \Pi}''(n)$ 输出 1 时, \mathcal{A}' 就会输出其给定 RSA 实例的正确解。这是因为 $\text{Sig-forge}_{\mathcal{A}, \Pi}''(n) = 1$ 意味着 $j = i$ 且 $\sigma^e = H(m_i) \bmod N$ 。现在, 当 $j = i$ 时, 算法 \mathcal{A}' 不会中止, 而且 $H(m_i) = y$ 。因此 $\sigma^e = H(m_i) = y \bmod N$, 所以 σ 是所需的逆。使用等式 (12.1), 这意味着

$$\Pr[\text{RSA-inv}_{\mathcal{A}', \text{GenRSA}}(n) = 1] = \Pr[\text{Sig-forge}_{\mathcal{A}, \Pi}''(n) = 1] = \frac{\Pr[\text{Sig-forge}_{\mathcal{A}, \Pi}(n) = 1]}{q(n)} \quad (12.2)$$

如果相对于 GenRSA 而言 RSA 问题是困难的, 则存在一个可忽略的函数 negl 使得 $\Pr[\text{RSA-inv}_{\mathcal{A}', \text{GenRSA}}(n) = 1] \leq \text{negl}(n)$ 。由于 $q(n)$ 是多项式, 我们从等式 (12.2) 得出结论 $\Pr[\text{Sig-forge}_{\mathcal{A}, \Pi}(n) = 1]$ 也是可忽略的。这就完成了证明。

RSA PKCS #1 v2.1。 RSA PKCS #1 v2.1 标准包含一个签名方案, 可以视为 RSA-FDH 的变体。更准确地说, 该标准定义了一个方案, 其中消息上的签名取决于签名者在签名生成时选择的**盐** (即随机值)。如果签名者将此盐固定为 NULL——这是标准允许的——则得到的方案与 RSA-FDH 非常相似。 H 的范围接近 \mathbb{Z}_N^* 是至关重要的; 特别是, 简单地让 H 是一个“现成”的密码哈希函数, 例如 SHA-1, 是不够的。(SHA-1 的输出长度比实践中使用的 RSA 模数的长度小得多。) 事实上, 如果 H 的输出长度太小 (例如, 如果直接使用 SHA-1, 输出长度为 160 比特), 则已知对构造 12.6 的实际攻击。在 PKCS #1 v2.1 签名方案中, H 是通过重复应用底层密码哈希函数来构造的。

12.5 基于离散对数问题的签名

签名方案也可以基于**离散对数假设**, 尽管该假设不像 RSA 假设那样容易用于签名。在第 12.5.1 节中, 我们描述了 Claus Schnorr 引入的签名方案, 该方案可以在随机谕示模型中被证明是安全的。在第 12.5.2 节中, 我们描述了广泛使用的 DSA 和 ECDSA 签名方案。

12.5.1 Schnorr 签名方案

Schnorr 签名方案背后的基本直觉最好通过稍微绕道讨论 (公钥) **身份识别方案** 来解释。然后我们描述了可用于将身份识别方案转换为随机谕示模型中的签名方案的 **Fiat-Shamir 变换**。最后, 我们介绍了基于离散对数问题的 Schnorr 身份识别方案及其相应的签名方案。

身份识别方案

身份识别方案是一种交互式协议, 允许一方证明其身份 (即, **认证自己**) 给另一方。这是一个

非常自然的概念，现在通常在登录网站时进行身份认证。我们将进行身份识别的一方（例如用户）称为“证明者”，将验证身份的一方（例如网络服务器）称为“验证者”。在这里，我们对**公钥设置**感兴趣，在这种设置中，证明者和验证者事先不共享任何秘密信息（例如密码）；相反，验证者只知道证明者的公钥。身份识别协议的成功执行使验证者相信它正在与预期的证明者而不是冒名顶替者进行通信。

我们将只考虑**特定形式**的三轮身份识别协议，其中证明者由两个算法 $\mathcal{P}_1, \mathcal{P}_2$ 指定，验证者由算法 \mathcal{V} 指定。证明者使用其私钥 sk 运行 $\mathcal{P}_1(sk)$ 以获得初始消息 I 以及一些状态 st ，并通过发送 I 来启动协议给验证者。作为响应，验证者发送一个挑战 r ，该挑战从由证明者的公钥 pk 定义的某个集合 Ω_{pk} 中均匀选择。接下来，证明者运行 $\mathcal{P}_2(sk, st, r)$ 来计算响应 s ，并将其发送回验证者。最后，验证者计算 $\mathcal{V}(pk, r, s)$ ，当且仅当结果是初始消息 I 时才接受；见图 12.1。当然，为了正确性，我们要求如果合法的证明者正确执行协议，则验证者应该总是接受。

图 12.1：一个三轮身份识别方案。证明者 $(sk) (I, st) \leftarrow \mathcal{P}_1(sk)$ $s := \mathcal{P}_2(sk, st, r)$ **验证者** $(pk) \xleftarrow{I} \xrightarrow{r \leftarrow \Omega_{pk}} \xleftarrow{s}$ 检查 $\mathcal{V}(pk, r, s) \stackrel{?}{=} I$

出于技术原因，我们假设身份识别方案是“非退化的”，直观地意味着存在许多可能的初始消息 I ，并且没有一个被发送的概率很高。形式上，如果对于每个私钥 sk 和任何固定的初始消息 I ， $\mathcal{P}_1(sk)$ 输出 I 的概率是**可忽略的**，那么一个方案是**非退化的**。（任何身份识别方案都可以通过发送一个均匀的 n 比特字符串以及初始消息来平凡地修改为非退化。）

身份识别方案的基本安全要求是，不知道证明者私钥的敌手应该无法欺骗验证者接受。即使攻击者能够被动窃听证明者和验证者之间多次（诚实）执行协议，这也应该成立。我们通过一个谕示 Trans_{sk} 来形式化这种窃听，该谕示在没有输入的情况下被调用时，运行协议的诚实执行并向敌手返回整个**记录** (I, r, s) 的交互。

令 $\Pi = (\text{Gen}, \mathcal{P}_1, \mathcal{P}_2, \mathcal{V})$ 为一个身份识别方案，并考虑以下针对敌手 \mathcal{A} 和参数 n 的实验：

身份识别实验 $\text{Ident}_{\mathcal{A}, \Pi}(n)$:

1. 运行 $\text{Gen}(1^n)$ 以获得密钥 (pk, sk) 。
2. 敌手 \mathcal{A} 被给予 pk 并可以访问谕示 Trans_{sk} ，它可以根据需要多次查询。
3. 在实验期间的任何时候， \mathcal{A} 输出一个消息 I 。一个均匀的挑战者 $r \in \Omega_{pk}$ 被选择并给予 \mathcal{A} ， \mathcal{A} 用一些 s 响应。 $(\mathcal{A}$ 甚至在接收到 r 之后也可以继续查询 Trans_{sk} 。)
4. 当且仅当 $\mathcal{V}(pk, r, s) \stackrel{?}{=} I$ 时，实验输出 1。

定义 12.8 如果对于所有概率多项式时间敌手 \mathcal{A} ，存在一个**可忽略的**函数 negl 使得：

$$\Pr[\text{Ident}_{\mathcal{A}, \Pi}(n) = 1] \leq \text{negl}(n)$$

那么身份识别方案 $\Pi = (\text{Gen}, \mathcal{P}_1, \mathcal{P}_2, \mathcal{V})$ 在被动攻击下是安全的，或简称安全的。

也可以考虑更强的安全性概念，例如，敌手还可以通过冒充验证者并可能发送恶意选择的值 r 来对协议进行主动攻击。对于我们应用于签名方案的应用，我们将不需要这个。

从身份识别方案到签名

Fiat-Shamir 变换（构造 12.9）提供了一种将任何（交互式）身份识别方案转换为（非交互式）签名方案的方法。基本思想是让签名者充当证明者，自行运行身份识别协议。也就是说，要签署消息 m ，签名者首先计算 I ，然后通过对 I 和 m 应用某个函数 H 来生成挑战 r 。然后它导出正确的响应 s 。 m 上的签名是 (r, s) ，可以通过 (1) 重新计算 $I := \mathcal{V}(pk, r, s)$ 然后 (2) 检查 $H(I, m) = ? r$ 来验证。

构造 12.9 令 $(\text{Gen}_{\text{id}}, \mathcal{P}_1, \mathcal{P}_2, \mathcal{V})$ 为一个身份识别方案，并构造一个签名方案如下：

- Gen：输入 1^n ，只需运行 $\text{Gen}_{\text{id}}(1^n)$ 获得密钥 pk, sk 。公钥 pk 指定了一组挑战 Ω_{pk} 。作为密钥生成的一部分，指定了一个函数 $H : \{0, 1\}^* \rightarrow \Omega_{pk}$ ，但我们在此省略。
- Sign：输入私钥 sk 和消息 $m \in \{0, 1\}^*$ ，执行：
 - i. 计算 $(I, st) \leftarrow \mathcal{P}_1(sk)$ 。
 - ii. 计算 $r := H(I, m)$ 。
 - iii. 计算 $s := \mathcal{P}_2(sk, st, r)$ 。输出签名 (r, s) 。
- Vrfy：输入公钥 pk 、消息 m 和签名 (r, s) ，计算 $I := \mathcal{V}(pk, r, s)$ ，当且仅当 $H(I, m) = r$ 时输出 1。

Fiat-Shamir 变换。

签名 (r, s) 与特定消息 m “绑定”，因为 r 是 I 和 m 的函数；改变 m 会导致完全不同的 r 。如果 H 被建模为将输入均匀映射到 Ω_{pk} 的随机谕示，则挑战 r 是均匀的；直观上，对于不知道 sk 的敌手来说，找到消息 m 上的有效签名 (r, s) 就像在协议的诚实执行中冒充证明者一样困难。这种直觉在以下定理的证明中得到形式化。

定理 12.10 令 Π 是一个身份识别方案，并令 Π' 是应用 Fiat-Shamir 变换得到的签名方案。如果 Π 是安全的，并且 H 被建模为随机谕示，那么 Π' 是安全的。

证明 令 \mathcal{A}' 是一个攻击签名方案 Π' 的概率多项式时间敌手，令 $q = q(n)$ 是 \mathcal{A}' 对 H 进行查询次数的多项式上限。我们做了一些简化假设，不失一般性。首先，我们假设 \mathcal{A}' 对 H 的任何给定查询只进行一次。我们还假设在收到消息 m 上带有 $\mathcal{V}(pk, r, s) = I$ 的签名 (r, s)

后，敌手 \mathcal{A}' 绝不会查询 $H(I, m)$ （因为它知道答案将是 r ）。最后，我们假设如果 \mathcal{A}' 输出消息 m 上的伪造签名 (r, s) 并且 $\mathcal{V}(pk, r, s) = I$ ，那么 \mathcal{A}' 之前已经查询了 $H(I, m)$ 。

我们构造一个高效的敌手 \mathcal{A} ，它使用 \mathcal{A}' 作为子程序并攻击身份识别方案 Π ：

算法 \mathcal{A} : 算法被给予 pk 和对谕示 Trans_{sk} 的访问权限。

1. 均匀选择 $j \in \{1, \dots, q\}$ 。
2. 运行 $\mathcal{A}'(pk)$ 。回答其查询如下：当 \mathcal{A}' 进行其第 i 个随机谕示查询 $H(I_i, m_i)$ 时，回答如下：
 - 如果 $i = j$ ，输出 I_j 并接收作为回报的挑战 r 。返回 r 作为对 \mathcal{A}' 的查询的答案。
 - 如果 $i \neq j$ ，选择均匀的 $r \in \Omega_{pk}$ 并返回 r 作为查询的答案。当 \mathcal{A}' 请求消息 m 上的签名时，回答如下：(a) 查询 Trans_{sk} 以获取协议的诚实执行的记录 (I, r, s) 。
(b) 返回签名 (r, s) 。
3. 如果 \mathcal{A}' 输出消息 m 上的伪造签名 (r, s) ，计算 $I := \mathcal{V}(pk, r, s)$ 并检查 $(I, m) \stackrel{?}{=} (I_j, m_j)$ 。如果是，则输出 s 。否则，中止。

当 \mathcal{A}' 作为 \mathcal{A} 的子程序在实验 $\text{Ident}_{\mathcal{A}, \Pi}(n)$ 中运行时， \mathcal{A}' 的视图与 \mathcal{A}' 在实验 $\text{Sig-forge}_{\mathcal{A}, \Pi'}''(n)$ 中的视图几乎完全相同。事实上， \mathcal{A}' 进行的所有 H 查询都用 Ω_{pk} 的均匀值回答，并且 \mathcal{A}' 进行的所有签名查询都用具有正确分布的有效签名回答。 \mathcal{A}' 在作为 \mathcal{A} 的子程序运行时与 \mathcal{A}' 视图的唯一区别是， \mathcal{A}' 从其对 H 的查询中收到的答案可能存在不一致：具体来说，如果 \mathcal{A} 曾经使用记录 (I, r, s) 回答消息 m 的签名查询，其中 $H(I, m)$ 已经被定义（即 \mathcal{A}' 之前已经查询了 (I, m) 到 H ）并且 $H(I, m) \neq r$ ，就会发生这种情况。然而，如果 Π 是非退化的，那么这只以可忽略的概率发生。因此，当作为 \mathcal{A} 的子程序运行时， \mathcal{A}' 输出伪造的概率是 $\text{Sig-forge}_{\mathcal{A}, \Pi'}''(n) - \text{negl}(n)$ ，其中 $\text{negl}(n)$ 是某个可忽略函数。

考虑实验 $\text{Ident}_{\mathcal{A}, \Pi}(n)$ 的一次执行，其中 \mathcal{A}' 输出消息 m 上的伪造签名 (r, s) ，并令 $I := \mathcal{V}(pk, r, s)$ 。由于 j 是均匀的且独立于其他一切， $(I, m) = (I_j, m_j)$ 的概率（即使以 \mathcal{A}' 输出伪造的事件为条件）正好是 $1/q$ 。（回想一下我们假设如果 \mathcal{A}' 输出消息 m 上带有 $\mathcal{V}(pk, r, s) = I$ 的伪造签名 (r, s) ，那么 \mathcal{A}' 之前已经查询了 $H(I, m)$ 。）当这两个事件都发生时， \mathcal{A} 成功地冒充了证明者。事实上， \mathcal{A} 发送 I_j 作为其初始消息，接收作为响应的挑战 r ，并用 s 响应。但是 $H(I_j, m_j) = r$ 并且（因为伪造签名是有效的） $\mathcal{V}(pk, r, s) = I$ 。将所有内容放在一起，我们看到

$$\Pr[\text{Ident}_{\mathcal{A}, \Pi}(n) = 1] \geq \frac{1}{q(n)} \cdot (\Pr[\text{Sig-forge}_{\mathcal{A}, \Pi'}''(n) = 1] - \text{negl}(n))$$

或

$$\Pr[\text{Sig-forge}_{\mathcal{A}', \Pi'}''(n) = 1] \leq q(n) \cdot \Pr[\text{Ident}_{\mathcal{A}, \Pi}(n) = 1] + \text{negl}(n)$$

如果 Π 是安全的，则 $\Pr[\text{Ident}_{\mathcal{A}, \Pi}(n) = 1]$ 是可忽略的；由于 $q(n)$ 是多项式，这意味着 $\Pr[\text{Sig-forge}_{\mathcal{A}', \Pi'}''(n) = 1]$ 也是可忽略的。因为 \mathcal{A}' 是任意的，这意味着 Π' 是安全的。

Schnorr 身份识别方案

Schnorr 身份识别方案基于**离散对数问题**的困难性。令 \mathcal{G} 是一个多项式时间算法，它将 1^n 作为输入，并输出循环群 \mathbb{G} 的描述，其阶 q （其中 $\|q\| = n$ ）和一个生成元 g 。为了生成其密钥，证明者运行 $\mathcal{G}(1^n)$ 以获得 (\mathbb{G}, q, g) ，选择均匀的 $x \in \mathbb{Z}_q$ ，并设置 $y := g^x$ ；公钥是 (\mathbb{G}, q, g, y) ，私钥是 x 。为了执行协议（见图 12.2），证明者首先选择一个均匀的 $k \in \mathbb{Z}_q$ 并设置 $I := g^k$ ；它发送 I 作为初始消息。验证者选择并发送一个均匀的挑战 $r \in \mathbb{Z}_q$ ；作为响应，证明者计算 $s := [rx + k \bmod q]$ 。当且仅当 $g^s \cdot y^{-r} \stackrel{?}{=} I$ 时，验证者接受。正确性成立，因为

$$g^s \cdot y^{-r} = g^{rx+k} \cdot (g^x)^{-r} = g^{rx+k} \cdot g^{-rx} = g^k = I$$

请注意 I 在 \mathbb{G} 中是均匀的，因此该方案是非退化的。

在给出证明之前，我们提供一些高层直觉。第一个重要的观察是**被动窃听**对攻击者没有帮助。原因是攻击者可以仅基于公钥**模拟**诚实执行的记录，而**无需知道**私钥。为此，攻击者只需颠倒步骤的顺序：它首先选择均匀且独立的 $r, s \in \mathbb{Z}_q$ ，然后设置 $I := g^s \cdot y^{-r}$ 。在一个诚实记录 (I, r, s) 中，初始消息 I 是 \mathbb{G} 的均匀元素，挑战 r 是 \mathbb{Z}_q 的独立、均匀元素，然后 s 唯一地确定为 $s = \log_g(I \cdot y^r)$ 。攻击者构造的模拟记录

图 12.2：Schnorr 身份识别方案的执行。 证明者 $(x) \quad k \leftarrow \mathbb{Z}_q \quad I := g^k \quad s := [rx + k \bmod q]$ 验证者 $(\mathbb{G}, q, g, y) \xleftarrow{I} \xrightarrow{r \leftarrow \mathbb{Z}_q} \xleftarrow{s} \text{检查 } g^s \cdot y^{-r} \stackrel{?}{=} I$

具有相同的分布： $r \in \mathbb{Z}_q$ 是均匀的，并且因为 s 在 \mathbb{Z}_q 中是均匀的并且独立于 r ，我们看到 I 在 \mathbb{G} 中是均匀的并且独立于 r 。最后， s 唯一地确定为满足与以前相同的约束。因此，我们可以有效地假设，在攻击身份识别方案时，攻击者根本不窃听诚实执行。

因此，我们简化为一个攻击者，他获得一个公钥 y ，发送一个初始消息 I ，作为响应获得一个均匀挑战 r ，然后必须发送一个响应 s 使得 $g^s \cdot y^{-r} = I$ 。非正式地，如果攻击者能够以高概率做到这一点，那么他必须特别能够计算对至少两个不同挑战 $r_1, r_2 \in \mathbb{Z}_q$ 的正确响应 s_1, s_2 。请注意

$$g^{s_1} \cdot y^{-r_1} = I = g^{s_2} \cdot y^{-r_2}$$

因此 $g^{s_1-s_2} = y^{r_1-r_2}$ 。但这暗示了攻击者（回想一下，他能够生成对 r_1 的响应 s_1 和对 r_2 的响应 s_2 ）可以隐式地计算离散对数

$$\log_g y = [(s_1 - s_2) \cdot (r_1 - r_2)^{-1} \bmod q]$$

与离散对数问题的假设困难性相矛盾。

定理 12.11 如果相对于 \mathcal{G} 而言离散对数问题是困难的，那么 Schnorr 身份识别方案是安全的。

证明 令 Π 表示 Schnorr 身份识别方案，并令 \mathcal{A} 是一个攻击该方案的 PPT 敌手。我们构造以下 PPT 算法 \mathcal{A}' 解决相对于 \mathcal{G} 的离散对数问题：

算法 \mathcal{A}' : 算法被给予 \mathbb{G}, q, g, y 作为输入。

1. 运行 $\mathcal{A}(pk)$ ，回答其对 Trans_{sk} 的所有查询，如前面直觉中所述。
2. 当 \mathcal{A} 输出 I 时，选择均匀的 $r_1 \in \mathbb{Z}_q$ 作为挑战。将 r_1 给予 \mathcal{A} ， \mathcal{A} 用 s_1 响应。
3. 第二次（从头开始）运行 $\mathcal{A}(pk)$ ，使用与以前相同的随机性，除了使用均匀且独立于 r_1 的 $r_2 \in \mathbb{Z}_q$ 。最终， \mathcal{A} 用 s_2 响应。
4. 如果 $g^{s_1} \cdot y^{-r_1} = I$ 且 $g^{s_2} \cdot y^{-r_2} = I$ 且 $r_1 \neq r_2$ ，则输出 $[(s_1 - s_2) \cdot (r_1 - r_2)^{-1} \bmod q]$ 。否则，不输出任何内容。

考虑 \mathcal{A} 作为 \mathcal{A}' 的子程序的一次运行，令 ω 表示该执行中使用的随机性，除了挑战本身。因此， ω 包括 \mathcal{G} 使用的任何随机性，选择（未知）私钥 x ， \mathcal{A} 本身使用的任何随机性，以及 \mathcal{A}' 在回答对 Trans_{sk} 的查询时使用的随机性。定义 $V(\omega, r)$ 等于 1 当且仅当在使用随机性 ω 执行的其余部分中， \mathcal{A} 正确响应挑战 r 。对于任何固定的 ω ，定义 $\delta_\omega \stackrel{\text{def}}{=} \Pr_r[V(\omega, r) = 1]$ ；固定 ω 后，这是 \mathcal{A} 正确响应的挑战选择的概率。定义 $\delta(n) \stackrel{\text{def}}{=} \Pr[\text{Ident}_{\mathcal{A}, \Pi}(n) = 1]$ 。由于 Trans_{sk} 意示的模拟是完美的，我们有

$$\delta(n) = \Pr_{\omega, r}[V(\omega, r) = 1] = \sum_{\omega} \Pr[\omega] \cdot \delta_\omega$$

此外，证明之前的直觉表明，只要 \mathcal{A} 成功两次并且 $r_1 \neq r_2$ ， \mathcal{A}' 就会正确计算 y 的离散对数。因此：

$$\begin{aligned} \Pr[\text{DLog}_{\mathcal{A}', \mathcal{G}}(n) = 1] &= \Pr_{\omega, r_1, r_2}[V(\omega, r_1) \wedge V(\omega, r_2) \wedge r_1 \neq r_2] \\ &\geq \Pr_{\omega, r_1, r_2}[V(\omega, r_1) \wedge V(\omega, r_2)] - \Pr_{\omega, r_1, r_2}[r_1 = r_2] \end{aligned}$$

$$\begin{aligned}
&= \sum_{\omega} \Pr[\omega] \cdot (\delta_{\omega})^2 - 1/q \\
&\geq \left(\sum_{\omega} \Pr[\omega] \cdot \delta_{\omega} \right)^2 - 1/q \\
&= \delta(n)^2 - 1/q
\end{aligned}$$

在倒数第二步中使用了 Jensen 不等式⁴。如果相对于 \mathcal{G} 而言离散对数问题是困难的，那么 $\Pr[\text{DLog}_{\mathcal{A}, \mathcal{G}}(n) = 1]$ 是可忽略的。由于 $1/q$ 是可忽略的（因为 $\|q\| = n$ ），这意味着 $\delta(n)$ 也是可忽略的，因此 Π 是一个安全的身份识别方案。

Schnorr 签名方案是通过将 Fiat-Shamir 变换应用于 Schnorr 身份识别方案获得的。见构造 12.12。

构造 12.12 令 \mathcal{G} 如正文所述。

- Gen: 运行 $\mathcal{G}(1^n)$ 获得 (\mathbb{G}, q, g) 。选择一个均匀的 $x \in \mathbb{Z}_q$ 并设置 $y := g^x$ 。私钥是 x ，公钥是 (\mathbb{G}, q, g, y) 。作为密钥生成的一部分，指定了一个函数 $H : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ ，但我们在此省略。
- Sign: 输入私钥 x 和消息 $m \in \{0, 1\}^*$ ，选择均匀的 $k \in \mathbb{Z}_q$ 并设置 $I := g^k$ 。然后计算 $r := H(I, m)$ ，接着是 $s := [rx + k \bmod q]$ 。输出签名 (r, s) 。
- Vrfy: 输入公钥 (\mathbb{G}, q, g, y) 、消息 m 和签名 (r, s) ，计算 $I := g^s \cdot y^{-r}$ ，当且仅当 $H(I, m) \stackrel{?}{=} r$ 时输出 1。

Schnorr 签名方案。

12.5.2 DSA 和 ECDSA

数字签名算法 (DSA) 和**椭圆曲线数字签名算法 (ECDSA)** 基于不同类别群中的离散对数问题。它们自 1991 年以来以某种形式存在，并且都包含在 NIST 发布的当前**数字签名标准 (DSS)** 中。这两种方案都遵循一个共同的模板，可以被视为由一个底层身份识别方案构造而成（见上一节）。

令 \mathbb{G} 是一个素数阶 q 的循环群，生成元为 g 。考虑以下身份识别方案，其中证明者的私钥是 x ，公钥是 (\mathbb{G}, q, g, y) 且 $y = g^x$ ：

1. 证明者选择均匀的 $k \in \mathbb{Z}_q$ 并发送 $I := g^k$ 。
2. 验证者选择并发送均匀的 $a, r \in \mathbb{Z}_q$ 作为挑战。

3. 证明者发送 $s := [k^{-1} \cdot (a + xr) \bmod q]$ 作为响应。
4. 当且仅当 $s \neq 0$ 且 $g^{as^{-1}} \cdot y^{rs^{-1}} \stackrel{?}{=} I$ 时，验证者接受。

请注意，除非 $a = -xr \bmod q$ （以可忽略的概率发生），否则 $s \neq 0$ 。假设 $s \neq 0$ ，则逆 $s^{-1} \bmod q$ 存在，并且

$$g^{as^{-1}} \cdot y^{rs^{-1}} = g^{as^{-1}} \cdot g^{xrs^{-1}} = g^{(a+xr)s^{-1}} = g^{(a+xr) \cdot k^{-1} \cdot (a+xr)^{-1}} = g^k = I$$

因此我们看到，除了可忽略的概率之外，正确性成立。

可以证明，如果相对于 \mathbb{G} 而言离散对数问题是困难的，则该身份识别方案是安全的。我们仅勾勒论证，假设熟悉上一节的结果。首先，可以模拟诚实执行的记录：为此，只需选择均匀的 $a, r \in \mathbb{Z}_q$ 和 $s \in \mathbb{Z}_q$ ，然后设置 $I := g^{as^{-1}} \cdot y^{-rs^{-1}}$ 。（这不再是完美模拟，但已经足够接近。）此外，如果攻击者输出初始消息 I ，并且可以对不同挑战 $(a, r_1), (a, r_2)$ 给出正确响应 $s_1, s_2 \in \mathbb{Z}_q$ ，那么

$$g^{as_1^{-1}} \cdot y^{r_1 s_1^{-1}} = I = g^{as_2^{-1}} \cdot y^{r_2 s_2^{-1}}$$

因此 $g^{a(s_1^{-1} - s_2^{-1})} = y^{r_2 s_2^{-1} - r_1 s_1^{-1}}$ ，并且可以像上一节中那样计算 $\log_g y$ 。如果攻击者对不同挑战 $(a_1, r), (a_2, r)$ 给出正确响应，也会发生同样的情况。

DSA/ECDSA 签名方案是通过将上述身份识别方案“坍缩”为由签署者运行的非交互式算法来构造的。然而，与 Fiat-Shamir 变换相反，这里的变换如下执行（见构造 12.13）：

- 设置 $a := H(m)$ ，其中 m 是被签署的消息， H 是一个密码哈希函数。
- 设置 $r := F(I)$ ，其中 $F : \mathbb{G} \rightarrow \mathbb{Z}_q$ 是一个（指定的）函数。这里， F 是一个“简单”函数，不旨在充当随机谕示。

函数 F 取决于群 \mathbb{G} ，而群 \mathbb{G} 又取决于方案。在 DSA 中， \mathbb{G} 被认为是 \mathbb{Z}_p^* 的一个 q 阶子群，其中 p 是素数（参见第 8.3.3 节），并且 $F(I) \stackrel{\text{def}}{=} [I \bmod q]$ 。在 ECDSA 中， \mathbb{G} 是椭圆曲线群 $E(\mathbb{Z}_p)$ 的一个 q 阶子群，其中 p 是素数。回想一下第 8.3.4 节，这样一个群的任何元素都可以表示为 $\mathbb{Z}_p \times \mathbb{Z}_p$ 中的一对 (x, y) 。在这种情况下，函数 F 定义为 $F((x, y)) \stackrel{\text{def}}{=} [x \bmod q]$ 。

构造 12.13 令 \mathcal{G} 如正文所述。

- Gen：输入 1^n ，运行 $\mathcal{G}(1^n)$ 获得 (\mathbb{G}, q, g) 。选择均匀的 $x \in \mathbb{Z}_q$ 并设置 $y := g^x$ 。公钥是 (\mathbb{G}, q, g, y) ，私钥是 x 。作为密钥生成的一部分，指定了两个函数 $H : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ 和 $F : \mathbb{G} \rightarrow \mathbb{Z}_q$ ，但我们在此省略。

- Sign: 输入私钥 x 和消息 $m \in \{0, 1\}^*$, 选择均匀的 $k \in \mathbb{Z}_q$ 并设置 $r := F(g^k)$ 。然后计算 $s := [k^{-1} \cdot (H(m) + xr) \bmod q]$ 。（如果 $r = 0$ 或 $s = 0$, 则重新开始, 重新选择 k 。）输出签名 (r, s) 。
- Vrfy: 输入公钥 (\mathbb{G}, q, g, y) 、消息 m 和签名 (r, s) , 其中 $r, s \neq 0 \bmod q$, 当且仅当

$$r \stackrel{?}{=} F(g^{H(m)s^{-1}} \cdot y^{rs^{-1}})$$

时输出 1。

DSA 和 ECDSA - 抽象。

假设离散对数问题的困难性, 如果 H 和 F 都被建模为随机谕示, 则可以证明 DSA 和 ECDSA 是安全的。然而, 正如我们上面讨论的, 虽然随机谕示模型对于 H 可能是合理的, 但对于 F 来说, 它不是一个合适的模型。对于标准中 F 的特定选择, 没有已知的安全性证明。尽管如此, DSA 和 ECDSA 已经被使用和研究了数十年, 没有发现任何攻击。

k 的正确生成。 DSA/ECDSA 方案指定签名者在计算签名时应选择一个**均匀**的 $k \in \mathbb{Z}_q$ 。未能正确选择 k (例如, 由于不良的随机数生成) 可能导致**灾难性**后果。首先, 如果攻击者可以预测用于计算消息 m 上的签名 (r, s) 的 k 值, 那么他们就可以计算签名者的私钥 x 。这是真的, 因为 $s = k^{-1} \cdot (H(m) + xr) \bmod q$, 如果 k 是已知的, 那么唯一的未知数是私钥 x 。

即使 k 是不可预测的, 如果**相同的** k 被用于生成两个不同的签名, 攻击者也可以计算签名者的私钥。如果发生这种情况, 攻击者可以很容易地知道, 因为 r 也会重复。假设 (r, s_1) 和 (r, s_2) 分别是消息 m_1 和 m_2 上的签名。那么

$$s_1 = k^{-1} \cdot (H(m_1) + xr) \bmod q$$

$$s_2 = k^{-1} \cdot (H(m_2) + xr) \bmod q$$

相减得到 $s_1 - s_2 = k^{-1}(H(m_1) - H(m_2)) \bmod q$, 从中可以计算 k ; 给定 k , 攻击者可以像上一段中那样确定私钥 x 。黑客在 2010 年就是利用这种攻击提取了 Sony PlayStation (PS3) 的主私钥。

12.6 *基于哈希函数的签名

有趣的是——也许有些令人惊讶——签名方案可以基于**密码哈希函数**来构造, **不依赖于数论假设**。(这与公钥加密形成对比, 公钥加密似乎需要具有某种代数结构的困难问题。) 在本节中, 我们探讨了这种构造。我们将看到的方案也很有趣, 因为它们不依赖于随机谕示, 与本章

中我们之前描述的所有构造相反。

12.6.1 Lamport 签名方案

我们开始研究基于哈希函数的签名方案，从相对较弱的一次性安全签名方案概念开始。非正式地，只要给定的私钥仅用于签署单个消息，此类方案就是“安全”的。满足此安全概念的方案可能适用于某些应用，并且还可以作为实现更强安全性概念的有用“构建块”，正如我们将在下一节中看到的那样。

令 $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$ 为一个签名方案，并考虑以下针对敌手 \mathcal{A} 和参数 n 的实验：

一次性签名实验 $\text{Sig-forge}_{\mathcal{A}, \Pi}^{\text{one-time}}(n)$:

1. 运行 $\text{Gen}(1^n)$ 以获得密钥 (pk, sk) 。
2. 敌手 \mathcal{A} 被给予 pk 并向其谕示 $\text{Sign}_{sk}(\cdot)$ 询问单个查询 m' 。然后 \mathcal{A} 输出 (m, σ) ，其中 $m \neq m'$ 。
3. 当且仅当 $\text{Vrfy}_{pk}(m, \sigma) = 1$ 时，实验的输出定义为 1。

定义 12.14 如果对于所有概率多项式时间敌手 \mathcal{A} ，存在一个可忽略的函数 negl 使得：

$$\Pr[\text{Sig-forge}_{\mathcal{A}, \Pi}^{\text{one-time}}(n) = 1] \leq \text{negl}(n)$$

那么签名方案 $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$ 在单消息攻击下是存在不可伪造的，或是一个一次性安全签名方案。

Leslie Lamport 在 1979 年展示了一个一次性安全签名方案的构造。我们以签署 3 比特消息为例说明这个想法。令 H 是一个密码哈希函数。私钥包含六个均匀值

$x_{1,0}, x_{1,1}, x_{2,0}, x_{2,1}, x_{3,0}, x_{3,1} \in \{0, 1\}^n$ ，相应的公钥包含将 H 应用于这些元素中的每一个的结果。这些密钥可以可视化为二维数组：

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & y_{3,0} \\ y_{1,1} & y_{2,1} & y_{3,1} \end{pmatrix} \quad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{pmatrix}$$

要签署消息 $m = m_1 m_2 m_3$ （其中 $m_i \in \{0, 1\}$ ），签名者为消息的每个比特发布相应的原像 x_{i, m_i} ；签名 σ 由三个值 $(x_{1, m_1}, x_{2, m_2}, x_{3, m_3})$ 组成。验证以自然的方式进行：给定消息 $m = m_1 m_2 m_3$ 的候选签名 (x_1, x_2, x_3) ，当且仅当 $H(x_i) = ? y_{i, m_i}$ 对于 $1 \leq i \leq 3$ 时，接受。这在图 12.3 中以图形方式显示，对于任何长度 l 的消息的一般情况在构造 12.15 中正式描述。

图 12.3：用于签署消息 $m = 011$ 的 Lamport 方案。签署 $m = 011$ ：

$$sk = \begin{pmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{pmatrix} \implies \sigma = (x_{1,0}, x_{2,1}, x_{3,1})$$

验证 $m = 011$ 和 $\sigma = (x_1, x_2, x_3)$:

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & y_{3,0} \\ y_{1,1} & y_{2,1} & y_{3,1} \end{pmatrix} \implies \begin{cases} H(x_1) \stackrel{?}{=} y_{1,0} \\ H(x_2) \stackrel{?}{=} y_{2,1} \\ H(x_3) \stackrel{?}{=} y_{3,1} \end{cases}$$

观察到消息上的签名后，希望伪造任何其他消息上的签名的攻击者必须找到公钥中三个“未使用”元素之一的原像。如果 H 是单向的（见定义 8.72），那么找到任何这样的原像在计算上是困难的。

定理 12.16 令 l 是任意多项式。如果 H 是一个单向函数，那么构造 12.15 是一个一次性安全签名方案。

构造 12.15 令 $H : \{0, 1\}^* \rightarrow \{0, 1\}^*$ 是一个函数。构造一个长度为 $l = l(n)$ 的消息的签名方案如下：

- Gen: 输入 1^n ，执行以下步骤对于 $i \in \{1, \dots, l\}$:
 - 选择均匀的 $x_{i,0}, x_{i,1} \in \{0, 1\}^n$ 。
 - 计算 $y_{i,0} := H(x_{i,0})$ 和 $y_{i,1} := H(x_{i,1})$ 。公钥 pk 和私钥 sk 是

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{l,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{l,1} \end{pmatrix} \quad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{l,0} \\ x_{1,1} & x_{2,1} & \cdots & x_{l,1} \end{pmatrix}$$

- Sign: 输入如上所述的私钥 sk 和消息 $m \in \{0, 1\}^l$ ，其中 $m = m_1 \dots m_l$ ，输出签名 $(x_{1,m_1}, \dots, x_{l,m_l})$ 。
- Vrfy: 输入如上所述的公钥 pk 、消息 $m \in \{0, 1\}^l$ ，其中 $m = m_1 \dots m_l$ ，和签名 $\sigma = (x_1, \dots, x_l)$ ，当且仅当 $H(x_i) = y_{i,m_i}$ 对于所有 $1 \leq i \leq l$ 时输出 1。

Lamport 签名方案。

证明 令 $l = l(n)$ 贯穿始终。正如前面提到的，关键的观察是：假设攻击者 \mathcal{A} 请求消息 m' 上的签名，并考虑任何其他消息 $m \neq m'$ 。在 m 和 m' 不同的至少一个位置 $i^* \in \{1, \dots, l\}$ 上一定存在。假设 $m_{i^*} = b \neq m'_{i^*}$ 。那么伪造 m 上的签名至少需要找到公钥的元素 $y_{i^*,b}$ 的原像（在 H 下）。由于 H 是单向的，这应该在计算上是不可行的。现在我们形式化这个直觉。

令 Π 表示 Lamport 方案，并令 \mathcal{A} 是一个概率多项式时间对手。在 $\text{Sig-forge}_{\mathcal{A}, \Pi}^{\text{one-time}}(n)$ 的特定执行中，令 m' 表示 \mathcal{A} 请求签名的消息（我们假设不失一般性， \mathcal{A} 总是请求消息上的签名），并令 (m, σ) 是 \mathcal{A} 的最终输出。我们说 \mathcal{A} 在 (i, b) 处输出伪造，如果 $\text{Vrfy}_{pk}(m, \sigma) = 1$ 并且此外 $m_i \neq m'_i$ （即消息 m 和 m' 在它们的第 i 个位置上不同）并且 $m_i = b \neq m'_i$ 。请注意，每当 \mathcal{A} 输出伪造时，它总是在某个 (i, b) 处输出伪造。

考虑以下尝试反转 H 的 PPT 算法 \mathcal{I} :

算法 \mathcal{I} : 算法被给予 1^n 和 y 作为输入。

1. 均匀选择 $i^* \in \{1, \dots, l\}$ 和 $b^* \in \{0, 1\}$ 。设置 $y_{i^*, b^*} := y$ 。
2. 对于所有 $i \in \{1, \dots, l\}$ 和 $b \in \{0, 1\}$ ，如果 $(i, b) \neq (i^*, b^*)$ ：
 - 选择均匀的 $x_{i,b} \in \{0, 1\}^n$ 并设置 $y_{i,b} := H(x_{i,b})$ 。
3. 在输入

$$pk := \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{l,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{l,1} \end{pmatrix}$$

上运行 \mathcal{A} 。

4. 当 \mathcal{A} 请求消息 m' 上的签名时：
 - 如果 $m'_{i^*} = b^*$ ，则 \mathcal{I} 中止执行。
 - 否则， \mathcal{I} 返回签名 $\sigma = (x_{1,m'_1}, \dots, x_{l,m'_l})$ 。
5. 当 \mathcal{A} 输出 (m, σ) ，其中 $\sigma = (x_1, \dots, x_l)$ 时：
 - 如果 \mathcal{A} 在 (i^*, b^*) 处输出伪造，则输出 x_{i^*} 。

每当 \mathcal{A} 在 (i^*, b^*) 处输出伪造时，算法 \mathcal{I} 成功地反转了其给定的输入 y 。我们感兴趣的是当 \mathcal{I} 的输入是通过选择均匀的 $x \in \{0, 1\}^n$ 并设置 $y := H(x)$ （参见定义 8.72）生成时，这种情况发生的概率。

想象一个“心智实验”，其中 \mathcal{I} 在一开始就被给予 x ，设置 $x_{i^*, b^*} := x$ ，然后总是在步骤 4 中返回对 \mathcal{A} 的签名（即，即使 $m'_{i^*} = b^*$ ）。当 \mathcal{A} 在这个心智实验中作为 \mathcal{I} 的子程序运行时， \mathcal{A} 的视图与 \mathcal{A} 在实验 $\text{Sig-forge}_{\mathcal{A}, \Pi}^{\text{one-time}}(n)$ 中的视图具有相同的分布。因为 (i^*, b^*) 是在实验开始时均匀选择的，并且 \mathcal{A} 的视图与这个选择无关，所以 \mathcal{A} 输出伪造的概率，以 \mathcal{A} 输出伪造为条件，至少是 $1/(2l)$ 。（这是因为签名伪造意味着至少在一个点 (i, b) 处存在伪造。由于有 $2l(n)$ 个点，伪造发生在 (i^*, b^*) 的概率至少是 $1/(2l)$ 。）我们得出结论，在这个心智实验中， \mathcal{A} 在 (i^*, b^*) 处输出伪造的概率至少是 $\frac{1}{2l} \cdot \Pr[\text{Sig-forge}_{\mathcal{A}, \Pi}^{\text{one-time}}(n) = 1]$ 。

回到最初描述的涉及 \mathcal{I} 的**真实**实验，关键的观察是 \mathcal{A} 在 (i^*, b^*) 处输出伪造的概率是**不变**

的。这是因为心智实验和真实实验仅在 \mathcal{A} 在 (i^*, b^*) 处输出伪造时重合。也就是说，实验仅在 $m'_{i^*} = b^*$ 时不同，但如果发生这种情况，那么 \mathcal{A} （根据定义）不可能随后在 (i^*, b^*) 处输出伪造。因此， \mathcal{A} 在 (i^*, b^*) 处输出伪造的概率仍然至少是 $\frac{1}{2l} \cdot \Pr[\text{Sig-forge}_{\mathcal{A}, \Pi}^{\text{one-time}}(n) = 1]$ 。换句话说，

$$\Pr[\text{Invert}_{\mathcal{I}, H}(n) = 1] \geq \frac{1}{2l} \cdot \Pr[\text{Sig-forge}_{\mathcal{A}, \Pi}^{\text{one-time}}(n) = 1]$$

因为 H 是一个单向函数，所以存在一个可忽略的函数 negl 使得

$$\text{negl}(n) \geq \Pr[\text{Invert}_{\mathcal{I}, H}(n) = 1]$$

由于 l 是多项式，这意味着 $\Pr[\text{Sig-forge}_{\mathcal{A}, \Pi}^{\text{one-time}}(n) = 1]$ 是可忽略的，完成了证明。

推论 12.17 如果存在单向函数，那么对于任何多项式 l ，存在一个用于长度为 l 的消息的一次性安全签名方案。

12.6.2 基于链的签名

使用给定的私钥只能签署单个消息显然是一个重大的缺点。我们在这里展示了一种基于**抗碰撞哈希函数**的方法，它允许签名者签署**任意数量**的消息，代价是必须**维护状态**，该状态在每次签名生成后必须更新。在第 12.6.3 节中，我们讨论了这种方法的一个更有效的变体（仍然需要状态），然后描述了如何使这种修改后的构造**无状态**。结果表明，满足定义 12.2 的签名方案可以基于抗碰撞哈希函数来构造。

我们首先定义允许签名者维护在每次签名生成后更新的**状态**的签名方案。

定义 12.18 一个**有状态签名方案**是概率多项式时间算法的四元组 $(\text{Gen}, \text{Sign}, \text{Vrfy})$ ，满足：

1. **密钥生成算法** Gen 将安全参数 1^n 作为输入，并输出 (pk, sk, s_0) 。这些分别称为**公钥**、**私钥**和**初始状态**。我们假设 pk 和 sk 各自的长度至少为 n ，并且 n 可以从 pk 或 sk 确定。
2. **签名算法** Sign 将私钥 sk 、值 s_{i-1} 和消息 $m \in \{0, 1\}^*$ 作为输入。它输出一个签名 σ 和一个值 s_i 。
3. **确定性验证算法** Vrfy 将公钥 pk 、消息 m 和签名 σ 作为输入。它输出一个比特 b 。

我们要求对于每个 n ，由 $\text{Gen}(1^n)$ 输出的每个 (pk, sk, s_0) ，以及任何消息 $m_1, \dots, m_t \in \{0, 1\}^*$ ，如果迭代计算 $(\sigma_i, s_i) \leftarrow \text{Sign}_{sk, s_{i-1}}(m_i)$ 对于 $i = 1, \dots, t$ ，那么对于每个 $i \in \{1, \dots, t\}$ ，都满足 $\text{Vrfy}_{pk}(m_i, \sigma_i) = 1$ 。

我们强调，验证者**不需要**知道签名者的状态即可验证签名；事实上，在某些方案中，状态必须由签名者保密才能保证安全。不维护状态的签名方案（如定义 12.1 中所述）被称为**无状态方案**，以区别于有状态方案。显然，无状态方案更可取（尽管有状态方案仍然可能有用）。我们引入有状态签名作为通往最终无状态构造的垫脚石。

有状态签名方案的安全性与定义 12.2 **完全类似**，唯一的微妙之处在于签名谕示只返回签名（而不是状态），并且每次调用时都会更新状态。

对于任何多项式 $t = t(n)$ ，我们可以很容易地构造一个有状态的“ t 次安全”签名方案。（这里的安全定义将是定义 12.14 的明显推广。）我们可以通过简单地让公钥（分别私钥）包含 t 个独立生成的一次性安全签名方案的公钥（分别私钥）来实现这一点；即，设置 $pk := (pk_1, \dots, pk_t)$ 和 $sk := (sk_1, \dots, sk_t)$ ，其中每个 (pk_i, sk_i) 是某个一次性安全签名方案独立生成的密钥对。状态是一个计数器 i ，初始设置为 1。要使用私钥 sk 和当前状态 $i < t$ 签署消息 m ，计算 $\sigma \leftarrow \text{Sign}_{sk_i}(m)$ （即，使用私钥 sk_i 生成 m 上的签名），并输出 (σ, i) ；状态更新为 $i := i + 1$ 。由于状态从 1 开始，这意味着第 i 个消息是使用 sk_i 签署的。签名 (σ, i) 在消息 m 上的验证是通过检查 σ 是否是相对于 pk_i 的 m 上的有效签名来完成的。该方案在用于签署 t 个消息时是安全的，因为底层一次性安全方案的每个私钥仅用于签署单个消息。

如上所述，签名具有**恒定长度**（即，与 t 无关），但公钥的长度与 t **线性**相关。有可能通过让签名者计算 t 个底层一次性安全方案公钥的 Merkle 树 $h := \text{MT}_t(pk_1, \dots, pk_t)$ （见第 5.6.2 节）来权衡公钥和签名的长度。也就是说，公钥现在将是 (t, h) ，签名在第 i 个消息上将包括 (σ, i) ，如前所述，以及第 i 个值 pk_i 和证明 π_i ，证明这是对应于 h 的正确值。（验证以自然的方式进行。）公钥现在具有**恒定大小**，签名长度仅与 t **对数**增长。

由于 t 可以是任意多项式，为什么以前的方案没有给我们提供我们正在寻找的解决方案呢？主要的缺点是需要**提前**固定可签署的消息数量的上限 t ，在密钥生成时。这是一个潜在的严重限制，因为一旦达到上限，就必须生成和分发一个新的公钥。我们宁愿拥有一个**单一、固定**的公钥，可用于签署**无限制**数量的消息。

令 $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$ 为一个一次性安全签名方案。在我们刚刚描述的方案中（忽略 Merkle 树优化），签名者运行 t 次 Gen 以获得公钥 pk_1, \dots, pk_t ，并将这些公钥中的每一个都包含在其**实际**公钥 pk 中。签名者随后被限制最多签署 t 个消息。

我们可以通过使用**“基于链”的方案做得更好，其中**签名者**动态地**生成额外的公钥，根据需要。

在基于链的方案中，公钥仅包含使用 Gen 生成的单个公钥 pk_1 ，私钥只是相关的私钥 sk_1 。

要签署第一个消息 m_1 , 签名者首先使用 Gen 生成一个新的密钥对 (pk_2, sk_2) , 然后使用 sk_1 签署 m_1 和 pk_2 , 获得 $\sigma_1 \leftarrow \text{Sign}_{sk_1}(m_1 || pk_2)$ 。输出的签名包括 pk_2 和 σ_1 , 签名者将 $(m_1, pk_2, sk_2, \sigma_1)$ 添加到其当前状态。

通常, 当需要签署第 i 个消息 m_i 时, 签名者将在其状态中存储 $\{(m_j, pk_{j+1}, sk_{j+1}, \sigma_j)\}_{j=1}^{i-1}$ 。要签署第 i 个消息 m_i , 签名者首先使用 Gen 生成一个新的密钥对 (pk_{i+1}, sk_{i+1}) , 然后使用 sk_i 签署 m_i 和 pk_{i+1} , 获得签名 $\sigma_i \leftarrow \text{Sign}_{sk_i}(m_i || pk_{i+1})$ 。输出的**实际**签名包括 pk_{i+1}, σ_i , 以及值 $\{m_j, pk_{j+1}, \sigma_j\}_{j=1}^{i-1}$ 。签名者然后将 $(m_i, pk_{i+1}, sk_{i+1}, \sigma_i)$ 添加到其状态。参见图 12.4, 这是该过程的图形描述。

图 12.4: 基于链的签名: 签署第三个消息 m_3 之前和之后的情况。

签署 m_3 之前:

$$pk_1 \xrightarrow{\sigma_1} m_1 || pk_2 \xrightarrow{\sigma_2} m_2 || pk_3$$

签署 m_3 之后:

$$pk_1 \xrightarrow{\sigma_1} m_1 || pk_2 \xrightarrow{\sigma_2} m_2 || pk_3 \xrightarrow{\sigma_3} m_3 || pk_4$$

要验证消息 $m = m_i$ 上的签名 $(pk_{i+1}, \sigma_i, \{m_j, pk_{j+1}, \sigma_j\}_{j=1}^{i-1})$, 相对于公钥 pk_1 , 接收者验证链中公钥 pk_j 和下一个公钥 pk_{j+1} 之间的每个链接, 以及最后一个公钥 pk_i 和 m 之间的链接。也就是说, 当且仅当 $\text{Vrfy}_{pk_j}(m_j || pk_{j+1}, \sigma_j) \stackrel{?}{=} 1$ 对于所有 $j \in \{1, \dots, i\}$ 时 (参考图 12.4), 验证输出 1。

不难确信——至少在直觉层面上——这个签名方案在适应性选择消息攻击下是**存在不可伪造的** (无论签署了多少消息)。非正式地, 这再次是由于每个密钥对 (pk_i, sk_i) 仅用于签署**单个**“消息”, 在这种情况下, “消息”实际上是消息/公钥对 $m_i || pk_{i+1}$ 。由于我们将在下一节中证明一个更有效方案的安全性, 因此我们在此不证明基于链的方案的安全性。

在基于链的方案中, 每个公钥 pk_i 用于签署**一个消息和另一个公钥**。因此, 底层的一次性安全签名方案 II 必须能够签署**比公钥更长**的消息, 这一点至关重要。第 12.6.1 节中介绍的 Lamport 方案不具备此属性。然而, 如果我们将第 12.3 节的哈希-然后-签名范例应用于 Lamport 方案, 我们确实获得了**可以签署任意长度消息**的一次性安全签名方案。(尽管定理 12.4 仅针对满足定义 12.2 的签名方案进行了说明, 但很容易看出, 相同的证明适用于一次性安全签名方案。) 由于这个结果对下一节至关重要, 我们在此正式说明。(请注意, 抗碰撞哈希函数的存在意味着单向函数的存在; 参见练习 7.4。)

引理 12.19 如果抗碰撞哈希函数存在, 那么存在一个**一次性安全签名方案** (适用于任意长度的

消息)。

基于链的签名方案是一个有状态签名方案，它在适应性选择消息攻击下是**存在不可伪造的**。然而，它有许多缺点。首先，没有直接的方法消除状态（回想一下我们的最终目标是满足定义 12.2 的无状态方案）。它的效率也不高，因为签名长度、状态大小和验证时间都与已签署消息的数量**线性**相关。最后，每个签名都揭示了**所有**先前签署的消息，这在某些情况下可能是不希望的。

12.6.3 基于树的签名

上一节的基于链的方案中的签名者可以被视为维护一棵**度为 1**的树，根位于公钥 pk_1 ，深度等于迄今为止签署的消息数量（参见图 12.4）。提高效率的一种自然方法是使用**二叉树**，其中每个节点具有**度为 2**。与以前一样，签名将对应于树中从叶子到根的“已签署”路径；只要树具有多项式深度（即使它具有指数大小！），验证就可以在多项式时间内完成。

具体来说，为了签署长度为 n 的消息，我们将使用深度为 n 的二叉树，它有 2^n 个叶子。与以前一样，签名者将根据需要“动态地”向树中添加节点。然而，与基于链的方案相比，只有叶子（而不是内部节点）将用于签署消息。树的每个叶子将对应于长度为 n 的可能消息之一。

更详细地说，我们想象一棵深度为 n 的二叉树，其中根标记为 ε （即空字符串），标记为二进制字符串 w （长度小于 n ）的节点有一个标记为 $w0$ 的左孩子和一个标记为 $w1$ 的右孩子。这棵树**从未**完全构造出来（注意它具有指数大小），而是由签名者根据需要**构建**。

对于每个节点 w ，我们关联一个一次性安全签名方案 II 的密钥对 pk_w, sk_w 。根的公钥 pk_ε 是签名者的**实际公钥**。为了签署消息 $m \in \{0, 1\}^n$ ，签名者执行以下操作：

1. 它首先（根据需要）为从根到标记为 m 的叶子的路径上的所有节点生成密钥。（其中一些公钥可能是在签署先前消息的过程中生成的，在这种情况下不会再次生成。）
2. 接下来，它通过使用私钥 sk_w 计算 $pk_{w0}||pk_{w1}$ 上的签名，为从根到标记为 m 的叶子的路径上的每个**真前缀** w 的字符串“认证”该路径。
3. 最后，它通过使用私钥 sk_m 计算 m 上的签名来“认证” m 本身。

m 上的最终签名由相对于 pk_m 的 m 上的签名，以及验证从标记为 m 的叶子到根的路径所需的所有信息组成；见图 12.5。此外，签名者通过存储在上述签名过程中生成的所有密钥来更新其状态。该方案的正式描述在构造 12.20 中给出。

图 12.5：基于树的签名（概念图）。

请注意，该方案中使用的每个底层密钥仅用于签署**单个**“消息”。与内部节点关联的每个密钥签

署一对其他公钥，并且叶子处的密钥用于签署单个消息。由于每个密钥用于签署一对其他密钥，我们再次需要一次性安全签名方案 Π 能够签署比公钥更长的消息。引理 12.19 表明，此类方案可以基于抗碰撞哈希函数来构造。

在证明这种基于树的方法的安全性之前，请注意它在许多方面都改进了基于链的方案。它仍然允许签署无限制数量的消息。（尽管只有 2^n 个叶子，但消息空间仅包含 2^n 个消息。无论如何， 2^n 最终大于 n 的任何多项式函数。）在效率方面，签名长度和验证时间现在与消息长度 n 成比例，但与已签署消息的数量无关。该方案仍然是有状态的，但我们将证明以下结果后看到如何避免这种情况。

定理 12.21 令 Π 是一个一次性安全签名方案。那么构造 12.20 是一个安全签名方案。

证明 令 Π^* 表示构造 12.20。令 \mathcal{A}^* 是一个概率多项式时间敌手，令 $l^* = l^*(n)$ 是 \mathcal{A}^* 进行签名查询次数的（多项式）上限，并设置 $l = l(n) \stackrel{\text{def}}{=} 2nl^*(n) + 1$ 。

构造 12.20 令 $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$ 是一个签名方案。对于二进制字符串 m ，令 $m_{|i} \stackrel{\text{def}}{=} m_1 \dots m_i$ 表示 m 的 i 比特前缀（其中 $m_{|0} \stackrel{\text{def}}{=} \varepsilon$ ，空字符串）。构造方案 $\Pi^* = (\text{Gen}^*, \text{Sign}^*, \text{Vrfy}^*)$ 如下：

- Gen^* : 输入 1^n ，计算 $(pk_\varepsilon, sk_\varepsilon) \leftarrow \text{Gen}(1^n)$ ，并输出公钥 pk_ε 。私钥和初始状态是 sk_ε 。
- Sign^* : 输入消息 $m \in \{0, 1\}^n$ ，执行以下操作。
 - i. 对于 $i = 0$ 到 $n - 1$:
 - 如果 $pk_{m_{|i}0}, pk_{m_{|i}1}$ 和 $\sigma_{m_{|i}}$ 不在状态中，计算 $(pk_{m_{|i}0}, sk_{m_{|i}0}) \leftarrow \text{Gen}(1^n)$ ， $(pk_{m_{|i}1}, sk_{m_{|i}1}) \leftarrow \text{Gen}(1^n)$ ，和 $\sigma_{m_{|i}} \leftarrow \text{Sign}_{sk_{m_{|i}}} (pk_{m_{|i}0} || pk_{m_{|i}1})$ 。此外，将所有这些值添加到状态中。
 - ii. 如果 σ_m 尚未包含在状态中，计算 $\sigma_m \leftarrow \text{Sign}_{sk_m}(m)$ 并将其存储为状态的一部分。
 - iii. 输出签名 $(\{\sigma_{m_{|i}}, pk_{m_{|i}0}, pk_{m_{|i}1}\}_{i=0}^{n-1}, \sigma_m)$ 。
 - Vrfy^* : 输入公钥 pk_ε 、消息 m 和签名

$$\sigma^* = (\{\sigma_{m_{|i}}, pk_{m_{|i}0}, pk_{m_{|i}1}\}_{i=0}^{n-1}, \sigma_m)$$

当且仅当：

- i. $\text{Vrfy}_{pk_{m_{|i}}} (pk_{m_{|i}0} || pk_{m_{|i}1}, \sigma_{m_{|i}}) \stackrel{?}{=} 1$ 对于所有 $i \in \{0, \dots, n - 1\}$ 。
- ii. $\text{Vrfy}_{pk_m} (m, \sigma_m) \stackrel{?}{=} 1$ 。时输出 1。

一个“基于树”的签名方案。

请注意 l 上限了生成 l^* 个签名所需的 Π 的公钥数量。这是因为 Π^* 中的每个签名最多需要 Π 的 $2n$ 个新密钥（在最坏情况下），并且 Π 的一个附加密钥用作实际公钥 pk_ε 。

考虑以下攻击一次性安全签名方案 Π 的 PPT 敌手 \mathcal{A} :

敌手 \mathcal{A} : \mathcal{A} 被给予公钥 pk 作为输入（安全参数 n 是隐式的）。

- 选择一个均匀的索引 $i^* \in \{1, \dots, l\}$ 。构造一个密钥列表 pk^1, \dots, pk^l 如下：
 - 设置 $pk^{i^*} := pk$ 。
 - 对于 $i \neq i^*$ ，计算 $(pk^i, sk^i) \leftarrow \text{Gen}(1^n)$ 。
- 在输入公钥 $pk_\varepsilon = pk^1$ 上运行 \mathcal{A}^* 。当 \mathcal{A}^* 请求消息 m 上的签名时，执行：
 - i. 对于 $i = 0$ 到 $n - 1$ ：
 - 如果值 $pk_{m|_i}0, pk_{m|_i}1$ ，和 $\sigma_{m|_i}$ 尚未定义，则设置 $pk_{m|_i}0$ 和 $pk_{m|_i}1$ 等于接下来的两个未使用的公钥 pk^i 和 pk^{i+1} ，并计算 $\sigma_{m|_i}$ 在 $pk_{m|_i}0 || pk_{m|_i}1$ 上相对于 $pk_{m|_i}$ 的签名。⁶
 - 如果 σ_m 尚未定义，计算 σ_m 在 m 上相对于 pk_m 的签名（参见脚注 6）。
 - 将 $(\{\sigma_{m|_i}, pk_{m|_i}0, pk_{m|_i}1\}_{i=0}^{n-1}, \sigma_m)$ 给予 \mathcal{A}^* 。
 - ii. 假设 \mathcal{A}^* 输出消息 m （它之前没有请求签名）和签名 $\sigma = (\{\sigma'_{m|_i}, pk'_{m|_i}0, pk'_{m|_i}1\}_{i=0}^{n-1}, \sigma'_m)$ 。如果这是一个有效签名，则：
- **情况 1:** 假设存在 $j \in \{0, \dots, n - 1\}$ 使得 $pk'_{m|_j}0 \neq pk_{m|_j}0$ 或 $pk'_{m|_j}1 \neq pk_{m|_j}1$ ；这包括 $pk_{m|_j}0$ 或 $pk_{m|_j}1$ 从未由 \mathcal{A} 定义的情况。取最小的 j ，并令 i 使得 $pk^i = pk_{m|_j}$ （由于 j 的最小性，存在这样的 i ）。如果 $i = i^*$ ，输出 $(pk_{m|_j}0 || pk_{m|_j}1, \sigma'_{m|_j})$ 。
- **情况 2:** 如果情况 1 不成立，则 $pk'_m = pk_m$ 。令 i 使得 $pk^i = pk_m$ 。如果 $i = i^*$ ，输出 (m, σ'_m) 。

在实验 $\text{Sig-forge}_{\mathcal{A}, \Pi}^{\text{one-time}}(n)$ 中，当 \mathcal{A}^* 作为 \mathcal{A} 的子程序运行时， \mathcal{A}^* 的视图与 \mathcal{A}^* 在实验 $\text{Sig-forge}_{\mathcal{A}^*, \Pi^*}(n)$ 中的视图是完全相同的。⁷ 因此，当 \mathcal{A} 在这个实验中作为子程序运行时， \mathcal{A}^* 输出伪造的概率正好是 $\delta(n)$ 。给定 \mathcal{A}^* 输出伪造，考虑上面描述的两种可能情况中的每一种：

情况 1: 由于 i^* 是均匀的且独立于 \mathcal{A}^* 的视图，概率 $i = i^*$ 正好是 $1/l$ 。如果 $i = i^*$ ，那么 \mathcal{A} 请求了关于消息 $pk_{m|_j}0 || pk_{m|_j}1$ 的签名，相对于公钥 $pk = pk^{i^*} = pk_{m|_j}$ ，并且获得了它（没有请求其他签名）。此外，

$$pk'_{m|_j}0 || pk'_{m|_j}1 \neq pk_{m|_j}0 || pk_{m|_j}1$$

然而 $\sigma'_{m|_j}$ 是在 $pk'_{m|_j}0 || pk'_{m|_j}1$ 上相对于 pk 的有效签名。因此， \mathcal{A} 在这种情况下输出伪造。

情况 2：同样，由于 i^* 是均匀随机选择的，并且独立于 \mathcal{A}^* 的视图，概率 $i = i^*$ 正好是 $1/l$ 。如果 $i = i^*$ ，那么 \mathcal{A} 没有请求关于公钥 $pk = pk^{i^*} = pk_m$ 的任何签名，然而 σ'_m 是在 m 上相对于 pk 的有效签名。

我们看到，以 \mathcal{A}^* 输出伪造为条件， \mathcal{A} 以恰好 $1/l$ 的概率输出伪造。这意味着

$$\Pr[\text{Sig-forge}_{\mathcal{A}, \Pi}^{\text{one-time}}(n) = 1] = \Pr[\text{Sig-forge}_{\mathcal{A}^*, \Pi^*}(n) = 1]/l(n)$$

因为 Π 是一个一次性安全签名方案，所以存在一个可忽略的函数 negl 使得

$$\Pr[\text{Sig-forge}_{\mathcal{A}, \Pi}^{\text{one-time}}(n) = 1] \leq \text{negl}(n)$$

由于 l 是多项式，这意味着 $\Pr[\text{Sig-forge}_{\mathcal{A}^*, \Pi^*}(n) = 1]$ 是可忽略的。

一个无状态的解决方案

如前所述，签名者是根据需要**动态地**生成状态。但是，我们可以想象让签名者**提前**生成整个树中所有节点所需的必要信息，在密钥生成时。（也就是说，在密钥生成时，签名者可以生成所有二进制字符串 w 长度最多为 n 的密钥 $\{(pk_w, sk_w)\}$ 和签名 $\{\sigma_w\}$ 。）如果密钥生成以这种方式完成，签名者将根本不必更新其状态；所有这些值都可以作为（巨大）私钥的一部分存储，我们将获得一个**无状态**方案。当然，这种方法的问题是生成所有这些值将需要**指数时间**，并且存储所有这些值将需要**指数内存**。

另一种选择是存储一些**随机性**，这些随机性可用于根据需要生成值 $\{(pk_w, sk_w)\}$ 和 $\{\sigma_w\}$ ，而不是存储值本身。也就是说，签名者可以为每个 w 存储一个随机字符串 r_w ，并且每当需要值 pk_w, sk_w 时，签名者可以计算 $(pk_w, sk_w) := \text{Gen}(1^n; r_w)$ ，其中这表示使用随机硬币 r_w 生成长度为 n 的密钥。同样，如果签名过程是概率性的，签名者可以存储 r'_w ，然后设置 $\sigma_w := \text{Sign}_{sk_w}(pk_{w0} || pk_{w1}; r'_w)$ （这里假设 $|w| < n$ ）。生成和存储足够多的随机字符串，然而，仍然需要**指数时间和内存**。

这种替代方案的一个简单修改提供了一个**多项式时间**解决方案。不是如上所述存储随机 r_w 和 r'_w ，签名者可以存储伪随机函数 F 的两个密钥 k, k' 。当需要时，值 pk_w, sk_w 现在可以通过以下两步过程生成：

1. 计算 $r_w := F_k(w)$ 。⁸
2. 计算 $(pk_w, sk_w) := \text{Gen}(1^n; r_w)$ （如前所述）。

此外，密钥 k' 用于生成用于计算签名 σ_w 的值 r'_w 。这提供了一个**无状态**方案，其中密钥生成（以及签名和验证）可以在**多项式时间**内完成。直观上，这是安全的，因为存储随机函数等同

于存储所有所需的 r_w 和 r'_w 值，并且存储伪随机函数“就像一样好”。我们将其作为一个练习，以给出该修改后的方案保持安全的正式证明。

由于抗碰撞哈希函数的存在意味着单向函数的存在（参见练习 7.4），而后者意味着伪随机函数的存在（参见第 7 章），我们有：

定理 12.22 如果抗碰撞哈希函数存在，那么存在一个**（无状态）安全签名方案**。

我们注意到，可以从（最小的）假设**单向函数存在**来构造满足定义 12.2 的签名方案；这个结果的证明超出了本书的范围。

12.7 *证书和公钥基础设施

在本节中，我们简要讨论数字签名的主要应用之一：**公钥的安全分发**。这使我们对公钥密码学的讨论回到了原点。在本章和前一章中，我们看到了如何在公钥安全分发后使用公钥密码学。现在我们展示了公钥密码学本身如何可用于安全地分发公钥。这听起来可能是循环的，但事实并非如此。我们将展示，一旦**单个公钥**（属于受信任方）以安全方式分发，该密钥就可以用于“引导”任意数量其他公钥的安全分发。因此，至少原则上，只需要解决一次安全密钥分发的问题。

这里的关键概念是**数字证书**，它只是一个将实体绑定到某个公钥的签名。具体来说，假设一方 **Charlie** 为一个安全数字签名方案生成了一个密钥对 (pk_C, sk_C) （在本节中，我们只关注满足定义 12.2 的签名方案）。进一步假设另一方 **Bob** 也生成了一个密钥对 (pk_B, sk_B) （在当前的讨论中，这些可能是签名方案或公钥加密方案的密钥），并且 Charlie 知道 pk_B 是 Bob 的公钥。然后 Charlie 可以计算签名

$$\text{cert}_{C \rightarrow B} \stackrel{\text{def}}{=} \text{Sign}_{sk_C}(\text{'Bob's key is } pk_B\text{'})$$

并将其签名交给 Bob。我们称 $\text{cert}_{C \rightarrow B}$ 为 Bob 密钥的**证书**，由 Charlie 颁发。

实际上，证书应该明确标识持有特定公钥的实体，因此会使用比“Bob”更独特的描述性术语，例如 Bob 的全名和电子邮件地址，或 Bob 网站的 URL。

现在假设 Bob 想要与已经知道 pk_C 的另一方 **Alice** 通信。Bob 可以将 $(pk_B, \text{cert}_{C \rightarrow B})$ 发送给 Alice，Alice 随后可以验证 $\text{cert}_{C \rightarrow B}$ 确实是相对于 pk_C 的消息“'Bob's key is pk_B '”上的有效签名。假设验证成功，Alice 现在知道 Charlie 已经签署了指示的消息。如果 Alice 信任 Charlie，她可以接受 pk_B 作为 Bob 的合法公钥。

Bob 和 Alice 之间的所有通信都可以通过**不安全且未经认证的信道**进行。如果一个主动敌手干

扰了从 Bob 到 Alice 的 $(pk_B, \text{cert}_{C \rightarrow B})$ 传输，那么该敌手将无法生成将 Bob 链接到任何其他公钥 pk'_B 的有效证书，除非 Charlie 之前已经签署了将 Bob 链接到 pk'_B 的某个其他证书（在这种情况下，无论如何这都不是一个严重的攻击）。所有这一切都假设 Charlie 不是不诚实的，并且他的私钥没有被泄露。

我们在上面的描述中省略了许多细节。最突出的是，我们没有讨论 Alice 最初是如何得知 pk_C 的；Charlie 如何确定 pk_B 是 Bob 的公钥；以及 Alice 如何决定是否信任 Charlie。完全指定这些细节（以及其他）定义了**公钥基础设施 (PKI)**，它实现了公钥的广泛分发。已经提出了各种不同的 PKI 模型，我们现在提到一些比较流行的模型。我们在这里的讨论将保持在相对较高的水平，建议有兴趣了解更多细节的读者查阅本章末尾的参考文献。

单个证书颁发机构。最简单的 PKI 假设一个**单个证书颁发机构 (CA)** 受到所有人的完全信任，并且它为每个人的公钥颁发证书。证书颁发机构通常不会是人，而更可能是一家以认证公钥为业务的公司、一个政府机构，或者组织内部的一个部门（尽管在后一种情况下，CA 可能只被组织内部的人使用）。

任何想要依赖 CA 服务的人都必须获得 CA 的公钥 pk_{CA} 的合法副本。显然，这一步必须以**安全**方式执行，因为如果某一方获得了 pk_{CA} 的不正确版本，那么该方可能无法获得任何其他人的公钥的真实副本。这意味着 pk_{CA} 必须通过**认证信道**分发。最简单的方法是通过**物理方式**：例如，如果 CA 在一个组织内部，那么任何员工在工作的第一天就可以直接从 CA 获得 pk_{CA} 的真实副本。如果 CA 是一家公司，那么其他用户必须在某个时候去这家公司，并例如拿走一个包含 CA 公钥的 USB 驱动器。这个不便的步骤只需要执行一次。

实际上，CA 分发其公钥的一种常见方式是将其公钥与其他软件“捆绑”。例如，这在当今许多流行的网络浏览器中发生：CA 的公钥与浏览器一起提供，并且浏览器被编程为在证书到达时自动验证它们。（实际上，现代网络浏览器在其代码中硬编码了**多个** CA 的公钥，因此更准确地属于下面讨论的“多 CA”模型。）

CA 向 Bob 颁发证书的机制也必须受到非常仔细的控制，尽管细节可能因 CA 而异。作为一个例子，Bob 可能必须亲自带着他的公钥 pk_B 的副本以及证明他的名字（或他的电子邮件地址）是他声称的身份的身份数证明。只有这样，CA 才会颁发证书。

在只有一个 CA 的模型中，各方完全信任这个 CA 仅在适当的时候颁发证书；这就是为什么在颁发证书之前使用详细的验证过程至关重要。因此，如果 Alice 收到 $\text{cert}_{CA \rightarrow B}$ 证书，证明 pk_B 是 Bob 的公钥，Alice 将接受此断言为有效，并使用 pk_B 作为 Bob 的公钥。

多个证书颁发机构。只有一个 CA 的模型虽然简单且有吸引力，但并不实用。首先，在一个单一组织之外，不太可能每个人都信任同一个 CA。这并不意味着有人认为 CA 是腐败的；可

能仅仅是因为有人发现 CA 的验证过程不充分（例如，CA 在生成证书时只要求一种形式的身份证明，而 Alice 更希望使用两种）。此外，CA 是整个系统的**单点故障**。如果 CA 腐败，或可以被贿赂，甚至只是 CA 在保护其私钥方面疏忽，已颁发证书的合法性可能会受到质疑。对于所有想要证书的各方来说，必须联系这个 CA 也很不方便。

缓解这些问题的一种方法是依赖**多个 CA**。一方 Bob 想要获得他的公钥证书，可以选择他想要哪个 CA 颁发证书，而向 Alice 呈现证书（甚至是不同 CA 颁发的多个证书）的一方 Alice 可以选择她信任哪些 CA 的证书。

Bob 从多个 CA 获得证书没有坏处（除了有些不便和费用），但 Alice 必须更加小心，因为她的通信安全最终只取决于她信任的**最不安全**的 CA。也就是说，假设 Alice 信任两个 CA：CA₁ 和 CA₂，而 CA₂ 被敌手破坏。那么，虽然这个敌手无法伪造由 CA₁ 颁发的证书，但他将能够以 CA₂ 的名义为他选择的任何身份/公钥颁发伪造证书。这在当前系统中是一个**真实**问题。如前所述，操作系统/网络浏览器通常预先配置了许多 CA 的公钥，默认设置是将所有这些 CA 视为同样可信。事实上，任何愿意付费的公司都可以被包含为 CA。因此，预先配置的 CA 列表包括一些信誉良好、历史悠久的公司以及其他一些信誉不容易确立的新公司。用户可以手动配置他们的设置，以便只接受他们信任的 CA 颁发的证书。

委派和证书链。另一种缓解单个 CA 负担（但不解决单点故障的安全问题）的方法是使用**证书链**。我们提出了长度为 2 的证书链的想法，尽管很容易看出我们所说的一切都可以推广到任意长度的链。

假设 Charlie，充当 CA，为 Bob 颁发证书，如我们最初的讨论所示。进一步假设 Bob 的密钥 pk_B 是签名方案的公钥。Bob 反过来可以为其他方颁发他自己的证书。例如，Bob 可以为 Alice 颁发形式为

$$\text{cert}_{B \rightarrow A} \stackrel{\text{def}}{=} \text{Sign}_{sk_B}(\text{'Alice's key is } pk_A\text{'})$$

的证书。

现在，如果 Alice 想要与知道 Charlie 公钥（但不知道 Bob 公钥）的第四方 **Dave** 通信，那么 Alice 可以发送

$$pk_A, \text{cert}_{B \rightarrow A}, pk_B, \text{cert}_{C \rightarrow B}$$

给 Dave。Dave 可以从中推断出什么？好吧，他可以首先验证 Charlie（他信任且其公钥已在他手中）是否签署了证书 $\text{cert}_{C \rightarrow B}$ ，表明 pk_B 确实属于一个名叫 Bob 的人。Dave 还可以验证这个名叫 Bob 的人是否签署了证书 $\text{cert}_{B \rightarrow A}$ ，表明 pk_A 确实属于 Alice。如果 Dave

信任 Charlie 仅向值得信赖的人颁发证书，那么 Dave 可以接受 pk_A 是 Alice 的真实密钥。

我们强调，在这个例子中，证书 $\text{cert}_{C \rightarrow B}$ 关联了**更强的语义**。在我们之前的讨论中，这种形式的证书只是断言 Bob 持有公钥 pk_B 。现在，证书断言 Bob 持有公钥 pk_B 并且 **Bob 被信任颁发其他证书**。当 Charlie 为 Bob 签署具有这些更强语义的证书时，Charlie 实际上是在**委派**他颁发证书的能力给 Bob。Bob 现在可以充当 Charlie 的**代理**，以 Charlie 的名义颁发证书。

回到基于 CA 的 PKI，我们可以想象一个“根” CA 和 n 个“二级” CA CA_1, \dots, CA_n 。根 CA 可以为每个二级 CA 颁发证书，然后这些二级 CA 又可以为持有公钥的其他主体颁发证书。这减轻了根 CA 的负担，也使得各方更容易获得证书（因为他们现在可以联系离他们最近的二级 CA，例如）。另一方面，管理这些二级 CA 可能很困难，它们的存在意味着系统中现在有更多的攻击点。

“信任网络”模型。我们将讨论的最后一个 PKI 示例是一个完全**分布式的**模型，没有中央信任点，称为**“信任网络”**模型。PGP（“**非常好的隐私**”）电子邮件加密软件的一个变体使用了这种模型来分发公钥。

在“信任网络”模型中，任何人都可以为任何人颁发证书，每个用户必须自己决定对其他用户颁发的证书给予多少信任。作为一个例子，假设用户 Alice 已经拥有了一些用户 C_1, C_2, C_3 的公钥 pk_1, pk_2, pk_3 。（我们将在下面讨论 Alice 最初是如何获得这些公钥的。）另一个想要与 Alice 通信的用户 Bob 可能拥有证书 $\text{cert}_{C_1 \rightarrow B}, \text{cert}_{C_3 \rightarrow B}$ ，和 $\text{cert}_{C_4 \rightarrow B}$ ，并将这些证书（以及他的公钥 pk_B ）发送给 Alice。Alice 无法验证 $\text{cert}_{C_4 \rightarrow B}$ （因为她没有 C_4 的公钥），但她可以验证其他两个证书。现在她必须决定对 C_1 和 C_3 给予多少信任。如果她**明确地**信任 C_1 ，或者如果她对 C_1 和 C_3 都给予**较少**的信任，她可能会决定接受 pk_B 。（例如，她可能认为 C_1 或 C_3 之一腐败的可能性很大，但认为它们两者都腐败的可能性很小。）

如上所述，在这个模型中，用户被期望收集其他方的公钥以及他们自己公钥的证书。在 PGP 的上下文中，这曾经是在“密钥签名派对”上完成的，PGP 用户聚集在一起（例如在会议上），互相给出他们公钥的真实副本，并互相颁发证书。通常，密钥签名派对上的用户可能彼此不认识，但他们可以检查驾驶执照等，然后接受或颁发证书。

公钥和证书也可以存储在**中央数据库**中，PGP 就是这样做的（参见 <http://pgp.mit.edu>）。当 Alice 想要向 Bob 发送加密消息时，她可以在这个数据库中搜索 Bob 的公钥；除了 Bob 的公钥之外，数据库还将返回它持有的为 Bob 的公钥颁发的所有证书列表。数据库中也可能找到 Bob 的多个公钥，并且这些公钥中的每一个都可能由不同方的集合颁发证书。Alice 再次需要决定在使用这些公钥之前对其中任何一个给予多少信任。

信任网络模型很有吸引力，因为它**不要求**信任任何中央颁发机构。另一方面，虽然它可能适用于加密电子邮件的普通用户，但对于安全更关键的设置，或对于组织公钥的分发（例如，网络上的电子商务），它似乎不合适。例如，如果用户想与他的银行通信，他不太可能信任他在会议上遇到的人来认证他银行的公钥，银行代表也不太可能去参加密钥签名派对以获取银行的密钥认证。

证书作废

我们还没有完全涉及的一个重要问题是，证书通常**不应该无限期有效**。员工可能会离开公司，在这种情况下，他或她不再被允许接收来自公司内部其他人的加密通信；用户的私钥可能被盗，在这种情况下，用户（假设他们知道被盗）将希望生成一个新的密钥对，并从流通中删除旧的公钥。在任何一种情况下，我们都需要一种方法来使以前颁发的证书**无效**。

处理这些问题的方法多种多样且复杂，我们只会提到两种相对简单的想法，它们在某种意义上代表了相反的极端。（改进这些方法是现实世界网络安全研究的一个活跃领域。）

过期。 防止证书无限期使用的一种方法是将**过期日期**作为证书的一部分包含在内。CA Charlie 为 Bob 的公钥颁发的证书现在可能具有以下形式

$$\text{cert}_{C \rightarrow B} \stackrel{\text{def}}{=} \text{Sign}_{sk_C}('Bob's key is pk_B', \text{date})$$

其中 date 是未来的某个日期，届时证书将失效。（例如，从证书颁发之日起一年。）当另一个用户验证此证书时，他们不仅需要知道 pk_B ，还需要知道过期日期，并且他们现在不仅需要检查签名是否有效，还需要检查过期日期是否已过。持有证书的用户必须在当前证书过期时联系 CA 以获得新证书；此时，CA 会在颁发另一个证书之前再次验证用户的身份/凭据。

使用过期日期为前面提到的问题提供了一个非常**粗粒度**的解决方案。如果一名员工在获得证书的第二天离开公司，并且证书在颁发日期一年后过期，那么该员工可以在**整整一年内**非法使用他或她的公钥，直到过期日期过去。因此，这种方法通常与我们接下来描述的其他方法结合使用。

撤销。 当员工离开组织，或用户的私钥被盗时，我们希望为他们的公钥颁发的证书**立即**失效，或者至少**尽快**失效。这可以通过让 CA 明确**撤销**证书来实现。为简单起见，我们假设有一个 CA，但我们所说的一切更普遍地适用于用户拥有多个 CA 颁发的证书的情况。

有许多不同的方法可以处理撤销。一种可能性（我们将讨论的唯一一种）是让 CA 在其颁发的每个证书中包含一个**序列号**；也就是说，证书现在将具有以下形式

$$\text{cert}_{C \rightarrow B} \stackrel{\text{def}}{=} \text{Sign}_{sk_C}('Bob's key is pk_B', \#\#\#)$$

其中“####”代表此证书的序列号。每个证书都应具有**唯一的**序列号，CA 将存储它生成的每个证书的信息 ($\text{Bob}, \text{pk}_B, \text{####}$)。如果用户 Bob 对应于公钥 pk_B 的私钥被盗，Bob 可以将此事告知 CA。（CA 必须在这里验证 Bob 的身份，以防止另一个用户虚假地撤销颁发给 Bob 的证书。）CA 将搜索其数据库以找到与颁发给 Bob 和 pk_B 的证书关联的序列号。例如，在每天结束时，CA 将生成一个**证书撤销列表 (CRL)**，其中包含所有被撤销证书的序列号，并签署 CRL 和当前日期。然后，签署的 CRL 被广泛分发或以其他方式提供给潜在的验证者。现在，验证证书需要检查证书中的签名是否有效，检查序列号是否未出现在最新的撤销列表中，并验证 CA 在撤销列表上的签名本身。

以我们描述的方式，这种方法在证书失效之前最多有**一天**的间隔。这比仅基于过期日期的方法提供了更大的灵活性。

12.8 总结 - SSL/TLS

作为我们迄今为止在本书中涵盖内容的总结，我们讨论了**传输层安全 (TLS)** 协议，该协议广泛用于保护网络通信；TLS 是您连接到使用 https 而不是 http 的网站时浏览器使用的协议。我们应该清楚，我们在这里的目标是关注核心 TLS 协议中使用的底层密码学，而不是从网络安全角度来看有趣但与我们的关注无关的各种其他方面。像往常一样，我们对协议的某些部分进行了轻微的简化和抽象，以传达要点，我们的描述不应被依赖于实现。最后，我们没有正式定义或声明协议的安全性；事实上，TLS 的形式化分析是活跃研究的主题。

TLS 是一个标准化协议，基于一个名为 SSL（或**安全套接字层**）的先驱，该先驱由 Netscape 在 20 世纪 90 年代中期开发；可用的最后一个版本是 SSL 3.0。TLS 1.0 版本于 1999 年发布，2006 年更新到 1.1 版本，并在 2008 年再次更新到 1.2 版本（当前版本）。截至撰写本文时，大约 50% 的网站仍在使用 TLS 1.0 而不是更新的版本；所有主要的网络浏览器都支持 TLS 1.2，尽管在某些情况下默认使用更早版本的 TLS。在大多数情况下，我们下面的描述处于足够高的抽象级别，以至于版本之间的差异对于我们的目的来说并不重要。但是，我们提醒，早期版本存在一些已知的攻击。

TLS 允许**客户端**（例如网络浏览器）和**服务器**（例如网站）就一组共享密钥达成一致，然后使用这些密钥来加密和认证他们后续的通信。它由两部分组成：一个**握手协议**，执行认证密钥交换以建立共享密钥，以及一个**记录层协议**，使用这些共享密钥来加密/认证双方的通信。尽管 TLS 允许客户端向服务器认证，但它主要用于**仅对客户端认证**服务器，因为通常只有服务器拥有证书。（在 TLS 会话建立后，如果需要，用户到服务器的认证可以在网络堆栈的应用层完成，例如发送密码。）

握手协议。我们现在描述握手协议的基本流程。在协议开始时，客户端 C 持有一组 CA 的公

钥 $\{pk_1, \dots, pk_n\}$, 服务器 S 拥有一对密钥 (pk_S, sk_S) 用于 KEM 以及由 C 知道其公钥的 CA 之一颁发的证书 $cert_{i \rightarrow S}$ 。要连接到 S , 双方运行以下步骤。

1. C 首先向 S 发送一条消息, 其中包括有关客户端支持的协议版本、客户端支持的**密码套件** (例如客户端允许使用哪些哈希函数或分组密码) 以及一个**均匀值** (一个“nonce”) N_C 的信息。
2. S 响应, 选择它支持的最新协议版本以及一个合适的密码套件。此外, 它发送其公钥 pk_S 、其证书 $cert_{i \rightarrow S}$, 以及它自己的均匀值 N_S 。
3. C 检查它持有的 CA 公钥中是否有一个, 例如 pk_i , 与颁发 S 证书的 CA 匹配。如果是, C 验证证书 (并检查它是否没有过期或被撤销) 并在成功时得知 pk_S 是 S 的公钥。然后它运行 $(c, pmk) \leftarrow \text{Encaps}_{pk_S}(1^n)$ (见第 11.3 节) 以获得密文 c 和所谓的**预主密钥** pmk 。它将 c 发送给服务器。
 pmk 用于使用应用于 pmk, N_C , 和 N_S 的**密钥派生函数** (参见第 5.6.4 节) 派生**主密钥** mk 。客户端随后应用**伪随机生成器**到 mk 以派生四个密钥 k_C, k'_C, k_S, k'_S 。
最后, C 计算 $\tau_C \leftarrow \text{Mac}_{mk}(\text{transcript})$, 其中 transcript 表示 C 和 S 之间迄今为止交换的所有消息。客户端然后将 τ_C 发送给 S 。 (事实上, τ_C 本身被加密和认证, 就像下面描述的记录层中的通信一样。)
4. S 计算 $pmk := \text{Decaps}_{sk_S}(c)$, 从中它可以像客户端一样派生 mk 和 k_C, k'_C, k_S, k'_S 。如果 $\text{Vrfy}_{mk}(\text{transcript}, \tau_C) \neq 1$, 则 S 中止。否则, 它设置 $\tau_S \leftarrow \text{Mac}_{mk}(\text{transcript}')$, 其中 $\text{transcript}'$ 表示 C 和 S 之间迄今为止交换的所有消息 (即, 包括 C 发送的最后一条消息)。 S 然后将 τ_S 发送给 C 。 (同样, τ_S 实际上被加密和认证为记录层流量。)
5. 如果 $\text{Vrfy}_{mk}(\text{transcript}', \tau_S) \neq 1$, 则客户端中止。

在握手协议成功执行结束时, C 和 S 共享一组四个对称密钥 k_C, k'_C, k_S, k'_S 。

TLS 1.2 支持两种 KEM: 如构造 11.19 中基于 CDH/DDH 的 KEM, 或基于 RSA 的加密方案 PKCS #1 v1.5。为了防止对后一种方案的 Bleichenbacher 式攻击, 如果 Decaps 在步骤 4 中失败, 服务器**不应报告**解密错误。相反, 如果解密失败, 它应该选择一个均匀的 pmk 并继续使用该值运行协议。 (在这种情况下, τ_C 当然不会验证, S 会中止; 重点是攻击者无法区分这是否是由于解密失败。)

握手协议安全性的直觉是, 由于 C 验证了证书, 它知道只有合法的服务器 S 才能知道 pmk , 从而知道 mk 。因此, 如果协议成功终止, C 知道它与合法的 S 共享密钥 k_C, k'_C, k_S, k'_S , 并且没有敌手知道有关这些密钥的任何信息。即使在主动敌手存在的情况下, 这也应该成立, 尽管正式证明非常复杂。然而, 我们提醒, **消息认证码**在记录上的使用是为了防止**中间人攻击者**更改在握手协议开始时发送的协议版本和密码套件。这可以防止攻击者导致 S 或 C 使用旧

的、较弱的协议版本，或较弱的分组密码和短密钥。

记录层协议。一旦 C 和 S 之间就密钥达成一致，双方就会使用这些密钥来加密和认证他们后续的所有通信。 C 使用 k_C （分别 k'_C ）来**加密**（分别**认证**）它发送给 S 的所有消息；类似地， S 使用 k_S 和 k'_S 来加密和认证它发送的所有消息。**序列号**用于防止重放攻击，如第 4.5.3 节所述。TLS 1.2 使用**先认证后加密**的方法，正如我们在第 4.5.2 节中看到的那样，这可能是有问题的。

12.9 *签名加密

为了结束本章，我们简要地、非正式地讨论了公钥设置中**联合机密性和完整性**的问题。虽然这与我们从第 4.5 节开始的讨论相似，但我们现在处于公钥设置的事实引入了几个额外的复杂性。

为简单起见，我们考虑一个网络，其中所有相关方都拥有用于加密和签名的公钥/私钥对。我们用 (ek, dk) 表示（公共）加密密钥和（私人）解密密钥，并使用 (vk, sk) 表示（公共）验证密钥和（私人）签名密钥。我们假设所有人都知道其他所有人的公钥。

非正式地，我们的目标是设计一种机制，允许发送方 S 向接收方 R 发送消息 m ，同时确保（1）网络中没有其他方可以了解有关 m 的任何信息（即**机密性**），以及（2） R 确信消息来自 S （即**完整性**）。我们将希望考虑这些安全属性，即使是针对网络中其他方的主动（例如，选择密文）攻击。

根据我们在第 4.5 节中的讨论，一个自然的思路是使用“**先加密后认证**”的方法，其中 S 将 $(S, c, \text{Sign}_{sk_S}(c))$ 发送给 R ，其中 c 是使用 R 的加密密钥 ek_R 对 m 的加密。（为方便起见，我们明确包括了发送方的身份。）然而，无论使用何种加密方案，这里都有一个**巧妙的选择密文攻击**。观察到如上所述的传输后，另一个（敌对的）方 A 可以剥离 S 的签名并用自己的签名替换，发送 $(A, c, \text{Sign}_{sk_A}(c))$ 给 R 。在这种情况下， R 不会检测到任何错误，并且会错误地认为 A 发送了消息 m 。如果 R 回复 A ，或者以某种方式对待 A ，使其行为取决于消息的内容，那么 A 可能会了解未知消息 m 。（这种方案的另一个问题，尽管与我们在这里的讨论有些独立，是它不再提供**不可否认性**。也就是说， R 不能轻易向第三方证明 S 签署了消息 m ，至少不能在不泄露其自己的解密密钥 dk_R 的情况下。）

可以尝试使用“**先认证后加密**”的方法。在这里， S 将首先计算签名 $\sigma \leftarrow \text{Sign}_{sk_S}(m)$ ，然后发送

$$(S, \text{Enc}_{ek_R}(m||\sigma))$$

(请注意，这解决了上面提到的不可否认性问题。) 如果加密方案只是 CPA 安全的，那么就会出现与第 4.5 节中提到的类似问题，所以我们假设使用了 CCA 安全的加密方案。即使如此，恶意 R 也可以进行攻击。在收到 $(S, \text{Enc}_{ek_R}(m||\sigma))$ 后，恶意 R 可以解密以获得 $m||\sigma$ ，然后重新加密并发送 $(S, \text{Enc}_{ek_{R'}}(m||\sigma))$ 给另一个接收方 R' 。这个（诚实的）接收方 R' 然后会认为 S 发送了消息 m 。这可能会产生严重的后果，例如，如果 m 是消息“*I owe you \$100.*”。

如果各方更仔细地处理**标识符**，可以防止这些攻击。在加密时，发送方应将其**自己的身份**包含在消息中；在签名时，一方应签署**预期接收方的身份**以及正在签署的内容。例如，第二种方法将被修改，以便 S 首先计算 $\sigma \leftarrow \text{Sign}_{sk_S}(m||R)$ ，然后发送 $(S, \text{Enc}_{ek_R}(S||m||\sigma))$ 给 R 。在解密时，接收方应检查结果解密值是否包含（声称的）发送方身份；在验证时，接收方应检查被签署的内容是否包含其自己的身份。

以这种方式包含身份时，如果使用 CCA 安全加密方案和**强安全**签名方案（其定义与 MAC 的定义 4.3 相似），则**先认证后加密**和**先加密后认证**都是安全的。

参考文献和延伸阅读

关于签名的著名早期工作包括 Diffie 和 Hellman、Rabin、Rivest、Shamir 和 Adleman 以及 Goldwasser、Micali 和 Yao。有关超出此处涵盖范围的签名方案的广泛处理，请参见。

Goldwasser、Micali 和 Rivest 定义了**适应性选择消息攻击下存在不可伪造性**的概念，并给出了第一个满足此定义的有状态签名方案的构造。Goldreich 提出了一种使 Goldwasser-Micali-Rivest 方案无状态的方法，我们基本上采用了第 12.6.3 节中的 Goldreich 的想法。

朴素 RSA 签名可以追溯到最初的 RSA 论文。RSA-FDH 是由 Bellare 和 Rogaway 在他们关于随机谕示模型的原创论文中提出的，尽管使用密码哈希函数来防止代数攻击的想法（未经证明）可以追溯到 Rabin。后来的改进作为 PKCS #1 v2.1 的一部分被标准化，可在 <http://www.emc.com/emc-plus/rsa-labs> 获取。

Fiat-Shamir 变换 和 Schnorr 签名方案都可追溯到 20 世纪 80 年代末，尽管我们对定理 12.10 的证明归功于，我们对定理 12.11 的证明受到的启发。DSA 和 ECDSA 标准中有描述。

Lamport 的签名方案于 1979 年发表，尽管它已经在 中描述。与构造 12.20 精神相似的基于树的构造是由 Merkle 提出的，尽管基于树的方法也用于。Naor 和 Yung 表明**单向置换**足以构造可以签署任意长度消息的**一次性安全签名**，这被 Rompel 改进，他表明**单向函数**就足够了。（另见。）正如我们在第 12.6.3 节中看到的，这种一次性安全签名可用于构造安全签名

方案，这意味着单向函数足以用于（无状态）安全签名的存在。

证书的概念最初由 Kohnfelder 在他的本科论文中描述。公钥基础设施在 [102, Chapter 15] 中有更详细的讨论。另见。TLS 协议的进一步细节可以在 等中找到。An 等人 给出了公钥设置中联合机密性和完整性的形式化处理。